On the Decidability of Model-Checking Information Flow Properties

Deepak D'Souza[1], Raveendra Holla[1], Janardhan Kulkarni[1],
Raghavendra K. Ramesh[1], and Barbara Sprick[2]

[1] Department of Computer Sc. & Automation, Indian Institute of Science, India
{deepakd,raveendra,janardhan,raghavendrakr}@csa.iisc.ernet.in
[2] Department of Computer Science, Modeling and Analysis of Information Systems,
TU Darmstadt, Germany
sprick@mais.informatik.tu-darmstadt.de

Abstract. Current standard security practices do not provide substantial assurance about information flow security: the end-to-end behavior of a computing system. Noninterference is the basic semantical condition used to account for information flow security. In the literature, there are many definitions of noninterference: Non-inference, Separability and so on. Mantel presented a framework of *Basic Security Predicates* (BSPs) for characterizing the definitions of noninterference in the literature. Model-checking these BSPs for finite state systems was shown to be decidable in [8]. In this paper, we show that verifying these BSPs for the more expressive system model of pushdown systems is undecidable. We also give an example of a simple security property which is undecidable even for finite-state systems: the property is a weak form of non-inference called WNI, which is not expressible in Mantel's BSP framework.

1 Introduction

Current standard security practices do not provide substantial assurance that the end-to-end behavior of a computing system satisfies important security policies such as confidentiality. Military, medical and financial information systems, as well as web based services such as mail, shopping and business-to-business transactions are all applications that create serious privacy concerns. The standard way to protect data is (discretionary) access control: some privilege is required in order to access files or objects containing the confidential data. Access control checks place restrictions on the release of information but not its propagation. Once information is released from its container, the accessing program may, through error or malice, improperly transmit the information in some form. Hence there is a lack of end-to-end security guarantees in access control systems.

Information flow security aims at answering end-to-end security. Most often the concept is studied in the setting of multi-level security [5] with data assigned levels in a security lattice, such that levels higher in the lattice correspond to data of higher sensitivity. A flow of information from a higher level in the security lattice to a lower one could breach confidentiality and a flow from a lower level

R. Sekar and A.K. Pujari (Eds.): ICISS 2008, LNCS 5352, pp. 26–40, 2008.

to a higher one might indicate a breach of integrity. Information flow security has been a focal research topic in computer security for more than 30 years. Nevertheless, the problem of securing the flow of information in systems is far from being solved. The two main research problems are, firstly, finding adequate formal properties of a "secure" system and, secondly, developing sound and efficient verification techniques to check systems for these properties.

Noninterference is the basic semantical condition used to account for information flow security. It requires that high-level behavior should not interfere with low-level observations. There are several semantical models for which noninterference has been studied. In [20], the state of the art was surveyed for approaches to capture and analyze information flow security of concrete programs. A broad spectrum of notions has been developed for noninterference at the level of more abstract specifications, in particular event systems (e.g., [13,17]), state based system (e.g,[10]) and process algebras (e.g.,[9]).

In this article, we look at information flow security at the level of abstract specifications. A system is viewed as generating traces containing "confidential" and "visible" events. The assumption is that any "low level-user", e.g. an attacker, knows the set of all possible behaviors (traces) but can observe only visible events. The information flow properties specify restrictions on the kind of traces the system may generate, so as to restrict the amount of information a low-level user can infer from his observations about confidential events having taken place in the system. For example, the "non-inference" [18,17,24] property states that for every trace produced by the system, its projection to only visible events must also be a possible trace of the system. Thus if a system satisfies the non-inference information flow property, a low-level user who observes the visible behavior of the trace is unable to infer whether or not any high-level behavior has taken place. There are other security properties defined in the literature: *separability* [17] (which requires that every possible low-level behavior interleaved with every possible high-level behaviour must be a possible behaviour of a system), *generalized noninterference* [16] (which requires that for every possible trace and every possible perturbation there is a correction to the perturbation such that the resulting trace is again a possible trace of the system), *nondeducability* [22], *restrictiveness* [16], the *perfect security property* [24] etc.

In [15] Mantel provides a framework for reasoning about the various information flow properties presented in the literature, in a modular way. He identifies a set of basic information flow properties which he calls "Basic Security Predicates" or BSPs, which are shown to be the building blocks of most of the known trace-based properties in the literature. The framework is modular in that BSPs that are common to several properties of interest for the given system need only be verified once for the system.

In this paper we consider the problem of model-checking information flow properties – that is, given a system model and a security property, can we algorithmically check whether the system satisfies the property? The most popular verification techniques are security type systems and program logics at the level of programs (see, e.g. [4]) and the unwinding technique at the level of specifications (see, e.g., [14,11,7]). Other approaches for abstract systems are the more recent "model-checking" technique in [8] and the algorithmic approach in [23].

The unwinding technique is based on identifying structural properties of the system model which ensure the satisfaction of the information flow property. The method is not complete in general, in that a system could satisfy the information flow property but fail the unwinding condition. In [15] Mantel gives unwinding conditions for most of the BSPs he identifies. However, finding a useful unwinding relation is not trivial and remains up to the verifier.

The model-checking approach in [8] on the other hand is both sound and complete for all information flow properties that can be characterised in terms of BSPs and relies on an automata-based approach (when the given system is finite-state) to check language-theoretic properties of the system. Meyden and Zhang [23] also develop algorithms for model-checking information flow properties for finite-state systems and characterize the complexity of the associated verification problem.

In this article we investigate the problem of model-checking BSPs for systems modelled as "pushdown" systems. These systems can make use of an unbounded stack, and are useful in modelling programs with procedure calls but no dynamic memory allocation ("boolean programs" [2]), or recursive state machines [1]. We show that this problem is undecidable for all the BSPs, and hence we cannot hope to find an algorithmic solution to the problem. This result does not immediately tell us that verifying the non-interference properties in literature (which Mantel showed can be expressed as conjuctions of his BSP's) is undecidable for pushdown systems. However, we can conclude that (a) it is not possible to algorithmically check BSPs – which are useful security properities in their own right – for pushdown system models; and (b) Mantel's framework is not beneficial for the purpose of algorithmic verification of noninterference properties for pushdown systems, unlike the case of finite-state systems.

Systematic approaches investigating the decidability of noninterference properties for infinite state systems are, to the best of our knowledge, still missing. In the context of language based notions of noninterference, [4] shows that the notion of strong low bisimulation as defined by [21] is decidable for a simple parallel while language. Alternative notions of noninterference, defined for example in [3], turn out to be undecidable.

In the second part of the paper we consider a natural security property we call *Weak Non-inference* or *WNI*. The property essentially says that by observing a visible system trace a low-level user cannot pinpoint the exact sequence of confidential events that have taken place in the system. The property is thus in the the same spirit as the BSPs and noninterference properties as they all say that by seeing a visible trace a low-level user cannot infer "too much" about the confidential events that have taken place in the system. In fact, *WNI* can be seen to be weaker than all these properties. We show that the problem of model-checking *WNI* is undecidable not just for pushdown systems but for *finite-state* systems as well.

This result is interesting for a couple of reasons: Firstly, it follows from our results that *WNI*, an interesting noninterference property, is *not* definable in Mantel's BSP framework. Secondly, this result sheds some light on a broader question: Is there a natural first-order logic which can express properties like the BSPs *and* which can be model-checked for finite-state systems? We note that BSPs make use of the operations of concatenation (implicitly) and projection to a subset of the alphabet. By earlier undecidability results in the literature ([19]) a general FO logic that allows concatenation is necessarily undecidable. By our undecidability result for *WNI*, it follows that a FO logic that uses projection (and negation, which BSPs don't use) is necessarily undecidable.

The rest of the paper is organized as follows. Section 2 defines the technical terms used in the paper. Section 3 proves the undecidability of model-checking BSPs for pushdown systems. Section 4 introduces a property called "Weak Non Inference", which cannot be characterized in terms of BSPs and gives its decidability results. Finally Section 5 concludes the article.

2 Preliminaries

By an alphabet we will mean a finite set of symbols representing *events* or *actions* of a system. For an alphabet Σ we use Σ^* to denote the set of finite strings over Σ, and Σ^+ to denote the set of non-empty strings over Σ. The null or empty string is represented by the symbol ϵ. For two strings α and β in Σ^* we write $\alpha\beta$ for the concatenation of α followed by β. A *language* over Σ is just a subset of Σ^*.

We fix an alphabet of events Σ and assume a partition of Σ into V, C, N, which in the framework of [13] correspond to events that are *visible*, *confidential*, and *neither* visible nor confidential, from a particular user's point of view.

Let $X \subseteq \Sigma$. The projection of a string $\tau \in \Sigma^*$ to X is written $\tau \upharpoonright_X$ and is obtained from τ by deleting all events that are not elements of X. The projection of the language L to X, written $L \upharpoonright_X$, is defined to be $\{\tau \upharpoonright_X \mid \tau \in L\}$.

A *(labelled) transition system* over an alphabet Σ is a structure of the form $\mathcal{T} = (Q, s, \longrightarrow)$, where Q is a set of states, $s \in Q$ is the start state, and $\longrightarrow \subseteq Q \times \Sigma \times Q$ is the transition relation. We write $p \xrightarrow{a} q$ to stand for $(p, a, q) \in \longrightarrow$, and use $p \xrightarrow{w}^* q$ to denote the fact that we have a path labelled w from p to q in the underlying graph of the transition system $\mathcal{T}$. If some state q has an edge labelled a, then we say a is enabled at q.

The language generated by $\mathcal{T}$ is defined to be

$$L(\mathcal{T}) = \{\alpha \in \Sigma^* \mid \text{there exists a } t \in Q \text{ such that } s \xrightarrow{\alpha}^* t\}.$$

We begin by recalling the *basic security predicates* (BSPs) of Mantel [15]. It will be convenient to use the notation $\alpha =_Y \beta$ where $\alpha, \beta \in \Sigma^*$ and $Y \subseteq \Sigma$, to mean α and β are the same "modulo a correction on Y-events". More precisely, $\alpha =_Y \beta$ iff $\alpha \upharpoonright_{\overline{Y}} = \beta \upharpoonright_{\overline{Y}}$, where $\overline{Y}$ denotes $\Sigma - Y$. By extension, for languages L and M over Σ, we say $L \subseteq_Y M$ iff $L \upharpoonright_{\overline{Y}} \subseteq M \upharpoonright_{\overline{Y}}$.

In the definitions below, we assume L to be a language over Σ.

1. L satisfies R *(Removal of events)* iff for all $\tau \in L$ there exists $\tau' \in L$ such that $\tau' \restriction_C = \epsilon$ and $\tau' \restriction_V = \tau \restriction_V$.
2. L satisfies D *(stepwise Deletion of events)* iff for all $\alpha c \beta \in L$, such that $c \in C$ and $\beta \restriction_C = \epsilon$, we have $\alpha' \beta' \in L$ with $\alpha' =_N \alpha$ and $\beta' =_N \beta$.
3. L satisfies I *(Insertion of events)* iff for all $\alpha\beta \in L$ such that $\beta \restriction_C = \epsilon$, and for every $c \in C$, we have $\alpha' c \beta' \in L$, with $\beta' =_N \beta$ and $\alpha' =_N \alpha$.
4. Let $X \subseteq \Sigma$. Then L satisfies *IA (Insertion of Admissible events)* w.r.t X iff for all $\alpha\beta \in L$ such that $\beta \restriction_C = \epsilon$ and for some $c \in C$, there exists $\gamma c \in L$ with $\gamma \restriction_X = \alpha \restriction_X$, we have $\alpha' c \beta' \in L$ with $\beta' =_N \beta$ and $\alpha' =_N \alpha$.
5. L satisfies *BSD (Backwards Strict Deletion)* iff for all $\alpha c \beta \in L$ such that $c \in C$ and $\beta \restriction_C = \epsilon$, we have $\alpha \beta' \in L$ with $\beta' =_N \beta$.
6. L satisfies *BSI (Backwards Strict Insertion)* iff for all $\alpha\beta \in L$ such that $\beta \restriction_C = \epsilon$, and for every $c \in C$, we have $\alpha c \beta' \in L$, with $\beta' =_N \beta$.
7. Let $X \subseteq \Sigma$. Then L satisfies *BSIA (Backwards Strict Insertion of Admissible events)* w.r.t X iff for all $\alpha\beta \in L$ such that $\beta \restriction_C = \epsilon$ and there exists $\gamma c \in L$ with $c \in C$ and $\gamma \restriction_X = \alpha \restriction_X$, we have $\alpha c \beta' \in L$ with $\beta' =_N \beta$.
8. Let $X \subseteq \Sigma$, $V' \subseteq V$, $C' \subseteq C$, and $N' \subseteq N$. Then L satisfies *FCD (Forward Correctable Deletion)* w.r.t V', C', N' iff for all $\alpha c v \beta \in L$ such that $c \in C'$, $v \in V'$ and $\beta \restriction_C = \epsilon$, we have $\alpha \delta v \beta' \in L$ where $\delta \in (N')^*$ and $\beta' =_N \beta$.
9. Let, $V' \subseteq V$, $C' \subseteq C$, and $N' \subseteq N$. Then L satisfies *FCI (Forward Correctable Insertion)* w.r.t C', V', N' iff for all $\alpha v \beta \in L$ such that $v \in V'$, $\beta \restriction_C = \epsilon$, and for every $c \in C'$ we have $\alpha c \delta v \beta' \in L$, with $\delta \in (N')^*$ and $\beta' =_N \beta$.
10. Let $X \subseteq \Sigma$, $V' \subseteq V$, $C' \subseteq C$, and $N' \subseteq N$. Then L satisfies *FCIA (Forward Correctable Insertion of admissible events)* w.r.t. X, V', C', N' iff for all $\alpha v \beta \in L$ such that: $v \in V'$, $\beta \restriction_C = \epsilon$, and there exists $\gamma c \in L$, with $c \in C'$ and $\gamma \restriction_X = \alpha \restriction_X$; we have $\alpha c \delta v \beta' \in L$ with $\delta \in (N')^*$ and $\beta' =_N \beta$.
11. L satisfies *SR (Strict Removal)* iff for all $\tau \in L$ we have $\tau \restriction_{\overline{C}} \in L$.
12. L satisfies *SD (Strict Deletion)* iff for all $\alpha c \beta \in L$ such that $c \in C$ and $\beta \restriction_C = \epsilon$, we have $\alpha\beta \in L$.
13. L satisfies *SI (Strict Insertion)* iff for all $\alpha\beta \in L$ such that $\beta \restriction_C = \epsilon$, and for every $c \in C$, we have $\alpha c \beta \in L$.
14. Let $X \subseteq \Sigma$. L satisfies *SIA (Strict Insertion of Admissible events)* w.r.t X iff for all $\alpha\beta \in L$ such that $\beta \restriction_C = \epsilon$ and there exists $\gamma c \in L$ with $c \in C$ and $\gamma \restriction_X = \alpha \restriction_X$, we have $\alpha c \beta \in L$.

We say a Σ-labelled transition system $\mathcal{T}$ satisfies a BSP iff $L(\mathcal{T})$ satisfies the BSP.

3 BSPs and Pushdown Systems

The model-checking problem for BSPs is the following: Given a BSP P and a system modelled as transition system $\mathcal{T}$, does $\mathcal{T}$ satisfy P? The problem was shown to be decidable when the system is presented as a finite-state transition system [8]. In this section we show that if the system is modelled as a pushdown system, the model-checking problem becomes undecidable. We use a reduction from the emptiness problem of Turing machines, which is known to be undecidable.

This section is organized as follows. First, we introduce Pushdown Systems and show that they accept exactly the class of prefix-closed context-free languages. Then we show for the BSP D that verifying D for Pushdown systems is undecidable by reducing the emptiness problem for Turing machines to this problem. More concretely, for a given Turing machine M we construct a prefix closed context-free language L_M such that L_M satisfies BSP D iff the language $L(M)$ of the turing machine M is empty. By adjusting the prefix-closed context-free language L_M appropriately, we can show the same for all other BSPs defined in Mantel's MAKS framework. We will give all the respective languages for the other BSPs in this chapter. The detailed undecidability proofs can be found in the technical report [6].

We assume that Σ, the set of events, contains two visible events v_1 and v_2, and one confidential event c.

A *pushdown system (PDS)* given by $P = (Q, \Sigma_P, \Gamma_P, \Delta, s, \bot)$ consists of a finite set of control states Q, a finite input alphabet Σ_P, a finite stack alphabet Γ_P, and a transition relation $\Delta \subseteq ((Q \times (\Sigma_P \cup \{\epsilon\}) \times \Gamma_P) \times (Q \times \Gamma_P^*))$, a starting state $s \in Q$ and an initial stack symbol $\bot \in \Gamma_P$. If $((p, a, A), (q, B_1 B_2 \cdots B_k)) \in \Delta$, this means that whenever the machine is in state p reading input symbol a on the input tape and A on top of the stack, it can pop A off the stack, push $B_1 B_2 \cdots B_k$ onto the stack (such that B_1 becomes the new stack top symbol), move its read head right one cell past the a, and enter state q. If $((p, \epsilon, A), (q, B_1 B_2 \cdots B_k)) \in \Delta$, this means that whenever the machine is in state p and A on top of the stack, it can pop A off the stack, push $B_1 B_2 \cdots B_k$ onto the stack (such that B_1 becomes the new stack top symbol), leave its read head unchanged and enter state q.

A configuration of pushdown system P is an element of $Q \times \Sigma_P^* \times \Gamma_P^*$ describing the current state, the portion of the input yet unread and the current stack contents. For example, when x is the input string, the start configuration is $(s, x, \bot)$. The next configuration relation $\longrightarrow$ is defined for any $y \in \Sigma_P^*$ and $\beta \in \Gamma_P^*$, as $(p, ay, A\beta) \longrightarrow (q, y, \gamma\beta)$ if $((p, a, A), (q, \gamma)) \in \Delta$ and $(p, y, A\beta) \longrightarrow (q, y, \gamma\beta)$ if $((p, \epsilon, A), (q, \gamma)) \in \Delta$. Let $\longrightarrow^*$ be defined as the reflexive transitive closure of $\longrightarrow$. Then P accepts w iff $(s, w, \bot) \longrightarrow^* (q, \epsilon, \gamma)$ for some $q \in Q$, $\gamma \in \Gamma_P^*$.

Pushdown systems also appear in other equivalent forms in literature. For example *Recursive state machines (RSM's)* [1] and *Boolean programs* [2]. Pushdown systems then capture an interesting class of systems. The class of languages accepted by pushdown systems is closely related to context-free languages.

Lemma 1. *The class of languages accepted by pushdown systems is exactly the class of prefix closed context-free languages.*

Proof. Pushdown systems can be seen as a special case of pushdown automata with all its control states treated as final states. Hence pushdown systems generate a context-free language. It is easy to see that if a PDS accepts a string w, then it also accepts all the prefixes of w.

Conversely, let L be a prefix closed context-free language. Then there exists a context-free grammar (CFG) G generating L. Without loss of generality, we assume that G is in Greibach Normal Form with every non-terminal deriving a terminal string. A nondeterminstic pushdown automata P with a single state q, accepting by empty stack, with S (starting non-terminal in G) as the initial stack symbol, accepting exactly $L(G)$ can be constructed [12]. The idea of the construction is that for each production $A \rightarrow cB_1B_2 \cdots B_k$ in G, a transition $((q, c, A), (q, B_1B_2 \cdots B_k))$ is added to P. We now observe that if P has a run on w with γ as the string of symbols in stack, then there exists a left-most sentential form $S \longrightarrow^* w\gamma$ in G. Then every terminal prefix of a left-most sentential form in G has an extension in L (since every non- terminal derives a string). Now view P as a PDS P' (ignoring the empty stack condition for acceptance). Clearly P' accepts all the strings in L. Further, if P' has a run on some string w with γ as the string of symbols (corresponds to non-terminals in G) in stack, then w corresponds to a prefix of a left-most sentential form in G, and hence has an extension in L. Since L is a prefix closed, w also belongs to L. Hence $L(P') = L$.

We use the emptiness problem of Turing machines to show the undecidability of verifying BSPs for pushdown systems. Let M be a Turing machine defined as $M = (Q, \Sigma_M, \Gamma_M, \vdash, \sqcup, \delta, s, t, r)$, where Q is a finite set of states, Σ_M is a finite input alphabet, Γ_M is a finite tape alphabet containing Σ_M, $\vdash \in \Gamma_M \setminus \Sigma_M$ is the left endmarker, $\sqcup \in \Gamma_M \setminus \Sigma_M$ is the blank symbol, $\delta : Q \times \Gamma_M \rightarrow Q \times \Gamma_M \times \{L, R\}$ is the transition function, $s \in Q$ is the start state, $t \in Q$ is the accept state and $r \in Q$ is the reject state with $r \neq t$ and no transitions defined out of r and t. The configuration x of M at any instant is defined as a triple (q, y, n) where $q \in Q$ is a state, y is a non-blank finite string describing the contents of the tape and n is a non negative integer describing the head position. The next configuration relation $\rightsquigarrow$ is defined as $(p, a\beta, n) \rightsquigarrow (q, b\beta, n + 1)$, when $\delta(p, a) = (q, b, R)$. Similarly $(p, a\beta, n) \rightsquigarrow (q, b\beta, n - 1)$, when $\delta(p, a) = (q, b, L)$.

We can encode configurations as finite strings over the alphabet $\Gamma_M \times (Q \cup \{-\})$, where $- \notin Q$. A pair in $\Gamma_M \times (Q \cup \{-\})$ is written vertically with the element of Γ_M on top. We represent a computation of M as a sequence of configurations x_i separated by $\#$. It will be of the form $\#x_1\#x_2\# \cdots \#x_n\#$. Each letter in $(\Gamma_M \times (Q \cup \{-\})) \cup \{\#\}$ can be encoded as a string of v_1's and a string (computation) in $(\Gamma_M \times (Q \cup \{-\})) \cup \{\#\}$ can be represented as a string of v_1's and v_2's, with v_2 used as the separator. Recall the assumption that Σ contains v_1, v_2 (visible events) and c (confidential event). For a computation w, we use $enc(w)$ to denote this encoding.

We now show that the problem of verifying BSP D for pushdown systems is as hard as checking the language emptiness problem of a Turing machine. To do so, we construct a language L_M out of a given Turing machine M and show, that L_M satisfies BSP D iff $L(M)$ is empty. Afterwards we show, that L_M can be generated by a Pushdown system and is hence a prefix closed context-free language.

Let $M = (Q, \varSigma_M, \varGamma_M, \vdash, \sqcup, \delta, s, t, r)$ be a Turing machine. Consider the language L_M to be the prefix closure of $L_1 \cup L_2$ where

$$L_1 = \{c \cdot enc(\#x_1\#x_2 \cdots x_n\#) \mid x_1 \text{ is a starting configuration, } x_n \text{ is an accepting configuration}\}$$

$$L_2 = \{enc(\#x_1\#x_2 \cdots x_n\#) \mid x_1 \text{ is a starting configuration, } x_n \text{ is an accepting configuration, exists } i : x_i \rightsquigarrow x_{i+1} \text{ invalid transition}\}$$

Lemma 2. *L_M satisfies BSP D iff $L(M) = \emptyset$.*

Proof. ($\Leftarrow$:) Let us assume $L(M) = \emptyset$. Consider any string containing a confidential event in L_M. The string has to be of the form $c\tau$ in L_1 or a prefix of it. τ cannot be a valid computation (encoded) of M, since $L(M) = \emptyset$. So, if we delete the last c, we will get τ, which is included in L_2. Also, all the prefixes of $c\tau$ and τ will be in L_M, as it is prefix closed. Hence L_M satisfies D.

($\Rightarrow$:) If $L(M)$ is not empty then there exists some string τ which is a valid computation (encoded). L_1 contains $c\tau$. If we delete the last c, the resulting string τ is not present in L_M. Hence D is not satisfied. Hence $L(M)$ is empty, when L_M satisfies the BSP D.

To complete the reduction, we need to show, that L_M is a prefix-closed context-free language, and we do this by translating M into a PDS accepting L_M.

Since CFLs are closed under prefix operation (adding the prefixes of the strings in the language) and as prefix closed CFLs are equivalent to PDS (from Lemma 1), it is enough to show that $L_1 \cup L_2$ is a CFL.

Let $T = (\varGamma_M \times (Q \cup \{-\})) \cup \{\#\}$. Consider the above defined language $L_2 \subseteq T^*$ (with the strings being unencoded versions).

$$L_2 = \{\#x_1\#x_2\# \cdots \#x_n\# \mid x_1 \text{ is a starting configuration, } x_n \text{ is an accepting configuration, exists } i : x_i \rightsquigarrow x_{i+1} \text{ invalid transition}\}$$

L_2 can be thought of as the intersection of languages A_1, A_2, A_3, A_4 and A_5, where

$A_1 = \{w \mid w \text{ begins and ends with } \#\}$
$A_2 = \{w \mid \text{ the string between any two } \#\text{s must be a valid configuration}\}$
$A_3 = \{\#x\#w \mid x \in (T \setminus \{\#\})^* \text{ is a valid starting configuration}\}$
$A_4 = \{w\#x\# \mid x \in (T \setminus \{\#\})^* \text{ is a valid accepting configuration}\}$
$A_5 = \{\#x_1\# \cdots \#x_n\# \mid \text{exists some } i, \text{ with } x_i \rightsquigarrow x_{i+1} \text{ not a valid transition}\}$

We now show that the languages from A_1 to A_4 are regular and A_5 is a CFL. Since CFLs are closed under intersection with regular languages, L_2 will be shown to be context-free. Note that L_1 (unencoded) is the intersection of A_1, A_2, A_3 and A_4, and hence regular. Since CFLs are closed under union, $L_1 \cup L_2$ will also be context-free.

The language A_1 can be generated by the regular expression $\#(T \setminus \{\#\})^*\#$. For A_2, we only need to check that between every #'s there is exactly one symbol with a state q on the bottom, and $\vdash$ occurs on the top immediately after each # (except the last) and nowhere else. This can be easily checked with a finite automaton. The set A_3 can be generated by the regular expression $\#(\vdash, s)K^*\#T^*$, with $K = \Gamma \setminus \{\vdash\} \times \{-\}$. To check that a string is in A_4, we only need to check that t appears someplace in the string. This can be easily checked by an automaton. For A_5, we construct a nondeterministic pushdown automaton (NPDA). The NPDA nondeterministically guesses i for x_i and then it scans to the three-length substring in x_i with a state in the middle component of the three-length string and checks with the corresponding three-length substring in x_{i+1} using the stack. Then, these substrings are checked against the transition relation of M, accepting if it is not a valid transition. Interested readers are referred to [12], for detailed proofs of above languages to be regular and context-free languages. Now, it follows that L_1 is regular and L_2 is context-free. As languages accepted by pushdown systems (Lemma 1) are equivalent to prefix closed context-free languages, L_M is a language of a pushdown system.

We have shown, that the prefix closed context-free language L_M satisfies BSP D iff the language $L(M)$ of Turing machine M is empty. Since the emptiness problem of Turing machines is undecidable, we get the following theorem.

Lemma 3. *The problem of verifying BSP D for pushdown systems is undecidable.*

Undecidability for the rest of the BSPs can be shown in a similar fashion. For the BSPs R, SR, SD, BSD we can use the same language L_M and get

Lemma 4. *L_M satisfies BSP R (and SR and SD and BSD) iff $L(M) = \emptyset$.*

For the other BSPs we need to adapt the languages appropriately as shown in the following. The detailed proofs for these languages can be found in the technical report [6].

To show undecidability of BSPs I, BSI, and SI, we consider L^I_M to be the prefix closure of $L_1 \cup L_2$ where

$$\begin{aligned} L_1 = \{enc(\#x_1\#x_2\cdots x_n\#) \mid\ & x_1 \text{ is a starting configuration,} \\ & x_n \text{ is an accepting configuration}\} \end{aligned}$$

$$\begin{aligned} L_2 = \{w \in (\{v_1, v_2\} \cup C \cup N)^* \mid\ & w\restriction_{\{v_1,v_2\}} = enc(\#x_1\#x_2\cdots x_n\#), \\ & x_1 \text{ is a starting configuration,} \\ & x_n \text{ is an accepting configuration,} \\ & \text{exists } i : x_i \rightsquigarrow x_{i+1} \text{ invalid transition}\} \end{aligned}$$

Lemma 5. *L^I_M satisfies BSP I (and SI and BSI) iff $L(M) = \emptyset$.*

To show undecidability of BSPs IA^X, $BSIA^X$, and SIA^X with $X \subseteq \Sigma$, we consider L_M^{IAX} to be the prefix closure of $L_1 \cup L_2 \cup L_3$ where

$$\begin{aligned} L_1 = \{enc(\#x_1\#x_2\cdots x_n\#) \mid\ & x_1 \text{ is a starting configuration,}\\ & x_n \text{ is an accepting configuration}\} \end{aligned}$$

$$\begin{aligned} L_2 = \{w \in (\{v_1, v_2\} \cup C)^* \mid\ & w\restriction_{\{v_1,v_2\}} = enc(\#x_1\#x_2\cdots x_n\#),\\ & x_1 \text{ is a starting configuration,}\\ & x_n \text{ is an accepting configuration,}\\ & \text{exists } i : x_i \rightsquigarrow x_{i+1}\text{invalid transition}\} \end{aligned}$$

$$L_3 = X^* \cdot C$$

Lemma 6. *L_M^{IAX} satisfies BSP IA^X (and SIA^X and $BSIA^X$) iff $L(M) = \emptyset$.*

To show undecidability of BSP *FCD* we assume $V' \subseteq V, N' \subseteq N, C' \subseteq C$ to be given and $v_2 \in V'$ and $c \in C'$. We consider language L_M^{FCD} to be the prefix closure of $L_1 \cup L_2$ where

$$\begin{aligned} L_1 = \{cv_2 \cdot enc(x_1x_2\cdots x_n) \mid\ & x_1 \text{ is a starting configuration,}\\ & x_n \text{ is an accepting configuration}\} \end{aligned}$$

$$\begin{aligned} L_2 = \{v_2 \cdot enc(x_1x_2\cdots x_n) \mid\ & x_1 \text{ is a starting configuration,}\\ & x_n \text{ is an accepting configuration,}\\ & \text{exists } i : x_i \rightsquigarrow x_{i+1} \text{ invalid transition}\} \end{aligned}$$

Lemma 7. *L_M^{FCD} satisfies BSP FCD iff $L(M) = \emptyset$.*

To prove undecidability of BSP *FCI* we assume $V' \subseteq V, N' \subseteq N, C' \subseteq C$ to be given and $v_2 \in V'$. We consider language L_M^{FCI} to be the prefix closure of $L_1 \cup L_2$ where

$$\begin{aligned} L_1 = \{v_2 \cdot enc(x_1x_2\cdots x_n) \mid\ & x_1 \text{ is a starting configuration,}\\ & x_n \text{ is an accepting configuration}\} \end{aligned}$$

$$\begin{aligned} L_2 = \{w \in (\{v_1, v_2\} \cup C')^* \mid\ & w\restriction_{\{v_1,v_2\}} = v_2 \cdot enc(x_1x_2\cdots x_n),\\ & x_1 \text{ is a starting configuration,}\\ & x_n \text{ is an accepting configuration,}\\ & \text{exists } i : x_i \rightsquigarrow x_{i+1} \text{ invalid transition}\} \end{aligned}$$

Lemma 8. *L_M^{FCI} satisfies FCI iff $L(M) = \emptyset$.*

Finally, to show undecidability of BSP $FCIA^X$, where $X \subseteq \Sigma$ we assume $V' \subseteq V, N' \subseteq N, C' \subseteq C$ to be given with $v_2 \in V'$. We consider language L_M^{FCIAX} to be the prefix closure of $L_1 \cup L_2 \cup L_3$ where

$$\begin{aligned} L_1 = \{v_2 \cdot enc(x_1 x_2 \cdots x_n) \mid\ & x_1 \text{ is a starting configuration,} \\ & x_n \text{ is an accepting configuration}\} \end{aligned}$$

$$\begin{aligned} L_2 = \{w \in (\{v_1, v_2\} \cup C')^* \mid\ & w \restriction_{\{v_1,v_2\}} = v_2 \cdot enc(x_1 x_2 \cdots x_n), \\ & x_1 \text{ is a starting configuration,} \\ & x_n \text{ is an accepting configuration,} \\ & \text{exists } i : x_i \rightsquigarrow x_{i+1} \text{ invalid transition}\} \end{aligned}$$

$$L_3 = X^* \cdot C'$$

Lemma 9. *L_M^{FCIAX} satisfies BSP $FCIA^X$ iff $L(M) = \emptyset$.*

The proofs of undecidability for these BSPs are given in the technical report [6]. We have shown that the BSPs defined by Mantel are undecidable for pushdown systems. Hence we get the following theorem.

Theorem 1. *The problem of model-checking any of the BSPs for pushdown systems is undecidable.*

4 Weak Non-Inference

In this section we introduce a natural information flow property which we call *Weak Non-Inference* (*WNI*) as it is weaker than most of the properties proposed in the literature. We show that model-checking this property even for finite-state transition systems is undecidable.

A set of traces L over Σ is said to satisfy *WNI* iff

$$\forall \tau \in L,\ \exists \tau' \in L : (\tau \restriction_C \neq \epsilon\ \Rightarrow\ (\tau \restriction_V = \tau' \restriction_V \wedge \tau \restriction_C \neq \tau' \restriction_C)).$$

The classical non-inference property can be phrased as $\forall \tau \in L,\ \exists \tau' \in L : (\tau' = \tau \restriction_V)$. Thus it is easy to see that any language that satisfies non-inference also satisfies *WNI*.

We show that checking whether a finite-state transition system satifies this property is undecidable, by showing a reduction from Post's Correspondence Problem (PCP). We recall that an instance of PCP is a collection of dominos $P = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ where each $x_i, y_i \in \Delta^+$ where Δ is a finite alphabet (recall that Δ^+ is the set of non-empty words over Δ). A *match* in P is a sequence $i_1, i_2, \ldots i_l$ such that $x_{i_1} \cdots x_{i_l} = y_{i_1} \cdots y_{i_l}$. The PCP problem is to determine if a given instance P has a match, and this problem is known to be undecidable.

We construct a transition system $\mathcal{T}_P = (Q, s_1, \rightarrow)$ on the alphabet $\Sigma' = V \cup C$, where $V = \{v_1, \ldots, v_n\}$, $C = \Delta$ and the transition relation $\rightarrow$ is as shown in Fig. 1. The states s_2 and s_3 have n loops each on them: state s_3 has loops on the strings $v_1 x_1 v_1$ through $v_n x_n v_n$, and s_3 has loops on the strings $v_1 y_1 v_1$ through $v_n y_n v_n$. We choose each v_i to be different from the symbols in Δ.

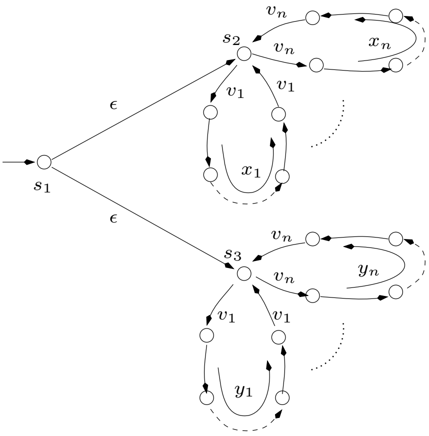

Fig. 1.

We call a string "interesting" if it is of the form $v_{i_1}w_{i_1}v_{i_1}v_{i_2}w_{i_2}v_{i_2}\cdots v_{i_k}w_{i_k}v_{i_k}$ for some $i_1,\ldots,i_k$ with $k \geq 1$, and w_{i_j} in Δ^* – in other words, the string must be non-empty and each visible alphabet occurs twice in succession. Thus every interesting string generated by $\mathcal{T}_P$ will end up in state s_2 or in state s_3. We observe that for every interesting string z in $L(\mathcal{T}_P)$, there is exactly one other string z' in $L(\mathcal{T}_P)$ with the same projection to V, which is moreover also an interesting string. If z passes through state s_2, then z' is the string which mimics the run of z but through s_3. Any other string will have a different projection to V.

Lemma 10. *A PCP instance P has a match iff $\mathcal{T}_P$ does not satisfy WNI.*

Proof. ($\Leftarrow$:) Suppose $\mathcal{T}_P$ does not satisfy *WNI*. Then there must exist a string τ in $L(\mathcal{T}_P)$ such that there is no other string in $L(\mathcal{T}_P)$ with the same projection to V and different projection to C. Now τ must be an interesting string – all uninteresting strings trivially satisfy the *WNI* condition since we can append or remove one confidential event from each of these strings. By our earlier observation, we know that there exists an interesting string τ' in $L(\mathcal{T}_P)$ which takes the symmetric path in $\mathcal{T}_P$ and has the same projection to V as τ. Now by our choice of τ, the projection of τ' on C is the same as the projection of τ on C. But this means that we have found a match for P, given by the indices corresponding to the loops traces out by τ and τ'.

($\Rightarrow$:) Let $i_1, i_2, \ldots, i_l$ be a match for P. Thus $x_{i_1}\cdots x_{i_l} = y_{i_1}\cdots y_{i_l}$. Consider the interesting string $\tau = v_{i_1}x_{i_1}v_{i_1}\cdots v_{i_l}x_{i_l}v_{i_l}$ in $L(\mathcal{T})$. Now there is exactly one other string τ' with the same projection to V, given by $v_{i_1}y_{i_1}v_{i_1}\cdots v_{i_l}y_{i_l}v_{i_l}$. However, as the indices chosen are a match for P, the projections of τ and τ' to C are identical. Thus $\mathcal{T}$ does not satisfy *WNI*. □

This completes the reduction of PCP to the problem of model-checking *WNI* for finite-state systems, and we conclude:

Theorem 2. *The problem of model-checking the property WNI for finite-state systems is undecidable.* □

We note that if the property *WNI* were expressible as a boolean combination of BSPs, the decision procedure for model-checking BSPs for finite-state systems in [8] would imply that model-checking *WNI* for finite-state systems is decidable. Hence we can conclude that:

Corollary 1. *The property WNI is* not *expressible as a boolean combination of Mantel's BSPs.* □

However, we can show that a restricted version of the problem is decidable. If we have a system model (finite-state or pushdown) that uses only *one* confidential event and *one* visible event, the problem of checking *WNI* becomes decidable.

Theorem 3. *The problem of model-checking WNI for pushdown systems when* $|V| = 1$ *and* $|C| = 1$*, is decidable.*

Proof. Consider an alphabet $\Sigma = \{v, c\}$, where v is the only visible event and c is the only confidential event. Recall that the *Parikh vector* of a string x over Σ, denoted $\psi(x)$, is the vector (n_v, n_c) where n_v is the number of occurrences of event v in x and n_c is the number of occurrences of event c in x. For a language L over Σ, its Parikh map $\psi(L)$ is defined to be the set of vectors $\{\psi(x) \mid x \in L\}$. By Parikh's theorem (see [12]), we know that whenever L is context-free, its Parikh map $\psi(L)$ forms a "semi-linear" set of vectors. To explain what this means, let us first introduce some notation. For a finite set of vectors $X = \{(n_1, m_1), \ldots, (n_k, m_k)\}$ we denote the set of vectors *generated* by X, with initial vector (n_0, m_0), to be the set $gen_{(n_0,m_0)}(X) = \{(n_0, m_0) + t_1(n_1, m_1) + \cdots + t_k(n_k, m_k) \mid t_1, \ldots, t_k \in \mathbb{N}\}$. Then $\psi(L)$ is semi-linear means that there exist finite non-empty sets of vectors $X_1, \ldots, X_l$, and initial vectors $(n_0^1, m_0^1), \ldots, (n_0^l, m_0^l)$, such that

$$\psi(L) = \bigcup_{i \in \{1,\ldots,l\}} gen_{(n_0^i, m_0^i)}(X_i).$$

Now let L be the given context-free language over the alphabet $\Sigma = \{v, c\}$, with Parikh map given by the sets $X_1, \ldots, X_l$, and initial vectors (n_0^1, m_0^1),. . . ,(n_0^l, m_0^l). Let each $X_i = \{(m_1^i, n_1^i), \ldots, (m_{k_i}^i, n_{k_i}^i)\}$. To verify *WNI* property for L, we need to check that for every vector in $\psi(L)$ there exists another vector which is in $gen_{(n_0^i, m_0^i)}(X_i)$ for some i, such that their visible event count is same, and confidential event count is different. For $l = 1$, L satisfies *WNI* iff the following formula in Presberger arithmetic[1] is valid:

[1] Presburger arithmetic is the first-order theory of natural numbers with addition which is known to be decidable in double-exponential time.

$$\begin{aligned}
&\forall t_1, \dots, t_{k_1} \in \mathbb{N}, \exists t'_1, \dots, t'_{k_1} \in \mathbb{N} \\
&\Big(n^1_0 + t_1 n^1_1 + \cdots + t_{k_1} n^1_{k_1} > 0 \implies \\
&\big(m^1_0 + t_1 m^1_1 + \cdots + t_{k_1} m^1_{k_1} = m^1_0 + t'_1 m^1_1 + \cdots + t'_{k_1} m^1_{k_1} \wedge \\
&n^1_0 + t_1 n^1_1 + \cdots + t_{k_1} n^1_{k_1} \neq n^1_0 + t'_1 n^1_1 + \cdots + t'_{k_1} n^1_{k_1}\big)\Big).
\end{aligned}$$

The validity of this formula can be found using the decision procedure for Presburger arithmetic. The formula is extended in the expected way for higher values of l.

In fact, any property which just asks for the counting relationship between visible and confidential events can be decided using the above technique.

5 Conclusions

In this paper we have studied the problem of model-checking Mantel's Basic Security Predicates for systems modelled as pushdown systems. We have shown that unlike the case of finite-state system models, this problem is undecidable. We also studied the model-checking problem for a property called *WNI* which is a weak form of the property of non-inference studied earlier in the literature. We show that this problem is undecidable, not just for pushdown systems but also for finite-state systems. It follows from this result that *WNI* is *not* expressible in Mantel's BSP framework. We also show that for a restricted class of systems (with only one visible and confidential event) this property is decidable for both finite-state and pushdown system models.

In future work we plan to see if similar reductions can be used to argue that the model-checking problem for noninterference properties in the literature (of which Mantel showed the BSPs to be the basic building blocks) – are also undecidable. Another open question is whether the model-checking problem continues to be undecidable even when we consider *deterministic* pushdown systems.

References

1. Alur, R., Etessami, K., Yannakakis, M.: Analysis of recursive state machines. In: Berry, G., Comon, H., Finkel, A. (eds.) CAV 2001. LNCS, vol. 2102, pp. 207–220. Springer, Heidelberg (2001)
2. Ball, T., Rajamani, S.K.: Bebop: A symbolic model checker for boolean programs. In: Havelund, K., Penix, J., Visser, W. (eds.) SPIN 2000. LNCS, vol. 1885, pp. 113–130. Springer, Heidelberg (2000)
3. Boudol, G., Castellani, I.: Noninterference for concurrent programs and thread systems. Theoretical Computer Science 1-2(281), 109–130 (2002)
4. Dam, M.: Decidability and proof systems for language-based noninterference relations. In: Proceedings POPL 2006, Charleston, South Carolina (2006)
5. Denning, D.E.: A lattice model of secure information flow. Commun. ACM 19(5), 236–243 (1976)

6. D'Souza, D., Holla, R., Kulkarni, J., Raghavendra, K.R., Sprick, B.: On the decidablity of model-checking information flow properties. Technical Report IISc-CSA-TR-2008-2 (2008)
7. D'Souza, D., Raghavendra, K.R.: Checking unwinding conditions for finite state systems. In: Proceedings of the VERIFY 2006 workshop, pp. 85–94 (2006)
8. D'Souza, D., Raghavendra, K.R., Sprick, B.: An automata based approach for verifying information flow properties. In: Proceedings of the second workshop on Automated Reasoning for Security Protocol Analysis (ARSPA 2005). ENTCS, vol. 135, pp. 39–58 (2005)
9. Focardi, R., Gorrieri, R.: A classification of security properties for process algebras. Journal of Computer Security 1, 5–33 (1995)
10. Goguen, J.A., Meseguer, J.: Security policies and security models. In: Proc. IEEE Symp. on Security and Privacy, April 1982, pp. 11–20 (1982)
11. Goguen, J.A., Meseguer, J.: Unwinding and inference control. In: Proc. IEEE Symp. on Security and Privacy, pp. 75–86 (April 1984)
12. Kozen, D.C.: Automata and Computability. Springer, Heidelberg (1997)
13. Mantel, H.: Possibilistic Definitions of Security – An Assembly Kit. In: Proceedings of the 13th IEEE Computer Security Foundations Workshop, Cambridge, UK, July 3–5, 2000, pp. 185–199. IEEE Computer Society, Los Alamitos (2000)
14. Mantel, H.: Unwinding Possibilistic Security Properties. In: Cuppens, F., Deswarte, Y., Gollmann, D., Waidner, M. (eds.) ESORICS 2000. LNCS, vol. 1895, pp. 238–254. Springer, Heidelberg (2000)
15. Mantel, H.: A Uniform Framework for the Formal Specification and Verification of Information Flow Security. PhD thesis, Universität des Saarlandes (2003)
16. McCullough, D.: Specifications for multilevel security and a hookup property. In: Proc. 1987 IEEE Symp. Security and Privacy (1987)
17. McLean, J.: A general theory of composition for trace sets closed under selective interleaving functions. In: Proc. IEEE Symposium on Research in Security and Privacy, pp. 79–93. IEEE Computer Society Press, Los Alamitos (1994)
18. O'Halloran, C.: A calculus of information flow. In: Proceedings of the European Symposium on Research in Computer Security, ESORICS 1990 (1990)
19. Quine, W.V.: Concatenation as a basis for finite arithmetic. J. Symbolic Logic 11(4) (1946)
20. Sabelfeld, A., Myers, A.: Language-based information-flow security. IEEE Journal on Selected Areas in Communications 21(1) (2003)
21. Sabelfeld, A., Sands, D.: A per model of secure information flow in sequential programs. Higher-Order and Symbolic Computation 1(14), 59–91 (2001)
22. Sutherland, D.: A model of information. In: Proceedings of the 9th National Computer Security Conference (1986)
23. van der Meyden, R., Zhang, C.: Algorithmic verification of noninterference properties. Electron. Notes Theor. Comput. Sci. 168, 61–75 (2007)
24. Zakinthinos, A., Lee, E.S.: A general theory of security properties. In: SP 1997: Proceedings of the 1997 IEEE Symposium on Security and Privacy, Washington, DC, USA, p. 94. IEEE Computer Society Press, Los Alamitos (1997)