Beyond Regular Model Checking

Dana Fisman and Amir Pnueli

Faculty of Mathematics and Computer Science
The Weizmann Institute of Science, Rehovot, Israel
{dana,amir}@wisdom.weizmann.ac.il

Abstract. In recent years, it has been established that *regular model checking* can be successfully applied to several parameterized verification problems. However, there are many parameterized verification problems that cannot be described by regular languages, and thus cannot be verified using regular model checking. In this study we try to practice symbolic model checking using classes of languages more expressive than the regular languages. We provide three methods for the uniform verification of non-regular parameterized systems.

1 Introduction

During the last two decades, several formal methods have been developed to answer the *verification problem* of finite-state systems. The verification problem asks the question of whether a given reactive system is correct relative to some specification. Although many interesting concurrent programs are in fact finite state, they are often given semantically in terms of a parameter n, representing the number of concurrent processes. Such a schematic program really represents an infinite family of uniformly defined programs. Such programs are often referred to as *parameterized systems*. A challenging problem is to provide methods for the *uniform* verification of parameterized systems, i.e., proving correctness for all possible programs obtained by instantiating the parameter. We refer to this problem as the *parameterized verification problem.* In 1986 Apt and Kozen proved that, in general, the parameterized verification problem is undecidable, even when each instance is finite-state [AK86]. However, for specific families the problem may be solvable.

Model Checking (MC) is an automatic technique for answering the verification problem i.e., verifying that a given program satisfies a given specification. In this framework, specification are usually expressed by a propositional temporal logic and the programs are modeled by state-transition systems. The model checking procedure performs an exhaustive search of the state space of the system to determine whether the system satisfies the specification. The use of an exhaustive state-space exploration limits the application of model checking to finite-state systems.

However, it is possible to perform a state-space exploration of infinite-state systems as well, by using an implicit representation for sets of states. The framework of model checking where sets of states are represented implicitly using some symbolic representation is known as *symbolic model checking (SMC)* [BCM$^+$92]. Symbolic model checking can be applied to infinite-state systems, although in this case, the termination of the procedure is not guaranteed.

Regular model checking is an application of symbolic model checking where regular expressions are used to represent symbolically sets of states [KMM$^+$97]. Regular model checking can be applied to any verification problem that is expressible using regular languages. Such an application is successful if the regular model checking procedure has terminated. In recent years, it has been established that regular model checking can be successfully applied to several types of parameterized verification problems.

However, many interesting parameterized systems cannot be handled by regular model checking since the class of regular languages is not strong enough to express them. An example for such a verification problem is the Peterson algorithm for mutual exclusion among n processes [Pet81]. The existence of such examples, is the main motivation for this study.

In this study we try to practice symbolic model checking using as symbolic representation classes of languages more expressive than the regulars. As a first attempt, we use context-free languages at the last step of the symbolic model checking procedure, while regular languages are used in the procedure until this last step. This seemingly slight change, already enables us to verify mutual exclusion for the Peterson algorithm for an arbitrary number of processes.

By carefully examining the model checking procedure, one can compile a list of the requirements a class of languages must meet in order to be *adequate* for symbolic model checking [KMM$^+$97]. Such a list consists of several operations the class must be effectively closed under, and several questions that must be effectively decidable for the class. We recognize that the class of languages accepted by deterministic pushdown automata $\mathcal{L}_{DPDA}$, meets all requirements but two: the class is not closed under projection and there is no known efficient algorithm to decide equivalence. We thus direct our effort to find a class, which is a subset of the class $\mathcal{L}_{DPDA}$, and possesses efficient algorithms for computing projection and deciding equivalence. This class must also satisfy the rest of the requirements.

We succeed to define a sub-class of $\mathcal{L}_{DPDA}$, which we denote $\mathcal{L}_{DPDA-M}$, for which there exists a semi-algorithm [1] to compute projection. In addition, there exists an efficient algorithm to answer the equivalence problem for this class. This class also satisfies all other requirements. Thus, we establish a class, which is more expressive than the class of regular languages, and yet is adequate for symbolic model checking. The Peterson example can be symbolically model checked, using languages in this class.

[1] By semi-algorithm we mean an "algorithm" that is not guaranteed to halt. The semi-algorithm is assured to give the right answer in case it does halt, however for some instances the semi-algorithm may run forever. In practice, when faced with a semi-algorithm one usually bound the running time by a special "aborting" parameter. If the semi-algorithm reaches the aborting bound then the run is aborted without results.

Recall that the standard fix-point computation in the symbolic model checking procedure is not guaranteed to terminate when applied to infinite-state systems. This difficulty must be addressed when considering a class of languages to be adequate for symbolic model checking. In order for such a class to be *practically* adequate for symbolic model checking, it must also provide means to tackle this difficulty. Indeed, for regular model checking, many techniques to overcome this problem have been developed (see related work section). The common idea behind these techniques lies in calculating the effect of taking an arbitrary number of system-transitions in one step, often refer to as calculating meta-transitions or "accelerations".[PS00,BJNT00].

Finally, we focus on a special case of the class that we have found. For this class there exists an algorithm to compute projection. In addition, we show that all techniques developed for calculating meta-transitions for regular model checking, can be applied to this class. This class is therefore *practically* adequate for symbolic model checking. The Peterson example, escorting us through our study, can also be verified using languages in this class.

Related Work The problem of uniform verification of a parameterized systems is one of the most thoroughly researched problems in computer science. Many methods have been proposed for the uniform verification of parameterized systems. These include explicit induction [EN96,SG92], network invariants, which can be viewed as implicit induction [WL89,HLR92,LHR97,KM95,KP00], methods that can be viewed as abstraction and approximation of network invariants [BCG86,SG89,CGJ95], and other methods that can be viewed as based on abstraction [ID96,EN96].

Regular model checking has been advocated by [KMM$^+$97] and [WB98] as a uniform paradigm for algorithmic verification of several classes of parameterized and infinite-state systems. The use of regular languages to express state properties goes back to [**?**]. The problem of calculating meta-transitions is one of the most laborious problems in this field of research. It has been thoroughly researched [ABJ98,ABJN99][FO97], resulting in several corrective techniques, such as acceleration [PS00][BF00], calculation of the transitive closure [JN00,BJNT00] and widening [BJNT00,CC77,LHR97].

The study of model checking pushdown systems has recently been receiving growing attention. Various definitions and extensions to pushdown systems can be found in the literature [BS95], [BG96], [BM96], [BGWW97], [BEM97], [WB98] and [EHRS00]. There are numerous works on model checking of pushdown automata with various temporal logics. Walukiewicz [Wal96] considers the μ-calculus, Bouajjani et al. [BEM97] consider the linear μ-calculus and the alternation free μ-calculus, while [FWW97] consider LTL and CTL*.

Nevertheless, these efforts do not pertain to this study for two reasons: first, we use the pushdown automata to represent the set of states, while the above work considers the pushdown automata as the system being analyzed. Furthermore, a fundamental difference is that all systems previously considered share a common characteristic: a regular language represents their state space, whereas our main interest is systems whose state space cannot be represented by a regular language.

There has been other works also pursuing symbolic representations which are more expressive than regular languages. Studies by Boigelot and Wolper [WB94] and Comon and Jurski [CJ98] give symbolic representations for configurations of systems with counters. Both studies provide methods to compute meta-transitions. Boigelot and Wolper [WB98] also furnish a semi-algorithm for deciding inclusion, and an algorithm for determining equivalence.

Perhaps the work most related to this research is by Bouajjani and Habermehl [BH97]. They define a symbolic representation, denoted CQDD, which is an extension of the QDDs defined by [BG96]. While QDDs are finite automata, CQDDs are a combination of restricted finite automata and linear constrains on a number of occurrences of symbols. The CQDDs are a symbolic representation which is more expressive than regular languages. As an example, they show that CQDDs accept the language $a^n b^m a^n b^m$. However, to our understanding, the CQDDs are not strong enough to express the Peterson example, which we were able to verify using all our methods.

Outline The paper is organized as follows. Section 2 provides some preliminaries. Section 3 discusses adequacy of classes of automata for symbolic model checking. Section 4,5, and 6 present the three methods we suggest for the uniform verification of non-regular parameterized systems. Section 7 concludes.

2 Preliminaries

2.1 System Model

Definition 1. Let Σ be an alphabet. A *bi-letter* over Σ is an element of $\Sigma \times \Sigma$. A *bi-word* over Σ is a string of bi-letters over Σ (i.e., an element of $(\Sigma \times \Sigma)^*$). A *bi-language* over Σ is a set of bi-words over Σ (i.e., a subset of $(\Sigma \times \Sigma)^*$). We use the notations $\left[{a \atop b}\right]$ and $\left[{u \atop v}\right]$ to denote respectively the bi-letter (a, b) and the bi-word $\left[{a_1 \atop b_1}\right] \left[{a_2 \atop b_2}\right] \cdots \left[{a_n \atop b_n}\right]$ where $u = a_1 a_2 ... a_n$ and $v = b_1 b_2 ... b_n$.

Given a bi-language L over Σ, we denote by $L{\Downarrow_1}$ ($L{\Downarrow_2}$) the *projection* of L on the first (respectively, second) coordinate. Given a language L over Σ, we denote by $L \times \Sigma^*$ the language $\left\{ \left[{w \atop u}\right] \ : \ w \in L,\, u \in \Sigma^*,\, |u| = |v| \right\}$, referred to as *left lifting*. Similarly, we define *right lifting* and use the notation $\Sigma^* \times L$. We use the standard notations $\overline{L}$ and $L_1 \cap L_2$ to denote the complement of a language and the intersection of two languages respectively.

Definition 2. A *system* is a quadruple $(\Sigma, \chi, \Theta, \rho)$ where

- Σ is a finite alphabet.
- $\chi \subseteq \Sigma^*$ is a language over Σ, denoting the set of *states*.
- $\Theta \subseteq \chi$ is a language over Σ, denoting the set of *initial states*.
- $\rho \subseteq \chi \times \chi$ is a bi-language over Σ, denoting the *transition relation*.

2.2 Symbolic Model Checking and Regular Model Checking

The *verification problem* for a system M and a property φ is to decide whether φ holds over all computations of M. *Model checking* is an automatic technique for answering the verification problem. In *symbolic model checking*, some symbolic representation $\mathcal{C}$ is used to represent sets of states. In the framework of symbolic model checking using the representation $\mathcal{C}$, the system and the property to be verified are represented as expressions in $\mathcal{C}$: The system is defined using $\mathcal{C}$-expressions to describe its components, which are essentially sets of states or relations over sets of states. The invariance property is defined using some expression in $\mathcal{C}$ to describe the sets of states satisfying it. Then, the model checking procedures operate by manipulating sets of states (instead of individual states) through operations on the expressions of $\mathcal{C}$. Symbolic model checking can be applied to infinite-state systems, although in this case, the termination of the procedure is not guaranteed. We refer to this problem as the *convergence problem.*

2.3 Regular Model Checking

Regular model checking [KMM+97,WB98] is an application of symbolic model checking, where regular expressions are used as the (symbolic) assertional language for describing sets of states. Regular expressions have been proven to be useful representations for several classes of infinite-state system, e.g., parameterized systems, counter systems and unbounded FIFO-channel systems.

Using this framework, the system's components, and the property to be verified are defined by means of regular expressions. Usually, we assume a given alphabet Σ to represent a local state. The set of global states χ, the set of initial states Θ and the property φ, are specified by a regular expression (defining a regular language) over Σ. The transition relation ρ is specified by a regular expression using bi-letters over Σ (sometimes referred to as a bi-regular expression).

Example 1. (TOKEN-STRING)
Consider an array of processes that pass a token from left to right. We define the alphabet $\Sigma = \{1, 0\}$ to denote the local state of a process, i.e., that the process has or has not the token. A global state of a system consisting of n processes is defined by a word of length n, each letter describes the state of one process. The set of global states χ is defined to be the language of all words of positive length, given by the regular expression $(0+1)^+$. The set of initial states Θ, is given by the regular expression 10^*, indicating that the leftmost process holds the token. The transition relation ρ can be specified by the bi-regular expression $\left(\left[\begin{smallmatrix}0\\0\end{smallmatrix}\right] + \left[\begin{smallmatrix}1\\1\end{smallmatrix}\right]\right)^* \left[\begin{smallmatrix}1\\0\end{smallmatrix}\right] \left[\begin{smallmatrix}0\\1\end{smallmatrix}\right] \left(\left[\begin{smallmatrix}0\\0\end{smallmatrix}\right] + \left[\begin{smallmatrix}1\\1\end{smallmatrix}\right]\right)^*$ defined over $\Sigma = \{0, 1\}$. Alternatively, we can specify the transition relation by the length-preserving rewrite rule $x\ 1\ 0\ y \rightarrow x\ 0\ 1\ y$ (where x and y are arbitrary words over $\{0, 1\}^*$). The invariance property stating that there is exactly one token at all times, can be given by $\varphi = 0^*\ 1\ 0^*$.

2.4 Deterministic Pushdown Automata

Definition 3. A *Deterministic Pushdown Automaton* (DPDA) is a tuple $\langle \Sigma, S, s_0, \Gamma, \bot, \rho, F \rangle$ such that:

- Σ is an alphabet called the *input alphabet.*
- S is a finite set of *states.*
- $s_0 \in S$ is the *initial state.*
- Γ is an alphabet called the *stack alphabet.*
- $\bot \in \Gamma$ is a stack symbol called *stack bottom* (we assume that $\bot$ can be neither put nor removed from the stack).
- ρ is a partial function from $S \times \Sigma \times \Gamma$ to $S \times Com(\Gamma)$ called the *transition function* where $Com(\Gamma) \subseteq \Gamma^*$ is the set of *stack commands* which are strings that replace the symbol at the top of the stack.
- $F \subseteq S$ is the set of *accepting states.*

A *configuration* of a DPDA M is a triplet (s, w, γ) where $s \in S$, $w \in \Sigma^*$ and $\gamma \in \Gamma^+$. The configuration (s_2, w_2, γ_2) is defined to be an *immediate successor* of the configuration (s_1, w_1, γ_1), denoted $(s_1, w_1, \gamma_1) \vdash_M (s_2, w_2, \gamma_2)$, if there exists an input $\sigma \in \Sigma \cup \{\epsilon\}$, a stack symbol $z \in \Gamma$, and stack words $\gamma, \beta \in \Gamma^*$ such that $w_1 = \sigma w_2$, $\gamma_1 = \gamma z$, $\gamma_2 = \gamma\beta$, and $(s_2, \beta) \in \rho(s_1, \sigma, z)$. Thus on moving from configuration $(s_1, w_1, \gamma_1) = (s_1, \sigma w_2, \gamma z)$ to configuration $(s_2, w_2, \gamma_2) = (s_2, w_2, \gamma\beta)$, the DPDA consumes σ from w_1 and replaces z by β at the top of the stack. In this case we say that γ_2 is the result of applying β to γ_1 and denote $\gamma_2 = \beta(\gamma_1)$. When M is understood from the context, we omit the M subscript and simply write $(s_1, w_1, \gamma_1) \vdash (s_2, w_2, \gamma_2)$.

The *initial configuration* of a DPDA $M = (\Sigma, S, s_0, \Gamma, \bot, \rho, F)$ on a word w over Σ is $(s_0, w, \bot)$. We use $\vdash^*$ to denote the reflexive and transitive closure of $\vdash$. A *partial run* of M is a sequence of configurations $c_0, c_1, ..., c_m$ such that c_0 is the initial configuration, and c_{i+1} is an immediate successor of c_i, for each $i = 0, \ldots, m-1$. For a PDA $M = (\Sigma, S, s_0, \Gamma, \bot, \rho, F)$ we define $\mathcal{L}(M)$, the *language accepted by* M, to be $\Big\{w \ : \ (s_0, w, \bot) \vdash^* (t, \epsilon, \gamma) \text{ for some } t \text{ in } F \text{ and } \gamma \text{ in } \Gamma^*\Big\}$.

3 Adequacy of Classes of Automata for Symbolic Model Checking

Regular model checking is an important application of symbolic model checking which enables the verification of several classes of parameterized and infinite-state systems. Nevertheless, there are instances where a regular language is not expressive enough to describe the system or the property at hand.

Our aim is to find a class of languages that is more expressive than the regular languages, yet is adequate for symbolic model checking. Naturally, we wish for this class of languages to be as large as possible. However, the size of the class of languages that is still adequate is bounded by the requirements symbolic model checking compels.

The *rich-language symbolic model checking methodology* described in [KMM$^+$97] lists a set of minimal requirements from an assertional language, in order for it to be adequate for symbolic model checking. We reformulate these requirements here in terms of classes of languages rather than assertions. We classify the languages involved in the symbolic model checking procedures into different classes according to the basic operations they undergo. Our intention is to allow deployment of different classes of languages within the symbolic model checking process. We present *backward* and *forward model checking procedures* in terms of this classification, using only basic operations on languages.

Let $\mathcal{M}$, $\mathcal{R}$ and $\mathcal{A}$ be three classes of languages. Let $M_\varphi \in \mathcal{M}$ and $A_\varphi \in \mathcal{A}$ be languages adequate for specifying the property to be verified. Let $A_\Theta \in \mathcal{A}$ and $M_\Theta \in \mathcal{M}$ be languages adequate for representing the initial state of the system. Let $R_\rho \in \mathcal{R}$ be a bi-language adequate for representing the transition relation of the system, augmented by the identity relation (idle transition). We will use the auxiliary languages $M_0, M_1, M_2, ... \in \mathcal{M}$ to represent system states. The following procedures describe backward and forward model checking:

```
Procedure Backward MC
    $M_0 := \overline{M_\varphi}$
    For $i = 0, 1, \ldots$ repeat
        $M_{i+1} := ((\Sigma^* \times M_i) \cap R_\rho)\Downarrow_1$
    until $M_{i+1} = M_i$
return $M_i \cap A_\Theta = \emptyset$
```

```
Procedure Forward MC
    $M_0 := M_\Theta$
    For $i = 0, 1, \ldots$ repeat
        $M_{i+1} := ((M_i \times \Sigma^*) \cap R_\rho)\Downarrow_2$
    until $M_{i+1} = M_i$
return $M_i \cap \overline{A_\varphi} = \emptyset$
```

The classes $\mathcal{M}$, $\mathcal{R}$ and $\mathcal{A}$ are *adequate* for symbolic model checking, if the following requirements hold:

Procedure *Backward MC* is *applicable using the classes* $\mathcal{M}$, $\mathcal{R}$ and $\mathcal{A}$ if the following requirements hold:

1. $\mathcal{R}$ is adequate for representing ρ, and $\mathcal{M}$ and $\mathcal{A}$ are adequate for specifying φ and Θ respectively .
2. $\mathcal{M}$ is effectively closed under complementation.
3. $\mathcal{M}$ is effectively closed under lifting.
4. $\mathcal{M}$ is effectively closed under intersection with $\mathcal{R}$.
5. $\mathcal{M}$ is effectively closed under projection.
6. Either $\mathcal{M}$ is effectively closed under intersection with $\mathcal{A}$ and emptiness is effectively decidable for $\mathcal{M}$, or $\mathcal{A}$ is effectively closed under intersection with $\mathcal{M}$ and emptiness is effectively decidable for $\mathcal{A}$.
7. Equivalence of two languages in $\mathcal{M}$ is effectively decidable.

Procedure *Forward MC* is *applicable using the classes* $\mathcal{M}$, $\mathcal{R}$ and $\mathcal{A}$ if the following requirements hold:

1. $\mathcal{A}$ and $\mathcal{M}$ are adequate for specifying φ and Θ respectively, and $\mathcal{R}$ is adequate for representing ρ.
2. $\mathcal{A}$ is effectively closed under complementation.
3. Requirements $3 - 7$ above hold.

We say that the classes $\mathcal{M}$, $\mathcal{R}$ and $\mathcal{A}$ are *adequate* for symbolic model checking, if either procedure *Backward MC* is applicable using the classes $\mathcal{M}$, $\mathcal{R}$ and $\mathcal{A}$ or procedure *Forward MC* is applicable using the classes $\mathcal{M}$, $\mathcal{R}$ and $\mathcal{A}$.

Note that there is no requirement for closure under union on either of the classes, although [KMM$^+$97] requires closure under disjunction from the candidate assertional language. This is achieved by requiring that ρ always include the identity relation (corresponding to the stuttering step).

3.1 Meeting the Requirements

Clearly, taking $\mathcal{M}$, $\mathcal{R}$ and $\mathcal{A}$ to be the class of regular languages, $\mathcal{L}_{FA}$, meets all requirements, yielding regular model checking.

We can take one step further and choose the class of languages accepted by deterministic pushdown automata, $\mathcal{L}_{DPDA}$, for $\mathcal{A}$, leaving $\mathcal{M}$ and $\mathcal{R}$ to be the $\mathcal{L}_{FA}$ class. Since the $\mathcal{L}_{DPDA}$ class is closed under intersection with a regular language, and emptiness is decidable for $\mathcal{L}_{DPDA}$, all the requirements are met. Further, the fact that $\mathcal{L}_{DPDA}$ is closed under complementation enables us to perform both forward and backward model checking. The class of languages accepted by (any) pushdown automata, $\mathcal{L}_{PDA}$, is also closed under intersection with a regular language, and the emptiness question is decidable for it as well. Thus, choosing $\mathcal{L}_{PDA}$ for $\mathcal{A}$ is also acceptable. Because $\mathcal{L}_{PDA}$ is not closed under complementation, this choice restricts us to backward model checking[2]. In the following section we show how this seemingly slight change can help us to verify properties of algorithms that cannot be verified using regular model checking.

A more intricate step is considering $\mathcal{M}$ to be the $\mathcal{L}_{DPDA}$ class, leaving the $\mathcal{L}_{FA}$ class for $\mathcal{R}$ and $\mathcal{A}$. Requirement 3 is met as $\mathcal{L}_{DPDA}$ is closed under inverse homomorphism, which is a generalization of lifting. Requirements 2, 4 and 6 are met as $\mathcal{L}_{DPDA}$ is closed under complementation and intersection with a regular language, and the emptiness problem for $\mathcal{L}_{DPDA}$ is decidable. However, requirement 5 is not met: $\mathcal{L}_{DPDA}$ is not closed under projection. Requirement 7, decidability of the equivalence problem, was an open question until recently. In 1997 it was proven positively by Senizergues [Sen97]. However, the algorithm he provides is not very effective [Sti01].

We thus concentrate our efforts to find a sub-class of $\mathcal{L}_{DPDA}$ which is adequate for symbolic model checking. Therefore, we must find effective algorithms to compute projection and to decide equivalence for languages in this class. In addition, this class must satisfy the rest of the closure and decidability properties discussed above. This is the topic of Sections 5 and 6.

[2] If we can specify the negation of the property to be verified (directly) using a context-free language, then we can perform forward model checking as well.

4 Symbolic Model Checking using Context-free Languages and Regular Languages

In this section we concentrate on the framework of symbolic model checking where the class $\mathcal{A}$ is chosen to be the $\mathcal{L}_{PDA}$ class or the $\mathcal{L}_{DPDA}$ class, while $\mathcal{M}$ and $\mathcal{R}$ are the $\mathcal{L}_{FA}$ class. If we choose to apply backward model checking we can describe the property to be verified by some language in the $\mathcal{L}_{PDA}$ (or $\mathcal{L}_{DPDA}$) class, but then we have to compromise with a regular language for describing the initial condition. Alternatively, we can choose to apply forward model checking, which results in the initial condition being in $\mathcal{L}_{DPDA}$, while the property must be described by a regular language.

4.1 Peterson's Algorithm

We focus on the verification of the Peterson algorithm for mutual exclusion among N processes presented in the figure below. We show that this algorithm cannot be verified using regular model checking, yet it can be verified using the suggested framework.

```
                $N$ : **natural initially** $N > 1$
                $y$ : **array** $[1..N]$ **of** $[0..N-1]$ **initially** $y = 0$
                $s$ : **array** $[1..N-1]$ **of** $[1..N]$

                $t$ : **integer**
                $\ell_0$: **loop forever do**
                    $\ell_1$: **Non-Critical**
$\overset{N}{\underset{i=1}{||}} P[i]$ ::     $\ell_2$: **for** $t := 1$ **to** $N-1$ **do**
                        $\ell_3$: $(y[i], s[t]) := (t, i)$
                        $\ell_4$: **await** $s[t] \neq i \ \vee\ \forall j \neq i : y[j] < y[i]$
                    $\ell_5$: **Critical**
                    $\ell_6$: $y[i] := 0$
```

Program Peterson(N).

The Peterson algorithm can be explained as follows. Each process $P[i]$ has a priority variable $y[i]$. The range of the priority variable are the numbers from 0 to $N-1$. In addition there are $N-1$ signature variables $s[1], s[2], \ldots s[N-1]$. The domain of the signature variables consists of the processes indices $1, 2, ..., N$. The variable $s[i]$ holds the signature (index) of the last process that received priority i. Assume process $P[i]$ has priority j. In order to increment its priority (to $j+1$), it must be either the process with the highest priority, or not the last who signed in the signature $s[j]$. A process can enter the critical section if it has the highest priority $(N-1)$ and it is <u>not</u> the last who signed in (in the signature of the highest priority, $s[N-1]$). When a process exits the critical section its priority is reset.

4.2 An Encoding for Peterson

In order to apply regular model checking or our method, we must first model the system by languages as defined in Definition 2. We can encode a state of the Peterson system as a word of the form

$$\underbrace{\bigcirc\cdots\bigcirc}_{y=0} \mid \underbrace{\overset{s[1]}{\overset{\downarrow}{\bigcirc}}\cdots\bigcirc}_{y=1} \mid \underbrace{\overset{s[2]}{\overset{\downarrow}{\bigcirc}}\cdots\bigcirc}_{y=2} \mid \cdots \mid \underbrace{\overset{s[N-1]}{\overset{\downarrow}{\bigcirc}}\cdots\bigcirc}_{y=N-1} \mid \underbrace{\bigcirc\cdots\bigcirc}_{y=N-1}$$

where all processes $P[i]$ within partition $k = 0, 1, \ldots, N-1$ have their priority variable $y[i]$ set to k. The leftmost process in partition k is the one signed in the signature $s[k]$. Processes in the rightmost partition are the ones which are in the critical section. Note that there are $N+1$ partitions, separated by N border markers. The set of global states can thus be given by the language $\chi = \{w \in (\bigcirc + \mid)^* : \ \sharp(\mid, w) = \sharp(\bigcirc, w) > 1\}$.

This is the language over the alphabet $\{\bigcirc, \mid\}$, each of whose words has an equal number of "$\mid$" and "$\bigcirc$". The initial state of a concrete system consisting of N processes is described by the word $\bigcirc^N \mid^N$; the N processes have their priority variable y set to 0 and thus are located in the first partition. Hence, the set of initial states of the parameterized system can be given by the language $\Theta = \{\bigcirc^i \mid^i : \ i > 1\}$.

The transition relation ρ can be specified using five length-preserving rewrite rules: $\rho_1, \rho_2, \rho_3, \rho_4$ and ρ_{id} defined as follows:

$$\begin{array}{lll}
\rho_1 : & x \bigcirc \mid y \mapsto x \mid \bigcirc y & \text{where } x \in \bigcirc^*, y \in (\mid + \bigcirc)^* \\
\rho_2 : & x \mid \bigcirc \mid y \mapsto x \mid \mid \bigcirc y & \text{where } x \in (\mid + \bigcirc)^*, y \in \mid^* \\
\rho_3 : & x \bigcirc \bigcirc \mid y \mapsto x \bigcirc \mid \bigcirc y & \text{where } x, y \in (\mid + \bigcirc)^* \\
\rho_4 : & x \bigcirc \mapsto \bigcirc x & \text{where } x \in (\mid + \bigcirc)^* \\
\rho_{id} : & x \mapsto x & \text{where } x \in (\mid + \bigcirc)^*
\end{array}$$

The first rewrite rule states that a process can move unconditionally from the first partition to the second partition. This corresponds to increasing the priority variable y from 0 to 1. The second rewrite rule states that a single process within a partition can move to the next partition provided that all the partitions to the right are empty. This corresponds to the situation where the process has the highest priority. The third rewrite rule transition states that a process can move to the next partition provided that there is a process to its left in the same partition. This corresponds to the situation where the process is not the last to sign in the signature corresponding to its current priority. The fourth rewrite rule describes exiting from the critical section and the last rewrite rule describes the stuttering step.

4.3 Another Encoding for Peterson

The encoding described above is not regular, since languages χ and Θ are non-regular. It is possible to model the Peterson algorithm using regular languages as follows. Define the alphabet $\Sigma = \{0, 1, 2\}$. Each priority queue is represented by exactly one alphabet letter. An empty partition is represented by 0, a partition with exactly one process is represented by 1 and a partition with two or more processes is represented by 2. A state of a concrete system with N processes is thus of length $N + 1$. Assuming there are at least two processes, we require a state to be of length at least 3 and to have at least one partition with two or more processes or two partitions with one process each. Thus, the set of global states can be given as follows:

$$\chi = \{uavbw \ : \ u, v, w \in \Sigma^*, \quad a, b \in \Sigma, \quad |uavbw| \geq 3 \text{ and } a + b \geq 2\}.$$

The set of initial states can be specified by the language $\Theta = \{20^i \ : \ i \geq 2\}$. The transition relation, using this encoding, consists of many rewrite rules. We classified them to several groups corresponding to the first encoding.

$$\rho_1 : \begin{cases} 2\ 0\ y \mapsto 2\ 1\ y \mid 1\ 1\ y \\ 2\ 1\ y \mapsto 2\ 2\ y \mid 1\ 2\ y \\ 1\ 0\ y \mapsto 0\ 1\ y \\ 1\ 1\ y \mapsto 0\ 2\ y \\ 1\ 2\ y \mapsto 0\ 2\ y \end{cases}$$

$$\rho_2 : x\ 1\ 0\ y \mapsto x\ 0\ 1\ y \text{ where } Y \in 0^*$$

$$\rho_3 : \begin{cases} x\ 2\ 0\ y \mapsto x\ 1\ 1\ y \mid x\ 2\ 1\ y \\ x\ 2\ 1\ y \mapsto x\ 1\ 2\ y \mid x\ 2\ 2\ y \\ x\ 2\ 2\ y \mapsto x\ 1\ 2\ y \end{cases}$$

$$\rho_4 : \begin{cases} 0\ x\ 2 \mapsto 1\ x\ 1 \mid 1\ x\ 2 \\ 0\ x\ 1 \mapsto 1\ x\ 0 \\ 1\ x\ 2 \mapsto 2\ x\ 1 \mid 2\ x\ 2 \\ 1\ x\ 1 \mapsto 2\ x\ 0 \\ 2\ x\ 1 \mapsto 2\ x\ 0 \\ 2\ x\ 2 \mapsto 2\ x\ 1 \mid 2\ x\ 2 \end{cases}$$

$$\rho_{id} : x \mapsto x$$

4.4 Verification of Peterson's Algorithm

We begin by trying to apply regular model checking, using the second encoding. We have calculated the set of reachable states using the tool TLVP [Sha96,PS00]. The tool offers several acceleration techniques, and we used local acceleration and binary left-to-right acceleration. The computed set of reachable states ψ can be given by $\chi \cap (\Theta \cup L_1 \cup L_2)$ where $L_1 = \{au2v \ : \ a \in \{0, 1, 2\} \ \ u \in \{1, 2\}^* \text{ and } v \in 0^*\}$ and $L_2 = \{auv1w \ : \ a \in \{0, 1, 2\} \ \ u \in \{1, 2\}^* \text{ and } v, w \in 0^*\}$.

We are interested in verifying the property of mutual exclusion. Using the suggested encoding, we can describe the negation of mutual exclusion as follows: $\overline{\varphi} = \Sigma^* 2$ indicating that there are two or more processes in the critical section. There is a nonempty intersection between the set of reachable states ψ and the negation of mutual exclusion $\overline{\varphi}$. It is given by $\psi \cap \overline{\varphi} = \{au2 \ : \ a \in \{0, 1, 2\} \ \ u \in \{1, 2\}^*\}$. Thus, we were unable to verify the mutual exclusion property for Peterson(N), using regular model checking.

However, if we recruit the context-free languages to aid us we can win the task. We consider the first encoding suggested for Peterson. We compute the set of reachable states, starting with an over-approximation of the set of initial states: $\bigcirc^{+}|^{+}$. The set of reachable states ψ is specified using this encoding as $\psi = \chi \cap (\Theta \cup L_1 \cup L_2)$ where $L_1 = \bigcirc^{*}|\ (\bigcirc\ \bigcirc^{*}|)^{*}\ \bigcirc\ \bigcirc\ \bigcirc^{*}\ (\epsilon + |\ |^{*})$ and $L_2 = \bigcirc^{*}|\ (\bigcirc\ \bigcirc^{*}|)^{*}\ |^{*}\ (\bigcirc\ |\ |^{*} + \bigcirc)$. The negation of the property of mutual exclusion is now given by $\overline{\varphi} = (\bigcirc + |)^{*}\ \bigcirc\ \bigcirc$. Now, the intersection between the set of reachable states ψ and the negation of the property $\overline{\varphi}$ is empty. This is due to the context free language χ. Thus, using this method we have proved, what we failed to do using regular model checking, that Peterson(N) satisfied mutual exclusion.

5 Symbolic Model Checking using a Cascade Product 1DPDA ∘ DFA

In this section we seek to find a subset of $\mathcal{L}_{DPDA}$ (a superset of $\mathcal{L}_{FA}$) which is adequate for symbolic model checking. Given a single state DPDA M we define the class $\mathcal{L}_{DPDA-M}$. We provide a semi-algorithm for computing projection for this class, and an efficient algorithm to decide equivalence. This class, in addition, is closed under all basic operations required in order to be adequate for symbolic model checking. By this we provide the missing link, whose absence caused the full $\mathcal{L}_{DPDA}$ class to be inadequate.

5.1 The Classes $\mathcal{L}_{DPDA-M}$

We concentrate on DPDAs as defined in the preliminaries. We often consider single-state DPDAs with an empty set of accepting conditions. We refer to a single-state DPDA with an empty set of accepting states as a 1DPDA. A 1DPDA $M = \langle \Sigma, \{q\}, q, \Gamma, \bot, \Delta, \emptyset \rangle$ can be represented by the quadruple $\langle \Sigma, \Gamma, \bot, \widehat{\Delta} \rangle$ where $\widehat{\Delta} : \Sigma \times \Gamma \rightarrow Com(\Gamma)$. The following definitions are needed for the sequel.

Definition 4. (*cascade product*)
Let $R = \langle V \times \Gamma, S, s_0, \delta, F \rangle$ be a DFA and $\phi : V \rightarrow \Sigma$ a substitution mapping each letter of V to a letter in Σ. The *cascade product* $M \circ_{\phi} R$ is the DPDA $\langle V, S, s_0, \Gamma, \bot, \rho, F \rangle$ where $\rho(s, \sigma, z) = (\delta(s, \langle \sigma, z \rangle), \Delta(\phi(\sigma), z))$.

Definition 5. (*stack-consistent with M*)
A DPDA A is said to be *stack-consistent with M* (*M-consistent* for short), if there exists a substitution $\phi : V \rightarrow \Sigma$ and a DFA $R = \langle V \times \Gamma, S, s_0, \delta, F \rangle$ such that $A = M \circ_{\phi} R$.

Let $A = M \circ_{\phi} R$. A run π_A of A on the word $w = \sigma_1 \sigma_2 \cdots \sigma_n$ can be decomposed into two runs: a run π_M of M on $\phi(w)$ and a run π_R of R on w as depicted below.

$$\begin{array}{lccccccc}
\pi_A : & (s_0, \gamma_0) & \stackrel{\sigma_1/x_1}{\longmapsto} & (s_1, \gamma_1) & \stackrel{\sigma_2/x_2}{\longmapsto} & \ldots & \stackrel{\sigma_n/x_n}{\longmapsto} & (s_n, \gamma_n) \\
\pi_M : & \gamma_0 & \stackrel{\phi(\sigma_1)/x_1}{\longmapsto} & \gamma_1 & \stackrel{\phi(\sigma_2)/x_2}{\longmapsto} & \ldots & \stackrel{\phi(\sigma_n)/x_n}{\longmapsto} & \gamma_n \\
\pi_R : & s_0 & \stackrel{\sigma_1, \top(\gamma_0)}{\longmapsto} & s_1 & \stackrel{\sigma_2, \top(\gamma_1)}{\longmapsto} & \ldots & \stackrel{\sigma_n, \top(\gamma_{n-1})}{\longmapsto} & s_n
\end{array}$$

The decision upon the stack command at the i-th step x_i is governed only by the 1DPDA M, which is not aware of the state s_i of R. Yet, the DFA R can look at the top symbol on the stack of M (we use $\top(\gamma)$ to denote the stack symbol at the top of the stack content γ). Automata R and M move simultaneously: when A reads a letter $\sigma_i \in V$, the DFA R makes a move as if it was reading the pair $(\sigma_i, \top(\gamma_{i-1}))$, where γ_{i-1} is the current stack-content of M, while M makes a move as if it was reading $\phi(\sigma_i)$. The automaton A decides to accept a word w if, when the run terminates, R is in state $s_n \in F$.

Therefore, an automaton $A = \langle V, S, s_0, \Gamma, \bot, \rho, F\rangle$ which is M-consistent with respect to ϕ can be characterized by the pair $\langle R, \phi\rangle$ where R is the DFA such that $A = M \circ_\phi R$.

Definition 6. (*The M-stack consistent class, $\mathcal{L}_{DPDA-M}$*)
Given a 1DPDA M, we define the M-consistent class $\mathcal{L}_{DPDA-M}$ to be the class of languages that are accepted by some DPDA which is M-consistent with respect to some substitution ϕ.

The M-consistent automaton A can be viewed as a case in which the automaton has been decomposed into a stack-manipulator, which behaves exactly like M in its decisions about the stack transformations, and a finite-state controller R which affects the selection of the next state. Two M-consistent automata A_1 and A_2 share the same stack-manipulator and may differ in their respective finite-state controller. Note that the substitution ϕ may differ between two automata A_1 and A_2 in the class.

Claim. The class $\mathcal{L}_{DPDA-M}$ is *effectively closed* under complementation, lifting and intersection with a regular language.

Claim. Equivalence and Emptiness are *effectively decidable* for the class $\mathcal{L}_{DPDA-M}$.

5.2 Computing Projection

Viewing the claims above, for any 1DPDA M the class $\mathcal{L}_{DPDA-M}$ satisfies requirements 1,2,3,4,6 and 7. Thus, if the class is also effectively closed under projection, then it is adequate for symbolic model checking (in collaboration with the $\mathcal{L}_{FA}$ class). In the following we provide a semi-algorithm for computing projection.

Let A be a DPDA which is M-consistent with respect to ϕ, and let R be A's characteristic DFA. For simplification we assume the input alphabet of A is $\Sigma \times \Sigma$, where Σ is the input alphabet of M. Also, we assume A is M-consistent with respect to $\phi = \Downarrow_2$ and we wish to calculate the projection of $\mathcal{L}(A)$ on its first coordinate. That is, we would like to compute another DPDA $\widetilde{A}$ over the alphabet Σ which is M-consistent with respect to the identity relation and accepts the projection of A on its first coordinate.

We claim that procedure *Project* described below calculates the correct projection (unless it aborts). The procedure can be explained as follows. To simplify notations, we present ρ as a relation, a subset of $S \times \Sigma \times \Gamma \times Com(\Gamma) \times S$ instead of a function from $S \times \Sigma \times \Gamma$ to $S \times Com(\Gamma)$.

```
Procedure Project
   Input:   a DPDA $A = \langle \Sigma \times \Sigma, S, s_0, \Gamma, \perp, \rho, F \rangle$, a positive integer $k$
   Output: a DPDA $\widetilde{A} = \langle \Sigma, \widetilde{S}, \widetilde{s_0}, \Gamma, \perp, \widetilde{\rho}, \widetilde{F} \rangle$
1. Annotation:
      $\widehat{A} = \langle \Sigma \times \Sigma, \widehat{S}, \widehat{s_0}, \Gamma, \perp, \widehat{\rho}, \widehat{F} \rangle := Annotate(A, k)$
2. Projection:
      $\ddot{A} := \langle \Sigma, \ddot{S}, \ddot{s_0}, \Gamma, \perp, \ddot{\rho}, \ddot{F} \rangle$ where $\ddot{S} = \widehat{S}$, $\ddot{s_0} = \widehat{s_0}$, $\ddot{F} = \widehat{F}$ and
         $\ddot{\rho} = \{(s, z_1, \sigma_1, x_1, s') \ : \ \exists z_2, \sigma_2, x_2$ such that $(s, z_1, z_2, \sigma_1, \sigma_2, x_1, x_2, s') \in \widehat{\rho}\}$
3. Determinization:
      $\ddot{R} := \langle \Sigma \times \Gamma, \ddot{S}, \ddot{s_0}, \ddot{\delta}, \ddot{F} \rangle$ where $\ddot{\delta}(s, (\sigma, z)) = s' \iff \exists x : \ (s, z, \sigma, x, s') \in \ddot{\rho}$
      $\widetilde{R} = \langle \Sigma \times \Gamma, \widetilde{S}, \widetilde{s}, \widetilde{\delta}, \widetilde{F} \rangle := Subset\_Construction(\ddot{R})$
Return $\widetilde{A} := M \circ_{id} \widetilde{R}$
```

In Phase 1 procedure *Project* calls procedure *Annotate* described below. Procedure *Annotate* guesses how the 1DPDA M will operate when looking at the first coordinate instead of the second. For each edge (s, s') labeled by $(\left[\begin{smallmatrix}\sigma_1\\ \sigma_2\end{smallmatrix}\right], z_2, x_2)$[3] the procedure aims to find all possible stack letters z_1 and adequate stack commands x_1 such that if M looks at the first coordinate then, when moving from state s to s', it will have z_1 on the top of the stack and decide on the stack command x_1.

For each state, it saves an information describing the difference between the actual stack of M and the "guessed stack" (the stack of M if it was looking at the first coordinate). For this it uses the notation $\langle \beta_1, \beta_2 \rangle$ with the intention that if the maximal common prefix of the actual stack and the guessed stack is w then the "guessed stack" is described by $w\beta_1$ and the actual stack is described by $w\beta_2$. An original state s of A may appear more than once in $\widehat{A}$, each time labeled with a different notation $\langle \beta_1, \beta_2 \rangle$.

Note that z_2 must be the top symbol of $w\beta_2$. The procedure will choose z_1 to be the top symbol of $w\beta_1$. The stack command x_1 is then determined by $\Delta(\sigma_1, z_1)$. Considering that the difference between the guessed stack and actual stack in state s is described by $\langle \beta_1, \beta_2 \rangle$, and the stack commands for the guessed and actual stack are x_1 and x_2 respectively, we can compute the new difference $\langle \beta_1', \beta_2' \rangle$ for state s'. Let γ_1 and γ_2 be the result of applying x_1 and x_2 to the stack contents $w\beta_1$ and $w\beta_2$ respectively. Given w' is the maximal common prefix of γ_1 and γ_2, then $\beta_1' = \gamma_1 / w'$ and $\beta_2' = \gamma_2 / w'$ (where u/v denotes the right division of u by r).

[3] If $(s', x) \in \rho(s, \sigma, z)$ we say that the edge (s, s') is labeled by (σ, z, x).

```
Procedure Annotate
   Input:   $A = \langle \Sigma \times \Sigma, S, s_0, \Gamma, \perp, \rho, F\rangle$, a positive integer $k$
   Output: $\widehat{A} = \langle \Sigma \times \Sigma, \widehat{S}, \widehat{s_0}, \Gamma, \perp, \widehat{\rho}, \widehat{F}\rangle$ where $\widehat{S} \subseteq S \times (\Gamma^{\leq k} \times \Gamma^{\leq k})$ and
              $\widehat{\rho} \subseteq S \times \Sigma^2 \times \Gamma^2 \times Com(\Gamma)^2 \times S$.
   $\widehat{s_0} := (s_0, \langle \epsilon, \epsilon\rangle)$
   $f\mathsf{reg}(\widehat{s_0}) := Compute\_Stack\_Language(A, s_0)$
   $\widehat{\rho} := \emptyset$
   $\widehat{S} := \{\widehat{s_0}\}$
   $Q := \{\widehat{s_0}\}$
   While $Q \neq \emptyset$
      Pick $\widehat{s} = (s, \langle \beta_1, \beta_2\rangle) \in Q$
      For all $(r, r', z_1, z_2, x_1, x_2, \sigma_1, \sigma_2, \beta_1, \beta_2, \beta_1', \beta_2') \in ReachableTransitions$ s.t.
      $(s, \sigma_1, \sigma_2, z_2, x_2, s') \in \rho$ and $r = f\mathsf{reg}(s)$ repeat
         If $|\beta_1'| \geq k \ \vee \ |\beta_2'| \geq k$ then abort
         $\widehat{s}' := (s', \langle \beta_1', \beta_2'\rangle)$
         If $\widehat{s}' \notin \widehat{S}$   then $f\mathsf{reg}(\widehat{s}') := r'$   else $f\mathsf{reg}(\widehat{s}') := f\mathsf{reg}(\widehat{s}') \cup r'$
         $\widehat{\rho} := \widehat{\rho} \cup \{(\widehat{s}, z_1, z_2, \sigma_1, \sigma_2, x_1, x_2, \widehat{s}')\}$
         $\widehat{S} := \widehat{S} \cup \{\widehat{s}'\}$
         $Q := Q \cup \{\widehat{s}'\}$
      end for all
      $Q := Q \setminus \{\widehat{s}\}$
   end while
   $\widehat{s_0} := \{(s, \langle \beta_1, \beta_2\rangle) \ : \ s = s_0\}$
   $\widehat{F} := \{(s, \langle \beta_1, \beta_2\rangle) \ : \ s \in F\}$
end procedure
```

To guarantee termination, the procedure uses the second parameter – a bound $k \in N$. The procedure aborts if the length of either β_1 or β_2 exceeds k. It may be the case that the procedure decides to abort when exploring an *unreachable* edge. We say that an edge (s, s') labeled by $(\left[\begin{smallmatrix}\sigma_1\\ \sigma_2\end{smallmatrix}\right], z_2, x_2)$ where s is annotated $\langle \beta_1, \beta_2\rangle$ is unreachable if there is no prefix w such that $w\beta_2$ is in the reachable stack language of s. To avoid this situation, the procedure computes for each state s the reachable stack language of s. It uses a labeling function $f\mathsf{reg}$ that labels each state with its reachable stack language, $\mathcal{L}(s)$. When exploring the edge (s, s') the procedure checks that $w\beta_2$ is in the reachable stack language of s, given by $r = f\mathsf{reg}(s)$. The procedure updates the reachable stack language of s', $f\mathsf{reg}(s')$, to contain the set of words obtained by applying x_2 to a word in r whose top symbol is z_2.

The complete relation between all involved components, is summarized as follows. Let $z_1, z_2 \in \Gamma$, $\sigma_1, \sigma_2 \in \Sigma$, $\beta_1, \beta_2, \beta_1', \beta_2' \in \Gamma^*$, $x_1, x_2 \in Com(\Gamma)$, $r, r' \in \mathcal{R}(\Gamma)$. $(r, r', z_1, z_2, x_1, x_2, \sigma_1, \sigma_2, \beta_1, \beta_2, \beta_1', \beta_2') \in ReachableTransitions$ if and only if the following holds:

1. $\exists w \in \Gamma^*$ such that $w\beta_2 \in r$

2. $z_1 = \top(w\beta_1)$ and $z_2 = \top(w\beta_2)$
3. $x_1 = \Delta(\sigma_1, z_1)$ and $x_2 = \Delta(\sigma_2, z_2)$
4. $\beta'_1 = x_1(w\beta_1)/w'$ and $\beta'_2 = x_2(w\beta_2)/w'$
   where $w' = \mathit{max\_common\_prefix}(x_1(w\beta_1), x_2(w\beta_2))$
5. $r' = \{x_2(uz_2) \mid uz_2 \in r\}$

The search is conducted on-the-fly starting at the initial state. For the initial state both the actual stack and guessed stack are $\perp$ and hence the initial state is annotated by $\langle\epsilon,\epsilon\rangle$. The reachable stack language of the initial state is computed by calling procedure *Compute_Stack_Languages* (see for example [FWW97]). Then, for each state s labeled $f\mathsf{reg}(s) = r$ and annotated by $\langle\beta_1, \beta_2\rangle$, and for each out-going edge (s, s') labeled by $(\left[\begin{smallmatrix}\sigma_1\\ \sigma_2\end{smallmatrix}\right], z_2, x_2)$, it annotates the state s' by $\langle\beta'_1, \beta'_2\rangle$, updates $f\mathsf{reg}(s')$ to contain r' and adds the annotation z_1, x_1 to the edge, where $(r, r', z_1, z_2, x_1, x_2, \sigma_1, \sigma_2, \beta_1, \beta_2, \beta'_1, \beta'_2)$ satisfy the conditions of a reachable transition.

In Phase 2 procedure *Project* simply projects each edge on the first coordinate. From each edge on it eliminates the second component σ_2 of the bi-letter $\left[\begin{smallmatrix}\sigma_1\\ \sigma_2\end{smallmatrix}\right]$, the stack letter z_2 and the stack command x_2.

In Phase 3, as a first step the procedure extracts from the PDA $\ddot{A}$ the finite-state automaton $\ddot{R}$. Note that $\ddot{R}$ is non-deterministic (i.e. it is an NFA): from one state s there could be two outgoing edges (s, s') and (s, s'') with the same label. This non-determinism is the result of the projection performed in the previous phase. Hence, as a second step, the algorithm applies the subset construction and obtains the DFA $\widetilde{R}$.

6 Symbolic Model Checking using a Product PDA × DFA

In the previous section, we saw that there is a semi-algorithm to compute projection of a language given as a cascade product of a single-state DPDA and a DFA. We now consider the case where the cascade product is degenerate, i.e. the DFA does not look at the stack of the 1DPDA. In this case the language is given by a simple product of a DPDA and a DFA. We show that if the initial state of a system can be given by such a language, and the language is in some sense "preserved" by the transition relation of the system, then projection can be computed on the DFA part alone.

Definition 7. We say that a language L is *left preserved* by a bi-language R if $((L \times \Sigma^*) \cap R)\Downarrow_2 = L$. We say that a language L is *right preserved* by a bi-language R if $((\Sigma^* \times L) \cap R)\Downarrow_1 = L$.

Note that in order to check that a language given by a cascade product is right or left preserved, one can compute the projection using the procedure presented in the previous section.

Claim. Let $\langle D, \varphi\rangle$ be a verification problem where $D = (\Sigma, \chi, \Theta, \rho)$.

- If $\Theta = L \cap R_0$ and L is left-preserved by ρ and ρ is regular, then the verification problem $\langle D, \varphi \rangle$ can be solved by applying the procedure for forward model checking given below.
- If $\neg\varphi = L \cap R_0$ and L is right-preserved by ρ and ρ is regular, then the verification problem $\langle D, \varphi \rangle$ can be solved by applying the procedure for backward model checking given below.

```
Procedure Backward MC
  $M_0 := L \cap R_0$
  For $i = 0, 1, \ldots$ repeat
    $M_{i+1} := ((\Sigma^* \times R_i) \cap R_\rho)\Downarrow_1 \cap L$
  until $M_{i+1} = M_i$
return  $M_i \cap A_\Theta = \emptyset$
```

```
Procedure Forward MC
  $M_0 := L \cap R_0$
  For $i = 0, 1, \ldots$ repeat
    $M_{i+1} := L \cap ((R_i \times \Sigma^*) \cap R_\rho)\Downarrow_2$
  until $M_{i+1} = M_i$
return  $M_i \cap \overline{A_\varphi} = \emptyset$
```

Note that applying symbolic model checking using this method, has another advantage when considering the convergence problem. Any technique developed for regular model checking can be applied here as well. Since the DFA part is separated, one can apply the technique on the DFA part alone, obtaining necessarily correct results.

Peterson Example It can easily be seen that Peterson's algorithm, when modeled using the first encoding, fits into this frame. Recall that the set of global states is given by $\chi = \{w \;:\; \sharp(\,|\,, w) = \sharp(\bigcirc, w) > 1\}$. The set of initial states specified before by $\Theta = \{\bigcirc^i\,|^i \;:\; i > 1\}$ can be given by $\chi \cap \bigcirc^+|^+$.

Recall that the transition relation ρ is specified using the five length-preserving rewrite rules below:

$\rho_1 \;:\; x\bigcirc\,|\,y \mapsto x\,|\,\bigcirc y$	where $x \in \bigcirc^*, y \in (\,|\,+\bigcirc)^*$
$\rho_2 \;:\; x\,|\,\bigcirc\,|\,y \mapsto x\,|\,|\,\bigcirc y$	where $x \in (\,|\,+\bigcirc)^*, y \in\, |^*$
$\rho_3 \;:\; x\bigcirc\bigcirc\,|\,y \mapsto x\bigcirc\,|\,\bigcirc y$	where $x, y \in (\,|\,+\bigcirc)^*$
$\rho_4 \;:\; x\bigcirc \mapsto \bigcirc x$	where $x \in (\,|\,+\bigcirc)^*$
$\rho_{id} \;:\; x \mapsto x$	where $x \in (\,|\,+\bigcirc)^*$

Since in all rewrite rules, the number of sticks and stones does not change between the left hand side and right hand side of the rule, χ is right and left preserved by ρ. Hence, by Claim 6, we can apply forward model checking when projection is performed only on the transition relation.

7 Conclusions and Future Work

Our research tried to give an answer to the verification problem of parameterized system, which cannot be answered using regular model checking.

First, we showed that we can solve the verification problem of parameterized systems whose initial condition can be specified by a context free language, while all other components can be specified by regular languages.

Second, we defined a class, which is a subset of the $\mathcal{L}_{DPDA}$ class, and showed that it is adequate for symbolic model checking. We obtained this by giving a semi-algorithm for computing the projection and an efficient algorithm for deciding equivalence. By this we provided the missing link, whose absence caused the full $\mathcal{L}_{DPDA}$ class to be inadequate.

Third, we focused on a special case of the class we have defined. We gave a simple algorithm to compute projection for this class. In addition, we showed that all acceleration and widening techniques developed in order to achieve convergence of the regular model checking procedure, can be applied to this class. Thus we have shown that the class is *practically* adequate for symbolic model checking.

We showed that the methods we have developed can be used for verifying non-regular parameterized systems. All suggested methods have been successfully used to prove mutual exclusion of the Peterson algorithm for an arbitrary number of processes [Pet81] and termination of a termination detection algorithm extracted from Ricart and Agrawalas' algorithm [RA81].

Future Work It is left to study the relation ship between these methods. Is one stronger than the other or are they incomparable, and one works on some instances while the second works on other.

In this paper we consider only safety properties. It is important to extend our methods to verify liveness properties as well.

In order to apply regular model checking or any of the methods we have developed here to a given verification problem, the verification problem must be modeled by languages and AI-languages. In several cases, in order to model the system, we used an encoding which is an abstraction of the system. It is interesting to see if one can automatically verify the correctness of such an abstraction, or even more, automatically produce a correct encoding (abstraction).

When dealing with regular languages one can alternate between the different, yet equivalently expressive, formalisms FA, WS1S and regular expressions (as done in [JJK94] for example). It will be useful if we had an assertional language as expressive as our restricted class, because it would allow convenient alternation between representations according to the need. Such a direction of investigation may be very interesting.

The class of languages introduced in this paper, similar to regular expressions, allows many automatic manipulations. It is interesting to explore if this class can be useful in the deductive realm as well.

Much like the class introduced here, it is interesting to see if there are other classes, which are more expressive than regular expressions, and yet are adequate for symbolic model checking.

A Proofs

Proofs of Subsection 5.1

Claim. The class $\mathcal{L}_{DPDA-M}$ is effectively closed under complementation.

Proof. Let A be an automaton in $\mathcal{L}_{DPDA-M}$, characterized by $\langle R, \phi \rangle$ where $R = (V \times \Gamma, S, s_0, \delta, F)$. We claim that the automaton $\overline{A}$ characterized by $\langle R^c, \phi \rangle$ where $\overline{R} = (V \times \Gamma, S, s_0, \delta, S \setminus F)$ accepts the complemented language of A. We note that since R and $\overline{R}$ differ only in the set of accepting states, any run of A is a run of $\overline{A}$ and vice versa. By definition of $\overline{F} = S \setminus F$ a state $s \in \overline{F}$ if and only if $s \notin F$. Therefore whenever the automaton A accepts a run π, the automaton $\overline{A}$ rejects it and vice versa. □

Claim. The class $\mathcal{L}_{DPDA-M}$ is effectively closed under lifting.

Proof. Let A be an automaton in the class $\mathcal{L}_{DPDA-M}$, characterized by $\langle R, \phi \rangle$. To simplify notations, we focus on the trivial case where $V = \Sigma$, thus, the input alphabet for the DFA R is $\Sigma \times \Gamma$ and the substitution ϕ is the identity. We construct the proof for the case of left-lifting, the case of right-lifting is symmetric. Let $R = (\Sigma \times \Gamma, S, s_0, \delta, F)$. Define a DFA $R^{\mathsf{lifting}} = (\Sigma \times \Sigma \times \Gamma, S, s_0, \delta^{\mathsf{lifting}}, F)$ where for all $\sigma_2 \in \Sigma$, $\delta^{\mathsf{lifting}}(s, \langle \left[\begin{smallmatrix}\sigma_1\\ \sigma_2\end{smallmatrix}\right], z \rangle) = \delta(s, \langle \sigma_1, z \rangle)$. We claim that the automaton A^{lifting} characterized by $\langle R^{\mathsf{lifting}}, \Downarrow_1 \rangle$ accepts the left-lifting of $\mathcal{L}(A)$.

$$\begin{aligned}
\left[\begin{smallmatrix}w_1\\ w_2\end{smallmatrix}\right] \in \mathcal{L}(A) \times \Sigma^* &\iff w_1 \in \mathcal{L}(A)\\
&\iff \text{the run } \pi_A \text{ of } A \text{ on } w_1 \text{ is accepting}\\
&\iff \text{the run } \pi_R \text{ of the DFA } R \text{ on } \langle w_1, \gamma \rangle \text{ ends in state } s \in F\\
&\qquad \text{where } \gamma \text{ is the sequence of stack tops observed in the}\\
&\qquad \text{run } \pi_M \text{ of } M \text{ on } w_1.\\
&\iff \text{the run } \pi_R \text{ of } R^{\mathsf{lifting}} \text{ on } \langle \left[\begin{smallmatrix}w_1\\ w_2\end{smallmatrix}\right], \gamma \rangle \text{ ends in state } s \in F\\
&\qquad \text{where } \gamma \text{ is the sequence of stack tops observed in the}\\
&\qquad \text{run } \pi_M \text{ of } M \text{ on } \left[\begin{smallmatrix}w_1\\ w_2\end{smallmatrix}\right]\Downarrow_1 = w_1.\\
&\iff \left[\begin{smallmatrix}w_1\\ w_2\end{smallmatrix}\right] \in \mathcal{L}(A^{\mathsf{lifting}})
\end{aligned}$$

□

Claim. The class $\mathcal{L}_{DPDA-M}$ is effectively closed under intersection with a regular language.

Proof. Let A be an automaton in the class $\mathcal{L}_{DPDA-M}$, characterized by $\langle R, \phi \rangle$ where $R = (V \times \Gamma, S, s_o, \delta, F)$. Let $r = (V \times \Gamma, S_r, s_0^r, \delta_r, F_r)$ be a DFA. We define the automaton $A^{\cap r}$, a member of $\mathcal{L}_{DPDA-M}$ as follows. $A^{\cap r} = \langle R^{\cap r}, \phi \rangle$ where $R^{\cap r} = (V \times \Gamma, S \times S_r, (s_0, s_0^r), \delta^{\cap r}, F \times F_r)$ and $\delta^{\cap r}((s, s_r), \langle \sigma, z \rangle) = (\delta(s, \langle \sigma, z \rangle), \delta_r(s_r, \langle \sigma, z \rangle))$. It can easily be seen that $w \in \mathcal{L}(A) \cap \mathcal{L}(r)$ if and only if $w \in \mathcal{L}(A^{\cap r})$. □

Claim. Let A_1, A_2 be two DPDAs over the alphabet V which are stack-consistent with a 1DPDA M, with the same substitution ϕ. The question whether $\mathcal{L}(A_1) = \mathcal{L}(A_2)$ is effectively decidable.

Proof. The proof is derived from the decidability of the equivalence problem for general deterministic pushdown automata [Sen97]. However, in this case, a much shorter proof and a more efficient algorithm can be given. We sketch it as follows: Assume A_1 and A_2 are characterized by $\langle R_1, \phi\rangle$ and $\langle R_2, \phi\rangle$ respectively. $\mathcal{L}(A_1) = \mathcal{L}(A_2)$ if and only if $\mathcal{L}(M \circ (R_1 \cap R_2^c)) = \emptyset$ and $\mathcal{L}(M \circ (R_1^c \cap R_2)) = \emptyset$. Because emptiness is effectively decidable for DPDA we are done. □

Claim. Let A be a DPDA which is M-consistent. The question whether $\mathcal{L}(A) = \emptyset$ is effectively decidable.

Proof. The proof is derived from the decidability of the emptiness problem for general pushdown automata. □

Proofs of Subsection 5.2

Theorem 1. *Let A be a 1DPDA which is M-consistent with respect to $\phi = \Downarrow_2$. Assuming procedure Project did not abort when constructing $\widetilde{A}$, then $\mathcal{L}(\widetilde{A}) = \mathcal{L}(A)\Downarrow_1$.*

Lemma 1. (*Correctness of procedure Annotate*)
Let $u_1 = \sigma_1^1\sigma_2^1 \ldots \sigma_n^1$ and $u_2 = \sigma_1^2\sigma_2^2 \ldots \sigma_n^2$. Let π_A and π_M be runs of A and M on $\left[\begin{smallmatrix}u_1\\u_2\end{smallmatrix}\right]$ and u_1 respectively as follows.

$$\begin{array}{llclclcll}
\pi_A : & (s_0, \gamma_0^2) & \xmapsto{\left[\begin{smallmatrix}\sigma_1^1\\\sigma_1^2\end{smallmatrix}\right]/x_1^2} & (s_1, \gamma_1^2) & \xmapsto{\left[\begin{smallmatrix}\sigma_2^1\\\sigma_2^2\end{smallmatrix}\right]/x_2^2} & \ldots & \xmapsto{\left[\begin{smallmatrix}\sigma_n^2\\\sigma_n^2\end{smallmatrix}\right]/x_n^2} & (s_n, \gamma_n^2) \\
\pi_M : & \gamma_0^1 & \xmapsto{\sigma_1^1/x_1^1} & \gamma_1^1 & \xmapsto{\sigma_2^1/x_2^1} & \ldots & \xmapsto{\sigma_n^1/x_n^1} & \gamma_n^1
\end{array}$$

For all $0 \le i \le n$, let w_i be the maximal common prefix of γ_i^1 and γ_i^2. Let $\beta_i^1 = \gamma_i^1/w_i$ and $\beta_i^2 = \gamma_i^2/w_i$. Denote by m the maximal i such that $|\beta_i^1| \le k$ and $|\beta_i^2| \le k$.
Then, $\widehat{s_0}, \widehat{s_1}, \widehat{s_2}, \widehat{s_3}, \ldots, \widehat{s_m}$ is a partial run of $\widehat{A}$ and for all $0 \le i \le m$ the $\widehat{s_i} = (s_i, \langle\beta_i^1, \beta_i^2\rangle)$ and $\gamma_i^2 \in f\mathsf{reg}(\widehat{s_i})$ and for all $0 \le i < m$ the edge $(\widehat{s_i}, \widehat{s_{i+1}})$ is labeled by $(\sigma_{i+1}^1, \sigma_{i+1}^2, \top(\gamma_i^1), \top(\gamma_i^2), x_{i+1}^1, x_{i+1}^2)$.

Proof. The proof is by induction on the length of the path, m. For the base case where $m = 0$, we have that the initial stack contents γ_0^1 and γ_0^2 are $\perp$ implying their maximal common prefix w_0 is also $\perp$. Hence both β_0^1 and β_0^2 are ϵ. This proves the first part of the lemma, as by procedure *Annotate*, the initial state is annotated $\langle\epsilon, \epsilon\rangle$. The $f\mathsf{reg}$-label of the initial state is the set of reachable stack languages of the initial state in the automaton A and thus contains $\perp = \gamma_0^2$.

For the inductive step assume the claim holds for paths of length m we prove it holds for paths of length $m + 1$. Let π_A and π_M be partial runs of A and M on $\left[\begin{smallmatrix}u_1\\u_2\end{smallmatrix}\right]$ and u_1 respectively as follows.

$$\begin{array}{llclclcll}
\pi_A : & (s_0, \gamma_0^2) & \xmapsto{\left[\begin{smallmatrix}\sigma_1^1\\\sigma_1^2\end{smallmatrix}\right]/x_1^2} & (s_1, \gamma_1^2) & \xmapsto{\left[\begin{smallmatrix}\sigma_2^1\\\sigma_2^2\end{smallmatrix}\right]/x_2^2} & \ldots & \xmapsto{\left[\begin{smallmatrix}\sigma_{m+1}^2\\\sigma_{m+1}^2\end{smallmatrix}\right]/x_{m+1}^2} & (s_{m+1}, \gamma_{m+1}^2) \\
\pi_M : & \gamma_0^1 & \xmapsto{\sigma_1^1/x_1^1} & \gamma_1^1 & \xmapsto{\sigma_2^1/x_2^1} & \ldots & \xmapsto{\sigma_{m+1}^1/x_{m+1}^1} & \gamma_{m+1}^1
\end{array}$$

By induction hypothesis there exists a partial run $\widehat{s_0}, \widehat{s_1}, \widehat{s_2} \ldots, \widehat{s_m}$ in $\widehat{A}$ such that the lemma holds. That is, given $w_m = \mathit{max\_common\_prefix}(\gamma^1_m, \gamma^2_m)$, $\widehat{s_m} = (s_m, \langle \beta^1_m, \beta^2_m \rangle)$ where $\beta^1_m = \gamma^1_m / w_m$ and $\beta^2_m = \gamma^2_m / w_m$. Denote by z^1_m and z^2_m the top symbols of γ^1_m and γ^2_m respectively. From the fact that π_A is a run of A on $\left[\begin{smallmatrix} u_1 \\ u_2 \end{smallmatrix}\right]$ we get that $x^2_m = \Delta(\sigma^2_m, z^2_m)$ since A is M-consistent with respect to $\Downarrow_2$. Similarly, from the fact that π_M is a run of M on w_1 we get that $x^1_m = \Delta(\sigma^1_m, z^1_m)$. The same arguments imply that $\gamma^2_{m+1} = x^2_m(\gamma^2_m)$ and $\gamma^1_{m+1} = x^1_m(\gamma^1_m)$. Let r_m be the $f\mathsf{reg}$-label of state $\widehat{s_m}$, $f\mathsf{reg}(\widehat{s_m})$. By induction hypothesis γ^m_2 is in r_m, which implies $w_m \beta^2_m \in r_m$. Define $r_{m+1} = \{x^2_m(wz_2) \;:\; wz_2 \in r_m\}$. It follows that

$$(r_m, r_{m+1}, z^1_m, z^2_m, x^1_m, x^2_m, \sigma^1_m, \sigma^2_m, \beta^1_m, \beta^2_m, \beta^1_{m+1}, \beta^2_{m+1}) \in \\ \mathit{ReachableTransitions}.$$

Hence, procedure *Annotate* will add the edge (s_m, s_{m+1}) to $\widehat{A}$ with the annotation $(\ \sigma^1_{m+1}, \sigma^2_{m+1}, \top(\gamma^1_m), \top(\gamma^2_m),\ x^1_{m+1}, x^2_{m+1}\ )$ and the state s_{m+1} with the annotation $\langle \beta^1_{m+1}, \beta^2_{m+1} \rangle$. Since $\gamma^2_{m+1} \in r_{m+1}$ and the procedure updates $f\mathsf{reg}(s')$ to contain r_{m+1}, the proof is completed.

□

Lemma 2. *If w is in $\mathcal{L}(A)\Downarrow_1$, then w is in $\mathcal{L}(\widetilde{A})$.*

Proof. Let $w_1 = \sigma^1_1 \sigma^1_2 \ldots \sigma^1_n \in \mathcal{L}(A)\Downarrow_1$. Then there exists a word $w_2 = \sigma^2_1 \sigma^2_2 \ldots \sigma^n_2$ such that $\left[\begin{smallmatrix} w_1 \\ w_2 \end{smallmatrix}\right] \in \mathcal{L}(A)$. Let π_A be the accepting run of A on $\left[\begin{smallmatrix} w_1 \\ w_2 \end{smallmatrix}\right]$ and let π_M be the run of M on w_1 as follows.

$$\begin{tikzcd}
\pi_A :\ (s_0, \gamma^2_0) \arrow[r, mapsto, "{\left[\begin{smallmatrix} \sigma^1_1 \\ \sigma^2_1 \end{smallmatrix}\right] / x^2_1}"] & (s_1, \gamma^2_1) \arrow[r, mapsto, "{\left[\begin{smallmatrix} \sigma^1_2 \\ \sigma^2_2 \end{smallmatrix}\right] / x^2_2}"] & \ldots \arrow[r, mapsto, "{\left[\begin{smallmatrix} \sigma^2_n \\ \sigma^2_n \end{smallmatrix}\right] / x^2_n}"] & (s_n, \gamma^2_n) \\
\pi_M :\ \gamma^1_0 \arrow[r, mapsto, "\sigma^1_1 / x^1_1"] & \gamma^1_1 \arrow[r, mapsto, "\sigma^1_2 / x^1_2"] & \ldots \arrow[r, mapsto, "\sigma^1_n / x^1_n"] & \gamma^1_n
\end{tikzcd}$$

Since π_A is accepting, $s_n \in F$. By Lemma 1 $\widehat{s_0}, \widehat{s_1}, \widehat{s_2}, \ldots, \widehat{s_n}$ is a partial run of $\widehat{A}$ where for all $0 \le i \le n$ $\widehat{s_i} = (\langle \beta^1_i, \beta^2_i \rangle)$ and for all $0 \le i < n$ the edge (s_i, s_{i+1}) is labeled by $(\sigma^1_{i+1}, \sigma^2_{i+1}, \top(\gamma^1_i), \top(\gamma^2_i), x^1_{i+1}, x^2_{i+1})$. By definition of $\widehat{F}$, $(s_n, \langle \beta^1_n, \beta^2_n \rangle) \in \widehat{F}$. Hence, by procedure *Project*, $\widehat{s_0}, \widehat{s_1}, \widehat{s_2}, \ldots, \widehat{s_n}$ is a partial run of $\ddot{A}$ where for all $0 \le i < m$ the edge (s_i, s_{i+1}) is labeled by $(\sigma^1_{i+1}, \top(\gamma^1_i), x^1_{i+1})$. By definition of $\ddot{R}$ the run $\pi_{\ddot{R}}$ described below is a run of the NFA $\ddot{R}$ on w_1 :

$$\begin{tikzcd}
\pi_{\ddot{R}} :\ \widehat{s_0} \arrow[r, mapsto, "{\sigma^1_1, \top(\gamma^1_0)}"] & \widehat{s_1} \arrow[r, mapsto, "{\sigma^1_2, \top(\gamma^1_1)}"] & \widehat{s_2} \arrow[r, mapsto, "{\sigma^1_3, \top(\gamma^1_2)}"] & \ldots \arrow[r, mapsto, "{\sigma^1_n, \top(\gamma^1_n)}"] & \widehat{s_n}
\end{tikzcd}$$

By definition of $\ddot{F}$, $\widehat{s_n} \in \ddot{F}$. Hence by definition of $\widetilde{R}$ (the subset construction) there exists $\widetilde{s_0}, \widetilde{s_1}, \widetilde{s_2} \ldots \widetilde{s_n}$ such that $\widetilde{s_i} \ni \widehat{s_i}$ such that $\pi_{\widetilde{R}}$ described below is a run of the DFA $\widetilde{R}$ on w_1 :

$$\begin{tikzcd}
\pi_{\widetilde{R}} :\ \widetilde{s_0} \arrow[r, mapsto, "{\sigma^1_1, \top(\gamma^1_0)}"] & \widetilde{s_1} \arrow[r, mapsto, "{\sigma^1_2, \top(\gamma^1_1)}"] & \widetilde{s_2} \arrow[r, mapsto, "{\sigma^1_3, \top(\gamma^1_2)}"] & \ldots \arrow[r, mapsto, "{\sigma^1_n, \top(\gamma^1_{n-1})}"] & \widetilde{s_n}
\end{tikzcd}$$

By definition of $\widetilde{F}$, $\widetilde{s_n} \in \widetilde{F}$. By definition of $\widetilde{A}$ the following is an accepting run of $\widetilde{A}$ on w_1:

$$\pi_{\widetilde{A}} : \; (\widetilde{s_0}, \gamma_0^1) \xmapsto{\sigma_1^1/x_1^1} (\widetilde{s_1}, \gamma_1^1) \xmapsto{\sigma_2^1/x_2^1} \dots \xmapsto{\sigma_n^1,/x_n^1} (\widetilde{s_n}, \gamma_n^1)$$

Hence $w_1 \in \mathcal{L}(\widetilde{A})$. □

Lemma 3. *If w is in $\mathcal{L}(\widetilde{A})$, then w is in $\mathcal{L}(A)\Downarrow_1$.*

Proof. Let $w_1 = \sigma_1^1\sigma_2^1 \dots \sigma_n^1$ be a word in $\mathcal{L}(\widetilde{A})$. Then, the run $\pi_{\widetilde{A}}$ of $\widetilde{A}$ on w_1 described below, is accepting, .i.e, $\widetilde{s_n} \in \widetilde{F}$.

$$\pi_{\widetilde{A}} : \; (\widetilde{s_0}, \gamma_0^1) \xmapsto{\sigma_1^1/x_1^1} (\widetilde{s_1}, \gamma_1^1) \xmapsto{\sigma_2^1/x_2^1} \dots \xmapsto{\sigma_n^1,/x_n^1} (\widetilde{s_n}, \gamma_n^1) \cdot$$

Since $\widetilde{A}$ is M-consistent with respect to the identity relation, there exists a run $\pi_{\widetilde{R}}$ of $\widetilde{A}$'s characteristic DFA $\widetilde{R}$ and a run π_M of M, both on w_1 as follows.

$$\pi_{\widetilde{R}} : \; \widetilde{s_0} \xmapsto{\sigma_1^1, \top(\gamma_0^1)} \widetilde{s_1} \xmapsto{\sigma_2^1, \top(\gamma_1^1)} \widetilde{s_2} \xmapsto{\sigma_3^1, \top(\gamma_2^1)} \dots \xmapsto{\sigma_n^1, \top(\gamma_{n-1}^1)} \widetilde{s_n}$$
$$\pi_M : \gamma_0^1 \xmapsto{\sigma_1^1/x_1^1} \gamma_1^1 \xmapsto{\sigma_2^1/x_2^1} \gamma_2^1 \xmapsto{\sigma_3^1/x_3^1} \dots \xmapsto{\sigma_n^1/x_n^1} \gamma_n^1$$

By correctness of the subset construction, there exists a run $\pi_{\ddot{R}}$ of the NFA $\ddot{R}$ as follows, where for all i, $\widetilde{s_i} \in \ddot{s_i}$ and $\ddot{s_n} \in \ddot{F}$.

$$\pi_{R_3} : \; \ddot{s_0} \xmapsto{\sigma_1^1, \top(\gamma_0^1)} \ddot{s_1} \xmapsto{\sigma_2^1, \top(\gamma_1^1)} \ddot{s_2} \xmapsto{\sigma_3^1, \top(\gamma_2^1)} \dots \xmapsto{\sigma_n^1, \top(\gamma_{n-1}^1)} \ddot{s_n}$$

By definition of $\ddot{R}$, there exists $x_1^1, x_2^1, \dots, x_n^1$ such that the following is a partial run in $\ddot{A}$ and $\ddot{s_n} \in F$.

$$\pi_{\ddot{A}} : \; \ddot{s_0} \xmapsto{\sigma_1^1, \top(\gamma_0^1)/x_1^1} \ddot{s_1} \xmapsto{\sigma_2^1, \top(\gamma_1^1)/x_2^1} \ddot{s_2} \xmapsto{\sigma_3^1, \top(\gamma_2^1)/x_3^1} \dots \xmapsto{\sigma_n^1, \top(\gamma_{n-1}^1)/x_n^1} \ddot{s_n}$$

By definition of $\ddot{A}$ there exists $\sigma_1^2, \sigma_2^2, \dots, \sigma_n^2$, $z_1^2, z_2^2, \dots, z_n^2$ and $x_1^2, x_2^2, \dots, x_n^2$, such that the following is a partial run of $\widehat{A}$.

$$\pi_{\widehat{A}} : \; \ddot{s_0} \xmapsto{\left[\begin{smallmatrix}\sigma_1^1\\ \sigma_1^2\end{smallmatrix}\right], \substack{\top(\gamma_0^1)\\ z_1^2} / \substack{x_1^1\\ x_1^2}} \ddot{s_1} \xmapsto{\left[\begin{smallmatrix}\sigma_2^1\\ \sigma_2^2\end{smallmatrix}\right], \substack{\top(\gamma_1^1)\\ z_2^2} / \substack{x_2^1\\ x_2^2}} \ddot{s_2} \quad \dots \quad \ddot{s_{n-1}} \xmapsto{\left[\begin{smallmatrix}\sigma_n^1\\ \sigma_n^2\end{smallmatrix}\right], \substack{\top(\gamma_{n-1}^1)\\ z_n^2} / \substack{x_n^1\\ x_n^1}} \ddot{s_n}$$

Hence, there exists $s_0, s_1, ..., s_n$ such that the following is a partial run of A where $s_n \in F$ and γ_0^2 is defined as $\perp$ and $\gamma_i^2 = x_i^2(\gamma_{i-1}^2)$.

$$\pi_A : (s_0, \gamma_0^2) \xmapsto{\left[\begin{smallmatrix}\sigma_1^1\\ \sigma_1^2\end{smallmatrix}\right], z_1^2/x_1^2} (s_1, \gamma_1^2) \xmapsto{\left[\begin{smallmatrix}\sigma_2^1\\ \sigma_2^2\end{smallmatrix}\right], z_2^2/x_2^2} (s_2, \gamma_2^2) \quad \dots \quad \xmapsto{\left[\begin{smallmatrix}\sigma_n^1\\ \sigma_n^2\end{smallmatrix}\right], z_n^2/x_n^1} (s_n, \gamma_n^2)$$

Therefore, the run of A on $\left[\begin{smallmatrix}w_1\\ w_2\end{smallmatrix}\right]$ where $w_2 = \sigma_1^2\sigma_2^2 \dots \sigma_n^2$ is accepting. Hence w_1 is in $\mathcal{L}(A)\Downarrow_1$. □

The last two lemmas provide a proof for the theorem.

Proofs of Section 6

Claim. Let $M_0 = L \cap R_0$ for some languages L and R_0, and let R be a bi-language.

- Define $M_{i+1} = ((M_i \times \Sigma^*) \cap R)\Downarrow_2$. If L is left preserved by the bi-language R then for all $i \geq 1$ the languages M_i can be given by $M_i = L \cap R_i$ where $R_i = ((R_{i-1} \times \Sigma^*) \cap R)\Downarrow_2$.
- Define $M_{i+1} = ((\Sigma^* \times M_i) \cap R)\Downarrow_2$. If L is right preserved by the bi-language R then for all $i \geq 1$ the languages M_i can be given by $M_i = L \cap R_i$ where $R_i = ((\Sigma^* \times R_{i-1}) \cap R)\Downarrow_2$.

Proof. We prove the claim for the first case, the second case is symmetric. Let $M_0 = L \cap R_0$ for some languages L and R_0 and $L = ((L \times \Sigma^*) \cap R)\Downarrow_2$. We prove that $M_i = L \cap R_i$ where $R_i = ((R_{i-1} \times \Sigma^*) \cap R)\Downarrow_2$ by induction on i. For the base case where $i = 0$ the claim trivially holds. For the inductive step, we assume $M_i = L \cap R_i$ where $R_i = ((R_{i-1} \times \Sigma^*) \cap R)\Downarrow_2$ and compute M_{i+1}.

$$\begin{array}{rcl}
M_{i+1} & = & ((M_i \times \Sigma^*) \cap R)\Downarrow_2 \\
 & & \text{by definition of } M_i \\
 & = & (((L \cap R_i) \times \Sigma^*) \cap R)\Downarrow_2 \\
 & & \text{by induction hypothesis } M_i = L \cap R_i \\
 & = & (((L \times \Sigma^*) \cap (R_i \times \Sigma^*)) \cap R)\Downarrow_2 \\
 & & \text{by distribution of } \times \text{ with respect to } \cap \\
 & = & (((L \times \Sigma^*) \cap R) \cap ((R_i \times \Sigma^*) \cap R))\Downarrow_2 \\
 & & \text{by distribution of } \cap \\
 & = & (((L \times \Sigma^*) \cap R)\Downarrow_2) \cap (((R_i \times \Sigma^*) \cap R)\Downarrow_2) \\
 & & \text{by distribution of} \Downarrow \text{ with respect to } \cap \\
 & = & L \cap ((R_i \times \Sigma^*) \cap R)\Downarrow_2 \\
 & & \text{since } L = (L \times \Sigma^* \cap R)\Downarrow_2 \\
 & = & L \cap R_{i+1} \\
 & & \text{by induction hyp. } R_{i+1} = (R_i \times \Sigma^* \cap R)\Downarrow_2
\end{array}$$

□

Claim. Let $\langle D, \phi \rangle$ be a verification problem where $D = (\Sigma, \chi, \Theta, \rho)$.

- If $\Theta = L \cap R_0$ and L is left-preserved by ρ and ρ is regular, then the verification problem $\langle D, \phi \rangle$ can be solved by applying the procedure for forward model checking given in the figure of Claim 6.
- If $\neg\phi = L \cap R_0$ and L is right-preserved by ρ and ρ is regular, then the verification problem $\langle D, \phi \rangle$ can be solved by applying the procedure for backward model checking given in the figure of Claim 6.

Proof. Corollary of the claim above. □

References

[ABJ98] P.A. Abdulla, A. Bouajjani, and B. Jonsson. On-the-fly analysis of systems with unbounded, lossy FIFO channels. In A.J. Hu and M.Y. Vardi, editors, *Proc. 10th Intl. Conference on Computer Aided Verification (CAV'98)*, volume 1427 of *Lect. Notes in Comp. Sci.*, pages 305–318. Springer-Verlag, 1998.

[ABJN99] P.A. Abdulla, A. Bouajjani, B. Jonsson, and M. Nilsson. Handling global conditions in parametrized system verification. In N. Halbwachs and D. Peled, editors, *Proc. 11st Intl. Conference on Computer Aided Verification (CAV'99)*, volume 1633 of *Lect. Notes in Comp. Sci.*, pages 134–145. Springer-Verlag, 1999.

[AK86] K. R. Apt and D. Kozen. Limits for automatic program verification of finite-state concurrent systems. *Information Processing Letters*, 22(6), 1986.

[BCG86] M.C. Browne, E.M. Clarke, and O. Grumberg. Reasoning about networks with many finite state processes. In *Proc. 5th ACM Symp. Princ. of Dist. Comp.*, pages 240–248, 1986.

[BCM$^+$92] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992.

[BEM97] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In A. Mazurkiewicz and J. Winkowski, editors, *8th International Conference on Concurrency Theory (CONCUR'97)*, volume 1243 of *Lect. Notes in Comp. Sci.*, pages 135–150. Springer-Verlag, 1997.

[BF00] J-P. Bodeveix and M. Filali. Experimenting acceleration methods for the validation of infinite state systems. In *International Workshop on Distributed System Validation and Verification (DSVV'2000)*, Taipei, Taiwan, April 2000.

[BG96] B. Boigelot and P. Godefroid. Symbolic verification of coomunication protocols with infinite state spaces using QDDs. In R. Alur and T. Henzinger, editors, *Proc. 8th Intl. Conference on Computer Aided Verification (CAV'96)*, volume 1102 of *Lect. Notes in Comp. Sci.*, pages 1–12. Springer-Verlag, 1996.

[BGWW97] B. Boigelot, P. Godefroid, B. Willems, and P. Wolper. The power of QDDs. In *Proc. of the Fourth International Static Analysis Symposium*, Lect. Notes in Comp. Sci. Springer-Verlag, 1997.

[BH97] A. Bouajjani and P. Habermehl. Symbolic reachability analysis of FIFO-channel systems with non-regular sets of configurations. In *Proc. 24th Int. Colloq. Aut. Lang. Prog.*, volume 1256 of *Lect. Notes in Comp. Sci.* Springer-Verlag, 1997.

[BJNT00] A. Bouajjani, B. Jonsson, M. Nilsson, and T. Touilli. Regular model checking. In E.A.Emerson and A.P.Sistla, editors, *Proc. 12th Intl. Conference on Computer Aided Verification (CAV'00)*, volume 1855 of *Lect. Notes in Comp. Sci.*, pages 403–418. Springer-Verlag, 2000.

[BM96] A. Bouajjani and O. Maler. Reachability analysis of push-down automata. In *Workshop on Infinite-state Systems*, Pisa, 1996.

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

[CC77] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th Annual Symposium on Principles of Programming Languages*. ACM Press, 1977.

[CGJ95] E.M. Clarke, O. Grumberg, and S. Jha. Verifying parametrized networks using abstraction and regular languages. In *6th International Conference on Concurrency Theory (CONCUR'95)*, pages 395–407, Philadelphia, PA, August 1995.

[CJ98] H. Comon and Y. Jurski. Multiple counters automata, safety analysis and presburger arithmetic. In *Proc. 10th Intl. Conference on Computer Aided Verification (CAV'98)*, Lecture Notes in Computer Science, pages 268–279, 1998.

[EHRS00] J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon. Efficient algorithms for model checking pushdown systems. In E.A.Emerson and A.P.Sistla, editors, *Proc. 12th Intl. Conference on Computer Aided Verification (CAV'00)*, volume 1855 of *Lect. Notes in Comp. Sci.*, pages 232–247. Springer-Verlag, 2000.

[EN96] E.A. Emerson and K.S. Namjoshi. Automatic verification of parameterized synchronous systems. In R. Alur and T. Henzinger, editors, *Proc. 8th Intl. Conference on Computer Aided Verification (CAV'96)*, Lect. Notes in Comp. Sci. Springer-Verlag, 1996.

[FO97] L. Fribourg and H. Olsen. Reachability sets of parametrized rings as regular languages. In *Workshop on Verification of Infinite State Systems (Infinity'97), Electronic Notes in Theoretical Computer Science*, volume 9, July 1997.

[FP] Dana Fisman and Amir Pnueli. Beyond regular model checking. In *Proc. 21th Conference on the Foundations of Software Technology and Theoretical Computer Science*, Lecture Notes in Computer Science. To appear.

[FWW97] A. Finkel, B. Willems, and P. Wolper. A direct symbolic approach to model checking pushdown systems (extended abstract). In *Proc. Infinity'97, Electronic Notes in Theoretical Computer Science*, 1997.

[HLR92] N. Halbwachs, F. Lagnier, and C. Ratel. An experience in proving regular networks of processes by modular model checking. *Acta Informatica*, 29(6/7):523–543, 1992.

[ID96] C.N. Ip and D. Dill. Verifying systems with replicated components in Murφ. In R. Alur and T. Henzinger, editors, *Proc. 8th Intl. Conference on Computer Aided Verification (CAV'96)*, Lect. Notes in Comp. Sci. Springer-Verlag, 1996.

[JJK94] Jakob L. Jensen, Michael E. Jørgensen, and Nils Klarlund. Monadic second-order logic for parameterized verification. Technical Report RS-94-10, Basic Research in Computer Science, May 1994. 14 pp.

[JN00] B. Jonsson and M. Nilsson. Transitive closures of regular relations for verifying infinite-state systems. In S. Graf, editor, *Proceedings of TACAS'00*, Lect. Notes in Comp. Sci., 2000. To Appear.

[KM95] R.P. Kurshan and K.L. McMillan. A structural induction theorem for processes. *Information and Computation*, 117:1–11, 1995.

[KMM+97] Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic model checking with rich assertional languages. In O. Grumberg, editor, *Proc. 9th Intl. Conference on Computer Aided Verification (CAV'97)*, Lect. Notes in Comp. Sci., pages 424–435. Springer-Verlag, 1997.

[KP00] Y. Kesten and A. Pnueli. Control and data abstractions: The cornerstones of practical formal verification. *Software Tools for Technology Transfer*, 4(2):317–426, 2000.

[LHR97] D. Lesens, N. Halbwachs, and P. Raymond. Automatic verification of parameterized linear networks of processes. In *24th ACM Symposium on Principles of Programming Languages, POPL'97*, Paris, 1997.

[Pet81] G.L. Peterson. Myths about the mutual exclusion problem. *Info. Proc. Lett.*, 12(3):115–116, 1981.

[PS00] A. Pnueli and E. Shahar. Liveness and acceleration in parameterized verification. In E.A.Emerson and A.P.Sistla, editors, *Proc. 12th Intl. Conference on Computer Aided Verification (CAV'00)*, volume 1855 of *Lect. Notes in Comp. Sci.*, pages 328–343. Springer-Verlag, 2000.

[RA81] G. Ricart and A.K. Agrawala. An optimal algorithm for mutual exclusion in computer networks. *Comm. ACM*, 24(1):9–17, 1981. Corr. ibid. 1981, p.581.

[Sen97] G. Senizergues. The equivalence problem for deterministic pushdown automata is decidable. volume 1256 of *Lect. Notes in Comp. Sci.*, pages 671–681, 1997.

[SG89] Z. Shtadler and O. Grumberg. Network grammars, communication behaviors and automatic verification. In J. Sifakis, editor, *Automatic Verification Methods for Finite State Systems*, volume 407 of *Lect. Notes in Comp. Sci.*, pages 151–165. Springer-Verlag, 1989.

[SG92] A.P. Sistla and S.M. German. Reasoning about systems with many processes. *J. ACM*, 39:675–735, 1992.

[Sha96] E. Shahar. The tlv system and its application. Master's thesis, Weizmann Institute, 1996.

[Sti01] Colin Stirling. Decidability of dpda equivalence. In *To appear in FODDACS 2001*, Lect. Notes in Comp. Sci. Springer-Verlag, 2001.

[Wal96] I. Walukiewicz. Pushdown processes: Games and model checking. In R. Alur and T. Henzinger, editors, *Proc. 8th Intl. Conference on Computer Aided Verification (CAV'96)*, Lect. Notes in Comp. Sci. Springer-Verlag, 1996.

[WB94] P. Wolper and B. Boigelot. Symbolic verification with periodic sets. In D. Dill, editor, *Proc. 6th Intl. Conference on Computer Aided Verification (CAV'94)*, volume 818 of *Lect. Notes in Comp. Sci.*, pages 55–67. Springer-Verlag, 1994.

[WB98] Pierre Wolper and Bernard Boigelot. Verifying systems with infinite but regular state spaces. In A.J. Hu and M.Y. Vardi, editors, *Proc. 10th Intl. Conference on Computer Aided Verification (CAV'98)*, volume 1427 of *Lect. Notes in Comp. Sci.*, pages 88–97. Springer-Verlag, 1998.

[WL89] P. Wolper and V. Lovinfosse. Verifying properties of large sets of processes with network invariants. In J. Sifakis, editor, *Automatic Verification Methods for Finite State Systems*, volume 407 of *Lect. Notes in Comp. Sci.*, pages 68–80. Springer-Verlag, 1989.