Model Checking LTL with Regular Valuations for Pushdown Systems[1]

Javier Esparza
Division of Informatics
University of Edinburgh
Edinburgh EH9 3JZ
United Kingdom
E-mail: jav@dcs.ed.ac.uk

and

Antonín Kučera[2]
Faculty of Informatics
Masaryk University
Botanická 68a
60200 Brno
Czech Republic
E-mail: tony@fi.muni.cz

and

Stefan Schwoon[3]
Institute for Informatics
Technische Universität München
Arcisstr. 21
80290 Munich
Germany
E-mail: schwoon@in.tum.de

Version: November 25, 2002

Recent works have proposed pushdown systems as a tool for analyzing programs with (recursive) procedures, and the model-checking problem for LTL has received special attention. However, all these works impose a strong restriction on the possible valuations of atomic propositions: whether a configuration of the pushdown system satisfies an atomic proposition or not can only depend on the current control state of the pushdown automaton and on its topmost stack symbol. In this paper we consider LTL with regular valuations: the set of configurations satisfying an atomic proposition can be an arbitrary regular language. The model-checking problem is solved via two different techniques, with an eye on efficiency. The resulting algorithms are polynomial in certain measures of the

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

[2]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.

[3]On leave at LFCS, Division of Informatics, University of Edinburgh.

problem which are usually small, but can be exponential in the size of the problem instance. However, we show that this exponential blowup is inevitable. The extension to regular valuations allows to model problems in different areas; for instance, we show an application to the analysis of systems with checkpoints. We claim that our model-checking algorithms provide a general, unifying and efficient framework for solving them.

Key Words: Model checking; linear-time logic; pushdown automata

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, 10] and certain data-flow analysis problems [8]. The model-checking problem has been considered for various logics, and quite efficient algorithms have emerged for linear time logics [3, 7, 11].

In this paper we revisit the model-checking problem for LTL and pushdown systems. 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 [3, 7, 11] valuations were completely determined by the control location and/or the topmost stack symbol (we call these valuations 'simple' in the following). Here we study (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.)

We proceed as follows. Section 2 contains basic definitions. Most of the technical content is in Section 3, where we formally define simple and regular valuations and propose our solutions to the model-checking problem. The solutions are based on a reduction to the case of simple valuations, which allows us to re-use most of the theory from [7]. While the reduction itself is based on a standard method, we pay special attention to ensuring its efficiency. This requires to modify the algorithm of [7] 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, because their efficiency in practice may depend on the concrete application.

In Section 4 we present two applications of the extension to regular valuations. A first application area (Section 4.1) is systems with checkpoints. In these systems computation is suspended at certain points to allow for a property of the stack content to be checked; resumption of the computation depends on the result of this inspection. This part of our work is motivated by the advent of programming languages that 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 [2, 12] have proposed a formal framework for such systems. Using their techniques one can prove the validity of control-flow based global security properties as well as to detect (and remove) redundant checkpoints. The formal framework of [2] can be reformulated in terms of *pushdown systems with checkpoints*, to which our model-checking algorithms can be applied. Our results improve on those of [2] in several ways: the approach of [2] only allows to check reachability properties, while our approach allows to check arbitrary LTL properties; we provide a detailed complexity analysis, with both upper and lower bounds; finally, our approach can be combined with symbolic techniques like BDDs for dealing with data-flow [10]. In Section 4.2 we present another application of our results, this time of a more theoretical nature: the extension to regular valuations leads to 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 [6]. We show that for pushdown systems model checking CTL* can be reduced to model checking LTL *with regular valuations*.

Our model-checking algorithms are polynomial in the size of certain parameters of the problem which are usually small. However, in the worst case those parameters can be *exponential* in the size of a problem instance, and so it is natural to ask if polynomial algorithms in the size of the problem instance exist. In Section 5 we provide a negative answer by establishing **EXPTIME** lower bounds. Hence, all of our algorithms are asymptotically optimal. We draw our conclusions in Section 6.

2. PRELIMINARIES

2.1. 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):

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

where $\mathcal{X}$ and $\mathcal{U}$ are the *next* and *until* operators, respectively. Let $At(\varphi)$ be the set of atomic propositions which appear in φ (note that $At(\varphi)$ is finite). Formulae are interpreted on infinite words over the alphabet $2^{At(\varphi)}$ (formally, an infinite word w is a function $w\colon \mathbb{N}_0 \to 2^{At(\varphi)}$). We denote that w satisfies a formula φ by $w \models \varphi$. The satisfaction relation is defined inductively on the structure of φ as follows, where w_i denotes the suffix of w starting with $w(i)$:

$$\begin{array}{lcl}
w \models \mathtt{tt} & & \\
w \models A & \iff & A \in w(0) \\
w \models \neg\varphi & \iff & w \not\models \varphi \\
w \models \varphi_1 \wedge \varphi_2 & \iff & w \models \varphi_1 \text{ and } w \models \varphi_2 \\
w \models \mathcal{X}\varphi & \iff & w_1 \models \varphi \\
w \models \varphi_1 \,\mathcal{U}\, \varphi_2 & \iff & \exists i\colon (w_i \models \varphi_2) \wedge (\forall j < i\colon w_j \models \varphi_1)
\end{array}$$

We also define $\Diamond\varphi \equiv \mathtt{tt}\,\mathcal{U}\,\varphi$ and $\Box\varphi \equiv \neg(\Diamond\neg\varphi)$.

Hence, every LTL formula φ defines a language $L(\varphi)$ consisting of all infinite words w over the alphabet $2^{At(\varphi)}$ such that $w \models \varphi$. It is well known (see, e.g., [13]) that given an LTL formula φ, one can effectively construct a Büchi automaton $\mathcal{B}_\varphi$

of size $\mathcal{O}(2^{|\varphi|})$ which recognizes the language $L(\varphi)$. This Büchi automaton is a tuple $\mathcal{B} = (Q, 2^{At(\varphi)}, \delta, q_0, F)$, where Q is the set of states, $2^{At(\varphi)}$ is the alphabet, q_0 is the initial state, F is the set of accepting states, and $\delta\colon Q \times 2^{At(\varphi)} \to 2^Q$ is the transition function. An *infinite* word w over the alphabet $2^{At(\varphi)}$ is accepted by $\mathcal{B}$ iff there is an (infinite) run of $\mathcal{B}$ over w that visits some accepting state infinitely often.

2.2. Transition Systems

A *transition system* is a triple $\mathcal{T} = (S, \to, r)$ where S is a set of *states* (not necessarily finite), $\to\, \subseteq S \times S$ 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 $i \geq 0$. To interpret the logic LTL over runs and states of $\mathcal{T}$, we first need to fix the meaning of atomic predicates by a *valuation*, which is a function $\nu\colon At \to 2^S$. Given a valuation ν and a formula φ, each run $w = s_0 s_1 s_2 \ldots$ determines a unique infinite word $w[\nu,\varphi]$ over the alphabet $2^{At(\varphi)}$ given by $w[\nu,\varphi](i) = \{A \in At(\varphi) \mid s_i \in \nu(A)\}$. The run w satisfies φ w.r.t. ν, denoted by $w \models^\nu \varphi$, iff $w[\nu,\varphi] \models \varphi$. Similarly, φ is true at a state $s \in S$ w.r.t. ν, written $s \models^\nu \varphi$, iff for each run w starting in s we have that $w \models^\nu \varphi$.

Observe that whether φ holds for runs and states of $\mathcal{T}$ is influenced only by the restriction of ν to $At(\varphi)$. In the next sections we denote this restriction by ν_φ (i.e., $\nu_\varphi\colon At(\varphi) \to 2^S$ is a function from a *finite* domain which agrees with ν on every argument).

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 use 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 or the complexity bounds derived in Section 3.2 (but makes the proofs for the latter slightly easier).

3. LTL ON PUSHDOWN SYSTEMS

Let $\mathcal{P} = (P, \Gamma, \Delta, q_0, \omega)$ be a pushdown system, φ an LTL formula, and $\nu\colon At \to 2^{P\times\Gamma^*}$ 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 sets of configurations defined above.

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*, $\delta\colon Q \times \Gamma \to 2^Q$ 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 $\delta(p, w) \cap F \neq \emptyset$. A set of configurations is *regular* if it is recognized by some $\mathcal{P}$-automaton.

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 $\nu\colon At \to 2^{P\times\Gamma^*}$, one can easily express undecidable properties of pushdown configurations just by atomic propositions). This motivates the search for 'reasonable' restrictions that do not limit the expressive power too much but allow to construct efficient model-checking algorithms at the same time. In the valuations considered in [3, 7], whether a configuration satisfies an atomic proposition or not depends only on its control location and the topmost stack symbol (it can be safely assumed that the stack is always nonempty). We define these valuations formally:

Definition 2. Let $\mathcal{P} = (P, \Gamma, \Delta, q_0, \omega)$ be a pushdown system. A set of configurations of $\mathcal{P}$ is *simple* if it has the form $\{\, \langle p, \alpha w\rangle \mid w \in \Gamma^* \,\}$ for some $p \in P$, $\alpha \in \Gamma$. A valuation $\nu\colon At \to 2^{P\times\Gamma^*}$ is *simple* if $\nu(A)$ is a union of simple sets for every $A \in At$.

In this 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 algorithms by presenting relevant complexity results.

Definition 3. Let $\mathcal{P} = (P, \Gamma, \Delta, q_0, \omega)$ be a pushdown system. A valuation $\nu\colon At \to 2^{P \times \Gamma^*}$ is *regular* if $\nu(A)$ is a regular set for every $A \in At$ and does not contain any configuration with an empty stack.

Notice that, since we assume a bottom stack symbol that is never removed, the requirement that configurations with empty stack cannot satisfy any atomic proposition is not a real restriction.

Since regular sets can be infinite, we need to represent regular valuations by finite means. We fix an adequate representation for our purposes.

Notation 1. Let ν be a regular valuation. For every atomic proposition A and control location p, we denote by $\mathcal{M}_A^p$ a deterministic finite-state automaton over the alphabet Γ having a total transition function and satisfying

$$\nu(A) = \{ \langle p, w \rangle \mid p \in P,\ w^R \in L(\mathcal{M}_A^p) \}$$

where w^R denotes the reverse of w.

Hence, A is true at $\langle p, w \rangle$ iff the automaton $\mathcal{M}_A^p$ enters a final state after reading the stack contents bottom-up. Since $\nu(A)$ does not contain any configuration with empty stack, the initial state of $\mathcal{M}_A^p$ is not accepting. This will simplify our constructions.

3.1. Model Checking with Simple Valuations

The model-checking problems (I) to (III) of the previous section have been solved in [7] for simple valuations. In this section we briefly recall the solutions. The model-checking problem for the initial configuration, problem (I), is solved in three steps as follows:

(1) The problem is reduced to the emptiness problem for Büchi pushdown systems. A *Büchi pushdown system* is a pushdown system with a subset of *accepting* control locations. A run of a Büchi pushdown system is *accepting* if it visits some accepting control location infinitely often. A Büchi pushdown system is *empty* if it has no accepting run.

(2) The emptiness problem for Büchi pushdown systems is reduced to computing the set of predecessors of certain regular sets of configurations. The set of predecessors of a set C of configurations, denoted by $pre^*(C)$, contains the configurations $\langle p, w \rangle$ such that $\langle p, w \rangle \to^* \langle p', w' \rangle$ for some $\langle p', w' \rangle \in C$.

(3) An algorithm is presented that, given an arbitrary $\mathcal{P}$-automaton $\mathcal{A}$ recognizing a regular set C of configurations, computes another $\mathcal{P}$-automaton $\mathcal{A}_{pre^*}$ recognizing the set $pre^*(C)$. (It can be shown that if C is regular, then so is $pre^*(C)$.)

We now give some more details about these steps.

Step 1. Let $\mathcal{P} = (P, \Gamma, \Delta, p_0, \omega)$ be a pushdown system and let φ be a formula of LTL. We first construct a Büchi automaton $\mathcal{B} = (Q, 2^{At(\varphi)}, \delta, q_0, F)$ recognizing $L(\neg\varphi)$. We then construct a Büchi pushdown system $\mathcal{BP} = ((P \times Q), \Gamma, \Delta', (p_0, q_0), \omega, G)$ (where G is the set of accepting control locations) by "synchronizing" $\mathcal{P}$ and $\mathcal{B}$ as follows:

- $\langle (p,q), \alpha \rangle \hookrightarrow \langle (p',q'), v \rangle \in \Delta'$ if
  - $\langle p, \alpha \rangle \hookrightarrow \langle p', v \rangle$; and
  - $q' \in \delta(q, \sigma)$ where σ is the set of all atomic propositions of $At(\varphi)$ that are true at the configuration $\langle p, \alpha \rangle$.
- $(p,q) \in G$ if $q \in F$.

If valuations are simple, then, whenever the rule $\langle (p,q), \alpha \rangle \hookrightarrow \langle (p',q'), v \rangle \in \Delta'$ is applied to derive a transition $\langle (p,q), \alpha w \rangle \to \langle (p',q'), vw \rangle$, we always have that A is true at the configuration $\langle p, \alpha w \rangle$. Using this property, it is easy to show that $\mathcal{P}$ has a run violating φ if and only if $\mathcal{BP}$ is nonempty.

Step 2. Let $\mathcal{BP} = (P, \Gamma, \Delta, p_0, \omega, G)$ be an arbitrary Büchi pushdown system. The *head* of a transition rule $\langle p, \alpha \rangle \hookrightarrow \langle p', w \rangle$ of Δ is the configuration $\langle p, \alpha \rangle$. A head $\langle p, \alpha \rangle$ is *repeating* if there exists $v \in \Gamma^*$ such that $\langle p, \alpha v \rangle$ can be reached from $\langle p, \alpha \rangle$ by means of a sequence of transitions that visits some control location of G. Let *Rep* be the set of repeating heads, and let $Rep\Gamma^*$ denote the set $\{ \langle p, \alpha w \rangle \mid \langle p, \alpha \rangle \in Rep,\ w \in \Gamma^* \}$. It is shown in [7] that $\mathcal{BP}$ is nonempty if and only if $\langle p_0, \omega \rangle \in pre^*(Rep\Gamma^*)$. Therefore, emptiness can be decided by computing first *Rep* and then $pre^*(Rep\Gamma^*)$.

In order to compute *Rep* we construct a *head reachability graph* whose nodes are the heads of Δ. There is an edge from $\langle p, \alpha \rangle$ to $\langle p', \beta \rangle$ if there is a rule $\langle p, \alpha \rangle \hookrightarrow \langle p'', v_1 \beta v_2 \rangle$ in Δ such that $\langle p'', v_1 \rangle \to^* \langle p', \varepsilon \rangle$. If either $p \in G$ or $\langle p', \varepsilon \rangle$ can be reached from $\langle p'', v_1 \rangle$ visiting a final control location along the way, then we say that the edge is *marked*. It is shown in [7] that a head is repeating if and only if it belongs to a strongly connected component of the head reachability graph that contains at least one marked edge.

These results show that the emptiness problem can be reduced to computing $pre^*(\{ \langle p, \varepsilon \rangle \mid p \in P \}$, finding the strongly connected components in the graph, and computing $pre^*(Rep\Gamma^*)$.

Step 3. Without loss of generality, we assume that $\mathcal{A}$ has no transition leading to an initial state. We compute $pre^*(C)$ by means of a saturation procedure. The procedure adds new transitions to $\mathcal{A}$, but no new states. New transitions are added according to the following *saturation rule*:

> If $\langle p, \gamma \rangle \hookrightarrow \langle p', w \rangle$ and $p' \xrightarrow{w} q$ in the current $\mathcal{P}$-automaton, add a transition (p, γ, q).

The saturation procedure eventually reaches a fixpoint because the number of possible new transitions is finite. An efficient implementation of the procedure is presented in [7].

This finishes the discussion of problem (I). Problem (II) has the same solution, because the automaton recognizing $pre^*(Rep\Gamma^*)$ is a finite representation of all the configurations violating φ. Problem (III) is solved by showing that the set of reachable configurations of $\mathcal{P}$ is regular, and by proposing an algorithm – very similar to that for $pre^*(C)$ – that constructs a $\mathcal{P}$-automaton recognizing this set. The product of this automaton and that for $pre^*(Rep\Gamma^*)$ is a finite representation of all the reachable configurations of $\mathcal{P}$ that violate φ. The following theorems are taken from [7].

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 $\mathcal{A}$ with $\mathcal{O}(|P| \cdot |\mathcal{B}|)$ states and $\mathcal{O}(|P| \cdot |\Delta| \cdot |\mathcal{B}|^2)$ transitions 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) where $\langle p, w\rangle \not\models^\nu \varphi$;*
- *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.*

3.2. Model Checking with Regular Valuations

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 corresponding to $\neg\varphi$ (see Section 2.1) is denoted by $\mathcal{B} = (R, 2^{At(\varphi)}, \eta, r_0, G)$. Let $At(\varphi) = \{A_1, \ldots, A_n\}$, 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 $1 \le i \le n$. Since $\mathcal{M}^p_{A_i}$ is deterministic, we treat ϱ^p_i as a function that maps pairs from (Q^p_i, Γ) into Q^p_i (rather than $2^{Q^p_i}$).

In the next subsections we design two algorithms for model checking with regular valuations. Actually, both of them reduce the problem to model checking with simple valuations discussed in Section 3.1; in both cases, the idea is to encode the $\mathcal{M}^p_{A_i}$ automata into the structure of $\mathcal{P}$ and simulate them 'on the fly' during the computation of $\mathcal{P}$. In practice, some of the $\mathcal{M}^p_{A_i}$ automata can be identical (see, e.g., the examples in Section 4.1). Of course, it does not make much sense to simulate the execution of the *same* automaton within $\mathcal{P}$ twice, and our constructions should reflect this. For simplicity, we assume that whenever $i \ne j$ or $p \ne q$, then the $\mathcal{M}^p_{A_i}$ and $\mathcal{M}^q_{A_j}$ automata are either identical or have disjoint sets of states. So, let $\{\mathcal{M}_1, \ldots, \mathcal{M}_m\}$ be the *set* of all $\mathcal{M}^p_{A_i}$ automata where $1 \le i \le n$ and $p \in P$ (hence, if some of the $\mathcal{M}^p_{A_i}$ automata are identical, then $m < n \cdot |P|$), and let Q_j be the set of states of $\mathcal{M}_j$ for each $1 \le j \le m$. The Cartesian product $\prod_{1\le j\le m} Q_j$ is denoted by *States*. For given $\mathbf{r} \in$ *States*, $p \in P$, and $1 \le i \le n$, we denote by $\mathbf{r}^p_i$ the element of Q^p_i which appears in $\mathbf{r}$ (observe that we can have $\mathbf{r}^p_i = \mathbf{r}^q_j$ even if $i \ne j$ or $p \ne q$). The vector of initial states (i.e., the only element of *States* where each component is the initial state of some $\mathcal{M}^p_{A_i}$) is denoted by $\mathbf{s}$. Furthermore, we write $\mathbf{t} = \varrho(\mathbf{r}, \alpha)$ if $\mathbf{t}_i^p = \varrho_i^p(\mathbf{r}_i^p, \alpha)$ 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 $|\mathcal{P}| + |\varphi| + |\nu_\varphi|$, where $|\nu_\varphi|$ is the total size of all employed automata. However, in practice we usually work with small formulae and a number of simple automata (see Section 4); therefore, we measure the complexity of our algorithms in $|\mathcal{B}|$ and $|\mathit{States}|$ rather than in $|\varphi|$ and $|\nu_\varphi|$ (in general, $\mathcal{B}$ and States can be *exponentially* larger than φ and ν_φ). This allows for a detailed complexity analysis whose results better match the reality because $|\mathcal{B}|$ and $|\mathit{States}|$ 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.

3.2.1. Stack Extension

The idea behind this technique is to evaluate the atomic propositions $A_1, \ldots, A_n$ by storing the vectors of *States* in the stack of $\mathcal{P}$ and updating them after each transition. As we will see, we conveniently use the assumptions that the $\mathcal{M}_{A_i}^p$ automata are deterministic, have total transition functions, and read the stack bottom-up.

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

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

A configuration $\langle q, (\alpha_k, \mathbf{r}_k) \cdots (\alpha_1, \mathbf{r}_1)\rangle$ is called *consistent* iff $\mathbf{r}_1 = \mathbf{s}$ and $\mathbf{r}_{j+1} = \varrho(\mathbf{r}_j, \alpha_j)$ for all $1 \leq j < k$. In other words, $\langle q, (\alpha_k, \mathbf{r}_k) \cdots (\alpha_1, \mathbf{r}_1)\rangle$ is consistent iff $\mathbf{r}_1, \ldots, \mathbf{r}_k$ are the states encountered when simulating the $\mathcal{M}_{A_i}^p$ automata on the input $\alpha_1 \cdots \alpha_k$, i.e. the stack contents read in bottom-up direction. Let $\langle p, w\rangle$ be a configuration of $\mathcal{P}$ and $\langle p, w'\rangle$ be the (unique) associated consistent configuration of $\mathcal{P}'$. Now we can readily confirm that

(A) if $\langle p, w\rangle \rightarrow \langle q, v\rangle$, then $\langle p, w'\rangle \rightarrow' \langle q, v'\rangle$ where $\langle q, v'\rangle$ is the unique consistent configuration associated to $\langle q, v\rangle$;

(B) if $\langle p, w'\rangle \rightarrow' \langle q, v'\rangle$, then $\langle q, v'\rangle$ is consistent and $\langle p, w\rangle \rightarrow \langle q, v\rangle$ is a transition of $\mathcal{T}_\mathcal{P}$ (where $v \in \Gamma^*$ is obtained by stripping the state vector components from v').

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 $\langle p, w'\rangle$ be its associated consistent configuration of $\mathcal{P}'$. Then the parts of $\mathcal{T}_\mathcal{P}$ and $\mathcal{T}_{\mathcal{P}'}$ that are reachable from $\langle p, w\rangle$ and $\langle p, w'\rangle$, respectively, are isomorphic.

The simple valuation ν' is defined by

$$\nu'(A_i) = \{\, \langle p, (\alpha, \mathbf{r})w' \rangle \mid p \in P,\ \alpha \in \Gamma,\ \varrho_i^p(\mathbf{r}_i^p, \alpha) \in F_i^p,\ w' \in (\Gamma')^* \,\}$$

It is now easy to see that $\langle q, \alpha_k \cdots \alpha_1 \rangle \models^{\nu} \varphi \iff \langle q, (\alpha_k, \mathbf{r}_k) \cdots (\alpha_1, \mathbf{r}_1) \rangle \models^{\nu'} \varphi$ where $\langle q, (\alpha_k, \mathbf{r}_k) \cdots (\alpha_1, \mathbf{r}_1) \rangle$ is consistent.

Using the fact that each $\mathcal{M}_{A_i}^p$ is deterministic, we observe that $|\Delta'| = |\Delta| \cdot |\mathit{States}|$. Applying Theorem 2, we obtain that using the Stack Extension technique we can solve the model-checking problems (I) and (II) for $\mathcal{P}'$ and ν' in linear time and space (w.r.t. $|\mathit{States}|$) and the model-checking problem (III) in quadratic or cubic time and space. However, the automata computed as solutions to problems (II) and (III) can also accept inconsistent configurations. To solve the model-checking problems for $\mathcal{P}$ and the regular valuation ν, we have to get rid of these. Closer analysis reveals that we can do so without increasing the asymptotic complexity of problems (I) and (II) *and* that there is a better solution for problem (III).

THEOREM 3. *The Stack Extension technique gives us the following bounds on the model-checking problems with regular valuations:*

1. *Problems (I) and (II) can be solved in* $\mathcal{O}(|P|^2 \cdot |\Delta| \cdot |\mathit{States}| \cdot |\mathcal{B}|^3)$ *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.*

In other words, all problems take only *linear* time and space in $|\mathit{States}|$.

Proof. Since $|\Delta'| = |\Delta| \cdot |\mathit{States}|$, Theorem 1 allows to compute a $\mathcal{P}'$-automaton $\mathcal{D} = (D, \Gamma', \gamma, P, G)$ of size $\mathcal{O}(|P| \cdot |\Delta| \cdot |\mathit{States}| \cdot |\mathcal{B}|^2)$ 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 such that $\mathcal{D}$ recognizes all configurations of $\mathcal{P}'$ which violate φ; then, to solve problem (I), we just look if $\mathcal{D}$ accepts $\langle q_0, \omega' \rangle$.

$\mathcal{D}$ 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 $Q = (D \times \mathit{States}) \cup P$, $F = G \times \{\mathbf{s}\}$, and δ is as follows:

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

Notice that $\mathcal{A}$ is the same size as $\mathcal{D}$ since in every transition $\mathbf{t}$ is uniquely determined by $\mathbf{r}$ and α. Now, for every configuration $\langle p, (\alpha_k, \mathbf{r}_k) \cdots (\alpha_1, \mathbf{r}_1) \rangle$, one can easily prove (by induction on k) that

$$\gamma(p, (\alpha_k, \mathbf{r}_k) \cdots (\alpha_1, \mathbf{r}_1)) \ni q \text{ where } \mathbf{r}_{j+1} = \varrho(\mathbf{r}_j, \alpha_j) \text{ for all } 1 \le j < k$$

iff

$$\delta(p, \alpha_k \cdots \alpha_1) = \delta((p, \mathbf{r}), \alpha_k \cdots \alpha_1) \ni (q, \mathbf{r}_1) \text{ where } \mathbf{r} = \varrho(\mathbf{r}_k, \alpha_k).$$

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 prove the second part, we simply need to synchronize $\mathcal{A}$ with an automaton $\mathcal{R}$ recognizing all reachable configurations of $\mathcal{P}$. 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, requires too much computation time. We can do better by employing the following trick from [7]: first all transitions (q, α, q') of $\mathcal{R}$ are sorted into buckets labeled by α. Then each transition of $\mathcal{A}$ is synchronized 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 $\mathcal{O}(|P| \cdot |\Delta| \cdot (|P| + |\Delta|)^2 \cdot |\mathit{States}| \cdot |\mathcal{B}|^2)$ time and space. Alternatively, we can sort the transitions of $\mathcal{A}$ into buckets of size $\mathcal{O}(|P|^2 \cdot |\mathcal{B}|^2 \cdot |\mathit{States}|)$ (exploiting the fact that one of the *States* components in each transition is determined by the other) and construct the product in $\mathcal{O}(|P|^3 \cdot |\Delta| \cdot (|P|+|\Delta|) \cdot |\mathit{States}| \cdot |\mathcal{B}|^2)$ time and space. If we add the time and space needed to construct $\mathcal{A}$ and $\mathcal{R}$, we get the results stated in the second part of Theorem 3. ∎

3.2.2. Control Extension

An alternative approach to model checking with regular valuations is to store the product of the $\mathcal{M}^p_{A_i}$ automata in the finite control of $\mathcal{P}$. As we shall see, we need one *additional* assumption to make this construction work:

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

This assumption is truly restrictive – there are quite simple regular languages that cannot be recognized by finite-state automata that are both deterministic and backward deterministic, for example the language $\{\, a^i \mid i > 0 \,\}$. Therefore, the Control Extension technique is less general than the Stack Extension technique. Nonetheless, we present it for the sake of completeness but omit the proofs, which can be found in the technical report version of this paper [9].

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

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

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

A configuration $\langle (q, \mathbf{r}), w\rangle$ of $\mathcal{P}'$ is *consistent* iff $\mathbf{r} = \varrho(\mathbf{s}, w^R)$, i.e. if $\mathbf{r}$ 'reflects' the stack contents w. The simple valuation ν' is defined by

$$\nu'(A_i) = \{\, \langle (p, \mathbf{r}), w\rangle \mid (p, \mathbf{r}) \in P',\ \mathbf{r}^p_i \in F^p_i,\ w \in \Gamma^+ \,\}$$

for all $1 \le i \le n$. It is easy to see that for all $p \in P$ and $w \in \Gamma^*$ we have

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

Earlier, we noted that $|P'| = |P| \cdot |States|$ and $|\Delta'| = |\Delta| \cdot |States|$. Applying Theorem 2 naïvely, we obtain that using the Control Extension technique, 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 4. *The Control Extension technique gives us the same bounds on the model-checking problems with regular valuations as the Stack Extension technique, 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.*

Incidentally, neither of the two techniques could be easily adapted to the case where the $\mathcal{M}^p_{A_i}$ automata are nondeterministic. For instance, checking the formula $\Diamond\varphi$ could fail if a valuation automaton can choose between an accepting and a non-accepting state, thus permitting a run in which no configuration seems to satisfy φ.

4. APPLICATIONS

4.1. Pushdown Systems with Checkpoints

Pushdown systems find a natural application in the analysis of sequential programs with procedures or methods.[4] However, modern programming languages contain methods for performing run-time inspections of the stack of activation records that cannot be modelled by pushdown systems. An example is the method `checkPermission(perm)` of the class `java.security.AccessController` in the Java API. An invocation of `checkPermission(perm)` terminates silently or raises an exception depending not only on which is the current method, but also on the current stack of activation records.

In this section we introduce pushdown systems with checkpoints and show that they can be used to model these and more general security features. We show how natural analysis problems about the security behaviour can be formulated as model-checking problems for LTL and pushdown systems with checkpoints. We then apply our results on regular valuations to provide an efficient model-checking algorithm. Our work is inspired by [12], which also deals with the problem of modelling security features, and by [2], which improves upon [12]. The approach of [2] is also based on finite automata to model the set of stack contents which allow computation to continue. A finite abstraction of the program is constructed, and it is shown that the program can reach a 'dangerous' configuration if and only if the abstraction can also reach it. While [12, 2] deserve full credit for showing that the security problems can be formalised using an extension of pushdown systems, we think that our checking procedures have several advantages. First, we can check arbitrary LTL properties. Second, we provide a more detailed complexity analysis: in [2] no estimation of the time required to build the abstraction is provided (only an upper bound on its size), whereas Theorems 3 and 4 give accurate information; moreover, the results

[4] Since Java will be our example of application, we will use 'method' in the sequel.

of section 5 show that the reachability problem of [2] requires exponential time. Finally, our algorithms have been extended to incorporate symbolic representation of data using BDDs [10].

Let us start by briefly describing how to model a class without security features by a pushdown automaton. We consider a class with possibly recursive methods, even with mutual invocations between methods. However, we require that the data types used in the program be finite, and that the heap of references to objects cannot grow arbitrarily large. In fact, in order to simplify the presentation, we assume that the methods cannot create any object at all, and that all methods have the same local variables.

We represent the class by a system of *flow graphs*, one for each method. The nodes of a flow graph correspond to control points in the method, and its edges are annotated with statements, e.g. assignments or invocations of other methods. Non-deterministic control flow is allowed and might for instance result from abstraction. Let the sets G and L contain the possible valuations of the global and local variables, respectively. E.g., if the class contains three boolean global variables and each method has two boolean local variables, then we have $G = \{0,1\}^3$ and $L = \{0,1\}^2$. We can now describe the pushdown system $(P, \Gamma, \Delta, q_0, w)$ corresponding to the class. We take $P = G$, and let q_0 contain the initial values of the global variables. The stack alphabet Γ contains all pairs (n, loc), where n is a control point of a flow graph, and $\mathit{loc} \in L$. Thus, a configuration has the form $\langle \mathit{glob}, (n_1, \mathit{loc}_1) \ldots (n_k, \mathit{loc}_k)\rangle$, where n_1 is the current control point, and the sequence $(n_2, \mathit{loc}_2) \ldots (n_k, \mathit{loc}_k)$ models the stack of activation records. Each flow-graph edge is modelled by a set of rules of the pushdown system. An edge e from node n_1 to node n_2 labelled by an assignment is modelled by a set of rules of the form

$$\langle \mathit{glob}, (n_1, \mathit{loc})\rangle \hookrightarrow \langle \mathit{glob}', (n_2, \mathit{loc}')\rangle$$

where glob and glob' (loc and loc') are the values of the global (local) variables before and after the assignment. An edge from node n_1 to node n_2 labelled by a method invocation is translated into a set of rules with a right-hand side of length two according to the following scheme:

$$\langle \mathit{glob}, (n_1, \mathit{loc})\rangle \hookrightarrow \langle \mathit{glob}', (m_0, \mathit{loc}')\,(n_2, \mathit{loc}'')\rangle$$

Here m_0 is the start node of the invoked method, and loc' denotes initial values of its local variables; loc'' saves the local variables of the caller method. A return statement is modelled by rules with an empty right-hand side:

$$\langle \mathit{glob}, (n_1, \mathit{loc})\rangle \hookrightarrow \langle \mathit{glob}', \varepsilon\rangle$$

Methods that return values can be simulated by introducing an additional global variable and assigning the return value to it.

Let us now consider edges labelled by `checkPermission(perm)` for some permission *perm*. According to the Java API, the operational semantics of an edge from n_1 to n_2 labelled with `checkPermission(perm)` is as follows. Suppose that control is at n_1, and execution has traversed the caller methods $m_1, m_2, \ldots m_k$ (e.g., method m_1 called m_2, which called m_3, etc.). Then `checkPermission(perm)` checks iteratively if $m_k, m_{k-1}, \ldots, m_1$ (in this order) have permission `perm`, and grants access to n_2 if the answer is 'yes' in all cases, or if there is a $j \leq k$ such that the answer is 'yes' for $m_k, m_{k-1}, \ldots, m_{j+1}$, and m_j is a so-called 'privileged' method. Otherwise, `checkPermission` throws an exception.

Since this semantics requires to inspect the whole stack, and pushdown rules can only inspect the top stack symbol, the behaviour of `checkPermission` cannot be modelled using pushdown systems. For this reason, we introduce pushdown systems with checkpoints. Loosely speaking, a checkpoint is a pair (p, α), where p is a control state and α is a stack symbol. Each checkpoint (p, α) is associated with a regular language of stack contents represented by a finite automaton $\mathcal{M}^p_\alpha$. The semantics of pushdown systems with checkpoints restricts the contexts in which a rule $\langle p, \alpha\rangle \to \langle q, v\rangle$ can be applied: instead of being applicable in all contexts w, i.e., in all configurations $\langle p, \alpha w\rangle$, applicability now depends on whether $w^R\alpha$ is accepted by $\mathcal{M}^p_\alpha$ or not. For technical convenience, we introduce three different classes of rules: positive rules, which are applicable when $w^R\alpha \in \mathcal{M}$, negative rules, applicable when $w^R\alpha \notin \mathcal{M}$, and independent rules, which behave like the rules of pushdown automata, i.e., they are always applicable.

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.
- $\xi \subseteq P \times \Gamma$ is a set of *checkpoints*. Each checkpoint (p, α) is *implemented* by its associated 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 $L(\mathcal{M}^p_\alpha) \subseteq \{\, w\alpha \mid w \in \Gamma^* \,\}$.
- $\eta\colon \Delta \to \{+, -, 0\}$ is a function which partitions the set of transition rules into *positive*, *negative*, and *independent* ones. 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. 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.

A configuration of $\mathcal{P}$ is an element of $P \times \Gamma^*$. To $\mathcal{P}$ we associate a unique transition system $\mathcal{T}_\mathcal{P}$ where the set of states is the set of all configurations, $\langle q_0, \omega\rangle$ 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 $w \in \Gamma^*$.

We can now model the operational semantics of `checkPermission` using pushdown systems with checkpoints. Assume for simplicity (see [2] for a detailed discussion) that whether a node has permission `perm` and whether it is privileged or not can be determined statically. The set P and Γ of control locations and stack symbols are as before. The checkpoints are the pairs $(glob, (n, loc)$ such that n has some outgoing edge labelled by `checkPermission(perm)`. The language associated to a checkpoint $(glob, (n, loc))$ is given by the sequences $(n_k, loc_k) \ldots (n_1, loc_1)$ such that either $n_k, \ldots, n_1$ have permission `perm` or $n_k \ldots n_{j+1}$ have `perm` and n_j is privileged for some $1 \le j \le k$. It is easy to see that this language is regular. Rules corresponding to edges labelled by `checkPermission` are positive, and all other rules are independent.

Notice that negative rules are not used in this example. However, it is not hard to imagine systems that perform `if-then-else` commands where the condition is based on dynamic checks; these could be modelled using positive and negative rules.

Some natural problems for pushdown processes with checkpoints are listed below. Notice that all of them are relevant when modelling security features.

- 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.
- 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 overhead.

It is 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 $\mathcal{M}_A^p$ has $|w| + 1$ 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.

4.1.1. Model Checking LTL for Pushdown Systems with Checkpoints

Let $\mathcal{C} = (\mathcal{P}, \xi, \eta)$ be a pushdown system with checkpoints, where $\mathcal{P}$ has the form $(P, \Gamma, \Delta, q_0, \omega)$. 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 $\mathcal{C}$ (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 regular valuation ν' for configurations of $\mathcal{P}'$ as follows:

- If $A \in At \setminus \{Check, Neg, Pos\}$, then $\nu'(A) = \{\langle (p, x), w\rangle \mid \langle p, w\rangle \in \nu(A), x \in \{+, -, 0\}\}$. Hence, the automaton $\mathcal{M}_A^{(p,x)}$ is the same as the automaton $\mathcal{M}_A^p$ for each $x \in \{+, -, 0\}$.

- $\nu'(\mathit{Check}) = \bigcup_{(p,\alpha)\in\xi}\{\langle (p,x), w\rangle \mid w^R \in L(\mathcal{M}^p_\alpha), x \in \{+,-,0\}\}$. Hence, for each $x \in \{+,-,0\}$, $\mathcal{M}^{(p,x)}_{\mathit{Check}}$ is the product automaton (constructed out of all $\mathcal{M}^p_\alpha$) which accepts the union of all $L(\mathcal{M}^p_\alpha)$. Notice that we need to perform a product because $\mathcal{M}^{(p,x)}_{\mathit{Check}}$ has to be deterministic.

- $\nu'(\mathit{Neg}) = \{\langle (p,-), w\rangle \mid p \in P, w \in \Gamma^+\}$. So, $\mathcal{M}^{(p,-)}_{\mathit{Neg}}$ is an automaton with two states that accepts Γ^+.

- $\nu'(\mathit{Pos}) = \{\langle (p,+), w\rangle \mid p \in P, w \in \Gamma^+\}$.

In other words, *Check* holds for a configuration $\langle (p,x), \alpha w\rangle$ if and only if (p,α) is a checkpoint and the associated stack inspection yields a positive result. 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 observe 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; notice 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, i.e. we regard only paths in which we never use a negative rule if an inspection at a checkpoint is positive, and we never use a positive rule if an inspection is negative. 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 ν' above), we can conclude that both paths either satisfy or do not satisfy φ. ∎

The previous theorem in fact says that the model-checking problem for LTL and pushdown systems with checkpoints 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 3.2. 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* $\mathcal{O}(|P| \cdot |\Delta| \cdot (|P| + |\Delta|)^2 \cdot |States| \cdot |\mathcal{B}|^3)$ *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. 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 as $\mathcal{B}$. 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 the automata which represent atomic predicates of $At(\psi \implies \varphi)$ (see above) and consider the state space of their product, we see that it has *exactly* the size $2 \cdot States$ because all of the automata associated to *Pos* and *Neg* are the same and have only two states. ∎

4.2. Model checking CTL*

In this section, we apply the model-checking algorithm to the logic CTL* which extends LTL with existential path quantification [5]. More precisely, CTL* formulae are built according to the following abstract syntax equation:

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

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

For finite-state systems, model checking CTL* can be reduced to checking LTL as follows [6]: 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 [11], 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 3.2 (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 nondeterministic, reads the stack top-down, and has $\mathcal{O}(|P| \cdot |R| \cdot |States|)$ 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 determinize it. Reversal does not increase the size, and due to the determinism of $\mathcal{M}^p_{A_i}$ (in bottom-up direction) the determinization 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 $\{\, (q, s) \mid q \in 2^{P\times R}, q \ni (p, r_0), s \in \mathit{States} \,\}$. The cross product of these new automata with the 'old' $\mathcal{M}^p_{A_i}$ automata takes only $O(|\mathit{States}| \cdot 2^{|P|\cdot|R|})$ 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 $s \in \mathit{States}$.

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 determinize the resulting automaton. By the previous arguments, each determinization only blows up the nondeterministic 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 subformulae and using Theorem 3 we get that the model-checking procedure takes at most

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

time and

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

space. The algorithm hence remains linear in both $|\Delta|$ and $|\mathit{States}|$. The algorithm of Burkart and Steffen [4], 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 determinization 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 $|\mathcal{P}|+|\mathcal{B}|+|\mathit{States}|$, and not in the size of *problem instance* which is (as we already mentioned in Remark 1) $|\mathcal{P}|+|\varphi|+|\nu_\varphi|$. 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 $|\mathit{States}|$ – in Section 4 we have seen that there are interesting practical problems where the size of $|\mathit{States}|$ does not explode. Nevertheless, from the point of view of worst-case analysis (where we measure the complexity in the size of a 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 other 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 are (polynomially) reducible 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 linearly bounded automata (which is known to be **EXPTIME**-complete). An *alternating linearly bounded automaton (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 nondeterministic LBA (in particular, $\vdash \in \Sigma$ and $\dashv \in \Sigma$ are the left-end and the right-end markers, resp.), and $p: Q \to \{\forall, \exists, acc, rej\}$ 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 universal and existential configuration has exactly 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 $\mathcal{M}$ labeled 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 labeled by a configuration reachable from uqv in one computational step (according to δ);
- if q is universal, then N has two successors labeled by the two configurations reachable from uqv in one computational step.

$\mathcal{M}$ *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 a 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 the checkpoints are used to verify that there was no cheating (i.e. pushing of inconsistent sequences of symbols which do *not* model a computation of $\mathcal{M}$) during the simulation. We start with the definition of $\mathcal{P}$. Configurations of $\mathcal{M}$ will be represented by words over the alphabet $\Sigma \times (Q \cup \{-\})$ of length $n+2$ (notice that w is surrounded by the '$\vdash$' and '$\dashv$' markers). Each such word contains exactly one symbol $(X, t) \in \Sigma \times Q$

which encodes the current control state and the current position of the head. We put $\mathcal{P} = (\{g, a, r\}, \Gamma, \Delta, g, \omega)$ 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. During this phase, the family of β_i symbols is used as a 'counter' (rules 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). The only purpose of γ_i symbols 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'. 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 $length(v_j) = n+2$, $\varrho_j \in \#_1^e, \#_2^e, \#_1^u, \#_2^u, A, R$ 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 $\mathcal{M}$. 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 $\langle a, \omega\rangle$. ∎

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 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 such that $\nu(\mathit{fin}) = \{\langle a, \omega w\rangle \mid w \in \Gamma^*\}$. ∎

It follows that model checking LTL for pushdown systems with checkpoints is **EXPTIME**-complete even when we have only *simple* valuations.

Now we analyze the complexity of model checking for (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 $\psi \Longrightarrow \Box(\neg \mathit{fin})$. 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* $(\Box \mathit{correct}) \Longrightarrow (\Box\neg \mathit{fin})$.

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 constructed regular valuation ν 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. Now we put $\nu(\mathit{fin}) = \{\langle a, \omega w\rangle \mid w \in \Gamma^*\}$. We see that $\mathcal{M}$ accepts w iff there is a run starting in $\langle g, \omega\rangle$ along which *correct* holds and *fin* holds in at least one of its states. In other words, $\mathcal{M}$ accepts w iff $\langle g, \omega\rangle \not\models^{\nu} (\Box \mathit{correct}) \Longrightarrow (\Box \neg \mathit{fin})$ where ν is the constructed regular valuation. ∎

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 [7]. Both techniques take linear time and space in $|\mathit{States}|$ 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 [10]. 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 an advantage over the approach hitherto made in the first of our two applications, namely the analysis of security properties of Java programs [2, 12]. Another advantage is the ability to check liveness properties. We also provide a detailed worst-case complexity analysis, which was missing in [2, 12].

Our second application is a model-checking algorithm for CTL*. Our complexity estimation differs from that of Burkart and Steffen (applied to CTL* formulae) in an interesting way. In [4], the exponential blowup occurs because configurations are extended with a 'context' signifying the subformulae that will be true once the topmost symbol is removed from the stack. In our case, the exponential blowup occurs because we need to determinise finite automata. However, these automata are derived from Büchi automata, which are usually constructed by a tableau method that labels states with certain subformulae. Therefore, the complexity estimations of the two algorithms might be related in a subtle fashion which would be interesting to examine.

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] F. Besson, T. Jensen, D.L. Métayer, and T. Thorn. Model checking security properties of control flow graphs. *Journal of Computer Security*, 9:217–250, 2001.

[3] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model checking. In *Proceedings of CONCUR'97*, volume 1243 of *LNCS*, pages 135–150. Springer, 1997.

[4] 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.

[5] E.A. Emerson. Temporal and modal logic. *Handbook of Theoretical Comp. Sci.*, B, 1991.

[6] E.A. Emerson and C. Lei. Modalities for model checking: Branching time logic strikes back. *Science of Computer Programming*, 8(3):275–306, 1987.

[7] 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.

[8] 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.

[9] J. Esparza, A. Kučera, and S. Schwoon. Model-checking LTL with regular valuations for pushdown systems. Technical Report EDI-INF-RR0044, Division of Informatics, University of Edinburgh, 2001.

[10] J. Esparza and S. Schwoon. A BDD-based model checker for recursive programs. In *Proc. CAV'01*, LNCS 2102, pages 324–336. Springer, 2001.

[11] A. Finkel, B. Willems, and P. Wolper. A direct symbolic approach to model checking pushdown systems. *Electronic Notes in Theoretical Computer Science*, 9, 1997.

[12] 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.

[13] M. Y. Vardi and P. Wolper. Automata Theoretic Techniques for Modal Logics of Prog rams. *Journal of Computer and System Sciences*, 32:183–221, 1986.