Efficient Algorithms for Model Checking Pushdown Systems

Javier Esparza, David Hansel, Peter Rossmanith, and Stefan Schwoon

Technische Universität München, Arcisstr. 21, 80290 München, Germany
{esparza,hanseld,rossmani,schwoon}@in.tum.de

Abstract. We study model checking problems for pushdown systems and linear time logics. We show that the global model checking problem (computing the set of configurations, reachable or not, that violate the formula) can be solved in $O(g_{\mathcal{P}}{}^3 g_{\mathcal{B}}{}^3)$ time and $O(g_{\mathcal{P}}{}^2 g_{\mathcal{B}}{}^2)$ space, where $g_{\mathcal{P}}$ and $g_{\mathcal{B}}$ are the size of the pushdown system and the size of a Büchi automaton for the negation of the formula. The global model checking problem for reachable configurations can be solved in $O(g_{\mathcal{P}}{}^4 g_{\mathcal{B}}{}^3)$ time and $O(g_{\mathcal{P}}{}^4 g_{\mathcal{B}}{}^2)$ space. In the case of pushdown systems with constant number of control states (relevant for our application), the complexity becomes $O(g_{\mathcal{P}} g_{\mathcal{B}}{}^3)$ time and $O(g_{\mathcal{P}} g_{\mathcal{B}}{}^2)$ space and $O(g_{\mathcal{P}}{}^2 g_{\mathcal{B}}{}^3)$ time and $O(g_{\mathcal{P}}{}^2 g_{\mathcal{B}}{}^2)$ space, respectively. We show applications of these results in the area of program analysis and present some experimental results.

1 Introduction

Pushdown systems (PDSs) are pushdown automata seen under a different light: We are not interested in the languages they recognise, but in the transition system they generate. These are infinite transition systems having configurations of the form (control state, stack content) as states.

PDSs have already been investigated by the verification community. Model checking algorithms for both linear and branching time logics have been proposed in [1–3, 7, 11]. The model checking problem for CTL and the mu-calculus is known to be DEXPTIME-complete even for a fixed formula [1, 11]. On the contrary, the model checking problem for LTL or the linear time mu-calculus is polynomial in the size of the PDS [1, 7]. This makes linear time logics particularly interesting for PDSs. It must be observed, however, that the model checking problem for branching time logics is only exponential in the number of control states of the PDS; for a fixed number of states the algorithms of [2, 11] are polynomial.

Inspired by the work of Steffen and others on the connection between model checking and dataflow analysis (see for instance [9]), it has been recently observed that relevant dataflow problems for programs with procedures (so-called interprocedural dataflow problems), as well as security problems for Java programs can be reduced to different variants of the model checking problem for PDSs and LTL [5, 6, 8]. Motivated by this application, we revisit the model checking problem for linear time logics. We follow the symbolic approach of [1], in which infinite sets of configurations are finitely represented by multi-automata. We obtain an efficient implementation of the algorithm of [1], which was described there as 'polynomial', without further details. Our algorithm has the same time complexity and better space complexity than the algorithm of [7]. This better space complexity turns out to be important for our intended applications to dataflow analysis.

The paper is structured as follows. Sections 2 and 3 contain basic definitions, and recall some results of [1]. The 'abstract' solution of [1] to the model-checking problem is described and refined in Sections 4 to 6. In Section 7 we provide an efficient implementation for this solution. Applications and experimental results are presented in Section 9.

All proofs and some constructions have been omitted due to lack of space. They can all be found in the technical report version of this paper [4].

2 Pushdown systems and $\mathcal{P}$-automata

A pushdown system is a triplet $\mathcal{P} = (P, \Gamma, \Delta)$ where P is a finite set of *control locations*, Γ is a finite *stack alphabet*, and $\Delta \subseteq (P \times \Gamma) \times (P \times \Gamma^*)$ is a finite set of *transition rules*. If $((q,\gamma),(q',w)) \in \Delta$ then we write $\langle q,\gamma\rangle \hookrightarrow \langle q',w\rangle$ (we reserve $\rightarrow$ to denote the transition relations of finite automata).

Notice that pushdown systems have no input alphabet. We do not use them as language acceptors but are rather interested in the behaviours they generate.

A *configuration* of $\mathcal{P}$ is a pair $\langle p,w\rangle$ where $p \in P$ is a control location and $w \in \Gamma^*$ is a *stack content*. The set of all configurations is denoted by $\mathcal{C}$.

If $\langle q,\gamma\rangle \hookrightarrow \langle q',w\rangle$, then for every $v \in \Gamma^*$ the configuration $\langle q,\gamma v\rangle$ is an *immediate predecessor* of $\langle q',wv\rangle$, and $\langle q',wv\rangle$ an *immediate successor* of $\langle q,\gamma v\rangle$. The *reachability relation* $\Rightarrow$ is the reflexive and transitive closure of the immediate successor relation; the transitive closure is denoted by $\stackrel{+}{\Longrightarrow}$. A *run* of $\mathcal{P}$ is a maximal sequence of configurations such that for each two consecutive configurations $c_i c_{i+1}$, c_{i+1} is an immediate successor of c_i.

The predecessor function $pre\colon 2^{\mathcal{C}} \rightarrow 2^{\mathcal{C}}$ of $\mathcal{P}$ is defined as follows: c belongs to $pre(C)$ if some immediate successor of c belongs to C. The reflexive and transitive closure of pre is denoted by pre^*. Clearly, $pre^*(C) = \{c \in \mathcal{C} \mid \exists c' \in C.\ c \Rightarrow c'\}$. Similarly, we define $post(C)$ as the set of immediate successors of elements in C and $post^*$ as the reflexive and transitive closure of $post$.

2.1 $\mathcal{P}$-Automata

Given a pushdown system $\mathcal{P} = (P, \Gamma, \Delta)$, we use so-called $\mathcal{P}$*-automata* in order to represent sets of configurations of $\mathcal{P}$. A $\mathcal{P}$-automaton uses Γ as alphabet, and P as set of initial states (we consider automata with possibly many initial states). Formally, a $\mathcal{P}$-automaton is an automaton $\mathcal{A} = (\Gamma, Q, \delta, P, F)$ where Q is the finite set of states, $\delta \subseteq Q \times \Gamma \times Q$ is the set of *transitions*, P is the set of *initial states* and $F \subseteq Q$ the set of *final states*. We define the *transition relation* $\rightarrow \subseteq Q \times \Gamma^* \times Q$ as the smallest relation satisfying:

- $q \xrightarrow{\varepsilon} q$ for every $q \in Q$,
- if $(q, \gamma, q') \in \delta$ then $q \xrightarrow{\gamma} q'$, and
- if $q \xrightarrow{w} q''$ and $q'' \xrightarrow{\gamma} q'$ then $q \xrightarrow{w\gamma} q'$.

All the automata used in this paper are $\mathcal{P}$-automata, and so we drop the $\mathcal{P}$ from now on. An automaton *accepts* or *recognises* a configuration $\langle p, w\rangle$ if $p \xrightarrow{w} q$ for some $p \in P$, $q \in F$. The set of configurations recognised by an automaton $\mathcal{A}$ is denoted by $Conf(\mathcal{A})$. A set of configurations of $\mathcal{P}$ is *regular* if it is recognized by some automaton.

Notation In the paper, we use the symbols p, p', p'' etc., eventually with indices, to denote initial states of an automaton (i.e., the elements of P). Non-initial states are denoted by s, s', s'' etc., and arbitrary states, initial or not, by q, q', q''.

3 Model-checking problems for linear time logics

In this section we define the problems we study, as well as the automata-theoretic approach.

Let $Prop$ be a finite set of atomic propositions, and let $\Sigma = 2^{Prop}$. It is well known that the semantics of properties expressed in linear time temporal logics like LTL or the linear-time μ-calculus are ω-regular sets over the alphabet Σ, and there exist well-known algorithms which construct Büchi automata recognizing these sets. This is all we need to know about these logics in this paper in order to give model-checking algorithms for pushdown systems.

Let $\mathcal{P} = (P, \Gamma, \Delta)$ be a pushdown system, and let $\Lambda \colon (P \times \Gamma) \to \Sigma$ be a labelling function, which intuitively associates to a pair $\langle p, \gamma\rangle$ the set of propositions that are true of it. We extend this mapping to arbitrary configurations: $\langle p, \gamma w\rangle$ satisfies an atomic proposition if $\langle p, \gamma\rangle$ does.[1]

Given a formula φ of such an ω-regular logic we wish to solve these problems:

- The global model-checking problem: compute the set of configurations, reachable or not, that violate φ.
- The global model-checking problem for reachable configurations: compute the set of reachable configurations that violate φ.

Our solution to these problems uses the automata-theoretic approach. We start by constructing a Büchi automaton $\mathcal{B} = (\Sigma, Q, \delta, q_0, F)$ corresponding to the negation of φ. The product of $\mathcal{P}$ and $\mathcal{B}$ yields a Büchi pushdown system $\mathcal{BP} = ((P \times Q), \Gamma, \Delta', G)$, where

- $\langle (p, q), \gamma\rangle \hookrightarrow \langle (p', q'), w\rangle \in \Delta'$ if $\langle p, \gamma\rangle \hookrightarrow \langle p', w\rangle$, $q \xrightarrow{\sigma} q'$, and $\sigma \subseteq \Lambda(\langle p, \gamma\rangle)$.
- $(p, q) \in G$ if $q \in F$.

[1] We could also define $\Lambda \colon P \to \Sigma$, but our definition is more general, and it is also the one we need for our applications.

The global model checking problem reduces to the *accepting run problem*:

> Compute the set $\mathcal{C}_a$ of configurations c of $\mathcal{BP}$ such that $\mathcal{BP}$ has an accepting run starting from c (i.e., a run which visits infinitely often configurations with control locations in G).

Notice that the emptiness problem of Büchi pushdown systems (whether the initial configuration has an accepting run) also reduces to the accepting run problem; it suffices to check if the initial configuration belongs to the set $\mathcal{C}_a$. The following proposition characterises the configurations from which there are accepting runs.

Definition 1. *Let $\mathcal{BP} = (P, \Gamma, \Delta, G)$ be a Büchi pushdown system.*

The relation $\stackrel{r}{\Longrightarrow}$ between configurations of $\mathcal{BP}$ is defined as follows: $c \stackrel{r}{\Longrightarrow} c'$ if $c \Rightarrow \langle g, u\rangle \stackrel{+}{\Longrightarrow} c'$ for some configuration $\langle g, u\rangle$ with $g \in G$.

The head *of a transition rule $\langle p, \gamma\rangle \hookrightarrow \langle p', w\rangle$ is the configuration $\langle p, \gamma\rangle$. A head $\langle p, \gamma\rangle$ is* repeating *if there exists $v \in \Gamma^*$ such that $\langle p, \gamma\rangle \stackrel{r}{\Longrightarrow} \langle p, \gamma v\rangle$. The sets of heads and repeating heads of $\mathcal{BP}$ are denoted by H and R, respectively.*

Proposition 1. [1] *Let c be a configuration of a Büchi pushdown system $\mathcal{BP} = (P, \Gamma, \Delta, G)$ and let $R\Gamma^*$ denote the set $\{\, \langle p, \gamma w\rangle \mid \langle p, \gamma\rangle \in R,\ w \in \Gamma^* \,\}$. $\mathcal{BP}$ has an accepting run starting from c if and only if $c \in pre^*(R\Gamma^*)$.*

Proposition 1 reduces the global model-checking problem to computing the set $pre^*(R\Gamma^*)$; the global model-checking problem for reachable configurations reduces to computing $post^*(\{c\}) \cap pre^*(R\Gamma^*)$ for a given initial configuration c.

In the next sections we present a solution to these problems. We first recall the algorithm of [1] that computes $pre^*(C)$ for an arbitrary regular language C (observe that $R\Gamma^*$ is regular since R is finite). Then we present a new algorithm for computing R, obtained by modifying the algorithm for $pre^*(C)$. We also present an algorithm for computing $post^*(C)$ which is needed to solve the model-checking problem for reachable configurations.

4 Computing $pre^*(C)$ for a regular language C

Our input is an automaton $\mathcal{A}$ accepting C. Without loss of generality, we assume that $\mathcal{A}$ has no transition leading to an initial state. We compute $pre^*(C)$ as the language accepted by an automaton $\mathcal{A}_{pre^*}$ obtained from $\mathcal{A}$ 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 automaton, add a transition (p, γ, q).

Notice that all new transitions start at initial states. Let us illustrate the procedure by an example. Let $\mathcal{P} = (P, \Gamma, \Delta)$ be a pushdown system with $P = \{p_0, p_1, p_2\}$ and Δ as shown in in the left half of Figure 1. Let $\mathcal{A}$ be the automaton

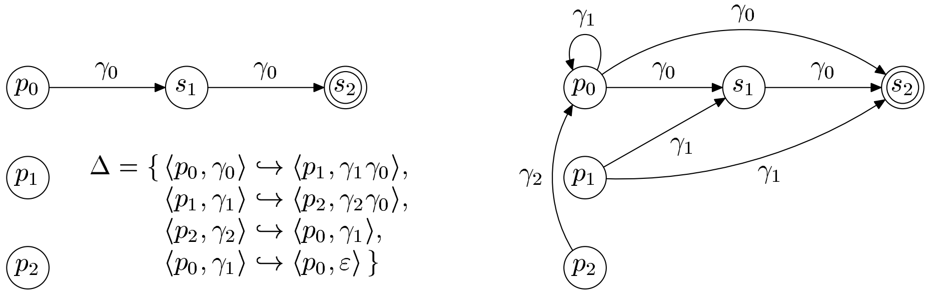


Fig. 1. The automata $\mathcal{A}$ (left) and $\mathcal{A}_{pre^*}$ (right)

that accepts the set $C = \{\langle p_0, \gamma_0\gamma_0 \rangle\}$, also shown in the figure. The result of the algorithm is shown in the right half of Figure 1.

The saturation procedure eventually reaches a fixpoint because the number of possible new transitions is finite. Correctness was proved in [1].

5 Computing the set R of repeating heads

We provide an algorithm more efficient than that of [1]. The problem of finding the repeating heads in a Büchi pushdown system is reduced to a graph-theoretic problem. More precisely, given a Büchi pushdown system $\mathcal{BP} = (P, \Gamma, \Delta, G)$ we construct a head reachability graph $\mathcal{G} = ((P \times \Gamma), E)$ whose nodes are the heads of $\mathcal{BP}$. The set of edges $E \subseteq (P \times \Gamma) \times \{0,1\} \times (P \times \Gamma)$ generates the reachability relation between heads. Define $G(p) = 1$ if $p \in G$ and $G(p) = 0$ otherwise. E consists of exactly the following edges:

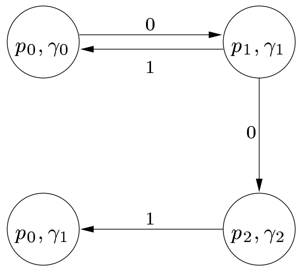
If $\langle p, \gamma \rangle \hookrightarrow \langle p'', v_1\gamma' v_2 \rangle$ and $\langle p'', v_1 \rangle \Rightarrow \langle p', \varepsilon \rangle$, then $((p,\gamma), G(p), (p',\gamma')) \in E$.
If, moreover, $\langle p'', v_1 \rangle \stackrel{r}{\Longrightarrow} \langle p', \varepsilon \rangle$, then $((p,\gamma), 1, (p',\gamma')) \in E$.

The reachability relation $\rightarrow\, \subset (P \times \Gamma) \times \{0,1\} \times (P \times \Gamma)$ is then defined as the smallest relation satisfying

- $(p,\gamma) \xrightarrow{0} (p,\gamma)$ for every $(p,\gamma) \in (P \times \Gamma)$.
- If $((p,\gamma), b, (p',\gamma')) \in E$, then $(p,\gamma) \xrightarrow{b} (p',\gamma')$.
- If $((p,\gamma), b, (p',\gamma')) \in E$ and $(p',\gamma') \xrightarrow{b'} (p'',\gamma'')$, then $(p,\gamma) \xrightarrow{b \vee b'} (p'',\gamma'')$.

Once the graph is constructed, R can be computed by exploiting the fact that some head $\langle p, \gamma \rangle$ is repeating if and only if (p,γ) is part of a strongly connected component of $\mathcal{G}$ which has an internal 1-labelled edge. The instances for which $\langle p, v \rangle \stackrel{r}{\Longrightarrow} \langle p', \varepsilon \rangle$ holds can be found with a small modification of the algorithm for pre^*.

Let $\mathcal{BP} = (P, \Gamma, \Delta, G)$ be a Büchi pushdown system with P, Γ and Δ as in the previous example and $G = \{p_2\}$. The left part of Figure 2 shows the graph $\mathcal{G}$.



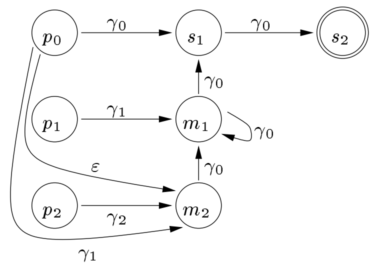


Fig. 2. Left: The graph $\mathcal{G}$. Right: $\mathcal{A}_{post^*}$

6 Computing $post^*(C)$ for a regular set C

We provide a solution for the case in which each transition rule $\langle p, \gamma \rangle \hookrightarrow \langle p', w \rangle$ of Δ satisfies $|w| \leq 2$. This restriction is not essential; our solution can easily be extended to the general case. Moreover, any pushdown system can be transformed into an equivalent one in this form, and the pushdown systems in the application discussed in Section 9 directly satisfy this condition.

Our input is an automaton $\mathcal{A}$ accepting C. Without loss of generality, we assume that $\mathcal{A}$ has no transition leading to an initial state. We compute $post^*(C)$ as the language accepted by an automaton $\mathcal{A}_{post^*}$ with ϵ-moves. We denote the relation $(\xrightarrow{\varepsilon})^* \xrightarrow{\gamma} (\xrightarrow{\varepsilon})^*$ by $\overset{\gamma}{\Longrightarrow}$. $\mathcal{A}_{post^*}$ is obtained from $\mathcal{A}$ in two stages:

- Add to $\mathcal{A}$ a new state r for each transition rule $r \in \Delta$ of the form $\langle p, \gamma \rangle \hookrightarrow \langle p', \gamma'\gamma'' \rangle$, and a transition (p', γ', r).
- Add new transitions to $\mathcal{A}$ according to the following saturation rules:

> If $\langle p, \gamma \rangle \hookrightarrow \langle p', \epsilon \rangle \in \Delta$ and $p \overset{\gamma}{\Longrightarrow} q$ in the current automaton, add a transition (p', ϵ, q).
>
> If $\langle p, \gamma \rangle \hookrightarrow \langle p', \gamma' \rangle \in \Delta$ and $p \overset{\gamma}{\Longrightarrow} q$ in the current automaton, add a transition (p', γ', q).
>
> If $r = \langle p, \gamma \rangle \hookrightarrow \langle p', \gamma'\gamma'' \rangle \in \Delta$ and $p \overset{\gamma}{\Longrightarrow} q$ in the current automaton, add a transition (r, γ'', q).

Consider again the pushdown system $\mathcal{P}$ and the automaton $\mathcal{A}$ from Figure 1. Then the automaton shown in the right part of Figure 2 is the result of the algorithm above and accepts $post^*(\{\langle p_0, \gamma_0\gamma_0 \rangle\})$.

7 Efficient Algorithms

In this section we present efficient implementations of the abstract algorithms given in sections 4 through 6. We restrict ourselves to pushdown systems which satisfy $|w| \leq 2$ for every rule $\langle p, \gamma \rangle \hookrightarrow \langle p', w \rangle$; any pushdown system can be put into such a normal form with linear size increase.

Algorithm 1
Input: a pushdown system $\mathcal{P} = (P, \Gamma, \Delta)$ in normal form;
a $\mathcal{P}$-Automaton $\mathcal{A} = (\Gamma, Q, \delta, P, F)$ without transitions into P
Output: the set of transitions of $\mathcal{A}_{pre^*}$

$\mathit{rel} \leftarrow \emptyset$; $\mathit{trans} \leftarrow \delta$; $\Delta' \leftarrow \emptyset$;
for all $\langle p, \gamma \rangle \hookrightarrow \langle p', \varepsilon \rangle \in \Delta$ **do** $\mathit{trans} \leftarrow \mathit{trans} \cup \{(p, \gamma, p')\}$;
while $\mathit{trans} \neq \emptyset$ **do**
  pop $t = (q, \gamma, q')$ from trans;
  if $t \notin \mathit{rel}$ **then**
    $\mathit{rel} \leftarrow \mathit{rel} \cup \{t\}$;
    for all $\langle p_1, \gamma_1 \rangle \hookrightarrow \langle q, \gamma \rangle \in (\Delta \cup \Delta')$ **do**
      $\mathit{trans} \leftarrow \mathit{trans} \cup \{(p_1, \gamma_1, q')\}$;
    for all $\langle p_1, \gamma_1 \rangle \hookrightarrow \langle q, \gamma\gamma_2 \rangle \in \Delta$ **do**
      $\Delta' \leftarrow \Delta' \cup \{\langle p_1, \gamma_1 \rangle \hookrightarrow \langle q', \gamma_2 \rangle\}$;
      for all $(q', \gamma_2, q'') \in \mathit{rel}$ **do**
        $\mathit{trans} \leftarrow \mathit{trans} \cup \{(p_1, \gamma_1, q'')\}$;
return rel

7.1 Computing $pre^*(C)$

Given an automaton $\mathcal{A}$ accepting the set of configurations C, we compute $pre^*(C)$ by constructing the automaton $\mathcal{A}_{pre^*}$.

Algorithm 1 computes the transitions of $\mathcal{A}_{pre^*}$, implementing the saturation rule from section 4. The sets *rel* and *trans* contain the transitions that are known to belong to $\mathcal{A}_{pre^*}$; *rel* contains the transitions that have already been examined. No transition is examined more than once.

The idea of the algorithm is to avoid unnecessary operations. When we have a rule $\langle p, \gamma \rangle \hookrightarrow \langle p', \gamma'\gamma'' \rangle$, we look out for pairs of transitions $t_1 = (p', \gamma', q')$ and $t_2 = (q', \gamma'', q'')$ (where q', q'' are arbitrary states) so that we may insert (p, γ, q'') – but we don't know in which order such transitions appear in *trans*. If every time we see a transition like t_2 we check the existence of t_1, many checks might be negative and waste time to no avail. However, once we see t_1 we know that all subsequent transitions (q', γ'', q'') must lead to (p, γ, q''). It so happens that the introduction of an extra rule $\langle p, \gamma \rangle \hookrightarrow \langle q', \gamma'' \rangle$ is enough to take care of just these cases. We collect these extra rules in a set called Δ'; this notation should make it clear that the pushdown system itself is not changed. Δ' is merely needed for the computation and can be thrown away afterwards.

For a better illustration, consider again the example shown in Figure 1.

The initialisation phase evaluates the ε-rules and adds (p_0, γ_1, p_0). When the latter is taken from *trans*, the rule $\langle p_2, \gamma_2 \rangle \hookrightarrow \langle p_0, \gamma_1 \rangle$ is evaluated and (p_2, γ_2, p_0) is added. This, in combination with (p_0, γ_0, s_1) and the rule $\langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_2, \gamma_2\gamma_0 \rangle$, leads to (p_1, γ_1, s_1), and Δ' now contains $\langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_0, \gamma_0 \rangle$. We now have $p_1 \xrightarrow{\gamma_1} s_1 \xrightarrow{\gamma_0} s_2$, so the next step adds (p_0, γ_0, s_2), and Δ' is extended by $\langle p_0, \gamma_0 \rangle \hookrightarrow \langle s_1, \gamma_0 \rangle$. Because of Δ', (p_0, γ_0, s_2) leads to (p_1, γ_1, s_2). Finally, Δ' is extended by $\langle p_0, \gamma_0 \rangle \hookrightarrow \langle s_2, \gamma_0 \rangle$, but no other transitions can be added and the algorithm terminates.

Algorithm 2
Input: a Büchi pushdown system $\mathcal{BP} = (P, \Gamma, \Delta, G)$ in normal form
Output: the set of repeating heads in $\mathcal{BP}$

$\mathit{rel} \leftarrow \emptyset$; $\mathit{trans} \leftarrow \emptyset$; $\Delta' \leftarrow \emptyset$;
for all $\langle p, \gamma \rangle \hookrightarrow \langle p', \varepsilon \rangle \in \Delta$ **do**
  $\mathit{trans} \leftarrow \mathit{trans} \cup \{(p, [\gamma, G(p)], p')\}$;
while $\mathit{trans} \neq \emptyset$ **do**
  pop $t = (p, [\gamma, b], p')$ from trans;
  if $t \notin \mathit{rel}$ **then**
    $\mathit{rel} \leftarrow \mathit{rel} \cup \{t\}$;
    for all $\langle p_1, \gamma_1 \rangle \hookrightarrow \langle p, \gamma \rangle \in \Delta$ **do**
      $\mathit{trans} \leftarrow \mathit{trans} \cup \{(p_1, [\gamma_1, b \vee G(p_1)], p')\}$;
    for all $\langle p_1, \gamma_1 \rangle \overset{b'}{\hookrightarrow} \langle p, \gamma \rangle \in \Delta'$ **do**
      $\mathit{trans} \leftarrow \mathit{trans} \cup \{(p_1, [\gamma_1, b \vee b'], p')\}$;
    for all $\langle p_1, \gamma_1 \rangle \hookrightarrow \langle p, \gamma\gamma_2 \rangle \in \Delta$ **do**
      $\Delta' \leftarrow \Delta' \cup \{\langle p_1, \gamma_1 \rangle \overset{b \vee G(p_1)}{\hookrightarrow} \langle p', \gamma_2 \rangle\}$;
      for all $(p', [\gamma_2, b'], p'') \in \mathit{rel}$ **do**
        $\mathit{trans} \leftarrow \mathit{trans} \cup \{(p_1, [\gamma_1, b \vee b' \vee G(p_1)], p'')\}$;

$R \leftarrow \emptyset$; $E \leftarrow \emptyset$;
for all $\langle p, \gamma \rangle \hookrightarrow \langle p', \gamma' \rangle \in \Delta$ **do** $E \leftarrow E \cup \{((p, \gamma), G(p), (p', \gamma'))\}$;
for all $\langle p, \gamma \rangle \overset{b}{\hookrightarrow} \langle p', \gamma' \rangle \in \Delta'$ **do** $E \leftarrow E \cup \{((p, \gamma), b, (p', \gamma'))\}$;
for all $\langle p, \gamma \rangle \hookrightarrow \langle p', \gamma'\gamma'' \rangle \in \Delta$ **do** $E \leftarrow E \cup \{((p, \gamma), G(p), (p', \gamma'))\}$;
find strongly connected components in $\mathcal{G} = ((P \times \Gamma), E)$;
for all components C **do**
  if C has a 1-edge **then** $R \leftarrow R \cup C$;
return R

Theorem 1. *Let $\mathcal{P} = (P, \Gamma, \Delta)$ be a pushdown system and $\mathcal{A} = (\Gamma, Q, \delta, P, F)$ be an automaton. There exists an automaton $\mathcal{A}_{pre^*}$ recognising $pre^*(\mathit{Conf}(\mathcal{A}))$. Moreover, $\mathcal{A}_{pre^*}$ can be constructed in $O(n_Q^2 n_\Delta)$ time and $O(n_Q n_\Delta + n_\delta)$ space, where $n_Q = |Q|$, $n_\delta = |\delta|$, and $n_\Delta = |\Delta|$.*

Observe that a naive implementation of the abstract procedure of section 4 leads to an $O(n_{\mathcal{P}}^2 n_{\mathcal{A}}^3)$ time and $O(n_{\mathcal{P}} n_{\mathcal{A}})$ space algorithm, where $n_{\mathcal{P}} = |P| + |\Delta|$, and $n_{\mathcal{A}} = |Q| + |\delta|$.

7.2 Computing the set of repeating heads

Given a Büchi pushdown system (P, Γ, Δ, G) we want to compute the set R introduced in section 3, i.e. the set of transition heads $\langle p, \gamma \rangle$ that satisfy $\langle p, \gamma \rangle \overset{r}{\Longrightarrow} \langle p, \gamma v \rangle$ for some $v \in \Gamma^*$.

Algorithm 2 runs in two phases. In the first phase, the cases for which $\langle p, w \rangle \Rightarrow \langle p', \varepsilon \rangle$ holds are computed. To this end, we employ the algorithm for pre^* on the set $\{\, \langle p, \varepsilon \rangle \mid p \in P \,\}$. Then every resulting transition (p, γ, p') signifies that $\langle p, \gamma \rangle \Rightarrow \langle p', \varepsilon \rangle$ holds.

However, we also need the information whether $\langle p, \gamma \rangle \overset{r}{\Longrightarrow} \langle p', \varepsilon \rangle$ holds. To this end, we enrich the automaton's alphabet; instead of transitions of the form

(p, γ, p') we now have transitions $(p, [\gamma, b], p')$ where b is a boolean. The meaning of a transition $(p, [\gamma, 1], p')$ should be that $\langle p, \gamma \rangle \stackrel{r}{\Longrightarrow} \langle p', \varepsilon \rangle$.

The second phase of the algorithm constructs the graph $\mathcal{G}$ using the results of the first phase. Finally, Tarjan's algorithm [10] can be used to find the strongly connected components of $\mathcal{G}$ and thus to determine the repeating heads.

In the example from Figure 2, the components of $\mathcal{G}$ are $\{(p_0, \gamma_0), (p_1, \gamma_1)\}$, $\{(p_0, \gamma_1)\}$, and $\{(p_2, \gamma_2)\}$. Of these, the first one has an internal 1-edge, meaning that $\langle p_0, \gamma_0 \rangle$ and $\langle p_1, \gamma_1 \rangle$ are the repeating heads of this example.

Theorem 2. *Let $\mathcal{BP} = (P, \Gamma, \Delta, G)$ be a Büchi pushdown system. The set of repeating heads R can be computed in $O(n_P^2 n_\Delta)$ time and $O(n_P n_\Delta)$ space, where $n_P = |P|$ and $n_\Delta = |\Delta|$.*

A direct implementation of the procedure of [1] for computing the repeating heads leads to $O(n_{\mathcal{BP}}^5)$ time and $O(n_{\mathcal{BP}}^2)$ space, where $n_{\mathcal{BP}} = |P| + |\Delta|$.

7.3 Computing $post^*(C)$

Given a regular set of configurations C, we want to compute $post^*(C)$, i.e. the set of successors of C. Without loss of generality, we assume that $\mathcal{A}$ has no ε-transitions.

Algorithm 3 calculates the transitions of $\mathcal{A}_{post^*}$, implementing the saturation rule from section 6. The approach is in some ways similar to the solution for pre^*; again we use *trans* and *rel* to store the transitions that we need to examine. Note that transitions from states outside of P go directly to *rel* since these states cannot occur in rules.

The algorithm is very straightforward. We start by including the transitions of $\mathcal{A}$; then, for every transition that is known to belong to $\mathcal{A}_{post^*}$, we find its successors. A noteworthy difference to the algorithm in section 6 is the treatment of ε-moves: ε-transitions are eliminated and simulated with non-ε-transitions; we maintain the sets $eps(q)$ for every state q with the meaning that whenever there should be an ε-transition going from p to q, $eps(q)$ contains p.

Again, consider the example in Figure 2. In that example, m_1 is the node associated with the rule $\langle p_0, \gamma_0 \rangle \hookrightarrow \langle p_1, \gamma_1 \gamma_0 \rangle$, and m_2 is associated with $\langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_2, \gamma_2 \gamma_0 \rangle$. The transitions (p_1, γ_1, m_1) and (m_1, γ_0, s_1) are a consequence of (p_0, γ_0, s_1); the former leads to (p_2, γ_2, m_2) and (m_2, γ_0, m_1) and, in turn, to (p_0, γ_1, m_2). Because of $\langle p_0, \gamma_1 \rangle \hookrightarrow \langle p_0, \varepsilon \rangle$, we now need to simulate an ε-move from p_0 to m_2. This is done by making copies of all the transitions that leave m_2; in this example, (m_2, γ_0, m_1) is copied and changed to (p_0, γ_0, m_1). The latter finally leads to (p_1, γ_1, m_1) and (m_1, γ_0, m_1). Figure 3 shows the result, similar to Figure 2 but with the ε-transition resolved.

Theorem 3. *Let $\mathcal{P} = (P, \Gamma, \Delta)$ be a pushdown system, and $\mathcal{A} = (\Gamma, Q, \delta, P, F)$ be an automaton. There exists an automaton $\mathcal{A}_{post^*}$ recognising $post^*(Conf(\mathcal{A}))$. Moreover, $\mathcal{A}_{post^*}$ can be constructed in $O(n_P n_\Delta (n_Q + n_\Delta) + n_P n_\delta)$ time and space, where $n_P = |P|$, $n_\Delta = |\Delta|$, $n_Q = |Q|$, and $n_\delta = |\delta|$.*

Algorithm 3
Input: a pushdown system $\mathcal{P} = (P, \Gamma, \Delta)$ in normal form;
a $\mathcal{P}$-Automaton $\mathcal{A} = (\Gamma, Q, \delta, P, F)$ without transitions into P
Output: the automaton $\mathcal{A}_{post^*}$

```
trans ← δ ∩ (P × Γ × Q);
rel ← δ \ trans;  Q' ← Q;  F' ← F;
for all r = ⟨p, γ⟩ ↪ ⟨p', γ1γ2⟩ ∈ Δ do
  Q' ← Q' ∪ {q_r};
  trans ← trans ∪ {(p', γ1, q_r)};
for all q ∈ Q' do eps(q) ← ∅;
while trans ≠ ∅ do
  pop t = (p, γ, q) from trans;
  if t ∉ rel then
    rel ← rel ∪ {t};
    for all ⟨p, γ⟩ ↪ ⟨p', ε⟩ ∈ Δ do
      if p' ∉ eps(q) then
        eps(q) ← eps(q) ∪ {p'};
        for all (q, γ', q') ∈ rel do
          trans ← trans ∪ {(p', γ', q')};
        if q ∈ F' then F' ← F' ∪ {p'};
    for all ⟨p, γ⟩ ↪ ⟨p', γ1⟩ ∈ Δ do
      trans ← trans ∪ {(p', γ1, q')};
    for all r = ⟨p, γ⟩ ↪ ⟨p', γ1γ2⟩ ∈ Δ do
      rel ← rel ∪ {(q_r, γ2, q)};
      for all p'' ∈ eps(q_r) do
        trans ← trans ∪ {(p'', γ2, q)};
return (Γ, Q', rel, P, F')
```

In [7] the same problem was considered (with different restrictions on the rules in the pushdown system). The complexity of the *post** computation for the initial configuration was given as $O(n_{\mathcal{P}}^3)$ where $n_{\mathcal{P}}$ translates to $n_P + n_\Delta$. An extension to compute *post**(C) for arbitrary regular sets C is also proposed. The different restrictions on the pushdown rules make a more detailed comparison difficult, but it is safe to say that our algorithm is at least as good as the one in [7]. Also, we give an explicit bound for the computation of *post** for arbitrary regular sets of configurations.

8 The Model-checking Problem

Using the results from the previous section we can now compute the complexity of the problems presented in section 3. The following steps are necessary to solve the global model-checking problem for a given formula φ:

- Construct the Büchi automaton $\mathcal{B} = (\Sigma, Q, \delta, q_0, F)$ corresponding to $\neg\varphi$.
- Compute $\mathcal{BP}$ as the product of $\mathcal{B}$ and the pushdown system $\mathcal{P} = (P, \Gamma, \Delta)$.
- Compute the set of repeating heads R of $\mathcal{BP}$.

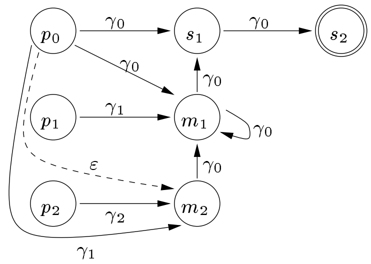


Fig. 3. $\mathcal{A}_{post^*}$ as computed by Algorithm 3.

- Construct an automaton $\mathcal{A}$ accepting $R\Gamma^*$. $\mathcal{A}$ has one final state r and contains transitions (p, γ, r) for every $\langle p, \gamma\rangle \in R$ and (r, γ, r) for every $\gamma \in \Gamma$.
- Compute $\mathcal{A}_{pre^*}$. A configuration $\langle p, w\rangle$ violates φ exactly if $\langle (p, q_0), w\rangle$ is accepted by $\mathcal{A}_{pre^*}$.

The dominant factors in these computations are the time needed to compute the repeating heads, and the space needed to store $\mathcal{A}_{pre^*}$.

Theorem 4. *Let $g_\mathcal{P}$ denote the size of $\mathcal{P}$ and $g_\mathcal{B}$ the size of $\mathcal{B}$. The global model-checking problem can be solved in $O({g_\mathcal{P}}^3 {g_\mathcal{B}}^3)$ time and $O({g_\mathcal{P}}^2 {g_\mathcal{B}}^2)$ space.*

In [7] an algorithm is presented for deciding if the initial configuration satisfies a given LTL property. (The problem of obtaining a representation for the set of configurations violating the property is not discussed.) The algorithm takes cubic time but also cubic space in the size of the pushdown system. More precisely, it is based on a saturation routine which requires $\Theta({g_\mathcal{P}}^3 {g_\mathcal{B}}^3)$ space. Observe that the space consumption is Θ, and not O. Our solution implies therefore an improvement in the space complexity without losses in time. To solve the problem for reachable configurations we need these additional steps:

- Rename the states (p, q_0) into p for all $p \in P$ and take P as the new set of initial states.
- Compute $post^*$ for the initial configuration of $\mathcal{P}$.
- Compute the intersection of $\mathcal{A}_{pre^*}$ and $\mathcal{A}_{post^*}$. The resulting automaton $\mathcal{A}_i$ accepts the set of reachable configurations violating φ.

Theorem 5. *The global model-checking problem for reachable configurations can be solved in $O({g_\mathcal{P}}^4 {g_\mathcal{B}}^3)$ time and $O({g_\mathcal{P}}^4 {g_\mathcal{B}}^2)$ space.*

9 Application

As an application, we use pushdown systems to model sequential programs with procedures (written in C or Java, for instance). We concentrate on the control flow and abstract away information about data. The model establishes a relation between the control states of a program and the configurations of the corresponding pushdown system.

The model is constructed in two steps. In the first step, we represent the program by a system of flow graphs, one for each procedure. The nodes of a flow graph correspond to control points in the procedure, and its edges are annotated with statements, e.g. calls to other procedures. Control flow is interpreted non-deterministically since we abstract from the values of variables.

Given a system of flow graphs with a set N of control points, we construct a pushdown system with N as its stack alphabet. More precisely, a configuration $\langle p, nw\rangle$, $w \in N^*$ represents the situation that execution is currently at control point n where w represents the return addresses of the calling procedures. Pushdown systems of this kind need only one single control state (called p in the following). The transition rules of such a pushdown system are:

- $\langle p, n\rangle \hookrightarrow \langle p, n'\rangle$ if control passes from n to n' without a procedure call.
- $\langle p, n\rangle \hookrightarrow \langle p, f_0 n'\rangle$ if an edge between point n and n' contains a call to procedure f, assuming that f_0 is f's entry point. n' can be seen as the return address of that call.
- $\langle p, n\rangle \hookrightarrow \langle p, \varepsilon\rangle$ if an edge leaving n contains a return statement.

Let us examine how the special structure of these systems affects the complexity of the model-checking problem. Under the assumption that the number of control states is one (or, more generally, constant), the number of states in the Büchi pushdown system $\mathcal{BP}$ only depends on the formula, and we get the following results:

Theorem 6. *If the number of control states in $\mathcal{P}$ is constant, the global model-checking problem can be solved in $O(g_{\mathcal{P}} g_{\mathcal{B}}{}^3)$ time and $O(g_{\mathcal{P}} g_{\mathcal{B}}{}^2)$ space, and the problem for reachable configurations in $O(g_{\mathcal{P}}{}^2 g_{\mathcal{B}}{}^3)$ time and $O(g_{\mathcal{P}}{}^2 g_{\mathcal{B}}{}^2)$ space.*

9.1 A small example

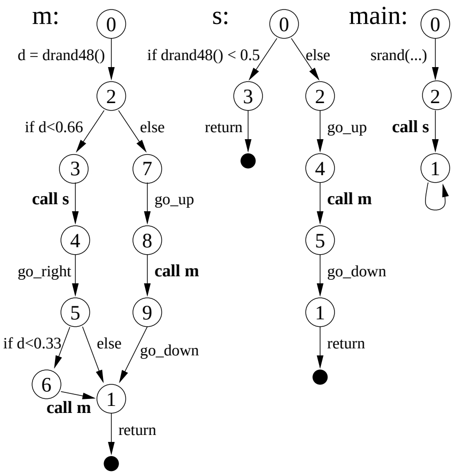
As an example, consider the program in Figure 4. This program controls a plotter, creating random bar graphs via the commands *go_up*, *go_right*, and *go_down*. Among the correctness properties is the requirement that an upward movement should never be immediately followed by a downward movement and vice versa which we shall verify in the following.

```
void m() {
    double d = drand48();
    if (d < 0.66) {
        s(); go_right();
        if (d < 0.33) m();
    } else {
        go_up(); m(); go_down();
    }
}

void s() {
    if (drand48() < 0.5) return;
    go_up(); m(); go_down();
}

main() {
    srand48(time(NULL));
    s();
}
```

Fig. 4. An example program.



$$
\begin{array}{ll}
\langle p, m_0\rangle \hookrightarrow \langle p, m_2\rangle & \langle p, m_9\rangle \hookrightarrow \langle p, m_1\rangle \\
\langle p, m_2\rangle \hookrightarrow \langle p, m_3\rangle & \langle p, m_1\rangle \hookrightarrow \langle p, \varepsilon\rangle \\
\langle p, m_3\rangle \hookrightarrow \langle p, s_0\, m_4\rangle & \langle p, s_0\rangle \hookrightarrow \langle p, s_2\rangle \\
\langle p, m_4\rangle \hookrightarrow \langle p, m_5\rangle & \langle p, s_0\rangle \hookrightarrow \langle p, s_3\rangle \\
\langle p, m_5\rangle \hookrightarrow \langle p, m_1\rangle & \langle p, s_3\rangle \hookrightarrow \langle p, \varepsilon\rangle \\
\langle p, m_5\rangle \hookrightarrow \langle p, m_6\rangle & \langle p, s_2\rangle \hookrightarrow \langle p, s_4\rangle \\
\langle p, m_6\rangle \hookrightarrow \langle p, m_0\, m_1\rangle & \langle p, s_4\rangle \hookrightarrow \langle p, m_0\, s_5\rangle \\
\langle p, m_2\rangle \hookrightarrow \langle p, m_7\rangle & \langle p, s_5\rangle \hookrightarrow \langle p, s_1\rangle \\
\langle p, m_7\rangle \hookrightarrow \langle p, m_8\rangle & \langle p, s_1\rangle \hookrightarrow \langle p, \varepsilon\rangle \\
\langle p, m_8\rangle \hookrightarrow \langle p, m_0\, m_9\rangle & \\
\langle p, main_0\rangle \hookrightarrow \langle p, main_2\rangle & \\
\langle p, main_2\rangle \hookrightarrow \langle p, s_0\, main_1\rangle & \\
\langle p, main_1\rangle \hookrightarrow \langle p, main_1\rangle &
\end{array}
$$

Fig. 5. Flowgraph of the program in Figure 4 (left) and associated PDS (right)

The left side of Figure 5 displays the set of flow graphs created from the original program. Calls to external functions are treated as ordinary statements. (In this example we assume that the *go...* functions are external.) The procedure *main* ends in an infinite loop which ensures that all executions are infinite. The right side shows the resulting pushdown system. The initial configuration is $\langle p, main_0\rangle$. The desired properties can be expressed as follows:

$$\mathbf{G}(up \rightarrow (\neg down\ \mathbf{U}\ right)) \quad \text{and} \quad \mathbf{G}(down \rightarrow (\neg up\ \mathbf{U}\ right))$$

where the atomic proposition up is true of configurations $\langle p, m_7 w\rangle$ and $\langle p, s_2 w\rangle$, i.e. those that correspond to program points in which *go_up* will be the next statement. Similarly, $down$ is true of configurations with m_9 or s_5 as the topmost stack symbol, and $right$ is true of m_4. Analysis of the program with our methods yields that both properties are fulfilled.

9.2 Experimental results

Apart from the example, we solved the global model-checking problem on a series of randomly generated flow graphs. These flow graphs model programs with procedures. The structure of each statement was decided randomly; the average proportion of sequences, branches and loops was 0.6 : 0.2 : 0.2 (these numbers were taken from the literature). After generating one flow graph for each procedure we connected them by inserting procedure calls, making sure that each procedure was indeed reachable from the start. The formulas we checked were of the form $\mathbf{G}(n \rightarrow \mathbf{F}n')$, where n and n' were random control states.

Table 1 lists the execution times in seconds for programs with an average of 20 resp. 40 lines per procedure. One fifth of the statements in these programs contained a procedure call. Results are given for programs with recursive and mutual procedure calls. The table lists the times needed to compute the sets

	avg. 20 lines/procedure				avg. 40 lines/procedure			
lines	*RH*	pre^*	total	space	*RH*	pre^*	total	space
				recursive procedure calls				
1000	0.05	0.02	0.23	0.91 M	0.05	0.02	0.20	0.84 M
2000	0.12	0.05	0.46	1.82 M	0.14	0.05	0.48	1.84 M
5000	0.36	0.13	1.23	4.46 M	0.34	0.14	1.20	4.29 M
10000	0.76	0.26	2.55	8.79 M	0.74	0.30	2.52	8.62 M
20000	1.64	0.55	5.43	17.56 M	1.75	0.64	5.57	17.69 M
				mutual procedure calls				
1000	0.06	0.03	0.24	0.97 M	0.05	0.03	0.22	0.90 M
2000	0.14	0.06	0.49	1.91 M	0.14	0.08	0.49	1.87 M
5000	0.43	0.18	1.35	4.72 M	0.42	0.18	1.32	4.67 M
10000	0.97	0.39	2.91	9.68 M	0.95	0.39	2.84	9.39 M
20000	2.10	0.81	6.21	19.27 M	2.08	0.84	6.15	18.93 M

Table 1. Results for programs with recursive and mutual procedure calls

of repeating heads, pre^*, and the total time for the model-checking which includes several other tasks. In fact, in our experiments the majority of time was spent reading the pushdown system and computing the product with a Büchi automaton. Also, memory usage is given. All computations were carried out on an Ultrasparc 60 with sufficient amount of memory.

Obviously, these experiments can only give a rough impression of what the execution times would be like when analysing real programs. However, these experiments already constitute 'stress tests', i.e. their construction is based on certain exaggerated assumptions which affect execution times negatively. For instance, the number of procedure calls is very high, and with mutual calls we allowed every procedure to call every other procedure (which greatly increases the time needed to find repeating heads). We have run preliminary experiments with three real programs of ten to thirty thousand lines, and the time for the model-checking was never more than a few seconds even for the largest program.

10 Conclusions

We have presented detailed algorithms for model-checking linear time logics on pushdown systems. The global model-checking problem can be solved in $O({g_\mathcal{P}}^3 {g_\mathcal{B}}^3)$ time and $O({g_\mathcal{P}}^2 {g_\mathcal{B}}^2)$ space. In the case of pushdown systems with one single control state the problem can be solved in $O(g_\mathcal{P} {g_\mathcal{B}}^3)$ time and $O(g_\mathcal{P} {g_\mathcal{B}}^2)$ space. Our results improve on [7], where the model-checking problem (i.e., deciding if an initial configuration satisfies a property, without computing the set of configurations violating the property) was solved in $O(n^3)$ time but also $O(n^3)$ space in the size of the pushdown system.

Our work needs to be carefully compared to that on branching time logics. For arbitrary pushdown systems, linear time logics can be checked in polynomial time in the size of the system, while checking CTL provably requires exponential time. However, in the case of pushdown systems with one single control state, a modal mu-calculus formula of alternation depth k can be checked in time $O(n^k)$ [3, 11], where n is the size of the system. In this case, linear time formulas can be checked in $O(n)$ time, and so we obtain linear complexity in the size of the system for both linear time logics and for the alternation-free mu-calculus.

The information provided by the branching-time and linear-time algorithms is different. Walukiewicz's algorithm [11] only says whether the initial configuration satisfies the property or not. The algorithm of Burkart and Steffen [2, 3] returns a set of *predicate transformers*, one for each stack symbol; the predicate transformer for the symbol X shows which formulas hold for a stack content $X\alpha$ as a function of the formulas that hold in α.[2] Essentially, this allows to determine for each symbol X if there exists some stack content of the form $X\alpha$ that violates the formula, but doesn't tell which these stack contents are. Finally, our algorithm returns a finite representation of the set of configurations that violate the formula.

Whether one needs all the information provided by our algorithm or not depends on the application. For certain dataflow analysis problems we wish to compute the set of control locations p such that some reachable configuration $\langle p, w\rangle$ violates the property. This information can be efficiently computed by the algorithm of [2, 3]. Sometimes we may need more. For instance, in [8] an approach is presented to the verification of control flow based security properties. Systems are modelled by pushdown automata, and security properties as properties that all reachable configurations should satisfy. It is necessary to determine which configurations violate the security property to modify the system accordingly. The model checking algorithm of [2, 3] seems to be inadequate in this case. It may be possible to modify the algorithm in order to obtain an automata representation of the configurations that violate the formula. This is surely an interesting research question.

References

1. A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *Proceedings of CONCUR '97*, LNCS 1243, pages 135–150, 1997.
2. O. Burkart and B. Steffen. Composition, decomposition and model checking of pushdown processes. *Nordic Journal of Computing*, 2(2):89–125, 1995.
3. O. Burkart and B. Steffen. Model–checking the full–modal mu–calculus for infinite sequential processes. In *Proceedings of ICALP'97, Bologna*, LNCS 1256, pages 419–429, 1997.

[2] Not for arbitrary formulas, but only for those in the closure of the original formula to be checked.

4. J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon. Efficient algorithms for model checking pushdown systems. Technical Report TUM-I0002, Technische Universität München, Department of Computer Science, Feb. 2000.
5. J. Esparza and J. Knoop. An automata-theoretic approach to interprocedural dataflow analysis. In *Proceedings of TACAS '99*, LNCS 1578, pages 14 – 30, 1999.
6. J. Esparza and A. Podelski. Efficient algorithms for pre* and post* on interprocedural parallel flow graphs. In *Proceedings of POPL '00*, 2000.
7. A. Finkel, B. Willems, and P. Wolper. A direct symbolic approach to model checking pushdown systems. *Electronic Notes in Theoretical Computer Science*, 9, 1997.
8. T. Jensen, D. Le Métayer, and T. Thorn. Verification of control flow based security properties. Technical Report 1210, IRISA, 1998.
9. D. Schmidt and B. Steffen. Program analysis as model checking of abstract interpretations. In *Proceedings of SAS'98, Pisa*, LNCS 1503, pages 351–380, 1998.
10. R. E. Tarjan. Depth first search and linear graph algorithms. In *SICOMP 1*, pages 146–160, 1972.
11. I. Walukiewicz. Pushdown Processes: Games and Model Checking. In *CAV'96*. LNCS 1102, 1996.