Model-Checking Trace-based Information Flow Properties *

Deepak D'Souza Raveendra Holla Raghavendra K. R.
Department of Computer Science and Automation,
Indian Institute of Science, Bangalore, India.
{deepakd, raveendra, raghavendrakr}@csa.iisc.ernet.in

Barbara Sprick†
TU Darmdstadt,
Fachbereich Informatik, Germany.
sprick@mais.informatik.tu-darmstadt.de

Abstract

In this paper we consider the problem of verifying trace-based information flow properties for different classes of system models. We begin by proposing an automata-theoretic technique for model-checking trace-based information flow properties for finite-state systems. We do this by showing that Mantel's Basic Security Predicates (BSPs), which were shown to be the building blocks of most trace-based properties in the literature, can be verified in an automated way for finite-state system models. We also consider the problem for the class of pushdown system models, and show that it is undecidable to check such systems for any of the trace-based information flow properties. Finally we consider a simple trace-based property we call "weak non-inference" and show that it is undecidable even for finite-state systems.

1 Introduction

Information flow properties are a way of specifying security properties of systems, dating back to the work of Goguen and Meseguer [12] in the eighties. In this framework, a system is modelled as having high-level (or confidential) events as well as low-level (or public) events, and a typical property requires that the high-level events should not "influence" the outcome of low-level events. In other words, the sequence of low-level events observed from a system execution, should not reveal "too much" information about the high-level events that may have taken place.

There is a great variety of information flow properties proposed in the literature, and they can be broadly classified into three different categories. The

*Preliminary results in this paper appeared in [9] and [7]

†Work partially done while visiting Indian Institute of Science, Bangalore.

original formulation of "non-interference" by Goguen and Meseguer was *state-based* in the sense that it spoke about the state of the system after a sequence of events: The state reached by the system after executing a sequence of low and high-level events, must be the same (from the low-level observer's point of view) as the state reached after executing only the low-level events in the sequence. Some information flow properties are *trace-based* in that they specify a property of the set of traces or executions produced by the system. For example, the "non-inference" property [20, 19, 25] states that for every trace produced by the system, its projection to low-level events must also be a possible trace of the system. Finally, properties could be based on *structural* properties of the system model. For example, the property "Bisimulation-based Strong Nondeterministic Non-interference" (BSNNI) of Focardi and Gorrieri [10] states that the system model with high-level events made "silent", should be bisimilar to the system model with high-level transitions deleted. Many of these three kinds of properties also appear in a "language-based" setting, in that they are phrased over programs directly. For example, Denning and Denning [6] say a program satisfies non-interference if the program started in two states that are low-level equivalent (i.e. they agree on the values of low-level variables), always terminates in states that are low-level equivalent.

In [16, 17] Mantel studies the trace-based properties in the literature, and provides a framework for reasoning about them 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. Some of the well-known properties covered in this framework include the *non-inference* property mentioned above, *separability* [19] (which requires that every possible low-level behavior interleaved with every possible high-level behaviour must be a possible behaviour of the system), *generalized non-interference* [18] (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), *non-deducibility* [23], *restrictiveness* [18], *forward correctability* [14], and the *perfect security property* [25]. Mantel's 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. Focardi and Gorrieri [10] also study some trace-based properties and classify their relative expressiveness.

Our focus in this paper is concerned with the problem of verifying trace-based information flow properties for a given class of system models. Much of the work in the literature has focussed on using "unwinding" conditions to solve the problem [12, 17]. Unwinding conditions are simulation relations on the states of a system model which satisfy certain additional constraints. In [17] Mantel gives unwinding conditions for most of the BSPs he identifies. These unwinding conditions are shown to be sound in general, and complete for a restricted class of system models in which all events are classified as either high or low. For general systems however these unwinding conditions are not complete [17, Section 5.4.5]: a system could satisfy the information flow property but *fail* the corresponding unwinding condition. Thus, while these unwinding conditions can be verified algorithmically (at least for the class of finite-state system models, see [8]), they do not lead to a verification (or "model-checking") algorithm for general systems.

In this paper, we use the framework of Mantel as a starting point, and present

a model-checking algorithm for all trace-based information flow properties in his taxonomy, for systems modeled as finite-state transition systems. We define a set of (formal) language-theoretic operations that represent the allowed perturbations and corrections for each BSP, and show that the question of whether a set of traces L satisfies a BSP P boils down to checking whether a language $op_1(L)$ is contained in a language $op_2(L)$, where op_1 and op_2 are particular language-theoretic operations. Thus these characterisations can be viewed as analogues of Focardi and Gorrieri's trace-based static characterisations [10]. Finally we show that the language-theoretic operations are *regularity-preserving*. Thus if L is specified by a finite-state transition system (and is hence regular), then $op_1(L)$ and $op_2(L)$ are also regular, and the question of whether $op_1(L) \subseteq op_2(L)$ can be answered effectively by standard automata-theoretic techniques. This gives us a model-checking algorithm for verifying BSPs for finite-state systems. Using Mantel's characterisations, we immediately obtain model-checking algorithms for all the information-flow properties in his taxonomy.

A natural question to ask is whether this technique also extends to the class of pushdown systems, which are one of the most tractable classes of infinite-state systems. For example reachability of a given configuration is decidable for this class of systems [3]. Pushdown systems can make use of an unbounded stack, and are useful in modelling programs with procedure calls and variables with bounded values. Such systems also occur in the literature as recursive state machines [1] and boolean programs [2]. Unfortunately, we show that the problem of model-checking such systems for *any* of the BSPs is undecidable, and hence we cannot hope to find an algorithmic solution to the problem. We go on to show that this undecidability result also applies to the trace-based properties in Mantel's taxonomy.

Finally, in the third part of the paper we consider a natural trace-based 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 same spirit as the BSPs and the other trace-based non-interference properties, as it also says 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 the trace-based properties mentioned above. 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*, a natural non-interference 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 [21] a general first order (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.

Turning now to related work, our decision procedure for the verification problem for finite-state systems is closest to that of Focardi and Gorrieri [10].

They have also implemented their decision procedure for systems modeled in process algebra in a tool called CoSec [11]. In contrast, our decision procedures are applicable to a much larger class of trace-based properties. In principle, our "static characterisations" can be used to extend the CoSec tool to handle all trace-based properties in Mantel's taxonomy.

In work subsequent to ours van der Meyden and Zhang [24] give model-checking algorithms for the trace-based properties: non-deducibility, generalized non-interference and forward correctability, apart from state-based non-interference properties, and the bisimulation-based version of the restrictiveness property. They also show some hardness results for these algorithms, in particular PSPACE-completeness for the three trace-based properties above. Their decision procedures are similar to that of Focardi-Gorrieri and ours. Though we do not address the issue of completeness, our decision procedures are easily seen to be in PSPACE.

In [5] Dam gives a decision procedure for checking the language-based bisimulation property called *strong low bisimulation* as defined in [22], for simple parallel while programs. He also shows the undecidability of the bisimulation-based non-interference property studied in [4], for this class of programs. The class of programs considered are infinite-state systems, with finite control but variables with unbounded values. Hence they contain strictly the class of finite-state systems as well as the class of pushdown systems we consider. In contrast our undecidability result for WNI is for a simple trace-based property and for *finite-state* systems.

The rest of the paper is organized as follows. Section 2 defines the technical terms and recalls Mantel's BSPs and trace-based definitions of well known non-interference properties. Section 3 defines the language-theoretic operations, and characterizes BSPs in terms of these operations. Section 4 shows that these operations are regularity-preserving, leading to a model checking algorithm for BSPs for finite-state systems. In Section 5 we prove the undecidability of the model-checking for BSPs for pushdown systems. Section 6 shows the undecidability of the model-checking problem for the trace-based properties in Mantel's taxonomy, for pushdown systems. In Section 7 we introduce the Weak Non-Inference property, show its undecidability for finite-state systems. Finally Section 8 concludes the article with some discussion and future work.

2 Preliminaries

In this section we recall Mantel's Basic Security Predicates and trace-based definitions of information flow properties. As we are interested in addressing the verification problem in this paper, we only recall the definitions of these properties and point the reader to [17] and [10] for motivation and example applications.

We begin with some preliminary notions. 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 and β. A *language* L over Σ is just a subset of Σ^*.

A *marked* language M over an alphabet Σ is a language over the alphabet $\Sigma \cup \{\natural\}$, where '$\natural$' is a special "mark" symbol different from those in Σ, and each

string in M contains exactly one occurrence of $\natural$. A marked language is thus a subset of $\Sigma^*\natural\Sigma^*$.

A *view* w.r.t. an alphabet Σ is a partition $\mathcal{V} = (V, N, C)$ of Σ into visible events V, confidential events C, and neither visible nor confidential events N, from a particular user's point of view.

For the rest of the paper we fix an alphabet of events Σ. Let w be a string over Σ, and let $X \subseteq \Sigma$. Then $w \upharpoonright_X$ denotes the string obtained by deleting all events from w that are not elements of X. 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_1 and L_2 over Σ or $\Sigma \cup \{\#\}$, we say $L_1 \subseteq_Y L_2$ iff $L_1 \upharpoonright_{\overline{Y}} \subseteq L_2 \upharpoonright_{\overline{Y}}$.

We begin by recalling the *basic security predicates* (BSPs) of Mantel [17]. Let L be a language over Σ. Let $\mathcal{V} = (V, N, C)$ be a view of Σ. The definitions below say when L satisfies a particular BSP w.r.t. the view $\mathcal{V}$:

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 $V' \subseteq V$, $C' \subseteq C$, 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. V', C', 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 $\tau \restriction_{\overline{C}} \in L$.

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 also recall the definitions of some of the trace-based properties in Mantel's taxonomy [17]. These properties are defined w.r.t. a partition of Σ into high (H) and low (L) events, and another partition of Σ into input (I) and output (O) events. We write HI for $\mathsf{H} \cap \mathsf{I}$ and LI for $\mathsf{L} \cap \mathsf{I}$. We denote by $\mathcal{H}$ the view $(\mathsf{L}, \emptyset, \mathsf{H})$, and by $\mathcal{HI}$ the view $(\mathsf{L}, \mathsf{H} \setminus \mathsf{HI}, \mathsf{HI})$ induced by such a partition. In the logical formulas below, the quantification is assumed to be over the strings in Σ^* unless specified otherwise.

Non-inference (NF) [20, 19, 25] property states that for every trace produced by the system, its projection to low 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. Formally, L satisfies NF if

$$\forall \tau \in L \ (\tau \restriction_{\mathsf{L}} \ \in L).$$

This property is expressed as the BSP R w.r.t. the view $\mathcal{H}$.

Separability (SEP) [19] requires that every possible low-level behavior interleaved with every possible high-level behaviour must be a possible behaviour of a system. Formally, L satisfies SEP if

$$\forall \tau_1, \tau_2 \in L \ (\mathit{interleaving}(\tau_1, \tau_2) \subseteq L),$$

where $\mathit{interleaving}(\tau_1, \tau_2) = \{\tau \mid \tau \restriction_{\mathsf{H}} = \tau_1 \restriction_{\mathsf{H}} \wedge \tau \restriction_{\mathsf{L}} = \tau_2 \restriction_{\mathsf{L}}\}$.

This property is expressed as the conjunction of the BSPs *BSIA* w.r.t. the set H, and *BSD*, both w.r.t. the view $\mathcal{H}$.

Generalized Non-Interference (GNI) [18] 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. Formally, L satisfies GNI if

$$\forall \tau_1, \tau_2, \tau_3 \ ((\tau_1\tau_2 \in L \wedge \tau_2 \restriction_{\Sigma \setminus \mathsf{HI}} = \tau_3 \restriction_{\Sigma \setminus \mathsf{HI}}) \Rightarrow \ \exists \tau_4 \ (\tau_1\tau_4 \in L \wedge \tau_4 \restriction_{\mathsf{L} \cup \mathsf{HI}} = \tau_3 \restriction_{\mathsf{L} \cup \mathsf{HI}})).$$

This property is expressed as the conjunction of BSPs *BSD* and *BSI*, w.r.t. the view $\mathcal{HI}$.

Forward Correctability (FC) [14] is a weaker version of an earlier information flow property called *Restrictiveness* retaining its compositionality property. Formally, L satisfies FC if

$$\begin{array}{l}\forall \tau_1\tau_2\ \forall c \in \mathsf{HI}\ \forall v' \in \mathsf{LI} \\ ((\tau_1 v'\tau_2 \in L\ \wedge\ \tau_2 \restriction_{\mathsf{HI}} = \epsilon)\ \Rightarrow\ \exists \tau_3\ (\tau_1 c v' \tau_3 \in L \wedge\ \tau_3 \restriction_{\mathsf{L}} = \tau_2 \restriction_{\mathsf{L}} \wedge \tau_3 \restriction_{\mathsf{HI}} = \epsilon) \wedge \\ (\tau_1 c v'\tau_2 \in L\ \wedge\ \tau_2 \restriction_{\mathsf{HI}} = \epsilon)\ \Rightarrow\ \exists \tau_3\ (\tau_1 v' \tau_3 \in L \wedge\ \tau_3 \restriction_{\mathsf{L}} = \tau_2 \restriction_{\mathsf{L}} \wedge \tau_3 \restriction_{\mathsf{HI}} = \epsilon)).\end{array}$$

This property is expressed as the conjunction of the BSPs *BSD*, *BSI*, and *FCD* w.r.t. the sets $\mathsf{LI}, \emptyset, \mathsf{HI}$, and *FCI* w.r.t. the sets $\mathsf{LI}, \emptyset, \mathsf{HI}$; all w.r.t. the view $\mathcal{HI}$.

Non-deducibility for Outputs (NDO) [13] Non-deducibility for Outputs permits non-critical information flow from the low level to the high level to some extent. More specifically, the high-level behavior may depend on input that is provided by low-level users. The possible user inputs are modelled by a set $\mathsf{UI} \subseteq \mathsf{I}$. Formally, L satisfies NDO if

$$\forall \tau_1, \tau_2 \in L\ \forall \tau_3\ ((\tau_3 \restriction_{\mathsf{L}} = \tau_1 \restriction_{\mathsf{L}} \wedge\ \tau_3 \restriction_{\mathsf{H} \cup (\mathsf{L} \cap \mathsf{UI})} = \tau_2 \restriction_{\mathsf{H} \cup (\mathsf{L} \cap \mathsf{UI})}) \Rightarrow \tau_3 \in L).$$

This property is expressed as the conjunction of the BSPs *BSIA* w.r.t. the set UI and *BSD*, w.r.t. the view $\mathcal{H}$.

3 Expressing BSPs Language-Theoretically

In this section we introduce a number of language-theoretic operations that will be used to characterize Mantel's Basic Security Predicates.

Let L be a language over Σ and M a marked language over Σ. Let $\mathcal{V} = (V, N, C)$ be a view of Σ. We now describe the language-theoretic operations on L w.r.t. the view $\mathcal{V}$.

1. Let $X \subseteq \Sigma$. The *projection* of the language L to X, written $L \restriction_X$, is defined to be $\{\tau \restriction_X \mid \tau \in L\}$.

2. *l-del*$(L) = \{\alpha\beta \mid c \in C,\ \beta \restriction_C = \epsilon,\ \text{and } \alpha c \beta \in L\}$. Thus the operation *l-del* corresponds to the deletion of the last confidential event in a string. More precisely, *l-del*(L) contains all strings than can be obtained from a string in L by deleting the last C-event.

3. *l-ins*$(L) = \{\alpha c \beta \mid c \in C,\ \beta \restriction_C = \epsilon,\ \text{and } \alpha\beta \in L\}$. Thus *l-ins* corresponds to the insertion of a confidential event in strings of L, only at a position after which no confidential events occur.

4. Let $X \subseteq \Sigma$. Then *l-ins-adm*$^X(L)$ is defined to be the set

$$\{\alpha c\beta \mid c \in C,\ \alpha\beta \in L,\ \beta \restriction_C = \epsilon,\ \text{and } \exists \gamma c \in L : \gamma \restriction_X = \alpha \restriction_X\}.$$

Operation *l-ins-adm* (w.r.t. X) corresponds to insertion of "X-admissible" confidential events in strings of L. The insertion of a C-event is *X-admissible* after a prefix α in a string τ iff there exists another string $\gamma c \in L$ with γ projected to X being the same as α projected to X.

5. *l-del-mark*$(L) = \{\alpha \natural \beta \mid c \in C,\ \alpha c\beta \in L,\ \text{and } \beta \restriction_C = \epsilon\}$. The operation *l-del-mark* corresponds to the "marked" deletion of the last confidential event. More precisely, this operation replaces the last C-event in every string of L by the special mark symbol $\natural$.

6. *l-ins-mark*$(L) = \{\alpha c \natural \beta \mid c \in C,\ \alpha\beta \in L,\ \beta \restriction_C = \epsilon\}$. The operation *l-ins-mark* corresponds to the "marked" insertion of a confidential event. It is similar to the operation *l-ins*, but additionally introduces a mark $\natural$ after the newly inserted C-event.

7. Let $X \subseteq \Sigma$. Then we define *l-ins-adm-mark*$^X(L)$ to be the set

$$\{\alpha c \natural \beta \mid c \in C,\ \alpha\beta \in L,\ \beta \restriction_C = \epsilon,\ \text{and } \exists \gamma c \in L : \gamma \restriction_X = \alpha \restriction_X\}.$$

Operation *l-ins-adm-mark* (w.r.t. X) corresponds to the marked insertion of X-admissible events. This operation is similar to *l-ins-adm*, but a mark $\natural$ is introduced after the newly inserted X-admissible event.

8. *mark*$(L) = \{\alpha \natural \beta \mid \alpha\beta \in L\}$. Thus operation *mark* corresponds to the insertion of a mark at an arbitrary position in strings of L.

9. Let $X \subseteq \Sigma$. Then the marked projection of the marked language M to X, written $M \restriction^m_X$, is defined as

$$M \restriction^m_X = \{\alpha \natural \beta' \mid \alpha \natural \beta \in M \text{ and } \beta' = \beta \restriction_X\}.$$

Thus marked projection operates on a marked language M and is similar to projection, but leaves every string intact upto the mark and projects to set X the suffix after the mark.

10. Let $C' \subseteq C$ and $V' \subseteq V$. Then *l-del-con-mark*$_{C',V'}(L)$ is defined to be the set

$$\{\alpha v \natural \beta \,|\, c \in C',\ v \in V',\ \alpha c v \beta \in L \text{ and } \beta \restriction_C = \epsilon,\}.$$

Thus the operation *l-del-con-mark* (w.r.t. C' and V') corresponds to marked deletion in the "context" of an event in V'. This operation deletes the last confidential event c in a string and inserts a mark, provided c belongs to C' and and is immediately followed by a V'-event in the string. For convenience we place the mark *after* the V'-event.

11. Let $C' \subseteq C$ and $V' \subseteq V$. Then *l-ins-con-mark*$_{C',V'}(L)$ is defined to be the set

$$\{\alpha c v \natural \beta \mid \ c \in C',\ v \in V',\ \alpha v \beta \in L, \ \text{ and } \beta \restriction_C = \epsilon\}.$$

Operation *l-ins-con-mark* (w.r.t. C' and V') thus corresponds to marked insertion of C'-events in the "context" of a V' event, in the sense above.

12. Let $X \subseteq \Sigma$, $C' \subseteq C$ and $V' \subseteq V$. Then *l-ins-adm-con-mark*$^X_{C',V'}(L)$ is defined to be the set

$$\{\alpha c v \natural \beta \mid c \in C',\ v \in V',\ \alpha v \beta \in L,\ \beta \restriction_C = \epsilon,\ \text{and } \exists \gamma c \in L,\ \gamma \restriction_X = \alpha \restriction_X\}.$$

The operation *l-ins-adm-con-mark* (w.r.t. X, C' and V') corresponds to the marked insertion of X-admissible C'-events in the context of a V' event. It is similar to *l-ins-con-mark* but allows only the insertion of X-admissible C'-events.

13. Let $N' \subseteq N$ and $V' \subseteq V$. Then *erase-con-mark*$_{N',V'}(L)$ is defined to be the set

$$\{\alpha v \natural \beta \mid v \in V', \text{ and } \exists \delta \in (N')^* : \alpha \delta v \beta \in L\}.$$

The operation *erase-con-mark* (w.r.t. N' and V') corresponds to the marked erasure of N'-events in the context of V'-events. More precisely, *erase-con-mark*$_{N',V'}(L)$ contains all strings obtained from a string in L by the erasure of a consecutive sequence of zero or more N' events which end before a V' event v. The mark symbol is inserted *after* the event v in the string.

We can now express the BSPs in terms of the language-theoretic operations just defined.

Lemma 1 *The BSPs of Mantel can be characterized in terms of language theoretic operations as follows. Let L be a language over Σ and $\mathcal{V} = (V, N, C)$ be a view over Σ. In the following statements we assume the properties are w.r.t. the view $\mathcal{V}$. Then:*

1. *L satisfies R iff $L \restriction_V \subseteq_N L$.*
2. *L satisfies D iff l-del$(L) \subseteq_N L$.*
3. *L satisfies I iff l-ins$(L) \subseteq_N L$.*
4. *L satisfies IA w.r.t. X iff l-ins-adm$^X(L) \subseteq_N L$.*
5. *L satisfies BSD iff l-del-mark$(L) \restriction^m_N \subseteq$ mark$(L) \restriction^m_N$.*
6. *L satisfies BSI iff l-ins-mark$(L) \restriction^m_N \subseteq$ mark$(L) \restriction^m_N$.*
7. *L satisfies BSIA w.r.t. X iff l-ins-adm-mark$^X(L) \restriction^m_N \subseteq$ mark$(L) \restriction^m_N$.*
8. *L satisfies FCD w.r.t. V', C', N' iff*

$$\textit{l-del-con-mark}_{C',V'}(L) \restriction^m_N \subseteq \textit{erase-con-mark}_{N',V'}(L) \restriction^m_N$$

9. *L satisfies FCI w.r.t.* V', C', N' *iff*

$$\textit{l-ins-con-mark}_{C',V'}(L) \restriction^m_N \subseteq \textit{erase-con-mark}_{N',V'}(L) \restriction^m_N$$

10. *L satisfies FCIA w.r.t.* X, V', C', N' *iff*

$$\textit{l-ins-adm-con-mark}^X_{C',V'}(L) \restriction^m_N \subseteq \textit{erase-con-mark}_{N',V'}(L) \restriction^m_N .$$

11. *L satisfies SR iff* $L \restriction_{\overline{C}} \subseteq L$.

12. *L satisfies SD iff* $\textit{l-del}(L) \subseteq L$.

13. *L satisfies SI iff* $\textit{l-ins}(L) \subseteq L$.

14. *L satisfies SIA iff* $\textit{l-ins-adm}^X(L) \subseteq L$.

Proof

1. Suppose L satisfies R. Let $\tau \in L \restriction_V$. Then there exists some τ' in L such that $\tau' \restriction_V = \tau$. Since L satisfies R and $\tau' \in L$, there exists τ'' in L such that $\tau'' \restriction_C = \epsilon$ and $\tau'' \restriction_V = \tau' \restriction_V = \tau \restriction_V$. Since both τ'' and τ don't contain any C-events, they differ from each other only on N-events (i.e. $\tau =_N \tau''$). Thus τ belongs to L modulo a correction on N-events. Hence $L \restriction_V \subseteq_N L$.

   Suppose $L \restriction_V \subseteq_N L$. Let $\tau \in L$. Since $L \restriction_V \subseteq_N L$, there exists a string τ' in L, such that $\tau' =_N \tau \restriction_V$. Since $\tau \restriction_V$ has no C-events, $\tau' \restriction_C = \epsilon$ and $\tau' \restriction_V = (\tau \restriction_V) \restriction_V = \tau \restriction_V$. Hence R is satisfied.

2. Assume L satisfies D and consider a string τ in $\textit{l-del}(L)$. Then τ must be of the form $\alpha\beta$ and there must exist $\alpha c \beta \in L$ such that $c \in C$, and $\beta \restriction_C = \epsilon$. Since L satisfies D, we must have $\alpha'\beta' \in L$ such that $\alpha' =_N \alpha$ and $\beta' =_N \beta$. But this just means that $\tau = \alpha\beta =_N \alpha'\beta'$. Hence $\textit{l-del}(L) \subseteq_N L$.

   Suppose $\textit{l-del}(L) \subseteq_N L$. Consider a string $\alpha c \beta$ in L with $c \in C$ and $\beta \restriction_C = \epsilon$. By the definition of $\textit{l-del}(L)$, we have $\alpha\beta \in \textit{l-del}(L)$. Since $\textit{l-del}(L) \subseteq_N L$, there exists $\tau \in L$ such that $\alpha\beta =_N \tau$. Thus τ' can be expressed as $\alpha'\beta'$ with $\alpha =_N \alpha'$ and $\beta =_N \beta'$. Hence D is satisfied.

3. Assume L satisfies I and consider a string τ in $\textit{l-ins}(L)$. Then τ will be of the form $\alpha c \beta$ for some $c \in C$ and there must exist $\alpha\beta \in L$ such that $\beta \restriction_C = \epsilon$. Since L satisfies I, there exists $\alpha' c \beta' \in L$ such that $\alpha' =_N \alpha$ and $\beta' =_N \beta$. But this just means that $\tau = \alpha c \beta =_N \alpha' c \beta'$. Hence $\textit{l-ins}(L) \subseteq_N L$.

   Suppose $\textit{l-ins}(L) \subseteq_N L$. Consider a string $\alpha\beta \in L$ with $\beta \restriction_C = \epsilon$. By the definition of $\textit{l-ins}(L)$, for any $c \in C$, we have $\alpha c \beta \in \textit{l-ins}(L)$. Since $\textit{l-ins}(L) \subseteq_N L$, there exists $\tau \in L$ such that $\alpha c \beta =_N \tau$. Thus τ can be expressed as $\alpha' c \beta'$ where $\alpha' =_N \alpha$ and $\beta =_N \beta$. Hence I is satisfied.

The remaining proofs are similar and can be found in Appendix A. □

4 Model-Checking BSPs for Finite-State Systems

We now show how the language-theoretic characterisations of BSPs lead to a decision procedure for checking whether a finite-state system satisfies a given BSP. We first introduce the necessary terminology, beginning with the required notions in finite-state automata.

A *(finite-state) transition system* over an alphabet Δ is a structure of the form $\mathcal{T} = (Q, s, \longrightarrow)$, where Q is a finite set of states, $s \in Q$ is the start state, and $\longrightarrow \subseteq Q \times \Delta \times Q$ is the transition relation. We write $p \xrightarrow{a} q$ to stand for $(p, a, q) \in \longrightarrow$, and use $p \xrightarrow{\alpha}{}^* q$ to denote the fact that we have a path labelled α from p to q in the underlying graph of the transition system $\mathcal{T}$. The language *accepted* (or *generated*) by the transition system $\mathcal{T}$ is defined to be $L(\mathcal{T}) = \{\alpha \in \Delta^* \mid s \xrightarrow{\alpha}{}^* q \text{ for some } q \in Q\}$.

A *(finite-state) automaton* (FSA) over an alphabet Δ is of the form $\mathcal{A} = (Q, s, \longrightarrow, F)$ where $(Q, s, \longrightarrow)$ forms a transition system and $F \subseteq Q$ is a set of final states. The language accepted by $\mathcal{A}$ is defined to be $L(\mathcal{A}) = \{\alpha \in \Delta^* \mid s \xrightarrow{\alpha}{}^* q \text{ for some } q \in F\}$. A transition system can thus be thought of as an automaton in which all states are final.

It will be convenient to make use of automata with *ϵ-transitions*. Thus the automaton is also allowed transitions of the form $p \xrightarrow{\epsilon} q$. The language accepted by automata with ϵ-transitions is defined similarly, except that the ϵ labels don't contribute to the label of a path. As is well-known ϵ-transitions don't add to the expressive power of automata.

The class of languages accepted by FSA's is termed the class of *regular* languages. Regular languages are effectively closed under intersection and complementation. Moreover their language emptiness problem – i.e. given an FSA $\mathcal{A}$, is $L(\mathcal{A}) = \emptyset$? – is efficiently decidable (by simply checking if there is a final state reachable from the initial state). It follows that the language inclusion problem (whether $L(\mathcal{A}) \subseteq L(\mathcal{B})$?) is also decidable for automata, since we can check equivalently that $L(\mathcal{A}) \cap (\Delta^* - L(\mathcal{B})) = \emptyset$.

Returning now to our problem of verifying BSPs, we say that a system modelled as a finite-state transition system $\mathcal{T}$ satisfies a given BSP P iff $L(\mathcal{T})$ satisfies P. In the previous section we showed that the question of whether a language L satisfies P boils down to checking whether $L_1 \subseteq L_2$, where L_1 and L_2 are obtained from L by successive applications of some language-theoretic operations. If L is a regular language to begin with, and if each language-theoretic operation *op* of Section 2 is *regularity preserving* (in the sense that if M is a regular language, then so is $op(M)$), then L_1 and L_2 are also regular languages and the question $L_1 \subseteq L_2$ can be effectively answered. To obtain a decision procedure for our BSP verification problem, it is thus sufficient to show that the language-theoretic operations are regularity preserving. In the rest of this section we concentrate on showing this.

The language operations of Section 3 are of the following kinds: they either take a language over Σ and return a language over Σ, or they take a language over Σ and return a marked language over Σ, or they take a marked language over Σ and return a marked language over Σ. In all cases we show that if they take a regular language, they return a regular language. Once again we assume the operations below are w.r.t. a view $\mathcal{V} = (V, N, C)$ of Σ.

1. *Projection*: Let L be a language over Σ accepted by an FSA $\mathcal{A}$, and let $X \subseteq \Sigma$. Then we can construct $\mathcal{A}'$ accepting $L \restriction_X$ by simply replacing transitions in $\mathcal{A}$ of the form $p \xrightarrow{a} q$, with $a \notin X$, by ϵ-transitions $p \xrightarrow{\epsilon} q$.

2. *l-del*: Let L be a language over Σ, with $L = L(\mathcal{A})$. We construct $\mathcal{A}'$ for *l-del*(L) as follows. We create two copies of $\mathcal{A}$. The initial state of $\mathcal{A}'$ is the initial state of the first copy. In the second copy all transitions of the form $p \xrightarrow{c} q$ with $c \in C$ are deleted. In the first copy we add an ϵ-transition from a state p in the first copy to state q in the second copy if $p \xrightarrow{c} q$ in $\mathcal{A}$, with $c \in C$. The final states in the first copy are marked non-final and the final states in the second copy are retained as final.

   This construction can be described formally as follows. Let $\mathcal{A} = (Q, s, \longrightarrow, F)$. Define $\mathcal{A}' = (Q', s', \longrightarrow', F')$ where $Q' = Q \times \{1, 2\}$, $s' = (s, 1)$, $\longrightarrow'$ is given by

   $$\begin{array}{lll} (p,1) \xrightarrow{a}{}' (q,1) & \text{iff} & p \xrightarrow{a} q \text{ in } \mathcal{A} \\ (p,1) \xrightarrow{\epsilon}{}' (q,2) & \text{iff} & p \xrightarrow{c} q \text{ in } \mathcal{A} \text{ with } c \in C \\ (p,2) \xrightarrow{a}{}' (q,2) & \text{iff} & p \xrightarrow{a} q \text{ and } a \notin C, \end{array}$$

   and $F' = F \times \{2\}$.

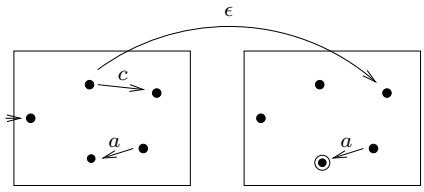

Figure 1: *l-del*(L)

   The construction is depicted in Fig. 1. The first and the second boxes represent the first and the second copy of the given automaton $\mathcal{A}$. The thick dots represent the states, circled dots represent the final states, and arcs represent the transitions.

3. *l-ins*: Let L be a language over Σ with $L = L(\mathcal{A})$. We construct $\mathcal{A}'$ for *l-ins*(L) as follows. We make two copies of $\mathcal{A}$. The start state of $\mathcal{A}'$ is the start state of the first copy, and the final states are the final states of the second copy. In the first copy for every state p we add a c transition (for every $c \in C$) from p in the first copy to p in the second copy. The c-transitions for $c \in C$ are deleted from the second copy. The construction is depicted in Fig 2.

4. *l-ins-adm*: Let L be a language over Σ with $L = L(\mathcal{A})$, and let $X \subseteq \Sigma$. We construct $\mathcal{A}'$ for *l-ins-adm*$^X(L)$ as follows. We have two copies of $\mathcal{A}$. In the first copy, the states have two components: the first component keeps track of a state from $\mathcal{A}$, while the second keeps track of a *set* of states of $\mathcal{A}$ that are reachable by words that are X-equivalent to the current word being read. We have a transition labelled c, with $c \in C$, from a state (p, T) in the first copy to p in the second copy, provided T contains a state t from which it is possible to do a c and reach a final state. Once in the

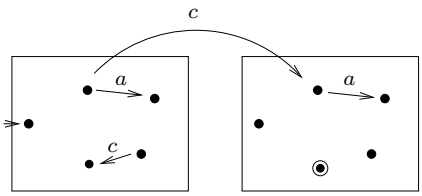

Figure 2: *l-ins*(L)

second copy, we allow only non-C transitions and retain the original final states. The construction is depicted in Figure 3.

More formally, we can define $\mathcal{A}'$ as follows. Let $\mathcal{A} = (Q, s, \longrightarrow, F)$ and let $\mathcal{B}$ be the automaton obtained from $\mathcal{A}$ by replacing transitions of the form $p \xrightarrow{a} q$ by $p \xrightarrow{\epsilon} q$ whenever $a \notin X$. Then $\mathcal{A}' = (Q', s', \longrightarrow', F')$ where $Q' = (Q \times 2^Q) \cup Q$; $s' = (s, S)$ where $S = \{q \in Q \mid s \xrightarrow{\epsilon}{}^* q \text{ in } \mathcal{B}\}$; $\longrightarrow'$ is given below:

$$
\begin{array}{lll}
(p,T) \xrightarrow{a}{}' (q,T) & \text{if} & p \xrightarrow{a} q \text{ and } a \notin X \\
(p,T) \xrightarrow{a}{}' (q,U) & \text{if} & p \xrightarrow{a} q, a \in X, \text{ and} \\
 & & U = \{r \mid \exists t \in T, t \xrightarrow{a}{}^* r \text{ in } \mathcal{B}\} \\
(p,T) \xrightarrow{c}{}' p & \text{if} & \exists t \in T, q \in F : t \xrightarrow{c} q \text{ and } c \in C; \\
p \xrightarrow{a}{}' q & \text{if} & p \xrightarrow{a} q \text{ and } a \notin C.
\end{array}
$$

and $F' = F$.

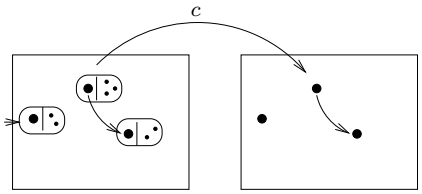

Figure 3: *l-ins-adm*$^X(L)$

The constructions for the rest of the language-theoretic operations are given in the appendix B.

Using the observations made in this section, we conclude that:

Theorem 1 *The problem of model checking BSPs for finite-state systems is decidable.* □

The running time of our procedure can be seen to be exponential in the number of states of the given finite-state transition system, in the worst case. This is because the automata constructions for the language-theoretic operations involve a blow-up in states of $O(n)$ in most cases, and $2^{O(n)}$ in the case of the BSPs based on the admissibility clause (here n is the number of states in the given transition system). Furthermore, no operation used on the right hand

side of the containment (recall that our characterisations are typically of the form $op_1(L) \subseteq op_2(L)$) introduces an exponential blow-up. Thus in checking containment, we have to complement an automaton of size at most $O(n)$, and hence we have a bound of $2^{O(n)}$ in the worst case. The algorithms can also be seen to run in PSPACE. This is because one can do reachability in a graph in log-space (in the size of the graph). It follows that our emptiness checks can be done in $O(n)$ space, where n is the size of the given system model.

As a simple consequence of Theorem 1 and Mantel's characterization, we can see that the problem of model-checking the trace-based non-interference properties of Section 2 is decidable for finite-state systems.

5 Model-Checking BSPs for Pushdown Systems

In this section we show that if the system is modelled as a pushdown system, the problem of model-checking BSPs becomes undecidable.

The notion of a pushdown system we consider is that of a pushdown automaton with all accepting states. Thus a *pushdown system (PDS)* is of the form $P = (Q, \Sigma_P, \Gamma_P, \Delta, s, \bot)$, where Q is a finite set of control states, Σ_P is a finite input alphabet, Γ_P is a finite stack alphabet, $\Delta \subseteq ((Q \times (\Sigma_P \cup \{\epsilon\}) \times \Gamma_P) \times (Q \times \Gamma_P^*))$ is the transition relation, $s \in Q$ is a starting state, and $\bot \in \Gamma_P$ is an initial stack symbol. 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 top of the stack 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 to be read 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 forms in literature, like Recursive State Machines [1] and Boolean Programs [2]. Pushdown systems thus capture an interesting class of system models.

The class of languages accepted by pushdown systems is closely related to context free languages:

Lemma 2 *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, in which all control states are final states. Hence pushdown systems generate only context-free languages. It is easy to see that if a PDS accepts a string w, then it also accepts all the prefixes of w. Hence PDSs accept only prefix-closed context-free languages.

Conversely, let L be a prefix-closed context-free language. Then there exists a context free grammar (CFG) $G = (N, \Sigma, R, S)$ generating L, where N is a set of non-terminals, R is a set of productions, and $S \in N$ is the start symbol. Without loss of generality, we assume that there are no useless non-terminals (i.e. those that do not generate a terminal string) and all productions of G are of the form $A \to cB_1B_2 \cdots B_k$ where $c \in \Sigma \cup \{\epsilon\}$, $B_1, B_2 \cdots B_k \in N$ and $k \geq 0$. Using a standard construction (see for example [15]) we can construct a language-equivalent non-deterministic pushdown system $P = (\{q\}, \Sigma, N, \delta, q, S)$, where q is the only state, Σ is the input alphabet, N is the stack alphabet, q is the starting state, S is the initial stack symbol, and δ is defined as: for each production $A \to cB_1B_2 \cdots B_k$, it contains the transition $((q, c, A), (q, B_1B_2 \cdots B_k))$. We now argue that $L(G) = L(P)$. The construction satisfies the following property (see [15]): for each $w, z \in \Sigma^*$, $\gamma \in N^*$, and $A \in N$, $A \xrightarrow{n}_G w\gamma$ via a leftmost derivation iff $(q, wz, A) \xrightarrow{n}_P (q, z, \gamma)$. Now for any string $w \in L(G)$, we have $S \longrightarrow^*_G w$ via a leftmost derivation. From the claim above, we have a run for w in P, and hence $w \in L(P)$. Conversely, for any run of P on a string w, we have $S \longrightarrow^*_G w\gamma$ via a leftmost derivation, for some $\gamma \in N^*$ (again from the above claim). Since there are no useless non-terminals in G, there exists some terminal string x such that $S \longrightarrow^*_G wx$. Thus $wx \in L(G)$ and since G is a prefix-closed language $w \in L(G)$. Hence the class of languages generated by PDSs are exactly the prefix-closed context-free languages. □

The problem we now consider, which we call the problem of model checking pushdown systems for BSPs, is the following: Given a PDS M over an input alphabet Σ, and a BSP P w.r.t. a view $\mathcal{V}$ of Σ, does $L(M)$ satisfy P?

We show that this problem is undecidable. We use a reduction from the emptiness problem of a Turing machine, which is known to be undecidable.

We recall that a Turing machine M is of the form $(Q, \Sigma_M, \Gamma_M, \vdash, \sqcup, \delta, s, t, r)$, 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 end-marker, $\sqcup \in \Gamma_M \setminus \Sigma_M$ is the blank symbol, $\delta : Q \times \Gamma_M \to Q \times \Gamma_M \times \{L, R\}$ 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 transitions out of r and t remain in r and t respectively. The configuration x of M at any instant is defined as a triple (q, z, n) where $q \in Q$ is a state, $z = y\sqcup^\omega$ is the infinite string describing the tape contents with $y \in \Gamma^*_M$, and n is a non-negative integer describing the head position. The next configuration relation $\rightsquigarrow$ is defined as $(p, xa\beta, n) \rightsquigarrow (q, xb\beta, n + 1)$, when $\delta(p, a) = (q, b, R)$ and $|x| = n$. Similarly $(p, xa\beta, n) \rightsquigarrow (q, xb\beta, n - 1)$, when $\delta(p, a) = (q, b, L)$.

We can represent a configuration of M as a finite string x in $(\Gamma_M \times \{-\})^*(\Gamma_M \times Q)(\Gamma_M \times \{-\})^*$. The string x will encode the "non-blank" part of the tape (the smallest prefix y of the tape contents which contains the tape head and such that the contents of the tape is $y\sqcup^\omega$) along with the head position and the current state. A pair in $\Gamma_M \times (Q \cup \{-\})$ is written vertically with the element of Γ_M on top. The pair (b, q) represents that the machine is currently reading the symbol b in state q.

We represent a computation of M as a sequence of configurations x_i separated by $\#$. It will be of the form $\#x_1\#x_2\# \cdots \#x_n\#$.

We now show that the problem of model-checking pushdown systems for BSP D is as hard as checking the language emptiness problem of a Turing

machine. Let $M = (Q, \Sigma_M, \Gamma_M, \vdash, \sqcup, \delta, s, t, r)$ be a Turing machine. Let $\Sigma = (\Gamma_M \times (Q \cup \{-\})) \cup \{\#\}$. Consider the language L_M over the alphabet $\Sigma \cup \{c\}$ defined to be the prefix closure of $L_1 \cup L_2$ where:

$$L_1 = \{c \cdot \#x_1\#x_2 \cdots x_n\# \mid \begin{array}{l} x_1 \text{ is a starting configuration,} \\ x_n \text{ is the only accepting configuration among} \\ \quad x_1, \ldots, x_n\} \end{array}$$

$$L_2 = \{\#x_1\#x_2 \cdots x_n\# \mid \begin{array}{l} x_1 \text{ is a starting configuration,} \\ x_n \text{ is the only accepting configuration among} \\ \quad x_1, \ldots, x_n, \\ \text{exists } i \in \{1, \ldots, n-1\} : x_i \not\rightsquigarrow x_{i+1}\}. \end{array}$$

Lemma 3 *L_M satisfies BSP D w.r.t. a view $\mathcal{V} = (\Sigma, \emptyset, \{c\}) \Leftrightarrow L(M) = \emptyset$.*

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. The string τ cannot be a valid computation 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. L_1 contains $c\tau$. If we delete the last c, the resulting string τ is not present in L_M. Thus D is not satisfied. Hence $L(M)$ is empty, when L_M satisfies the BSP D. □

To complete the reduction, we need to show how to translate M into a PDS accepting L_M. Since CFLs are closed under the prefix operation (adding the prefixes of the strings in the language) and as prefix closed CFLs are equivalent to PDS (from Lemma 2), it is enough to show that $L_1 \cup L_2$ is a CFL.

Consider the above defined language $L_2 \subseteq \Sigma^*$. L_2 can be thought of as the intersection of languages A_1, A_2, A_3, A_4 and $\neg A_5$, where

$$\begin{array}{ll} A_1 = & \{w \mid w \text{ begins and ends with } \#\} \\ A_2 = & \{w \mid \text{ the string between any two \#s must be a valid configuration}\} \\ A_3 = & \{\#x\#w \mid x \in (\Sigma \setminus \{\#\})^* \text{ is a starting configuration}\} \\ A_4 = & \{w\#x\# \mid x \in (\Sigma \setminus \{\#\})^* \text{ is the only accepting configuration in } w\#x\#\} \\ A_5 = & \{\#x_1\# \cdots \#x_n\# \mid \forall i\ 1 \leq i < n : \ x_i \rightsquigarrow x_{i+1}\}. \end{array}$$

We now show that the languages from A_1 to A_4 are regular, and $\neg A_5$ is context free. Since CFLs are closed under intersection with regular languages, it follows that L_2 is context free. Note that L_1 is the result of language-concatenation of $\{c\}$ with 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 languages A_1 to A_4 are regular. The language A_1 can be generated by the regular expression $\#(\Sigma \setminus \{\#\})^*\#$. 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 $\#(\vdash, s)K^*\#\Sigma^*$, with $K = \Gamma \setminus \{\vdash\} \times \{-\}$.

To check that a string is in A_4, we only need to check that the accepting state t appears exactly once in the string. This can be easily checked by an automaton.

For $\neg A_5$, we construct a nondeterminstic pushdown automaton (NPDA). Consider a string $\cdots\#\alpha\#\beta\#\cdots$ in the intersection of languages A_1 to A_4. Note that if $\alpha \rightsquigarrow \beta$, then the two configurations must agree in most symbols except for a few near the position of the head; and the differences that can occur near the position of the head must be consistent with the action of δ. We can check that $\alpha \rightsquigarrow \beta$ by checking for all three-element substrings (triples) u of α that the *corresponding* triple v of β differs from u in a way that is consistent with the operation of δ. By "corresponding" we mean occuring at the same distance from the closest # to its left. We can write down a table with all consistent pairs of triples over Σ. For any configurations α and β, if $\alpha \rightsquigarrow \beta$, then all corresponding triples of α and β are consistent. Conversely, if all corresponding triples of α and β are consistent, then $\alpha \rightsquigarrow \beta$. Thus to check that $\alpha \rightsquigarrow \beta$ does not hold, we need only check that there exists a triple of α such that the corresponding triple of β is not consistent with the action of δ. To be precise the consistency check may involve checking a triple against a corresponding pair (or vice-versa) when the head is at the right-most position of the non-blank part of the tape.

We now give an NPDA accepting $\neg A_5$. The NPDA nondeterministically guesses i for x_i and then it guesses the triple in x_i. It pushes the symbols from the start of x_i till the position of the guessed triple on to the stack, and records the triple starting at this position in x_i in its finite control. It then skips to the start of x_{i+1}. It now simultaneously reads a symbol from x_{i+1} and pops the top of the stack till the stack is empty. It has now found the start position of the corresponding triple in x_{i+1}. It reads the triple starting at this position in x_{i+1}. If the triple read is not consistent with the triple of x_i recorded in its finite control, the NPDA accepts the input; else it rejects the input. This check is standard and can be found for example in [15].

It now follows that L_1 is regular and L_2 is context free. As languages accepted by pushdown systems coincide with prefix-closed context-free languages (Lemma 2), L_M is a language of a pushdown system. Since the emptiness problem of Turing machine is undecidable, we get the following result:

Lemma 4 *The problem of model-checking pushdown systems for BSP D 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 5 *L_M satisfies BSP R (similarly SR, SD, BSD) w.r.t. a view $\mathcal{V} = (\Sigma, \emptyset, \{c\}) \Leftrightarrow L(M) = \emptyset$.* □

To show undecidability of BSPs *I*, *BSI*, and *SI*, we consider L_M^I to be the prefix closure of $L_3 \cup L_4$ where:

$$L_3 = \{\#x_1\#x_2\cdots x_n\# \mid \begin{array}{l} x_1 \text{ is a starting configuration,} \\ x_n \text{ is the only accepting configuration among} \\ \quad x_1, \ldots, x_n\} \end{array}$$

$$L_4 = \{w \in (\Sigma \cup \{c\})^* \mid \begin{array}{l} w \restriction_\Sigma \ = \#x_1\#x_2\cdots x_n\#, \\ x_1 \text{ is a starting configuration,} \\ x_n \text{ is the only accepting configuration among} \\ \quad x_1, \ldots, x_n, \\ \exists i \in \{1, \ldots, n-1\} : x_i \not\rightsquigarrow x_{i+1}\}. \end{array}$$

Lemma 6 *L_M^I satisfies BSP I (similarly SI, BSI) w.r.t. a view $\mathcal{V} = (\Sigma, \emptyset, \{c\}) \Leftrightarrow L(M) = \emptyset$.* □

Let $X \subseteq \Sigma$. To show undecidability of the BSPs *IA* w.r.t. X, *BSIA* w.r.t. X, and *SIA* w.r.t. X, we consider L_M^{IAX} to be the prefix closure of $L_3 \cup L_4 \cup L_5$ where:

$$L_5 = X^* \cdot \{c\}.$$

Lemma 7 *L_M^{IAX} satisfies BSP IA(similarly SIA, BSIA) w.r.t. X and the view $\mathcal{V} = (\Sigma, \emptyset, \{c\}) \Leftrightarrow L(M) = \emptyset$.* □

Let $V' \subseteq \Sigma, N' = \emptyset, C' = \{c\}$ with $v \in V'$. To show undecidability of BSP *FCD*, we consider language L_M^{FCD} to be the prefix closure of $L_6 \cup L_7$ where:

$$L_6 = \{cv \cdot \#x_1\#x_2\cdots x_n\# \mid \begin{array}{l} x_1 \text{ is a starting configuration,} \\ x_n \text{ is the only accepting configuration among} \\ \quad x_1, \ldots, x_n\} \end{array}$$

$$L_7 = \{v \cdot \#x_1\#x_2\cdots x_n\# \mid \begin{array}{l} x_1 \text{ is a starting configuration,} \\ x_n \text{ is the only accepting configuration among} \\ \quad x_1, \ldots, x_n, \\ \exists i \in \{1, \ldots, n-1\} : x_i \not\rightsquigarrow x_{i+1}\}. \end{array}$$

Lemma 8 *L_M^{FCD} satisfies BSP FCD w.r.t. V', N', C' and the view $\mathcal{V} = (\Sigma, \emptyset, \{c\}) \Leftrightarrow L(M) = \emptyset$.* □

Let $V' \subseteq \Sigma, N' = \emptyset, C' = \{c\}$ with $v \in V'$. To show undecidability of BSP *FCI*, we consider language L_M^{FCI} to be the prefix closure of $L_8 \cup L_9$ where:

$$L_8 = \{v \cdot \#x_1\#x_2\cdots x_n\# \mid \begin{array}{l} x_1 \text{ is a starting configuration,} \\ x_n \text{ is the only accepting configuration among} \\ \quad x_1, \ldots, x_n\} \end{array}$$

$$L_9 = \{w \in (\Sigma \cup C')^* \mid \begin{array}{l} w \restriction_\Sigma = v \cdot \#x_1\#x_2\cdots x_n\#, \\ x_1 \text{ is a starting configuration,} \\ x_n \text{ is the only accepting configuration among} \\ \quad x_1, \ldots, x_n, \\ \exists i \in \{1, \ldots, n-1\} : x_i \not\rightsquigarrow x_{i+1}\}. \end{array}$$

Lemma 9 *L_M^{FCI} satisfies FCI w.r.t. V', N', C' and the view $\mathcal{V} = (\Sigma, \emptyset, \{c\}) \Leftrightarrow L(M) = \emptyset$.* □

Let $X \subseteq \Sigma$. Let $V' \subseteq \Sigma, N' = \emptyset, C' = \{c\}$ with $v \in V'$. To show undecidability of BSP *FCIA*, we consider language L_M^{FCIAX} to be the prefix closure of $L_8 \cup L_9 \cup L_{10}$ where:

$$L_{10} = X^* \cdot C'.$$

Lemma 10 *L_M^{FCIAX} satisfies BSP FCIA w.r.t. X, V', N', C' and the view $\mathcal{V} = (\Sigma, \emptyset, \{c\}) \Leftrightarrow L(M) = \emptyset$.* □

The complete proofs of undecidability for these BSPs can be found in Appendix C.

We can now conclude:

Theorem 2 *The problem of model-checking pushdown systems for any of the BSPs is undecidable.* □

In fact, we can see that even if we restrict the model-checking problem to pushdown systems over an input alphabet of size 3, with respect to a view (V, N, C) with $|V| = 2$ and $|C| = 1$, the problem remains undecidable. The arguments follow the same constructions above, except that we now need to encode the computation sequence using the limited alphabet. Let $\Sigma' = \{v_1, v_2, c\}$ and let $\mathcal{V}$ be the view $(\{v_1, v_2\}, \emptyset, \{c\})$ of Σ'. Each letter in $(\Gamma_M \times (Q \cup \{-\})) \cup \{\#\}$ is encoded as a string of v_1's and a computation sequence in $(\Gamma_M \times (Q \cup \{-\})) \cup \{\#\}$ is represented as a string of v_1's and v_2's, with v_2 used as a separator. We can now use the same languages L_1 to L_{10}, except that we replace the computation sequence $\#x_1\# \cdots \#x_n\#$ by its encoding using the letters of Σ', as described above.

Theorem 3 *The problem of model-checking pushdown systems over an input alphabet of size 3, for any of the BSPs with respect to a view (V, N, C) with $|V| = 2$ and $|C| = 1$, is undecidable.* □

We can conclude from Theorem 2 that it is not possible to algorithmically check BSPs – which are useful security properties in their own right – for general pushdown system models.

6 Model-Checking Non-interference properties for Pushdown systems

The undecidability result from the preceding section does not immediately tell us that verifying the non-interference properties in the literature (which Mantel showed can be expressed as conjunctions of his BSPs) is undecidable for pushdown systems. In this section, we consider the problem of model-checking pushdown systems for some well-known non-interference properties.

Let L be a language over Σ and recall that H, L and I, O are two partitions of Σ into high and low events, and input and output events.

Recall that L satisfies Non-inference (NF) if

$$\forall \tau \in L,\ \tau \restriction_{\mathsf{L}} \ \in L.$$

With the view of $\mathcal{H}$, the property NF is simply the BSP R. We see that, checking NF is undecidable for pushdown systems, since we have already proved that checking R is undecidable for pushdown systems (Section 5).

Consider now Generalized Non-Interference (GNI). Recall that L satisfies GNI if

$$\forall \tau_1, \tau_2, \tau_3\ (\tau_1\tau_2 \in L \wedge \tau_2 \restriction_{\Sigma\setminus\mathsf{HI}} = \tau_3 \restriction_{\Sigma\setminus\mathsf{HI}}) \Rightarrow\ \exists \tau_4\ (\tau_1\tau_4 \in L \wedge \tau_4 \restriction_{\mathsf{L}\cup\mathsf{HI}} = \tau_3 \restriction_{\mathsf{L}\cup\mathsf{HI}}).$$

Consider the language L^I_M (from Section 5) with the view of $\mathcal{HI}$.

Lemma 11 *L^I_M satisfies GNI $\Leftrightarrow L(M) = \emptyset$.*

Proof ($\Leftarrow$:) Let us assume $L(M) = \emptyset$. Let us see if the prefix closure of L_4 independently satisfies GNI. Pick τ_1, τ_2 and τ_3 such that $\tau_1\tau_2$ is in this set and $\tau_2 \restriction_{\Sigma\setminus\mathsf{HI}} = \tau_3 \restriction_{\Sigma\setminus\mathsf{HI}}$. Now we can pick τ_4 to be τ_3 itself and we will have $\tau_1\tau_4$ in the set from the construction of this set. Hence the prefix closure of L_4 independently satisfies GNI. Now consider string τ_1, τ_2 and τ_3 such that $\tau_1\tau_2$ in the prefix closure of L_3. Since there are no valid computations as $L(M)$ is empty, we will have $\tau_1\tau_2$ in the prefix closure of L_4 also. From the above argument, we can say that L^I_M satisfies GNI.

($\Rightarrow$:) If $L(M)$ is not empty then there exists a valid computation. Then we have strings τ_1, τ_2 and τ_3 such that $\tau_1\tau_2$ is in the prefix closure of L_3 and not in the prefix closure of L_4 and $\tau_3 = c\tau_2$ where $c \in \mathsf{HI}$. Now, we cannot find any string τ_4 such that $\tau_1\tau_4$ is in L^I_M and $\tau_4 \restriction_{\mathsf{L}\cup\mathsf{HI}} = \tau_3 \restriction_{\mathsf{L}\cup\mathsf{HI}}$. Hence GNI is not satisfied. Hence, $L(M)$ is empty when L^I_M satisfies GNI. □

We can use the same language L^I_M and similar arguments for proving the undecidability of checking the property Separability (SEP) and Non-deducibility for outputs (NDO) for pushdown systems.

Recall that L satisfies Forward Correctability (FC) if

$$\begin{array}{l}\forall \tau_1\tau_2\ \forall c \in \mathsf{HI}\ \forall v' \in \mathsf{LI} \\ ((\tau_1 v' \tau_2 \in L\ \wedge\ \tau_2 \restriction_{\mathsf{HI}} = \epsilon)\ \Rightarrow\ \exists \tau_3\ (\tau_1 c v' \tau_3 \in L \wedge\ \tau_3 \restriction_{\mathsf{L}} = \tau_2 \restriction_{\mathsf{L}} \wedge \tau_3 \restriction_{\mathsf{HI}} = \epsilon) \wedge \\ (\tau_1 c v' \tau_2 \in L\ \wedge\ \tau_2 \restriction_{\mathsf{HI}} = \epsilon)\ \Rightarrow\ \exists \tau_3\ (\tau_1 v' \tau_3 \in L \wedge\ \tau_3 \restriction_{\mathsf{L}} = \tau_2 \restriction_{\mathsf{L}} \wedge \tau_3 \restriction_{\mathsf{HI}} = \epsilon))).\end{array}$$

Consider the view $\mathcal{HI}$. This property is vacuously true for $\mathsf{LI} = \emptyset$. We will prove undecidability for nonempty LI. Consider the language L^{FCI}_M with $V' = \mathsf{LI}, N' = \emptyset, C' = \mathsf{HI}$.

Lemma 12 *L^{FCI}_M satisfies FC $\Leftrightarrow L(M) = \emptyset$.*

Proof ($\Leftarrow$:) Let us assume $L(M) = \emptyset$. Now, $L_8 \subseteq L_9$. It is enough to prove that the set L_9 independently satisfies FC. Consider a string of the form $\tau_1 v' \tau_2$ in L_9 with $\tau_2 \restriction_{\mathsf{HI}} = \epsilon$ and $v' \in V'$. From the construction of L_9, the strings with events from HI at arbitrary points in $\tau_1 v' \tau_2$ are also in L_9. Now, consider the string $\tau_1 c v' \tau_2$ in L_9 with $\tau_2 \restriction_{\mathsf{HI}} = \epsilon$ and $c \in \mathsf{HI}$. Again by construction we have the string $\tau_1 v' \tau_2$ in L_9. Hence L_9 satisfies FC. Hence L^{FCI}_M satisfies FC.

($\Rightarrow$:) If $L(M)$ is not empty, there exists a valid computation of M. It means that there is a string $v_2\tau$ in L_8 and not in L_9. This string becomes a candidate

for L_M^{FCI} not satisfying FC, as we cannot find any string of the form $cv_2\tau$ in the language with some $c \in \mathsf{HI}$. Hence FC is not satisfied. Hence, $L(M)$ is empty when L_M^{FCI} satisfies FC. □

Hence we get the following theorem:

Theorem 4 *The problem of model-checking pushdown systems for Generalized Non-Interference, Non-inference, Separability, Non-deducibility of outputs and Forward Correctability is undecidable.*

In fact, it is apparent to see the undecidability holds for a restricted class of pushdown systems:

Theorem 5 *The problem of model-checking pushdown systems for*

1. *Generalized Non-Interference and Forward Correctability for systems with at least two low events and one high-input event, and*
2. *Non-inference, Separability and Non-deducibility of outputs for systems with at least two low events and one high event*

is undecidable. □

7 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 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 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 satisfies 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. 4. The states s_2 and s_3 have n loops each on them: state s_3 has loops on the strings $v_1x_1v_1$ through $v_nx_nv_n$, and s_3 has loops on the strings $v_1y_1v_1$ through $v_ny_nv_n$. We choose each v_i to be different from the symbols in Δ.

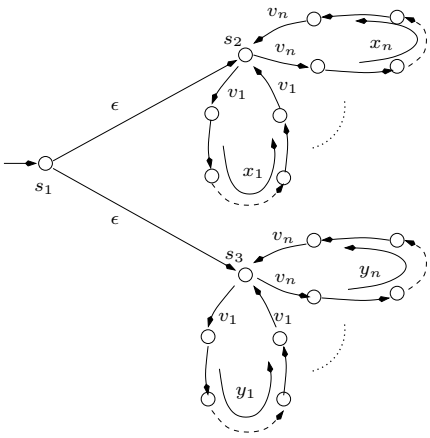

Figure 4:

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 event from V 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 13 *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 $L(\mathcal{T}_P)$ with non-empty projection to C events 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 traced out by τ and τ'.

($\Rightarrow$:) Let $i_1, i_2, \ldots, i_l$, $l \geq 1$, 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 non-empty and 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 6 *The problem of model-checking the property WNI for finite-state systems is undecidable.* □

We note that the problem is still undecidable if we restrict the view of the finite-state system to have two confidential events and two visible events. This is because (a) PCP is undecidable for an alphabet $|\Sigma| = 2$, which are modeled by confidential events and (b) the events $v_1 \cdots v_n$ can appropriately be modeled by two events, say v_1' and v_2' and we can consider v_1' and v_2' to be the only two visible events.

We also 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 [9] would imply that model-checking *WNI* for finite-state systems is decidable. Hence we can conclude that:

Corrollary 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 7 *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 [15]), 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 $X = \{(m_1, n_1), \ldots, (m_k, n_k)\}$ we denote the set of vectors *generated* by X, with initial vector (m_0, n_0), to be the set $gen_{(m_0,n_0)}(X) = \{(m_0, n_0) + t_1(m_1, n_1) + \cdots + t_k(m_k, n_k) \mid t_1, \ldots, t_k \in \mathbb{N}\}$. Then $\psi(L)$ is semi-linear means that there exist finite non-empty sets of vectors $X_1, \ldots, X_l$, and initial vectors $(m_0^1, n_0^1), \ldots, (m_0^l, n_0^l)$, such that

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

Now let L be the given context-free language over the alphabet $\Sigma = \{v, c\}$, with its Parikh map given by the sets $X_1, \ldots, X_l$, and initial vectors $(m_0^1, n_0^1), \ldots$, (m_0^l, n_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 $\psi(L)$, whose confidential event count is non-zero, there exists another vector which is in $gen_{(m_0^i, n_0^i)}(X_i)$ for some i, such that their visible event count is the same, and confidential event count is different. For $l = 1$, L satisfies *WNI* iff the following formula in Presburger arithmetic is valid (Presburger arithmetic is the first-order theory of natural numbers with addition which is known to be decidable in double-exponential time):

$$\forall t_1, \ldots, t_{k_1} \in \mathbb{N} \Big(n_0^1 + t_1 n_1^1 + \cdots + t_{k_1} n_{k_1}^1 > 0 \implies$$
$$\exists t'_1, \ldots, t'_{k_1} \in \mathbb{N} \left(m_0^1 + t_1 m_1^1 + \cdots + t_{k_1} m_{k_1}^1 = m_0^1 + t'_1 m_1^1 + \cdots + t'_{k_1} m_{k_1}^1 \wedge \right.$$
$$\left. n_0^1 + t_1 n_1^1 + \cdots + t_{k_1} n_{k_1}^1 \neq n_0^1 + t'_1 n_1^1 + \cdots + t'_{k_1} n_{k_1}^1 \right) \Big).$$

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. For example, let us consider the BSP R. For a view with $|C| = 0$, the BSP R is vacuously satisfied. For a view with $|V| = 0$, L satisfies BSP R when $L = \epsilon \vee L \cap N^* \neq \epsilon$. This is decidable for PDS as it amounts to checking the emptiness of the given PDS language or with its intersection with a regular language N^*. For a view with $|C| = 1$ and $|V| = 1$, the BSP R can be seen as a property with the counting relationship between visible and confidential events. The exact presburger formula is:

$$\forall t_1, \ldots, t_{k_1} \in \mathbb{N} \Big(n_0^1 + t_1 n_1^1 + \cdots + t_{k_1} n_{k_1}^1 > 0 \implies$$
$$\exists t'_1, \ldots, t'_{k_1} \in \mathbb{N} \left(m_0^1 + t_1 m_1^1 + \cdots + t_{k_1} m_{k_1}^1 = m_0^1 + t'_1 m_1^1 + \cdots + t'_{k_1} m_{k_1}^1 \wedge \right.$$
$$\left. n_0^1 + t'_1 n_1^1 + \cdots + t'_{k_1} n_{k_1}^1 = 0 \right) \Big).$$

Corrollary 2 *The problem of model-checking BSP R (and SR and Non-inference (NF)) for pushdown systems is decidable for a view $\mathcal{V} = (V, N, C)$ with $|V| \leq 1$ and $|C| \leq 1$.* □

8 Conclusion

We have demonstrated in this paper a way to algorithmically verify trace-based information flow properties for finite-state systems. We give characterisations of the properties in terms of language-theoretic operations on the set of traces of a system, rather than in terms of the structure of the system which is a stronger notion. This perhaps explains why we are able to obtain complete characterisations unlike the previous techniques in the literature that are based on unwinding relations.

We also considered the problem of model-checking trace-based information flow properties for pushdown systems. We have shown that this problem is undecidable in general.

We also studied the model-checking problem for a property called *WNI* which is a weak form of the property of non-inference. 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.

An interesting future direction to pursue is whether the technique of this paper can be extended to a more general logical language. The BSPs are all of the form "for every string in the language satisfying some conditions, there exists a string satisfying some other conditions", which are special cases of a first order logic interpreted over languages of strings. It would be interesting to investigate the decidability of such a logic when interpreted over regular languages, and to identify a natural decidable subclass of it which properly contains the BSPs of Mantel.

Another open question is whether the model-checking problem continues to be undecidable even when we consider *deterministic* pushdown systems.

Acknowledgements

We thank Janardhan Kulkarni for fruitful discussions on Weak Noninference property. We also thank the anonymous referees for several useful inputs.

References

[1] Rajeev Alur, Michael Benedikt, Kousha Etessami, Patrice Godefroid, Thomas Reps, and Mihalis Yannakakis. Analysis of recursive state machines. *ACM Transactions on Programming Languages and Systems*, 27(4):786–818, 2005.

[2] Thomas Ball and Sriram K. Rajamani. Bebop: A symbolic model checker for boolean programs. In Klaus Havelund, John Penix, and Willem Visser, editors, *SPIN*, volume 1885 of *Lecture Notes in Computer Science*, pages 113–130. Springer, 2000.

[3] Ahmed Bouajjani, Javier Esparza, and Oded Maler. Reachability analysis of pushdown automata: Application to model-checking. In *CONCUR '97: Proceedings of the 8th International Conference on Concurrency Theory*, pages 135–150, London, UK, 1997. Springer-Verlag.

[4] Gérard Boudol and Ilaria Castellani. Noninterference for concurrent programs and thread systems. *Theoretical Computer Science*, 1-2(281):109–130, 2002.

[5] Mads Dam. Decidability and proof systems for language-based noninterference relations. In *POPL '06: Conference record of the 33rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 67–78, New York, NY, USA, 2006. ACM.

[6] Dorothy E. Denning and Peter J. Denning. Certification of programs for secure information flow. *Commun. ACM*, 20(7):504–513, 1977.

[7] Deepak D'Souza, Raveendra Holla, Janardhan Kulkarni, Raghavendra K R, and Barbara Sprick. On the decidability of model-checking information flow properties. In *Proceedings of International Conference on Information Systems Security (ICISS)*, 2008.

[8] Deepak D'Souza and Raghavendra K. R. Checking unwinding conditions for finite state systems. In *Proceedings of VERIFY workshop*, 2006.

[9] Deepak D'Souza, Raghavendra K. R., and Barbara Sprick. An automata based approach for verifying information flow properties. *Proceedings of the second workshop on Automated Reasoning for Security Protocol Analysis (ARSPA 2005), Electronic Notes in Theoretical Computer Science*, 135(1):39–58, July 2005.

[10] Riccardo Focardi and Roberto Gorrieri. A classification of security properties for process algebras. *Journal of Computer Security, IOS Press*, 3(1):5–33, 1995.

[11] Riccardo Focardi and Roberto Gorrieri. The compositional security checker: A tool for the verification of information flow security properties. *Software Engineering*, 23(9):550–571, 1997.

[12] Joseph A. Goguen and José Meseguer. Security policies and security models. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 11–20, April 1982.

[13] Joshua D. Guttman and Mark E. Nadel. What needs securing. In *Proceedings of Computer Security Foundations Workshop*, pages 34–57, 1988.

[14] Dale M. Johnson and F. Javier Thayer. Security and the composition of machines. *Proceedings of Computer Security Foundations Workshop*, pages 72–89, 1988.

[15] Dexter Kozen. *Automata and Computability.* Springer-Verlag, New York, 1997.

[16] 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.

[17] Heiko Mantel. *A Uniform Framework for the Formal Specification and Verification of Information Flow Security.* PhD thesis, Universität des Saarlandes, 2003.

[18] Daryl McCullough. Specifications for multilevel security and a hookup property. In *Proceedings of 1987 IEEE Symposium Security and Privacy*, 1987.

[19] John McLean. A general theory of composition for trace sets closed under selective interleaving functions. In *Proceedings of IEEE Symposium on Research in Security and Privacy*, pages 79 – 93. IEEE Computer Society Press, 1994.

[20] Colin O'Halloran. A calculus of information flow. In *Proceedings of the European Symposium on Research in Computer Security, ESORICS 90*, 1990.

[21] Willard V. Quine. Concatenation as a basis for finite arithmetic. *Journal of Symbolic Logic*, 11(4), 1946.

[22] Andrej Sabelfeld and David Sands. A per model of secure information flow in sequential programs. *Higher-Order and Symbolic Computation*, 1(14):59–91, 2001.

[23] David Sutherland. A model of information. In *Proceedings of the 9th National Computer Security Conference*, 1986.

[24] Ron van der Meyden and Chenyi Zhang. Algorithmic verification of noninterference properties. *Electronic Notes in Theoretical Computer Science*, 168:61–75, 2007.

[25] 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 A

Here are the proofs for the rest of the claims in Lemma 1 characterizing BSPs in terms of language theoretic operations.

1. L satisfies IA w.r.t. X iff $l\text{-}ins\text{-}adm^X(L) \subseteq_N L$.

   Proof ($\Rightarrow$:) Assume L satisfies IA and consider a string τ in $l\text{-}ins\text{-}adm^X(L)$. Then τ will be of the form $\alpha c\beta$ for some $c \in C$ and there must exist $\alpha\beta \in L$ such that $\beta \restriction_C = \epsilon$ and also $\gamma c \in L$ with $\gamma \restriction_X = \alpha \restriction_X$. Since L satisfies IA, there exists $\alpha' c\beta' \in L$ with $\alpha' =_N \alpha$ and $\beta' =_N \beta$. But this just means that $\tau = \alpha c\beta =_N \alpha' c\beta'$. Hence $l\text{-}ins\text{-}adm^X(L) \subseteq_N L$.

   ($\Leftarrow$:) Suppose $l\text{-}ins\text{-}adm^X(L) \subseteq_N L$. Consider a string $\alpha\beta \in L$ with $\beta \restriction_C = \epsilon$ and there exists $\gamma c \in L$ for some $c \in C$ with $\gamma \restriction_X = \alpha \restriction_X$. By the definition of $l\text{-}ins\text{-}adm^X(L)$, we have $\alpha c\beta \in l\text{-}ins\text{-}adm^X(L)$. Since $l\text{-}ins\text{-}adm^X(L) \subseteq_N L$, there exists $\tau \in L$ such that $\alpha c\beta =_N \tau$. Thus τ can be expressed as $\alpha' c\beta'$ such that $\alpha' =_N \alpha$ and $\beta' =_N \beta$. Hence IA is satisfied. □

2. L satisfies BSD iff $l\text{-}del\text{-}mark(L) \restriction^m_{\overline{N}} \subseteq mark(L) \restriction^m_{\overline{N}}$.

   Proof ($\Rightarrow$:) Assume L satisfies BSD and consider a string τ belonging to $l\text{-}del\text{-}mark(L) \restriction^m_{\overline{N}}$. Then τ must be of the form $\alpha\natural\beta'$, which comes from a string of the form $\alpha\natural\beta$ in $l\text{-}del\text{-}mark(L)$ with $\beta \restriction_{\overline{N}} = \beta'$, which in turn must be such that $\alpha c\beta \in L$ for some $c \in C$ with $\beta \restriction_C = \epsilon$. Since L satisfies BSD, we have $\alpha\beta'' \in L$ such that $\beta =_N \beta''$. By the definition of $mark(L)$, we have $\alpha\natural\beta'' \in mark(L)$. Deleting N-events from β'' results in β'. Thus $\alpha\natural\beta' = \tau$ must be in $mark(L) \restriction^m_{\overline{N}}$. Hence $l\text{-}del\text{-}mark(L) \restriction^m_{\overline{N}}$ is a subset of $mark(L) \restriction^m_{\overline{N}}$.

   ($\Leftarrow$:) Suppose $l\text{-}del\text{-}mark(L) \restriction^m_{\overline{N}} \subseteq mark(L) \restriction^m_{\overline{N}}$. Consider a string $\alpha c\beta \in L$, with $c \in C$ and $\beta \restriction_C = \epsilon$. By the definition of $l\text{-}del\text{-}mark(L)$, we have $\alpha\natural\beta \in l\text{-}del\text{-}mark(L)$ and hence $\alpha\natural\beta' \in l\text{-}del\text{-}mark(L) \restriction^m_{\overline{N}}$ where β' is obtained from β by deleting N-events. Since $l\text{-}del\text{-}mark(L) \restriction^m_{\overline{N}} \subseteq mark(L) \restriction^m_{\overline{N}}$, we have $\alpha\natural\beta' \in mark(L) \restriction^m_{\overline{N}}$. Then there must exist $\alpha\natural\beta'' \in mark(L)$ where $\beta'' \restriction_{\overline{N}} = \beta'$. Thus $\beta'' =_N \beta$. By the definition of $mark(L)$, we have $\alpha\beta'' \in L$. Hence BSD is satisfied. □

3. L satisfies BSI iff $l\text{-}ins\text{-}mark(L) \restriction^m_{\overline{N}} \subseteq mark(L) \restriction^m_{\overline{N}}$.

   Proof ($\Rightarrow$:) Assume L satisfies BSI and consider a string τ belonging to $l\text{-}ins\text{-}mark(L) \restriction^m_{\overline{N}}$. Then τ will be of the form $\alpha c\natural\beta'$ which comes from a string of the form $\alpha c\natural\beta$ in $l\text{-}ins\text{-}mark(L)$ which in turn must be such that $\beta \restriction_{\overline{N}} = \beta'$, $c \in C$, $\alpha\beta \in L$ and $\beta \restriction_C = \epsilon$. Since L satisfies BSI, we have $\alpha c\beta'' \in L$ such that $\beta =_N \beta''$. By the definition of $mark(L)$, we have $\alpha c\natural\beta'' \in mark(L)$. Deleting N-events from β'' results in β'. Thus $\alpha c\natural\beta' = \tau$ must be in $mark(L) \restriction^m_{\overline{N}}$. Hence $l\text{-}ins\text{-}mark(L) \restriction^m_{\overline{N}}$ is a subset of $mark(L) \restriction^m_{\overline{N}}$.

   ($\Leftarrow$:) Suppose $l\text{-}ins\text{-}mark(L) \restriction^m_{\overline{N}} \subseteq mark(L) \restriction^m_{\overline{N}}$. Consider a string $\alpha\beta \in L$ such that $\beta \restriction_C = \epsilon$. By the definition of $l\text{-}ins\text{-}mark(L)$, we have $\alpha c\natural\beta \in l\text{-}ins\text{-}mark(L)$ for any $c \in C$ and hence $\alpha c\natural\beta' \in l\text{-}ins\text{-}mark(L) \restriction^m_{\overline{N}}$ where β' is obtained from β by deleting N-events. Since $l\text{-}ins\text{-}mark(L) \restriction^m_{\overline{N}} \subseteq mark(L) \restriction^m_{\overline{N}}$, we have $\alpha c\natural\beta' \in mark(L) \restriction^m_{\overline{N}}$. Then there must exist $\alpha c\natural\beta'' \in$

$mark(L)$ where $\beta'' \restriction_{\overline{N}} = \beta'$. Thus $\beta'' =_N \beta$. By the definition of $mark(L)$, we have $\alpha c \beta'' \in L$. Hence *BSI* is satisfied. □

4. L satisfies *BSIA* w.r.t. X iff *l-ins-adm-mark*$^X(L) \restriction^m_{\overline{N}} \subseteq mark(L) \restriction^m_{\overline{N}}$.

   Proof ($\Rightarrow$:) Assume L satisfies *BSIA* and consider a string τ belonging to *l-ins-adm-mark*$^X(L) \restriction^m_{\overline{N}}$. Then τ will be of the form $\alpha c \natural \beta'$, which comes from a string of the form $\alpha c \natural \beta$ in *l-ins-adm-mark*$^X(L)$ with $\beta \restriction_{\overline{N}} = \beta'$, which in turn must be such that $\beta \restriction_C = \epsilon$, $c \in C$, $\alpha\beta \in L$, $\beta \restriction_C = \epsilon$ and there exists $\gamma c \in L$ with $\gamma \restriction_X = \alpha \restriction_X$. Since L satisfies *BSIA*, we have $\alpha c \beta'' \in L$ where $\beta =_N \beta''$. By the definition of $mark(L)$, we have $\alpha c \natural \beta'' \in mark(L)$. Deleting N-events from β'' results in β'. Thus $\alpha c \natural \beta' = \tau$ must be in $mark(L) \restriction^m_{\overline{N}}$. Hence *l-ins-adm-mark*$^X(L) \restriction^m_{\overline{N}}$ is a subset of $mark(L) \restriction^m_{\overline{N}}$.

   ($\Leftarrow$:) Suppose *l-ins-adm-mark*$^X(L) \restriction^m_{\overline{N}}$ is a subset of $mark(L) \restriction^m_{\overline{N}}$. Consider a string $\alpha\beta \in L$ such that $\beta \restriction_C = \epsilon$ and there exists $\gamma c \in L$ with $c \in C$ and $\gamma \restriction_X = \alpha \restriction_X$. By the definition of *l-ins-adm-mark*$^X(L)$, we have $\alpha c \natural \beta \in$ *l-ins-adm-mark*$^X(L)$ and hence $\alpha c \natural \beta' \in$ *l-ins-adm-mark*$^X(L) \restriction^m_{\overline{N}}$ where β' is obtained from β by deleting N-events. Since *l-ins-adm-mark*$^X(L) \restriction^m_{\overline{N}}$ is a subset of $mark(L) \restriction^m_{\overline{N}}$, we have $\alpha c \natural \beta' \in mark(L) \restriction^m_{\overline{N}}$. Then there must exist $\alpha c \natural \beta'' \in mark(L)$ where $\beta'' \restriction_{\overline{N}} = \beta'$. Thus $\beta'' =_N \beta$. By the definition of $mark(L)$, there exists $\alpha c \beta'' \in L$. Hence *BSIA* is satisfied. □

5. L satisfies *FCD* w.r.t. V', C', N' iff

$$\textit{l-del-con-mark}_{C',V'}(L) \restriction^m_{\overline{N}} \subseteq \textit{erase-con-mark}_{N',V'}(L) \restriction^m_{\overline{N}}$$

   Proof ($\Rightarrow$:) Assume L satisfies *FCD* and consider a string τ belonging to *l-del-con-mark*$_{C',V'}(L) \restriction^m_{\overline{N}}$. Then τ will be of the form $\alpha v \natural \beta'$ which comes from a string $\alpha v \natural \beta$ in *l-del-con-mark*$_{C',V'}(L)$ with $\beta \restriction_{\overline{N}} = \beta'$. Then we must have a string $\alpha c v \beta \in L$ with $\beta \restriction_C = \epsilon$, $c \in C'$ and $v \in V'$. Since L satisfies *FCD*, we have $\alpha\delta v\beta'' \in L$ where $\beta'' =_N \beta$. By the definition of *erase-con-mark*$_{N',V'}(L)$, we have $\alpha v \natural \beta'' \in$ *erase-con-mark*$_{N',V'}(L)$. Deleting N symbols from β'' results in β'. Thus $\alpha v \natural \beta'$ must be in *erase-con-mark*$_{N',V'}(L) \restriction^m_{\overline{N}}$. Hence *l-del-con-mark*$_{C',V'}(L) \restriction^m_{\overline{N}}$ is a subset of *erase-con-mark*$_{N',V'}(L) \restriction^m_{\overline{N}}$.

   ($\Leftarrow$:) Suppose *l-del-con-mark*$_{C',V'}(L) \restriction^m_{\overline{N}} \subseteq$ *erase-con-mark*$_{N',V'}(L) \restriction^m_{\overline{N}}$. Consider a string $\alpha c v \beta \in L$, with $c \in C'$, $v \in V'$ and $\beta \restriction_C = \epsilon$. From the definition of *l-del-con-mark*$_{C',N'}(L)$, we have $\alpha v \natural \beta \in$ *l-del-con-mark*$_{C',N'}(L)$. Thus $\alpha v \natural \beta' \in$ *l-del-con-mark*$_{C',N'}(L) \restriction^m_{\overline{N}}$ where β' is obtained from β by deleting N-events. Since *l-del-con-mark*$_{C',V'}(L) \restriction^m_{\overline{N}} \subseteq$ *erase-con-mark*$_{N',V'}(L) \restriction^m_{\overline{N}}$, we have $\alpha v \natural \beta' \in$ *erase-con-mark*$_{N',V'}(L) \restriction^m_{\overline{N}}$. Then there must exist $\alpha v \natural \beta'' \in$ *erase-con-mark*$_{N',V'}(L)$ where $\beta'' \restriction_{\overline{N}} = \beta'$. Thus $\beta'' =_N \beta$. By the definition of *erase-con-mark*$_{N',V'}(L)$, we have $\alpha\delta v\beta'' \in L$ for some δ such that $\delta \in (N')^*$. Hence *FCD* is satisfied. □

6. L satisfies *FCI* w.r.t. V', C', N' iff

$$\textit{l-ins-con-mark}_{C',V'}(L) \restriction^m_{\overline{N}} \subseteq \textit{erase-con-mark}_{N',V'}(L) \restriction^m_{\overline{N}}$$

   Proof ($\Rightarrow$:) Assume L satisfies *FCI* and consider a string τ belonging to *l-ins-con-mark*$_{C',V'}(L) \restriction^m_{\overline{N}}$. Then τ will be of the form $\alpha c v \natural \beta'$ which

comes from some string $\alpha cv\natural\beta$ in *l-ins-con-mark*$_{C',V'}(L)$ which in turn must be such that $c \in C'$, $v \in V'$, $\beta' = \beta \restriction_{\overline{N}}$, $\alpha v\beta \in L$ and $\beta \restriction_C = \epsilon$. Since L satisfies *FCI*, we have $\alpha c\delta v\beta'' \in L$ for some δ such that $\delta \in (N')^*$ and $\beta'' =_N \beta$. By the definition of *erase-con-mark*$_{N',V'}(L)$, we have $\alpha cv\natural\beta'' \in$ *erase-con-mark*$_{N',V'}(L)$. Deleting N-symbols from β'' results in β'. Thus $\alpha cv\natural\beta' = \tau$ must be in *erase-con-mark*$_{N',V'}(L) \restriction^m_{\overline{N}}$. Hence *l-ins-con-mark*$_{C',V'}(L) \restriction^m_{\overline{N}}$ is a subset of *erase-con-mark*$_{N',V'}(L) \restriction^m_{\overline{N}}$.

($\Leftarrow$:) Suppose *l-ins-con-mark*$_{C',V'}(L) \restriction^m_{\overline{N}} \subseteq$ *erase-con-mark*$_{N',V'}(L) \restriction^m_{\overline{N}}$. Consider a string $\alpha v\beta \in L$ with $v \in V'$ and $\beta \restriction_C = \epsilon$. By the definition of *l-ins-con-mark*$_{C',V'}(L)$, we have $\alpha cv\natural\beta \in$ *l-ins-con-mark*$_{C',V'}(L)$ for any $c \in C'$ and hence $\alpha cv\natural\beta' \in$ *l-ins-con-mark*$_{C',V'}(L) \restriction^m_{\overline{N}}$ where β' is obtained from β by deleting N-events. Since *l-ins-con-mark*$_{C',V'}(L) \restriction^m_{\overline{N}} \subseteq$ *erase-con-mark*$_{N',V'}(L) \restriction^m_{\overline{N}}$, we have $\alpha cv\natural\beta' \in$ *erase-con-mark*$_{N',V'}(L) \restriction^m_{\overline{N}}$. Then there must exist $\alpha cv\natural\beta'' \in$ *erase-con-mark*$_{N',V'}(L)$ where $\beta'' \restriction_{\overline{N}} = \beta'$. Thus $\beta'' =_N \beta$. By the definition of *erase-con-mark*$_{N',V'}(L)$, we have $\alpha c\delta v\beta'' \in L$. Hence *FCI* is satisfied. □

7. L satisfies *FCIA* w.r.t. X, V', C', N' iff

$$\textit{l-ins-adm-con-mark}^X_{C',V'}(L) \restriction^m_{\overline{N}} \subseteq \textit{ erase-con-mark}_{N',V'}(L) \restriction^m_{\overline{N}}.$$

Proof ($\Rightarrow$:) Suppose L satisfies *FCIA* w.r.t. X, V', C', N'. Consider a string $\tau \in$ *l-ins-adm-con-mark*$^X_{C',V'}(L) \restriction^m_{\overline{N}}$. τ must be of the form $\alpha cv\natural\beta'$, which comes from some string of the form $\alpha cv\natural\beta$ in *l-ins-adm-con-mark*$^X_{C',V'}(L)$ which in turn must be such that $\beta' = \beta \restriction_{\overline{N}}$, $c \in C'$, $v \in V'$, $\alpha v\beta \in L$, $\beta \restriction_C = \epsilon$, and there exists $\gamma c \in L$ with $\gamma \restriction_X = \alpha \restriction_X$. Now since $\alpha v\beta \in L$ and L satisfies *FCIA*, there must exist $\alpha c\delta v\beta'' \in L$ with $\delta \in (N')^*$ and $\beta'' =_N \beta$. From the definition of *erase-con-mark*$_{N',V'}(L)$, we have $\alpha cv\natural\beta'' \in$ *erase-con-mark*$_{N',V'}(L)$. Deleting N events from β'' results in β'. Thus $\alpha cv\beta' = \tau$ must be in *erase-con-mark*$_{N',V'}(L) \restriction^m_{\overline{N}}$. Hence *l-ins-adm-con-mark*$^X_{C',V'}(L) \restriction^m_{\overline{N}}$ is a subset of *erase-con-mark*$_{N',V'}(L) \restriction^m_{\overline{N}}$.

($\Leftarrow$:) Assume *l-ins-adm-con-mark*$^X_{C',V'}(L) \restriction^m_{\overline{N}} \subseteq$ *erase-con-mark*$_{N',V'}(L) \restriction^m_{\overline{N}}$. Consider a string of the form $\alpha v\beta \in L$, such that $v \in V'$, $\beta \restriction_C = \epsilon$, and there exists $\gamma c \in L$ with $c \in C'$ and $\gamma \restriction_X = \alpha \restriction_X$. By the definition of *l-ins-adm-con-mark*$^X_{C',V'}(L)$, $\alpha cv\natural\beta \in$ *l-ins-adm-con-mark*$^X_{C',V'}(L)$. Thus $\alpha cv\natural\beta'$ is in *l-ins-adm-con-mark*$^X_{C',V'}(L) \restriction^m_{\overline{N}}$ where β' is obtained from β by deleting N-events. Since *l-ins-adm-con-mark*$^X_{C',V'}(L) \restriction^m_{\overline{N}} \subseteq$ *erase-con-mark*$_{N',V'}(L) \restriction^m_{\overline{N}}$, we have $\alpha cv\natural\beta' \in$ *erase-con-mark*$_{N',V'}(L) \restriction^m_{\overline{N}}$. Hence $\alpha cv\natural\beta''$ must belong to *erase-con-mark*$_{N',V'}(L)$ for some β'' such that $\beta'' \restriction_{\overline{N}} = \beta'$. Thus $\beta'' =_N \beta$. By the definition of *erase-con-mark*$_{N',V'}(L)$, there exists $\delta \in (N')^*$ such that $\alpha c\delta v\beta'' \in L$. Hence L satisfies *FCIA*. □

8. L satisfies *SR* iff $L \restriction_{\overline{C}} \subseteq L$.

Proof ($\Rightarrow$:) Assume L satisfies *SR* and consider any string τ in $L \restriction_{\overline{C}}$. There must exist some $\tau' \in L$ such that $\tau' \restriction_{\overline{C}} = \tau$. Since L satisfies *SR*, we have $\tau = \tau' \restriction_{\overline{C}} \in L$. Hence $L \restriction_{\overline{C}} \subseteq L$.

($\Leftarrow$:) Suppose $L \restriction_{\overline{C}} \subseteq L$. Consider any string τ in L. Then $\tau \restriction_{\overline{C}} \in L \restriction_{\overline{C}}$. Since $L \restriction_{\overline{C}} \subseteq L$, we have $\tau \restriction_{\overline{C}} \in L$. Hence *SR* is satisfied. □

9. L satisfies SD iff $l\text{-}del(L) \subseteq L$.

   Proof ($\Rightarrow$:) Assume L satisfies SD and consider a string τ in $l\text{-}del(L)$. Then τ will be of the form $\alpha\beta$ which comes from a string of the form $\alpha c\beta \in L$ such that $c \in C$ and $\beta \restriction_C = \epsilon$. Since L satisfies SD, we have $\alpha\beta \in L$. Hence $l\text{-}del(L) \subseteq L$.

   ($\Leftarrow$:) Suppose $l\text{-}del(L) \subseteq L$. Consider a string $\alpha c\beta \in L$ with $c \in C$ and $\beta \restriction_C = \epsilon$. By the definition of $l\text{-}del(L)$, we have $\alpha\beta \in l\text{-}del(L)$. Since $l\text{-}del(L) \subseteq L$, we have $\alpha\beta \in L$. Hence SD is satisfied. □

10. L satisfies SI iff $l\text{-}ins(L) \subseteq L$.

    Proof ($\Rightarrow$:) Assume L satisfies SI and consider a string $\tau \in l\text{-}ins(L)$. Then τ will be of the form $\alpha c\beta$ which comes from a string of the form $\alpha\beta \in L$ such that $c \in C$ and $\beta \restriction_C = \epsilon$. Since L satisfies SI, we have $\alpha c\beta \in L$. Hence $l\text{-}ins(L) \subseteq L$.

    ($\Leftarrow$:) Suppose $l\text{-}ins(L) \subseteq L$. Consider a string $\alpha\beta \in L$ such that $\beta \restriction_C = \epsilon$. By the definition of $l\text{-}ins(L)$, we have $\alpha c\beta \in l\text{-}ins(L)$ for any $c \in C$. Since $l\text{-}ins(L) \subseteq L$, we have $\alpha c\beta \in L$. Hence SI is satisfied. □

11. L satisfies SIA w.r.t. X iff $l\text{-}ins\text{-}adm^X(L) \subseteq L$.

    Proof ($\Rightarrow$:) Assume L satisfies SIA and consider a string $\tau \in l\text{-}ins\text{-}adm^X(L)$. Then τ will be of the form $\alpha c\beta$ which comes from the string of the form $\alpha\beta \in L$ such that $c \in C$, $\beta \restriction_C = \epsilon$ and there exists $\gamma c \in L$ with $c \in C$ and $\gamma \restriction_X = \alpha \restriction_X$. Since L satisfies SIA, we have $\tau = \alpha c\beta \in L$. Hence $l\text{-}ins\text{-}adm^X(L) \subseteq L$.

    ($\Leftarrow$:) Suppose $l\text{-}ins\text{-}adm^X(L) \subseteq L$. Consider a string $\alpha\beta \in L$ such that $\beta \restriction_C = \epsilon$ and there exists $\gamma c \in L$ with $c \in C$ and $\gamma \restriction_X = \alpha \restriction_X$. By the definition of $l\text{-}ins\text{-}adm^X(L)$, $\alpha c\beta \in l\text{-}ins\text{-}adm^X(L)$. Since $l\text{-}ins\text{-}adm^X(L) \subseteq L$, $\alpha c\beta \in L$. Hence SIA is satisfied. □

Appendix B

The following arguments show that the language theoretic operations are regularity preserving and effectively so.

1. *l-del-mark*: This construction is similar to *l-del* except that the label of the ϵ-transitions we add from the first copy to the second, is now $\natural$.

2. *l-ins-mark*: The construction is similar to *l-ins*. Here instead of inserting a transition labelled c from the first copy to the second, we need to insert a transition labelled $c\natural$ from the first copy to the second. This can be carried out by having a third copy of $\mathcal{A}$ placed between the first and second. The third copy has all its transitions deleted, and all its states are non-initial and non-final. A c transition from p in the first copy now goes to p in the third copy, and from p in the third copy we add a $\natural$ transition to p in the second copy.

3. *l-ins-adm-mark*: The construction is similar to *l-ins-adm*. Instead of adding a c transition from the first copy to the second, we add one labelled $c\natural$ (once again this can be achieved using a third copy of $\mathcal{A}$).

4. *mark*: Given $\mathcal{A}$ for $L \subseteq \Sigma^*$, we construct $\mathcal{A}'$ which accepts the marked language $mark(L)$. $\mathcal{A}$ is obtained from $\mathcal{A}$ as follows. We again use two copies of $\mathcal{A}$. The initial state of $\mathcal{A}'$ is the initial state of the first copy, and the final states are only those of the second copy. From every state in the first copy we add a transition labelled $\natural$ to its copy in the second.

5. *Marked projection*: Given a marked language M, an FSA $\mathcal{A}$ accepting M, and $X \subseteq \Sigma$, we construct $\mathcal{A}'$ which accepts the marked language $M \restriction^m_X$. Once again we use two copies of $\mathcal{A}$. The initial state of the first copy is the initial state of $\mathcal{A}'$ and the final states of the second copy are the final states of $\mathcal{A}'$. From the first copy we delete transitions of the form $p \xrightarrow{\natural} q$ and add a transition labelled $\natural$ from p in the first copy to q in the second copy. In the second copy, we replace transition labels which are not in X by ϵ.

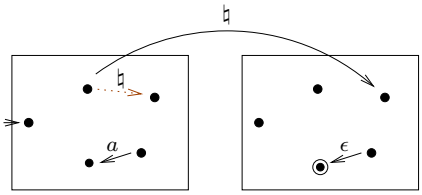

Figure 5: $L \restriction^m_X$

6. *l-del-con-mark*: Let L be a language over Σ and $\mathcal{A}$ be an FSA accepting L. Let $C' \subseteq C$ and $V' \subseteq V$. We construct $\mathcal{A}'$ accepting the marked language $\textit{l-del-con-mark}_{C',V'}(L)$ as follows. We have four copies of $\mathcal{A}$. The second and third copies have all transitions deleted from them, and the fourth copy has all C transitions deleted from it. The initial state of the first copy is the initial state of $\mathcal{A}'$ and the final states of the fourth copy are

the final states of $\mathcal{A}'$. For every transition $p \xrightarrow{c'} q$ with $c' \in C'$, we add an ϵ-transition from p in the first copy to q in the second copy. We add a v'-transition from a state r in the second copy to a state t in the third copy iff $r \xrightarrow{v'} t$, with $v' \in V'$, is a transition in $\mathcal{A}$. Finally, we add a $\natural$-transition from each state u in the third copy to u in the fourth copy.

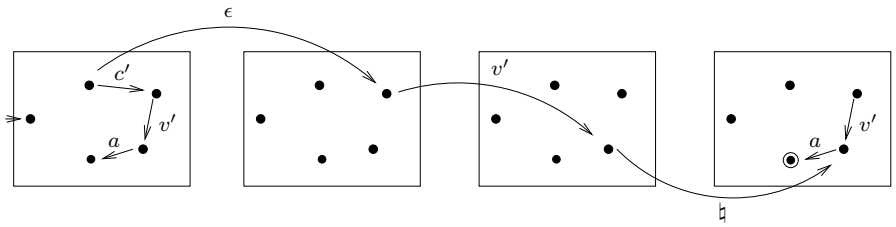

Figure 6: *l-del-con-mark*$_{C',V'}(L)$

7. *l-ins-con-mark*: Let L be a language over Σ and $\mathcal{A}$ be an FSA accepting L. Let $C' \subseteq C$ and $V' \subseteq V$. We construct $\mathcal{A}'$ accepting the marked language *l-ins-con-mark*$_{C',V'}(L)$ as follows. We have four copies of $\mathcal{A}$. The second and third copies have all transitions deleted from them, and the fourth copy has all C transitions deleted from it. The initial state of the first copy is the initial state of $\mathcal{A}'$ and the final states of the fourth copy are the final states of $\mathcal{A}'$. For every transition $p \xrightarrow{v'} q$ with $v' \in V'$, we add a c'-transition (for every $c' \in C'$) from p in the first copy to p in the second copy. We add a v'-transition from a state r in the second copy to a state t in the third copy iff $r \xrightarrow{v'} t$, with $v' \in V'$, is a transition in $\mathcal{A}$. Finally, we add a $\natural$-transition from each state u in the third copy to u in the fourth copy.

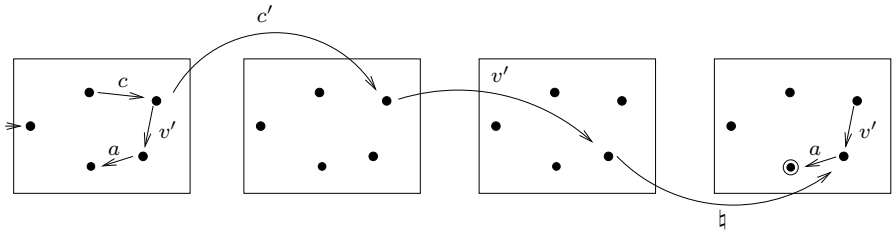

Figure 7: *l-ins-con-mark*$_{C',V'}(L)$

8. *l-ins-adm-con-mark*: Let L be a language over Σ with $L = L(\mathcal{A})$, and let $X \subseteq \Sigma$. Let $C' \subseteq C$ and $V' \subseteq V$. We construct $\mathcal{A}'$ for the language *l-ins-adm-con-mark*$^X_{C',V'}(L)$ as follows. We use four copies of $\mathcal{A}$. The first copy is exactly the same as in *l-ins-adm*$^X(L)$, where the states have two components, the first component keeping track of a state from $\mathcal{A}$, while the second keeps track of a *set* of states of $\mathcal{A}$ that are reachable by words that are X-equivalent to the current word being read. The second and third copies of $\mathcal{A}$ have all transitions deleted from them, and the fourth

copy has all C transitions deleted from it. The initial state of the first copy is the initial state of $\mathcal{A}'$ and the final states of the fourth copy are the final states of $\mathcal{A}'$. We have a transition labelled c', with $c' \in C'$, from a state (p, T) in the first copy to p in the second copy, provided T contains a state t from which it is possible to do a c' and go to a final state. We add a v'-transition from a state r in the second copy to a state u in the third copy iff $r \xrightarrow{v'} u$, with $v' \in V'$, is a transition in $\mathcal{A}$. Finally, we add a $\natural$-transition from each state w in the third copy to w in the fourth copy.

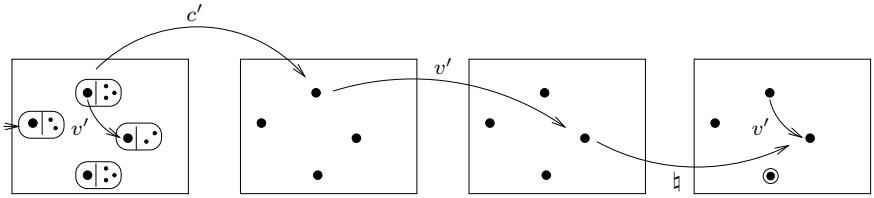

Figure 8: *l-ins-adm-con-mark*$^{X}_{C',V'}(L)$

9. *erase-con-mark*: Let $L \subseteq \Sigma^*$ and let $\mathcal{A}$ be an FSA with $L = L(\mathcal{A})$. Let $N' \subseteq N$ and $V' \subseteq V$. We construct $\mathcal{A}'$ accepting *erase-con-mark*$_{N',V'}(L)$ as follows. We have four copies of $\mathcal{A}$. The first and fourth copy have all their original transitions intact, the second has all transitions labeled with $a \notin N'$ deleted and transitions labelled n', with $n' \in N'$, replaced by ϵ-transitions; and the third has all its transitions deleted. We add an ϵ-transition from every state p in the first copy to p in the second copy; For every state p in the second copy such that $p \xrightarrow{v'} q$ in $\mathcal{A}$, we add a v'-transition from p in the second copy to q in the third copy. From every state r in the third copy we add a transition labelled $\natural$ to r in the fourth copy. The initial states of $\mathcal{A}'$ are the initial states of the first copy and the final states those of the fourth copy.

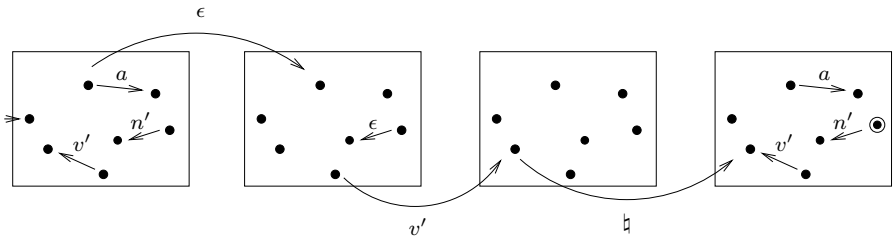

Figure 9: *erase-con-mark*$_{N',V'}(L)$

Appendix C

Now we prove the Lemmas 6 to 10. Each of the following context-free languages can be used to prove undecidability for a set of BSPs. We will give the proof only for one BSP out of each group. The other proofs follow analogously.

We use language L_M^I, to prove Lemma 6. Lemma 6 says:
L_M^I satisfies BSP I (similarly SI, BSI) w.r.t. a view $\mathcal{V} = (\Sigma, \emptyset, \{c\}) \Leftrightarrow L(M) = \emptyset$.

Proof ($\Leftarrow$:) Let us assume $L(M) = \emptyset$. Clearly, prefix closure of L_4 independently satisfies BSP I. Consider any string τ in L_3. τ is not a valid computation since $L(M) = \emptyset$. Note that τ interspersed with strings of c's is present in L_4. Also all the prefixes of τ interspersed with strings of c's are in L_M^I. Hence L_M^I satisfies I.

($\Rightarrow$:) If $L(M)$ is not empty then there exists a valid computation τ and it can be found in L_3. If we insert events from C at arbitrary points in τ, the resulting string will not be present in L_M^I. Hence I is not satisfied. Hence, $L(M)$ is empty when L_M^I satisfies BSP I. □

We can use a construction similar to the one of the L_M to show that this language is also accepted by a pushdown system. We note that $L_3 = A_1 \cap A_2 \cap \cap A_3 \cap A_4$ and thus a regular language. Let P be the PDS for L_2 with δ_P as its transition function and R be the finite automaton accepting $\{c\}^*$ with δ_R as its transition function. We can construct a PDS accepting L_4 with the transition function δ defined as:

$$\begin{aligned}\delta((q_1, q_2), a, A) = \quad & \{((q_1', q_2), B_1 \cdots B_k) \mid (q_1', B_1 \cdots B_k) \in \delta_P(q_1, a, A)\} \\ & \cup \{((q_1, \delta_R(q_2, a)), A)\}\end{aligned}$$

Hence L_M^I is accepted by a pushdown system. The rest of the languages used in the appendix also can be shown to be accepted by pushdown systems in a similar fashion.

We use language L_M^{IAX} to prove Lemma 7. Lemma 7 says:
L_M^{IAX} satisfies BSP IA(similarly SIA, $BSIA$) w.r.t. X and the view $\mathcal{V} = (\Sigma, \emptyset, \{c\}) \Leftrightarrow L(M) = \emptyset$.

Proof ($\Leftarrow$:) Let us assume $L(M) = \emptyset$. Clearly, prefix closure of $L_4 \cup L_5$ satisfies BSP IA^X independently. Consider any string of the form τ in L_3. The string τ is not a valid computation, since $L(M) = \emptyset$. Since $X^* \cdot \{c\}$ is a subset of L_M^{IAX}, we can insert c's at arbitrary points in τ and we will have the resulting string in L_4. Similarly for the prefixes of L_3, the corresponding strings which satisfies the condition will be present in L_4. Hence L_M^{IAX} satisfies IA^X.

($\Rightarrow$:) If $L(M)$ is not empty, there exists some string τ in L_3 and not in L_4 which is a valid computation. Since we have $X^* \cdot \{c\}$ as a subset of L_M^{IAX}, we can insert c's at arbitrary points in τ. So, the resulting string will not be present in L_M^{IAX}. Hence IA^X is not satisfied. Hence $L(M)$ is empty, when L_M^{IAX} satisfies IA^X. □

We use language L_M^{FCD} to prove Lemma 8. Lemma 8 says:
L_M^{FCD} satisfies BSP FCD w.r.t. V', N', C' and the view $\mathcal{V} = (\Sigma, \emptyset, \{c\}) \Leftrightarrow L(M) = \emptyset$.

Proof ($\Leftarrow$:) Let us assume $L(M) = \emptyset$. Consider any string containing confidential event in L_M^{FCD}. That string will be of the form $cv\tau$ in L_6. The string τ is not a valid computation since $L(M) = \emptyset$. Since $v \in V'$ and $c \in C'$, the string after deleting c is included in L_7. The strings corresponding to prefixes of the strings in L_8 will be found in prefixes of strings in L_9. Hence L_M^{FCD} satisfies FCD.

($\Rightarrow$:) If $L(M)$ is not empty, there exists some string τ which is a valid computation. L_6 contains $cv\tau$. The resulting string after deleting the last c is not included in the language L_M^{FCD}. Hence FCD is not satisfied. Hence $L(M)$ is empty when L_M^{FCD} satisfies FCD. □

We use language L_M^{FCI} to prove Lemma 9. Lemma 9 says:
L_M^{FCI} satisfies FCI w.r.t. V', N', C' and the view $\mathcal{V} = (\Sigma, \emptyset, \{c\}) \Leftrightarrow L(M) = \emptyset$.

Proof ($\Leftarrow$:) Let us assume $L(M) = \emptyset$. Clearly, prefix closure of L_9 independently satisfies FCI. Consider any string of the form $v\tau$ in L_8. The string τ is not a valid computation since $L(M) = \emptyset$. The string after inserting an event from C' preceding v is included in L_9. For any prefix of the string in L_8, the corresponding string will be the prefix of a string in L_9. Hence L_M^{FCI} satisfies FCI.

($\Rightarrow$:) If $L(M)$ is not empty, there exists some string τ in L_8 and not in L_9 which is a valid computation. The resulting string after inserting c preceding v is not included in L_M^{FCI}.Hence FCI is not satisfied. Hence $L(M)$ is empty when L_M^{FCI} satisfies FCI. □

We use language L_M^{FCIAX} to prove Lemma 10. Lemma 10 says:
L_M^{FCIAX} satisfies BSP $FCIA$ w.r.t. X, V', N', C' and the view $\mathcal{V} = (\Sigma, \emptyset, \{c\}) \Leftrightarrow L(M) = \emptyset$.

Proof ($\Leftarrow$:) Let us assume $L(M) = \emptyset$. Clearly, prefix closure of $L_9 \cup L_{10}$ independently satisfies $FCIA^X$. Consider any string of the form $v\tau$ in L_8. The string τ is not a valid computation since $L(M) = \emptyset$. Since $v \in V'$ and $X^* \cdot C'$ is contained in L_M^{FCIAX}, the string after inserting an event from C' preceding v, is included in L_9. For any prefix of the string in L_8, the corresponding string which satisfies the condition is included in the prefix of a string from L_9. Hence L_M^{FCIAX} satisfies $FCIA^X$.

($\Rightarrow$:) If $L(M)$ is not empty, there exists some string τ in L_8 and not in L_9 which is a valid computation. The resulting string after inserting a c from C' preceding v is not included in L_M^{FCIAX}. Hence $FCIA^X$ is not satisfied. Hence $L(M)$ is empty when L_M^{FCIAX} satisfies $FCIA^X$. □