TECHNISCHE UNIVERSITÄT MÜNCHEN

INSTITUT FÜR INFORMATIK

Sonderforschungsbereich 342:
Methoden und Werkzeuge für die Nutzung
paralleler Rechnerarchitekturen

Efficient Algorithm for Model Checking Pushdown Systems

Javier Esparza, David Hansel, Peter Rossmanith, Stefan Schwoon

TUM-I0002
SFB-Bericht Nr. 342/01/00 A
Januar 00

TUM–INFO–01-I0002-0/1.–FI

Anforderungen an: Prof. Dr. A. Bode
Sprecher SFB 342
Institut für Informatik
Technische Universität München
D-80290 München, Germany

Druck: Fakultät für Informatik der
Technischen Universität München

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, 2, 3, 6, 10]. The model checking problem for CTL and the mu-calculus is known to be DEXPTIME-complete even for a fixed formula [1, 10]. On the contrary, the model checking problem for LTL or the linear time mu-calculus is polynomial in the size of the PDS [1, 6]. 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, 10] are polynomial.

Inspired by the work of Steffen and others on the connection between model checking and dataflow analysis (see for instance [8]), 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 [4, 5, 7]. 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 [6]. 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 through 6. In Section 7 we provide an efficient implementation for this solution. Applications and experimental results are presented in Section 9.

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 $post$.

$\mathcal{P}$-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 $\mathit{Conf}(\mathcal{A})$. 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 exactly 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 $q \in F$.

Note We assume that all Büchi pushdown systems in this paper are generated as above and denote their control locations by just P, rather than $P \times Q$, to simplify the notation.

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 3.1 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 $g \in G$.

[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.

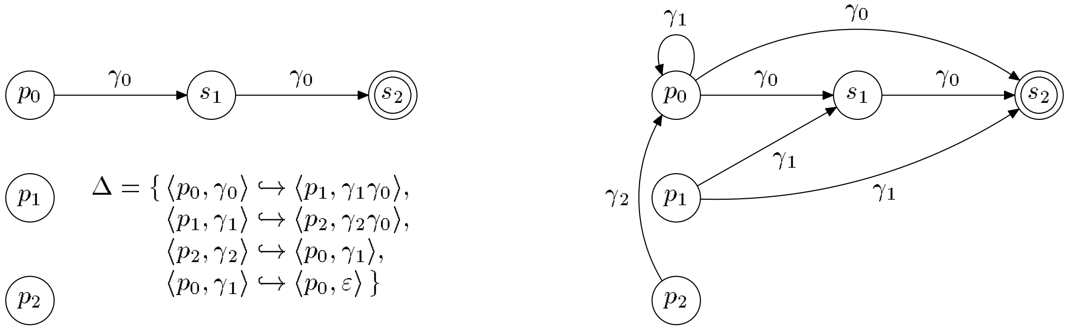


Figure 1: The automata $\mathcal{A}$ (left) and $\mathcal{A}_{pre^*}$ (right)

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 3.1 [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 3.1 reduces the global model-checking problem to computing $pre^*(R\,\Gamma^*)$; the global model-checking problem for reachable configurations reduces to computing the set $post^*(\{c\}) \cap pre^*(R\,\Gamma^*)$ 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) .

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 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.

Proof: The correctness was proved in [1], but for the sake of completeness we repeat the proof here.

Let $\mathcal{A}$ be a $\mathcal{P}$-automaton, and let $\mathcal{A}_{pre^*}$ be the automaton obtained from $\mathcal{A}$ by means of the saturation rule defined above. Observe that all new transitions added by the procedure start at an initial state, i.e., an element of P. In the sequel we use the following notation:

- $q \xrightarrow[i]{w} q'$ denotes that the automaton obtained after adding i transitions to $\mathcal{A}$ contains a path labelled by w leading from q to q'; in particular, $q \xrightarrow[0]{w} q'$ denotes that $\mathcal{A}$ contains such a path.

- $q \xrightarrow{w} q'$ denotes that there is an index i satisfying $q \xrightarrow[i]{w} q'$. Equivalently, it denotes that $\mathcal{A}_{pre^*}$ contains a path labelled by w leading from q to q'.

We show that $\mathcal{A}_{pre^*}$ recognises the set $pre^*(Conf(\mathcal{A}))$. The result is proved in Theorem 4.1 at the end of this subsection. We need two preliminary lemmata.

Lemma 4.1 *For every configuration $\langle p, v\rangle \in Conf(\mathcal{A})$, if $\langle p', w\rangle \Rightarrow \langle p, v\rangle$ then $p' \xrightarrow{w} q$ for some final state q of $\mathcal{A}_{pre^*}$.*

Proof: Let $\langle p', w\rangle \stackrel{k}{\Longrightarrow} \langle p, v\rangle$ denote that we derive $\langle p, v\rangle$ from $\langle p', w\rangle$ in k steps. We proceed by induction on k.

Basis. $k = 0$. Then $p' = p$ and $w = v$. Since $\langle p, v\rangle \in Conf(\mathcal{A})$, we have $p \xrightarrow[0]{v} q$ for some final state q, and so $p \xrightarrow{v} q$, which implies $p' \xrightarrow{w} q$.

Step. $k > 0$. Then, by the definition of $\stackrel{k}{\Longrightarrow}$, there is a configuration $\langle p'', u\rangle$ such that

$$\langle p', w\rangle \stackrel{1}{\Longrightarrow} \langle p'', u\rangle \stackrel{k-1}{\Longrightarrow} \langle p, v\rangle \ .$$

We apply the induction hypothesis to $\langle p'', u\rangle \stackrel{k-1}{\Longrightarrow} \langle p, v\rangle$, and obtain

$$p'' \xrightarrow{u} q \text{ for some } q \in F \ .$$

Since $\langle p', w\rangle \stackrel{1}{\Longrightarrow} \langle p'', u\rangle$, there are γ, w_1, v_1 such that

$$w = \gamma w_1,\ u = u_1 w_1, \text{ and } \langle p', \gamma\rangle \hookrightarrow \langle p'', u_1\rangle.$$

Let q_1 be a state of $\mathcal{A}_{pre^*}$ such that

$$p'' \xrightarrow{u_1} q_1 \xrightarrow{w_1} q.$$

By the saturation rule, we have

$$p' \xrightarrow{\gamma} q_1 \xrightarrow{w_1} q \ ,$$

which implies

$$p' \xrightarrow{\gamma w_1} q \ ,$$

and since $w = \gamma w_1$, we are done. $\square$

Lemma 4.2 *Let $p \xrightarrow{w} q$ be a path of $\mathcal{A}_{pre^*}$. The following properties hold:*

(a) $\langle p, w\rangle \Rightarrow \langle p', w'\rangle$ *for a configuration* $\langle p', w'\rangle$ *such that* $p' \xrightarrow[0]{w'} q$*; moreover,*

(b) *if q is an initial state, then $w' = \varepsilon$.*

Proof:

Let i be an index such that $p \xrightarrow[i]{w} q$ holds. We shall prove (a) by induction on i. Statement (b) then follows immediately from the fact that initial states have no imcoming arcs in $\mathcal{A}$.

Basis. $i = 0$. Then $p \xrightarrow{w} q$ is a path of $\mathcal{A}$. Take $p' = p$ and $w' = w$.

Step. $i \geq 1$. Let $t = (p_1, \gamma, q')$ be the i-th transition added to $\mathcal{A}$. (Notice that we can safely write (p_1, γ, q') instead of (q_1, γ, q') because all new transitions start at an initial state.) Let j be the number of times that t is used in $p \xrightarrow[i]{w} q$.

The proof is by induction on j. If $j = 0$, then we have $p \xrightarrow[i-1]{w} q$, and (a) follows from the induction hypothesis (induction on i). So assume that $j > 0$. Then there exist u and v such that $w = u\gamma v$ and

$$p \xrightarrow[i-1]{u} p_1 \xrightarrow[i]{\gamma} q' \xrightarrow[i]{v} q \tag{0}$$

The application of the induction hypothesis (induction on i) to $p \xrightarrow[i-1]{u} p_1$ yields (notice that p_1 is an initial state, and so both (a) and (b) can be applied):

$$\langle p, u\rangle \Rightarrow \langle p_1, \varepsilon\rangle \tag{1}$$

Since the transition (p_1, γ, q') has been added by applying the saturation rule, there exist p_2 and w_2 such that

$$\langle p_1, \gamma\rangle \hookrightarrow \langle p_2, w_2\rangle \tag{2.1}$$
$$p_2 \xrightarrow[i-1]{w_2} q' \tag{2.2}$$

From (0) and (2.2) we get

$$p_2 \xrightarrow[i-1]{w_2} q' \xrightarrow[i]{v} q \tag{3}$$

Since the transition t is used in (3) less often than in (0), we can apply the induction hypothesis (induction on j) to (3), and obtain

$$\langle p_2, w_2 v\rangle \Longrightarrow \langle p', w'\rangle \tag{4.1}$$
$$p' \xrightarrow[0]{w'} q \tag{4.2}$$

Putting (1), (2.1), and (4.1) together, we get

$$\langle p, u\gamma v\rangle \overset{(1)}{\Longrightarrow} \langle p_1, \gamma v\rangle \overset{(2.1)}{\Longrightarrow} \langle p_2, w_2 v\rangle \overset{(4.1)}{\Longrightarrow} \langle p', w'\rangle \tag{5}$$

We obtain the claim from (5) and (4.2).

□

Theorem 4.1 *Let $\mathcal{A}_{pre^*}$ be the automaton obtained from $\mathcal{A}$ by exhaustive application of the saturation rule defined in Section 4. $\mathcal{A}_{pre^*}$ recognises the set $pre^*(Conf(\mathcal{A}))$.*

Proof: Let $\langle p, w\rangle$ be a configuration of $pre^*(Conf(\mathcal{A}))$. Then $\langle p, w\rangle \Rightarrow \langle p', w'\rangle$ for a configuration $\langle p', w'\rangle \in Conf(\mathcal{A})$. By Lemma 4.1, $p \xrightarrow{w} q$ for some final state q of $\mathcal{A}_{pre^*}$. So $\langle p, w\rangle$ is recognised by $\mathcal{A}_{pre^*}$.

Conversely, let $\langle p, w\rangle$ be a configuration recognised by $\mathcal{A}_{pre^*}$. Then $p \xrightarrow{w} q$ in $\mathcal{A}_{pre^*}$ for some final state q. By Lemma 4.2(a), $\langle p, w\rangle \Rightarrow \langle p', w'\rangle$ for a configuration $\langle p', w'\rangle$ such that $p' \xrightarrow[0]{w'} q$. Since q is a final state, $\langle p', w'\rangle \in Conf(\mathcal{A})$, and so $\langle p, w\rangle \in pre^*(Conf(\mathcal{A}))$. $\square$

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:

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

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 $G = \{p_2\}$. Figure 2 shows the graph $\mathcal{G}$.

The correctness of the algorithm is proved in Theorem 5.1 at the end of this section. We need one preliminary lemma:

Lemma 5.1 *Let $(p, \gamma) \rightarrow (p', \gamma')$ denote that $(p, \gamma) \xrightarrow{b} (p', \gamma')$ for some $b \in \{0, 1\}$.*

(a) *$(p, \gamma) \rightarrow (p', \gamma')$ if and only if $\langle p, \gamma\rangle \Rightarrow \langle p', \gamma' v\rangle$ for some $v \in \Gamma^*$.*

(b) *Moreover, $(p, \gamma) \xrightarrow{1} (p', \gamma')$ if and only if $\langle p, \gamma\rangle \stackrel{r}{\Longrightarrow} \langle p', \gamma' v\rangle$.*

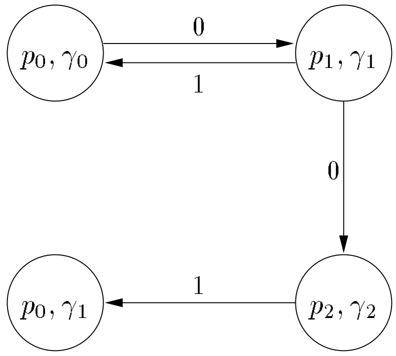


Figure 2: The graph $\mathcal{G}$

Proof:
"$\Rightarrow$": Let $(p,\gamma) \xrightarrow[i]{b} (p',\gamma')$ denote that (p',γ') is reachable from (p,γ) in i steps. We prove (a) and (b) by induction on i.

(a) *Basis.* $i = 0$. In this case, $(p,\gamma) = (p',\gamma')$ and $b = 0$, thus $\langle p,\gamma\rangle \Rightarrow \langle p',\gamma'\rangle$.

Step. $i > 0$. Then there exist p'', γ'' such that $((p,\gamma), b, (p'',\gamma'')) \in E$ for some b, and $(p'',\gamma'') \xrightarrow[i]{b'} (p',\gamma')$. From the induction hypothesis, there exists u such that $\langle p'',\gamma''\rangle \Rightarrow \langle p',\gamma' u\rangle$. Since $((p,\gamma), b, (p'',\gamma'')) \in E$ we have $\langle p,\gamma\rangle \Rightarrow \langle p'',\gamma'' w\rangle$ (for some $w \in \Gamma^*$), hence $\langle p,\gamma\rangle \Rightarrow \langle p',\gamma' uw\rangle$.

(b) *Basis.* $i = 0$. In this case we always have $b = 0$, see (a).

Step. $i > 0$. In the proof of (a) either $b' = 1$, then (by the induction hypothesis) $\langle p'',\gamma''\rangle \overset{r}{\Longrightarrow} \langle p',\gamma' u\rangle$ and hence $\langle p,\gamma\rangle \overset{r}{\Longrightarrow} \langle p',\gamma' uw\rangle$. Otherwise $b = 1$. This implies that there exists a rule $\langle p,\gamma\rangle \hookrightarrow \langle p_1, v_1\gamma'' w\rangle$, and either $p \in G$ or $\langle p_1, v_1\rangle \overset{r}{\Longrightarrow} \langle p'',\varepsilon\rangle$ holds. In the former case, we have $\langle p,\gamma\rangle \overset{r}{\Longrightarrow} \langle p_1, v_1\gamma'' w\rangle$, and in the latter case we have $\langle p_1, v_1\gamma'' w\rangle \overset{r}{\Longrightarrow} \langle p'',\gamma'' w\rangle$. By the induction hypothesis $\langle p'',\gamma''\rangle \Rightarrow \langle p',\gamma' u\rangle$. In both cases, $\langle p,\gamma\rangle \overset{r}{\Longrightarrow} \langle p',\gamma' uw\rangle$ holds.

"$\Leftarrow$": Again, let $\langle p,\gamma\rangle \overset{i}{\Longrightarrow} \langle p',\gamma' v\rangle$ denote that we derive $\langle p',\gamma' v\rangle$ in i steps. We proceed by induction on i.

(a) *Basis.* $i = 0$. In this case we have $\langle p,\gamma\rangle = \langle p',\gamma'\rangle$ and $v = \varepsilon$, and indeed $(p,\gamma) \xrightarrow{0} (p,\gamma)$ holds by the definition of $\rightarrow$.

Step. $i > 0$. Then there exists some configuration $\langle p'', u\rangle$ such that $\langle p, y\rangle \overset{1}{\Longrightarrow} \langle p'', u\rangle \overset{i}{\Longrightarrow} \langle p',\gamma' v\rangle$. There must be a rule $\langle p,\gamma\rangle \hookrightarrow \langle p'', u\rangle$, and moreover $|u| \geq 1$. Let l denote the minimal length of the stack on the path from $\langle p'', u\rangle$ to $\langle p',\gamma' v\rangle$. Then u can be written as $u = u''\gamma_1 u'$ where $|u'| = l - 1$. Furthermore, there exists p''' such that $\langle p'', u''\rangle \Rightarrow \langle p''', \varepsilon\rangle$ and, since u' has to remain on the stack for the rest of the path, v is of the form $v = v'u'$ for some $v' \in \Gamma^*$. This means that $\langle p''', \gamma_1\rangle \overset{j}{\Longrightarrow} \langle p', \gamma' v'\rangle$ with $j < i$. By the induction hypothesis, $(p''', \gamma_1) \rightarrow (p', \gamma')$ holds. Also, $((p,\gamma), b, (p''', \gamma_1))$ is an edge in E, hence $(p,\gamma) \rightarrow (p',\gamma')$.

(b) *Basis.* $i = 0$. $\langle p, \gamma \rangle \stackrel{r}{\Longrightarrow} \langle p', \gamma' v \rangle$ is impossible in zero steps.

Step. $i > 0$. In the step of (a), either $\langle p, \gamma \rangle \stackrel{r}{\Longrightarrow} \langle p'', u \rangle$, i.e. $p \in G$. Then it holds that $((p, \gamma), 1, (p''', \gamma_1)) \in E$ and $(p''', \gamma_1) \to (p', \gamma')$, thus $(p, \gamma) \stackrel{1}{\longrightarrow} (p', \gamma')$. Otherwise, $\langle p'', u \rangle \stackrel{r}{\Longrightarrow} \langle p', \gamma' v \rangle$ holds. Either we have $\langle p'', u'' \rangle \stackrel{r}{\Longrightarrow} \langle p''', \varepsilon \rangle$, then $((p, \gamma), 1, (p''', \gamma_1))$ is in E. Otherwise $(p''', \gamma_1) \stackrel{r}{\Longrightarrow} \langle p', \gamma' v' \rangle$, then by the induction hypothesis we have $(p''', \gamma_1) \stackrel{1}{\longrightarrow} (p', \gamma')$. In both cases we get the desired result.

□

We can now prove the following theorem.

Theorem 5.1 *Let $\mathcal{BP}$ be a Büchi pushdown system, and let $\mathcal{G}$ be the associated head reachability graph. A head $\langle p, \gamma \rangle$ of $\mathcal{BP}$ is repeating if and only if the corresponding node (p, γ) is in a strongly connected component of $\mathcal{G}$ which has an internal* 1*-labelled edge (i.e. an edge connecting nodes within the same component).*

Proof: Let $\langle p, \gamma \rangle \in R$, i.e. there exists $v \in \Gamma^*$ such that $\langle p, \gamma \rangle \stackrel{r}{\Longrightarrow} \langle p, \gamma v \rangle$ holds. By Lemma 5.1, this is the case if and only if $(p, \gamma) \stackrel{1}{\longrightarrow} (p, \gamma)$. From the definition of $\to$ it follows that E contains an edge $((p', \gamma'), 1, (p'', \gamma''))$ such that $(p, \gamma) \to (p', \gamma') \stackrel{1}{\longrightarrow} (p'', \gamma'') \to (p, \gamma)$. Then (p, γ), (p', γ'), and (p'', γ'') are all in the same strongly connected component, and this component contains a 1-labelled edge. Conversely, whenever (p, γ) is in a component with such an edge, $(p, \gamma) \stackrel{1}{\longrightarrow} (p, \gamma)$ holds. □

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, but leads to a simpler solution. 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 $(\stackrel{\varepsilon}{\longrightarrow})^* \stackrel{\gamma}{\longrightarrow} (\stackrel{\varepsilon}{\longrightarrow})^*$ by $\stackrel{\gamma}{\Longrightarrow}$. $\mathcal{A}_{post^*}$ 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 $p \stackrel{\gamma}{\Longrightarrow} q$ in the current automaton, add a transition (p', ϵ, q).
If $\langle p, \gamma \rangle \hookrightarrow \langle p', \gamma' \rangle \in \Delta$ and $p \stackrel{\gamma}{\Longrightarrow} q$ 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 $p \stackrel{\gamma}{\Longrightarrow} q$ in the current automaton, add a transition (r, γ'', q).

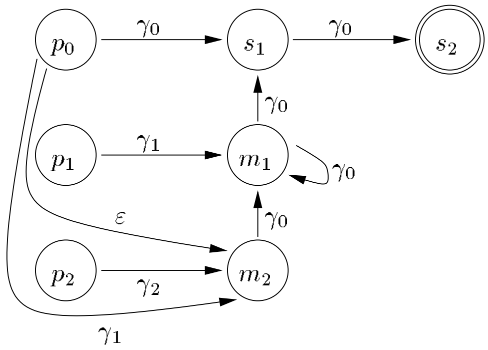


Figure 3: $\mathcal{A}_{post^*}$

Consider again the pushdown system $\mathcal{P}$ and the automaton $\mathcal{A}$ from Figure 1. Then the automaton shown in Figure 3 is the result of the new algorithm and accepts $post^*(\{\langle p_0, \gamma_0\gamma_0\rangle\})$.

Proof: Let $\mathcal{A}$ be a $\mathcal{P}$-automaton, and let $\mathcal{A}_{post^*}$ be the automaton obtained from $\mathcal{A}$ by means of the saturation rules. In the sequel we use the following notation:

- $q \xrightarrow[i]{w} q'$ denotes that the automaton obtained after the i-th application of the saturation rule contains a path labelled by w leading from q to q'; in particular, $q \xrightarrow[0]{w} q'$ denotes that $\mathcal{A}$ contains such a path (unless q' is a state added by the algorithm).
- $q \xrightarrow{w} q'$ denotes that there is an index i satisfying $q \xrightarrow[i]{w} q'$. Equivalently, it denotes that $\mathcal{A}_{post^*}$ contains a path labelled by w leading from q to q'.

We show that $\mathcal{A}_{post^*}$ recognises the set $post^*(\mathit{Conf}(\mathcal{A}))$. The result is proved in Theorem 6.1 at the end of this subsection. We need two preliminary lemmata.

Lemma 6.1 *For every configuration $\langle p, v\rangle \in \mathit{Conf}(\mathcal{A})$, if $\langle p, v\rangle \Rightarrow \langle p', w\rangle$ then $p' \xrightarrow{w} q$ for some final state q of $\mathcal{A}_{post^*}$.*

Proof: Let $\langle p, v\rangle \stackrel{k}{\Longrightarrow} \langle p', w\rangle$ denote that we derive $\langle p', w\rangle$ from $\langle p, v\rangle$ in k steps. We proceed by induction on k.

Basis. $k = 0$. Then $p' = p$ and $w = v$. Since $\langle p, v\rangle \in \mathit{Conf}(\mathcal{A})$, we have $p \xrightarrow[0]{v} q$ for some final state q, and so $p \xrightarrow{v} q$, which implies $p' \xrightarrow{w} q$.

Step. $k > 0$. Then, by the definition of $\stackrel{k}{\Longrightarrow}$, there is a configuration $\langle p'', u\rangle$ with

$$\langle p, v\rangle \stackrel{k-1}{\Longrightarrow} \langle p'', u\rangle \stackrel{1}{\Longrightarrow} \langle p', w\rangle .$$

We apply the induction hypothesis to $\langle p, v\rangle \stackrel{k-1}{\Longrightarrow} \langle p'', u\rangle$, and obtain

$$p'' \xrightarrow{u} q \text{ for some } q \in F .$$

Since $\langle p'', u\rangle \overset{1}{\Longrightarrow} \langle p', w\rangle$, there are γ, u_1, v_1, w_1 such that

$$u = \gamma u_1,\ w = w_1 u_1, \text{ and } \langle p'', \gamma\rangle \hookrightarrow \langle p', w_1\rangle.$$

There are three possible cases, according to the length of w_1. We consider only the case $|w_1| = 2$, the others being simpler. Since $|w_1| = 2$, we have $w_1 = \gamma'\gamma''$. Let q_1 be a state of $\mathcal{A}_{post^*}$ such that

$$p'' \xrightarrow{\gamma} q_1 \xrightarrow{u_1} q.$$

By the initialisation and the saturation rule, we have

$$p' \xrightarrow{\gamma'} r \xrightarrow{\gamma''} q_1 \xrightarrow{u_1} q\ ,$$

which implies

$$p' \xrightarrow{w_1 u_1} q\ ,$$

and since $w = w_1 u_1$, we are done. □

Lemma 6.2 *Let $p \xrightarrow{w} q$ be a path of $\mathcal{A}_{post^*}$. Then the following property holds: $\langle p', w'\rangle \Rightarrow \langle p, w\rangle$ for a configuration $\langle p', w'\rangle$ such that $p' \xrightarrow[0]{w'} q$.*

Proof: Let i be an index such that $p \xrightarrow[i]{w} q$. We prove the lemma by induction on i.

Basis. $i = 0$. Take $p' = p$ and $w' = w$.

Step. $i \geq 1$. Let t be the transition added to $\mathcal{A}$ in the i-th step. Let j be the number of times that t is used in $p \xrightarrow[i]{w} q$. $\mathcal{A}$ has no transitions leading to initial states, and the algorithm does not add any such transitions; therefore, if t starts in an initial state, t can be used at most once and only at the start of the path.

The proof is by induction on j. If $j = 0$, then we have $p \xrightarrow[i-1]{w} q$, and we apply the induction hypothesis (induction on i). So assume that $j > 0$. We distinguish the three possible cases of the saturation rule:

(i) and (ii): $t = (p_1, v, q_1)$, where $v = \varepsilon$ or $v = \gamma_1$: Then $j = 1$ and there exists w_1 such that $w = v w_1$ and q_1 such that

$$p = p_1 \xrightarrow[i]{v} q_1 \xrightarrow[i-1]{w_1} q \tag{0}$$

Since t was added via the saturation rule, there exist p_2 and γ_2 such that

$$\langle p_2, \gamma_2\rangle \hookrightarrow \langle p_1, v\rangle \tag{1.1}$$

$$p_2 \xrightarrow[i-1]{\gamma_2} q_1 \tag{1.2}$$

From (0) and (1.2) we get

$$p_2 \xrightarrow[i-1]{\gamma_2} q_1 \xrightarrow[i-1]{w_1} q \tag{2}$$

As t is not used in (2), applying the induction hypothesis (on j) we obtain

$$\langle p', w'\rangle \Longrightarrow \langle p_2, \gamma_2 w_1\rangle \tag{3.1}$$

$$p' \xrightarrow[0]{w'} q \tag{3.2}$$

Combining (1.1) and (3.1) we have

$$\langle p', w'\rangle \overset{(3.1)}{\Longrightarrow} \langle p_2, \gamma_2 w_1\rangle \overset{(1.1)}{\Longrightarrow} \langle p_1, v w_1\rangle = \langle p, w\rangle \tag{4}$$

The lemma follows from (3.2) and (4).

(iii) Let $t = (r, \gamma'', q')$ be a transition resulting from the application of the third part of the saturation rule. Then there exist u, v such that $w = u\gamma'' v$ and

$$p \xrightarrow[i-1]{u} r \xrightarrow[i]{\gamma''} q' \xrightarrow[i]{v} q \tag{5}$$

Because t was added via the saturation rule, we conclude that there exists some rule of the form

$$\langle p_2, \gamma_2\rangle \hookrightarrow \langle p_1, \gamma'\gamma''\rangle \tag{6.1}$$

$$\text{with } p_2 \xrightarrow[i-1]{\gamma_2} q' \tag{6.2}$$

i.e. (6.1) is the rule associated with r. Application of the induction hypothesis (on i) yields

$$\langle p_3, w_3\rangle \Rightarrow \langle p, u\rangle \tag{7.1}$$

$$p_3 \xrightarrow[0]{w_3} r \tag{7.2}$$

for some pair $\langle p_3, w_3\rangle$; due to the construction of the automaton it holds that $\langle p_3, w_3\rangle = \langle p_1, \gamma'\rangle$. Combining (5) and (6.2) we get

$$p_2 \xrightarrow[i-1]{\gamma_2} q' \xrightarrow[i]{v} q$$

Since t occurs less often than j in this path, we can apply the induction hypothesis (on j) to obtain the existence of some $\langle p', w'\rangle$ such that

$$\langle p', w'\rangle \Rightarrow \langle p_2, \gamma_2 v\rangle \tag{8.1}$$

$$p' \xrightarrow[0]{w'} q \tag{8.2}$$

Finally, if we put together (6.1), (7.1) and (8.1), we get

$$\langle p', w'\rangle \overset{(8.1)}{\Longrightarrow} \langle p_2, \gamma_2 v\rangle \overset{(6.1)}{\Longrightarrow} \langle p_1, \gamma'\gamma'' v\rangle = \langle p_3, w_3\gamma'' v\rangle \overset{(7.1)}{\Longrightarrow} \langle p, u\gamma'' v\rangle = \langle p, w\rangle \tag{9}$$

and (a) follows from (8.2) and (9). □

Theorem 6.1 *Let $\mathcal{A}_{post^*}$ be the automaton obtained from $\mathcal{A}$ by exhaustive application of the saturation rule defined in Section 6. $\mathcal{A}_{post^*}$ recognises the set $post^*(Conf(\mathcal{A}))$.*

Proof:

Let $\langle p, w\rangle$ be a configuration of $post^*(Conf(\mathcal{A}))$. Then $\langle p', w'\rangle \Rightarrow \langle p, w\rangle$ for a configuration $\langle p', w'\rangle \in Conf(\mathcal{A})$. By Lemma 6.1, $p \xrightarrow{w} q$ for some final state q of $\mathcal{A}_{post^*}$. So $\langle p, w\rangle$ is recognised by $\mathcal{A}_{post^*}$.

Conversely, let $\langle p, w\rangle$ be a configuration recognised by $\mathcal{A}_{post^*}$. Then $p \xrightarrow{w} q$ in $\mathcal{A}_{post^*}$ for some final state q. By Lemma 6.2, $\langle p', w'\rangle \Rightarrow \langle p, w\rangle$ for a configuration $\langle p', w'\rangle$ such that $p' \xrightarrow[0]{w'} q$. Since q is a final state, $\langle p', w'\rangle \in Conf(\mathcal{A})$, and so $\langle p, w\rangle \in post^*(Conf(\mathcal{A}))$. $\square$

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.

7.1 Computing $pre^*(C)$

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

Algorithm 1 computes the transitions of $\mathcal{A}_{pre^*}$ by 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 the

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 $\mathcal{A}_{pre^*}$

1 $\quad$ $rel \leftarrow \emptyset$; $trans \leftarrow \delta$; $\Delta' \leftarrow \emptyset$;
2 $\quad$ **for all** $\langle p, \gamma \rangle \hookrightarrow \langle p', \varepsilon \rangle \in \Delta$ **do** $trans \leftarrow trans \cup \{(p, \gamma, p')\}$;
3 $\quad$ **while** $trans \neq \emptyset$ **do**
4 $\quad\quad$ pop $t = (q, \gamma, q')$ from $trans$;
5 $\quad\quad$ **if** $t \notin rel$ **then**
6 $\quad\quad\quad$ $rel \leftarrow rel \cup \{t\}$;
7 $\quad\quad\quad$ **for all** $\langle p_1, \gamma_1 \rangle \hookrightarrow \langle q, \gamma \rangle \in (\Delta \cup \Delta')$ **do**
8 $\quad\quad\quad\quad$ $trans \leftarrow trans \cup \{(p_1, \gamma_1, q')\}$;
9 $\quad\quad\quad$ **for all** $\langle p_1, \gamma_1 \rangle \hookrightarrow \langle q, \gamma\gamma_2 \rangle \in \Delta$ **do**
10 $\quad\quad\quad\quad$ $\Delta' \leftarrow \Delta' \cup \{\langle p_1, \gamma_1 \rangle \hookrightarrow \langle q', \gamma_2 \rangle\}$;
11 $\quad\quad\quad\quad$ **for all** $(q', \gamma_2, q'') \in rel$ **do**
12 $\quad\quad\quad\quad\quad$ $trans \leftarrow trans \cup \{(p_1, \gamma_1, q'')\}$;
13 $\quad$ **return** rel

addition of (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.

Termination: rel is empty initially and can only grow afterwards. Q and Γ are finite sets, therefore rel can only take finitely many elements. Similarly, $trans$ can only be of finite size. Once no more transitions can be added to rel, $trans$ can no longer grow and will be empty eventually.

Because of the finiteness of rel and Δ there will be finitely many members of Δ', and so the loop at line 7 is traversed only finitely often.

The following lemma is needed in the subsequent proofs.

Lemma 7.1 *For every transition t, if $t \in trans$ at any time during the execution of the algorithm, then t will be "used" exactly once, i.e. the part between lines 6 and 12 will be entered exactly once for t.*

Proof: rel is empty initially and contains exactly the used transitions later. Until t is removed from $trans$, the algorithm cannot terminate. When t is removed from $trans$, it is used if and only if it is not yet a used transition, otherwise it will be ignored. $\square$

Correctness:

(1) Throughout the algorithm $rel \subseteq \delta_{pre^*}$ holds. rel contains only elements from $trans$, so we inspect the lines that change $trans$, and show that all the additions are in compliance with the algorithm given in section 4 the correctness of which has already been proved. More precisely, we show that all additions model that algorithm's saturation rule:

If $\langle p,\gamma\rangle \hookrightarrow \langle p',w\rangle$ and $p' \xrightarrow{w} q$, then add (p,γ,q).

- In line 1, *trans* is assigned δ which allows us to recognise C.
- Lines 2 and 12 directly model the saturation rule.
- In line 8, if the rule $\langle p_1,\gamma_1\rangle \hookrightarrow \langle q,\gamma\rangle$ is taken from Δ, then we directly model the saturation rule. Otherwise, $\langle p_1,\gamma_1\rangle \hookrightarrow \langle q,\gamma\rangle$ was added to Δ' because $\langle p_1,\gamma_1\rangle \hookrightarrow \langle p'',\gamma'\gamma\rangle \in \Delta$ and (p'',γ',q) in *trans* for some p'',γ' and again the saturation rule applies.

(2) After termination $\delta_{pre^*} \subseteq rel$ holds. Initially, *trans* contains δ. Because of Lemma 7.1 all of δ will eventually end up in *rel.* Moreover, we prove that all possible applications of the saturation rule are used. Assume that we have $\langle p,\gamma\rangle \hookrightarrow \langle p',w\rangle$ and $p' \xrightarrow{w} q$ w.r.t. *rel*.

- If $w=\varepsilon$, then $q=p'$, and (p,γ,p') is added in line 2.
- If $w=\gamma_1$, then there is some transition $t_1=(p',\gamma_1,q)$ in *rel.* Because *rel* contains only the used transitions, and because of line 8, (p,γ,q) will be added to *trans*.
- If $w=\gamma_1\gamma_2$, then *rel* contains $t_1=(p',y_1,q')$ and $t_2=(q',\gamma_2,q)$.
  - If t_1 is used before t_2, Δ' will have the rule $\langle p,\gamma\rangle \hookrightarrow \langle q',\gamma_2\rangle$. Then, when t_2 is used, (p,γ,q) is added in line 8.
  - If t_2 is used before t_1, it is in *rel* when t_1 is used. Then (p,γ,q) is added in line 12.

Complexity: Let $n_Q=|Q|$, $n_\delta=|\delta|$, and $n_\Delta=|\Delta|$, i.e. the number of states and transitions in $\mathcal{A}$, and the number of rules in the pushdown system. The size of $\mathcal{A}$ can be written as n_Q+n_δ.

Imagine that prior to running the algorithm, all rules of the form $\langle p,\gamma\rangle \hookrightarrow \langle p',\gamma' w\rangle$ have been put into "buckets" labelled (p',γ'). If the buckets are organised in a hash table this can be done in $O(n_\Delta)$ time and space. Similarly, all rules in Δ' can be put into such buckets at run-time, and the addition of one rule takes only constant time.

If *rel* and δ are implemented as hash tables, then addition and membership test take constant time. Moreover, if *trans* is a stack, then addition and removal of transitions (from the stack) take constant time, too. However, we will want to avoid adding any transition to *trans* more than once. This can be done (without asymptotical loss of time) by storing all transitions which are ever added to *trans* in an additional hash table. Transitions are added to the stack only if they are not already in the table.

When (q,γ,q') is used, we need to regard just the rules that are in the (q,γ)-bucket. Because of Lemma 7.1, every possible transition is used at most once. Based on these observations, let us compute how often the statements inside the main loop are executed.

Line 10 is executed once for each combination of rules $\langle p_1,\gamma_1\rangle \hookrightarrow \langle q,\gamma\gamma_2\rangle$ and transitions (q,γ,q'), i.e. $O(n_Q n_\Delta)$ times, therefore the size of Δ' is $O(n_Q n_\Delta)$, too. For the loop starting at line 11, q' and γ_2 are fixed, so line 12 is executed $O(n_Q^2 n_\Delta)$ times.

Line 8 is executed once for each combination of rules $\langle p_1, \gamma_1 \rangle \hookrightarrow \langle q, \gamma \rangle$ in $(\Delta \cup \Delta')$ and transitions (q, γ, q'). As stated previously, the size of Δ' is $O(n_Q n_\Delta)$, so line 8 is executed $O(n_Q^2 n_\Delta)$ times.

Let us now count the iterations of the main loop, i.e. how often line 4 is executed. This directly depends on the number of elements that are added to *trans*. Initially, there are $n_\delta + O(n_\Delta)$ elements from lines 1 and 2. Notice that $n_\delta = O(n_Q \cdot n_\Delta \cdot n_Q)$. We already know that the other additions to *trans* are no more than $O(n_Q^2 n_\Delta)$ in number. As a conclusion, the whole algorithm takes $O(n_Q^2 n_\Delta)$ time. Memory is needed for storing *rel*, *trans* and Δ'. The memory used for *trans* is never larger than the final size of *rel*.

- Line 1 adds n_δ transitions to *trans*.
- In line 2, there are $O(n_\Delta)$ additions.
- In lines 8 and 12, p_1 and γ_1 are taken from the head of a rule in Δ. This means that these lines can only add $O(n_\Delta n_Q)$ different transitions to *rel*.
- The size of Δ' is $O(n_Q n_\Delta)$ (see above).

From these facts it follows that the algorithm takes $O(n_Q n_\Delta + n_\delta)$ space, the size needed to store the result. Algorithm 1 is therefore optimal with respect to memory usage.

Theorem 7.1 *Let $\mathcal{P} = (P, \Gamma, \Delta)$ be a pushdown system and let $\mathcal{A} = (\Gamma, Q, \delta, P, F)$ be an automaton. There exists an automaton $\mathcal{A}_{pre^*}$ recognising $pre^*(Conf(\mathcal{A}))$. 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 $\langle p, \gamma \rangle \overset{r}{\Longrightarrow} \langle p', \varepsilon \rangle$.

The second phase of the algorithm constructs the graph $\mathcal{G}$ using the results of the first phase. Finally, Tarjan's algorithm [9] can be used to find the strongly connected components of $\mathcal{G}$ and thus to determine the repeating heads.

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

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

1 $\mathit{rel} \leftarrow \emptyset$; $\mathit{trans} \leftarrow \emptyset$; $\Delta' \leftarrow \emptyset$;
2 **for all** $\langle p, \gamma\rangle \hookrightarrow \langle p', \varepsilon\rangle \in \Delta$ **do**
3   $\mathit{trans} \leftarrow \mathit{trans} \cup \{(p, [\gamma, G(p)], p')\}$;
4 **while** $\mathit{trans} \neq \emptyset$ **do**
5   pop $t = (p, [\gamma, b], p')$ from trans;
6   **if** $t \notin \mathit{rel}$ **then**
7     $\mathit{rel} \leftarrow \mathit{rel} \cup \{t\}$;
8   **for all** $\langle p_1, \gamma_1\rangle \hookrightarrow \langle p, \gamma\rangle \in \Delta$ **do**
9       $\mathit{trans} \leftarrow \mathit{trans} \cup \{(p_1, [\gamma_1, b \vee G(p_1)], p')\}$;
10   **for all** $\langle p_1, \gamma_1\rangle \xhookrightarrow{b'} \langle p, \gamma\rangle \in \Delta'$ **do**
11       $\mathit{trans} \leftarrow \mathit{trans} \cup \{(p_1, [\gamma_1, b \vee b'], p')\}$;
12   **for all** $\langle p_1, \gamma_1\rangle \hookrightarrow \langle p, \gamma\gamma_2\rangle \in \Delta$ **do**
13       $\Delta' \leftarrow \Delta' \cup \{\langle p_1, \gamma_1\rangle \xhookrightarrow{b \vee G(p_1)} \langle p', \gamma_2\rangle\}$;
14   **for all** $(p', [\gamma_2, b'], p'') \in \mathit{rel}$ **do**
15         $\mathit{trans} \leftarrow \mathit{trans} \cup \{(p_1, [\gamma_1, b \vee b' \vee G(p_1)], p'')\}$;
16
17 $R \leftarrow \emptyset$; $E \leftarrow \emptyset$;
18 **for all** $\langle p, \gamma\rangle \hookrightarrow \langle p', \gamma'\rangle \in \Delta$ **do** $E \leftarrow E \cup \{((p, \gamma), G(p), (p', \gamma'))\}$;
19 **for all** $\langle p, \gamma\rangle \xhookrightarrow{b} \langle p', \gamma'\rangle \in \Delta'$ **do** $E \leftarrow E \cup \{((p, \gamma), b, (p', \gamma'))\}$;
20 **for all** $\langle p, \gamma\rangle \hookrightarrow \langle p', \gamma'\gamma''\rangle \in \Delta$ **do** $E \leftarrow E \cup \{((p, \gamma), G(p), (p', \gamma'))\}$;
21 find strongly connected components in $\mathcal{G} = ((P \times \Gamma), E)$;
22 **for all** components C **do**
23   **if** C has a 1-edge **then** $R \leftarrow R \cup C$;
24 **return** R

Termination: Termination follows from the fact that the pre^* algorithm terminates (we just have a larger alphabet), and from the fact that the components can be determined in finite time.

Correctness: The first phase of the algorithm computes $\mathit{pre}^*(\{\, (p, \varepsilon) \mid p \in P \,\})$. The only modification to the algorithm for pre^* is the addition of the boolean indicators. We have already proved the correctness of the normal pre^*-algorithm. Applying Lemmas 4.1 and 4.2 we get that $\langle p, \gamma\rangle \Rightarrow \langle p', \varepsilon\rangle$ if and only if the algorithm yields a transition $(p, [\gamma, b], p')$ for some b. It remains to show that the algorithm obtains $(p, [\gamma, 1], p')$ if and only if $\langle p, \gamma\rangle \stackrel{r}{\Longrightarrow} \langle p', \varepsilon\rangle$.

"$\Rightarrow$": We consider the circumstances under which 1-labelled transitions can be added to *trans*.

- Originally, the automaton contains no transitions at all.
- In line 3, we have $\langle p, \gamma\rangle \hookrightarrow \langle p', \varepsilon\rangle$ and $p \in G$. Hence, $\langle p, \gamma\rangle \stackrel{r}{\Longrightarrow} \langle p', \varepsilon\rangle$.

- In line 9, we have $\langle p_1,\gamma_1\rangle \hookrightarrow \langle p,\gamma\rangle$ and $(p,[\gamma,b],p')$. To make the expression $b \vee G(p_1)$ true, either $b=1$ (which means $\langle p,\gamma\rangle \overset{r}{\Longrightarrow} \langle p',e\rangle$) or $p_1 \in G$ (which means $\langle p_1,\gamma_1\rangle \overset{r}{\Longrightarrow} \langle p,\gamma\rangle$. In both cases we get $\langle p_1,\gamma_1\rangle \overset{r}{\Longrightarrow} p',\varepsilon$.

- In line 11, there is a transition $(p,[\gamma,b],p')$, and the existence of the rule means $\langle p_1,\gamma_1\rangle \hookrightarrow \langle p_2,\gamma_2\gamma\rangle$, $(p_2,[\gamma_2,b''],p)$, and $b' = G(p_1) \vee b''$ for some p_2,γ_2. In other words, $\langle p_1,\gamma_1\rangle \Rightarrow \langle p_2,\gamma_2\gamma\rangle \Rightarrow \langle p,\gamma\rangle \Rightarrow \langle p',\varepsilon\rangle$. If $b \vee b'$ is true, either $b=1$ (meaning $\langle p,\gamma\rangle \overset{r}{\Longrightarrow} \langle p',e\rangle$), or $p_1 \in G$ (meaning $\langle p_1,\gamma_1\rangle \overset{r}{\Longrightarrow} p_2,\gamma_2\gamma$), or $b''=1$ (meaning $\langle p_2,\gamma_2\gamma\rangle \overset{r}{\Longrightarrow} \langle p,\gamma\rangle$).

- In line 15, the situation is similar to the preceding case, only the roles of p,γ and p_2,γ_2 are reversed.

"$\Leftarrow$": Let $\langle p,\gamma\rangle \xrightarrow[i]{r} \langle p',\varepsilon\rangle$ denote that the derivation of the right-hand side occurs in i steps. We prove the property by induction on i.

Basis. $i=0$. By definition, $\langle p,\gamma\rangle \xrightarrow[0]{r} \langle p',\varepsilon\rangle$ is not possible.

Step. $i>0$. Then we can find a configuration $\langle p'',u\rangle$ such that $\langle p,\gamma\rangle \xrightarrow[1]{} \langle p'',w\rangle \xrightarrow[i]{} \langle p',\varepsilon\rangle$ holds. Then $p'' \xrightarrow{w} p'$ will be a path of A_{pre^*}.

Either $\langle p,\gamma\rangle \xrightarrow[1]{r} \langle p'',w\rangle$ holds, i.e. $p \in G$. The addition rule of the pre^*-algorithm leads to $(p,[\gamma,b],p')$ where $b=1$ because $G(p)=1$.

Otherwise $\langle p'',w\rangle \xrightarrow[i]{r} \langle p',\varepsilon\rangle$ holds. Then $|w| \geq 1$ since at least one step must occur. In the case $|w|=1$ assume $w=\gamma_1$. By the induction rule, $(p'',[\gamma_1,1],p')$ is in *rel*. Then $(p,[\gamma,1],p')$ will be added in line 9.

In the case $|w|=2$ assume $w=\gamma_1\gamma_2$. There exists p''' such that $\langle p'',\gamma_1\rangle \Rightarrow \langle p''',\varepsilon\rangle$ and $\langle p''',\gamma_2\rangle \Rightarrow \langle p',\varepsilon\rangle$. The length of both derivations will be at most i, and we conclude that the algorithm must find transitions of the form $(p'',[\gamma_1,b_1],p''')$ and $(p''',[\gamma_2,b_2],p')$. Either $\langle p'',\gamma_1\rangle \overset{r}{\Longrightarrow} \langle p''',\varepsilon\rangle$, then by the induction hypothesis $b_1=1$. Otherwise $\langle p''',\gamma_2\rangle \overset{r}{\Longrightarrow} \langle p',\varepsilon\rangle$ and $b_2=1$. Depending on which transition is processed first by the algorithm, $(p,[\gamma,1],p')$ is added in line 11 or 15.

It is easy to see that the second phase constructs the graph $\mathcal{G}$ according to the rule from section 5 for the special case that the right-hand side of a rule is no longer than two. Notice that $\langle p,\gamma\rangle \hookrightarrow \langle p',\gamma'\rangle \in \Delta'$ exactly if there is a rule $\langle p,\gamma\rangle \hookrightarrow \langle p'',\gamma''\gamma'\rangle \in \Delta$ and $(p'',[\gamma'',b],p') \in rel$ for some b.

Complexity: Let $n_P = |P|$ and $n_\Delta = |\Delta|$. The first phase is essentially the same as Algorithm 1 with an initially empty automaton; the addition of boolean indicators on the arcs can only lead to a linear increase in both time and space requirements. Therefore, Theorem 7.1 tells us that the first phase takes $O(n_P^2 n_\Delta)$ time and $O(n_P n_\Delta)$ space.

In the second phase, the rules of Δ contribute $O(n_\Delta)$ nodes and edges to the size of $\mathcal{G}$. As the size of Δ' is $O(n_P n_\Delta)$, the total size of $\mathcal{G}$ is $O(n_P n_\Delta)$, too. Determining the strongly connected components is possible in linear time in the size of the graph, and the same holds for searching each component for an internal 1-edge.

Theorem 7.2 *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.

Consider again the example in Figure 3. 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 4 shows the result, similar to Figure 3 but with the ε-transition resolved.

For better understanding of the following paragraphs it is useful to consider the structure of the transitions in $\mathcal{A}_{post^*}$. Let $Q_1 = (Q \setminus P)$ and $Q_2 = (Q' \setminus Q)$.

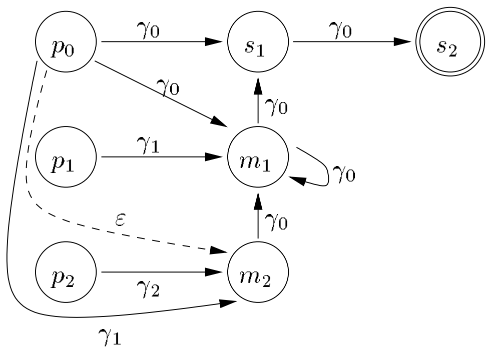


Figure 4: $\mathcal{A}_{post^*}$ as computed by Algorithm 3.

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 $\mathcal{A}_{post^*}$

1 $\mathit{trans} \leftarrow \delta \cap (P \times \Gamma \times Q)$;
2 $\mathit{rel} \leftarrow \delta \setminus \mathit{trans}$; $Q' \leftarrow Q$; $F' \leftarrow F$;
3 **for all** $r = \langle p, \gamma \rangle \hookrightarrow \langle p', \gamma_1\gamma_2 \rangle \in \Delta$ **do**
4 $\quad Q' \leftarrow Q' \cup \{q_r\}$;
5 $\quad \mathit{trans} \leftarrow \mathit{trans} \cup \{(p', \gamma_1, q_r)\}$;
6 **for all** $q \in Q'$ **do** $\mathit{eps}(q) \leftarrow \emptyset$;
7 **while** $\mathit{trans} \neq \emptyset$ **do**
8 $\quad$ pop $t = (p, \gamma, q)$ from trans;
9 $\quad$ **if** $t \notin \mathit{rel}$ **then**
10 $\quad\quad \mathit{rel} \leftarrow \mathit{rel} \cup \{t\}$;
11 $\quad\quad$ **for all** $\langle p, \gamma \rangle \hookrightarrow \langle p', \varepsilon \rangle \in \Delta$ **do**
12 $\quad\quad\quad$ **if** $p' \notin \mathit{eps}(q)$ **then**
13 $\quad\quad\quad\quad \mathit{eps}(q) \leftarrow \mathit{eps}(q) \cup \{p'\}$;
14 $\quad\quad\quad\quad$ **for all** $(q, \gamma', q') \in \mathit{rel}$ **do**
15 $\quad\quad\quad\quad\quad \mathit{trans} \leftarrow \mathit{trans} \cup \{(p', \gamma', q')\}$;
16 $\quad\quad\quad\quad$ **if** $q \in F'$ **then** $F' \leftarrow F' \cup \{p'\}$;
17 $\quad\quad$ **for all** $\langle p, \gamma \rangle \hookrightarrow \langle p', \gamma_1 \rangle \in \Delta$ **do**
18 $\quad\quad\quad \mathit{trans} \leftarrow \mathit{trans} \cup \{(p', \gamma_1, q')\}$;
19 $\quad\quad$ **for all** $r = \langle p, \gamma \rangle \hookrightarrow \langle p', \gamma_1\gamma_2 \rangle \in \Delta$ **do**
20 $\quad\quad\quad \mathit{rel} \leftarrow \mathit{rel} \cup \{(q_r, \gamma_2, q)\}$;
21 $\quad\quad\quad$ **for all** $p'' \in \mathit{eps}(q_r)$ **do**
22 $\quad\quad\quad\quad \mathit{trans} \leftarrow \mathit{trans} \cup \{(p'', \gamma_2, q)\}$;
23 **return** $(\Gamma, Q', \mathit{rel}, P, F')$

In the beginning, there are no transitions into P, i.e. we just have transitions from P into Q_1, and from Q_1 into Q_1. After line 5 is executed once, we also have transitions from P to Q_2. All the other additions to *rel* are now either from P to $Q_1 \cup Q_2$ except for line 20; here we have transitions from Q_2 to $Q_1 \cup Q_2$. We can summarise these observations in the following facts:

- After execution of the algorithm, *rel* contains no transitions leading into P.
- The algorithm does not add any transitions starting in Q_1.

Termination: The algorithm terminates. This can be seen from the fact that the size of Q' is bounded by $|Q| + |\Delta|$; hence, *rel*'s maximum size is $|Q'| \cdot |\Gamma| \cdot |Q'|$, and we can use a similar reasoning as in the algorithm for pre^*.

Correctness: We show that the algorithm is an implementation of the construction given earlier. In section 6 we defined δ_{post^*} to be the smallest set containing δ, containing a transition (p_1, γ_1, q_r) for every rule $r = \langle p, \gamma \rangle \hookrightarrow \langle p_1, \gamma_1\gamma_2 \rangle$ and satisfying the following:

- If $\langle p,\gamma\rangle \hookrightarrow \langle p',\epsilon\rangle \in \Delta$ and $p \stackrel{\gamma}{\Longrightarrow} q$, then $(p',\varepsilon,q) \in \delta_{post^*}$.
- If $\langle p,\gamma\rangle \hookrightarrow \langle p',\gamma'\rangle \in \Delta$ and $p \stackrel{\gamma}{\Longrightarrow} q$, then $(p',\gamma',q) \in \delta_{post^*}$.
- If $r = \langle p,\gamma\rangle \hookrightarrow \langle p',\gamma'\gamma''\rangle \in \Delta$ and $p \stackrel{\gamma}{\Longrightarrow} q$, then $(r,\gamma'',q) \in \delta_{post^*}$.

The automaton constructed in Algorithm 3 does not have ε-transitions, so we cannot show that $rel = \delta_{post^*}$. Instead, we show that after execution $((q,\gamma,q') \in rel)$ exactly if $(q \stackrel{\gamma}{\Longrightarrow} q')$ for all $q,q' \in Q$, $\gamma \in \Gamma$.

"$\Rightarrow$" Since elements from *trans* flow into *rel*, we inspect all the lines that change *trans* or *rel*:

- Lines 1 and 2 add elements from δ which is a subset of δ_{pre^*}.
- Line 5 is a consequence of the initialisation rule.
- In line 15 we have (p,γ,q) and (q,γ',q') in *rel*, $\langle p,\gamma\rangle \hookrightarrow \langle p',\varepsilon\rangle$ in Δ, hence $p' \xrightarrow{\varepsilon} q \xrightarrow{\gamma'} q'$, so (p,γ',q') does not change the desired property.
- Line 18 is a direct implementation of the second saturation property.
- Line 20 is a direct implementation of the third saturation property.
- In line 22 it holds that $p'' \in eps(q_r)$; from this we conclude the existence of some rule $\langle p_1,\gamma_1\rangle \hookrightarrow \langle p'',\varepsilon\rangle$ and some transition (p_1,γ_1,q_r). So, $p'' \xrightarrow{\varepsilon} q_r \xrightarrow{\gamma_2} q$, and the addition of (p'',γ_2,q) doesn't change the property.

"$\Leftarrow$" By the same argumentation as in Lemma 7.1 we can say that all the elements in *trans* eventually end up in *rel*. Therefore it is sufficient to prove that any element of δ_{post^*} is added to either *rel* or *trans* during execution of the algorithm.

We observe the following: Since there are no transitions leading into P, the ε-transitions can only go from states in P to states in $Q_1 \cup Q_2$; therefore no two ε-transitions can be adjacent. The relation $p \stackrel{\gamma}{\Longrightarrow} q$ from section 6 can be written as follows:

$$(p \stackrel{\gamma}{\Longrightarrow} q) \iff (p,\gamma,q) \vee \exists q' \colon ((p,\varepsilon,q') \wedge (q',\gamma,q))$$

The desired property follows from the following facts:

- Because of lines 1 and 2, after execution $\delta \subseteq rel$ holds.
- If $\langle p',\gamma'\rangle \hookrightarrow \langle p,\varepsilon\rangle$ and $(p',\gamma',q) \in rel$, then p is added to $eps(q)$. We will see that whenever there is p in $eps(q')$ and (q',γ,q) in *rel* for some $p,q',q \in Q'$ and $y \in \Gamma$, eventually (p,γ,q) will be in *rel*.
  * Either (q',γ,q) is known before $p \in eps(q')$ is known. Then (p,γ,q) is added in line 15;
  * Or $p \in eps(q')$ is known before (q',γ,q). Recall that $q' \in Q_1 \cup Q_2$. ε-transitions are added only after the initialisation phase, and the only transitions starting in $Q_1 \cup Q_2$ added after initialisation are those in line 20. In this case (p,γ,q) is added in line 22.
- If $\langle p',\gamma'\rangle \hookrightarrow \langle p,\gamma\rangle$ and $(p',\gamma',q) \in rel$, then (p,γ,q) is added in line 18.
- If $r = \langle p',\gamma'\rangle \hookrightarrow \langle p,\gamma\gamma''\rangle$ then (p,γ,r) is added in line 5. If moreover $(p',\gamma',q) \in rel$, (r,γ'',q) is added in line 20.

Complexity: Let $n_P, n_Q, n_\Delta, n_\delta$ be the sizes of P, Q, Δ and δ, respectively. Once again let *rel* and δ be implemented as a hash table and *trans* as a stack (without duplicates), so that all the needed addition, membership test and removal operations take constant time. The sets *eps* can be implemented as bit-arrays with one entry for each state in P; again addition and membership test cost only constant time.

The rules in Δ can be sorted into buckets according to their left-hand side (at the cost of $O(\Delta)$ time and space); i.e. put every rule $\langle p, \gamma\rangle \hookrightarrow \langle p', w\rangle$ into the bucket labelled (p, γ). The transitions in *rel* can be sorted into buckets according to the source state (i.e. a transition (q, γ, q') would be sorted into a bucket labelled q); since no transition is added to *rel* more than once, this costs no more than $O(|rel|)$ time and space.

For every transition $t = (p, \gamma, q) \in rel$, the part from line 10 and 22 is executed only once. Because we just need to take a look at the (p, γ)-bucket for rules, we can make the following statements:

- Line 5 is executed $O(n_\Delta)$ times.
- Line 18 is executed once for every combination of rules $\langle p, \gamma\rangle \hookrightarrow \langle p', \gamma'\rangle$ and transitions (p, γ, q) of which there are at most $|\Delta|\,|Q'|$ many, i.e. there are $O(n_\Delta(n_Q + n_\Delta))$ many executions.
- Line 20 is executed $O(n_\Delta(n_Q + n_\Delta))$ times (like line 18).
- Since $eps(q)$, $\ q \in Q'$ can contain at most n_P entries, the statement in line 22 is executed $O(n_P n_\Delta(n_Q + n_\Delta))$ times.
- For line 15, analysis becomes more complicated. First, observe that the part from line 13 to 16 is executed only once for every $(p, \varepsilon, q) \in \delta_{post^*}$. Let us distinguish two cases:
  - $q \in Q_1$: Altogether, there are $O(n_\delta)$ many transitions going out from the states in Q_1, and each of them can be copied at most once to every state in P which means $O(n_P n_\delta)$ many operations (remember that the algorithm adds no transitions leaving states in Q_1!)
  - $q \in Q_2$: If $r = \langle p_1, \gamma_1\rangle \hookrightarrow \langle p_2, \gamma_2\gamma_3\rangle$, then all the transitions leaving q_r are of the form (q_r, γ_3, q'') where q'' may be in $Q_1 \cup Q_2$. There are $O(n_\Delta)$ many states in Q_2, so we end up with $O(n_P n_\Delta(n_Q + n_\Delta))$ many operations.
- Line 8 is executed at most once for every transition in δ and for every transition added in the lines discussed above, i.e. $O(n_P n_\Delta(n_Q + n_\Delta) + n_P n_\delta)$ times. This is also an upper bound for the size of *rel* resp. *trans*.
- The initialisation phase can be completed in $O(n_\delta + n_\Delta + n_Q)$ (for the transitions in δ we just need to decide whether the source state is in P and add the transition to either *trans* or *rel*).
- We need $O(n_Q + n_\Delta)$ space to store Q', $O(n_Q)$ for F' and $O(n_P(n_Q + n_\Delta))$ for *eps*.

Theorem 7.3 *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 $n_\delta = |\delta|$.*

In [6] 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 $post^*(\mathrm{C})$ 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 [6]. 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 the negation of the given formula φ.
- Compute the Büchi pushdown system $\mathcal{BP}$ as the product of $\mathcal{B}$ and the pushdown system $\mathcal{P} = (P, \Gamma, \Delta)$ under consideration. Let $\langle p_0, w_0\rangle$ be the initial configuration of P, and let $n_P = |P|$, $n_\Delta = |\Delta|$, $n_Q = |Q|$, and $n_\delta = |\delta|$. Moreover, let $g_{\mathcal{P}}$ and $g_{\mathcal{B}}$ denote the sizes of $\mathcal{P}$ and $\mathcal{B}$, respectively, i.e. $g_{\mathcal{P}} = n_P + n_\Delta + |w_0|$ and $g_{\mathcal{B}} = n_Q + n_\delta$. Then $\mathcal{BP}$ has at most $s = n_P n_Q$ states and $r = n_\Delta n_\delta$ rules and can be computed in $O(r)$ time.
- Compute the set of repeating heads R of $\mathcal{BP}$. According to Theorem 7.2, this takes $O(rs^2)$ time and $O(rs)$ space.
- Construct an automaton $\mathcal{A}$ accepting $R\,\Gamma^*$. $\mathcal{A} = ((P \times Q) \cup \{q\}, \Gamma, \delta_{\mathcal{A}}, P \times Q, \{q\})$, where $\delta_{\mathcal{A}}$ contains transitions (p, γ, q) for every $\langle p, \gamma\rangle \in R$ and (q, γ, q) for every $\gamma \in \Gamma$. In other words, $\mathcal{A}$ has $O(s)$ states and $O(r)$ transitions.
- Compute $pre^*(R\,\Gamma^*)$; according to Theorem 7.1 this takes $O(rs^2)$ time and $O(rs + r)$ space. A configuration $\langle p, w\rangle$ violates φ exactly if $\mathcal{A}_{pre^*}$ accepts $\langle (p, q_0), w\rangle$.

The complexity of this procedure is dominated by the time needed to compute the repeating heads and the space needed to store $\mathcal{A}_{pre^*}$. Since both both r and s depend on the size of $\mathcal{P}$ and $\mathcal{B}$ in linear fashion we have the following result:

Theorem 8.1 *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 [6] 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 global model-checking problem for reachable configurations we need some additional steps:

- In $\mathcal{A}_{pre^*}$, rename the states (p, q_0) into p for all $p \in P$ and take P as the new set of initial configurations.
- Compute $post^*$ for the initial configuration of $\mathcal{P}$. An automaton accepting $\{\langle p_0, w_0\rangle\}$ has $n_P + |w_0|$ states and $|w_0|$ transitions. Then, according to Theorem 7.3, $\mathcal{A}_{post^*}$ can be computed in $O(n_P n_\Delta (n_P + |w_0| + n_\Delta))$ time and space. $\mathcal{A}_{post^*}$ has $O(n_P + |w_0| + n_\Delta) = O(g_\mathcal{P})$ many states and $O(n_P n_\Delta g_\mathcal{P})$ many transitions.
- Compute the intersection of $\mathcal{A}_{pre^*}$ and $\mathcal{A}_{post^*}$. The resulting automaton $\mathcal{A}_{inter}$ accepts the set of reachable configurations violating φ. The transitions of $\mathcal{A}_{inter}$ are computed as follows:

  If (q, γ, q') in $\mathcal{A}_{pre^*}$ and (r, γ, r') in $\mathcal{A}_{post^*}$, then $((q, r), \gamma, (q', r'))$ in $\mathcal{A}_{inter}$.

  This intersection can be computed more efficiently if the following trick is employed: First all the transitions (r, γ, r') in $\mathcal{A}_{post^*}$ are sorted into buckets labelled γ. Then every transition of $\mathcal{A}_{pre^*}$ is multiplied with the transitions in the respective bucket. Since $\mathcal{A}_{post^*}$ has $O(g_\mathcal{P})$ states, no bucket can have more than $O({g_\mathcal{P}}^2)$ many entries. Hence, the intersection can be computed in $O({g_\mathcal{P}}^4 {g_\mathcal{B}}^2)$ time and space. Since this factor dominates the additional computations, we have:

Theorem 8.2 *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 (more precisely, the number of states in $\mathcal{B}$), i.e. $s = O(n_Q)$. As a consequence, we get the following results:

- The complexity of the repeating heads algorithm reduces to $O(g_{\mathcal{P}} {g_{\mathcal{B}}}^3)$ time and $O(g_{\mathcal{P}} {g_{\mathcal{B}}}^2)$ space.
- The complexity of the pre^* algorithm becomes $O(g_{\mathcal{P}} {g_{\mathcal{B}}}^3)$ time and $O(g_{\mathcal{P}} {g_{\mathcal{B}}}^2)$ space. $\mathcal{A}_{pre^*}$ has $O(g_{\mathcal{B}})$ states and $O(g_{\mathcal{P}} {g_{\mathcal{B}}}^2)$ transitions.
- $\mathcal{A}_{post^*}$ can be computed in $O({g_{\mathcal{P}}}^2)$ time and space; $\mathcal{A}_{post^*}$ has $O(g_{\mathcal{P}})$ states and $O({g_{\mathcal{P}}}^2)$ transitions.
- Because $\mathcal{A}_{pre^*}$ has only $O(g_{\mathcal{B}})$ states, no transition in $\mathcal{A}_{post^*}$ needs to be multiplied with more than $O({g_{\mathcal{B}}}^2)$ transitions from $\mathcal{A}_0$. Hence, the intersection automaton can be computed in $O({g_{\mathcal{P}}}^2 {g_{\mathcal{B}}}^2)$ time and space.

Theorem 9.1 *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 5. 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.

The left side of Figure 6 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 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.

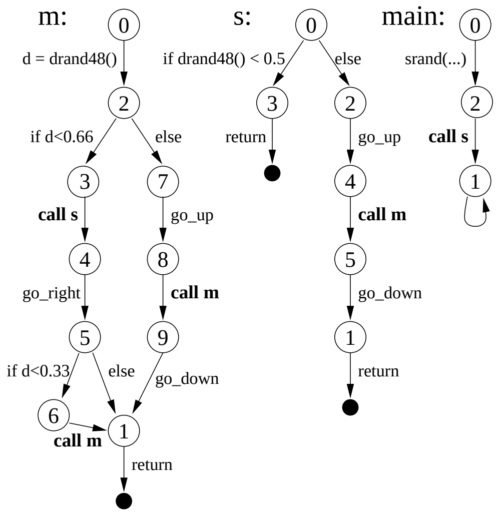
Obviously, these experiments can only give a rough impression of what the execution times would be like when analysing real programs. However, these experiments already constitute ‘stress tests’, i.e. their construction is based on certain exaggerated assumptions which affect execution times negatively. 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.

```
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();
}
```

Figure 5: An example program.



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

Figure 6: Flowgraph of the program in Figure 5 (left) and associated PDS (right)

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 [6], 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, 10], 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 [10] 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

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

	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

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 [7] 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.

[4] J. Esparza and J. Knoop. An automata-theoretic approach to interprocedural dataflow analysis. In *Proceedings of TACAS '99*, LNCS 1578, pages 14 – 30, 1999.

[5] J. Esparza and A. Podelski. Efficient algorithms for pre* and post* on interprocedural parallel flow graphs. In *Proceedings of POPL '00*, 2000.

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

[7] T. Jensen, D. Le Métayer, and T. Thorn. Verification of control flow based security properties. Technical Report 1210, IRISA, 1998.

[8] 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.

[9] R. E. Tarjan. Depth first search and linear graph algorithms. In *SICOMP 1*, pages 146–160, 1972.

[10] I. Walukiewicz. Pushdown Processes: Games and Model Checking. In *CAV'96*. LNCS 1102, 1996.

SFB 342: Methoden und Werkzeuge für die Nutzung paralleler Rechnerarchitekturen

bisher erschienen :

Reihe A

Liste aller erschienenen Berichte von 1990-1994
auf besondere Anforderung

342/01/95 A	Hans-Joachim Bungartz: Higher Order Finite Elements on Sparse Grids
342/02/95 A	Tao Zhang, Seonglim Kang, Lester R. Lipsky: The Performance of Parallel Computers: Order Statistics and Amdahl's Law
342/03/95 A	Lester R. Lipsky, Appie van de Liefvoort: Transformation of the Kronecker Product of Identical Servers to a Reduced Product Space
342/04/95 A	Pierre Fiorini, Lester R. Lipsky, Wen-Jung Hsin, Appie van de Liefvoort: Auto-Correlation of Lag-k For Customers Departing From Semi-Markov Processes
342/05/95 A	Sascha Hilgenfeldt, Robert Balder, Christoph Zenger: Sparse Grids: Applications to Multi-dimensional Schrödinger Problems
342/06/95 A	Maximilian Fuchs: Formal Design of a Model-N Counter
342/07/95 A	Hans-Joachim Bungartz, Stefan Schulte: Coupled Problems in Microsystem Technology
342/08/95 A	Alexander Pfaffinger: Parallel Communication on Workstation Networks with Complex Topologies
342/09/95 A	Ketil Stølen: Assumption/Commitment Rules for Data-flow Networks - with an Emphasis on Completeness
342/10/95 A	Ketil Stølen, Max Fuchs: A Formal Method for Hardware/Software Co-Design
342/11/95 A	Thomas Schnekenburger: The ALDY Load Distribution System
342/12/95 A	Javier Esparza, Stefan Römer, Walter Vogler: An Improvement of McMillan's Unfolding Algorithm
342/13/95 A	Stephan Melzer, Javier Esparza: Checking System Properties via Integer Programming
342/14/95 A	Radu Grosu, Ketil Stølen: A Denotational Model for Mobile Point-to-Point Dataflow Networks
342/15/95 A	Andrei Kovalyov, Javier Esparza: A Polynomial Algorithm to Compute the Concurrency Relation of Free-Choice Signal Transition Graphs
342/16/95 A	Bernhard Schätz, Katharina Spies: Formale Syntax zur logischen Kernsprache der Focus-Entwicklungsmethodik
342/17/95 A	Georg Stellner: Using CoCheck on a Network of Workstations
342/18/95 A	Arndt Bode, Thomas Ludwig, Vaidy Sunderam, Roland Wismüller: Workshop on PVM, MPI, Tools and Applications

Reihe A

342/19/95 A	Thomas Schnekenburger: Integration of Load Distribution into ParMod-C
342/20/95 A	Ketil Stølen: Refinement Principles Supporting the Transition from Asynchronous to Synchronous Communication
342/21/95 A	Andreas Listl, Giannis Bozas: Performance Gains Using Subpages for Cache Coherency Control
342/22/95 A	Volker Heun, Ernst W. Mayr: Embedding Graphs with Bounded Treewidth into Optimal Hypercubes
342/23/95 A	Petr Jančar, Javier Esparza: Deciding Finiteness of Petri Nets up to Bisimulation
342/24/95 A	M. Jung, U. Rüde: Implicit Extrapolation Methods for Variable Coefficient Problems
342/01/96 A	Michael Griebel, Tilman Neunhoeffer, Hans Regler: Algebraic Multigrid Methods for the Solution of the Navier-Stokes Equations in Complicated Geometries
342/02/96 A	Thomas Grauschopf, Michael Griebel, Hans Regler: Additive Multilevel-Preconditioners based on Bilinear Interpolation, Matrix Dependent Geometric Coarsening and Algebraic-Multigrid Coarsening for Second Order Elliptic PDEs
342/03/96 A	Volker Heun, Ernst W. Mayr: Optimal Dynamic Edge-Disjoint Embeddings of Complete Binary Trees into Hypercubes
342/04/96 A	Thomas Huckle: Efficient Computation of Sparse Approximate Inverses
342/05/96 A	Thomas Ludwig, Roland Wismüller, Vaidy Sunderam, Arndt Bode: OMIS — On-line Monitoring Interface Specification
342/06/96 A	Ekkart Kindler: A Compositional Partial Order Semantics for Petri Net Components
342/07/96 A	Richard Mayr: Some Results on Basic Parallel Processes
342/08/96 A	Ralph Radermacher, Frank Weimer: INSEL Syntax-Bericht
342/09/96 A	P.P. Spies, C. Eckert, M. Lange, D. Marek, R. Radermacher, F. Weimer, H.-M. Windisch: Sprachkonzepte zur Konstruktion verteilter Systeme
342/10/96 A	Stefan Lamberts, Thomas Ludwig, Christian Röder, Arndt Bode: PFSLib – A File System for Parallel Programming Environments
342/11/96 A	Manfred Broy, Gheorghe Ştefănescu: The Algebra of Stream Processing Functions
342/12/96 A	Javier Esparza: Reachability in Live and Safe Free-Choice Petri Nets is NP-complete
342/13/96 A	Radu Grosu, Ketil Stølen: A Denotational Model for Mobile Many-to-Many Data-flow Networks
342/14/96 A	Giannis Bozas, Michael Jaedicke, Andreas Listl, Bernhard Mitschang, Angelika Reiser, Stephan Zimmermann: On Transforming a Sequential SQL-DBMS into a Parallel One: First Results and Experiences of the MIDAS Project

Reihe A

342/15/96 A	Richard Mayr: A Tableau System for Model Checking Petri Nets with a Fragment of the Linear Time μ -Calculus
342/16/96 A	Ursula Hinkel, Katharina Spies: Anleitung zur Spezifikation von mobilen, dynamischen Focus-Netzen
342/17/96 A	Richard Mayr: Model Checking PA-Processes
342/18/96 A	Michaela Huhn, Peter Niebert, Frank Wallner: Put your Model Checker on Diet: Verification on Local States
342/01/97 A	Tobias Müller, Stefan Lamberts, Ursula Maier, Georg Stellner: Evaluierung der Leistungsfähigkeit eines ATM-Netzes mit parallelen Programmierbibliotheken
342/02/97 A	Hans-Joachim Bungartz and Thomas Dornseifer: Sparse Grids: Recent Developments for Elliptic Partial Differential Equations
342/03/97 A	Bernhard Mitschang: Technologie f"ur Parallele Datenbanken - Bericht zum Workshop
342/04/97 A	nicht erschienen
342/05/97 A	Hans-Joachim Bungartz, Ralf Ebner, Stefan Schulte: Hierarchische Basen zur effizienten Kopplung substrukturierter Probleme der Strukturmechanik
342/06/97 A	Hans-Joachim Bungartz, Anton Frank, Florian Meier, Tilman Neunhoeffer, Stefan Schulte: Fluid Structure Interaction: 3D Numerical Simulation and Visualization of a Micropump
342/07/97 A	Javier Esparza, Stephan Melzer: Model Checking LTL using Constraint Programming
342/08/97 A	Niels Reimer: Untersuchung von Strategien für verteiltes Last- und Ressourcenmanagement
342/09/97 A	Markus Pizka: Design and Implementation of the GNU INSEL-Compiler gic
342/10/97 A	Manfred Broy, Franz Regensburger, Bernhard Schätz, Katharina Spies: The Steamboiler Specification - A Case Study in Focus
342/11/97 A	Christine Röckl: How to Make Substitution Preserve Strong Bisimilarity
342/12/97 A	Christian B. Czech: Architektur und Konzept des Dycos-Kerns
342/13/97 A	Jan Philipps, Alexander Schmidt: Traffic Flow by Data Flow
342/14/97 A	Norbert Fröhlich, Rolf Schlagenhaft, Josef Fleischmann: Partitioning VLSI-Circuits for Parallel Simulation on Transistor Level
342/15/97 A	Frank Weimer: DaViT: Ein System zur interaktiven Ausführung und zur Visualisierung von INSEL-Programmen
342/16/97 A	Niels Reimer, Jürgen Rudolph, Katharina Spies: Von FOCUS nach INSEL - Eine Aufzugssteuerung
342/17/97 A	Radu Grosu, Ketil Stølen, Manfred Broy: A Denotational Model for Mobile Point-to-Point Data-flow Networks with Channel Sharing
342/18/97 A	Christian Röder, Georg Stellner: Design of Load Management for Parallel Applications in Networks of Heterogenous Workstations

Reihe A

342/19/97 A	Frank Wallner: Model Checking LTL Using Net Unfoldings
342/20/97 A	Andreas Wolf, Andreas Kmoch: Einsatz eines automatischen Theorembeweisers in einer taktikgesteuerten Beweisumgebung zur Lösung eines Beispiels aus der Hardware-Verifikation – Fallstudie –
342/21/97 A	Andreas Wolf, Marc Fuchs: Cooperative Parallel Automated Theorem Proving
342/22/97 A	T. Ludwig, R. Wismüller, V. Sunderam, A. Bode: OMIS - On-line Monitoring Interface Specification (Version 2.0)
342/23/97 A	Stephan Merkel: Verification of Fault Tolerant Algorithms Using PEP
342/24/97 A	Manfred Broy, Max Breitling, Bernhard Schätz, Katharina Spies: Summary of Case Studies in Focus - Part II
342/25/97 A	Michael Jaedicke, Bernhard Mitschang: A Framework for Parallel Processing of Aggregat and Scalar Functions in Object-Relational DBMS
342/26/97 A	Marc Fuchs: Similarity-Based Lemma Generation with Lemma-Delaying Tableau Enumeration
342/27/97 A	Max Breitling: Formalizing and Verifying TimeWarp with FOCUS
342/28/97 A	Peter Jakobi, Andreas Wolf: DBFW: A Simple DataBase FrameWork for the Evaluation and Maintenance of Automated Theorem Prover Data (incl. Documentation)
342/29/97 A	Radu Grosu, Ketil Stølen: Compositional Specification of Mobile Systems
342/01/98 A	A. Bode, A. Ganz, C. Gold, S. Petri, N. Reimer, B. Schiemann, T. Schnekenburger (Herausgeber): ”‘Anwendungsbezogene Lastverteilung”’, ALV’98
342/02/98 A	Ursula Hinkel: Home Shopping - Die Spezifikation einer Kommunikationsanwendung in FOCUS
342/03/98 A	Katharina Spies: Eine Methode zur formalen Modellierung von Betriebssystemkonzepten
342/04/98 A	Stefan Bischof, Ernst-W. Mayr: On-Line Scheduling of Parallel Jobs with Runtime Restrictions
342/05/98 A	St. Bischof, R. Ebner, Th. Erlebach: Load Balancing for Problems with Good Bisectors and Applications in Finite Element Simulations: Worst-case Analysis and Practical Results
342/06/98 A	Giannis Bozas, Susanne Kober: Logging and Crash Recovery in Shared-Disk Database Systems
342/07/98 A	Markus Pizka: Distributed Virtual Address Space Management in the MoDiS-OS
342/08/98 A	Niels Reimer: Strategien für ein verteiltes Last- und Ressourcenmanagement
342/09/98 A	Javier Esparza, Editor: Proceedings of INFINITY’98
342/10/98 A	Richard Mayr: Lossy Counter Machines
342/11/98 A	Thomas Huckle: Matrix Multilevel Methods and Preconditioning

Reihe A

342/12/98 A	Thomas Huckle: Approximate Sparsity Patterns for the Inverse of a Matrix and Preconditioning
342/13/98 A	Antonin Kucera, Richard Mayr: Weak Bisimilarity with Infinite-State Systems can be Decided in Polynomial Time
342/01/99 A	Antonin Kucera, Richard Mayr: Simulation Preorder on Simple Process Algebras
342/02/99 A	Johann Schumann, Max Breitling: Formalisierung und Beweis einer Verfeinerung aus FOCUS mit automatischen Theorembeweisern – Fallstudie –
342/03/99 A	M. Bader, M. Schimper, Chr. Zenger: Hierarchical Bases for the Indefinite Helmholtz Equation
342/04/99 A	Frank Strobl, Alexander Wisspeintner: Specification of an Elevator Control System
342/05/99 A	Ralf Ebner, Thomas Erlebach, Andreas Ganz, Claudia Gold, Clemens Harlfinger, Roland Wismüller: A Framework for Recording and Visualizing Event Traces in Parallel Systems with Load Balancing
342/06/99 A	Michael Jaedicke, Bernhard Mitschang: The Multi-Operator Method: Integrating Algorithms for the Efficient and Parallel Evaluation of User-Defined Predicates into ORDBMS
342/07/99 A	Max Breitling, Jan Philipps: Black Box Views of State Machines
342/08/99 A	Clara Nippl, Stephan Zimmermann, Bernhard Mitschang: Design, Implementation and Evaluation of Data Rivers for Efficient Intra-Query Parallelism
342/09/99 A	Robert Sandner, Michael Mauderer: Integrierte Beschreibung automatisierter Produktionsanlagen - eine Evaluierung praxisnaher Beschreibungstechniken
342/10/99 A	Alexander Sabbah, Robert Sandner: Evaluation of Petri Net and Automata Based Description Techniques: An Industrial Case Study
342/01/00 A	Javier Esparza, David Hansel, Peter Rossmanith, Stefan Schwoon: Efficient Algorithm for Model Checking Pushdown Systems
342/02/00 A	Barbara König: Hypergraph Construction and Its Application to the Compositional Modelling of Concurrency

SFB 342 : Methoden und Werkzeuge für die Nutzung paralleler Rechnerarchitekturen

Reihe B

342/1/90 B	Wolfgang Reisig: Petri Nets and Algebraic Specifications
342/2/90 B	Jörg Desel: On Abstraction of Nets
342/3/90 B	Jörg Desel: Reduction and Design of Well-behaved Free-choice Systems
342/4/90 B	Franz Abstreiter, Michael Friedrich, Hans-Jürgen Plewan: Das Werkzeug runtime zur Beobachtung verteilter und paralleler Programme
342/1/91 B	Barbara Paech1: Concurrency as a Modality
342/2/91 B	Birgit Kandler, Markus Pawlowski: SAM: Eine Sortier- Toolbox -Anwenderbeschreibung
342/3/91 B	Erwin Loibl, Hans Obermaier, Markus Pawlowski: 2. Workshop über Parallelisierung von Datenbanksystemen
342/4/91 B	Werner Pohlmann: A Limitation of Distributed Simulation Methods
342/5/91 B	Dominik Gomm, Ekkart Kindler: A Weakly Coherent Virtually Shared Memory Scheme: Formal Specification and Analysis
342/6/91 B	Dominik Gomm, Ekkart Kindler: Causality Based Specification and Correctness Proof of a Virtually Shared Memory Scheme
342/7/91 B	W. Reisig: Concurrent Temporal Logic
342/1/92 B	Malte Grosse, Christian B. Suttner: A Parallel Algorithm for Set-of-Support Christian B. Suttner: Parallel Computation of Multiple Sets-of-Support
342/2/92 B	Arndt Bode, Hartmut Wedekind: Parallelrechner: Theorie, Hardware, Software, Anwendungen
342/1/93 B	Max Fuchs: Funktionale Spezifikation einer Geschwindigkeitsregelung
342/2/93 B	Ekkart Kindler: Sicherheits- und Lebendigkeitseigenschaften: Ein Literaturüberblick
342/1/94 B	Andreas Listl; Thomas Schnekenburger; Michael Friedrich: Zum Entwurf eines Prototypen für MIDAS