Model-Checking LTL with Regular Valuations for Pushdown Systems *

Javier Esparza[1], Antonín Kučera**[2], and Stefan Schwoon[1]

[1] Institute for Informatics TUM, Arcisstr. 21, 80290 Munich, Germany, {esparza,schwoon}@in.tum.de
[2] Faculty of Informatics MU, Botanická 68a, 60200 Brno, Czech Republic, tony@fi.muni.cz

Abstract. Recent works have proposed pushdown systems as a tool for analyzing programs with (recursive) procedures. In particular, the model-checking problem for LTL has been studied. In this paper we examine an extension of this, namely model-checking with regular valuations. The problem is solved via two different techniques, with an eye on efficiency – both techniques can be shown to be essentially optimal. Our methods are applicable to problems in different areas, e.g., data-flow analysis, analysis of systems with checkpoints, etc., and provide a general, unifying and efficient framework for solving these problems.

1 Introduction

Pushdown systems can be seen as a natural abstraction of programs written in procedural, sequential languages such as C. They generate infinite-state transition systems whose states are pairs consisting of a control location (which stores global information about the program) and stack content (which keeps the track of activation records, i.e., previously called procedures and their local variables).

Previous research has established applications of pushdown systems for the analysis of Boolean Programs [1, 8] and certain data-flow analysis problems [7]. The model-checking problem has been considered for various logics, and quite efficient algorithms have emerged for linear time logics [2, 6, 9].

In this paper we revisit the model-checking problem for LTL and pushdown systems. Generally speaking, the problem is undecidable for arbitrary valuations, i.e., the functions that map the atomic propositions of a formula to the respective sets of pushdown configurations that satisfy them. However, it remains decidable for some restricted classes of valuations. In [2, 6, 9] valuations were completely determined by the control location and/or the topmost stack symbol (we call these valuations 'simple' in the following). Here we propose (and solve) the problem for valuations depending on regular predicates over the complete stack content. We argue that this solution provides a general, efficient, and unifying framework for problems from different areas (e.g., data-flow analysis, analysis of systems with checkpoints, etc.)

* This work was partially supported by the project "Advanced Validation Techniques for Telecommunication Protocols" of the Information Societies Technology Programme of the European Union.

** On leave at the Institute for Informatics, TU Munich. Supported by a Research Fellowship granted by the Alexander von Humboldt Foundation and by a grant GA ČR No. 201/00/1023.

We proceed as follows: Section 2 contains basic definitions. Most technical content is in Section 3 where we formally define simple and regular valuations and propose our solutions to the model-checking problem with regular valuations, based on a reduction to the case of simple valuations. We can thus re-use most of the theory from [6]. While the reduction itself is based on a standard method, we pay special attention to ensure its efficiency, modifying the algorithm of [6] to take advantage of specific properties of our constructions. We propose two different techniques – one for regular valuations in general and another for a restricted subclass – both of which increase the complexity by only a linear factor (in the size of an automaton for the atomic regular predicates). By contrast, a blunt reduction and analysis would yield up to a quadric ('n^4') blowup. Even though one technique is more powerful than the other at the same asymptotic complexity, we present them both since it is not clear how they might perform in practice.

In Section 4 we consider applicability of our abstract results. The first area (Section 4.1) are problems of interprocedural data-flow analysis. Here, regular valuations can be used to 'gather' pieces of information which dynamically depend on the history of procedure calls. LTL can express quite complex relationships among those dynamic properties and allows to solve relatively complicated problems using our model-checking algorithm. To give a concrete example, we indicate how to decide whether a given variable Y is dead at a given point of a recursive program with dynamic scoping. Another application (Section 4.2) is connected to systems with checkpoints. First, we introduce a formal model for such systems, called *pushdown systems with checkpoints*. The idea is that the computation is interrupted at certain points and some property of the stack content is checked. Further computational steps depend on the result of this inspection. This part of our work is motivated by the advent of programming languages which can enforce security requirements. Newer versions of Java, for instance, enable programs to perform local security checks in which the methods on the stack are checked for correct permissions. Jensen et al [10] first proposed a formal framework for such systems. With their techniques one can prove the validity of control flow based global security properties as well as to detect (and remove) redundant checkpoints. Our methods are more general, however; for instance, we are not restricted to safety properties, and our model can represent data-flow as well. Properties of pushdown systems with checkpoints can be expressed in LTL for which we provide an efficient model-checking algorithm. In Section 4.3 we present and analyze a model-checking algorithm for CTL*. In the context of finite-state systems it is well-known that model-checking the more powerful logic CTL* can be reduced to checking LTL [5]. This technique can be transferred to pushdown systems using model-checking with regular valuations.

The complexity of all of the previously developed algorithms is measured in certain parameters of the problem which are usually small, and our complexity bounds are polynomials in those parameters. In general, those parameters can be *exponential* in the size of a problem instance. Therefore, it is natural to ask whether it is possible to solve some of the studied problems more efficiently by other (possibly quite different) techniques. This question is answered (negatively) in Section 5 where we establish **EXPTIME** lower bounds for those problems (even for rather restricted forms of them). Hence, all of our algorithms are essentially optimal. Complexity measures are discussed in more detail in Remark 1 and in Section 5. We draw our conclusions in Section 6.

2 Preliminaries

2.1 Transition Systems

A *transition system* is a triple $\mathcal{T} = (S, \to, r)$ where S is a set of *states*, $\to \subseteq S \times S$ is a *transition relation*, and $r \in S$ is a distinguished state called *root*.

As usual, we write $s \to t$ instead of $(s,t) \in \to$. The reflexive and transitive closure of $\to$ is denoted by $\to^*$. We say that a state t is *reachable from a state* s if $s \to^* t$. A state t is *reachable* if it is reachable from the root.

A *run* of $\mathcal{T}$ is an infinite sequence of states $w = s_0 s_1 s_2 \ldots$ such that $s_i \to s_{i+1}$ for each $i \geq 0$. Observe that an arbitrary suffix of a run is again a run – for every $i \in \mathbb{N}_0$ we define the i^{th} suffix of w, denoted $w(i)$, to be the run $s_i s_{i+1} s_{i+2} \ldots$

2.2 The Logic LTL

Let $At = \{A, B, C, \ldots\}$ be a (countable) set of *atomic propositions*. LTL formulae are built according to the following abstract syntax equation (where A ranges over At):

$$\varphi ::= \mathtt{tt} \mid A \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \mathcal{X}\varphi \mid \varphi_1 \,\mathcal{U}\, \varphi_2$$

Let $\mathcal{T} = (S, \to, r)$ be a transition system. A *valuation* of atomic propositions is a function $\nu \colon At \to 2^S$. The *validity* of an LTL formula φ for a run $w = s_0 v$ w.r.t. a valuation ν, denoted $w \models^\nu \varphi$, is defined inductively on the structure of φ as follows:

- $w \models^\nu \mathtt{tt}$
- $w \models^\nu A \iff s_0 \in \nu(A)$
- $w \models^\nu \neg\varphi \iff w \not\models^\nu \varphi$
- $w \models^\nu \varphi_1 \wedge \varphi_2 \iff w \models^\nu \varphi_1$ and $w \models^\nu \varphi_2$
- $w \models^\nu \mathcal{X}\varphi \iff w(1) \models^\nu \varphi$
- $w \models^\nu \varphi_1 \,\mathcal{U}\, \varphi_2 \iff \exists i \colon (w(i) \models^\nu \varphi_2) \wedge (\forall j < i \colon w(j) \models^\nu \varphi_1)$

We also define $\Diamond\varphi \equiv \mathtt{tt}\,\mathcal{U}\,\varphi$ and $\Box\varphi \equiv \neg(\Diamond\neg\varphi)$. An LTL formula φ is valid in a state s w.r.t. ν, written $s \models^\nu \varphi$, iff $w \models^\nu \varphi$ for each run w which starts in the state s.

2.3 Pushdown systems

A *pushdown system* is a tuple $\mathcal{P} = (P, \Gamma, \Delta, q_0, \omega)$ where P is a finite set of *control locations*, Γ is a finite *stack alphabet*, $\Delta \subseteq (P \times \Gamma) \times (P \times \Gamma^*)$ is a finite set of *transition rules*, $q_0 \in P$ is an *initial control location*, and $\omega \in \Gamma$ is a *bottom stack symbol*.

We use Greek letters $\alpha, \beta, \ldots$ to denote elements of Γ, and small letters $v, w, \ldots$ from the end of the alphabet to denote elements of Γ^*. We also adopt a more intuitive notation for transition rules, writing $\langle p, \alpha \rangle \hookrightarrow \langle q, w \rangle$ instead of $((p,\alpha),(q,w)) \in \Delta$.

A *configuration* of $\mathcal{P}$ is an element of $P \times \Gamma^*$. To $\mathcal{P}$ we associate a unique transition system $\mathcal{T}_\mathcal{P}$ whose states are configurations of $\mathcal{P}$, the root is $\langle q_0, \omega \rangle$, and the transition relation is the least relation $\to$ satisfying the following:

$$\langle p, \alpha \rangle \hookrightarrow \langle q, v \rangle \implies \langle p, \alpha w \rangle \to \langle q, vw \rangle \text{ for every } w \in \Gamma^*$$

Without loss of generality we require that ω is never removed from the stack, i.e., whenever $\langle p, \omega \rangle \hookrightarrow \langle q, w \rangle$ then w is of the form $v\omega$.

Pushdown systems can be conveniently used as a model of recursive sequential programs. In this setting, the (abstracted) stack of activation records increases if a new procedure is invoked, and decreases if the current procedure terminates. In particular, it means that the height of the stack can increase at most by one in a single transition. Therefore, from now on we assume that all pushdown systems we work with have this property. This assumption does not influence the expressive power of pushdown systems, but it has some impact on the complexity analysis carried out in Section 3.1.

3 LTL on pushdown systems

Let $\mathcal{P} = (P, \Gamma, \Delta, q_0, \omega)$ be a pushdown system, φ an LTL formula, and $\nu\colon At \rightarrow 2^{P \times \Gamma^*}$ a valuation. We deal with the following variants of the *model checking problem*:

(I) The model checking problem for the initial configuration: does $\langle q_0, \omega \rangle \models^{\nu} \varphi$?
(II) The global model checking problem: compute (a finite description of) the set of all configurations, reachable or not, that violate φ.
(III) The global model checking problem for reachable configurations: compute (a finite description of) the set of all reachable configurations that violate φ.

In this paper we use so-called $\mathcal{P}$-*automata* to encode infinite sets of configurations of a pushdown system $\mathcal{P}$. As we shall see, in some cases we can solve the problems (II) and (III) by computing $\mathcal{P}$-automata recognizing the above defined sets of configurations.

Definition 1. *Let $\mathcal{P} = (P, \Gamma, \Delta, q_0, \omega)$ be a pushdown system. A $\mathcal{P}$-automaton is a tuple $\mathcal{A} = (Q, \Gamma, \delta, P, F)$ where Q is a finite set of* states*, Γ (i.e., the stack alphabet of $\mathcal{P}$) is the* input alphabet*, $\delta\colon Q \times \Gamma \rightarrow 2^Q$ is the* transition function*, P (i.e., the set of control locations of $\mathcal{P}$) is the set of* initial states*, and $F \subseteq Q$ is a finite set of* accepting states*. We extend δ to elements of $Q \times \Gamma^*$ in the standard way. A configuration $\langle p, w \rangle$ of $\mathcal{P}$ is* recognized *by $\mathcal{A}$ iff $\delta(p, w) \cap F \neq \emptyset$.*

In general, all of the above mentioned variants of the model checking problem are undecidable – if there are no 'effectivity assumptions' about valuations (i.e., if a valuation is an *arbitrary* function $\nu\colon At \rightarrow 2^{P \times \Gamma^*}$), one can easily express undecidable properties of pushdown configurations just by atomic propositions. Therefore, we search for 'reasonable' restrictions which do not limit the expressive power too much but allow to construct efficient model-checking algorithms at the same time. For example, we can restrict ourselves to those valuations which are completely determined by associating atomic propositions with subsets of $P \times \Gamma$ (see, e.g., [2, 6]).

Definition 2. *Let $\mathcal{P} = (P, \Gamma, \Delta)$ be a pushdown system, $f\colon (At \times P) \rightarrow 2^{\Gamma}$ a function. A* simple valuation *$\nu\colon At \rightarrow 2^{P \times \Gamma^*}$ (specified by f) is defined as follows: $\nu(A) = \{\langle p, \alpha w \rangle \mid \alpha \in f(A, p), w \in \Gamma^*\}$.*

In other words, (in)validity of an atomic proposition in a given configuration depends only on its control location and the topmost stack symbol (in our framework, we are mainly interested in reachable configurations where the stack is always nonempty). Consequently, if we are given a pushdown system $\mathcal{P}$, an LTL formula φ, and a simple valuation ν, we can easily synchronize $\mathcal{P}$ with a Büchi automaton which recognizes exactly the models of $\neg\varphi$, reducing the model-checking problem to the problem whether a given Büchi pushdown system has an accepting run [2, 6]. Here, it is crucial that atomic propositions are evaluated in a completely 'static' way because otherwise we could not perform the aforementioned synchronization.

In our paper, we propose a more general kind of valuations which are encoded by finite-state automata. We advocate this approach in the next sections by providing several examples of its applicability to practical problems; moreover, we show that this technique often results in rather efficient (or, at least, essentially optimal) algorithms by presenting relevant complexity results.

Definition 3. *Let $\mathcal{P} = (P, \Gamma, \Delta, q_0, \omega)$ be a pushdown system, f a function which assigns to each pair (A, p) of $At \times P$ a deterministic finite-state automaton $\mathcal{M}_A^p$ over the alphabet Γ with a total transition function; we also assume that the initial state of $\mathcal{M}_A^p$ is not accepting. A* regular valuation ν: $At \to 2^{P \times \Gamma^*}$ *(specified by f) is defined as follows: $\nu(A) = \{\langle p, w\rangle \mid w^R \in L(\mathcal{M}_A^p)\}$ where w^R denotes the reverse of w.*

Hence, an atomic proposition A is valid in a configuration $\langle p, w\rangle$ iff the automaton $\mathcal{M}_A^p$ enters a final state after reading the stack contents bottom-up. As we shall see, the requirement that $\mathcal{M}_A^p$ is deterministic is rather natural and has an important impact on complexity analysis. The assumption that the initial state of $\mathcal{M}_A^p$ is not accepting simplifies our next constructions – as we already mentioned, we are only interested in reachable configurations where it is impossible to empty the stack. Hence, this assumption does not bring any 'real' restrictions from a practical point of view.

3.1 Model-Checking with Regular Valuations

The variants of the model checking problem defined in the previous section have been considered in [6] for simple valuations. The following theorems are taken from there:

Theorem 1. *Let $\mathcal{P} = (P, \Gamma, \Delta, q_0, \omega)$ be a pushdown system, φ an LTL formula, and ν a simple valuation. Let $\mathcal{B}$ be a Büchi automaton for $\neg\varphi$. Then one can compute*

- *a $\mathcal{P}$-automaton $\mathcal{R}$ with $\mathcal{O}(|P|+|\Delta|)$ states and $\mathcal{O}(|P|\cdot|\Delta|\cdot(|P|+|\Delta))$ transitions in $\mathcal{O}(|P|\cdot|\Delta|\cdot(|P|+|\Delta))$ time and space such that $\mathcal{R}$ recognizes exactly the reachable configurations of $\mathcal{P}$;*
- *a $\mathcal{P}$-automaton $\mathcal{A}$ of size $\mathcal{O}(|P|\cdot|\Delta|\cdot|\mathcal{B}|^2)$ in $\mathcal{O}(|P|^2\cdot|\Delta|\cdot|\mathcal{B}|^3)$ time using $\mathcal{O}(|P|\cdot|\Delta|\cdot|\mathcal{B}|^2)$ space such that $\mathcal{A}$ recognizes exactly those configurations $\langle p, w\rangle$ of $\mathcal{P}$ (reachable or not) such that $\langle p, w\rangle \not\models^\nu \varphi$;*
- *a $\mathcal{P}$-automaton $\mathcal{A}'$ of size $\mathcal{O}(|P|\cdot|\Delta|\cdot(|P|+|\Delta|)^2\cdot|\mathcal{B}|^2)$ in $\mathcal{O}(|P|\cdot|\Delta|\cdot(|P|+|\Delta|)^2\cdot|\mathcal{B}|^3)$ time using $\mathcal{O}(|P|\cdot|\Delta|\cdot(|P|+|\Delta|)^2\cdot|\mathcal{B}|^2)$ space such that $\mathcal{A}'$ recognizes exactly those reachable configurations $\langle p, w\rangle$ of $\mathcal{P}$ such that $\langle p, w\rangle \not\models^\nu \varphi$.*

Theorem 2. *Let $\mathcal{P} = (P, \Gamma, \Delta, q_0, \omega)$ be a pushdown system, φ an LTL formula, and ν a simple valuation. Let $\mathcal{B}$ be a Büchi automaton which corresponds to $\neg\varphi$.*

– *Problems (I) and (II) can be solved in* $\mathcal{O}(|P|^2 \cdot |\Delta| \cdot |\mathcal{B}|^3)$ *time and* $\mathcal{O}(|P| \cdot |\Delta| \cdot |\mathcal{B}|^2)$ *space.*
– *Problem (III) can be solved in either* $\mathcal{O}(|P| \cdot |\Delta| \cdot (|P| + |\Delta|)^2 \cdot |\mathcal{B}|^3)$ *time and* $\mathcal{O}(|P| \cdot |\Delta| \cdot (|P| + |\Delta|)^2 \cdot |\mathcal{B}|^2)$ *space, or* $\mathcal{O}(|P|^3 \cdot |\Delta| \cdot (|P| + |\Delta|) \cdot |\mathcal{B}|^3)$ *time and* $\mathcal{O}(|P|^3 \cdot |\Delta| \cdot (|P| + |\Delta|) \cdot |\mathcal{B}|^2)$ *space.*

Our aim here is to design efficient model checking algorithms for regular valuations. We show that one can actually build on top of Theorem 1.

For the rest of this section we fix a pushdown system $\mathcal{P} = (P, \Gamma, \Delta, q_0, \omega)$, an LTL formula φ, and a regular valuation ν. The Büchi automaton which corresponds to $\neg\varphi$ is denoted by $\mathcal{B} = (R, 2^{At}, \eta, r_0, G)$. Let $\{A_1, \ldots, A_n\}$ be the subset of atomic propositions which appear in φ, and let $\mathcal{M}^p_{A_i} = (Q^p_i, \Gamma, \varrho^p_i, s^p_i, F^p_i)$ be the deterministic finite-state automaton associated to (A_i, p) for all $p \in P$ and $1 \leq i \leq n$. Observe that we do *not* require the $\mathcal{M}^p_{A_i}$ automata to be pairwise different; as we shall see in Section 4.2, there are several 'safety' problems which can be reduced to the model-checking problem for pushdown systems and the LTL logic with regular valuations. In this case, many of the $\mathcal{M}^p_{A_i}$ automata are identical and this fact substantially influences the complexity. For simplicity, we assume that whenever $i \neq j$ or $p \neq q$, then the $\mathcal{M}^p_{A_i}$ and $\mathcal{M}^q_{A_j}$ automata are either identical or have disjoint sets of states. Let $\{\mathcal{M}_1, \ldots, \mathcal{M}_m\}$ be the *set* of all $\mathcal{M}^p_{A_i}$ automata where $1 \leq i \leq n$ and $p \in P$, and let Q_j be the set of states of $\mathcal{M}_j$ for each $1 \leq j \leq m$. The Cartesian product $\prod_{1 \leq j \leq m} Q_j$ is denoted by $States$. For given $\boldsymbol{r} \in States$, $p \in P$, and $1 \leq i \leq n$, we denote by $\boldsymbol{r}^p_i$ the element of Q^p_i which appears in $\boldsymbol{r}$ (observe that we can have $\boldsymbol{r}^p_i = \boldsymbol{r}^q_j$ even if $i \neq j$ or $p \neq q$). The vector of initial states (i.e., the only element of $States$ where each component is an initial state of some $\mathcal{M}^p_{A_i}$) is denoted by $\boldsymbol{s}$. Furthermore, we write $\boldsymbol{t} = \boldsymbol{\varrho}(\boldsymbol{r}, \alpha)$ if $\boldsymbol{t}^p_i = \varrho^p_i(\boldsymbol{r}^p_i, \alpha)$ for all $1 \leq i \leq n$, $p \in P$. Now we present and evaluate two techniques for solving the model checking problems with $\mathcal{P}$, φ, and ν.

Remark 1 (On the complexity measures). The size of an instance of the model-checking problem for pushdown systems and LTL with regular valuations is given by $|\mathcal{P}| + |\nu| + |\varphi|$, where $|\nu|$ is the total size of all employed automata. However, in practice we usually work with small formulae and a small number of rather simple automata (see Section 4); therefore, we measure the complexity of our algorithms in $|\mathcal{B}|$ and $|States|$ rather than in $|\varphi|$ and $|\nu|$ (in general, $\mathcal{B}$ and $States$ can be *exponentially* larger than φ and ν). This allows for a detailed complexity analysis whose results better match the reality because $|\mathcal{B}|$ and $|States|$ stay usually 'small'. This issue is discussed in greater detail in Section 5 where we provide some lower bounds showing that all algorithms developed in this paper are also essentially optimal from the point of view of worst-case analysis.

Technique 1 – extending the finite control

The idea behind this technique is to evaluate the atomic propositions $A_1, \ldots, A_n$ 'on the fly' by storing the (product of) $\mathcal{M}^p_{A_i}$ automata in the finite control of $\mathcal{P}$ and updating the vector of states after each transition according to the (local) change of stack contents. Note that here we conveniently use the assumptions that the $\mathcal{M}^p_{A_i}$ automata are deterministic, have total transition functions, and read the stack bottom-up. However, we also need one *additional* assumption to make this construction work:

Each automaton $\mathcal{M}^p_{A_i}$ is also *backward deterministic*, i.e., for every $u \in Q^p_i$ and $\alpha \in \Gamma$ there is at most one state $t \in Q^p_i$ such that $\varrho^p_i(t, \alpha) = u$.

Note that this assumption is truly restrictive – there are quite simple regular languages which cannot be recognized by finite-state automata which are both deterministic and backward deterministic (as an example we can take the language $\{a^i \mid i > 0\}$).

We define a pushdown system $\mathcal{P}' = (P', \Gamma, \Delta', q'_0, \omega)$ where $P' = P \times States$, $q'_0 = (q_0, \boldsymbol{\varrho}(\boldsymbol{s}, \omega))$, and the transition rules Δ' are determined as follows: $\langle (p, \boldsymbol{r}), \alpha\rangle \hookrightarrow' \langle (q, \boldsymbol{u}), w\rangle$ iff the following conditions hold:

- $\langle p, \alpha\rangle \hookrightarrow \langle q, w\rangle$,
- there is $\boldsymbol{t} \in States$ such that $\boldsymbol{\varrho}(\boldsymbol{t}, \alpha) = \boldsymbol{r}$ and $\boldsymbol{\varrho}(\boldsymbol{t}, w^R) = \boldsymbol{u}$.

Observe that due to the backward determinism of $\mathcal{M}^p_{A_i}$ there is at most one $\boldsymbol{t}$ with the above stated properties; and thanks to determinism of $\mathcal{M}^p_{A_i}$ we further obtain that for given $\langle p, \alpha\rangle \hookrightarrow \langle q, w\rangle$ and $\boldsymbol{r} \in States$ there is at most one $\boldsymbol{u} \in States$ such that $\langle (p, \boldsymbol{r}), \alpha\rangle \hookrightarrow' \langle (q, \boldsymbol{u}), w\rangle$. From this it follows that $|\Delta'| = |\Delta| \cdot |States|$.

A configuration $\langle (q, \boldsymbol{r}), w\rangle$ of $\mathcal{P}'$ is *consistent* iff $\boldsymbol{r} = \boldsymbol{\varrho}(\boldsymbol{s}, w^R)$ (remember that $\boldsymbol{s}$ is the vector of initial states of $\mathcal{M}^p_{A_i}$ automata). In other words, $\langle (q, \boldsymbol{r}), w\rangle$ is consistent iff $\boldsymbol{r}$ 'reflects' the stack contents w. Let $\langle p, w\rangle$ be a configuration of $\mathcal{P}$ and $\langle (p, \boldsymbol{r}), w\rangle$ be the (unique) associated consistent configuration of $\mathcal{P}'$. Now we can readily confirm that

(A) if $\langle p, w\rangle \to \langle q, v\rangle$, then $\langle (p, \boldsymbol{r}), w\rangle \to \langle (q, \boldsymbol{u}), v\rangle$ where $\langle (q, \boldsymbol{u}), v\rangle$ is consistent;
(B) if $\langle (p, \boldsymbol{r}), w\rangle \to \langle (q, \boldsymbol{u}), v\rangle$, then $\langle (q, \boldsymbol{u}), v\rangle$ is consistent and $\langle p, w\rangle \to \langle q, v\rangle$ is a transition of $\mathcal{T}_{\mathcal{P}}$.

As the initial configuration of $\mathcal{P}'$ is consistent, we see (due to (B)) that each reachable configuration of $\mathcal{P}'$ is consistent (but not each consistent configuration is necessarily reachable). Furthermore, due to (A) and (B) we also have the following:

(C) let $\langle p, w\rangle$ be a configuration of $\mathcal{P}$ (not necessarily reachable) and let $\langle (p, \boldsymbol{r}), w\rangle$ be its associated consistent configuration of $\mathcal{P}'$. Then the parts of $\mathcal{T}_{\mathcal{P}}$ and $\mathcal{T}_{\mathcal{P}'}$ which are reachable from $\langle p, w\rangle$ and $\langle (p, \boldsymbol{r}), w\rangle$, respectively, are isomorphic.

The underlying function f of the simple valuation ν' is defined by

$$f(A_i, (p, \boldsymbol{r})) = \begin{cases} \Gamma & \text{if } \boldsymbol{r}^p_i \in F^p_i \\ \emptyset & \text{otherwise} \end{cases}$$

for all $1 \le i \le n$, $(p, \boldsymbol{r}) \in P'$. Now it is easy to see (due to (C)) that for all $p \in P$ and $w \in \Gamma^*$ we have

$$\langle p, w\rangle \models^{\nu} \varphi \iff \langle (p, \boldsymbol{r}), w\rangle \models^{\nu'} \varphi \text{ where } \boldsymbol{r} = \boldsymbol{\varrho}(\boldsymbol{s}, w^R) \tag{1}$$

During the construction of $\mathcal{P}'$ we observed that $|P'| = |P| \cdot |States|$ and $|\Delta'| = |\Delta| \cdot |States|$. Applying Theorem 2 naïvely, we obtain that using Technique 1, the model-checking problems (I) and (II) can be solved in cubic time and quadratic space (w.r.t. $|States|$), and that model-checking problem (III) takes even quadric time and space. However, closer analysis reveals that we can do *much* better.

Theorem 3. *Technique 1 (extending the finite control) gives us the following bounds on the model checking problems with regular valuations:*

1. *Problems (I) and (II) can be solved in* $\mathcal{O}(|P|^2 \cdot |\Delta| \cdot |States| \cdot |\mathcal{B}|^3)$ *time and* $\mathcal{O}(|P| \cdot |\Delta| \cdot |States| \cdot |\mathcal{B}|^2)$ *space.*
2. *Problem (III) can be solved in either* $\mathcal{O}(|P| \cdot |\Delta| \cdot (|P| + |\Delta|)^2 \cdot |States| \cdot |\mathcal{B}|^3)$ *time and* $\mathcal{O}(|P| \cdot |\Delta| \cdot (|P| + |\Delta|)^2 \cdot |States| \cdot |\mathcal{B}|^2)$ *space, or* $\mathcal{O}(|P|^3 \cdot |\Delta| \cdot (|P| + |\Delta|) \cdot |States| \cdot |\mathcal{B}|^3)$ *time and* $\mathcal{O}(|P|^3 \cdot |\Delta| \cdot (|P| + |\Delta|) \cdot |States| \cdot |\mathcal{B}|^2)$ *space.*

In other words, all problems take only *linear (!)* time and space in $|States|$.

Proof. We say that a $\mathcal{P}'$-automaton is *well-formed* iff its set of states is of the form $Q \times States$ where Q is a set such that $P \subseteq Q$. A transition $((p, \boldsymbol{t}), \alpha, (p', \boldsymbol{u}))$ of a well-formed $\mathcal{P}'$-automaton is *consistent* (w.r.t. $\boldsymbol{\varrho}$) iff $\boldsymbol{\varrho}(\boldsymbol{u}, \alpha) = \boldsymbol{t}$. A $\mathcal{P}'$-automaton is consistent iff it is well-formed and contains only consistent transitions.

For the proof we revisit the algorithms presented in [6]. Algorithm 1 shows the computation of pre^* from [6], restated for the special case of consistent $\mathcal{P}'$-automata. We first show that computation of pre^* upholds the consistency of a $\mathcal{P}'$-automaton.

Algorithm 1
Input: a pushdown system $\mathcal{P}' = (P \times States, \Gamma, \Delta', q_0, \omega)$;
a consistent $\mathcal{P}'$-automaton $\mathcal{A} = (Q \times States, \Gamma, \delta, P \times States, F)$
Output: the set of transitions of $\mathcal{A}_{pre^*}$

$rel \leftarrow \emptyset$; $trans \leftarrow \delta$; $\Delta'' \leftarrow \emptyset$;
for all $\langle (p, \boldsymbol{u}), \alpha \rangle \hookrightarrow \langle (p', \boldsymbol{u}'), \varepsilon \rangle \in \Delta'$ **do** $trans \leftarrow trans \cup \{((p, \boldsymbol{u}), \alpha, (p', \boldsymbol{u}'))\}$;
while $trans \neq \emptyset$ **do**
  pop $t = ((q, \boldsymbol{u}), \alpha, (q', \boldsymbol{u}'))$ from *trans*;
  if $t \notin rel$ **then**
    $rel \leftarrow rel \cup \{t\}$;
    for all $\langle (p_1, \boldsymbol{u}_1), \alpha_1 \rangle \hookrightarrow \langle (q, \boldsymbol{u}), \alpha \rangle \in (\Delta' \cup \Delta'')$ **do**
      $trans \leftarrow trans \cup \{((p_1, \boldsymbol{u}_1), \alpha_1, (q', \boldsymbol{u}'))\}$;
    for all $\langle (p_1, \boldsymbol{u}_1), \alpha_1 \rangle \hookrightarrow \langle (q, \boldsymbol{u}), \alpha\alpha_2 \rangle \in \Delta'$ **do**
      $\Delta'' \leftarrow \Delta'' \cup \{\langle (p_1, \boldsymbol{u}_1), \alpha_1 \rangle \hookrightarrow \langle (q', \boldsymbol{u}'), \alpha_2 \rangle\}$;
      for all $((q', \boldsymbol{u}'), \alpha_2, (q'', \boldsymbol{u}'')) \in rel$ **do**
        $trans \leftarrow trans \cup \{((p_1, \boldsymbol{u}_1), \alpha_1, (q'', \boldsymbol{u}''))\}$;
return *rel*

Lemma 1. *When receiving a consistent* $\mathcal{P}'$*-automaton as input, Algorithm 1 will output a consistent* $\mathcal{P}'$*-automaton.*

Proof. Recall that Algorithm 1 implements the saturation procedure of [2], i.e., all additions to the automaton correspond to the following situation:

> If $\langle (p, \boldsymbol{u}), \alpha \rangle \hookrightarrow \langle (p', \boldsymbol{u}'), w \rangle$ in Δ' and $(p', \boldsymbol{u}') \xrightarrow{w} (q, \boldsymbol{u}'')$ in the current automaton, then add a transition $((p, \boldsymbol{u}), \alpha, (q, \boldsymbol{u}''))$.

From the existence of the transition rule in Δ' we know that there exists $\boldsymbol{t} \in States$ such that $\boldsymbol{\varrho}(\boldsymbol{t}, \alpha) = \boldsymbol{u}$ and $\boldsymbol{\varrho}(\boldsymbol{t}, w^R) = \boldsymbol{u}'$. Provided that the automaton is consistent, we know that $\boldsymbol{\varrho}(\boldsymbol{u}'', w^R) = \boldsymbol{u}'$. Exploiting the backward determinism we get $\boldsymbol{t} = \boldsymbol{u}''$, and hence the added transition is consistent. $\diamond$

The fact that the algorithm only has to deal with consistent transitions influences the complexity analysis:

Lemma 2. *Given a consistent $\mathcal{P}'$-automaton $\mathcal{A} = (Q \times States, \Gamma, \delta, P \times States, F)$, we can compute $pre^*(L(\mathcal{A}))$ in $\mathcal{O}(|Q|^2 \cdot |\Delta| \cdot |States|)$ time and $\mathcal{O}(|Q| \cdot |\Delta| \cdot |States| + |\delta|)$ space.*

Proof. A complete proof for the general case, discussing data structures and other details is given in [6]. Here we just point out the important differences for the special case of consistent automata.

- Line 10 will be executed once for every combination of a rule $\langle (p_1, \boldsymbol{u}_1), \alpha_1 \rangle \hookrightarrow \langle (q, \boldsymbol{u}), \alpha\alpha_2 \rangle$ and a (consistent) transition $((q, \boldsymbol{u}), \alpha, (q', \boldsymbol{u}'))$. Since $\boldsymbol{u}'$ is the single state for which $\boldsymbol{\varrho}(\boldsymbol{u}', \alpha) = \boldsymbol{u}$ holds, there are $\mathcal{O}(|\Delta'| \cdot |Q|) = \mathcal{O}(|Q| \cdot |\Delta| \cdot |States|)$ such combinations. Thus, the size of Δ'' is also $\mathcal{O}(|Q| \cdot |\Delta| \cdot |States|)$.
- For the loop starting at line 11, $(q', \boldsymbol{u}')$ and α_2 (and hence $\boldsymbol{u}''$) are fixed, so line 12 is executed $\mathcal{O}(|Q|^2 \cdot |\Delta| \cdot |States|)$ times.
- Line 8 is executed once for every combination of rules $\langle (p_1, \boldsymbol{u}_1), \alpha_1 \rangle \hookrightarrow \langle (q, \boldsymbol{u}), \alpha \rangle$ in $\Delta' \cup \Delta''$ and transitions $((q, \boldsymbol{u}), \alpha, (q', \boldsymbol{u}'))$. Since the size of Δ'' is $\mathcal{O}(|Q| \cdot |\Delta| \cdot |States|)$ and $\boldsymbol{u}'$ is unique, we have $\mathcal{O}(|Q|^2 \cdot |\Delta| \cdot |States|)$ such combinations. $\diamond$

Lemma 3. *The repeating heads of the product of $\mathcal{P}'$ and $\mathcal{B}$ can be computed in $\mathcal{O}(|P|^2 \cdot |\Delta| \cdot |States| \cdot |\mathcal{B}|^3)$ time and $\mathcal{O}(|P| \cdot |\Delta| \cdot |States| \cdot |\mathcal{B}|^2)$ space.*

Proof. (analogous to [6]) The algorithm first computes the set $pre^*(\{ \langle p', \varepsilon \rangle \mid p' \in P' \times R \})$. Since this set can be represented by a consistent automaton with $|P| \times |R| \times |States|$ many states and no transitions, this step is bounded by the aforementioned limitations on time and space. From the results a head reachability graph of size $\mathcal{O}(|P| \cdot |\Delta| \cdot |States| \cdot |\mathcal{B}|^2)$ is constructed. To find the repeating heads, we identify the strongly connected components of that graph which takes time linear in its size. $\diamond$

We can now conclude the proof of Theorem 3. The steps required to solve the model-checking problems are as follows:

- Compute the set of repeating heads RH of the product of $\mathcal{P}'$ and $\mathcal{B}$. According to Lemma 3, this takes $\mathcal{O}(|P|^2 \cdot |\Delta| \cdot |States| \cdot |\mathcal{B}|^3)$ time and $\mathcal{O}(|P| \cdot |\Delta| \cdot |States| \cdot |\mathcal{B}|^2)$ space, and we have $|RH| = \mathcal{O}(|\Delta| \cdot |States| \cdot |\mathcal{B}|)$.
- Construct an automaton $\mathcal{A}$ accepting exactly the consistent subset of $RH\Gamma^*$. Take $\mathcal{A} = (((P \times R) \cup \{s\}) \times States, \Gamma, \delta, P \times \{r_0\} \times States, \{(s, \boldsymbol{s})\})$. For every repeating head $\langle (p, r, \boldsymbol{u}), \alpha \rangle$, add to δ the unique transition $((p, r, \boldsymbol{u}), \alpha, (s, \boldsymbol{u}'))$ with $\boldsymbol{\varrho}(\boldsymbol{u}', \alpha) = \boldsymbol{u}$. For every triple $(\boldsymbol{u}, \alpha, \boldsymbol{u}')$ such that $\boldsymbol{\varrho}(\boldsymbol{u}', \alpha) = \boldsymbol{u}$ add to δ the transition $((s, \boldsymbol{u}), \alpha, (s, \boldsymbol{u}'))$. There are at most $\mathcal{O}(|States| \cdot |\Gamma|) \subseteq \mathcal{O}(|\Delta| \cdot |States|)$ such triples. This automaton is consistent and has $\mathcal{O}(|P| \cdot |States| \cdot |\mathcal{B}|)$ states and $\mathcal{O}(|\Delta| \cdot |States| \cdot |\mathcal{B}|)$ transitions.

- Compute the automaton $\mathcal{A}' = (((P \times R) \cup \{s\}) \times States, \Gamma, \delta', P \times \{r_0\} \times States, \{(s, \boldsymbol{s})\})$ corresponding to $pre^*(L(\mathcal{A}))$. According to Lemma 2, this takes $\mathcal{O}(|P|^2 \cdot |\Delta| \cdot |States| \cdot |\mathcal{B}|^3)$ time and $\mathcal{O}(|P| \cdot |\Delta| \cdot |States| \cdot |\mathcal{B}|^2)$ space. (Recall that the size of $\mathcal{B}$ is also a factor in the size of the product transition rules.)
- Due to Lemma 1, $\mathcal{A}'$ is consistent and accepts only consistent configurations. According to Proposition 3.1 in [6] we have $c \not\models^{\nu'} \varphi$ for every configuration in $L(\mathcal{A}')$. According to (1), we then have $\langle p, w\rangle \not\models^{\nu'} \varphi$ for every $c = \langle (p, \boldsymbol{u}), w\rangle$ in $L(\mathcal{A}')$. Hence, we can solve the problem (II) by modifying $\mathcal{A}'$ slightly; let $\mathcal{A}''$ be the automaton $\mathcal{A}'' = (((P \times R) \cup \{s\}) \times States \cup P, \Gamma, \delta'', P, \{(s, \boldsymbol{s})\})$ where $\delta'' = \delta' \cup \{\, (p, \alpha, q) \mid ((p, r, \boldsymbol{u}), \alpha, q) \in \delta' \,\}$. Problem (I) is solved by checking whether $\langle q_0, \omega\rangle \in L(\mathcal{A}'')$. Since none of the steps required to compute $\mathcal{A}''$ takes more than $\mathcal{O}(|P|^2 \cdot |\Delta| \cdot |States| \cdot |\mathcal{B}|^3)$ time and $\mathcal{O}(|P| \cdot |\Delta| \cdot |States| \cdot |\mathcal{B}|^2)$ space, the first part of our theorem is proven.
- To prove the second part we simply need to synchronise $\mathcal{A}''$ with the automaton $\mathcal{R}$ which recognizes all reachable configurations. Computing $\mathcal{R}$ takes $\mathcal{O}(|P| \cdot |\Delta| \cdot (|P| + |\Delta|))$ time and space according to Theorem 1. The synchronization is performed as follows: For all transitions (p, α, p') of $\mathcal{A}''$ and (q, α, q') of $\mathcal{R}$ we add a transition $((p, q), \alpha, (p', q'))$ to the product. A straightforward procedure however would give us a higher result than necessary. We can do better by employing the following trick from [6]: first all transitions (q, α, q') of $\mathcal{R}$ are sorted into buckets labelled by α. Then each transition of $\mathcal{A}''$ is multiplied with the transitions in the respective bucket. As $\mathcal{R}$ has $\mathcal{O}(|P| + |\Delta|)$ states, each bucket contains $\mathcal{O}((|P| + |\Delta|)^2)$ items. Hence, the product can be computed in $\mathcal{O}(|P| \cdot |\Delta| \cdot (|P| + |\Delta|)^2 \cdot |States| \cdot |\mathcal{B}|^2)$ time and space. Alternatively, we can sort the transitions of $\mathcal{A}''$ into buckets of size $\mathcal{O}(|P|^2 \cdot |\mathcal{B}|^2 \cdot |States|)$ (exploiting the consistency of $|\mathcal{A}''|$) and construct the product in $\mathcal{O}(|P|^3 \cdot |\Delta| \cdot (|P| + |\Delta|) \cdot |States| \cdot |\mathcal{B}|^2)$ time and space. If we add the time and space which is needed to construct $\mathcal{A}''$ and $\mathcal{R}$, we get the results stated in the second part of Theorem 3. □

Technique 2 – extending the stack

An alternative approach to model-checking with regular valuations is to store the vectors of $States$ in the stack of $\mathcal{P}$. This technique works without any additional limitations, i.e., we do *not* need the assumption of backward determinism of $\mathcal{M}^p_{A_i}$ automata.

We define a pushdown system $\mathcal{P}' = (P, \Gamma', \Delta', q_0, \omega')$ where $\Gamma' = \Gamma \times States$, $\omega' = (\omega, \boldsymbol{s})$ where $\boldsymbol{s}$ is the vector of initial states of $\mathcal{M}^p_{A_i}$ automata, and the set of transition rules Δ' is determined as follows:

- $\langle p, (\alpha, \boldsymbol{r})\rangle \hookrightarrow' \langle q, \varepsilon\rangle \iff \langle p, \alpha\rangle \hookrightarrow \langle q, \varepsilon\rangle$
- $\langle p, (\alpha, \boldsymbol{r})\rangle \hookrightarrow' \langle q, (\beta, \boldsymbol{r})\rangle \iff \langle p, \alpha\rangle \hookrightarrow \langle q, \beta\rangle$
- $\langle p, (\alpha, \boldsymbol{r})\rangle \hookrightarrow' \langle q, (\beta, \boldsymbol{u})(\gamma, \boldsymbol{r})\rangle \iff \langle p, \alpha\rangle \hookrightarrow \langle q, \beta\gamma\rangle \wedge \boldsymbol{u} = \boldsymbol{\varrho}(\boldsymbol{r}, \gamma)$

Intuitively, the reason why we do not need the assumption of backward determinism in our second approach is that the stack carries complete information about the computational history of the $\mathcal{M}^p_{A_i}$ automata.

A configuration $\langle q, (\alpha_k, \boldsymbol{r(k)}) \cdots (\alpha_1, \boldsymbol{r(1)})\rangle$ is called *consistent* iff $\boldsymbol{r(1)} = \boldsymbol{s}$ and $\boldsymbol{r(j+1)} = \boldsymbol{\varrho}(\boldsymbol{r(j)}, \alpha_j)$ for all $1 \leq j < k$.

The underlying function f of the simple valuation ν' is defined by

$$f(A_i, p) = \{(\alpha, \boldsymbol{r}) \mid \alpha \in \Gamma, \varrho_i^p(\boldsymbol{r}_i^p, \alpha) \in F_i^p\}$$

It is easy to show that $\langle q, \alpha_k \cdots \alpha_1 \rangle \models^{\nu} \varphi \iff \langle q, (\alpha_k, \boldsymbol{r(k)}) \cdots (\alpha_1, \boldsymbol{r(1)})\rangle \models^{\nu'} \varphi$ where $\langle q, (\alpha_k, \boldsymbol{r(k)}) \cdots (\alpha_1, \boldsymbol{r(1)})\rangle$ is consistent.

Theorem 4. *Technique 2 (extending the stack) gives us the same bounds on the model checking problems with regular valuations as Technique 1, i.e.:*

1. *Problems (I) and (II) can be solved in $\mathcal{O}(|P|^2 \cdot |\Delta| \cdot |States| \cdot |\mathcal{B}|^3)$ time and $\mathcal{O}(|P| \cdot |\Delta| \cdot |States| \cdot |\mathcal{B}|^2)$ space.*
2. *Problem (III) can be solved in either $\mathcal{O}(|P| \cdot |\Delta| \cdot (|P| + |\Delta|)^2 \cdot |States| \cdot |\mathcal{B}|^3)$ time and $\mathcal{O}(|P| \cdot |\Delta| \cdot (|P| + |\Delta|)^2 \cdot |States| \cdot |\mathcal{B}|^2)$ space, or $\mathcal{O}(|P|^3 \cdot |\Delta| \cdot (|P| + |\Delta|) \cdot |States| \cdot |\mathcal{B}|^3)$ time and $\mathcal{O}(|P|^3 \cdot |\Delta| \cdot (|P| + |\Delta|) \cdot |States| \cdot |\mathcal{B}|^2)$ space.*

Proof. Since $|\Delta'| = |\Delta| \cdot |States|$ (here we use the fact that each $\mathcal{M}_{A_i}^p$ is deterministic), we can compute a $\mathcal{P}$-automaton $\mathcal{D} = (D, \Gamma', \gamma, P, G)$ of size $\mathcal{O}(|P| \cdot |\Delta| \cdot |States| \cdot |\mathcal{B}|^2)$ in $\mathcal{O}(|P|^2 \cdot |\Delta| \cdot |States| \cdot |\mathcal{B}|^3)$ time and $\mathcal{O}(|P| \cdot |\Delta| \cdot |States| \cdot |\mathcal{B}|^2)$ space such that $\mathcal{D}$ recognizes all configurations of $\mathcal{P}'$ which violate φ (see Theorem 1); then, to solve problem (I), we just look if $\mathcal{D}$ accepts $\langle q_0, \omega' \rangle$.

The problem with $\mathcal{D}$ is that it can also accept inconsistent configurations of $\mathcal{P}'$. Fortunately, it is possible to perform a kind of 'synchronization' with the reversed $\mathcal{M}_{A_i}^p$ automata. We define $\mathcal{A} = (Q, \Gamma, \delta, P, F)$ where $Q = (D \times States) \cup P$, $F = G \times \{\boldsymbol{s}\}$, and δ is defined as follows:

- if $g \xrightarrow{(\alpha, \boldsymbol{r})} h$ is a transition of γ, then δ contains a transition $(g, \boldsymbol{t}) \xrightarrow{\alpha} (h, \boldsymbol{r})$ where $\boldsymbol{t} = \boldsymbol{\varrho}(\boldsymbol{r}, \alpha)$;
- if $(p, \boldsymbol{r}) \xrightarrow{\alpha} (g, \boldsymbol{t})$, $p \in P$, is a transition of δ, then $p \xrightarrow{\alpha} (g, \boldsymbol{t})$ is also a transition of δ.

Notice that $\mathcal{A}$ is the same size as $\mathcal{D}$ since in every transition $\boldsymbol{t}$ is uniquely determined by $\boldsymbol{r}$ and α. Now, for every configuration $\langle p, (\alpha_k, \boldsymbol{r(k)}) \cdots (\alpha_1, \boldsymbol{r(1)})\rangle$, one can easily prove (by induction on k) that

$$\gamma(p, (\alpha_k, \boldsymbol{r(k)}) \cdots (\alpha_1, \boldsymbol{r(1)})) = q \text{ where } \boldsymbol{r(j+1)} = \boldsymbol{\varrho}(\boldsymbol{r(j)}, \alpha_j) \text{ for all } 1 \leq j < k$$

iff

$$\delta((p, \boldsymbol{r}), \alpha_k \cdots \alpha_1) = (q, \boldsymbol{r(1)}) \text{ where } \boldsymbol{r} = \boldsymbol{\varrho}(\boldsymbol{r(k)}, \alpha_k).$$

From this we immediately obtain that $\mathcal{A}$ indeed accepts exactly those configurations of $\mathcal{P}$ which violate φ. Moreover, the size of $\mathcal{A}$ and the time and space bounds to compute it are the same as for $\mathcal{D}$ which proves the first part of the theorem.

To solve problem (III), one can try out the same strategies as in Theorem 3. Again, it turns out that the most efficient way is to synchronize $\mathcal{A}$ with the $\mathcal{P}$-automaton $\mathcal{R}$ which recognizes all reachable configurations of $\mathcal{P}$. Employing the same trick as in Theorem 3 (i.e., sorting transitions of $\mathcal{R}$ into buckets according to their labels), we obtain that the size of $\mathcal{A}'$ is $\mathcal{O}(|P| \cdot |\Delta| \cdot (|P| + |\Delta|)^2 \cdot |States| \cdot |\mathcal{B}|^2)$ and it can be computed in $\mathcal{O}(|P| \cdot |\Delta| \cdot (|P| + |\Delta|)^2 \cdot |States| \cdot |\mathcal{B}|^3)$ time using $\mathcal{O}(|P| \cdot |\Delta| \cdot (|P| + |\Delta|)^2 \cdot |States| \cdot |\mathcal{B}|^2)$ space. Using the alternative method (sorting transitions of $\mathcal{A}$ into buckets instead and exploiting determinism) we get an automaton of size $\mathcal{O}(|P|^3 \cdot |\Delta| \cdot (|P| + |\Delta|) \cdot |States| \cdot |\mathcal{B}|^2)$ in $\mathcal{O}(|P|^3 \cdot |\Delta| \cdot (|P| + |\Delta|) \cdot |States| \cdot |\mathcal{B}|^3)$ in time and $\mathcal{O}(|P|^3 \cdot |\Delta| \cdot (|P| + |\Delta|) \cdot |States| \cdot |\mathcal{B}|^2)$ space. $\square$

4 Applications

4.1 Interprocedural Data-Flow Analysis

Pushdown systems provide a very natural formal model for programs with recursive procedures. Hence, it should not be surprising that efficient analysis techniques for pushdown automata can be applied to some problems of interprocedural data-flow analysis (see, e.g., [7, 11]). Here we briefly discuss the convenience of regular valuations in this application area. We do not present any detailed results about the complexity of concrete problems, because this would necessarily lead to a quite complicated and lengthy development which is beyond the scope of our work (though the associated questions are very interesting on their own). Our aim is just to provide convincing arguments demonstrating the importance of the technical results achieved in Section 3.

A standard way of abstracting recursive programs for purposes of interprocedural data-flow analysis it to represent each procedure P by its associated *flow graph*. Intuitively, the flow graph of P is a labelled binary graph whose nodes correspond to 'program points', and an arc $n \xrightarrow{c} n'$ indicates that the control flow is shifted from the point n to n' by performing the instruction c. The entry and exit points of P are modeled by distinguished nodes. To avoid undecidabilities, the `if-then-else` command (and related instructions) are modeled by nondeterminism, i.e., there can be several arcs from the same node. Moreover, there are special arcs with labels of the form $call\ Q(args)$ which model procedure calls (where $args$ is a vector of terms which are passed as parameters). Flow graphs can be easily translated to pushdown systems; as transitions of pushdown systems are not labelled, we first perform a 'technical' modification of the flow graph, replacing each arc $n \xrightarrow{c} n'$ where n is a nondeterministic node (i.e., a node with more than one successor) by two arcs $n \xrightarrow{\varepsilon} n'' \xrightarrow{c} n'$ where n'' is a new node and ε is a 'dummy' instruction without any effect. This allows to associate the instruction of each arc $n \xrightarrow{c} n'$ directly to n (some nodes are associated to the dummy instruction). Now suppose there is a recursive system of procedures $P_1, \ldots, P_n, S$, where S is a distinguished starting procedure which cannot be called recursively. Their associated flow graphs can be translated to a pushdown automaton in the following way:

- for each node n of each flowgraph we introduce a fresh stack symbol X_n;
- for each arc of the form $n \xrightarrow{c} n'$ we add a rule $\langle \cdot, X_n \rangle \hookrightarrow \langle \cdot, X_{n'} \rangle$, where $\cdot$ is the (only) control location;
- for each arc of the form $n \xrightarrow{call\ Q(args)} n'$ we add the rule $\langle \cdot, X_n \rangle \hookrightarrow \langle \cdot, Q_{entry} X_{n'} \rangle$ where Q_{entry} is the stack symbol for the entry node of (the flow graph of) Q. Observe that one can also push special symbols corresponding to arguments if needed;

– for each procedure P different from S we add the rule $\langle\cdot, P_{exit}\rangle \hookrightarrow \langle\cdot, \varepsilon\rangle$, where P_{exit} corresponds to the exit node of P. For the starting procedure S we add the rule $\langle\cdot, S_{exit}\rangle \hookrightarrow \langle\cdot, S_{exit}\rangle$.

In other words, the top stack symbol corresponds to the current program point (and to the instruction which is to be executed), and the stack carries the information about the history of activation calls. Now, many of the well-known properties of data-flow analysis (e.g., liveness, reachability, very business, availability) can be expressed in LTL and verified by a model-checking algorithm (in some cases the above presented construction of a pushdown automaton must be modified so that all necessary information is properly reflected – but the principle is still the same). For example, if we want to check that a given variable Y is *dead* at a given program point n (i.e., whenever the program point n is executed, in each possible continuation we have that Y is either not used or it is redefined before it is used), we can model-check the formula

$$\Box(top_n \Longrightarrow ((\neg used_Y \,\mathcal{U}\, def_Y) \;\vee\; (\Box\neg used_Y)))$$

in the configuration $\langle\cdot, S_{entry}\rangle$, where $used_Y$, $used_Y$, and def_Y are atomic propositions which are valid in exactly those configurations where the topmost stack symbol corresponds to the program point n, to an instruction which uses the variable Y, or to an instruction which defines Y, respectively. Even in this simple example, we can see that regular valuations are indeed useful – if we have a language with dynamic scoping (e.g., LISP), we *cannot* resolve to which Y the instruction $Y := 3$ at a program point n refers to without examining the stack of activation records (the Y refers to a local variable Y of the topmost procedure in the stack of activation records which declares it). So, $used_Y$ and def_Y would be interpreted by regular valuations in this case.

The example above is quite simple. The 'real' power of regular valuations would become apparent in a context of more complicated problems where we need to examine complex relationships among dynamically gathered pieces of information. This is one of the subjects of intended future work.

4.2 Pushdown Systems with Checkpoints

Another area where the results of Section 3.1 find a natural application is the analysis of recursive computations with local security checks. Modern programming languages contain methods for performing run-time inspections of the stack of activation records, and processes can thus take dynamic decisions based on the gathered information. An example is the class `AccessController` implemented in Java Development Kit 1.2 offering the method `checkPermission` which checks whether all methods stored in the stack are granted a given permission. If not, the method rises an exception.

We propose a (rather general) formal model of such systems called *pushdown system with checkpoints*. Our work was inspired by the paper [10] which deals with the same problem. Our model is more general, however. The model of [10] is suitable only for checking safety properties, does not model data-flow, and forbids mutually recursive procedure calls whereas our model has none of these restrictions. Properties of pushdown systems with checkpoints can be expressed in LTL and we provide an efficient model-checking algorithm for LTL with regular valuations.

Definition 4. *A* pushdown system with checkpoints *is a triple* $\mathcal{C} = (\mathcal{P}, \xi, \eta)$ *where*

- $\mathcal{P} = (P, \Gamma, \Delta, q_0, \omega)$ *is a pushdown system,*
- ξ *is a function with domain* $\mathcal{D} \subseteq P \times \Gamma$ *which assigns to each pair* $(p, \alpha) \in \mathcal{D}$ *a deterministic finite-state automaton* $\mathcal{M}^p_\alpha = (Q^p_\alpha, \Gamma, \delta^p_\alpha, s^p_\alpha, F^p_\alpha)$*. For technical convenience, we assume that* δ^p_α *is total,* $s^p_\alpha \notin F^p_\alpha$*, and* $L(\mathcal{M}^p_\alpha) \subseteq \{w\alpha \mid w \in \Gamma^*\}$*. Elements of* $\mathcal{D}$ *are called* checkpoints.
- $\eta \colon \Delta \to \{+, -, 0\}$ *is a function which partitions the set of transition rules into* positive, negative, *and* independent *ones. We require that if* (p, α) *is* not *a checkpoint, then all rules of the form* $\langle p, \alpha \rangle \hookrightarrow \langle q, v \rangle$ *are independent.*

The function η determines whether a rule can be applied when an inspection of the stack at a checkpoint yields a positive or negative result, or whether it is independent of such tests. Using positive and negative rules, we can model systems which perform `if-then-else` commands where the condition is based on dynamic checks; hence, these checks can be nested to an arbitrary level. The fact that a rule $\langle p, \alpha \rangle \hookrightarrow \langle q, v \rangle$ is positive, negative, or independent is denoted by $\langle p, \alpha \rangle \hookrightarrow^+ \langle q, v \rangle$, $\langle p, \alpha \rangle \hookrightarrow^- \langle q, v \rangle$, or $\langle p, \alpha \rangle \hookrightarrow^0 \langle q, v \rangle$, respectively.

To $\mathcal{C}$ we associate a unique transition system $\mathcal{T}_\mathcal{C}$ where the set of states is the set of all configurations of $\mathcal{P}$, $\langle q_0, \omega \rangle$ is the root, and the transition relation is the least relation $\to$ satisfying the following:

- if $\langle p, \alpha \rangle \hookrightarrow^+ \langle q, v \rangle$, then $\langle p, \alpha w \rangle \to \langle q, vw \rangle$ for all $w \in \Gamma^*$ s.t. $w^R\alpha \in L(\mathcal{M}^p_\alpha)$;
- if $\langle p, \alpha \rangle \hookrightarrow^- \langle q, v \rangle$, then $\langle p, \alpha w \rangle \to \langle q, vw \rangle$ for all $w \in \Gamma^*$ s.t. $w^R\alpha \notin L(\mathcal{M}^p_\alpha)$;
- if $\langle p, \alpha \rangle \hookrightarrow^0 \langle q, v \rangle$, then $\langle p, \alpha w \rangle \to \langle q, vw \rangle$ for all $w \in \Gamma^*$.

Some natural problems for pushdown processes with checkpoints are listed below.

- The reachability problem: given a pushdown system with checkpoints, is a given configuration reachable?
- The checkpoint-redundancy problem: given a pushdown system with checkpoints and a checkpoint (p, α), is there a reachable configuration where the checkpoint (p, α) is (or is not) satisfied?
  This problem is important because redundant checkpoints can be safely removed together with all negative (or positive) rules, declaring all remaining rules as independent. Thus, one can decrease the runtime overhead.
- The global safety problem: given a pushdown system with checkpoints and a formula φ of LTL, do all reachable configurations satisfy φ?
  An efficient solution to this problem allows to make 'experiments' with checkpoints with the aim of finding a solution with a minimal runtime overhead.

Actually, it is quite easy to see that all these problems (and many others) can be encoded by LTL formulae and regular valuations. For example, to solve the reachability problem, we take a predicate A which is satisfied only by the configuration $\langle p, w \rangle$ whose reachability is in question (the associated automaton $\mathcal{M}^p_A$ has $length(w)$ states) and then we check the formula $\Box(\neg A)$. Observe that this formula in fact says that $\langle p, w \rangle$ is *unreachable*; the reachability itself is not directly expressible in LTL (we can only say that $\langle p, w \rangle$ is reachable in *every run*). However, it does not matter because we can simply negate the answer of the model-checking algorithm.

Model-Checking LTL for Pushdown Systems with Checkpoints Let $\mathcal{C} = (\mathcal{P}, \xi, \eta)$ be a pushdown system with checkpoints, where $\mathcal{P} = (P, \Gamma, \Delta, q_0, \omega)$. We define a pushdown system $\mathcal{P}' = (P \times \{+, -, 0\}, \Gamma, \Delta', (q_0, 0), \omega)$ where Δ' is the least set of rules satisfying the following (for each $x \in \{+, -, 0\}$);

- if $\langle p, \alpha\rangle \hookrightarrow^{+} \langle q, v\rangle \in \Delta$, then $\langle (p, x), \alpha\rangle \hookrightarrow \langle (q, +), v\rangle \in \Delta'$;
- if $\langle p, \alpha\rangle \hookrightarrow^{-} \langle q, v\rangle \in \Delta$, then $\langle (p, x), \alpha\rangle \hookrightarrow \langle (q, -), v\rangle \in \Delta'$;
- if $\langle p, \alpha\rangle \hookrightarrow^{0} \langle q, v\rangle \in \Delta$, then $\langle (p, x), \alpha\rangle \hookrightarrow \langle (q, 0), v\rangle \in \Delta'$.

Intuitively, $\mathcal{P}'$ behaves in the same way as the underlying pushdown system $\mathcal{P}$ of $\mathcal{C}$, but it also 'remembers' what kind of rule (positive, negative, independent) was used to enter the current configuration.

Let ν be a regular valuation for configurations of $\mathcal{C}$ with an underlying function f (see Definition 3), and let φ be an LTL formula. Let Check, Neg, and Pos be fresh atomic propositions which do not appear in φ. We define a function f', which is the underlying function for a regular valuation ν' for configurations of $\mathcal{P}'$, as follows:

- if $A \in \mathit{At} \setminus \{\mathit{Check}, \mathit{Neg}, \mathit{Pos}\}$, then $f'(A, (p, x)) = f(A, p)$ for $x \in \{+, -, 0\}$
- $f'(\mathit{Check}, (p, x)) = \mathcal{M}$ for $x \in \{+, -, 0\}$, where $\mathcal{M}$ is the product automaton accepting $\bigcup_{\alpha \in \mathcal{S}} L(\mathcal{M}^p_\alpha)$ where $\mathcal{S} = \{\, \alpha \mid (p, \alpha) \text{ is a checkpoint} \,\}$
- $f'(\mathit{Neg}, (p, -)) = f'(\mathit{Pos}, (p, +)) = \mathcal{M}$, where $\mathcal{M}$ is a one-state automaton accepting Γ^*.
- $f'(\mathit{Neg}, (p, 0)) = f'(\mathit{Neg}, (p, +)) = f'(\mathit{Pos}, (p, -)) = f'(\mathit{Pos}, (p, 0)) = \mathcal{M}$, where $\mathcal{M}$ is a one-state automaton accepting $\emptyset$.

Now we can readily confirm the following:

Theorem 5. *Let $\langle p, w\rangle$ be a configuration of $\mathcal{C}$. We have that*

$$\langle p, w\rangle \models^{\nu} \varphi \iff \langle (p, 0), w\rangle \models^{\nu'} \psi \Longrightarrow \varphi$$

where $\psi \equiv \Box((\mathit{Check} \Longrightarrow \mathcal{X}(\neg \mathit{Neg})) \wedge (\neg \mathit{Check} \Longrightarrow \mathcal{X}(\neg \mathit{Pos})))$.

Proof. It suffices to realize that

$$\langle p, w\rangle \equiv \langle p_0, w_0\rangle \to \langle p_1, w_1\rangle \to \langle p_2, w_2\rangle \to \langle p_3, w_3\rangle \to \cdots$$

is an infinite path in $\mathcal{T}_{\mathcal{C}}$ iff

$$\langle (p, 0), w\rangle \equiv \langle (p_0, x_0), w_0\rangle \to \langle (p_1, x_1), w_1\rangle \to \langle (p_2, x_2), w_2\rangle \to \cdots$$

is an infinite path in $\mathcal{T}_{\mathcal{P}'}$ satisfying ψ (where each x_i for $i > 0$ is either $+$, $-$, or 0; realize that all x_i are determined uniquely). Indeed, ψ ensures that all transitions in the latter path are 'consistent' with possible checkpoints in the former path. As all atomic propositions which appear in φ are evaluated identically for pairs $\langle p_i, w_i\rangle$, $\langle (p_i, x_i), w_i\rangle$ (see the definition of f' above), we can conclude that both paths either satisfy or do not satisfy φ. $\square$

The previous theorem in fact says that the model-checking problem for LTL and pushdown systems with check-points can be reduced to the model-checking problem for LTL and 'ordinary' pushdown systems. As the formula ψ is fixed and the atomic propositions Check, Neg, and Pos are regular, we can evaluate the complexity bounds for the resulting model-checking algorithm using the results of Section 3.1. Let $\{A_1, \ldots, A_n\}$ be the set of all atomic propositions which appear in φ, and let $\mathcal{N} = \{M_1, \ldots, \mathcal{M}_m\}$ be the set of all $\mathcal{M}^p_{A_i}$ automata. Let States be the Cartesian product of the sets of states of all $\mathcal{M}^p_\alpha$ automata and the automata of $\mathcal{N}$. Let $\mathcal{B}$ be a Büchi automaton which corresponds to $\neg\varphi$. Now we can state our theorem (remember that P is the set of control states and Δ the set of rules of the underlying pushdown system $\mathcal{P}$ of $\mathcal{C}$).

Theorem 6. *We have the following bounds on the model checking problems for LTL with regular valuations and pushdown systems with checkpoints:*

1. *Problems (I) and (II) can be solved in $\mathcal{O}(|P|^2 \cdot |\Delta| \cdot |\mathit{States}| \cdot |\mathcal{B}|^3)$ time and $\mathcal{O}(|P| \cdot |\Delta| \cdot |\mathit{States}| \cdot |\mathcal{B}|^2)$ space.*
2. *Problem (III) can be solved in either $\mathcal{O}(|P| \cdot |\Delta| \cdot (|P| + |\Delta|)^2 \cdot |\mathit{States}| \cdot |\mathcal{B}|^3)$ time and $\mathcal{O}(|P| \cdot |\Delta| \cdot (|P| + |\Delta|)^2 \cdot |\mathit{States}| \cdot |\mathcal{B}|^2)$ space, or $\mathcal{O}(|P|^3 \cdot |\Delta| \cdot (|P| + |\Delta|) \cdot |\mathit{States}| \cdot |\mathcal{B}|^3)$ time and $\mathcal{O}(|P|^3 \cdot |\Delta| \cdot (|P| + |\Delta|) \cdot |\mathit{States}| \cdot |\mathcal{B}|^2)$ space.*

Proof. We apply Theorem 5, which says that we can equivalently consider the model-checking problem for the pushdown system $\mathcal{P}'$, formula $\psi \implies \varphi$, and valuation ν'. First, let us realize that the Büchi automaton which corresponds to $\neg(\psi \implies \varphi)$ can be actually obtained by 'synchronizing' $\mathcal{B}$ with the Büchi automaton for ψ, because $\neg(\psi \implies \varphi) \equiv \psi \wedge \neg\varphi$. As the formula ψ is *fixed*, the synchronization increases the size of $\mathcal{B}$ just by a constant factor. Hence, the automaton for $\neg(\psi \implies \varphi)$ is asymptotically of the same size of $\mathcal{B}$. The same can be said about the sizes of P' and P, and about the sizes of Δ' and Δ. Moreover, if we collect all automata in the range of f' (see above) and consider the state space of their product, we see that it has *exactly* the size of States because the automata associated to Pos and Neg have only one state. □

4.3 Model-checking CTL*

In this section, we apply the model-checking algorithm to the logic CTL* which extends LTL with existential path quantification [4]. More precisely, CTL* formulae are built according to the following abstract syntax equation:

$$\varphi ::= \mathtt{tt} \mid A \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \mathcal{E}\varphi \mid \mathcal{X}\varphi \mid \varphi_1 \,\mathcal{U}\, \varphi_2$$

where A ranges over the atomic propositions (interpreted, say, by a regular valuation represented by a finite automaton of size $|\mathit{States}|$).

For finite-state systems, model-checking CTL* can be reduced to checking LTL as follows [5]: For a CTL* formula φ, call the *path depth* of φ the maximal nesting depth of existential path quantifiers within φ. Subformulae of φ can be checked in ascending order of path depth; subformulae of the form $\mathcal{E}\,\varphi'$ where φ' is $\mathcal{E}$-free are checked with an LTL algorithm which returns the set of states $S_{\varphi'}$ satisfying φ'. Then $\mathcal{E}\,\varphi'$ is replaced by a fresh atomic proposition whose valuation yields true exactly on $S_{\varphi'}$, and the procedure is repeated for subformulae of higher path depth. The method can be transferred to the case of pushdown systems; running the LTL algorithm on $\mathcal{E}\,\varphi'$ returns an automaton $\mathcal{M}_{\varphi'}$. We can then replace $\mathcal{E}\,\varphi'$ by a fresh atomic proposition whose valuation is given by $\mathcal{M}_{\varphi'}$. This method was already proposed in [9], but without any complexity analysis.

Let us review the complexity of this procedure. For the rest of this subsection fix a pushdown system $\mathcal{P} = (P, \Gamma, \Delta, q_0, \omega)$. Given an $\mathcal{E}$-free formula φ, let $\mathcal{B} = (R, 2^{At}, \eta, r_0, G)$ be a Büchi automaton corresponding to φ, and let $|States|$ be the size of the $\mathcal{M}^p_{A_i}$ automata encoding the regular valuations of propositions in At.

The algorithms from section 3.1 (in general we can only use Technique 2) yield an automaton $\mathcal{M}_\varphi$ which accepts exactly the configurations satisfying $\mathcal{E}\,\varphi$. Observe that $\mathcal{M}_\varphi$ is non-deterministic, reads the stack top-down, and has $\mathcal{O}(|P|\cdot|\mathcal{B}|\cdot|States|)$ states. We need to modify the automaton before we can use it as an encoding for the regular valuation of $\mathcal{E}\,\varphi$. More precisely, we need to reverse the automaton (i.e. make it read the stack bottom-up) and then determinise it. Reversal does not increase the size, and due to the determinism of $\mathcal{M}^p_{A_i}$ (in bottom-up direction) the determinisation explodes only the '$P \times R$ part' of the states, i.e. we get an automaton $\mathcal{M}'_\varphi$ of size $O(|States| \cdot 2^{|P|\cdot|R|})$.

To check subformulae of higher path depth we replace $\mathcal{E}\,\varphi$ by a fresh atomic proposition A_φ. With (A_φ, p) we associate the automaton $\mathcal{M}^p_{A_\varphi}$ which is a copy of $\mathcal{M}'_\varphi$ where the set F^p_φ of accepting states is taken as $\{\, (q,s) \mid q \in 2^{P\times R}, q \ni (p, r_0), s \in States \,\}$. The cross product of these new automata with the 'old' $\mathcal{M}^p_{A_i}$ automata takes only $O(|States| \cdot 2^{|P|\cdot|R|})$ states again; we need just one copy of the new automaton, and all reachable states are of the form $((q,s),s)$ where $q \in 2^{P\times R}$ and $s \in States$.

As we go up in path depth, we can repeat this procedure: First we produce a deterministic valuation automaton by taking the cross product of the automata corresponding the atomic propositions and those derived from model-checking formulae of lower path depth. Then we model-check the subformula currently under consideration, and reverse and determinise the resulting automaton. By the previous arguments, each determinisation only blows up the non-deterministic part of the automaton, i.e. after each stage the size of the valuation automaton increases by a factor of $2^{|P|\cdot|\mathcal{B}_i|}$ where $\mathcal{B}_i$ is a Büchi automaton for the subformula currently under consideration.

With this in mind, we can compute the complexity for formulae of arbitrary path depth. Let $\mathcal{B}_1, \ldots, \mathcal{B}_n$ be the Büchi automata corresponding to the individual subformulae of a formula φ. Adding the times for checking the subformulas and using Theorem 4 we get that the model-checking procedure takes at most

$$\mathcal{O}\Big(|P|^2 \cdot |\Delta| \cdot |States| \cdot 2^{|P|\cdot\Sigma_{i=1}^n |\mathcal{B}_i|} \cdot \sum_{i=1}^n |\mathcal{B}_i|^3\Big)$$

time and

$$\mathcal{O}\Big(|P| \cdot |\Delta| \cdot |States| \cdot 2^{|P|\cdot\Sigma_{i=1}^n |\mathcal{B}_i|} \cdot \sum_{i=1}^n |\mathcal{B}_i|^2\Big)$$

space. The algorithm hence remains linear in both $|\Delta|$ and $|States|$. The algorithm of Burkart and Steffen [3], applied to CTL* formulae which are in the second level of the alternation hierarchy, would yield an algorithm which is cubic in $|\Delta|$. On the other hand, the performance of our algorithm in terms of the formula is less clear. In practice, it would depend strongly on the size of the Büchi automata for the subformulae, and on the result of the determinisation procedures.

5 Lower Bounds

In previous sections we established reasonably-looking upper bounds for the model-checking problem for pushdown systems (first without and then also with checkpoints) and LTL with regular valuations. However, the algorithms are polynomial in $|\mathcal{P}| + |States| + |\mathcal{B}|$, and not in the size of *problem instance* which is (as we already mentioned in Remark 1) $|\mathcal{P}| + |\nu| + |\varphi|$. In Remark 1 we also explained *why* we use these parameters – it has been argued that typical formulae (and their associated Büchi automata) are small, hence the size of $\mathcal{B}$ is actually more relevant (a model-checking algorithm whose complexity is exponential *just* due to the blowup caused by the transformation of φ into $\mathcal{B}$ is usually efficient in practice). The same can be actually said about $|States|$ – in Section 4 we have seen that there are interesting practical problems where the size of $|States|$ does not explode. Nevertheless, from the point of view of worst-case analysis (where we measure the complexity in the size of problem instance) our algorithms are *exponential*. A natural question is whether this exponential blowup is *indeed necessary*, i.e., whether we could (in principle) solve the model-checking problems more efficiently by some 'better' technique. In this section we show it is *not* the case, because all of the considered problems are **EXPTIME**-hard (even in rather restricted forms).

We start with the natural problems for pushdown systems with checkpoints mentioned in the previous section (the reachability problem, the checkpoint redundancy problem, etc.) All of them can be (polynomially) reduced to the model-checking problem for pushdown systems with checkpoints and LTL with regular valuations and therefore are solvable in **EXPTIME**. The next theorem says that this strategy is essentially optimal, because even the reachability problem provably requires exponential time.

Theorem 7. *The reachability problem for pushdown systems with checkpoints (even for those with just three control states and no negative rules) is **EXPTIME**-complete.*

Proof. The membership to **EXPTIME** follows from Theorem 6. We show **EXPTIME**-hardness by reduction from the acceptance problem for alternating LBA (which is known to be **EXPTIME**-complete). An *alternating LBA* is a tuple $\mathcal{M} = (Q, \Sigma, \delta, q_0, \vdash, \dashv, p)$ where $Q, \Sigma, \delta, q_0, \vdash$, and $\dashv$ are defined as for ordinary non-deterministic LBA ($\vdash$ and $\dashv$ are the left-end and right-end markers, resp.), and $p\colon Q \to \{\forall, \exists, acc, rej\}$ is a function which partitions the states of Q into *universal*, *existential*, *accepting*, and *rejecting*, respectively. We assume (w.l.o.g.) that δ is defined so that 'terminated' configurations (i.e., the ones from which there are no further computational steps) are exactly accepting and rejecting configurations. Moreover, we also assume that $\mathcal{M}$ always halts and that its branching degree is 2 (i.e., each configuration has at most two immediate successors). A *computational tree* for $\mathcal{M}$ on a word $w \in \Sigma^*$ is any (finite) tree T satisfying the following: the root of T is (labeled by) the initial configuration $q_0{\vdash}w{\dashv}$ of $\mathcal{M}$, and if N is a node of $\mathcal{M}$ labelled by a configuration uqv where $u, v \in \Sigma^*$ and $q \in Q$, then the following holds:

- if q is accepting or rejecting, then N is a leaf;
- if q is existential, then N has one successor whose label is some configuration which can be reached from uqv in one computational step (according to δ);

– if q is universal, then N has m successors where $m \leq 2$ is the number of configurations which can be reached from uqv in one step; those configurations are used as labels of the successors in one-to-one fashion.

$\mathcal{M}$ *accepts* w iff there is a computational tree T such that all leaves of T are accepting configurations.

Now we describe a polynomial algorithm which for a given alternating LBA $\mathcal{M} = (Q, \Sigma, \delta, q_0, \vdash, \dashv, p)$ and a word $w \in \Sigma^*$ of length n constructs a pushdown system with checkpoints $\mathcal{C} = (\mathcal{P}, \xi, \eta)$ and its configuration $\langle a, \omega\rangle$ such that $\langle a, \omega\rangle$ is reachable from the initial configuration of $\mathcal{C}$ iff $\mathcal{M}$ accepts w. Intuitively, the underlying system $\mathcal{P}$ of $\mathcal{C}$ simulates the execution of $\mathcal{M}$ and checkpoints are used to verify that there is no cheating during the process. We start with the definition of $\mathcal{P}$, putting $\mathcal{P} = (\{g, a, r\}, \Gamma, \Delta, g, \omega)$ where

– $\Gamma = \Sigma \times (Q \cup \{-\}) \cup \{\beta_1, \cdots, \beta_{n+3}\} \cup \{\gamma_1, \cdots, \gamma_n\} \cup \{\#_1^e, \#_2^e, \#_1^u, \#_2^u, A, R, \omega\}$
– Δ contains the following (families of) rules:
  1. $\langle g, \omega\rangle \hookrightarrow \langle g, \beta_1\omega\rangle$
  2. $\langle g, \beta_i\rangle \hookrightarrow \langle g, \beta_{i+1}\varrho\rangle$ for all $1 \leq i \leq n+2$ and $\varrho \in \Sigma \times (Q \cup \{-\})$
  3. $\langle g, \beta_{n+3}\rangle \hookrightarrow \langle g, \gamma_1\varrho\rangle$ for every $\varrho \in \{\#_1^e, \#_1^u, A, R\}$
  4. $\langle g, \gamma_i\rangle \hookrightarrow \langle g, \gamma_{i+1}\rangle$ for every $1 \leq i \leq n-1$
  5. $\langle g, \gamma_n\rangle \hookrightarrow \langle g, \varepsilon\rangle$
  6. $\langle g, A\rangle \hookrightarrow \langle a, \varepsilon\rangle$, $\langle g, R\rangle \hookrightarrow \langle r, \varepsilon\rangle$, $\langle g, \#_1^e\rangle \hookrightarrow \langle g, \beta_1\#_1^e\rangle$, $\langle g, \#_1^u\rangle \hookrightarrow \langle g, \beta_1\#_1^u\rangle$
  7. $\langle a, \varrho\rangle \hookrightarrow \langle a, \varepsilon\rangle$ for every $\varrho \in \Sigma \times (Q \cup \{-\})$
  8. $\langle a, \#_1^u\rangle \hookrightarrow \langle g, \beta_1\#_2^u\rangle$, $\langle a, \#_2^u\rangle \hookrightarrow \langle a, \varepsilon\rangle$, $\langle a, \#_1^e\rangle \hookrightarrow \langle a, \varepsilon\rangle$, $\langle a, \#_2^e\rangle \hookrightarrow \langle a, \varepsilon\rangle$
  9. $\langle r, \varrho\rangle \hookrightarrow \langle r, \varepsilon\rangle$ for every $\varrho \in \Sigma \times (Q \cup \{-\})$
  10. $\langle r, \#_1^e\rangle \hookrightarrow \langle g, \beta_1\#_2^e\rangle$, $\langle r, \#_2^e\rangle \hookrightarrow \langle r, \varepsilon\rangle$, $\langle r, \#_1^u\rangle \hookrightarrow \langle r, \varepsilon\rangle$, $\langle r, \#_2^u\rangle \hookrightarrow \langle r, \varepsilon\rangle$

Intuitively, the execution of $\mathcal{P}$ starts by entering the state $\langle g, \beta_1\omega\rangle$ (rule 1). Then, exactly $n+2$ symbols of $\Sigma \times (Q \cup \{-\})$ are pushed to the stack; the compound symbol $(X, t) \in \Sigma \times (Q \cup \{-\})$ indicates that the tape contains the symbol X and, if $t \in Q$, that the head is at this position and the control state of $\mathcal{M}$ is t; if $t = -$ it means that the head is elsewhere. During this process, the family of β_i symbols is used as a 'counter' (rules 2). Realize that the word w of length n is surrounded by the '$\vdash$' and '$\dashv$' markers, so the total length of the configuration is $n+2$. The last symbol β_{n+3} is then rewritten to $\gamma_1\varrho$, where ϱ is one of $\#_1^e, \#_1^u, A, R$ (rules 3). The purpose of ϱ is to keep information about the just stored configuration (whether it is existential, universal, accepting, or rejecting) and the index of a rule which is to be used to obtain the next configuration (always the first one; remember that accepting and rejecting configurations are terminal). After that, γ_1 is successively rewritten to all of the γ_i symbols and disappears (rules 4,5). Their only purpose is to invoke several consistency checks – as we shall see, each pair (g, γ_i) is a checkpoint and all rules of 4,5 are positive. Depending on the previously stored ϱ (i.e., on the type of the just pushed configuration), we either continue with guessing the next one, or change the control state to a or r (if the configuration is accepting or rejecting, resp.) Hence, the guessing goes on until we end up with an accepting or rejecting configuration. This must happen eventually, because $\mathcal{M}$ always halts. If we find an accepting configuration, we successively remove all existential configurations and those universal configuration for which we have already checked both successors.

If we find a universal configuration with only one successor checked – it is recognized by the '$\#_1^u$' symbol – we change '$\#_1^u$' to '$\beta_1\#_2^u$' and check the other successor (rules 7 and 8). Similar things are done when a rejecting configuration is found. The control state is switched to r and then we remove all configurations until we (possibly) find an existential configuration for which we can try out the other successor (rules 9 and 10). We see that w is accepted by $\mathcal{M}$ iff we eventually pop the initial configuration when the control state is 'a', i.e., iff the state $\langle a, \omega\rangle$ is reachable.

To make all that work we must ensure that $\mathcal{P}$ cannot gain anything by 'cheating', i.e., by pushing inconsistent sequences of symbols which do *not* model a computation of $\mathcal{M}$ in the described way. This is achieved by declaring all pairs (g, γ_i) for $1 \leq i \leq n+1$ as checkpoints. The automaton $\mathcal{M}^g_{\gamma_i}$ for $1 \leq i \leq n$ accepts those words of the form $\omega v_1 \varrho_1 v_2 \varrho_2 \cdots v_m \varrho_m \gamma_i$, where $\mathit{length}(v_j) = n+2$, $\varrho_j \in \#_1^e, \#_2^e, \#_1^u, \#_2^u, A, R$ for every $1 \leq j \leq m$, such that the triples of symbols at positions $i, i+1, i+2$ in each pair of successive substrings v_k, v_{k+1} are consistent with the symbol ϱ_k w.r.t. the transition function δ of $\mathcal{M}$ (if some configuration has only one immediate successor, then $\mathcal{M}^g_{\gamma_i}$ 'ignores' the rule index stored in ϱ_k). Furthermore, the first configuration must be the initial one, and the last configuration v_m must be consistent with ϱ_m. Observe that $\mathcal{M}^g_{\gamma_i}$ needs just $\mathcal{O}(|\mathcal{M}|^6)$ states to store the two triples (after checking subwords v_k, ϱ_k, v_{k+1}, the triple of v_k is 'forgotten') the initial configuration, a 'counter' of capacity $n+2$, and some auxiliary information. Moreover, $\mathcal{M}^g_{\gamma_i}$ is deterministic and we can also assume that its transition function is total. As all rules associated with checkpoints are positive, any cheating move eventually results in entering a configuration where the system 'gets stuck', i.e., cheating cannot help to reach the configuration $\langle a, \omega\rangle$. $\square$

From the (technical) proof of Theorem 7 we can easily deduce the following:

Theorem 8. *The model-checking problem (I) for pushdown systems with checkpoints (even for those with just three control states and no negative rules) is* ***EXPTIME****-complete even for a fixed LTL formula* $\Box(\neg \mathit{fin})$ *where fin is an atomic predicate interpreted by a* simple *valuation* ν.

Proof. Let us consider the pushdown system with checkpoints $\mathcal{C} = (\mathcal{P}, \xi, \eta)$ constructed in the proof of Theorem 7. To ensure that each finite path in $\mathcal{T}_\mathcal{C}$ is a prefix of some run, we extend the set of transition rules of Δ by a family of independent rules of the form $\langle s, \alpha\rangle \hookrightarrow \langle s, \alpha\rangle$ for each control state s and each stack symbol α. Now it suffices to realize that the initial configuration $\langle g, \omega\rangle$ cannot reach the state $\langle a, \omega\rangle$ iff it cannot reach *any* state of the form $\langle a, \omega v\rangle$ (where $v \in \Gamma^*$) iff $\langle g, \omega\rangle \models^\nu \Box(\neg \mathit{fin})$ where ν is a simple valuation with the underlying function f such that $f(\mathit{fin}) = \{(a, \omega)\}$. $\square$

Hence, model-checking LTL for pushdown systems with checkpoints is **EXPTIME**-complete even for *simple* valuations.

Now we analyze the complexity of model-checking with (ordinary) pushdown systems and LTL formulae with regular valuations. First, realize that if we take any fixed formula and a subclass of pushdown systems where the number of control states is bounded by some constant, the model-checking problem is decidable in *polynomial* time. Now we prove that if the number of control states is not bounded, the model-checking problem becomes **EXPTIME**-complete even for a fixed formula. At this point, one is tempted to apply Theorem 5 to the formula $\Box(\neg \mathit{fin})$ of Theorem 8. Indeed, it allows to reduce the model-checking problem for pushdown systems with checkpoints and $\Box(\neg \mathit{fin})$ to the model-checking problem for ordinary pushdown systems and another fixed formula $\psi \implies \Box(\neg \mathit{fin})$. Unfortunately, this reduction is *not* polynomial because the atomic proposition Check occurring in ψ is interpreted with the help of several *product* automata constructed out of the original automata which implement checkpoints (see the previous section). Therefore we need one more technical proof.

Theorem 9. *The model-checking problem (I) for pushdown systems and LTL formulae with regular valuations is* ***EXPTIME****-complete even for a fixed formula* $(\Box \mathit{correct}) \implies (\Box \neg \mathit{fin})$.

Proof. This proof is similar to the proof of Theorem 7. Again, we construct a pushdown system $\mathcal{P}$ which simulates the execution of an alternating LBA $\mathcal{M} = (Q, \Sigma, \delta, q_0, \vdash, \dashv, p)$ on an input word $w \in \Sigma^*$ of length n. The difference is that, since there are no checkpoints, we must find a new way of 'cheating-detection', i.e., we must be able to recognize situations when the next configuration of $\mathcal{M}$ has not been guessed correctly. It is achieved by adding a family of control states $c_1, \ldots, c_n$; after guessing a new configuration, $\mathcal{P}$ successively switches its control state to $c_1, \ldots, c_n$ without modifying its stack. The underlying function f of the constructed regular valuation assigns to each pair $(\mathit{correct}, c_i)$ a deterministic automaton $\mathcal{M}^{c_i}_{correct}$ which checks that the triples of symbols at positions $i, i+1, i+2$ in each pair of successive configurations previously pushed to the stack are 'consistent' ($\mathcal{M}^{c_i}_{correct}$ is almost the same automaton as the $\mathcal{M}^{g}_{\gamma_i}$ of the proof of Theorem 7). All other pairs of the form $(\mathit{correct}, p)$ are assigned an automaton accepting Γ^*. The $\mathcal{P}$ is formally defined as follows: $\mathcal{P} = (\{g, a, r, c_1, \ldots, c_n\}, \Gamma, \Delta, g, \omega)$ where

- $\Gamma = \Sigma \times (Q \cup \{-\}) \cup \{\beta_1, \cdots, \beta_{n+3}\} \cup \{\#^e_1, \#^e_2, \#^u_1, \#^u_2, A, R, \omega\}$
- Δ contains the following (families of) rules:
  1. $\langle g, \omega\rangle \hookrightarrow \langle g, \beta_1\omega\rangle$
  2. $\langle g, \beta_i\rangle \hookrightarrow \langle g, \beta_{i+1}\varrho\rangle$ for all $1 \le i \le n+2$ and $\varrho \in \Sigma \times (Q \cup \{-\})$
  3. $\langle g, \beta_{n+3}\rangle \hookrightarrow \langle c_1, \varrho\rangle$ for every $\varrho \in \{\#^e_1, \#^u_1, A, R\}$
  4. $\langle c_i, \varrho\rangle \hookrightarrow \langle c_{i+1}, \varrho\rangle$ for every $1 \le i \le n-1$ and $\varrho \in \{\#^e_1, \#^u_1, A, R\}$
  5. $\langle c_n, \varrho\rangle \hookrightarrow \langle g, \varrho\rangle$ for every $\varrho \in \{\#^e_1, \#^u_1, A, R\}$
  6. $\langle g, A\rangle \hookrightarrow \langle a, \varepsilon\rangle$, $\langle g, R\rangle \hookrightarrow \langle r, \varepsilon\rangle$, $\langle g, \#^e_1\rangle \hookrightarrow \langle g, \beta_1\#^e_1\rangle$, $\langle g, \#^u_1\rangle \hookrightarrow \langle g, \beta_1\#^u_1\rangle$
  7. $\langle a, \varrho\rangle \hookrightarrow \langle a, \varepsilon\rangle$ for every $\varrho \in \Sigma \times (Q \cup \{-\})$
  8. $\langle a, \#^u_1\rangle \hookrightarrow \langle g, \beta_1\#^u_2\rangle$, $\langle a, \#^u_2\rangle \hookrightarrow \langle a, \varepsilon\rangle$, $\langle a, \#^e_1\rangle \hookrightarrow \langle a, \varepsilon\rangle$, $\langle a, \#^e_2\rangle \hookrightarrow \langle a, \varepsilon\rangle$
  9. $\langle r, \varrho\rangle \hookrightarrow \langle r, \varepsilon\rangle$ for every $\varrho \in \Sigma \times (Q \cup \{-\})$
  10. $\langle r, \#^e_1\rangle \hookrightarrow \langle g, \beta_1\#^e_2\rangle$, $\langle r, \#^e_2\rangle \hookrightarrow \langle r, \varepsilon\rangle$, $\langle r, \#^u_1\rangle \hookrightarrow \langle r, \varepsilon\rangle$, $\langle r, \#^u_2\rangle \hookrightarrow \langle r, \varepsilon\rangle$
  11. $\langle x, \varrho\rangle \hookrightarrow \langle x, \varrho\rangle$ for every control state x and every $\varrho \in \Gamma$.

Hence, the rules are almost the same as in the proof of Theorem 7, except for some changes in 3.,4.,5., and 11. The underlying function f of the constructed regular valuation assigns to (fin, a) an automaton recognizing all strings of Γ^* where the last symbol is ω, and to all other pairs of the form (fin, p) an automaton recognizing the empty language. We see that $\mathcal{M}$ accepts w iff there is an infinite path from the state $\langle g, \omega\rangle$ such that *correct* holds in all states of the path and *fin* holds in at least one state iff $\langle g, \omega\rangle \not\models^{\nu} (\Box \mathit{correct}) \implies (\Box \neg \mathit{fin})$ where ν is the constructed regular valuation. $\Box$

Observe that model-checking with pushdown systems and any fixed LTL formula whose predicates are interpreted by a *simple* valuation is already *polynomial* (see Theorem 1).

6 Conclusion

We have presented two different techniques for checking LTL with regular valuations on pushdown systems. Both techniques rely on a reduction to (and slight modification of) the problem for simple valuations discussed in [6]. Both techniques take linear time and space in $|States|$ where $States$ is the set of states of an automaton representing the regular predicates used in the formula. Since both take the same asymptotic time it would be interesting to compare their efficiency in practice (for cases where both techniques can be used).

The solution can be seamlessly combined with the concept of symbolic pushdown systems in [8]. These are used to achieve a succinct representation of Boolean Programs, i.e., programs with (recursive) procedures in which all variables are boolean.

The ability to represent data is a distinct advantage over the approaches hitherto made in our areas of application, namely data-flow analysis [7] and security properties [10]. For the latter, we have indicated that our model is more general. Our approach provides a unifying framework for these applications without losing efficiency. Both techniques take linear time in $|States|$ whereas the methods used in [7] were cubic (though erroneously reported as linear there, too). In [10] no complexity analysis was conducted.

References

1. T. Ball and S.K. Rajamani. Bebop: A symbolic model checker for boolean programs. In *SPIN 00: SPIN Workshop*, volume 1885 of *LNCS*, pages 113–130. Springer, 2000.
2. A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model checking. In *Proc. CONCUR'97*, LNCS 1243, pages 135–150.
3. O. Burkart and B. Steffen. Model checking the full modal mu-calculus for infinite sequential processes. In *Proc. ICALP'97*, volume 1256 of *LNCS*, pages 419–429. Springer, 1997.
4. E.A. Emerson. Temporal and modal logic. *Handbook of Theoretical Comp. Sci.*, B, 1991.
5. E.A. Emerson and C. Lei. Modalities for model checking: Branching time logic strikes back. *Science of Computer Programming*, 8(3):275–306, 1987.
6. J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon. Efficient algorithms for model checking pushdown systems. In *Proc. CAV'00*, LNCS 1855, pages 232–247. Springer, 2000.
7. J. Esparza and J. Knoop. An automata-theoretic approach to interprocedural data-flow analysis. In *Proceedings of FoSSaCS'99*, volume 1578 of *LNCS*, pages 14–30. Springer, 1999.
8. J. Esparza and S. Schwoon. A BDD-based model checker for recursive programs. In *Proc. CAV'01*, LNCS 2102, pages 324–336. Springer, 2001.
9. A. Finkel, B. Willems, and P. Wolper. A direct symbolic approach to model checking pushdown systems. *Electronic Notes in Theoretical Computer Science*, 9, 1997.
10. T. Jensen, D. Le Métayer, and T. Thorn. Verification of control flow based security properties. In *IEEE Symposium on Security and Privacy*, pages 89–103, 1999.
11. B. Steffen, A. Claßen, M. Klein, J. Knoop, and T. Margaria. The fixpoint-analysis machine. In *Proceedings of CONCUR'95*, volume 962 of *LNCS*, pages 72–87. Springer, 1995.