Model Checking Probabilistic Systems against Pushdown Specifications

Clemens Dubslaff, Christel Baier, Manuela Berg

Institut für Theoretische Informatik, Technische Universität Dresden, Germany

Abstract

Model checking is a fully automatic verification technique traditionally used to verify finite-state systems against regular specifications. Although regular specifications have been proven to be feasible in practice, many desirable specifications are non-regular. For instance, requirements which involve counting can not be formalized by regular specifications but using pushdown specifications, i.e., context-free properties represented by pushdown automata. Research on model-checking techniques for pushdown specifications is, however, rare and limited to the verification of non-probabilistic systems.

In this paper, we address the probabilistic model-checking problem for systems modeled by discrete-time Markov chains and specifications that are provided by deterministic pushdown automata over infinite words. We first consider finite-state Markov chains and show that the quantitative and qualitative model-checking problem is solvable via a product construction and techniques that are known for the verification of probabilistic pushdown automata. Then, we consider recursive systems modeled by probabilistic pushdown automata with an infinite-state Markov chain semantics. We first show that imposing appropriate compatibility (visibility) restrictions on the synchronizations between the pushdown automaton for the system and the specification, decidability of the probabilistic model-checking problem can be established. Finally we prove that slightly departing from this compatibility assumption leads to the undecidability of the probabilistic model-checking problem, even for qualitative properties specified by deterministic context-free specifications.

1. Introduction

In the traditional model-checking approach (see, e.g., [8, 5]), an operational system model is verified against a formal specification provided by some propositional temporal formula. Formulae of linear temporal logic (LTL) impose *regular* constraints on the runs of the system. The most prominent LTL model-checking approach relies on a representation of LTL formulae by finite-state automata [23].

Many relevant safety and liveness properties are indeed regular, but there are also interesting non-regular system properties. Examples are properties that involve some kind of counting, such as the property stating that in each finite run of the system, the number of request actions is greater or equal than the number of acknowledgements. Another example are pre-/post conditions for recursive programs stating that whenever condition a holds in the moment where some recursive procedure P is invoked, then condition b holds when returning from procedure P. More examples can be found in [15] and [4]. Research on model-checking techniques for non-regular specifications is, however, rare.

Verification techniques for pushdown specifications, i.e., context-free specification provided by some pushdown automaton, have been first addressed in [19] and were significantly enhanced by Kupfermann, Piterman and Vardi in [15]. In the latter paper, a model-checking algorithm for finite-state systems and nondeterministic pushdown tree automata with parity acceptance conditions is provided that runs in time exponentially in the size of the system and specification. For procedural systems modeled by pushdown automata, the model-checking problem against nondeterministic pushdown tree automata with parity acceptance conditions becomes undecidable. Decidability results can though be obtained by imposing some syntactic restrictions on the pushdown automata formalizing the systems and the specifications. *Visibly pushdown automata (VPAs)* were first introduced in [4]. Although more expressive than finite automata, the class of languages described by VPAs enjoys the same closure properties as regular languages. A convenient formalism to describe VPA specifications is the temporal logic CaRet that extends LTL with modalities for calls and returns [3].

The purpose of this paper is to study verification of probabilistic systems with an operational semantics based on discrete-time Markov chains against pushdown specifications. Typical examples for such systems contain unreliable components that behave faulty with some small probabilities (e.g., channels that might lose messages) or rely on a protocol with randomized actions (e.g., tossing a coin to break symmetry or to generate input samples or fingerprints) [13, 17]. In the linear-time setting, the task

Email addresses: `dubslaff@tcs.inf.tu-dresden.de` (Clemens Dubslaff), `baier@tcs.inf.tu-dresden.de` (Christel Baier), `manu@tcs.inf.tu-dresden.de` (Manuela Berg)

of the *quantitative model-checking problem* for a Markov chain $\mathcal{M}$ is to compute the probability that a given measurable linear-time property φ holds. When also taking a probability interval $I \subseteq [0,1]$ as input, the quantitative model-checking problem can be rephrased as decision problem asking whether the probability measure of the set of runs in $\mathcal{M}$ satisfying φ belongs to I. The *qualitative model-checking problem* for Markov chains and linear-time properties asks whether the given property holds almost surely, i.e., with probability 1. For specifications provided by LTL formulae and finite-state Markov chains, the model-checking problem is well understood. Both the qualitative and quantitative version are solvable using an automata-based approach [21, 5] or an incremental approach that modifies the given Markov chain by adding components for the subformulae iteratively [10].

For systems with an infinite-state Markov chain semantics and regular specifications, solving the model-checking problem is more involved. However, there are systems for which algorithms could be provided. Examples are systems modeled by probabilistic lossy channel systems [1], probabilistic vector addition systems [2] or probabilistic pushdown automata [11, 6, 24]. For the latter, i.e., probabilistic pushdown automaton (pPDA), and ω-regular specifications (including requirements expressed by LTL formulae), both the qualitative and the quantitative model-checking problem for pPDA and regular specifications turned out to be decidable.

In this paper, we address the "opposite" problem which asks for model-checking algorithms when the system is modeled by a Markov chain and the specification given by a pushdown automaton. To the best of our knowledge, this is the first attempt to study the model-checking problem for probabilistic system models and context-free specifications. The main contribution of this paper is to show the decidability of the qualitative and quantitative model-checking problem for

(1) finite-state Markov chains and context-free linear-time properties specified by deterministic pushdown automata (DPDAs), and
(2) recursive systems modeled by visibly pPDAs and deterministic context-free linear-time properties specified by visibly DPDAs.

In both cases, we consider DPDAs over infinite words (briefly called ω-DPDAs) and deal with Muller and Büchi acceptance conditions. To establish (1), we employ a product construction and provide a reduction to the model-checking problem for systems modeled by pPDA. Although the product construction is standard, some care is needed to treat ε-transitions in the ω-DPDA properly and to ensure that the product meets indeed the syntactic conditions of a pPDA. For (2), we adapt known concepts for verifying systems modeled by visibly PDA against visibly PDA-specifications [4] to the probabilistic setting. This requires an adequate definition of the visibility condition for pPDA and also relies on a reduction to the model-checking problem for pPDAs and regular specifications.

Given that two-stack automata have Turing power, one cannot expect a general decidability result of the model-checking problem where both the system and the requirement are modeled by some kind of PDAs. In fact, we show that if the visibility conditions of the pPDA for the system and the DPDA for the requirements are not compatible, even the qualitative model-checking problem becomes undecidable.

Outline. The remainder of the paper is organized as follows. Section 2 summarizes some basic concepts on ω-automata, Markov chains and pushdown systems and introduces the notations used throughout the paper. In Section 3, we present our results on model checking finite-state Markov chains against ω-DPDAs. The model-checking problem for systems modeled by pPDAs and ω-DPDA-specifications is presented in Section 4. The paper ends with some concluding remarks in Section 5.

2. Preliminaries

We denote the set of finite (respectively, infinite) words over an alphabet Σ by Σ^* (respectively, Σ^ω). If $\sigma \in \Sigma^* \cup \Sigma^\omega$, then σ_i denotes the $(i{+}1)$th symbol of σ and $|\sigma|$ its length. If $U_1, \ldots, U_k$ are sets then $\cdot|_{U_i} : U_1 \times \cdots \times U_k \to U_i$ denotes the projection function given by $u|_{U_i} = u_i$ for each tuple $u = (u_1, ..., u_k) \in U_1 \times \cdots \times U_k$. For sequences $\pi{=}\pi_0\,\pi_1 \cdots$ of tuples $\pi_j \in U_1 \times \cdots \times U_k$ the projection function is applied elementwise, i.e., $\pi|_{U_i} = \pi_0|_{U_i}\,\pi_1|_{U_i} \cdots$.

2.1. ω-Automata

We briefly summarize our notations for finite-state and pushdown automata over infinite words. For further details we refer to [20]. A *nondeterministic Büchi automaton (NBA)* is a tuple $\mathcal{B} = (Q, Q_0, \Sigma, \delta, Acc)$, where Q is a finite set of states, $Q_0 \subseteq Q$ is the set of initial states, Σ is an input alphabet, $\delta : Q \times \Sigma \to 2^Q$ is the transition function, and $Acc \subseteq Q$ is the acceptance condition of $\mathcal{B}$. The size of $\mathcal{B}$ (denoted by $|\mathcal{B}|$) is defined as the number of transitions in $\mathcal{B}$, i.e., $|\mathcal{B}| = \sum_{q \in Q, a \in \Sigma} |\delta(q,a)|$. $\mathcal{B}$ is called *total*, if $|\delta(q,a)| \geq 1$ for all $a \in \Sigma$, and *deterministic*, if $|Q_0| = 1$ and $|\delta(q,a)| \leq 1$ for all $q \in Q$ and $a \in \Sigma$. Deterministic $\mathcal{B}$ are abbreviated by DBA.

An infinite state sequence $\pi = \pi_0\,\pi_1 \cdots \in Q^\omega$ is called a *run*, if $\pi_0 \in Q_0$ and there exists a word $\sigma \in \Sigma^\omega$ such that $\pi_{i+1} \in \delta(\pi_i, \sigma_i)$ for all $i \in \mathbb{N}$. In this case, π is said to be *a run for* σ. We write $\mathrm{Runs}(\mathcal{B})$ to denote the set of all runs in $\mathcal{B}$. Note that if $\mathcal{B}$ is deterministic, there exists a unique run for every word σ. If $\pi \in Q^\omega$ is an infinite state sequence, then $\inf(\pi)$ denotes the set of all states occurring infinitely often in π, i.e.,

$$\inf(\pi) = \{q \in Q : \forall i \in \mathbb{N}\ \exists j > i \text{ such that } \pi_j = q\}.$$

A run π in $\mathcal{B}$ is called *accepting*, if $\inf(\pi) \cap Acc \neq \emptyset$. The *language of* $\mathcal{B}$ is defined as

$$\mathrm{L}_\omega(\mathcal{B}) = \{\sigma \in \Sigma^\omega \colon \text{ there exists an accepting run for } \sigma\}.$$

NBAs accept the full class of ω-regular languages, whereas DBAs constitute a smaller class of ω-regular languages.

An *ω-pushdown automaton (ω-PDA)* is a tuple $\mathcal{A} = (Q, \Gamma, q_0, \gamma_0, \Sigma, \delta, Acc)$, where Q is a finite set of *control states*, Γ is a stack alphabet, q_0 is the initial control state, γ_0 is the initial stack content, Σ is an input alphabet and $\delta : Q \times \Gamma \times (\Sigma \cup \{\varepsilon\}) \to 2^{Q \times \Gamma^{\leq 2}}$ is the transition function. Here, $\Gamma^{\leq 2} = \{\alpha \in \Gamma^* \colon |\alpha| \leq 2\}$. We postpone the exact definition of the acceptance condition Acc for $\mathcal{A}$ to the semantics paragraph below.
Throughout the paper, we use capital letters $X, Y, \ldots$ for elements of Γ, and $\alpha, \beta, \gamma, \ldots$ for words in Γ^*. A pair $(q, \gamma) \in Q \times \Gamma^*$ is called a *configuration* of $\mathcal{A}$, written as $q\gamma$. The configuration $q_0\gamma_0$ is called *initial.* Given a configuration $q\gamma$, where $\gamma = X\alpha$ with $X \in \Gamma$ and $\alpha \in \Gamma^*$, we call qX the *head of* $q\gamma$. The size of $\mathcal{A}$ (denoted by $|\mathcal{A}|$) is defined as the number of transitions in $\mathcal{A}$, i.e., $|\mathcal{A}| = \sum_{qX \in Q \times \Gamma, a \in \Sigma} |\delta(qX, a)|$. An ω-PDA $\mathcal{A}$ is called *deterministic ω-PDA* (denoted by ω-DPDA), if $|\delta(qX, a)| + |\delta(qX, \varepsilon)| \leq 1$ for all heads $qX \in Q \times \Gamma$ and input symbols $a \in \Sigma$. Note that for all heads in an ω-DPDA, there is either exactly one outgoing ε-transition or for any input symbols there is at most one outgoing transition. An ω-PDA $\mathcal{A}$ is called *total* if for all $qX \in Q \times \Gamma$ either $|\delta(qX, a)| \geq 1$ for all $a \in \Sigma$ or $\sum_{a \in \Sigma} |\delta(qX, a)| = 0$. Since every ω-PDA can be effectively transformed into a total ω-PDA that accepts the same language, we assume ω-PDAs to be total in the following.
Let $\pi = q_0\gamma_0\, q_1\gamma_1 \cdots$ be a sequence of configurations of an ω-PDA $\mathcal{A}$, such that for all $i \in \mathbb{N}$, the word γ_i has the form $X_i\beta_i \in \Gamma^*$. We call π a *run in* $\mathcal{A}$ if there exists a sequence $a_0\, a_1 \cdots \in (\Sigma \cup \{\varepsilon\})^\omega$ such that $q_{i+1}\alpha_{i+1} \in \delta(q_i X_i, a_i)$ with $\gamma_{i+1} = \alpha_{i+1}\beta_i$. In this case, we also say that π is a *run for* σ. Runs($\mathcal{A}$) denotes the set of all runs in the ω-PDA $\mathcal{A}$. As for DBAs, for every word σ there exists a unique run for σ in $\mathcal{A}$ if $\mathcal{A}$ is a ω-DPDA.
Several acceptance conditions for ω-PDAs have been studied, see, e.g., [9, 16, 20]. Recall that $\cdot|_Q : Q \times \Gamma^* \to Q$ denotes the function which maps a configuration to its control state. For Büchi acceptance, Acc is a subset of Q and a run π is called *Büchi accepting*, if $\inf(\pi|_Q) \cap Acc \neq \emptyset$. For Muller acceptance, Acc is a subset of 2^Q and a run π is called *Muller accepting*, if $\inf(\pi|_Q) \in Acc$. The *language* of an ω-PDA $\mathcal{A}$ is defined as

$$\mathrm{L}_\omega(\mathcal{A}) = \{\sigma \in \Sigma^\omega \colon \text{ there exists an accepting run for } \sigma\}.$$

Note that there can be accepting runs for finite words, since it is possible to only take ε-transitions from some moment on, visiting control states from the acceptance condition. Furthermore, Büchi acceptance is covered by Muller acceptance by simply choosing $2^{Acc} \setminus \{\emptyset\}$ as Muller acceptance condition. Thus, unless stated differently, we assume Muller acceptance. Whereas the class of ω-context-free languages is the same as the class of languages accepted by Muller accepting ω-PDAs or ω-DPDA, this is not the case for Büchi accepting ω-pushdown automata. Büchi accepting ω-PDAs constitute the class of ω-context-free languages, but the class of languages accepted by Büchi accepting ω-DPDAs is strictly smaller [9, 20].

Another class of ω-context-free languages is described by *ω-visibly pushdown automata*, introduced in [4]. They are ω-PDAs with syntactic constraints on the transition rules and focus on matching calls and returns in recursive sequential programs.
An *ω-visibly pushdown automaton (ω-VPA)* is an ω-PDA $\mathcal{A}{=}(Q, \Gamma, q_0, \gamma_0, \Sigma, \delta, Acc)$ where the alphabet Σ has a partition $\widetilde{\Sigma} = [\Sigma_c, \Sigma_r, \Sigma_l]$ into alphabets consisting of symbols for calls, returns and local events, respectively, such that for all $pX \in Q \times \Gamma$, $a \in \Sigma$ and $q\alpha \in \delta(pX, a) \in Q{\times}\Gamma^{\leq 2}$:

(push)	$a \in \Sigma_c$	iff	$\alpha = YX$ for some $Y \in \Gamma$
(pop)	$a \in \Sigma_r$	iff	$\alpha = \varepsilon$
(local)	$a \in \Sigma_l$	iff	$\alpha = X$

$\widetilde{\Sigma}$ is called the *visible alphabet* of $\mathcal{A}$. When $\widetilde{\Sigma}$ and $\widetilde{\Sigma}'$ are two visible alphabets, we write $\widetilde{\Sigma} \subseteq \widetilde{\Sigma}'$ if $\Sigma_\kappa \subseteq \Sigma'_\kappa$ for all $\kappa \in \{c, r, l\}$. In this case, $\widetilde{\Sigma}'$ is said to be *compatible* with $\widetilde{\Sigma}$. As usual for ω-VPAs, we assume w.l.o.g. that the transition function does not include ε-transitions.
In contrast to general ω-PDAs, the class of languages recognizable by ω-VPAs enjoys many properties of ω-regular languages [4]. In the automata-theoretic approach of model checking, especially the intersection-closure property is of interest. ω-VPA languages are closed under intersection since the type of the stack operation (e.g., push, pop and local operation) is encoded into the input symbol of a transition. This allows to synchronize over the stack operations. As in the case of ω-PDAs, ω-VPAs define a greater class of languages than its deterministic version, which we denote by ω-DVPA.

2.2. Probabilistic Systems

In this section we consider probabilistic system models, which are capable of modeling unreliability, randomized actions (e.g., tossing a coin) or stochastic assumptions on the system environment. We deal with labeled Markov chains where the states are labeled with a single symbol of an alphabet Σ. Note that this approach covers the case where states are labeled by a subset of atomic propositions Ap, which is commonly used to reason about temporal logics, by simply identifying Σ with 2^{Ap}.

We briefly explain our notations and refer to standard textbooks such as [14], for further details. A *Markov chain* $\mathcal{M}$ is a tuple $(S, s_0, \mathrm{Prob}_\mathcal{M}, \Sigma, \lambda)$, where S is a countable set of states, $s_0 \in S$ is the initial state and $\mathrm{Prob}_\mathcal{M}{:}S{\times}S \to [0, 1]$ is a transition probability function such that

$$\sum_{t \in S} \mathrm{Prob}_\mathcal{M}(s, t) \in \{0, 1\}$$

for all states $s \in S$. Σ is the state labeling alphabet and $\lambda : S \to \Sigma$ the state labeling function. A Markov chain is called *finite* if S is. $\mathrm{Paths}(\mathcal{M})$ denotes the set of finite sequences of states $s_0\, s_1\, \cdots\, s_n$ where $\mathrm{Prob}_{\mathcal{M}}(s_i, s_{i+1}) > 0$ for $0{\le}i{<}n$. Each infinite state sequences $s_0\, s_1\, \cdots$ where $\mathrm{Prob}_{\mathcal{M}}(s_i, s_{i+1}) > 0$ for all $i \in \mathbb{N}$ is called a *run* in $\mathcal{M}$. The set of all runs is denoted by $\mathrm{Runs}(\mathcal{M})$. The *trace* of a run $\pi = s_0\, s_1\, \cdots$ is defined as $\lambda(\pi) = \lambda(s_0)\, \lambda(s_1) \cdots \in \Sigma^\omega$. The probability measure $\mathrm{P}_{\mathcal{M}}$ over $\mathrm{Runs}(\mathcal{M})$ is defined in the standard way. Let $\mathrm{Cyl}(\hat{\pi})$ denote the *basic cylinder* of $\hat{\pi} \in \mathrm{Paths}(\mathcal{M})$, which is defined as $\{\pi \in \mathrm{Runs}(\mathcal{M})\colon \exists \rho \in S^\omega . \pi = \hat{\pi}\rho\}$. If $\mathcal{E}$ is the σ-algebra generated by the set $\{\mathrm{Cyl}(\hat{\pi})\colon \hat{\pi}{\in} \mathrm{Paths}(\mathcal{M})\}$, then $\mathrm{P}_{\mathcal{M}}$ denotes the unique probability measure on $\mathcal{E}$ where

$$\mathrm{P}_{\mathcal{M}}(\mathrm{Cyl}(\hat{\pi})) = \textstyle\prod_{i=1}^{|\hat{\pi}|-1} \mathrm{Prob}_{\mathcal{M}}(\hat{\pi}_{i-1}, \hat{\pi}_i).$$

Probabilistic pushdown automata [11] provide a model of probabilistic recursive sequential programs. They have the same expressiveness as the model of *recursive Markov chains*, which were studied, e.g., in [12].
Formally, a *probabilistic pushdown automaton (pPDA)* is a tuple $\mathcal{N} = (Q, \Gamma, q_0, \gamma_0, \mathrm{Prob}_{\mathcal{N}}, \Sigma, \lambda)$ where Q, Γ, q_0 and γ_0 are defined as for ω-PDAs and $\mathrm{Prob}_{\mathcal{N}} : (Q \times \Gamma) \times (Q \times \Gamma^{\le 2}) \to [0, 1]$ is the transition probability function, where for all $p \in Q$ and $X \in \Gamma$ we have

$$\textstyle\sum_{q\in Q, \alpha\in\Gamma^{\le 2}} \mathrm{Prob}_{\mathcal{N}}(pX, q\alpha) \in \{0, 1\}.$$

Configurations are labeled by symbols from the labeling alphabet Σ employing the labeling function $\lambda : Q \times \Gamma^* \to \Sigma$, where we restrict ourselves to labeling functions that only depend on the control state, i.e., for all $q\gamma, q'\gamma' \in Q{\times}\Gamma^*$ we have $\lambda(q\gamma) = \lambda(q'\gamma')$ if $q = q'$. We simply write $\lambda(q)$ to denote the labeling of all configurations $q\gamma \in Q{\times}\Gamma^*$. Using regular labeling functions, where the labeling depends on regular languages over the control state and the stack content, does not add expressiveness (see, e.g., [6]).
The semantics of a pPDA $\mathcal{N}$ is defined as the infinite Markov chain $\mathcal{M}_{\mathcal{N}} = (Q \times \Gamma^*, q_0\gamma_0, \mathrm{Prob}_{\mathcal{M}_{\mathcal{N}}}, \Sigma, \lambda)$ where $\mathrm{Prob}_{\mathcal{M}_{\mathcal{N}}}(pX\gamma, q\alpha\gamma) = \mathrm{Prob}_{\mathcal{N}}(pX, q\alpha)$ for all $\gamma \in \Gamma^*$. A *run* of a pPDA $\mathcal{N}$ is a run in $\mathcal{M}_{\mathcal{N}}$. $\mathrm{Runs}(\mathcal{N})$ denotes the set of runs in $\mathcal{N}$. We simply write $\mathrm{P}_{\mathcal{N}}$ rather than $\mathrm{P}_{\mathcal{M}_{\mathcal{N}}}$.

2.3. Model Checking Probabilistic Systems

For stating algorithmic problems over a probabilistic system, a finite representation thereof is required. In the following, we assume rational transition probabilities in the probabilistic systems. The size $|\mathcal{M}|$ of a finite Markov chain $\mathcal{M}$ (or pPDA $\mathcal{N}$, respectively) is then defined as the number of bits required to binary encode $\mathrm{Prob}_{\mathcal{M}}$ ($\mathrm{Prob}_{\mathcal{N}}$, respectively).

Let $\mathcal{M}$ be a finite representable Markov chain with probability measure $\mathrm{P}_{\mathcal{M}}$, labeled over an alphabet Σ by a labeling function λ. Furthermore, let $\mathcal{A}$ be an ω-automaton over Σ, formalizing the requirement against we intend to verify $\mathcal{M}$. Then, the quantitative model-checking problem is to compute the probability of runs in $\mathcal{M}$ those traces are contained in the language of $\mathcal{A}$. Here, we consider the decision problem version of this problem and define formally that the *(quantitative) model-checking problem* of $\mathcal{M}$ and $\mathcal{A}$ is to decide for a given binary relation $\sim\, \in\{=, >\}$ and a threshold $\rho \in [0, 1] \cap \mathbb{Q}$ whether

$$\mathrm{P}_{\mathcal{M}}(\{\pi \in \mathrm{Runs}(\mathcal{M}) : \lambda(\pi) \in \mathrm{L}_\omega(\mathcal{A})\}) \sim \rho.$$

Remark that allowing relations $\sim\in \{<, \le, =, \ge, >\}$ does not add expressiveness to the problem we stated. When $\mathcal{M}$ is a finite-state Markov chain or a pPDA, and $\mathcal{A}$ is an NBA, the set of runs $\Pi = \{\pi \in \mathrm{Runs}(\mathcal{M}) : \lambda(\pi) \in \mathrm{L}_\omega(\mathcal{A})\}$ is measurable [22]. The same techniques as in [6] can be used to show measurability of Π when $\mathcal{A}$ is an ω-PDA. The notion *qualitative model-checking problem* is used for the instance of the model-checking problem where $\rho \in \{0, 1\}$.

For finite Markov chains and ω-regular properties, the model-checking problem is well-studied (see, e.g., [22, 10]). Also the quantitative and qualitative model-checking problem for systems modeled by pPDAs and ω-regular specification is well-understood. The following theorem summarizes the results that are relevant for this paper:

Theorem 2.1 (see [7, 11, 12]). *The quantitative model-checking problem of a pPDA $\mathcal{N}$ and an NBA $\mathcal{B}$ is solvable by an algorithm that is polynomially space-bounded in $|\mathcal{N}|$ and exponentially space-bounded in $|\mathcal{B}|$. For the qualitative model-checking problem, the latter space-bound drops to a time-bound, i.e., there is a qualitative model-checking algorithm that runs in time exponential in $|\mathcal{B}|$ and is polynomially space-bounded in $|\mathcal{N}|$. If $\mathcal{B}$ is deterministic then the qualitative model-checking problem is solvable by an algorithm that is polynomially time-bounded in $|\mathcal{B}|$ and polynomially space-bounded in $|\mathcal{N}|$.*

3. Model Checking Markov Chains against Deterministic ω-Pushdown Specifications

In this section, we detail the probabilistic model-checking problem for finite-state Markov chains and pushdown specifications given by ω-DPDAs. We provide a reduction to the model-checking problem for pPDAs and ω-regular specifications.

Let us fix a Markov chain $\mathcal{M} = (S, s_0, \mathrm{Prob}_{\mathcal{M}}, \Sigma, \lambda)$ and an ω-DPDA $\mathcal{A} = (Q, \Gamma, q_0, \gamma_0, \Sigma, \delta, Acc)$. Note that $\mathcal{A}$ accepts words over the alphabet Σ and that states of $\mathcal{M}$ are labeled with symbols in Σ. As stated above, the task of the model-checking problem is to decide whether the probability measure of the runs π in $\mathcal{M}$ with $\lambda(\pi) \in \mathrm{L}_\omega(\mathcal{A})$ meets or is greater than a given rational threshold $\rho \in [0, 1]$. We propose an automata-theoretic approach, where a product construction for $\mathcal{M}$ and $\mathcal{A}$ is employed.

Definition 3.1 (Product of MCs and ω-DPDAs). *The product of $\mathcal{M}$ and $\mathcal{A}$ is defined as the tuple $\mathcal{M}{\times}\mathcal{A} = (S{\times}Q, \Gamma, s_0q_1, \gamma_1, \mathrm{Prob}_{\mathcal{M}\times\mathcal{A}}, Q, \cdot|_Q)$ where $sq\gamma|_Q = q$ for all $sq\gamma \in S{\times}Q{\times}\Gamma^*$, $q_1\gamma_1$ is uniquely defined by $q_1\gamma_1 \in$*

$\delta(q_0\gamma_0, \lambda(s_0))$ *and* $\mathrm{Prob}_{\mathcal{M}\times\mathcal{A}}$ *is given by the following rules:*

$$\frac{\mathrm{Prob}_{\mathcal{M}}(s,t) = x \quad q\alpha \in \delta(pX, \lambda(t))}{\mathrm{Prob}_{\mathcal{M}\times\mathcal{A}}(spX, tq\alpha) = x} \qquad \frac{s \in S \quad q\alpha \in \delta(pX, \varepsilon)}{\mathrm{Prob}_{\mathcal{M}\times\mathcal{A}}(spX, sq\alpha) = 1}$$

The intuition behind the product construction is that $\mathcal{A}$ consumes a symbol $a \in \Sigma$ stepwise when $\mathcal{M}$ moves to a state labeled with a. Thus, for a run ρ in $\mathcal{M}$ satisfying the specification given by $\mathcal{A}$, there is a run π in the product, where $\pi|_S = \rho$ and $\inf(\pi|_Q) \in Acc$. If the converse holds (i.e., for all accepting π there is a ρ s.t. $\lambda(\rho)$ is accepted by $\mathcal{A}$) and $\mathcal{M}\times\mathcal{A}$ is a pPDA, this product construction enables us to reduce the model-checking problem for $\mathcal{M}$ and $\mathcal{A}$ to a model-checking problem for pPDAs. But whereas it can be shown that the product leads always to a pPDA, the first condition is not true in general. Some runs in the product might have a corresponding accepting run in $\mathcal{A}$ for a finite sequence of states in $\mathcal{M}$.

Assume $\mathcal{A}$ is an ω-DPDA and $\pi = q_0\gamma_0\ q_1\gamma_1\ ...$ is a run for the finite word $\sigma \in \Sigma^*$ in $\mathcal{A}$ such that from some position $i \in \mathbb{N}$ on, only ε-transitions are taken in π. We then say that $\mathcal{A}$ contains an *ε-loop*. Since σ is finite, σ is not an element of the language $\mathrm{L}_\omega(\mathcal{A})$, even when π is accepting. In the product $\mathcal{M}\times\mathcal{A}$, the information whether σ is finite or infinite is abstracted away. Thus, there could be runs in $\mathcal{M}\times\mathcal{A}$ which correspond to accepting runs in $\mathcal{A}$, but have no according infinite run in $\mathcal{M}$. Without providing a proof, [9] states that from an ω-DPDA containing ε-transitions one can effectively construct another ω-DPDA without ε-transitions which accepts the same language. We provide a direct proof of a weaker statement, namely that one can effectively construct an ω-DPDA which accepts the same language, but does not contain any ε-loop.

Proposition 3.2 (Removing ε-loops from ω-DPDA). *For every ω-DPDA $\mathcal{A}$ there is an effectively constructible ω-DPDA $\mathcal{A}'$ without ε-loops where $\mathrm{L}_\omega(\mathcal{A}) = \mathrm{L}_\omega(\mathcal{A}')$.*

Proof. We say that the ω-DPDA $\mathcal{A}$ contains a *local ε-loop* if there is a sequence of configurations $p_0X_0\beta_0\ p_1X_1\beta_1\ ...\ p_nX_n\beta_n$, where $p_{i+1}\alpha \in \delta(p_iX_i, \varepsilon)$ with $\alpha\beta_i = X_{i+1}\beta_{i+1}$ for all $0 \le i < n$ and $p_0X_0\beta_0 = p_nX_n$. We claim that if $\mathcal{A}$ contains an ε-loop then $\mathcal{A}$ contains a local ε-loop. Note that local ε-loops do never decrease the size of the stack when taken in a run of $\mathcal{A}$.

Let $\mathcal{A}$ contain some ε-loop, i.e., there is a run $q_0\gamma_0\ q_1\gamma_1\ ...$ for a word $\sigma \in (\Sigma \cup \{\varepsilon\})^\omega$ in $\mathcal{A}$ and an index $k \in \mathbb{N}$ such that $\sigma_j = \varepsilon$ for all $j \ge k$. W.l.o.g. let k be the minimal index with this property. Let $\min(0)$ denote the least index greater than k such that $|\gamma_j| \ge |\gamma_{\min(0)}|$ for all $j > \min(0)$. We define $\min(i{+}1)$ inductively as the least index greater than $\min(i)$ such that $|\gamma_j| \ge |\gamma_{\min(i+1)}|$ for all $j > \min(i{+}1)$. Since $Q \times \Gamma$ is finite, there exists a head pX such that for infinitely many $i \in \mathbb{N}$, this is the head of $q_{\min(i)}\gamma_{\min(i)}$. Due to the definition of $\min(\cdot)$ and a run in $\mathcal{A}$, this concludes the proof of our claim.

As $\mathcal{A}$ is deterministic, it is decidable whether $\mathcal{A}$ contains a local ε-loop by simply applying ε-rules for all possible heads until a head pX is reached which has been visited before. Then, there is a local ε-loop starting in pX. Removing the unique rule for head pX from the rule set of $\mathcal{A}$ leads to an ω-DPDA $\mathcal{A}_\varepsilon$, which accepts the same language over infinite words as $\mathcal{A}$, since every run of $\mathcal{A}$ containing a configuration with head pX corresponds to a finite word and is not contained in $\mathrm{L}_\omega(\mathcal{A})$. If $\mathcal{A}_\varepsilon$ still contains a local ε-loop, the same argument can be applied repeatedly till an ω-DPDA $\mathcal{A}'$ is obtained which does not contain any local ε-loop. The existence of such an $\mathcal{A}'$ is guaranteed, since there are only finitely many $pX \in Q\times\Gamma$. Due to the claim stated at the beginning of this proof, $\mathcal{A}'$ does not contain any ε-loop. □

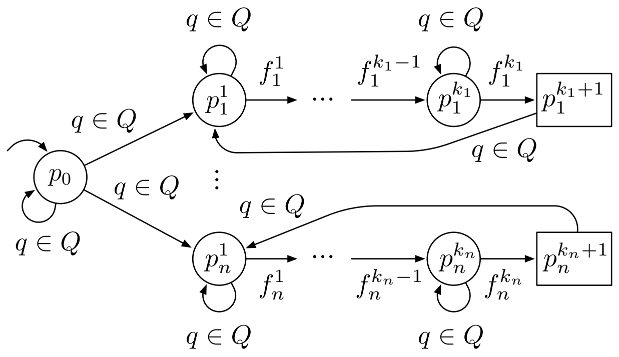

Figure 1: NBA $\mathcal{B}_{Acc}$ for Muller acceptance Acc

The proof of Proposition 3.2 shows that the size of constructed ε-loop-free ω-DPDA $\mathcal{A}'$ is bounded by the size of $\mathcal{A}$. Hence, in the sequel we can safely assume in the following that any ω-DPDA is ε-loop-free.

Assume we have already shown that $\mathcal{M}\times\mathcal{A}$ is a pPDA. Then, in order to directly apply Theorem 2.1 to solve the model-checking problem of $\mathcal{M}$ and $\mathcal{A}$, we encode the acceptance condition Acc of $\mathcal{A}$ by an NBA $\mathcal{B}_{Acc}$. Consider the Muller acceptance condition $Acc = \{F_1, F_2, ..., F_n\}$ where $F_i{=}\{f_i^1, f_i^2, ..., f_i^{k_i}\}$ for all $i{\in}\{1, ..., n\}$. We define an NBA

$$\mathcal{B}_{Acc} = (P, \{p_0\}, Q, \delta, \{p_i^{k_i+1} : 1{\le}i{\le}n\}),$$

where $P = \{p_0\} \cup \{p_i^j : 1{\le}i{\le}n, 1{\le}j{\le}k_i{+}1\}$ and the transition function is given in Figure 1. In this figure, boxes are used for accepting states and circles for non-accepting states. Transitions $p \xrightarrow{q\in Q} p'$ stand for multiple transitions $p' \in \delta(p, q)$ for all $q \in Q$. It is easy to see that using this encoding, the following proposition holds.

Proposition 3.3. *For every $\pi \in Q^\omega$ we have*

$$\inf(\pi) \in Acc \quad \textit{iff} \quad \pi \in \mathrm{L}_\omega(\mathcal{B}_{Acc}).$$

If $\mathcal{A}$ is a Büchi automaton, a simpler encoding of its acceptance condition $Acc \subseteq Q$ by a DBA is possible as shown in Figure 2.

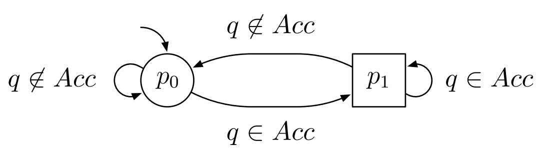

Figure 2: DBA $\mathcal{B}_{\mathrm{Acc}}$ for Büchi acceptance *Acc*

We are now ready to state a sufficient criterion on the product construction that yields the soundness of our reduction:

Definition 3.4 (Properness of the product). *The product $\mathcal{M}{\times}\mathcal{A}$ is called* proper*, if it is a pPDA and*

$$\mathrm{P}_{\mathcal{M}}(\{\pi \in \mathrm{Runs}(\mathcal{M}) : \ \lambda(\pi) \in \mathrm{L}_\omega(\mathcal{A})\}) \quad = \quad \mathrm{P}_{\mathcal{M}\times\mathcal{A}}(\{\pi \in \mathrm{Runs}(\mathcal{M}\times\mathcal{A}) : \ \pi|_Q \in \mathrm{L}_\omega(\mathcal{B}_{Acc})\})$$

Lemma 3.5 (Properness lemma). *If $\mathcal{M}$ is a finite Markov chain over Σ and $\mathcal{A}$ is a total ω-DPDA over Σ without ε-loops, then $\mathcal{M}{\times}\mathcal{A}$ is proper.*

Proof. First, we show that $\mathcal{M}{\times}\mathcal{A}$ is a pPDA. We only have to prove that for all heads $spX \in S{\times}Q{\times}\Gamma$ holds

$$\sum_{tq\alpha \in S\times Q\times\Gamma^{\leq 2}} \mathrm{Prob}_{\mathcal{M}\times\mathcal{A}}(spX, tq\alpha) \in \{0,1\}. \qquad (1)$$

Let us pick an arbitrary $spX \in S{\times}Q{\times}\Gamma$. If there is an ε-transition from pX in $\mathcal{A}$, $|\delta(pX,\varepsilon)| = 1$ and $|\delta(pX,\lambda(t))| = 0$ for all $t \in S$, since $\mathcal{A}$ is deterministic and total. Then, the sum of (1) equals 1 by the definition of $\mathcal{M}{\times}\mathcal{A}$. Otherwise, $|\delta(pX,\varepsilon)| = 0$ and for all $t \in S$ there is exactly one $q\alpha \in Q{\times}\Gamma^{\leq 2}$ with $q\alpha \in \delta(pX,\lambda(t))$ such that the sum in (1) equals $x = \sum_{t\in S} \mathrm{Prob}_{\mathcal{M}}(s,t)$. $\mathcal{M}$ is a Markov chain and thus, $x \in \{0,1\}$. Hence (1) holds and $\mathcal{M}{\times}\mathcal{A}$ is a pPDA.

Let $\Pi_{\mathcal{M}} = \{\pi \in \mathrm{Runs}(\mathcal{M}) : \lambda(\pi) \in \mathrm{L}_\omega(\mathcal{A})\}$ and $\Pi_{\mathcal{M}\times\mathcal{A}} = \{\pi \in \mathrm{Runs}(\mathcal{M}{\times}\mathcal{A}) : \pi|_Q \in \mathrm{L}_\omega(\mathcal{B}_{Acc})\}$. We have to show that

$$\mathrm{P}_{\mathcal{M}}(\Pi_{\mathcal{M}}) \quad = \quad \mathrm{P}_{\mathcal{M}\times\mathcal{A}}(\Pi_{\mathcal{M}\times\mathcal{A}}).$$

Let $\mu : \mathrm{Runs}(\mathcal{M}{\times}\mathcal{A}) {\to} \mathrm{Runs}(\mathcal{M})$ be the projection function that maps a run π in the product to the sequence of its first components, i.e., $\mu(\pi) = \pi|_S$. Clearly, $\mu(\pi)$ is a run in $\mathcal{M}$. Since $\mathcal{A}$ is deterministic and total and by the definition of the product, μ is bijective, i.e., for each run $\pi' \in \mathrm{Runs}(\mathcal{M})$ there is a unique run $\pi \in \mathrm{Runs}(\mathcal{M}{\times}\mathcal{A})$ with $\mu(\pi) = \pi'$. It suffices to show $\Pi_{\mathcal{M}} = \mu(\Pi_{\mathcal{M}\times\mathcal{A}})$, since probabilities are preserved by the product construction.

($\subseteq$): Let $\pi = s_0\, s_1 \ldots \in \Pi_{\mathcal{M}}$. Then, $\lambda(\pi) \in \mathrm{L}_\omega(\mathcal{A})$. Hence, there is an accepting infinite run $q_0\gamma_0\ q_1\gamma_1\ \ldots \in \mathrm{Runs}(\mathcal{A})$ for $\lambda(\pi)$. By the definition of the product, $\mu^{-1}(\pi) = s_0q_1\gamma_1\ s_1q_2\gamma_2\ \ldots \in \mathrm{Runs}(\mathcal{M}{\times}\mathcal{A})$ with $\inf(\mu^{-1}(\pi)|_Q) \in Acc$. Proposition 3.3 yields $\mu^{-1}(\pi)|_Q \in \mathrm{L}_\omega(\mathcal{B}_{Acc})$, and hence, $\mu^{-1}(\pi) \in \Pi_{\mathcal{M}\times\mathcal{A}}$.

($\supseteq$): Let $\pi = s_0q_1\gamma_1\ s_1q_2\gamma_2\ \ldots \in \mathrm{Runs}(\mathcal{M}{\times}\mathcal{A})$ with $\pi|_Q \in \mathrm{L}_\omega(\mathcal{B}_{Acc})$. Due to the definition of the product, $q_0\gamma_0\ q_1\gamma_1\ \ldots$ is a run in $\mathcal{A}$. By Proposition 3.3, $\inf(\pi|_Q) \in Acc$. Since $\mathcal{A}$ is ε-loop-free, $\mu(\pi)$ is an infinite run in $\mathcal{M}$ that is accepted by $\mathcal{A}$. Hence, $\lambda(\mu(\pi)) \in \mathrm{L}_\omega(\mathcal{A})$, and thus, $\mu(\pi) \in \Pi_{\mathcal{M}}$. □

Due to Lemma 3.5, model-checking algorithms for pPDAs can be used for model checking finite Markov chains against deterministic ω-pushdown specifications represented by $\mathcal{B}_{Acc}$, directly by applying Theorem 2.1. Recall that for Büchi accepting ω-DPDA, the acceptance condition can be encoded by a DBA (see Figure 2).

Theorem 3.6. *Let $\mathcal{M}$ be a finite Markov chain over Σ and $\mathcal{A}$ an ω-DPDA over Σ with control states Q and Muller acceptance condition Acc. Then, the quantitative model-checking problem of $\mathcal{M}$ and $\mathcal{A}$ can be solved by an algorithm which is polynomially space-bounded in $|\mathcal{M}|\cdot|\mathcal{A}|$ and exponentially space-bounded in $|Q|^2 \cdot |Acc|$.*

In order to solve the qualitative model-checking problem, the latter exponential space-bound drops to an exponential time-bound in $|Q|^2 \cdot |Acc|$, and even to a polynomial time-bound in $|Q| \cdot |Acc|$ if *Acc* represents a Büchi acceptance condition.

4. Model Checking pPDAs against Deterministic ω-Pushdown Specifications

It naturally arises the question whether the results of the previous section can be extended to verify pPDAs against deterministic ω-pushdown specifications. Since in the non-probabilistic case two stacks are sufficient to simulate the tape of a Turing machine [18], verification algorithms cannot be expected. However, Alur and Madhusudan [4] presented an algorithm to verify systems described by (non-probabilistic) ω-visibly pushdown automata (ω-VPA) and context-free properties described also by ω-VPAs.

First, we lift the idea of ω-VPA to probabilistic systems by introducing *visibly probabilistic pushdown automata*. For these systems and ω-DVPA specifications, the quantitative model-checking problem turns out to be decidable. Dropping the visibility condition on either the system model or specification automaton or on both leads to undecidability of the qualitative model-checking problem.

For the remainder of this section, we fix a pPDA $\mathcal{N} = (Q,\Gamma,q_0,\gamma_0,\mathrm{Prob}_{\mathcal{N}},\Sigma,\lambda)$ and an ω-DPDA $\mathcal{V} = (V,\Xi,v_0,\xi_0,\Sigma,\delta,Acc)$. In [4] it was shown that model checking a system given by an ω-VPA against an ω-VPA specification is decidable. This result relies mainly on the possibility to synchronize push, pop and local transitions. The information whether a rule is increasing, not changing or decreasing the stack size is encoded into the input alphabet. We lift these concepts into the setting of pPDA by introducing similar syntactic restrictions on pPDAs.

Definition 4.1 (pVPA). *A pPDA $\mathcal{N}$ over Σ is called a* visibly pPDA (pVPA) *if there is a partition $\widetilde{\Sigma} = [\Sigma_c,\Sigma_r,\Sigma_l]$ of Σ such that for all $pX \in Q{\times}\Gamma$ and $q\alpha \in Q{\times}\Gamma^{\leq 2}$ with* $\mathrm{Prob}_{\mathcal{N}}(pX,q\alpha) > 0$ *we have*

(push)	$\lambda(q) \in \Sigma_c$	*iff*	$\alpha = YX$ *for some* $Y \in \Gamma$
(pop)	$\lambda(q) \in \Sigma_r$	*iff*	$\alpha = \varepsilon$
(local)	$\lambda(q) \in \Sigma_l$	*iff*	$\alpha = X$

An example for pVPAs will be provided in the proof of Theorem 4.5. In contrast to ω-VPAs, the information whether a call, return or local transition is taken is not contained in the input alphabet but in the control state labeling. As for VPA, we refer to $\widetilde{\Sigma}$ as the visible alphabet of $\mathcal{N}$ and call $\mathcal{N}$ to be a pVPA over $\widetilde{\Sigma}$.

We will show that model checking pVPAs against ω-DVPA specifications is decidable using a similar product-construction approach as we proposed for model checking finite Markov chains against ω-DPDAs presented in the last section. For technical reasons, we assume w.l.o.g. that the unique element $v_1\xi_1 \in \delta(v_0\xi_0, \lambda(q_0))$ satisfies $|\gamma_0| = |\xi_1|$.

Definition 4.2 (Product of pVPA and ω-DVPA). *Let $\mathcal{N}$ be a pVPA over $\widetilde{\Sigma}'$ and $\mathcal{V}$ be a total ω-DVPA over $\widetilde{\Sigma}'$ such that $\widetilde{\Sigma} \subseteq \widetilde{\Sigma}'$. Then, the product of $\mathcal{N}$ and $\mathcal{V}$ is defined as the tuple*

$$\mathcal{N}\times\mathcal{V} = (Q\times V, \Gamma\times\Xi, q_0v_1, \langle\gamma_0,\xi_1\rangle, \mathrm{Prob}_{\mathcal{N}\times\mathcal{V}}, Q\times V, \cdot|_V),$$

where $qv\langle\gamma,\xi\rangle|_V = v$ for all $qv\langle\gamma,\xi\rangle \in Q\times V\times(\Gamma\times\Xi)^*$, $v_1\xi_1$ is uniquely defined by $v_1\xi_1 \in \delta(v_0\xi_0, \lambda(q_0))$ and $\mathrm{Prob}_{\mathcal{N}\times\mathcal{V}}$ is given by the following rules:

(push) $$\frac{\mathrm{Prob}_{\mathcal{N}}(pX, qYX) = x \ \wedge\ wY'X' \in \delta(vX', \lambda(q))}{\mathrm{Prob}_{\mathcal{N}\times\mathcal{V}}(pv\langle X,X'\rangle, qw\langle Y,Y'\rangle\langle X,X'\rangle) = x}$$

(pop) $$\frac{\mathrm{Prob}_{\mathcal{N}}(pX, q\varepsilon) = x \ \wedge\ w\varepsilon \in \delta(vX', \lambda(q))}{\mathrm{Prob}_{\mathcal{N}\times\mathcal{V}}(pv\langle X,X'\rangle, qw\varepsilon) = x}$$

(local) $$\frac{\mathrm{Prob}_{\mathcal{N}}(pX, qX) = x \ \wedge\ wX' \in \delta(vX', \lambda(q))}{\mathrm{Prob}_{\mathcal{N}\times\mathcal{V}}(pv\langle X,X'\rangle, qw\langle X,X'\rangle) = x}$$

Observe that the argumentation that every ω-VPA can be effectively transformed into a total ω-VPA accepting the same language departs slightly from the argumentation for ω-PDAs. Beside introducing a trap control state, one has to take care of the syntactical restrictions on the stack operations. Furthermore, ω-VPAs do not contain any ε-transitions. We hence can assume total and ε-loop-free ω-VPAs in the following.

We use the same definition of proper product constructions as in Definition 3.4 but for pPDAs $\mathcal{N}$ instead of Markov chains $\mathcal{M}$ and ω-DVPAs $\mathcal{V}$ instead of ω-DPDAs $\mathcal{A}$. Using an analogous proof as stated in Lemma 3.5 and by stressing the synchronization of stack operations of $\mathcal{N}$ and $\mathcal{V}$ over compatible visible alphabets, it turns out that $\mathcal{N} \times \mathcal{V}$ is proper.

Lemma 4.3 (Properness lemma (second version)). *Let $\mathcal{N}$ be a pVPA over $\widetilde{\Sigma}$ and $\mathcal{V}$ an ω-DVPA over $\widetilde{\Sigma}'$ with $\widetilde{\Sigma} \subseteq \widetilde{\Sigma}'$. Then, $\mathcal{N}\times\mathcal{V}$ is proper.*

Due to this lemma, we can follow the same argumentation which led to Theorem 3.6. Algorithms to solve model-checking problems for pPDAs then can also be used to model check pVPAs against ω-DVPA specifications, directly applying Theorem 2.1.

Theorem 4.4. *Let $\mathcal{N}$ be a pVPA over $\widetilde{\Sigma}$ and $\mathcal{V}$ an ω-DVPA over $\widetilde{\Sigma}'$ such that $\widetilde{\Sigma} \subseteq \widetilde{\Sigma}'$. Then, the probabilistic model-checking problems for $\mathcal{N}$ and $\mathcal{V}$ can be solved within the same complexities as in Theorem 3.6.*

It is worth noting that this theorem does not cover the results of Theorem 3.6, although every finite Markov chain is also a pVPA. The reason is the expressiveness of the specification formalism: the class of ω-DPDA specifications is strictly larger than the class of ω-DVPA specifications.

The next theorem stresses that decidability of checking pVPAs against ω-DVPAs mainly relies on the fact that push, pop or local actions are synchronized through compatible visible alphabets.

Theorem 4.5 (Undecidability). *The qualitative model-checking problem for a pVPA $\mathcal{N}$ and an ω-DVPA $\mathcal{V}$ is undecidable, even when both are defined over the same alphabet.*

Proof. We show that the qualitative model-checking problem for $\mathcal{N}$ and $\mathcal{V}$ can be reduced from the halting problem of two-counter Minsky machines. This problem is known to be undecidable due to [18]. A *two-counter Minsky machine* is a tuple $\mathcal{Y} = (Loc, \mathrm{Instr})$ where $Loc = \{l_1, ..., l_n\}$ is a set of *locations* and Instr is a function, which assigns an *instruction* to every location. An instruction is defined over two counters c_1 and c_2 and is either an *increment*, a *test and decrement* or a *halt* instruction. Every increment instruction is of the form $\mathrm{inc}(c, l_j)$ (counter c is incremented and the Minsky machine jumps to location l_j), every test and decrement instruction is of the form $\mathrm{tdec}(c, l_j, l_k)$ (if counter c is zero, jump to l_j - otherwise decrement c and jump to l_k), and every halt instruction is of the form halt (stay in the current location) for $c \in \{c_1, c_2\}$ and $1 \leq j, k \leq n$. A *configuration* of $\mathcal{Y}$ is a tuple $le_1e_2 \in Loc\times\mathbb{N}\times\mathbb{N}$ where l stands for the current location, and e_i for the current value of counter c_i, $i \in \{1, 2\}$. The set of configurations of $\mathcal{Y}$ is denoted by $\mathrm{Conf}(\mathcal{Y})$. The transition relation $\rightarrow$ is the least binary relation on Conf given by the rules shown Figure 3.

A *run* of $\mathcal{Y}$ is a sequence of configurations $y_0\, y_1 \ldots y_m \in \mathrm{Conf}(\mathcal{Y})$, where $y_0 = l_100$ and $y_i \rightarrow y_{i+1}$ for all $i \in \{0, ..., m-1\}$. Such a run is called a *halting run* if $y_m = le_1e_2$ for some $e_1, e_2 \in \mathbb{N}$ and $\mathrm{Instr}(l) = \mathrm{halt}$. The *halting problem* for $\mathcal{Y}$ asks whether there exists a halting run in $\mathcal{Y}$. This problem is known to be undecidable (see [18]).

The goal is to provide a reduction of the halting problem to the qualitative model-checking problem for pVPAs and ω-DVPAs.

We first construct the pVPA $\mathcal{N}_{\mathcal{Y}} = (Q, \Gamma, q_0, \gamma_0, \mathrm{Prob}_{\mathcal{N}_{\mathcal{Y}}}, \Sigma, \lambda)$ and then the ω-DVPA $\mathcal{V}_{\mathcal{Y}} = (V, \Xi, v_0, \xi_0, \Sigma, \rightarrow, Acc)$ such that $\mathrm{P}_{\mathcal{N}_{\mathcal{Y}}}(\{\pi \in \mathrm{Runs}(\mathcal{N}_{\mathcal{Y}}) : \lambda(\pi) \in \mathrm{L}_\omega(\mathcal{V}_{\mathcal{Y}})\}) > 0$ iff there is a halting run in $\mathcal{Y}$.

Let $Q = Loc\times\{\bullet, \circ, \varepsilon\}^2$, $\Gamma = \{\bullet, \#\}$, $q_0\gamma_0 = \langle l_1, \varepsilon, \varepsilon\rangle\#$,

$$\frac{le_1e_2 \in \mathrm{Conf}(\mathcal{Y}) \ \wedge\ \mathrm{Instr}(l) = \mathrm{inc}(c_1, l')}{le_1e_2 \longrightarrow l'(e_1+1)e_2}\ \textbf{(inc1)} \qquad \frac{le_1e_2 \in \mathrm{Conf}(\mathcal{Y}) \ \wedge\ \mathrm{Instr}(l) = \mathrm{inc}(c_2, l')}{le_1e_2 \longrightarrow l'e_1(e_2+1)}\ \textbf{(inc2)}$$

$$\frac{l0e_2 \in \mathrm{Conf}(\mathcal{Y}) \ \wedge\ \mathrm{Instr}(l) = \mathrm{tdec}(c_1, l', l'')}{l0e_2 \longrightarrow l'0e_2}\ \textbf{(zero1)} \qquad \frac{le_1e_2 \in \mathrm{Conf}(\mathcal{Y}) \ \wedge\ e_1 \neq 0 \ \wedge\ \mathrm{Instr}(l) = \mathrm{tdec}(c_1, l', l'')}{le_1e_2 \longrightarrow l''(e_1-1)e_2}\ \textbf{(dec1)}$$

$$\frac{le_10 \in \mathrm{Conf}(\mathcal{Y}) \ \wedge\ \mathrm{Instr}(l) = \mathrm{tdec}(c_2, l', l'')}{le_10 \longrightarrow l'e_10}\ \textbf{(zero2)} \qquad \frac{le_1e_2 \in \mathrm{Conf}(\mathcal{Y}) \ \wedge\ e_2 \neq 0 \ \wedge\ \mathrm{Instr}(l) = \mathrm{tdec}(c_2, l', l'')}{le_1e_2 \longrightarrow l''e_1(e_2-1)}\ \textbf{(dec2)}$$

Figure 3: Transition rules for the two-counter Minsky machine $\mathcal{Y}$

$$\frac{\langle l,\alpha,\beta\rangle X \in Q \times \Gamma \ \wedge\ \mathrm{Instr}(l) = \mathrm{inc}(c_1, l')}{\mathrm{Prob}_{\mathcal{N}}(\langle l,\alpha,\beta\rangle X, \langle l', \bullet, \varepsilon\rangle \bullet X) = 1}\ \textbf{(inc1)}_{\mathcal{N}} \qquad \frac{\langle l,\alpha,\beta\rangle X \in Q \times \Gamma \ \wedge\ \mathrm{Instr}(l) = \mathrm{inc}(c_2, l')}{\mathrm{Prob}_{\mathcal{N}}(\langle l,\alpha,\beta\rangle X, \langle l', \varepsilon, \bullet\rangle X) = 1}\ \textbf{(inc2)}_{\mathcal{N}}$$

$$\frac{\langle l,\alpha,\beta\rangle \in Q \ \wedge\ \mathrm{Instr}(l) = \mathrm{tdec}(c_1, l', l'')}{\begin{array}{c}\mathrm{Prob}_{\mathcal{N}}(\langle l,\alpha,\beta\rangle \#, \langle l', \varepsilon, \varepsilon\rangle \#) = 1 \\ \mathrm{Prob}_{\mathcal{N}}(\langle l,\alpha,\beta\rangle \bullet, \langle l'', \circ, \varepsilon\rangle \varepsilon) = 1\end{array}}\ \textbf{(tdec1)}_{\mathcal{N}} \qquad \frac{\langle l,\alpha,\beta\rangle X \in Q \times \Gamma \ \wedge\ \mathrm{Instr}(l) = \mathrm{tdec}(c_2, l', l'')}{\begin{array}{c}\mathrm{Prob}_{\mathcal{N}}(\langle l,\alpha,\beta\rangle X, \langle l', \varepsilon, \varepsilon\rangle X) = \frac{1}{2} \\ \mathrm{Prob}_{\mathcal{N}}(\langle l,\alpha,\beta\rangle X, \langle l'', \varepsilon, \circ\rangle X) = \frac{1}{2}\end{array}}\ \textbf{(tdec2)}_{\mathcal{N}}$$

$$\frac{\langle l,\alpha,\beta\rangle X \in Q \times \Gamma \ \wedge\ \mathrm{Instr}(l) = \mathrm{halt}}{\mathrm{Prob}_{\mathcal{N}}(\langle l,\alpha,\beta\rangle X, \langle l, \varepsilon, \varepsilon\rangle X) = 1}\ \textbf{(halt)}_{\mathcal{N}}$$

Figure 4: Transition probabilities for $\mathcal{N}_{\mathcal{Y}}$

$\Sigma = Q$ and $\lambda(q) = q$. We define $\mathrm{Prob}_{\mathcal{N}_{\mathcal{Y}}}$ according to the rules in Figure 4. $\mathcal{N}_{\mathcal{Y}}$ is in fact a pVPA, due to the visible alphabet $\Sigma_{\mathcal{N}_{\mathcal{Y}}} = [\Sigma^{\bullet}_{\mathcal{N}_{\mathcal{Y}}}, \Sigma^{\circ}_{\mathcal{N}_{\mathcal{Y}}}, \Sigma^{\varepsilon}_{\mathcal{N}_{\mathcal{Y}}}]$ where $\Sigma^{\alpha}_{\mathcal{N}_{\mathcal{Y}}}$ is defined as $\{\langle l,\alpha,\beta\rangle : l \in \mathit{Loc}, \beta \in \{\bullet,\circ,\varepsilon\}\}$. The main idea is to let $\mathcal{N}_{\mathcal{Y}}$ generate all possible runs only restricted by the counter c_1 simulated by the stack of $\mathcal{N}_{\mathcal{Y}}$. The operations on the two counters are furthermore encoded into the control state of $\mathcal{N}_{\mathcal{Y}}$, i.e., $\langle l,\alpha,\beta\rangle$ stands for an operation α on c_1 and β on c_2. An operation $\bullet$ increases (push/call), $\circ$ decreases (pop/return) and ε does not change the respective counter (local). $\mathcal{V}_{\mathcal{Y}}$ then simulates counter c_2 using the information encoded in the control states of $\mathcal{N}_{\mathcal{Y}}$ and is defined by $V = \mathit{Loc} \cup \{t\}$, $\Xi = \{\bullet, \#\}$, $v_0\xi_0 = l_1\#$, $\Sigma = Q$ and $\mathit{Acc} = \{l \in \mathit{Loc} : \mathrm{Instr}(l) = \mathrm{halt}\}$. (We only consider Büchi accepting $\mathcal{V}_{\mathcal{Y}}$, since Büchi acceptance is an instance of Muller acceptance). $\rightarrow$ is defined according to the rules in Figure 5 and by adding missing transitions to the trap state t making $\mathcal{V}_{\mathcal{Y}}$ total. $\mathcal{V}_{\mathcal{Y}}$ is an ω-VPA due to the visible alphabet $\Sigma_{\mathcal{V}_{\mathcal{Y}}} = [\Sigma^{\bullet}_{\mathcal{V}_{\mathcal{Y}}}, \Sigma^{\circ}_{\mathcal{V}_{\mathcal{Y}}}, \Sigma^{\varepsilon}_{\mathcal{V}_{\mathcal{Y}}}]$ where $\Sigma^{\beta}_{\mathcal{V}_{\mathcal{Y}}}$ is defined as $\{\langle l,\alpha,\beta\rangle : l \in \mathit{Loc}, \alpha \in \{\bullet,\circ,\varepsilon\}\}$. Note that $\mathcal{V}_{\mathcal{Y}}$ is in fact total and that $\Sigma_{\mathcal{V}_{\mathcal{Y}}} = \Sigma_{\mathcal{N}_{\mathcal{Y}}}$, i.e., $\mathcal{N}_{\mathcal{Y}}$ and $\mathcal{V}_{\mathcal{Y}}$ are both defined over the same input alphabet but have different visible alphabets.

Now, it is left to show that $\mathrm{P}_{\mathcal{N}_{\mathcal{Y}}}(\{\pi \in \mathrm{Runs}(\mathcal{N}_{\mathcal{Y}}) : \lambda(\pi) \in \mathrm{L}_{\omega}(\mathcal{V}_{\mathcal{Y}})\}) > 0$ iff there is a halting run in $\mathcal{Y}$.

($\Rightarrow$): There is a run $\pi = q_0\gamma_0\ q_1\gamma_1\ ... \in \mathrm{Runs}(\mathcal{N}_{\mathcal{Y}})$ and an accepting run $v_0\xi_0\ v_1\xi_1\ ... \in \mathrm{Runs}(\mathcal{V}_{\mathcal{Y}})$ for $\lambda(\pi)$. Due to the acceptance condition of $\mathcal{V}_{\mathcal{Y}}$ and the rules $\textbf{(halt)}_{\mathcal{N}}$ in Figure 4 and $\textbf{(trust)}_{\mathcal{V}}$ in Figure 5, there exists an index $k \in \mathbb{N}$ such that $q_k = \langle l, \#\rangle$ with $\mathrm{Instr}(l) = \mathrm{halt}$. We define a sequence $y_0\, y_1\, ...\, y_k \in \mathrm{Conf}(\mathcal{Y})$ of configurations in $\mathcal{Y}$ by setting $y_i = v_i(|\gamma_i| - 1)(|\xi_i| - 1)$. By induction, it is easy to prove that $y_0\, y_1\, ...\, y_k$ is a run in $\mathcal{Y}$ using the transition rules given in Figure 4 and Figure 5. This run is a halting run, since $v_k{=}l$ with $\mathrm{Instr}(l){=}\mathrm{halt}$.

($\Leftarrow$): Let $y_0\, y_1\, ...\, y_k \in \mathrm{Conf}(\mathcal{Y})$ be a halting run in $\mathcal{Y}$ with $y_i = l^i e_1^i e_2^i$ for $0{\leq}i{\leq}k$. Note that $\mathrm{Instr}(l^k) = \mathrm{halt}$. Then there is a run $\pi = q_0\gamma_0\ q_1\gamma_1\ ... \in \mathrm{Runs}(\mathcal{N}_{\mathcal{Y}})$ such that q_i is of the form $\langle l^i, X_i\rangle$ for all $0{\leq}i{\leq}k$ and q_i is $\langle l^k, \#\rangle$ for all $i{>}k$. This yields $\mathrm{P}_{\mathcal{N}_{\mathcal{Y}}}(\pi){>}0$. Observe that by definition of the transition rules given in Figure 4 and Figure 5 and the totality of $\mathcal{V}$, it is easy to show that $\lambda(\pi)$ is accepted by $\mathcal{V}_{\mathcal{Y}}$. Hence, $\mathrm{P}_{\mathcal{N}_{\mathcal{Y}}}(\{\pi{\in}\mathrm{Runs}(\mathcal{N}_{\mathcal{Y}}) : \lambda(\pi){\in}\mathrm{L}_{\omega}(\mathcal{V}_{\mathcal{Y}})\}) > 0$. ☐

The proof of Theorem 4.5 can be easily modified to prove a corresponding undecidability result for the model-checking problem of ω-DVPA systems against ω-DVPA specifications when their visible alphabets are not compatible. In essential, for the above reduction the set of control states is changed from $\mathit{Loc} \times \{\bullet,\circ,\varepsilon\}^2$ to Loc and for each transition rule $\mathrm{Prob}_{\mathcal{N}}(\langle l,\alpha,\beta\rangle X, \langle l',\alpha',\beta'\rangle\gamma) > 0$ we introduce a transition $l \xrightarrow{\langle l',\alpha',\beta'\rangle} l'$.

5. Conclusion and Discussion

To the best of our knowledge, this paper is the first approach that studies verification problems for probabilistic systems and pushdown specifications, i.e., context-free specifications given by pushdown automata. Our algorithms rely on reductions to model-checking problems for pPDAs that have been studied in the literature. In particular, we provided reductions for finite Markov chains and deterministic pushdown specifications over infinite words and for probabilistic visibly pushdown automata systems and deterministic visibly pushdown specifications. The

$$\frac{lX \in \mathit{Loc} \times \Xi \quad \wedge \quad \mathrm{Instr}(l) \neq \mathrm{tdec}(c_2, \dot{l}, \ddot{l})}{\begin{array}{ccc} lX & \xrightarrow{\langle l',\alpha,\bullet\rangle} & l'\bullet X \\ lX & \xrightarrow{\langle l',\alpha,\circ\rangle} & l'\varepsilon \\ lX & \xrightarrow{\langle l',\alpha,\varepsilon\rangle} & l'X \end{array}} \; (\mathbf{trust})_{\mathcal{V}} \qquad \frac{l \in \mathit{Loc} \quad \wedge \quad \mathrm{Instr}(l) = \mathrm{tdec}(c_2, l', l'')}{\begin{array}{ccc} l\bullet & \xrightarrow{\langle l'',\alpha,\circ\rangle} & l''\varepsilon \\ l\# & \xrightarrow{\langle l',\alpha,\varepsilon\rangle} & l'\# \end{array}} \; (\mathbf{tdec2})_{\mathcal{V}}$$

Figure 5: Transition rules for $\mathcal{V}_{\mathcal{Y}}$

upper complexity bounds presented here are direct consequences of the proposed reductions. It is an open question whether these bounds can be improved with specialized algorithms. It seems to be difficult finding a lower bound for the discussed probabilistic model-checking problem, since already much effort has been taken to find the exact lower bound for the probabilistic model-checking problem of pPDA. Due to the known reduction to the square-root-sum problem, it is very likely that the polynomial space bound is tight for model checking pPDAs [12]. This polynomial reduction can be also established for our probabilistic model-checking problems using minor modifications on the reduction proof presented in [12].

Other open questions are whether our approach can be extended to verify Markov decision processes against deterministic pushdown specifications or to verify Markov chains against arbitrary ω-context-free specifications represented by nondeterministic pushdown automata.

Acknowledgments. The authors would like to thank Marcus Größer for his support and fruitful discussions.

References

[1] Abdulla, P. A., Bertrand, N., Rabinovich, A. M., Schnoebelen, P., 2005. Verification of probabilistic systems with faulty communication. Information and Computation 202 (2), 141–165.

[2] Abdulla, P. A., Henda, N. B., Mayr, R., 2007. Decisive Markov chains. Logical Methods in Computer Science 3 (4).

[3] Alur, R., Etessami, K., Madhusudan, P., 2004. A temporal logic of nested calls and returns. In: Proc. 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). Vol. 2988 of Lecture Notes in Computer Science. pp. 467–481.

[4] Alur, R., Madhusudan, P., 2004. Visibly pushdown languages. In: Thirty-sixth annual ACM symposium on Theory of computing. STOC '04. ACM, New York, NY, USA, pp. 202–211.

[5] Baier, C., Katoen, J.-P., 2008. Principles of model checking. The MIT Press.

[6] Brazdil, T., 2008. Verification of probabilistic recursive sequential programs. Ph.D. thesis, Faculty of Informatics Masaryk University.

[7] Brazdil, T., Kucera, A., Strazovsky, O., 2005. On the decidability of temporal properties of probabilistic pushdown automata. In: Proc. 22th Symposium on Theoretical Aspects of Computer Science (STACS). Vol. 3404 of Lecture Notes in Computer Science. pp. 145–157.

[8] Clarke, E. M., Grumberg, O., Peled, D. A., 1999. Model Checking. The MIT Press.

[9] Cohen, R. S., Gold, A. Y., 1978. ω-computations on Deterministic Pushdown Machines. Journal of Computer and System Sciences 16, 275–300.

[10] Courcoubetis, C., Yannakakis, M., 1995. The complexity of probabilistic verification. Journal of the ACM 42 (4), 857–907.

[11] Esparza, J., Kucera, A., Mayr, R., 2006. Model checking probabilistic pushdown automata. Logical Methods in Computer Science 2 (1), 1–31.

[12] Etessami, K., Yannakakis, M., 2005. Recursive Markov chains, stochastic grammars, and monotone systems of non-linear equations. In: STACS. Vol. 3404 of LNCS. pp. 340–352.

[13] Gupta, R., Smolka, S. A., Bhaskar, S., 1994. On randomization in sequential and distributed algorithms. ACM Computing Surveys 26(1), 7–86.

[14] Kemeny, J., Snell, J., 1969. Finite Markov chains, repr Edition. University series in undergraduate mathematics. VanNostrand, New York.

[15] Kupferman, O., Piterman, N., Vardi, M. Y., 2002. Pushdown specifications. In: LPAR. No. 2514 in LNCS. pp. 262–277.

[16] Linna, M., 1976. On ω-sets associated with context-free languages. Inform. Control 31, 272–293.

[17] Lynch, N., 1996. Distributed Algorithms. Morgan Kaufmann Series in Data Management Systems.

[18] Minsky, M. L., 1967. Computation: finite and infinite machines. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[19] Peng, W., Iyer, S. P., 1995. A new type of pushdown automata on infinite trees. Int. J. Found. Comput. Sci. 6 (2), 169–186.

[20] Staiger, L., 1997. ω-languages. In: Rozenberg, G., Salomaa, A. (Eds.), Handbook of Formal Languages. Springer-Verlag, Berlin, pp. 339–387.

[21] Vardi, M., 1985. Automatic verification of probabilistic concurrent finite-state programs. In: 26th IEEE Symposium on Foundations of Computer Science (FOCS). pp. 327–338.

[22] Vardi, M. Y., 1985. Automatic verification of probabilistic concurrent finite state programs. In: 26th Annual Symposium on Foundations of Computer Science. IEEE Computer Society, Washington, DC, USA, pp. 327–338.

[23] Vardi, M. Y., Wolper, P., Jun. 1986. An automata-theoretic approach to automatic program verification. In: 1st Annual Symposium on Logic in Computer Science (LICS'86). IEEE Comp. Soc. Press, pp. 332–344.

[24] Yannakakis, M., Etessami, K., 2005. Checking LTL properties of recursive Markov chains. In: QEST. IEEE Computer Society, Washington, DC, USA, pp. 155–164.