On the Decidability of Model-Checking Information Flow Properties

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

[1] Dept. of Computer Sc. & Automation, Indian Institute of Science, India
{deepakd, raveendra, janardhan, raghavendrakr}@csa.iisc.ernet.in
[2] TU Darmstadt, Fachbereich Informatik, Modeling and Analysis of Information Systems, Hochschulstrasse 10, D-64289 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 [4]. 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 buisness-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. The two main research problems are, firstly, finding adequate, formal characterizations of a "secure system" and, secondly, developing sound and efficient verification techniques based on these characterizations. Information flow security has been a focal research topic in computer security for more than 30 years. Nevertheless,

the problem to secure the flow of information in systems is far from being solved. In [13], the state of the art was surveyed for approaches to capture and analyze information flow security of concrete programs. For information flow security at the level of more abstract specifications, a broad spectrum of approaches has been developed (see, e.g., [7, 11, 5]). The most popular verification techniques are the unwinding technique on the level of specifications (see, e.g., [8, 6]), and security type systems and program logics on the level of programs. In this article, we look at information flow security at the level of abstract specifications.

Noninterference is the basic semantical condition used to account for information flow security. Most often the concept is studied in the setting of multi-level security [1] with data assigned levels in a security lattice, such that levels higher in the lattice correspond to data of higher sensitivity. The question which noninterference aims to answer is one of *information flow*: 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 to a higher one might indicate a breach of integrity.

A system is viewed as generating traces containing "confidential" and "visible" events. BS THE ASSUMPTION IS THAT ANY "LOW LEVEL-USER", E.G. AN ATTACKER, KNOWS THE SET OF ALL POSSIBLE BEHAVIORS 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 about confidential events having taken place in the system. For example, the "non-inference" [12, 11, 15] property states that for every trace produced by the system, its projection to 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 not able to infer whether or not any high level behavior has taken place. There are other security properties defined in the literature: *separability* [11] (which requires that every possible low level behavior interleaved with every possible high level behaviour must be a possible behaviour of a system), *generalized non-interference* [10] (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* [14], *restrictiveness* [10], the *perfect security property* [15] etc.

In [9] 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 which are common to several properties of interest for the given system, need only be verified once for the system.

BS BS FOR ABSTRACT SYSTEMS there have been two approaches WHAT ABOUT SIMULATION? to the problem of verifying information flow properties for a given system: a traditional one based on "unwinding" [8, 6, 3] and the more recent "model-checking" technique in [4]. 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 [9] Mantel gives unwinding conditions for most of the BSPs he identifies. HOWEVER, FINDING A USEFUL UNWINDING RELATION REMAINS UP TO THE VERIFIER. The model-checking approach on the other hand is both sound and complete and relies on an automata-based approach (when the given system is finite-state) to check language-theoretic properties of the system. BS

In this article, we investigate the model-checking approach for the BSPs with respect to push-down systems and show, that all BSPs, and hence all classical non-intereference properties built out of them, are undecidable for push-down systems. SOME COMPARISON WITH MADS DAM'S RESULT OF DECIDABILITY OF NON-INTERFERENCE FOR STRONG LOW BISIMULATION??? BS BS

We also show that there are some interesting properties which cannot be characterized with respect to BSPs and we will give the decidability results for these properties with respect to finite state and push down systems. 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 5 introduces a property called "Weak Non Inference", which cannot be characterized in terms of BSPs and gives its decidability results. Finally Section 6 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. 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 [7] 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* (LTS) 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 will asssume in the sequel that all states in an LTS are reachable from the start state. We begin by recalling the *basic security predicates* (BSPs) of Mantel [9].

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 $\tau' \upharpoonright_C = \epsilon$ and $\tau' \upharpoonright_V = \tau \upharpoonright_V$.
2. L satisfies D *(stepwise Deletion of events)* iff for all $\alpha c \beta \in L$, such that $c \in C$ and $\beta \upharpoonright_C = \epsilon$, 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 $\beta \upharpoonright_C = \epsilon$, 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 $\beta \upharpoonright_C = \epsilon$ and for some $c \in C$, there exists $\gamma c \in L$ with $\gamma \upharpoonright_X = \alpha \upharpoonright_X$, 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 $\beta \upharpoonright_C = \epsilon$, 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 $\beta \upharpoonright_C = \epsilon$, 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 $\beta \upharpoonright_C = \epsilon$ and there exists $\gamma c \in L$ with $c \in C$ and $\gamma \upharpoonright_X = \alpha \upharpoonright_X$, 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 $\beta \upharpoonright_C = \epsilon$, 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 $v \in V'$, $\beta \upharpoonright_C = \epsilon$, 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: $v \in V'$, $\beta \upharpoonright_C = \epsilon$, and there exists $\gamma c \in L$, with $c \in C'$ and $\gamma \upharpoonright_X = \alpha \upharpoonright_X$; 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 $\tau \upharpoonright_{\overline{C}} \in L$.
12. L satisfies *SD (Strict Deletion)* iff for all $\alpha c \beta \in L$ such that $c \in C$ and $\beta \upharpoonright_C = \epsilon$, we have $\alpha \beta \in L$.
13. L satisfies *SI (Strict Insertion)* iff for all $\alpha \beta \in L$ such that $\beta \upharpoonright_C = \epsilon$, 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 $\beta \upharpoonright_C = \epsilon$ and there exists $\gamma c \in L$ with $c \in C$ and $\gamma \upharpoonright_X = \alpha \upharpoonright_X$, 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 [4]. 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 pushdown system 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 [2]. BS

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 $P = (Q, \Sigma_P, \Gamma_P, \Delta, s, \perp)$ 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 $\perp \in \Gamma_P$. 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 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 $(s, x, \perp)$. 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 $(s, w, \perp) \longrightarrow^* (q, \epsilon, \gamma)$ 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)* and *Boolean programming model* [**?**]. 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 [**?**]. 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 $M = (Q, \Sigma_M, \Gamma_M, \vdash, \sqcup, \delta, s, t, r)$ be a Turing machine. Consider the language L_M to be the prefix closure of $L_1 \cup L_2$ where

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

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

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

Proof. ($\Leftarrow$:) Let us assume $L(M) = \emptyset$. Now, 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.

The previos text was: To complete the reduction, we need to show how to translate M into a PDS accepting L_M. Question: Do we really need to do this to complete the reduction? Isn't it, that 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? BS

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

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

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 [**?**], 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 is equivalent to prefix closed context free languages (Lemma 1), L_M is a language of a pushdown system. Since the emptiness problem of Turing machine 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 can then be found in the appendix.

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

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

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

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

To show undecidability of BSPs IA^X, $BSIA^X$,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

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

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

$$L_3 = X \cdot \{c\}^*$$

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 $V', N', C' \subseteq \Sigma$ and to be given with $v_2 \in V'$ and $c \in C'$ and consider the new language L_M^{FCD} to be the prefix closure of $L_1 \cup L_2$ where

$$L_1 = \{cv_2 \cdot enc(x_1x_2 \cdots x_n) \mid x_1 \text{ is starting configuration}, x_n \text{ is accepting configuration}\}$$

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

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

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

$$L_1 = \{v_2 \cdot enc(x_1 x_2 \cdots x_n) \mid \; x_1 \text{ is starting configuration}, x_n \text{ is accepting configuration}\}$$

$$L_2 = \{c\}^* \cdot \{v_2 \cdot enc(x_1 x_2 \cdots x_n) \mid x_1 \text{ is starting configuration}, x_n \text{ is accepting configuration}, \text{exists } i : x_i \rightsquigarrow x_{i+1} \text{ invalid transition}\}$$

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

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

$$L_1 = \{v_2 \cdot enc(x_1 x_2 \cdots x_n) \mid \; x_1 \text{ is starting configuration}, x_n \text{ is accepting configuration}\}$$

$$L_2 = \{c\}^* \cdot \{v_2 \cdot enc(x_1 x_2 \cdots x_n) \mid x_1 \text{ is starting configuration}, x_n \text{ is accepting configuration}, \text{exists } i : x_i \rightsquigarrow x_{i+1} \text{ invalid transition}\}$$

$$L_3 = X \cdot \{c\}^*$$

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

The proofs of undecidability for these BSPs are given in the appendix. We have shown that the verifying 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 an information flow property weaker to *non-inference* called *Weak Non-Inference*(*WNI*). We show that model-checking this property for finite state systems is undecidable.

Given a set of traces L, *WNI* is defined as

$$\forall \tau \in L,\ \exists \tau' \in L \ \ (\tau \upharpoonright_C \neq \epsilon \ \Rightarrow \ \tau \upharpoonright_V = \tau' \upharpoonright_V \wedge \tau \upharpoonright_C \neq \tau' \upharpoonright_C)$$

It can be easily seen that Non-inference implies *WNI*. It turns out that *WNI* cannot be characterized in terms of BSP's. We show that this property is undecidable for finite state systems by reducing the post correspondence problem (PCP). An instance P of the PCP is a collection of dominos $\{(a_1, b_1),$

$(a_2, b_2), \cdots (a_n, b_n)\}$ where each $a_i, b_i \in \Sigma^+, i = 1 \cdots n$ and a match is a sequence $i_1, i_2, \ldots i_l$ such that $a_{i_1} \cdots a_{i_l} = b_{i_1} \cdots b_{i_l}$. The problem to determine if the given instance P has a match is known to be undecidable. We construct an automata (refer Fig. 2) $A_P = (Q, s_1, F)$ on the alphabet $\Sigma' = V \cup C$, where $V = \{v_1, \cdots, v_n\}$, $C = \Sigma$ and $F = \{s_1, s_2, s_3\}$. Without loss of generality we assume that each v_i is different and does not occur in Σ. Note that ϵ represents empty string and the loops in the Fig. 2 represent n cycles with fresh intermediate states, each starting with unique single visible alphabet v_i.

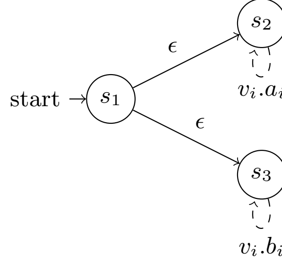

Fig. 1. Automata A_P

Lemma 10. *For every non-empty string in $L(A_P)$ there is exactly one another string in $L(A_P)$ with the same projection on set V.*

Proof. Consider a non-empty string τ in the language $L(A_P)$ with some number of visible events. Then the path τ either comes from the run $s_1, s_2, \cdots$ or $s_1, s_3, \cdots$. In either case, we can construct τ' from the other run by taking the loops in the same order as that of τ so that it will have same projection to V as τ. Further, any other string has to make transitions by going through loops in s_2 or s_3 in some different order which makes its projection to V differ from that of τ.

Theorem 2. *A PCP instance P has a match iff $L(A_P)$ does not satisfy WNI.*

Proof. ($\Leftarrow$:) Let $L(A_P)$ does not satisfy *WNI*. Then there exists a non-empty string τ in $L(A_P)$ such that there is no other string in $L(A)$ with the same projection to V and different projection to C. From Lemma 11, there exists a string τ' with same projection to V. So, its projection on C must also be same (otherwise it would satisfy *WNI*). This means that we have got indices $i_1, i_2, \cdots, i_l$ such that there is a match.

($\Rightarrow$:) If match exists for P, then there exists indices $i_1, i_2, \cdots, i_l$ such that $a_{i_1} \cdots a_{i_l} = b_{i_1} \cdots b_{i_l}$. Let $\tau = v_{i_1}.a_{i_1}.v_{i_2}.a_{i_2} \cdots v_{i_l}.a_{i_l}$ in $L(A_P)$. Then from the Lemma 11 there exists exactly one string $\tau' = v_{i_1}.b_{i_1}.v_{i_2}.b_{i_2} \cdots v_{i_l}.b_{i_l}$ with same projection to V. As this corresponds to a match, the projection to C is also same. Hence $L(A_P)$ does not satisfy *WNI*.

5 Weak Non Inference

In this section, we introduce an information flow property weaker to *non-inference* called *Weak Non-Inference*(*WNI*). We show that model-checking this property for labelled transition systems is undecidable.

Given a set of traces L, *WNI* is defined as

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

It can be easily seen that Non-inference implies *WNI*. It turns out that *WNI* cannot be characterized in terms of BSP's. We show that this property is undecidable for labelled transition systems by reducing the post correspondence problem (PCP). An instance P of the PCP is a collection of dominos $\{(a_1, b_1), (a_2, b_2), \cdots (a_n, b_n)\}$ where each $a_i, b_i \in \Sigma^+, i = 1 \cdots n$ and a match is a sequence $i_1, i_2, \ldots i_l$ such that $a_{i_1} \cdots a_{i_l} = b_{i_1} \cdots b_{i_l}$. The problem to determine if the given instance P has a match is known to be undecidable.

We construct a labelled transition system $\mathcal{T} = (Q, s_1, \longrightarrow)$ on the alphabet $\Sigma' = V \cup C$, where $V = \{v_1, \cdots, v_n\}$, $C = \Sigma$ and $\longrightarrow$ as shown in Fig. 2. Without loss of generality we assume that each v_i is different and does not occur in Σ. Note that ϵ represents empty string and the loops in the Fig. 2 represent n cycles with fresh intermediate states, each starting and ending with unique single visible alphabet v_i.

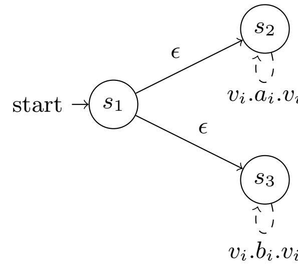

Fig. 2. Automata A_P

We call a string as interesting if it is non empty and each visible alphabet occurs twice consecutively. Note that, every interesting string will end up in state s_2 or in state s_3.

Lemma 11. *For every interesting string in $L(\mathcal{T})$ there is exactly one another string in $L(\mathcal{T})$ with the same projection on set V and that string will also be an interesting string.*

Proof. Consider an interesting string τ in the language $L(\mathcal{T})$ with some number of visible events. Then the path τ either comes from the run $s_1, s_2, \cdots, s_2$ or

$s_1, s_3, \cdots, s_3$. In either case, we can construct τ' from the other run by taking the loops in the same order as that of τ so that it will have same projection to V as τ. By the construction of τ' itself, it is also an interesting string. Further, any other string has to make transitions by going through loops partially or in some different order in states s_2 or s_3 which makes its projection to V differ from that of τ.

Theorem 3. *A PCP instance P has a match iff $L(\mathcal{T})$ does not satisfy WNI.*

Proof. ($\Leftarrow$:) Let $L(\mathcal{T})$ does not satisfy *WNI*. Then there must exist an interesting string τ in $L(\mathcal{T})$ such that there is no other string in $L(\mathcal{T})$ with the same projection to V and different projection to C (all uninteresting strings trivially satisfy this property because we can append or remove one confidential event from each of these strings). But from Lemma 11, there exists a string τ' with same projection to V. So, its projection on C must also be same (otherwise it would satisfy *WNI*). This means that we have got indices $i_1, i_2, \cdots, i_l$ such that there is a match.

($\Rightarrow$:) If match exists for P, then there exists indices $i_1, i_2, \cdots, i_l$ such that $a_{i_1} \cdots a_{i_l} = b_{i_1} \cdots b_{i_l}$. Consider an interesting string $\tau = v_{i_1}.a_{i_1}.v_{i_1}. \cdots v_{i_l}.a_{i_l}.v_{i_l}$ which is in $L(\mathcal{T})$. Then from the Lemma 11 there exists exactly one string $\tau' = v_{i_1}.b_{i_1}.v_{i_1}. \cdots v_{i_l}.b_{i_l}.v_{i_l}$ with same projection to V. As this corresponds to a match, the projection to C is also same. Hence $L(\mathcal{T})$ does not satisfy *WNI*.

Lemma 12. *WNI is undecidable even when number of confidential events is two.*

Proof. Since PCP is undecidable for an alphabet size of two, above proof holds true for this lemma as well.

Lemma 13. *WNI is decidable for pushdown systems when $|V| = 1$ and $|C| = 1$.*

Proof. Consider any context free language L defined over alphabet set $\Sigma = \{v, c\}$ where v is the visible alphabet and c is the confidential alphabet set. Let Parikh vector of a string s be a vector (n_v, n_c), where n_v counts number of occurrences of alphabet v and n_c counts c. By Parikh's theorem, Parikh vectors forms a semi-linear subset of $\mathbb{N}^k$. This means, we can find finite collection $VC_1, \ldots, VC_k$ of finite set of vectors where each collection VC_i will be a set of vectors of the form $(v_0, c_0), ..., (v_n, c_n)$ with initial vector as (v_0, t_0). Every Parikh vector can be stated as $(v_0, c_0) + t_1(v_1, c_1) + \cdots + t_n(v_n, c_n)$ for some VC_i and for some $t_1, \cdots, t_n \in \mathbb{N}$. Conversely, for every linear combination of vectors with in VC_i, there will exist a Parikh vector corresponding to some string present in the language.

In this setting, to verify *WNI* property, we need to check whether for every Parikh vector, there exists another Parikh vector which is a linear combination of vectors of one of the the VC's such that, their visible alphabet count is same and confidential alphabet count is different. For $k = 1$, *WNI* for L can be stated as:

$$
\begin{aligned}
&\forall t_1, \ldots, t_n \in \mathbb{N}, \exists t'_1, \ldots, t'_n \in \mathbb{N} \\
&\Big(c_0 + t_1.c_1 + \ldots + t_n.c_n > 0 \implies \\
&\big(v_0 + t_1.v_1 + \ldots + t_n.v_n = v_0 + t'_1.v_1 + \ldots + t'_n.v_n \ \bigwedge \\
&c_0 + t_1.c_1 + \ldots + t_n.c_n \neq c_0 + t'_1.c_1 + \ldots + t'_n.c_n\big)\Big)
\end{aligned}
$$

The truth value of this equation can be found using Pressburger arithmetic and if it is true then the given context-free language follows *WNI* property. It is possible to extend the equation for higher values of k by taking all the combination of VC's for the existance of another Parikh vector.

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

6 Conclusions

In this paper, we analyzed the model-checking technique for verifying information flow properties with respect to abstract specifications. Specifically we presented two results.

1. Model checking BSPs is undecidable for pushdown systems. This implies all the properties which are expressible in Mantel's framework are undecidable with respect to pushdown systems.
2. There are some interesting properties outside Mantel's framework. We have given one such property and shown that model-checking this property for finite state systems is undecidable.

BS Diskussion zusammenhang zwischen mads dams result und unserem!!

References

1. Dorothy E. Denning. A lattice model of secure information flow. *Commun. ACM*, 19(5):236–243, 1976.
2. Deepak D'Souza, Raveendra Holla, Janardhan Kulkarni, Raghavendra K R, and Barbara Sprick. Model-Checking Information Flow Properties. In *Technical Report IISc-CSA-TR-2008-2*.
3. Deepak D'Souza and Raghavendra K R. Checking unwinding conditions for finite state systems. In *Proceedings of the VERIFY'06 workshop*, pages 85–94, 2006.
4. Deepak D'Souza, Raghavendra K R, and Barbara Sprick. 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*, volume 135, pages 39–58, 2005.
5. J. A. Goguen and J. Meseguer. Security policies and security models. pages 11–20, April 1982.

6. J. A. Goguen and J. Meseguer. Unwinding and inference control. pages 75–86, April 1984.
7. Heiko Mantel. Possibilistic Definitions of Security – An Assembly Kit. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop*, pages 185–199, Cambridge, UK, July 3–5 2000. IEEE Computer Society.
8. Heiko Mantel. Unwinding Possibilistic Security Properties. In F. Cuppens, Y. Deswarte, D. Gollmann, and M. Waidner, editors, *European Symposium on Research in Computer Security (ESORICS)*, LNCS 1895, pages 238–254, Toulouse, France, October 4-6 2000. Springer.
9. Heiko Mantel. *A Uniform Framework for the Formal Specification and Verification of Information Flow Security.* PhD thesis, Universität des Saarlandes, 2003.
10. Daryl McCullough. Specifications for multilevel security and a hookup property. In *Proc. 1987 IEEE Symp. Security and Privacy*, 1987.
11. John McLean. A general theory of composition for trace sets closed under selective interleaving functions. In *Proc. IEEE Symposium on Research in Security and Privacy*, pages 79 – 93. IEEE Computer Society Press, 1994.
12. Colin O'Halloran. A calculus of information flow. In *Proceedings of the European Symposium on Research in Computer Security, ESORICS 90*, 1990.
13. A. Sabelfeld and A. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21(1), 2003.
14. David Sutherland. A model of information. In *Proceedings of the 9th National Computer Security Conference*, 1986.
15. A. Zakinthinos and E. S. Lee. A general theory of security properties. In *SP '97: Proceedings of the 1997 IEEE Symposium on Security and Privacy*, page 94, Washington, DC, USA, 1997. IEEE Computer Society.

Appendix

The following lemmas prove the undecidability of verifying BSPs for pushdown systems. The proofs for BSPs follows in the same fashion as the proof for BSP D. The same language L_M used in Section **??** can be used for the proofs of verification of the BSPs R, SR, BSD and SD.

Let us consider BSP I. Consider a new language L_M to be the prefix closure of $L_1 \cup L_2$ where

$$
\begin{aligned}
L_1 = \{enc(\#x_1\#x_2 \cdots x_n\#) \mid\ & x_1 \text{ is starting configuration}, \\
& x_n \text{ is accepting configuration}\} \\
L_2 = \{w \in \{v_1, v_2, c\}^* \mid\ & w\restriction_{\{v_1,v_2\}} = enc(\#x_1\#x_2 \cdots x_n\#), \\
& x_1 \text{ is starting configuration}, \\
& x_n \text{ is accepting configuration}, \\
& \text{exists } i : x_i \rightsquigarrow x_{i+1} \text{ invalid transition}\}
\end{aligned}
$$

Lemma 14. *L_M satisfies $I \Leftrightarrow L(M) = \emptyset$.*

Proof. ($\Leftarrow$:) Let us assume $L(M) = \emptyset$. Clearly, prefix closure of L_2 independently satisfies BSP I. Consider any string τ in L_1. τ is not a valid computation since $L(M) = \emptyset$. Note that τ interspersed with strings of c's is present in L_2. Also all the prefixes of τ interspersed with strings of c's is in L_M. Hence L_M satisfies I.

($\Rightarrow$:) If $L(M)$ is not empty then there exists a valid computation τ and it can be found in L_1. If we insert c's at arbitrary points in τ, the resulting string will not be present in L_M. Hence I is not satisfied. Hence, $L(M)$ is empty when L_M satisfies BSP I.

The language L_M can be easily seen to be a language accepted by a pushdown system. The same language L_M can be used for the proofs of BSPs BSI and SI.

Let us consider BSP IA^X, with $X \subseteq \Sigma$. Consider a new language L_M to be the prefix closure of $L_1 \cup L_2 \cup L_3$ where

$$
\begin{aligned}
L_1 = \{enc(\#x_1\#x_2 \cdots x_n\#) \mid\ & x_1 \text{ is starting configuration}, \\
& x_n \text{ is accepting configuration}\} \\
L_2 = \{w \in \{v_1, v_2, c\}^* \mid\ & w\restriction_{\{v_1,v_2\}} = enc(\#x_1\#x_2 \cdots x_n\#), \\
& x_1 \text{ is starting configuration}, \\
& x_n \text{ is accepting configuration}, \\
& \text{exists } i : x_i \rightsquigarrow x_{i+1} \text{invalid transition}\} \\
L_3 = X \cdot \{c\}^*
\end{aligned}
$$

Lemma 15. *L_M satisfies $IA^X \Leftrightarrow L(M) = \emptyset$.*

Proof. ($\Leftarrow$:) Let us assume $L(M) = \emptyset$. Clearly, prefix closure of $L_2 \cup L_3$ satisfies BSP IA^X independently. Consider any string of the form τ in L_1. τ is not a valid computation, since $L(M) = \emptyset$. Since $X \cdot \{c\}^*$ is a subset of L_M, we can insert c's at arbitrary points in τ and we will have the resulting string in L_2. Similarly for the prefixes of L_1, the corresponding strings which satisfies the condition will be present in L_2. Hence L_M satisfies IA^X.

($\Rightarrow$:) If $L(M)$ is not empty, there exists some string τ which is a valid computation. L_1 contains τ. Since we have $X \cdot \{c\}^*$ as a subset of L_M, we can insert c's at arbitrary points in τ. So, the resulting string will not be present in L_M. Hence IA^X is not satisfied. Hence $L(M)$ is empty, when L_M satisfies IA^X.

The same language can be taken for the proofs of BSPs $BSIA^X$ and SIA^X.

Let us consider BSP FCD. Given $V', N', C' \subseteq \Sigma$ and let $v_2 \in V'$ and $c \in C'$, consider the new language L_M to be the prefix closure of $L_1 \cup L_2$ where

$$L_1 = \{cv_2 \cdot enc(x_1 x_2 \cdots x_n) \mid x_1 \text{ is starting configuration}, x_n \text{ is accepting configuration}\}$$

$$L_2 = \{v_2 \cdot enc(x_1 x_2 \cdots x_n) \mid x_1 \text{ is starting configuration}, x_n \text{ is accepting configuration}, \text{exists } i : x_i \rightsquigarrow x_{i+1} \text{ invalid transition}\}$$

Lemma 16. *L_M satisfies $FCD \Leftrightarrow L(M) = \emptyset$.*

Proof. ($\Leftarrow$:) Let us assume $L(M) = \emptyset$. Consider any string containing confidential event in L_M. That string will be of the form $cv_2\tau$ in L_1. τ is not a valid computation since $L(M) = \emptyset$. Since $v_2 \in V'$ and $c \in C'$, the string after deleting c is included in L_2. The strings corresponding to prefixes of the strings in L_1 will be found in prefixes of strings in L_2. Hence L_M satisfies FCD.

($\Rightarrow$:) If $L(M)$ is not empty, there exists some string τ which is a valid computation. L_1 contains $cv_2\tau$. The resulting string after deleting the last c is not included in the language L_M. Hence FCD is not satisfied. Hence $L(M)$ is empty when L_M satisfies FCD.

Let us consider the BSP FCI. Given $V', N', C' \subseteq \Sigma$ and let $v_2 \in V'$ and $c \in C'$, consider the new language L_M to be the prefix closure of $L_1 \cup L_2$ where

$$L_1 = \{v_2 \cdot enc(x_1 x_2 \cdots x_n) \mid x_1 \text{ is starting configuration}, x_n \text{ is accepting configuration}\}$$

$$L_2 = \{c\}^* \cdot \{v_2 \cdot enc(x_1 x_2 \cdots x_n) \mid x_1 \text{ is starting configuration}, x_n \text{ is accepting configuration}, \text{exists } i : x_i \rightsquigarrow x_{i+1} \text{ invalid transition}\}$$

Lemma 17. *L_M satisfies $FCI \Leftrightarrow L(M) = \emptyset$.*

Proof. ($\Leftarrow$:) Let us assume $L(M) = \emptyset$. Clearly, prefix closure of L_2 independently satisfies *FCI*. Consider any string of the form $v_2\tau$ in L_1. τ is not a valid computation since $L(M) = \emptyset$. The string after inserting c preceding v_2 is included in L_2. For any prefix of the string in L_1, the corresponding string will be the prefix of a string in L_2. Hence L_M satisfies *FCI*.

($\Rightarrow$:) If $L(M)$ is not empty, there exists some string τ which is a valid computation. L_1 contains $v_2\tau$. The resulting string after inserting c preceding v_2 is not included in L_M.Hence *FCI* is not satisfied. Hence $L(M)$ is empty when L_M satisfies *FCI*.

Let us consider BSP $FCIA^X$, where $X \subseteq \Sigma$. Given $V', N', C' \subseteq \Sigma$ and let $v_2 \in V'$ and $c \in C'$, consider the new language L_M to be the prefix closure of $L_1 \cup L_2 \cup L_3$ where

$$L_1 = \{v_2 \cdot enc(x_1 x_2 \cdots x_n) \mid x_1 \text{ is starting configuration}, x_n \text{ is accepting configuration}\}$$

$$L_2 = \{c\}^* \cdot \{v_2 \cdot enc(x_1 x_2 \cdots x_n) \mid x_1 \text{ is starting configuration}, x_n \text{ is accepting configuration}, \text{exists } i : x_i \rightsquigarrow x_{i+1} \text{ invalid transition}\}$$

$$L_3 = X \cdot \{c\}^*$$

Lemma 18. *L_M satisfies $FCIA^X \Leftrightarrow L(M) = \emptyset$.*

Proof. ($\Leftarrow$:) Let us assume $L(M) = \emptyset$. Clearly, prefix closure of $L_2 \cup L_3$ independently satisfies $FCIA^X$. Consider any string of the form $v_2\tau$ in L_1. τ is not a valid computation since $L(M) = \emptyset$. Since $v_2 \in V'$ and $X \cdot \{c\}^*$ is contained in L_M, the string after inserting a c preceding v_2, is included in L_2. For any prefix of the string in L_1, the corresponding string which satisfies the condition is included in the prefix of a string from L_2. Hence L_M satisfies $FCIA^X$.

($\Rightarrow$:) If $L(M)$ is not empty, there exists some string τ which is a valid computation. L_1 contains $v_2\tau$. The resulting string after inserting a c preceding v_2 is not included in L_M. Hence $FCIA^X$ is not satisfied. Hence $L(M)$ is empty when L_M satisfies $FCIA^X$.