Model-Checking HyperLTL for Pushdown Systems

Adrien Pommelet, Tayssir Touili

Model-Checking HyperLTL for Pushdown Systems

Adrien Pommellet[1] and Tayssir Touili[2]

[1] LIPN and Université Paris-Diderot, France
[2] LIPN, CNRS, and Université Paris 13, France

Abstract. Temporal logics such as LTL are often used to express safety or correctness properties of programs. However, they cannot model complex formulas known as hyperproperties introducing relations between different execution paths of a same system. In order to do so, the logic *HyperLTL* adds existential and universal quantifications of path variables to LTL. The model-checking problem, that is, determining if a given representation of a program verifies a HyperLTL property, has been shown to be decidable for finite state systems. In this paper, we prove that this result does not hold for Pushdown Systems nor for the subclass of Visibly Pushdown Systems. We therefore introduce an algorithm that over-approximates the model-checking problem with an automata-theoretic approach. We also detail an under-approximation method based on a phase-bounded analysis of Multi-Stack Pushdown Systems. We then show how these approximations can be used to check security policies.

1 Introduction

The analysis of execution traces of programs can be used to prove correctness properties often expressed with the unifying framework of the *linear temporal logic* LTL. However, a LTL formula only quantifies a single execution trace of a system; LTL can't express properties on multiple, simultaneous executions of a program.

These properties on sets of execution traces are known as *hyperproperties*. Many safety and security policies can be expressed as hyperproperties; this is in particular true of information-flow analysis. As an example, the *non-interference* policy states that if two computations share the same public inputs, they should have identical public outputs as well, even if their private inputs differ. This property implies a relation between computations that can't be expressed as a simple LTL formula.

HyperLTL is an extension of LTL introduced by Clarkson et al. in [5] that allows the universal and existential quantifications of multiple *path variables* that range over traces of a system in order to define hyperproperties. As an example, the formula $\forall \pi_1, \forall \pi_2, (a_{\pi_1} \wedge a_{\pi_2}) \Rightarrow X\left((b_{\pi_1} \wedge b_{\pi_2}) \vee (c_{\pi_1} \wedge c_{\pi_2})\right)$ means that, given two path variables π_1 and π_2 in the set $Traces_\omega(S)$ of infinite traces of a system S, if π_1 and π_2 verify the same atomic property a at a given step, then they should both verify either b or c at the next step.

Clarkson et al. have shown that the model-checking problem $S \models \psi$ of HyperLTL, that is, knowing if the set of traces of a system S verifies the HyperLTL formula ψ, can be solved when S is a finite state transition system (i.e. equivalent to a finite state automaton). However, simple transition models cannot accurately model programs with infinite recursion and procedure calls. *Pushdown Systems* (PDSs) that can simulate the *call stack* of a program are commonly used instead. The call stack stores information about the active procedures of a program such as return addresses, passed parameters and local variables.

Unfortunately, we show in this paper that the model-checking problem of HyperLTL for PDSs is undecidable: the set of traces of a PDS is a context-free language, and deciding whether the intersection of two context-free languages is empty or not remains an undecidable problem that can be reduced to the model-checking problem by using a HyperLTL formula that synchronizes traces.

On the other hand, determining the emptiness of the intersection of two *visibly context-free languages* is decidable. This class of languages is generated by *Visibly Pushdown Automata* (VPDA), an input-driven subclass of *pushdown automata* (PDA) first introduced by Alur et al. in [1]: at each step of a computation, the next stack operation will be determined by the input letter in Σ read, depending on a partition of the input alphabet. We study the model-checking problem of HyperLTL for *Visibly Pushdown Systems* (VPDSs), and prove that it is also undecidable, as it happens to be a reduction of the emptiness problem for *Two-Stack Visibly Pushdown Automata* (2-VPDA), which has been shown to be undecidable by Carotenuto et al. in [4].

To overcome these undecidability issues, since the emptiness of the intersection of a context-free langage with regular sets is decidable, one idea is to consider the case where only one path variable of the formula ranges over the set of traces $Traces_\omega(\mathcal{P})$ of a PDS or VPDS $\mathcal{P}$, while the other variables range over a regular abstraction $\alpha(Traces_\omega(\mathcal{P}))$. Using an automata-theoretic approach, this idea allows us to over-approximate the model-checking problem of HyperLTL formulas that only use universal quantifiers $\forall$ with the exception of at most one path variable: if the HyperLTL formula holds for the over-approximation, it holds for the actual system as well.

On the other hand, under-approximations can be used to discover errors in programs: if a HyperLTL formula does not hold for an under-approximation of the model-checking problem, it does not hold for the actual system as well. We show that the model-checking problem for PDSs of HyperLTL formulas that only use universal quantifiers $\forall$ can be under-approximated by relying on a bounded-phase model-checking of a LTL formula for a *Multi-Stack Pushdown System* (MPDS), where a *phase* is a part of a run during which there is at most one stack that is popped from, as defined by Torre et al. in [13].

Related work. Clarkson and Schneider introduced *hyperproperties* in [6] to formalize security properties, using second-order logic. Unfortunately, this logic isn't verifiable in the general case.

However, fragments of it can be verified: in [5], Clarkson et al. formalized the temporal logics HyperLTL and HyperCTL*, extending the widespread and

flexible framework of linear time and branching time logics to hyperproperties. The model-checking problem of these logics for finite state systems has been shown to be decidable by a reduction to the satisfiability problem for the quantified propositional temporal logic QPTL defined in [12].

Proper model-checking algorithms were then introduced by Finkbeiner et al. in [8]. These algorithms follow the automata-theoretic framework defined by Vardi et al. in [14], and can be used to verify security policies in circuits. However, while circuits can be modelled as finite state systems, actual programs can feature recursive procedure calls and infinite recursion. Hence, a more expressive model such as PDSs is needed.

In [3,7], the forward and backward reachability sets of PDSs have been shown to be regular and effectively computable. As a consequence, the model-checking problem of LTL for PDSs is decidable; an answer can be effectively computed using an automata-theoretic approach. We try to extend this result to HyperLTL.

Multi-Stack Pushdown Systems (MPDSs) are unfortunately Turing powerful. Following the work of Qadeer et al. in [10], La Torre et al. introduced in [13] MPDSs with *bounded phases*: a run is split into a finite number of phases during which there is at most one stack that is popped from. Anil Seth later proved in [11] that the backward reachability set of a multi-stack pushdown system with bounded phases is regular; this result can then be used to solve the model-checking problem of LTL for MPDSs with bounded phases. We rely on a phase-bounded analysis of a MPDS to under-approximate an answer to the model-checking problem of HyperLTL for PDSs.

Paper outline. In Section 2 of this paper, we provide background on *Pushdown Systems* (PDSs) and *Visibly Pushdown Systems* (VPDSs). We define in Section 3 the hyper linear time logic *HyperLTL*, and prove that its model-checking problem for PDSs and VPDSs is undecidable. Then, in Section 4, we solve the model-checking problem of HyperLTL on constrained sets of traces then find an over-approximation of the model-checking problem for PDSs. In Section 5, we use *Multi-Stack Pushdown Systems* (MPDSs) and bounded phase analysis to under-approximate the model-checking problem. Finally, in section 6, we apply the logic HyperLTL to express security properties. Due to a lack of space, detailed proofs of some theorems can be found in the appendix.

2 Pushdown Systems

2.1 The Model

Pushdown systems are a natural model for sequential programs with recursive procedure calls [7].

Definition 1 (Pushdown System). *A* Pushdown System *(PDS) is a tuple* $\mathcal{P} = (P, \Sigma, \Gamma, \Delta, c_0)$ *where* P *is a finite set of control states,* Σ *a finite input alphabet,* Γ *a finite stack alphabet,* $\Delta \subseteq P \times \Gamma \times \Sigma \times P \times \Gamma^*$ *a finite set of transition rules, and* $c_0 \in P \times \Gamma^*$ *an initial configuration.*

If $d = (p, \gamma, a, p', w) \in \Delta$, we write $d = (p, \gamma) \xrightarrow{a} (p', w)$. We call a the *label* of Σ. We can assume without loss of generality that $\Delta \subseteq P \times \Gamma \times \Sigma \times P \times \Gamma^{\leq 2}$ and that c_0 is of the form $\langle p_0, \bot \rangle$, where $\bot \in \Gamma$ is a special bottom stack symbol shared by every PDS on the stack alphabet Γ and $p_0 \in P$. A *configuration* of $\mathcal{P}$ is a pair $\langle p, w \rangle$ where $p \in P$ is a control state and $w \in \Gamma^*$ a stack content.

For each $a \in \Sigma$, we define a transition relation $\xrightarrow{a}_{\mathcal{P}}$ on configurations as follows: if $(p, \gamma) \xrightarrow{a} (p', w) \in \Delta$, for each $w' \in \Gamma^*$, $\langle p, \gamma w' \rangle \xrightarrow{a}_{\mathcal{P}} \langle p', w w' \rangle$. We then consider the immediate successor relation $\rightarrow_{\mathcal{P}} = \bigcup_{a \in \Sigma} \xrightarrow{a}_{\mathcal{P}}$. We may omit the variable $\mathcal{P}$ when only a single PDS is being considered.

A *run* r is a sequence of configurations $r = (c_i)_{i \geq 0}$ such that $\forall i \geq 0$, $c_i \xrightarrow{a_i}_{\mathcal{P}} c_{i+1}$, c_0 being the initial configuration of $\mathcal{P}$. The word $(a_i)_{i \geq 0}$ is then said to be the *trace* of r. Traces and runs may be finite or infinite. Let $Traces_\omega(\mathcal{P})$ (resp. $Traces(\mathcal{P})$) be the set of all infinite (resp. finite) traces of $\mathcal{P}$.

A *Büchi Pushdown Automaton* (BPDA) is a pair $\mathcal{BP} = (\mathcal{P}, F)$, where $\mathcal{P} = (P, \Sigma, \Gamma, \Delta, c_0)$ is a PDS and $F \subseteq P$ a set of final states. An infinite run $r = (c_i)_{i \geq 0}$ of $\mathcal{BP}$ and its matching trace $(a_i)_{i \geq 0}$ are said to be *accepting* if there exists at least one infinitely often occurring state f in r such that $f \in F$. The language $L_\omega(\mathcal{BP})$ accepted by $\mathcal{BP}$ is the set of all accepting traces of $\mathcal{BP}$, and is said to be ω *context-free*.

2.2 Visibly Pushdown Systems

We consider a particular subclass of PDSs introduced by Alur et al. in [1]. Let $\langle \Sigma_c, \Sigma_r, \Sigma_l \rangle$ be a partition of the input alphabet, where Σ_c, Σ_r, and Σ_l stand respectively for the *call*, *return*, and *local* alphabets.

Definition 2 (Visibly Pushdown System). *A* Visibly Pushdown System *(VPDS) over a partition $\langle \Sigma_c, \Sigma_r, \Sigma_l \rangle$ of Σ is a PDS $\mathcal{P} = (P, \Sigma, \Gamma, \Delta, c_0)$ verifying the following properties:*

- *if $(p, \gamma_1) \xrightarrow{a} (p', \gamma_2) \in \Delta$, then $a \in \Sigma_l$, $\gamma_1 = \gamma_2$, and $\forall \gamma \in \Gamma$, $(p, \gamma) \xrightarrow{a} (p', \gamma) \in \Delta$;*
- *if $(p, \gamma) \xrightarrow{a} (p', \varepsilon) \in \Delta$, then $a \in \Sigma_r$;*
- *if $(p, \gamma_1) \xrightarrow{a} (p', \gamma_2 \gamma_1) \in \Delta$, then $a \in \Sigma_c$, and $\forall \gamma \in \Gamma$, $(p, \gamma) \xrightarrow{a} (p', \gamma_2 \gamma) \in \Delta$;*

VPDSs are an *input driven* subclass of PDSs: at each step of a computation, the next stack operation will be determined by the input letter in Σ read, depending on which subset of the partition $\langle \Sigma_c, \Sigma_r, \Sigma_l \rangle$ the aforementioned letter belongs to.

Visibly Pushdown Automata accept the class of *visibly pushdown languages*. If a BPDA $\mathcal{BP}$ is visibly pushdown according to a partition of Σ, we say it's a *Büchi Visibly Pushdown Automata* (BVPDA). The class of languages accepted by BVPDA is called ω *visibly pushdown languages*.

Unlike context-free languages, the emptiness of the intersection of visibly pushdown languages is a decidable problem and the complement of a visibly pushdown language is a visibly pushdown language that can be computed. The same properties also hold for ω visibly pushdown languages.

3 HyperLTL

3.1 The Logic

Let AP be a finite set of atomic propositions used to express facts about a program; a *path* is an infinite word in $(2^{AP})^\omega = \mathcal{T}$. Let $\mathcal{V}$ be a finite set of *path variables*. The *HyperLTL* logic relates multiple paths by introducing path quantifiers.

Definition 3 (Syntax of HyperLTL). Unquantified *HyperLTL formulas are defined according to the following syntax equation:*

$$\varphi ::= \bot \mid (a, \pi) \in AP \times \mathcal{V} \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \mathrm{X}\ \varphi \mid \varphi\ \mathrm{U}\ \varphi \mid\ \mathrm{G}\ \varphi \mid\ \mathrm{F}\ \varphi$$

From then on, we write $a_\pi = (a, \pi)$. HyperLTL formulas are defined according to the following syntax equation:

$$\psi ::= \exists \pi, \varphi \mid \forall \pi, \varphi \mid \varphi$$

where $\pi \in \mathcal{V}$ is a path variable.

The existential $\exists$ and universal quantifiers $\forall$ are used to define path variables, to which atomic propositions in AP are bound. A HyperLTL formula is said to be *closed* if there is no free variable: each path variable is bound by a path quantifier once.

As an example, the closed formula $\forall \pi_1, \exists \pi_2, \varphi$ means that for all paths π_1, there exists a path π_2 such that the formula φ holds for π_1 and π_2. Simple LTL formulas can be considered as a subclass of closed HyperLTL formulas of the form $\forall \pi, \varphi$ with a single path variable.

Let $\Pi : \mathcal{V} \to \mathcal{T}$ be a *path assignment function* of $\mathcal{V}$ that matches to each path variable π a path $\Pi(\pi) \in \mathcal{T}$. If $\Pi(\pi) = (t_j)_{j \geq 0}$, for all $i \geq 0$, we define the i-th value of the path $\Pi(\pi)[i] = t_i$ and a suffix assignment function $\Pi[i, \infty]$ such that $\Pi[i, \infty](\pi) = (t_j)_{j \geq i}$.

We first define the semantics of this logic for path assignment functions.

Definition 4 (Semantics of unquantified HyperLTL formulas). *Let φ be an unquantified HyperLTL formula. We define by induction on φ the following semantics on path assignment functions:*

$$\begin{aligned}
\Pi &\models a_\pi \Leftrightarrow a \in \Pi(\pi)[0] \\
\Pi &\models \neg\varphi \Leftrightarrow \Pi \not\models \neg\varphi \\
\Pi &\models \varphi_1 \vee \varphi_2 \Leftrightarrow (\Pi \models \varphi_1) \vee (\Pi \models \varphi_2) \\
\Pi &\models \varphi_1 \wedge \varphi_2 \Leftrightarrow (\Pi \models \varphi_1) \wedge (\Pi \models \varphi_2) \\
\Pi &\models \mathrm{X}\ \varphi \Leftrightarrow \Pi[1, \infty] \models \varphi \\
\Pi &\models \varphi\ \mathrm{U}\ \psi \Leftrightarrow \exists j \geq 0, \Pi[j, \infty] \models \psi \text{ and } \forall i \in \{0, \ldots, j-1\}, \Pi[i, \infty] \models \varphi \\
\Pi &\models\ \mathrm{G}\ \varphi \Leftrightarrow \forall i \geq 0, \Pi, i \models \varphi \\
\Pi &\models\ \mathrm{F}\ \varphi \Leftrightarrow \exists i \geq 0, \Pi, i \models \varphi
\end{aligned}$$

$\Pi \models \varphi$ if φ holds for a given assignment of path variables defined according to Π.

Let $T : \mathcal{V} \to 2^{\mathcal{T}}$ be a *set assignment function* of $\mathcal{V}$ that matches to each path variable $\pi \in \mathcal{V}$ a set of paths $T(\pi) \subseteq \mathcal{T}$. We can now define the semantics of closed HyperLTL formulas for set assignment functions.

Definition 5 (Semantics of closed HyperLTL formulas). *We consider a closed HyperLTL formula $\psi = \chi_0\pi_0, \ldots, \chi_n\pi_n, \varphi$, where each $\chi_i \in \{\forall, \exists\}$ is an universal or existential quantifier, and φ an unquantified HyperLTL formula using trace variables $\pi_0, \ldots, \pi_n$.*

For a given set assignment function T, we write that $T \models \psi$ if for $\chi_0 t_0 \in T(\pi_0)$, ..., $\chi_n t_n \in T(\pi_n)$, we have $\Pi \models \varphi$, where Π is the path assignment function such that $\forall i \in \{0, \ldots, n\}$, $\Pi(\pi_i) = t_i$.

As an example, if $\psi = \forall\pi_1, \exists\pi_2, \varphi$ is a closed HyperLTL formula and T is a set assignment function of $\mathcal{V}$, then $T \models \psi$ if $\forall t_1 \in T(\pi_1)$, $\exists t_2 \in T(\pi_2)$ such that $\Pi \models \varphi$, where $\Pi(\pi_1) = t_2$ and $\Pi(\pi_2) = t_2$. Intuitively, $T \models \psi$ if, assuming path variables belong to path sets defined according to T, the closed formula ψ holds. From then on, we assume that *every HyperLTL formula considered in this paper is closed.*

3.2 HyperLTL and PDSs

Let $\mathcal{P}$ be a PDS on the input alphabet $\Sigma = 2^{AP}$ and ψ a closed HyperLTL formula. We write that $\mathcal{P} \models \psi$ if and only if $T \models \psi$ where the set assignment function T is such that $\forall \pi \in \mathcal{V}, T(\pi) = Traces_\omega(\mathcal{P})$. Determining whether $\mathcal{P} \models \psi$ for a given PDS $\mathcal{P}$ and a given HyperLTL formula ψ is called the *model-checking* problem of HyperLTL on PDSs. The following theorem holds :

Theorem 1. *The model-checking problem of HyperLTL for PDSs is undecidable.*

We can prove this result by reducing the emptiness of the intersection of two context-free languages, a well-known undecidable problem, to the model-checking problem. Our intuition is to consider two context-free languages $\mathcal{L}_1$ and $\mathcal{L}_2$ on the alphabet Σ. As HyperLTL formulas apply to infinite words, we define two BPDA $\mathcal{BP}_1$ and $\mathcal{BP}_2$ that accept $\mathcal{L}_1 f^\omega$ and $\mathcal{L}_2 f^\omega$ respectively, where $f \notin \Sigma$ is a special ending symbol. We then define a PDS $\mathcal{P}$ that can simulate either $\mathcal{BP}_1$ or $\mathcal{BP}_2$.

We now introduce the formula $\psi = \exists\pi_1, \exists\pi_2, \varphi_{start} \wedge \varphi_{sync} \wedge \varphi_{end}$: φ_{start} expresses that trace variables π_1 and π_2 represent runs of $\mathcal{BP}_1$ and $\mathcal{BP}_2$ respectively, φ_{sync} means that the two traces are equal from their second letter onwards, and φ_{end} implies that the two traces are accepting. Hence, if $\mathcal{P} \models \psi$, then $\mathcal{BP}_1$ and $\mathcal{BP}_2$ share a common accepting run, and $\mathcal{L}_1 \cap \mathcal{L}_2 \neq \emptyset$.

On the other hand, if $\mathcal{L}_1 \cap \mathcal{L}_2 \neq \emptyset$, there is an accepting trace π common to $\mathcal{BP}_1$ and $\mathcal{BP}_2$ and we can define two traces π_1 and π_2 of $\mathcal{P}$ such that the formula $\varphi_{start} \wedge \varphi_{sync} \wedge \varphi_{end}$ holds. Since the emptiness problem is undecidable, so must be the model-checking problem.

The full proof is given in the appendix. As a consequence of Theorem 1, determining whether $T \models \psi$ for a generic set assignment function T and a given HyperLTL formula ψ is an undecidable problem.

3.3 HyperLTL and VPDSs

Since the emptiness of the intersection of visibly pushdown languages is decidable, the previous proof does not apply to VPDSs and one might wonder if the model-checking problem of HyperLTL for this particular subclass is decidable. Unfortunately, we can show that this is not the case:

Theorem 2. *The model-checking problem of HyperLTL for VPDSs is undecidable.*

In order to prove this theorem, we will rely on a class of two-stack automata called *2-Visibly Pushdown Automata* (2-VPDA) introduced in [4]. In a 2-VPDA, each stack is input driven, but follows its own partition of Σ. The same input letter may result in different pushdown rules being applied to the first and second stack: as an example, a transition can push a word on the first stack and pop the top letter of the second stack, depending on which partition is used by each stack. Moreover, in a manner similar to VPDA, transitions of 2-VPDA do not depend on the top stack symbols unless they pop them.

It has been shown in [4] that the emptiness problem is undecidable for 2-VPDA. Our intuition is therefore to prove Theorem 2 by reducing the emptiness problem for 2-VPDA to the model-checking problem of HyperLTL for VPDSs. To do so, for a given 2-VPDA $\mathcal{D}$, we define a VPDS $\mathcal{P}$ and a HyperLTL formula ψ on two trace variables such that $\mathcal{P} \models \psi$ if and only if $\mathcal{D}$ has an accepting run.

$\mathcal{P}$ is such that it can simulate either stack of the 2-VPDA. However, both stacks must be synchronized in order to properly represent the whole automaton: the content of one stack can lead to a control state switch that may enable a transition modifying the other stack. The HyperLTL formula ψ determines which trace variable is related to which stack, synchronizes two runs of $\mathcal{P}$ in such a manner that they can be used to define an execution path of $\mathcal{D}$, and ensure that this path is an accepting one. The full proof can be found in the appendix.

4 Model-checking HyperLTL with Constraints

Theorem 1 proves that the model-checking problem of HyperLTL for PDSs is undecidable. Intuitively, this issue stems from the undecidability of the intersection of context-free languages. However, since the emptiness problem of the intersection of a context-free language with regular sets is decidable, one can think of a way to abstract the set of runs of a PDS for some - but not all - path variables of a HyperLTL formula as a mean of regaining decidability.

As shown in [2,9], runs of a PDS can be over-approximated in a regular fashion. Hence, for a given PDS $\mathcal{P}$, if we consider a regular abstraction of the set of runs

$\alpha(Traces_\omega(\mathcal{P}))$, we can change the set assignment function for a path variable π in such a manner that $T(\pi) = \alpha(Traces_\omega(\mathcal{P}))$ instead of $T(\pi) = Traces_\omega(\mathcal{P})$.

For a set assignment function T on a set of path variables $\mathcal{V}$ and a variable $\pi \in \mathcal{V}$, we say that π is context-free w.r.t. to T if $T(\pi) = Traces_\omega(\mathcal{P})$ for a PDS $\mathcal{P}$. We define regular and visibly pushdown variables in a similar manner.

Let $\psi = \chi_0\pi_0, \ldots, \chi_n\pi_n, \varphi$ be a closed HyperLTL formula on the alphabet AP with $n+1$ trace variables $\pi_0, \ldots, \pi_n$, where $\chi_0 \ldots, \chi_n \in \{\forall, \exists\}$. In this section, we will present a procedure to determine whether $T \models \psi$ in two cases.

1. If the variable π_0 is context-free w.r.t. T, and all the other variables are regular, then we can determine whether $T \models \psi$ or not. We can then apply this technique in order to over-approximate the model-checking problem if $T(\pi_0) = Traces_\omega(\mathcal{P})$, $T(\pi_j) = \alpha(Traces_\omega(\mathcal{P}))$ for $j = 1 \ldots n$, and $\chi_1, \ldots, \chi_n = \forall$. The last n variables can only be universally quantified.

   $T \models \psi$ then implies that $\mathcal{P} \models \psi$: indeed, the universal quantifiers on the path variables that range over the abstracted traces are such that, if the formula φ holds for every run in the over-approximation, then it also holds for every run in the actual set of traces. This is an over-approximation of the actual model-checking problem.
2. If there exists a variable π_i such that π_i is visibly context-free w.r.t. T, and all the other variables are regular, then we can determine whether $T \models \psi$ or not. A single path variable at most can be visibly context-free (not necessarily π_0, though), and all the others must be regular. We can then apply this technique in order to over-approximate the model-checking problem if $\mathcal{P}$ is a VPDS, $T(\pi_i) = Traces_\omega(\mathcal{P})$, $T(\pi_j) = \alpha(Traces_\omega(\mathcal{P}))$ and $\chi_j = \forall$ for $j \neq i$. Each path variable with the exception of the visibly context-free one must be universally quantified.

   Because of the universal quantifiers on the regular path variables, $T \models \psi$ implies again that $\mathcal{P} \models \psi$. This is another over-approximation of the model-checking problem.

Moreover, these over-approximations are accurate for at least one variable in the trace variable set, as the original, ω context-free (or ω visibly pushdown) set of runs is assigned to this variable instead of an ω regular over-approximation.

4.1 With One Context-Free Variable and n Regular Variables

Let $\mathcal{P}$ be a PDS such that $T(\pi_0) = Traces_\omega(\mathcal{P})$, and $\mathcal{K}_1, \ldots, \mathcal{K}_n$, finite state transition systems (i.e. finite automata without final states) such that for $i = 1, \ldots, n$, $T(\pi_i) = Traces_\omega(\mathcal{K}_i)$.

Theorem 3. *If π_0 is context-free w.r.t. T and the other variables are regular, we can decide whether $T \models \chi_0\pi_0, \ldots, \chi_n\pi_n, \varphi$ or not.*

To do so, we use the following well-known result:

Lemma 1. *Let φ be an LTL formula. There exists a Büchi automaton $\mathcal{B}_\varphi$ on the alphabet 2^{AP} such that $L(\mathcal{B}_\varphi) = \{w \in (2^{AP})^\omega \mid w \models \varphi\}$. We say that $\mathcal{B}_\varphi$ accepts φ.*

An *unquantified* HyperLTL formula with m trace variables $\pi_1, \ldots, \pi_m$ can be considered as a LTL formula on the alphabet $(2^{AP})^m$: given a word w on $(2^{AP})^m$ and $a \in AP$, we say that $w \models a_{\pi_i}$ if $a \in w_i(0)$, where w_i is the i-th component of w. We then apply Lemma 1 and introduce a Büchi automaton $\mathcal{B}_\varphi$ on the alphabet $(2^{AP})^{n+1}$ accepting φ. We denote $\Sigma = 2^{AP}$.

We then compute inductively a sequence of Büchi automata $\mathcal{B}_{n+1}, \ldots, \mathcal{B}_1$ such that:

- $\mathcal{B}_{n+1}$ is equal to the Büchi automaton $\mathcal{B}_\varphi$ on the alphabet Σ^{n+1} ;
- if the quantifier χ_i is equal to $\exists$ and $\mathcal{B}_{i+1} = (Q, \Sigma^{i+1}, \delta, q_0, F)$ is a Büchi automaton on the alphabet Σ^{i+1}, let $\mathcal{K}_i = (S, \Sigma, \delta', s_0)$ be the finite state transition system generating $T(\pi_i)$; we now define the Büchi automaton $\mathcal{B}_i = (Q \times S, \Sigma^i, \rho, (q_0, s_0), F \times S)$ where the set ρ of transitions is such that if $q \xrightarrow{(a_0,\ldots,a_i)} q' \in \delta$ and $s \xrightarrow{a_i} s' \in \delta'$, then $(q, s) \xrightarrow{(a_0,\ldots,a_{i-1})} (q', s') \in \rho$. Intuitively, the Büchi automaton $\mathcal{B}_i$ represents the formula $\exists \pi_i, \chi_{i+1}\pi_{i+1}, \ldots, \chi_n \pi_n, \varphi$; its input alphabet Σ^i depends on the number of variables that are not quantified yet;
- if the quantifier χ_i is equal to $\forall$, we consider instead the complement $\mathcal{B}'_{i+1}$ of $\mathcal{B}_{i+1}$ and compute its product with $\mathcal{K}_i$ in a similar manner to the previous construction; $\mathcal{B}_i$ is then equal to the complement of this product; intuitively, $\forall \pi, \psi = \neg(\exists \pi, \neg\psi)$.

Having computed $\mathcal{B}_1 = (Q, \Sigma, \delta, q_0, F)$, let $\mathcal{P} = (P, \Sigma, \Gamma, \Delta, \langle p_0, \bot\rangle)$ be the PDS generating $T(\pi_0)$. We assume that $\chi_0 = \exists$. Let $\mathcal{BP} = (P \times Q, \Sigma, \Delta', \langle (p_0, q_0), \bot\rangle, P \times F)$ be a Büchi pushdown automaton, where the set of transitions Δ' is such that if $q \xrightarrow{a} q' \in \delta$ and $(p, \gamma) \xrightarrow{a} (p', w) \in \Delta$, then $((p, q), \gamma) \xrightarrow{a} ((p', q'), w) \in \Delta'$. $\mathcal{BP}$ represents the fully quantified formula $\exists \pi_0, \chi_1\pi_1, \ldots, \chi_n\pi_n, \varphi$. Obviously, $\mathcal{B}$ is not empty if and only if $T \models \psi$.

If $\chi_0 = \forall$, we consider instead the complement $\mathcal{B}'_1$ of $\mathcal{B}_1$, then define a Büchi pushdown automaton $\mathcal{BP}$ in a similar manner. $\mathcal{B}$ is empty if and only if $T \models \psi$.

It has been proven in [3, 7] that the emptiness problem is decidable for Büchi pushdown automata. Hence, given our initial constraints on T and ψ, we can determine whether $T \models \psi$ or not. □

The Büchi automaton $\mathcal{B}_\varphi$ has $O(2^{|\varphi|})$ states; if we assume that all variables are existentially quantified, the BPDS $\mathcal{BP}$ has $\nu = O(2^{|\varphi|}|\mathcal{P}||\mathcal{K}_1|\ldots|\mathcal{K}_n|)$ states. According to [7], checking the emptiness of $\mathcal{BP}$ can be done in $O(\nu^2 k)$ operations, where k is the number of transitions of $\mathcal{BP}$, hence, in $O(\nu^4|\Gamma|^2)$.

Complementation of a Büchi Automaton may increase its size exponentially, hence, this technique may incur an exponential blow-up depending on the number of universal quantifiers.

Application. If we consider that π_0 range over $Traces_\omega(\mathcal{P})$ and that $\pi_1, \ldots, \pi_n$ range over a regular abstraction $\alpha(Traces_\omega(\mathcal{P}))$ of the actual set of traces, and

we assume that $\chi_1, \ldots, \chi_n = \forall$, we can apply this result to over-approximate the model-checking problem, as detailed earlier in this section.

It is worth noting that the complement of an ω context-free language is not necessarily an ω context-free language. Hence, we can't use the previous procedure to check a HyperLTL formula of the form $\psi = \exists\pi, \forall\pi'\varphi$ where π' is a context-free variable and π is regular. We know, however, that ω visibly pushdown languages are closed under complementation. We therefore consider the case of a single visibly pushdown variable in the following subsection.

4.2 With One Visibly Pushdown Variable and n Regular Variables

Let $\mathcal{P}$ be a VPDS such that $T(\pi_i) = Traces_\omega(\mathcal{P})$, and $(\mathcal{K}_j)_{j\neq i}$ finite state transition systems such that for $j \neq i$, $T(\pi_j) = Traces_\omega(\mathcal{K}_j)$. Unlike the previous case, the visibly context-free variable no longer has to be the first one π_0.

Theorem 4. *If a variable π_i is visibly pushdown w.r.t. T and the other variables are regular, we can decide whether $T \models \chi_0\pi_0, \ldots, \chi_n\pi_n, \varphi$ or not.*

The proof of this theorem is similar to the proof of Theorem 3. We first build a sequence of Büchi automata $\mathcal{B}_{n+1}, \ldots, \mathcal{B}_{i+1}$ in a similar manner to the proof of Theorem 3, starting from a finite state automaton $\mathcal{B}_{n+1} = \mathcal{B}_\varphi$ on the alphabet Σ^{n+1} representing the unquantified formula φ then computing products with the transition systems $\mathcal{K}_{n+1}, \ldots, \mathcal{K}_{i+1}$ until we end up with a Büchi automaton $\mathcal{B}_{i+1}$ on the alphabet Σ^{i+1}.

Having computed $\mathcal{B}_{i+1} = (Q, \Sigma^{i+1}, \delta, q_0, F)$, let $\mathcal{P} = (P, \Sigma, \Gamma, \Delta,$
$\langle p_0, \bot\rangle)$ be the VPDS generating $T(\pi_i)$. We assume that $\chi_i = \exists$. Let $\mathcal{BP}_i = (P \times Q, \Sigma^{i+1}, \Delta', \langle(p_0, q_0),$
$\bot\rangle, P \times F)$ be a visibly Büchi pushdown automaton, where Δ' is such that if $q \xrightarrow{(a_0,\ldots,a_{i-1},a)} q' \in \delta$ and $(p, \gamma) \xrightarrow{a} (p', w) \in \Delta$, then $((p, q), \gamma) \xrightarrow{(a_0,\ldots,a_{i-1},a)} ((p', q'), w) \in \Delta'$. $\mathcal{BP}_i$ is indeed a BVPDA on the alphabet Σ^{i+1} as its stack operations only depend on its $i+1$-th variable. If $\chi_i = \forall$, we consider instead the complement $\mathcal{B}'_{i+1}$.

From the i-th variable onwards, we compute a sequence of visibly Büchi pushdown automata $\mathcal{BP}_i, \ldots, \mathcal{BP}_0$ on the alphabets $\Sigma^{i+1}, \ldots, \Sigma^i$ respectively. For $i \geq k \geq 1$, if $\mathcal{BP}_k = (P', \Sigma^{k+1}, \Delta', \langle p'_0, \bot\rangle, F')$, $\mathcal{K}_i = (S, \Sigma, \delta, s_0)$, and $\chi_k = \exists$, let $\mathcal{BP}_{k-1} = (P' \times S, \Sigma^k, \Delta'', \langle(p'_0, s_0), \bot\rangle, F'' \times S)$ be a visibly Büchi pushdown automaton, where the set of transitions Δ'' is such that if $(p, \gamma) \xrightarrow{(a_0,\ldots,a_{k-1},a_i)} (p', w) \in \Delta'$ and $q \xrightarrow{a_{k-1}} q' \in \delta$, then $((p, q), \gamma) \xrightarrow{(a_0,\ldots,a_{k-2},a_i)} ((p', q'), w) \in \Delta''$. The last letter of each tuple always stands for the visibly pushdown path variable π_i: $\mathcal{BP}_{k-1}$ is visibly pushdown as its stack operations only depend on this variable. If $\chi_k = \forall$, we consider the complement $\mathcal{BP}'_k$ of $\mathcal{BP}_k$ instead, which is a visibly pushdown automaton as well.

We can check the emptiness of $\mathcal{BP}_0$. If it is indeed empty, then $T \models \psi$. □

It has been proven in [1] that the complement of a VPDA incurs an exponential blow-up in terms of states. Hence, the technique shown here is exponential (in terms of time) in the size of $\mathcal{P}$ and φ.

Application. If we consider that π_i range over $Traces_\omega(\mathcal{P})$ and that $\pi_j, j \neq i$ range over a regular abstraction $\alpha(Traces_\omega(\mathcal{P}))$ of the actual set of traces, and we assume that $\chi_j = \forall$ for $j \neq i$, we can apply this result to over-approximate the model-checking problem, as detailed earlier in this section.

5 Model-checking HyperLTL with Bounded Phases

In this section, we use results on *Multi-Stack Pushdown Automata* to define an under-approximation of the model-checking problem of HyperLTL formulas with universal quantifiers for PDSs.

Multi-stack pushdown systems (MPDSs) are pushdown systems with multiple stacks. Their semantics is defined in a manner similar to PDSs, and so are configurations, traces, runs, *Multi-Stack Pushdown Automata* (MPDA), and the semantics of LTL. MPDA are unfortunately Turing powerful even with only two stacks. Thus, La Torre et al. introduced in [13] a restriction called *phase-bounding*:

Definition 6 (Phases of runs). *A run r of a MPDS $\mathcal{M}$ is said to be* k-phased *if it can be split in a sequence of k runs $r_1, \ldots, r_k$ of $\mathcal{M}$ (i.e. $r = r_1 \ldots r_k$) such that during the execution of a given run r_i, at most a single stack is popped from.*

For a given integer k, this restriction can be used to define a phase-bounded semantics on MPDSs: only traces matched to runs with at most k phases are considered. It has been proven in [11] that the backward reachability set of MPDSs with bounded phases is regular and can be effectively computed; this property can then be used to show that the following theorem holds:

Theorem 5 (Model-checking with bounded phases [11]). *The model-checking problem of LTL for MPDSs with bounded phases is decidable.*

Phase-bounding can be used to under-approximate the set of traces of a MPDS. If a given LTL property φ does not hold for a MPDS $\mathcal{M}$ with a phase-bounding constraint, it does not hold for the MPDS $\mathcal{M}$ w.r.t. the usual semantics as well. We write $\mathcal{M} \models_k \varphi$ if the LTL formula φ holds for traces of $\mathcal{M}$ with at most k phases.

We can use decidability properties of MPDSs with bounded phases to under-approximate the model-checking problem for pushdown systems. Let $\mathcal{P} = (P, \Sigma, \Gamma, \Delta, c_0\rangle)$ be a PDS on the input alphabet $\Sigma = 2^{AP}$, and $\psi = \forall\pi_1, \ldots, \forall\pi_n, \varphi$, a HyperLTL formula on n trace variables with only universal quantifiers.

Our intuition is to define a MPDS $\mathcal{M}$ such that each stack represents a path variable of the HyperLTL formula. This MPDS is the product of n copies of $\mathcal{P}$. Because ψ features *universal quantifiers* only, the model-checking problem of the LTL formula φ for $\mathcal{M}$ is then equivalent to the model-checking problem of ψ for $\mathcal{P}$: $\mathcal{M}$ simulates n runs of $\mathcal{P}$ simultaneously, hence, LTL formulas on $\mathcal{M}$ can be used to synchronize these runs. We can therefore use a phase-bounded approximation of the former problem to under-approximate the latter.

We introduce the MPDS $\mathcal{M} = (P^n, \Sigma^n, \Gamma^n, n, \Delta', c'_0)$, with an initial configuration $c'_0 = \langle (p_0, \ldots, p_0), \bot, \ldots, \bot \rangle) \in P^n \times \Gamma^n$ and a set of transitions Δ' defined as follows: $\forall d_1, \ldots, d_n \in \Delta^n$ where $d_i = (p_i, \gamma_i) \xrightarrow{a_i} (p'_i, w_i)$ for $i = 1, \ldots, n$, the transition $((p_1, \ldots, p_n), \gamma_1, \ldots, \gamma_n) \xrightarrow{(a_1, \ldots, a_n)} ((p'_1, \ldots, p'_n), w_1, \ldots, w_n)$ belongs to Δ'. The following lemma then holds:

Lemma 2. *$\mathcal{M} \models \varphi$ if and only if $P \models \psi$.*

As a consequence, if $\mathcal{M} \not\models \varphi$, then $P \not\models \psi$. We can then consider a phase-bounded analysis of $\mathcal{M}$: for a given integer k, if $\mathcal{M} \not\models_k \varphi$, then $\mathcal{M} \not\models \varphi$, hence $P \not\models \psi$. We can therefore under-approximate the model-checking problem of HyperLTL formulas with universal quantifiers only.

6 Applications to Security Properties

We apply in this section our results to information flow security, and remind how, as shown in [5], security policies can be expressed as HyperLTL formulas. If we model a given program as a PDS or a VPDS $\mathcal{P}$ following the method outlined in [7], we can then either over-approximate or under-approximate an answer to the model-checking problem $\mathcal{P} \models \psi$ of a policy represented by a HyperLTL formula ψ for this program.

6.1 Observational Determinism

The *strict non-interference* security policy is the following: an attacker should not be able to distinguish two computations from their outputs if they only vary in their secret inputs. Few actual programs meet this requirement, and different versions of this policy have thus been defined.

We partition variables of a program into high and low security variables, and into input and output variables. The *observational determinism* property holds if, assuming two starting configurations have identical low security input variables, their low security output variables will be equal as well.

We model the program as a PDS $\mathcal{P}$ on the input alphabet 2^{AP}, where atomic propositions in AP contain variable values: if a variable x can take a value a, then $(x, a) \in AP$. We can express the observational determinism policy as the following HyperLTL formula:

$$\psi_{OD} = \forall \pi_1, \forall \pi_2, (\bigwedge_{a \in LS_i} (a_{\pi_1} \Leftrightarrow a_{\pi_2})) \Rightarrow G\,(\bigwedge_{b \in LS_o} (b_{\pi_1} \Leftrightarrow b_{\pi_2}))$$

where LS_i (resp. LS_o) is the set of low security input (resp. output) variables values. Using our techniques detailed in Sections 5 and 4.1, we can both under-approximate and over-approximate the model-checking problem $\mathcal{P} \models \psi_{OD}$ that is otherwise undecidable.

A context-free example. Let $AP = \{i, o, h_1, h_2\}$, $LS_i = \{i\}$, $LS_o = \{o\}$, and let $HS_i = \{h_1, h_2\}$ be a set of high security inputs. We suppose we are given a

program that can be abstracted by the following PDS $\mathcal{P}$ on the alphabet $\Sigma = 2^{AP}$, the stack alphabet $\Gamma = \{\gamma, \bot\}$, and the set of states $P = \{p_0, p_1, p_2, p_3, p_4\}$, with the following set of transitions, as represented by Figure 1:

$$
\begin{array}{ll}
(init)\ (p_0, \bot) \xrightarrow{\{i\}} (p_0, \gamma\bot) & (\mu_2)\ (p_2, \gamma) \xrightarrow{\{h_1\}} (p_3, \varepsilon) \\
(\lambda_1)\ (p_0, \gamma) \xrightarrow{\{h_1\}} (p_1, \gamma\gamma) & (\mu_3)\ (p_3, \gamma) \xrightarrow{\{o\}} (p_2, \gamma) \\
(\lambda_2)\ (p_0, \gamma) \xrightarrow{\{h_2\}} (p_1, \gamma\gamma) & (\nu_1)\ (p_3, \bot) \xrightarrow{\{o\}} (p_4, \bot) \\
(\lambda_3)\ (p_1, \gamma) \xrightarrow{\{o\}} (p_0, \gamma) & (\nu_2)\ (p_4, \bot) \xrightarrow{\{o\}} (p_4, \bot) \\
(\mu_1)\ (p_1, \gamma) \xrightarrow{\{o\}} (p_2, \gamma) &
\end{array}
$$

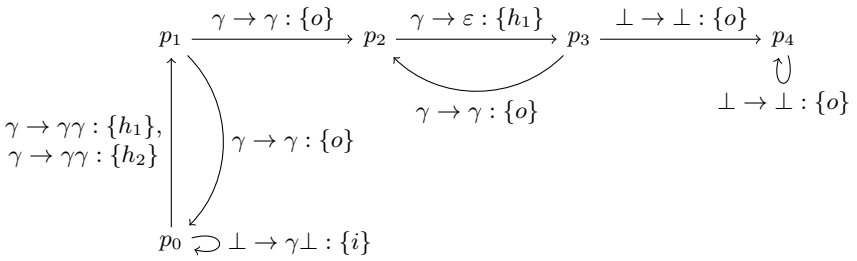

Fig. 1: The PDS $\mathcal{P}$

We would like to check if $P \models \psi_{OD}$, where ψ_{OD} is the observational determinism HyperLTL formula outlined above. Intuitively, it will not hold: two runs always have the same input i but, if they do not push the same number of symbols on the stack, their low-security outputs will differ.

Since transitions of $\mathcal{P}$ are only labelled by singletons, we can write ρ instead of $\{\rho\}$ when describing traces. The set $Traces_\omega(\mathcal{P})$ of infinite traces of $\mathcal{P}$ is equal to $\bigcup_{n \in \mathbb{N}} i \cdot ((h_1 + h_2) \cdot o)^n \cdot (h_1 \cdot o)^{n+1} \cdot o^*$: from the bottom symbol $\bot$, rules $(init)$, (λ_1), (λ_2), and (λ_3) push $n+1$ symbols γ on the stack, rules (μ_1), (μ_2), and (μ_3) pop these $(n+1)$ symbols, then rule (ν_2) loop in state p_4 once the bottom of the stack is reached again and rule (ν_1) has been applied. $Traces_\omega(\mathcal{P})$ is context-free, hence, we can't model-check the observational determinism policy on $\mathcal{P}$ using the algorithms outlined in [6].

Using the under-approximation technique outlined in Section 5, we can show that ψ_{OD} does not hold if we bound the number of phases to 2: we find a counter-example $\pi_1 = i \cdot h_2 \cdot o \cdot h_1 \cdot o \cdot o^*$ and $\pi_2 = i \cdot (h_2 \cdot o)^2 \cdot (h_1 \cdot o)^2 \cdot o^*$. We can therefore reach the conclusion that $\mathcal{P} \not\models \psi_{OD}$; the observational determinism security policy therefore does not hold for the original program.

6.2 Declassification

The strict non-interference security policy is very hard to enforce as many programs must, one way or another, leak secret information during their execution. Thus, we must relax the previously defined security properties.

We introduce instead a *declassification* policy: at a given step, leaking a specific high security variable is allowed, but the observational determinism must otherwise holds. As an example, let's consider a program accepting a password as a high security input in its initial state, whose correctness is then checked during the next execution step. The program's behaviour then depends on the password's correctness. We express this particular declassification policy as the following HyperLTL formula:

$$\psi_D = \forall \pi_1, \forall \pi_2, ((\bigwedge_{a \in LS_i} (a_{\pi_1} \Leftrightarrow a_{\pi_2})) \wedge X\,(\rho_{\pi_1} \Leftrightarrow \rho_{\pi_2})) \Rightarrow G\,(\bigwedge_{b \in LS_o} (b_{\pi_1} \Leftrightarrow b_{\pi_2}))$$

where ρ is a high security atomic proposition specifying that an input password is correct. Again, using our techniques detailed in Sections 5 and 4.1, we can both under-approximate and over-approximate the model-checking problem $\mathcal{P} \models \psi_D$.

Checking a password. We consider a program where the user can input a low-security username and a high-security password, then get different outputs depending on whether the password is true or not.

Let $AP = \{u, pw_1, pw_2, pw_3, o, \rho, h_1, h_2\}$, $LS_i = \{u\}$, $LS_o = \{o\}$, let ρ be a variable that is allowed to leak, and let $HS_i = \{pw_1, pw_2, pw_3, h_1, h_2\}$ be a set of high security inputs. Assuming there is only a single username u and three possible passwords pw_1, pw_2, pw_3, pw_3 being the only right answer, we can consider the following PDS $\mathcal{P}$ on the alphabet $\Sigma = 2^{AP}$, the stack alphabet $\Gamma = \{\gamma, \bot\}$, the set of states $P = \{p_0, p_1, p_2, p_3, p_{true}, p_{false}\}$, with the following set of transitions, as represented by Figure 2:

$$\begin{array}{llll}
(init_1) & (p_0, \bot) \xrightarrow{\{u, pw_1\}} (p_{false}, \bot) & (\mu_1) & (p_1, \bot) \xrightarrow{\{o\}} (p_1, \bot) \\
(init_2) & (p_0, \bot) \xrightarrow{\{u, pw_2\}} (p_{false}, \bot) & (\mu_2) & (p_2, \gamma) \xrightarrow{\{h_1\}} (p_2, \gamma\gamma) \\
(init_3) & (p_0, \bot) \xrightarrow{\{u, pw_3\}} (p_{true}, \bot) & (\mu_3) & (p_2, \gamma) \xrightarrow{\{h_2\}} (p_3, \gamma) \\
(pw_{true}) & (p_{true}, \bot) \xrightarrow{\{\rho\}} (p_1, \bot) & (\mu_4) & (p_3, \gamma) \xrightarrow{\{h_2\}} (p_3, \varepsilon) \\
(pw_{false}) & (p_{false}, \bot) \xrightarrow{\{o\}} (p_2, \gamma\bot) & (\mu_2) & (p_3, \bot) \xrightarrow{\{h_1\}} (p_3, \bot)
\end{array}$$

We would like to check if $P \models \psi_D$, where ψ_D is the declassification HyperLTL formula outlined above. Obviously, if we consider that $\rho \in LS_o$, then observational determinism does not hold: given the same username u, depending on whether the high-security password p_i chosen is right or not, the low-security output will differ. However, intuitively, the declassification policy should hold: given two different input passwords, the PDS will behave in the same manner as long as both are either true or false.

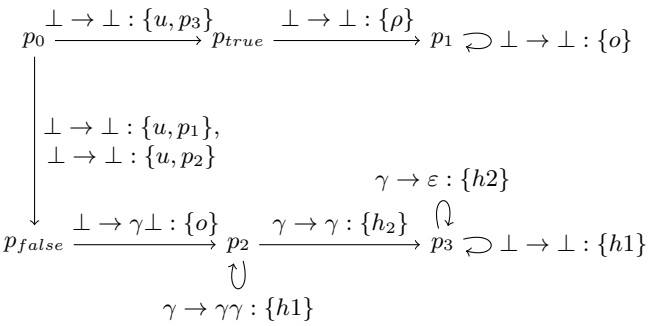

Fig. 2: The PDS $\mathcal{P}$

The set $Traces_\omega(\mathcal{P})$ of infinite traces of $\mathcal{P}$ is equal to $(\{u, p_3\} \cdot \{\rho\} \cdot \{o_1\}^*) \cup \bigcup_{n \in \mathbb{N}} ((\{u, p_1\} + \{u, p_2\}) \cdot \{o\} \cdot \{h_1\}^n \cdot \{h_2\}^{n+2} \cdot \{h_1\}^*)$: from the bottom symbol $\bot$, if the right password pw_3 has been input, rules $(init_3)$ and (p_{true}) lead to state p_1 where the PDS loops; otherwise, if the password is wrong, rules $(init_1), (init_2)$ and (p_{false}) push a symbol γ and lead to state p_2, where rule (μ_2) pushes n symbols γ on the stack, then the PDS switches to state p_3 where it pops these $(n + 1)$ symbols with rules (μ_3) and (μ_4) then loops with rule (μ_5) once the bottom of the stack has been reached. $Traces_\omega(\mathcal{P})$ is context-free, hence, we can't model-check the declassification policy on $\mathcal{P}$ using the algorithms outlined in [6].

Using the overapproximation techniques detailed in Section 4.1, we can consider the regular abstraction $\alpha(Traces_\omega(\mathcal{P})) = (\{u, p_3\} \cdot \{\rho\} \cdot \{o_1\}^*) \cup ((\{u, p_1\} + \{u, p_2\}) \cdot \emptyset \cdot \{h_1\}^* \cdot \{h_2\}^* \cdot \{h_1\}^*)$ of the actual set of traces. We can then reach the conclusion that $\mathcal{P} \models \psi_D$, since this property holds for the over-approximation as well; the declassification security policy therefore holds for this example.

6.3 Non-Inference

Non-inference is a variant of the non-interference security policy. It states that, should all high security input variables be replaced by a dummy input λ, the behaviour of low security variables should not change.

We express this property as the following HyperLTL formula:

$$\psi = \forall \pi_1, \exists \pi_2, G\,(\bigwedge_{x \in V_i^h} (x, \lambda)_{\pi_2}) \wedge G\,(\bigwedge_{b \in LS} (b_{\pi_1} \Leftrightarrow b_{\pi_2}))$$

where LS stands for the set of all low security variables values, V_i^h for the set of high security input variables, and (x, λ) means that variable x has value λ. We can't rely on the method outlined in 4.1 because π_2 is existentially quantified, but an over-approximation can nonetheless be found using the method detailed in Section 4.2, if we model the program as a VPDS $\mathcal{P}$, choose π_2 as the visibly

context-free path variable, and make it so that π_1 ranges over a regular abstraction of the traces.

References

1. Rajeev Alur and P. Madhusudan. Visibly pushdown languages. STOC '04.
2. Manuel E. Bermudez and Karl M. Schimpf. Practical arbitrary lookahead lr parsing. *Journal of Computer and System Sciences*, 1990.
3. Ahmed Bouajjani, Javier Esparza, and Oded Maler. Reachability analysis of pushdown automata: Application to model-checking. CONCUR '97.
4. Dario Carotenuto, Aniello Murano, and Adriano Peron. 2-visibly pushdown automata. DLT '15.
5. Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. Temporal logics for hyperproperties. POST '14.
6. Michael R. Clarkson and Fred B. Schneider. Hyperproperties. *J. Comput. Secur.*, 2010.
7. Javier Esparza, David Hansel, Peter Rossmanith, and Stefan Schwoon. Efficient algorithms for model checking pushdown systems. CAV '00.
8. Bernd Finkbeiner, Markus N. Rabe, and César Sánchez. Algorithms for model-checking hyperltl and hyperctl*. CAV '15.
9. Fernando C. N. Pereira and Rebecca N. Wright. Finite-state approximation of phrase structure grammars. ACL '91.
10. Shaz Qadeer and Jakob Rehof. Context-bounded model checking of concurrent software. In *TACAS'05*.
11. Anil Seth. Global reachability in bounded phase multi-stack pushdown systems. In *CAV'10*.
12. A. Prasad Sistla, Moshe Y. Vardi, and Pierre Wolper. The complementation problem for büchi automata with applications to temporal logic. *Theoretical Computer Science*, 49(2):217 – 237, 1987.
13. Salvatore La Torre, Parthasarathy Madhusudan, and Gennaro Parlato. A robust class of context-sensitive languages. In *LICS '07*.
14. Moshe Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Banff Higher Order Workshop*, volume 1043 of *Lecture Notes in Computer Science*, 1995.

A Proof of Theorem 1

Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be two context-free languages, and $\mathcal{P}_1 = (P_1, \Sigma \cup \{f\}, \Gamma, \Delta_1, \langle p_0^1, \bot\rangle, F_1)$ and $\mathcal{P}_2 = (P_2, \Sigma \cup \{f\}, \Gamma, \Delta_2, \langle p_0^2, \bot\rangle, F_2)$ two PDA accepting $\mathcal{L}_1$ and $\mathcal{L}_2$ respectively. Without loss of generality, we can consider that $P_1 \cap P_2 = \emptyset$. Let $e_1 \notin P_1$, $e_2 \notin P_2$, and $f \notin \Sigma$.

We define two BPDA $\mathcal{BP}_i = (P_i, 2^{\Sigma\cup\{f\}}, \Gamma, \Delta_i', \langle p_0^i, \bot\rangle, \{e_i\})$ for $i = 1, 2$, where Δ_i' is such that $(e_i, \gamma) \xrightarrow{\{f\}} (e_i, \gamma) \in \Delta_i$ and $(p_f, \gamma) \xrightarrow{\{f\}} (e_i, \gamma) \in \Delta_i$ for all $\gamma \in \Gamma$ and $p_f \in F_i$, and if $(p, \gamma) \xrightarrow{a} (p', w) \in \Delta_i$, then $(p, \gamma) \xrightarrow{\{a\}} (p', w) \in \Delta_i'$. If we consider that $\{a\}$ is equivalent to the label $a \in \Sigma \cup \{f\}$, $\mathcal{BP}_1$ and $\mathcal{BP}_2$ accept $\mathcal{L}_1 f^\omega$ and $\mathcal{L}_2 f^\omega$ respectively. Since HyperLTL formulas apply to infinite words in $2^A P$, for $i = 1, 2$, we have designed a BPDA $\mathcal{BP}_i$ that extends words in $\mathcal{L}_i$ by adding a final dead state e_i from which the automaton can only output an infinite sequence of a special ending symbol f.

We consider the PDS $\mathcal{P} = (\{p_0\} \cup P_1 \cup P_2, \{\{\iota^1\}, \{\iota^2\}\} \cup 2^{\Sigma\cup\{f\}}, \Gamma, \Delta, c_0)$, where $p_0 \notin P_1 \cup P_2$, $\iota^1, \iota^2 \notin \Sigma \cup \{f\}$, $c_0 = \langle p_0, \bot\rangle$, and $\Delta = \{(p_0, \bot) \xrightarrow{\{\iota^1\}} (p_0^1, \bot), (p_0, \bot) \xrightarrow{\{\iota^2\}} (p_0^2, \bot)\} \cup \Delta_1' \cup \Delta_2'$. The PDS $\mathcal{P}$ can simulate either $\mathcal{BP}_1$ or $\mathcal{BP}_2$, depending on whether it applies first a transition labelled by $\{\iota^1\}$ or $\{\iota^2\}$ from the initial configuration c_0.

We introduce the formula $\psi = \exists \pi_1, \exists \pi_2, \varphi_{start} \wedge \varphi_{sync} \wedge \varphi_{end}$ on $AP = \{\iota^1, \iota^2\} \cup \Sigma \cup \{f\}$, where $\varphi_{start} = \iota^1_{\pi_1} \wedge \iota^2_{\pi_2}$, $\varphi_{sync} = \text{XG} \bigwedge_{a \in AP} (a_{\pi_1} \Leftrightarrow a_{\pi_2})$, and $\varphi_{end} = \text{FG}\ (f_{\pi_1} \wedge f_{\pi_2})$. We suppose that $\mathcal{P} \models \psi$; φ_{start} expresses that trace variables π_1 and π_2 represent runs of $\mathcal{BP}_1$ and $\mathcal{BP}_2$ respectively. φ_{sync} means that the two traces are equal from their second letter onwards. φ_{end} implies that the two traces are accepting runs.

Therefore, if $\mathcal{P} \models \psi$, then $\mathcal{B}_1$ and $\mathcal{B}_2$ share a common accepting run and $\mathcal{L}_1 \cap \mathcal{L}_2 \neq \emptyset$. On the other hand, if $\mathcal{L}_1 \cap \mathcal{L}_2 \neq \emptyset$, there is an accepting run π common to $\mathcal{B}_1$ and $\mathcal{B}_2$, and we can then find two traces π_1 and π_2 of $\mathcal{P}$ such that the formula $\exists \pi_1, \exists \pi_2, \varphi_{start} \wedge \varphi_{sync} \wedge \varphi_{end}$ holds. The emptiness problem is, however, undecidable, and therefore so must be the model-checking problem.

B Proof of Theorem 2

In order to prove this theorem, we will rely on a class of two-stack automata called *2-Visibly Pushdown Automata* (2-VPDA) introduced in [4]. Let Σ be a finite input alphabet with two partitions $\Sigma = \Sigma_{c_j} \cup \Sigma_{r_j} \cup \Sigma_{l_j}$, $j \in \{1, 2\}$. We then introduce a *2-pushdown alphabet* $\aleph = \langle (\Sigma_{c_1}, \Sigma_{r_1}, \Sigma_{l_1}), (\Sigma_{c_2}, \Sigma_{r_2}, \Sigma_{l_2})\rangle$ on Σ.

Definition 7 (2-Visibly Pushdown Automaton). *A* 2-Visibly Pushdown Automaton *(2-VPDA) over $\aleph$ is a tuple $\mathcal{D} = (P, \Sigma, \Gamma, \Delta, c_0, F)$ where P is a finite set of control states, Σ a finite input alphabet, Γ a finite stack alphabet, $\Delta \subseteq (P \times \Gamma \times \Gamma) \times \Sigma \times (P \times \Gamma^* \times \Gamma^*)$ a finite set of transition rules, $c_0 =$*

$\langle p_0, \bot, \bot \rangle \in P \times \Gamma \times \Gamma$ *an initial configuration, and* $F \subseteq P$ *a set of final states. Moreover,* Δ *is such that* $\forall d \in \Delta$*, and for* $i \in \{1, 2\}$*:*

- *if* d *is labelled by a letter in* Σ_{c_i}*,* d *pushes a word on the* i*-th stack regardless of its top stack symbol;*
- *if* d *is labelled by a letter in* Σ_{r_i}*,* d *pops the top letter of of the* i*-th stack;*
- *if* d *is labelled by a letter in* Σ_{l_i}*,* d *does not modify the* i*-th stack.*

The semantics of 2-VPDA is defined in a manner similar to PDA, and so are configurations, runs, execution paths, languages, and 2-Büchi Visibly Pushdown Automata (2-BVPDA). The following theorem holds:

Theorem 6 (Undecidability of the emptiness problem for 2-VPDA [4]). *The emptiness problem for 2-VPDA is undecidable.*

Our intuition is to prove theorem 2 by reducing the emptiness problem for 2-VPDA to the model-checking problem of HyperLTL for VPDSs.

Let $\mathcal{D} = (P, \Sigma, \Gamma, \Delta, \langle p_0, \bot, \bot \rangle, F)$ be a 2-VPDA on an input alphabet Σ according to a partition $\aleph = \langle (\Sigma_{c_1}, \Sigma_{r_1}, \Sigma_{l_1}), (\Sigma_{c_2}, \Sigma_{r_2}, \Sigma_{l_2}) \rangle$. We introduce a 2-BVPDA $\mathcal{BD} = (P, 2^{\Sigma \cup \{f\}}, \Gamma, \Delta', \langle p_0, \bot, \bot \rangle, \{e\})$ such that $(e, \gamma, \gamma') \xrightarrow{\{f\}} (e, \gamma, \gamma') \in \Delta'$ and $(p_f, \gamma, \gamma') \xrightarrow{\{f\}} (e, \gamma, \gamma') \in \Delta'$ for all $\gamma, \gamma' \in \Gamma$ and $p_f \in F$, and if $(p, \gamma, \gamma') \xrightarrow{a} (p', w, w') \in \Delta$, then $(p, \gamma, \gamma') \xrightarrow{\{a\}} (p', w, w') \in \Delta'$ on the input alphabet $\Sigma \cup \{f\}$. Obviously, $\mathcal{BD}$ is visibly if we add the symbol f to Σ_{l_1} and Σ_{l_2} and accepts $\mathcal{L}(\mathcal{D}) f^\omega$, assuming the label $\{a\}$ is equivalent to the label $a \in \Sigma \cup \{f\}$.

Let P^1 and P^2 (resp. Δ^1 and Δ^2) be two disjoint copies of P (resp. Δ'). To each $p \in P$ (resp. $d \in \Delta'$), we match its copies $p^1 \in P^1$ and $p^2 \in P^2$ (resp. $d^1 \in \Delta^1$ and $d^2 \in \Delta^2$). We define a PDS $P = (\{\sigma\} \cup P^1 \cup P^2, \{\{\iota^1\}, \{\iota^2\}, \{f\}\} \cup 2^{\Delta^1} \cup 2^{\Delta^2}, \Gamma, \delta, \langle \sigma, \bot, \rangle)$. The set δ is such that, for each transition $d = (p, \gamma_1, \gamma_2) \xrightarrow{a} (p', w_1, w_2) \in \Delta$, $a \neq f$, we add two transitions $(p^1, \gamma_1) \xrightarrow{\{d^1\}} (p'^1, w_1)$ and $(p^2, \gamma_2) \xrightarrow{\{d^2\}} (p'^2, w_2)$ to δ. If $a = f$, we add instead $(p^1, \gamma_1) \xrightarrow{\{f\}} (p'^1, w_1)$ and $(p^2, \gamma_2) \xrightarrow{\{f\}} (p'^2, w_2)$. Transitions in δ are projections of the original transitions of the 2-BVPDA on one of its two stacks; their label depends on the original transition in Δ, unless they are labelled by f. Moreover, $(\sigma, \bot) \xrightarrow{\{\iota_1\}} (p_0^1, \bot)$ and $(\sigma, \bot) \xrightarrow{\{\iota_2\}} (p_0^2, \bot)$ both belong to δ.

$\mathcal{P}$ is such that it can either simulate the first or the second stack of the 2-BVPDA $\mathcal{BD}$, depending on which transition was used first. $\mathcal{P}$ is indeed a VPDS: a suitable partition of its input alphabet can be computed depending on which operation on the i-th stack transitions in Δ perform. As an example, if $d \in \Delta$ pushes a symbol on the first stack and pops from the second, d^1 belongs to the call alphabet and d^2, to the return alphabet.

Given a set of trace variables $\mathcal{V} = \{\pi_1, \pi_2\}$ and a predicate alphabet $AP = \{\iota^1, \iota^2, f\} \cup \Delta^1 \cup \Delta^2$, we then consider an unquantified HyperLTL formula φ of the form $\varphi = \varphi_{start} \wedge \varphi_{sync} \wedge \varphi_{end}$, where φ's sub-formulas are defined as follows:

Initialization formula: $\varphi_{start} = \iota^1_{\pi_1} \wedge \iota^2_{\pi_2}$; $\Pi \models \varphi_{start}$ if and only if for $i \in \{1, 2\}$, $\Pi[1, \infty](\pi_i)$ is a run that simulates the i-th stack of $\mathcal{BD}$;

Synchronization formula: $\varphi_{sync} = XG \bigwedge_{d \in \Delta} (d^1_{\pi_1} \Leftrightarrow d^2_{\pi_2})$; $\Pi \models \varphi_{start} \wedge \varphi_{sync}$ if and only if $\Pi[1, \infty](\pi_1)$ and $\Pi[1, \infty](\pi_2)$ can be matched to a common run of the 2-BVPDA $\mathcal{BD}$;

Acceptation formula: $\varphi_{end} = FG\,(f_{\pi_1} \wedge f_{\pi_2})$; $\Pi \models \varphi_{start} \wedge \varphi_{sync} \wedge \varphi_{end}$ if and only if $\Pi[1, \infty](\pi_1)$ and $\Pi[1, \infty](\pi_2)$ can be used to define an accepting run of the 2-BVPDA $\mathcal{BD}$.

Therefore, if $\Pi \models \varphi$, we have $\Pi(\pi_i) = (\iota^i, d^i_1, d^i_2 \ldots)$ for $i = 1, 2$, and the sequence of transitions $(d_1, d_2, \ldots) \in \Delta^\omega$ defines an accepting run on $\mathcal{BD}$. Therefore, we can solve the model-checking problem $\mathcal{P} \models \exists \pi_1, \exists \pi_2, \varphi$, if and only if we can determine whether $\mathcal{L}(\mathcal{BD})$ is empty or not, hence, $\mathcal{L}(\mathcal{D})$ as well. By here is a contradiction and the former problem is undecidable.