A Direct Symbolic Approach to Model Checking Pushdown Systems (extended abstract)

Alain Finkel

LSV
ENS de Cachan
94235 Cachan, France
finkel@lsv.ens-cachan.fr

Bernard Willems   Pierre Wolper

Université de Liège
Institut Montefiore, B28
4000 Liège, Belgium
{willems,pw}@montefiore.ulg.ac.be

Abstract

This paper gives a simple and direct algorithm for computing the always regular set of reachable states of a pushdown system. It then exploits this algorithm for obtaining model checking algorithms for linear-time temporal logic as well as for the logic CTL*. For the latter, a new technical tool is introduced: pushdown automata with transitions conditioned on regular predicates on the stack content. Finally, this technical tool is also used to establish that CTL* model checking remains decidable when the formulas are allowed to include regular predicates on the stack content.

1 Introduction

In many cases, there are a variety of different algorithms for solving a given verification problem. However, even if they have the same theoretical complexity, these algorithms are rarely equal with respect to building verification tools. First, there is the obvious fact that worst-case complexity is only a crude upper approximation of the actual behavior of an algorithm, especially when one limits the analysis to broad complexity classes (PTIME, PSPACE, ...). Second, factors such as the ease of implementation, the ease of integration with other techniques, and the possibility of separating concerns are crucial. For example, even though they have no theoretical complexity advantage, automata-theoretic algorithms for model checking [VW86,BVW94] have a number of assets with respect to incorporation in tools. Indeed, they separate the "logical part" of the algorithm (building the automaton) from the “combinatorial part” (exploring the extended state space). As a result automata-theoretic model checking is easily combined with a variety of state-space exploration optimizations [CVWY92,GW91] and is a natural addition to a tool such as SPIN [Hol91,GPVW95] that is built around a state-space exploration engine.

In this paper, we consider from this point of view the problem of verifying pushdown systems. These are systems that are finite-state except for the use of one pushdown stack and hence can be modeled by pushdown automata. For one familiar with classical automata theory, this is a natural class of systems to consider in order to obtain decidability results and there are indeed already a number of results on this and related topics [HS91,HJM94,BS95,Wal96], which is not surprising in light of the fact that any pushdown system has a decidable monadic theory [MS85]. However, our goal here is not just to obtain algorithms, but to obtain algorithms that are based on the simplest possible techniques and that extend as naturally as possible what is done in the pure finite-state case.

Our first step is thus to find a technique for computing a symbolic representation of the possibly infinite state-space of a pushdown system. For this, we exploit the not very widely known fact that the state space of such a system is a regular set [Cau92], precisely, for each control location the possible contents of the stack form a regular set, which we represent by a finite-automaton. Furthermore, the construction to obtain this representation is quite simple and can be done in a number of steps that is polynomial, in fact $O(n^3)$, in the size of the pushdown system.

Having obtained this familiar starting point we then turn to the model-checking problem for pushdown systems. For linear-time temporal logic, the automata-theoretic approach of [VW86] solves the problem fairly easily. Indeed, all that is required is a slight modification of the algorithm building the representation of the state space in order to be able to solve the problem of repeated reachability. We then turn to branching time and to the general logic CTL*. This turns out to require some new techniques. Specifically, we are led to consider pushdown automata with transitions that are conditioned by regular conditions on their stack content, which can be viewed as a particular type of stack automata [HU69]. We show that the set of reachable states of such automata can still be computed with an adaptation of our initial technique and exploit this to solve the CTL* model-checking problem for pushdown systems. Doing this we obtain immediately an interesting new result as a payoff: the model checking problem remains solvable when the temporal logic formula is built not only from atomic predicates interpreted on the control states but also from regular predicates on the stack content.

It must be stressed that, beyond this new result, we view the important contribution of our paper to reside in providing a simple, intuitive, and implementation oriented framework for solving model-checking problems on pushdown systems. Indeed, the decidability of model checking for CTL*, though without stack content predicates, is a consequence of the results in [Wal96], which solves the problem for the full μ-calculus, or of the decidability of the emptiness problem for pushdown infinite tree automata [HR94,PP92]. However, although they close the problem from a theoretical point of view, these results are of little help for implementing a usable model checker, especially if one is interested in the more practically relevant limited cases, such as restricted classes of linear-time formulas.

An approach related to ours has also been followed in [BM96]. However our results differ from, and extend, those presented there. First, in [BM96] backwards rather than forwards reachability is considered. This is mostly a detail, but can be a relevant one when trying, for instance, to combine the results presented here with methods for analyzing other types of infinite state systems, for instance those of [BG96], which are based on forwards reachability. Next, we provide the concrete complexity of our algorithm ($O(n^3)$) whereas the algorithm of [BM96] is only characterized as "polynomial". Furthermore, our technique for handling repeated reachability (needed for LTL model checking) is both simpler and more efficient than the one of [BM96]. Finally, we extend our results to CTL*(only CTL is handled in [BM96]) and to formulas that include regular stack content predicates.

2 Pushdown systems

We consider systems that can be modeled by a pushdown automaton. Usually, this pushdown automaton will be computed from the system representation, which might be a set of finite-state processes, one of which uses a pushdown stack as a data structure. Of course there are also a number of other representations from which the pushdown automaton might be obtained, but we do not consider this issue here.

We are working under the hypothesis that the pushdown automaton is a global representation of the system and thus we are not interested in its accepted language, the traditional focus of automata theory. We will thus, consider input-less pushdown automata without an acceptance condition. These are defined as follows.

Definition 2.1 A *pushdown automaton* A is a quadruple (Q, Γ, Δ, q^0) where Q is a finite set of control states, Γ is a stack alphabet, $\Delta \subset \{(Q \times \Gamma) \times (Q \times \{\epsilon\})\} \cup \{(Q \times \{\epsilon\}) \times (Q \times \Gamma)\}$ is a transition relation (ϵ denotes the empty word), and q^0 is the initial control state.

We call the elements of Q *control states* to distinguish them from the global states of the system (elements of $Q \times \Gamma^*$), often called *configurations* in automata theory. The form of the transitions is restricted in such a way that they modify the size of the stack by exactly one symbol, but this is not restrictive. We will represent a transition $((q, \varepsilon), (q', a))$ by (q, a_+, q') or $q \stackrel{a_+}{\to} q'$, and a transition $((q, a), (q', \varepsilon))$ by (q, a_-, q') or $q \stackrel{a_-}{\to} q'$.

The *initial state* of the system is (q^0, ε). A configuration (q', α') is *directly reachable* from a configuration (q, α) if $\alpha = \delta\gamma$, $\alpha' = \delta\gamma'$ and $((q, \gamma), (q', \gamma')) \in \Delta$. For this, we use the notation $(q, \alpha) \Rightarrow (q', \alpha')$ or $(q, \alpha) \stackrel{op}{\Rightarrow} (q', \alpha')$ where $op \in \{a_+, a_-\}$ when there is a need to make the stack operation explicit.

Furthermore, we denote by $\Rightarrow^*$ the reflexive and transitive closure of $\Rightarrow$. When there is a need to make explicit the sequence of stack operations leading from a state (q, α) to a state (q', α'), we write $(q, \alpha) \overset{\sigma}{\Rightarrow}^* (q', \alpha')$ where $\sigma \in \{a_+, a_-\}^*$. The *reachable states* (configurations) of the pushdown automaton are those that are reachable from its initial state.

A *run* of a pushdown automaton is a maximal (finite or infinite) sequence of configurations starting in the initial configuration and such that each configuration is directly reachable from the preceeding one.

3 Computing the Reachable States of a Pushdown Automaton

We are thus given a pushdown automaton $A = (Q, \Gamma, \Delta, q_0)$ and have to compute a representation of the subset of $Q \times \Gamma^*$ that is its set of reachable states. This representation will be a finite-state automaton, called the *reachability automaton* A_r of the pushdown automaton A, defined as follows: $A_r = (Q_r, \Sigma_r, \Delta_r, q_r^0, F_r)$ where

- the set of states Q_r is Q,
- $\Sigma_r = \Gamma$, i.e. the input alphabet of the reachability automaton is the stack alphabet of the pushdown automaton,
- $\Delta_r \subseteq Q_r \times (\Sigma_r \cup \{\varepsilon\}) \times Q_r$ is the smallest relation that satisfies the following conditions where Δ_r^* represents the usual transitive closure of Δ_r:
  · if $(q, a_+, q') \in \Delta$, then $(q, a, q') \in \Delta_r$ and,
  · if $(q, a_-, q') \in \Delta$ and $(q'', a, q) \in \Delta_r^*$, then $(q'', \varepsilon, q') \in \Delta_r$,
- $q_r^0 = q^0$, $F_r = Q_r$.

The relation between A and A_r is given by the following Theorem.

Theorem 3.1 *A state (q, α) is reachable in A iff the state q is reachable in A_r through the word α.*

In other words, the states of A_r are exactly the control states of A, and the stack contents with which a control state q is reachable are exactly the words accepted by A_r when q is taken as the unique final state.

The relation Δ_r can be constructed in the following way. Initially, the relation is empty. Then for every transition $(q, a_+, q') \in \Delta$, we add the transition (q, a, q') to Δ_r. Thereafter, we proceed as follows until saturation. For every transition $(q, a_-, q') \in \Delta$ and every transition (q'', a, q) in the current version of Δ_r^*, we add the transition (q'', ε, q') to Δ_r. This procedure terminates because of the finiteness of Q_r.

This procedure can also be implemented in time $O(n^3)$ where n is the size of the pushdown automaton. In fact, instead of constructing the relation Δ_r, we will construct the relation $\Delta_{r,1}$ which will play the same role as Δ_r. The relation $\Delta_{r,1} \subseteq Q_r \times (\Sigma_r \cup \{\varepsilon\}) \times Q_r$ is the smallest relation that satisfies the following conditions:

- if $(q, a_+, q') \in \Delta$, then $(q, a, q') \in \Delta_{r,1}$,

- for every $q \in Q_r$, $(q, \varepsilon, q) \in \Delta_{r,1}$,
- if $(q, a_+, q') \in \Delta$, $(q', \varepsilon, q'') \in \Delta_{r,1}$, and $(q'', a_-, q''') \in \Delta$, then $(q, \varepsilon, q''') \in \Delta_{r,1}$,
- if $(q, \varepsilon, q') \in \Delta_{r,1}$ and $(q', \varepsilon, q'') \in \Delta_{r,1}$, then $(q, \varepsilon, q'') \in \Delta_{r,1}$.

The transitions of Δ_r and $\Delta_{r,1}$ that are not labeled by ε are the same. Every ε-transitions of Δ_r belong to $\Delta_{r,1}$. The ε-transitions of $\Delta_{r,1}$ correspond to finite sequences of ε-transitions of Δ_r. So the theorem 3.1 still holds with $\Delta_{r,1}$ instead of Δ_r.

The algorithm given at Figure 1 computes all the ε-transitions of $\Delta_{r,1}$. Its time complexity is $O(|A^3|)$.

```
procedure saturate(A = (Q, Γ, Δ, q^0) : pushdown automaton)
begin
    stack := ∅; hash := ∅;
    for every q ∈ Q, put (q, q) on stack;
    for every q, q' ∈ Q,
        (q, q').c-direct := ∅;
        (q, q').c-trans := ∅;
    for every transitions a_{+,u→q} and a_{−,q'→v} of A,
        put (u, v) on (q, q').c-direct;
    for every q, q' ∈ Q,
        if q ≠ q' then
            for every t ∈ Q,
                    put [(q', t) → (q, t)] and [(t, q) → (t, q')] on (q, q').c-trans;
    while stack ≠ ∅
        α := pop(stack);
        put α in hash; (if it not already belongs to hash)
        transfer α.c-direct into stack;
        for every [γ → β] ∈ α.c-trans
            if γ ∈ hash then put β on stack
                    else put β on γ.c-direct;
    return hash;
end
```

Fig. 1. computation of the ε-transitions of $\Delta_{r,1}$

The idea of the algorithm is to do the computation of the transitive closure of ε-transitions only once by storing information on how ε-transitions can be combined. The ε-transitions that have been computed are stored in the hash table (constant access time) *hash*. A pushdown stack *stack* is used to store ε-transitions that have been computed but that have not yet been exploited in conjunction with other transitions. This stack is initialized with the trivial transitions (q, q).

To help determine the ε-transitions that can be added as a consequence of an ε-transition (q, q'), 2 stacks are used for each pair (q, q'). The first, (q, q')-*c-direct* is initialized to the set of ε-transitions that are consequences of an a_+-transition, an a_--transition, and the ε-transition from q to q'. Note that the cumulative size of the *c-direct* stacks is at most $O(|A^2|)$. The second stack, (q, q')-*c-trans* encodes the fact that, given the ε-transition (q, q'), for every known ε-transition (q', t), one can add the ε-transition (q, t), and similarly for every known ε-transition (t, q), one can add the ε-transition (t, q'). This is respectively encoded by the transition implications $[(q', t) \rightarrow (q, t)]$ and $[(t, q) \rightarrow (t, q')]$. The cumulative size of the elements of the *c-trans* stacks is at most $O(|A^3|)$.

The algorithm then proceeds by processing every transition on *stack* and determining their consequences with the help of the *c-direct* and *c-trans* stacks. To see that it operates within the announced complexity, notice that it contains initially $O(|A|)$ elements and that it handles each element present in a *c-direct* stack once and every element present in a *c-trans* stack at most twice.

To summarize, we have shown the following.

Theorem 3.2 *Given a pushdown automaton A, the corresponding reachability automaton can be computed in time $O(|A^3|)$.*

The construction we have just given computes the states that are reachable with an initially empty stack. It can easily be modified to compute the states that are reachable when the initial content of the stack is within a given regular language L represented by a finite automaton A_L. Indeed, the automaton A_r is then obtained by incorporating the automaton A_L into the construction given above. Precisely, one adds all states and transitions of A_L, takes the initial state to be that of A_L, adds ε-transitions between the accepting states of A_L and the initial state inherited from the pushdown automaton, and then completes Δ_r to ensure that it satisfies the given condition with respect to the a_--transitions of the pushdown automaton.

4 Linear-time temporal properties

We now turn to the problem of verifying a linear-time temporal logic [MP92] property on a pushdown system. Technically, we consider a temporal logic formula f built from a set of atomic propositions *Prop* and a pushdown automaton $A = (Q, \Gamma, \Delta, q^0)$ extended with a labeling function $\Lambda : Q \rightarrow 2^{\mathrm{Prop}}$ assigning truth values to the atomic propositions in each control state. The problem is to check that, with respect to the labeling function Λ, all infinite runs of A satisfy the formula f. To solve this problem, we adopt the approach of [VW86] and build a Büchi automaton $A_{\neg f}$ over the alphabet $\Sigma = 2^{Prop}$ accepting all the models of $\neg f$, the negation of f. The next step is to take the product of the pushdown automaton A and of the Büchi automaton $A_{\neg f}$ and check whether this product is empty or not. This product is actually a pushdown automaton of the form we have considered so far (note that once the product is computed the labeling function is no longer necessary) extended with a Büchi acceptance condition, we will call it a *Büchi pushdown automaton.*

So, we are left with the problem of testing emptiness of a Büchi pushdown automaton, i.e. a structure $A = (Q, \Gamma, \Delta, q^0, F)$ where F is a set of accepting control states. Such an automaton is nonempty if it has an infinite run going infinitely often through some element of F. At first thought, it might seem that constructing the reachability automaton as in Section 3 could solve the problem. This is not so, because we now have to solve a *repeated* reachability problem rather than just a reachability problem and our previous construction does not preserve the necessary information. For instance, the sequence of transitions $q_1 \stackrel{a+}{\to} q_2 \stackrel{a-}{\to} q_3$ in the pushdown automaton is only represented by the transition $q_1 \stackrel{\varepsilon}{\to} q_3$ in the reachability automaton, hence omitting the state q_2, which is problematic if q_2 is accepting but neither q_1 or q_3 is.

To solve this problem, we introduce a two steps construction to obtain a Büchi reachability automaton. The first step is similar to the one described in section 3, but introduces two types of ε-transitions : those that do not go through an accepting state (ε) and those that do go through an accepting state (ε_a). The second step then eliminates the ε_a-transitions by introducing a new set of dummy accepting states. Concretely, given a Büchi pushdown automaton $A = (Q, \Gamma, \Delta, q^0, F)$, the result of the first step is an automaton

$$A_{br_1} = (Q_{br_1}, \Sigma_{br_1}, \Delta_{br_1}, q^0_{br_1}, F_{br_1})$$

defined as the automaton A_r of Section 3, except that its set of accepting states F_{br_1} is the set F of accepting control states of A and that the transition relation Δ_{br_1} is the smallest relation included in $Q_{br_1} \times (\Sigma_{br_1} \cup \{\varepsilon, \varepsilon_a\}) \times Q_{br_1}$ that satisfies the following conditions:

- if $(q, a_+, q') \in \Delta$, then $(q, a, q') \in \Delta_{br_1}$ and,
- if $(q, a_-, q') \in \Delta$, $(q'', a, q) \in \Delta^*_{br_1}$, and either q is accepting, one of the states on an a-labeled path from q'' to q is accepting, or an ε_a-transition appears on such a path, then $(q'', \varepsilon_a, q') \in \Delta_{br_1}$,
- if $(q, a_-, q') \in \Delta$, $(q'', a, q) \in \Delta^*_{br_1}$, and $(q'', \varepsilon_a, q') \notin \Delta_{br_1}$, then $(q'', \varepsilon, q') \in \Delta_{br_1}$.

The second step then transforms A_{br_1} by adding new states and transitions. Specifically, we construct the *Büchi reachability automaton* $A_{br} = (Q_{br}, \Sigma_{br}, \Delta_{br}, q^0_{br}, F_{br})$ as follows:

- $Q_{br} = Q_{br_1} \cup Q^*_{br_1}$ with $Q^*_{br_1}$ being a new copy of Q_{br_1}, i.e. we duplicate the set of states of A_{br_1},
- every transition $q \stackrel{\varepsilon_a}{\to} q'$ is replaced by $q \stackrel{\varepsilon}{\to} q'^*$ and $q'^* \stackrel{\varepsilon}{\to} q'$,
- the set of accepting states F_{br} is $F_{br_1} \cup Q^*_{br_1}$.

With a careful implementation, the construction of this Büchi reachability automaton can be done in time $O(n^3)$ with respect to the size of the pushdown automaton.

The relation between A and A_{br} is given by the following theorem.

Theorem 4.1 *A Büchi pushdown automaton A has an accepting run iff the associated Büchi reachability automaton A_{br} also has an accepting run.*

In order to solve the model checking problem for the logic CTL* we will need to be able to compute the set of initial stack contents for which a linear-time temporal logic formula is satisfied by a pushdown automaton, i.e. the initial stack contents such that all infinite runs of the pushdown automaton do satisfy the temporal logic formula. Formally, if we denote by $A_{(q_0,m)} \models f$ the fact that all infinite runs of a pushdown automaton A starting in the configuration (q_0, m) satisfy f, we are looking for the set

$$\text{Init}(A, f) = \{m \in \Gamma^* \mid A_{(q_0,m)} \models f\}$$

where Γ is the stack alphabet of A. Our solution to this problem goes along the lines of the model checking procedure we have just given. First, we solve the complement of the problem, i.e. we compute the set $\overline{\text{Init}}(A, f)$ of initial stack contents from which *some* computation does not satisfy f (satisfies $\neg f$) and then complement the set that is obtained. To compute $\overline{\text{Init}}(A, f)$ we again take the product of A and $A_{\neg f}$ (the automaton accepting all sequences not satisfying f) to obtain a Büchi pushdown automaton $A_b = \{Q_b, \Gamma_b, \Delta_b, q_b^0, F_b\}$ from which we compute the Büchi reachability automaton A_{br}. Next, we use the following set of observations.

(i) If a word m is in $\overline{\text{Init}}(A, f)$, then all words in $\Gamma^* m$ are in $\overline{\text{Init}}(A, f)$. Indeed, having elements on the stack below m can only increase the set of possible computations.

(ii) An accepting computation of A_b cycles through a strongly connected component of this automaton that contains some accepting state. Furthermore, when there is an accepting computation, there is one that cycles in a purely periodic way through a strongly connected component. On such a periodic path, the stack might grow and shrink, but one can always identify a point in the periodic path such that the stack is never smaller than at that point. Let us call the corresponding control point a *cycle starting point*. The initial stack contents for which A_b has some accepting run are all those from which it is possible to reach such a cycle starting point.

(iii) Cycle starting points can be identified in A_{br}. To see this, remember that the states of A_{br} are either states of A_b or dummy accepting states. The cycle starting points are then the A_b states of the nontrivial strongly connected components of A_{br} that contain at least one accepting state.

So, we use A_{br} to determine the cycle starting points. Let Q_c be this set. We then build a finite automaton on finite words accepting the minimal stack contents that allow cycle starting states to be reached. This automaton is $A_{\overline{\text{Init}}} = (Q_{\overline{\text{Init}}}, \Gamma, \Delta_{\overline{\text{Init}}}, q^0_{\overline{\text{Init}}}, F_{\overline{\text{Init}}})$ where

- $Q_{\overline{\text{Init}}} = Q_b \cup \{q^0_{\overline{\text{Init}}}\}$ (the states are those of Q_b plus a new initial state),
- $\Delta_{\overline{\text{Init}}}$ is defined as follows:
  - $(q', a, q) \in \Delta_{\overline{\text{Init}}}$ iff $q \xrightarrow{a_-} q'$ is a transition of Δ_b (the inverted a_--transitions of A_b),
  - $(q', \varepsilon, q) \in \Delta_{\overline{\text{Init}}}$ iff $q \xrightarrow{\varepsilon} q'$ is a transition of $\Delta_{A_{br}}$ or $q \xrightarrow{\varepsilon_a} q'$ is a transition of the automaton A_{br_1} obtained in the first stage of the construction of A_{br} (we add the inverted ε-transitions of A_{br}),
  - $(q^0_{\overline{\text{Init}}}, \varepsilon, q) \in \Delta_{\overline{\text{Init}}}$ for all states $q \in Q_c$ (all cycle starting points are immediately reachable from the new initial state),

- $(q^0_{\overline{\mathrm{Init}}}, \gamma, q^0_{\overline{\mathrm{Init}}}) \in \Delta_{\overline{\mathrm{Init}}}$ for all $\gamma \in \Gamma$ (we allow prefixing with an arbitrary word),
- $F_{\overline{\mathrm{Init}}} = \{q^0_b\}$ (the only accepting state is the initial state of A_b).

Theorem 4.2 *The language accepted by $A_{\overline{\mathrm{Init}}}$ is $\overline{\mathrm{Init}}(A, f)$.*

So, to obtain $\mathrm{Init}(A, f)$ we just need to complement the automaton $A_{\overline{\mathrm{Init}}}$.

5 Büchi regularly constrained pushdown automata

In order to extend our results and in particular to obtain a model checking procedure for CTL*, we need to extend Büchi pushdown automata with the possibility of conditioning transitions by regular predicates on the stack content. We use the following definition.

Definition 5.1 A *Büchi regularly constrained pushdown automaton* A is a quadruple $A = (Q, \Gamma, \Delta, q^0, F)$ where

- Q is a finite set of control states, Γ is a stack alphabet,
- the transition relation Δ is a finite subset of

$$\{\mathcal{R} \times (Q \times \Gamma) \times (Q \times \{\varepsilon\})\} \cup \{\mathcal{R} \times (Q \times \{\varepsilon\}) \times (Q \times \Gamma)\}$$

  where $\mathcal{R}$ denotes the set of all regular subsets of Γ^* and ε the empty word,
- q^0 is the initial control state, F is a set of accepting control states.

We represent a transition $(C, (q, \varepsilon), (q', a))$ by $(q, C \to a_+, q')$ or $q \overset{C \to a_+}{\to} q'$, and a transition $(C, (q, a), (q', \varepsilon))$ by $(q, C \to a_-, q')$ or $q \overset{C \to a_-}{\to} q'$.

A configuration (q', α') is *directly reachable* from a configuration (q, α) if $\alpha = \delta\gamma$, $\alpha' = \delta\gamma'$ and there exists some $C \in \mathcal{R}$ such that $\alpha \in C$ and $(C, (q, \alpha), (q', \alpha')) \in \Delta$. From this, we define reachable configurations and runs. A *run* of a Büchi regularly constrained pushdown automaton is a maximal (finite or infinite) sequence of configurations starting in the initial configuration and such that each configuration is directly reachable from the preceeding one. An *accepting run* is an infinite run that visits accepting control states infinitely often (irrespectively of the stack content). An automaton is said to be *nonempty* if it admits some accepting run; otherwise, it is said to be *empty*.

Note that a Büchi pushdown automaton is a particular Büchi regularly constrained pushdown automaton (take $C = \Gamma^*$ as the regular guard). Regularly constrained pushdown automata are also a special case of *stack automata* as defined for instance in [HU69]. Stack automata accept more than the context-free languages.

However, here we are not interested in the accepted language, but in determining nonemptiness. We show that this remains decidable for Büchi regularly constrained pushdown automaton. This is shown by a construction that eliminates the regular constraints while preserving nonemptiness. The construction proceeds in two steps. The first step eliminates the a_--transitions similarly to what is done in Section 4. The second step then eliminates the regular constraints on the stack content.

First step: removing all a_--transitions

We proceed just like in the construction of the Büchi reachability automaton associated to a pushdown automaton but now the added ε- and ε_a-transitions will be labeled by a regular language representing the enabling condition for the corresponding sequence of transitions. Precisely, a path of the form

$$q_1 \xrightarrow{C_1 \to \tau_1} q_2 \ldots \xrightarrow{C_r \to \tau_r} q_{r+1} \xrightarrow{C_{r+1} \to a_+} q_{r+2} \xrightarrow{C_{r+2} \to \tau_{r+2}} q_{r+3} \ldots \xrightarrow{C_s \to \tau_s} q_{s+1} \xrightarrow{C_{s+1} \to a_-} q_{s+2}$$

where $0 \leq r < s$, C_i denotes a regular language and τ_i an ε- or an ε_a-transition, will cause the following transition to be added to the transition relation:

$$q_1 \xrightarrow{C \to \tau} q_{s+2} \text{ with } C = \bigcap_{1 \leq i \leq r+1} C_i \cap a_-(\bigcap_{r+2 \leq i \leq s+1} C_i)$$

where $a_-(L) = \{m \in \Gamma^* \mid m\ a \in L\}$ (note that $a_-(L_1 \cup L_2) = a_-(L_1) \cup a_-(L_2)$ and $a_-(L_1 \cap L_2) = a_-(L_1) \cap a_-(L_2)$), and τ is ε_a if at least one accepting control state has been visited explicitely or implicitly (through an ε_a-transition), otherwise τ is ε. Note that this will always generate a finite number of regular conditions since all constructed conditions are of the form $\vee_i \wedge_j (m_{ij})_-(C_{ij})$ where $m_{ij} \in \Gamma^*$ and $C_{ij} \in \mathcal{C}$ and since, for any regular language C, the set $|\{m_-(C) \mid m \in \Gamma^* \ \& \ C \in \mathcal{C}\}|$ is always finite.

Second step: reduction of Büchi constrained pushdown automaton without any a_--operations

After eliminating all a_--transitions, eliminating the regular conditions is fairly straightforward. Indeed, now the only transitions modifying the stack content are transitions adding an element a to the stack. A finite automaton for a stack condition can thus follow these transitions and indicate by its state whether the condition is satisfied or not. These finite state automata can then be incorporated into the control state of the pushdown automaton, hence yielding an automaton without constraints.

We summarize these construction and results in the following theorem.

Theorem 5.2 *Given a Büchi regularly constrained pushdown automaton A_c, we can construct a Büchi pushdown automaton A such that A_c and A are simultaneously empty.*

As we did for Büchi pushdown automata, we now give a procedure for computing the set $\mathrm{Init}(A)$ of initial stack contents that enable an accepting run for a Büchi regularly constrained pushdown automaton. As in Section 4, we first eliminate the a_- transitions using the construction given in the "first step" above with the goal of using the resulting automaton to determine when accepting cyclic computations are possible. However, we now have to take the constraints into account. To do this, we embody the constraints into finite automata as in the "second step" of the construction above. This then enables us to determine for each state of the constrained Büchi reachability automaton which language allows an accepting cycle to be repeatedly followed. Indeed, cycles that are possible in the product of the Büchi reachability automaton and of the automata representing the constraints are possible in the corresponding state of the Büchi reachability automaton for a stack content described by the constraint automata components of the product and their current state.

The final step of the computation of $\mathrm{Init}(A)$ is then to determine from which initial stack contents the various states can be reached with the stack content that makes a cycle possible. This is done similarly to the construction of Section 4, but incorporating the constraints, which is possible since they are given by finite automata.

6 Branching temporal properties: CTL*

As shown in [EL85], branching-time model checking [Eme90] can be reduced to linear-time model checking. The idea is to start with the innermost path formulas, verify them with a linear-time model checking procedure and then label the structure with the result. This being done, one can move to the next level of path formulas and repeat the procedure. In the context of pushdown systems, this does not quite work. Indeed, since we are dealing with infinite state spaces, we cannot label individual states. One could be tempted to just label the control states, but the problem with this is that the truth of a linear-time temporal formula can depend on the initial stack content.

So, the idea of our algorithm is to use the procedure we have given to compute the set of stack contents from which a linear formula is satisfied. Again, we do this for the innermost path formulas first. This gives us a labeling of the structure, in terms of regular sets, which can be represented as a Büchi constrained pushdown automaton. Using the procedure outlined in Section 5, we repeat this computation for the next level and so on until we reach the outermost level. At this point we solve the emptiness problem for the Büchi constrained pushdown automaton obtained and conclude.

As an immediate consequence of the procedure we have just described we can conclude that model checking of CTL* remains decidable on pushdown systems for formulas that include regular predicates on the stack content.

Theorem 6.1 *The* CTL* *model-checking problem for pushdown automata and formulas built from atomic propositions interpreted on the control states and regular predicates on the stack content is decidable.*

References

[BG96] B. Boigelot and P. Godefroid. Symbolic verification of communication protocols with infinite state spaces using QDDs. In *Proc. 8th Conference on Computer Aided Verification*, volume 1102, pages 1–12, New Brunswick, August 1996. Lecture Notes in Computer Science, Springer-Verlag.

[BM96] A. Bouajjani and O. Maler. Reachability analysis of pushdown automata. In *INFINITY workshop*, 1996.

[BS95] O. Burkart and B. Steffen. Composition, decomposition and model checking of pushdown processes. *Nordic Journal of Computing*, 2(2):89–125, 1995.

[BVW94] O. Bernholtz, M.Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. In *Computer Aided Verification, Proc. 6th Int. Workshop*, Stanford, California, June 1994. Lecture Notes in Computer Science, Springer-Verlag. Full version available from authors.

[Cau92] D. Caucal. On the regular structure of prefix rewriting. *Theoretical Computer Science*, 106:61–86, 1992.

[CVWY92] C. Courcoubetis, M.Y. Vardi, P. Wolper, and M. Yannakakis. Memory efficient algorithms for the verification of temporal properties. *Formal Methods in System Design*, 1:275–288, 1992.

[EL85] E.A. Emerson and C.-L. Lei. Modalities for model checking: Branching time logic strikes back. In *Proceedings of the Twelfth ACM Symposium on Principles of Programming Languages*, pages 84–96, New Orleans, January 1985.

[Eme90] E.A. Emerson. Temporal and modal logic. *Handbook of theoretical computer science*, pages 997–1072, 1990.

[GPVW95] R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Proc. 15th Work. Protocol Specification, Testing, and Verification*, Warsaw, June 1995. North-Holland.

[GW91] P. Godefroid and P. Wolper. A partial approach to model checking. In *Proc. 6th Symp. on Logic in Computer Science*, pages 406–415, Amsterdam, July 1991.

[HJM94] Yoram Hirshfeld, Mark Jerrum, and Faron Moller. A polynomial-time algorithm for deciding equivalence of normed context-free processes. In *35th Annual Symposium on Foundations of Computer Science*, pages 623–631, Santa Fe, New Mexico, 20–22 November 1994. IEEE.

[Hol91] G. Holzmann. *Design and Validation of Computer Protocols.* Prentice-Hall International Editions, 1991.

[HR94] Harel and Raz. Deciding emptiness for stack automata on infinite trees. *Information and Computation (formerly Information and Control)*, 113, 1994.

[HS91] Hans Hüttel and Colin Stirling. Actions speak louder than words: Proving bisimilarity for context-free processes. In *Proceedings, Sixth Annual IEEE Symposium on Logic in Computer Science*, pages 376–386, Amsterdam, The Netherlands, 15–18 July 1991. IEEE Computer Society Press.

[HU69] J.E. Hopcroft and J.D. Ullman. *Formal Languages and their Relation to Automata.* Addison-Wesley, 1969.

[MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification.* Springer-Verlag, Berlin, January 1992.

[MS85] Muller and Schupp. The theory of ends, pushdown automata, and second-order logic. *Theoretical Computer Science*, 37, 1985.

[PP92] Peng and Purushothaman. Empty stack pushdown omega-tree automata. In *Colloquium on Trees in Algebra and Programming.* LNCS 581, Springer-Verlag, 1992.

[VW86] M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the First Symposium on Logic in Computer Science*, pages 322–331, Cambridge, June 1986.

[Wal96] Igor Walukiewicz. Pushdown processes: Games and model checking:. In *Proc. 8th Workshop on Computer Aided Verification*, volume 1102 of *Lecture Notes in Computer Science*, pages 62–74, New Brunswick, NJ, USA, July/August 1996. Springer-Verlag.