

Branching-Time Model-Checking of Probabilistic Pushdown Automata

Tomáš Brázdil^{a,1}, Václav Brožek^a, Vojtěch Forejt^{b,a}, Antonín Kučera^{a,1}

^a*Faculty of Informatics, Masaryk University, Czech Republic*

^b*Department of Computer Science, University of Oxford, UK*

Abstract

In this paper we study the model-checking problem for probabilistic pushdown automata (pPDA) and branching-time probabilistic logics PCTL and PCTL*. We show that model-checking pPDA against general PCTL formulae is undecidable, but we yield positive decidability results for the qualitative fragments of PCTL and PCTL*. For these fragments, we also give a complete complexity classification.

1. Introduction

Markov chains are widely used as a formal model for discrete systems with random behaviour. The properties of such systems can be expressed in suitable temporal logics such as LTL, PCTL, or PCTL*. The corresponding model-checking problem has so far been studied mainly for finite-state Markov chains (see, e.g., [3, 24]). Model-checking of various classes of infinite-state Markov chains has been taken into consideration only recently. The most prominent classes include chains generated by probabilistic lossy channel systems (see, e.g., [2, 4, 26]) and recursive probabilistic systems encoded either by probabilistic pushdown automata [13, 15] or an expressively equivalent model of recursive Markov chains [19, 20, 23]. In this paper, we concentrate on model-checking probabilistic pushdown automata (pPDA) against branching-time probabilistic logics PCTL and PCTL*.

Intuitively, a pPDA consists of a finite set of stack symbols, a finite set of control states, and finitely many transition rules of the form $pX \xrightarrow{x} q\alpha$ where p, q are control states, X is a stack symbol, α is a (possibly empty) string of stack

¹Tomáš Brázdil and Antonín Kučera are supported by the Czech Science Foundation, project No. P202/10/1469.

symbols and $x \in (0, 1]$ is a rational probability. Configurations of pPDA are of the form $p\beta$ where p is a control state and β is a string of stack symbols, i.e., β represents the current stack content where the first symbol of β is on the top of the stack. Each pPDA induces an infinite-state Markov chain whose states are configurations and transitions are induced by transition rules. More precisely, a rule $pX \xrightarrow{x} q\alpha$ is applicable to every configuration of the form $pX\beta$ and induces a transition $pX\beta \xrightarrow{x} q\alpha\beta$.

The logics PCTL and PCTL* are probabilistic variants of the well-known branching-time logics CTL and CTL*, respectively. The only difference is that the universal and existential path quantifiers of CTL and CTL* are replaced with the *probabilistic operator* $\mathcal{P}^{\sim x}(\cdot)$, where $\sim \in \{\leq, \geq, <, >, =\}$ is a comparison and $x \in [0, 1]$ a rational probability bound. Intuitively, the formula $\mathcal{P}^{\sim x}(\varphi)$, where φ is a path formula, says that the probability of all runs satisfying φ is \sim -related to ϱ . The *qualitative* fragments of PCTL and PCTL*, denoted by qPCTL and qPCTL*, consist of all formulae where the probability bound ϱ only takes the values 0 or 1. Typical properties expressible by qualitative formulae include “the program terminates with probability one”, “every request is eventually serviced with probability one”, etc. Qualitative properties of finite-state Markov-chains are stable in the sense that they are not influenced by precise values of transition probabilities (it is only important which transitions have positive/zero probability). In infinite-state Markov chains, the validity of qualitative properties may change by modifying the values of transition probabilities, but they are still more stable than general quantitative properties.

In our analysis, we put a special attention to the subclass of *stateless* pPDA (denoted by pBPA), and we also distinguish between the *combined* complexity, where an instance is given both by a pPDA configuration and a branching-time formula, and *program* complexity, where the formula is fixed. The study of program complexity is motivated by the fact that formulae in practical model-checking instances are typically small.

Our contribution

The summary of our results about the model-checking problem for pPDA/pBPA and (qualitative) PCTL/PCTL* formulae is given in Fig. 1. The last two lines classify the program complexity for an arbitrary (but fixed) formula.

Technically, this paper is based mainly on conference papers [9, 13], but some of the proofs are completely new or substantially revised. More detailed comments are given below.

	pBPA	pPDA
PCTL	?	undecidable (fixed formula)
PCTL*	undecidable (fixed formula)	undecidable (fixed formula)
qPCTL	EXPTIME -complete	EXPTIME -complete
qPCTL*	2-EXPTIME -complete	2-EXPTIME -complete
qPCTL (p.c.)	P	EXPTIME -complete
qPCTL* (p.c.)	P	EXPTIME -complete

Figure 1: The summary of results.

- The undecidability of the model-checking problem for pPDA and PCTL has been originally obtained by reduction from the halting problem for Minsky machines [13]. In this paper, we give a simpler proof by reduction from the Post correspondence problem, using the idea of encoding words into probabilities originally presented in [13]. The undecidability proof for pBPA and PCTL* is obtained as a slight modification of the construction used for pPDA and PCTL. However, we have not managed to extend this technique to pBPA and PCTL, and this decidability question is left open.
- The **2-EXPTIME** and **EXPTIME** upper bounds on model-checking pPDA against qPCTL* and qPCTL are due to [9], but the original proofs have been completely rewritten in this paper. We rely heavily on the results of [23] for LTL model-checking of recursive Markov chains (note that path formulae of PCTL* correspond to LTL formulae). We show how to compute regular sets of configurations that almost surely satisfy a given path formula with regular valuations of atomic propositions. From this we obtain a model checking algorithm for qPCTL*.
- The **2-EXPTIME** and **EXPTIME** lower bounds on model-checking pPDA against qPCTL* and qPCTL are based on standard techniques from non-probabilistic model-checking [5, 29] and we include them mainly for the sake of completeness.

Related work

The model-checking problem for pPDA was first studied in [15] (see also [17]), where it was shown that quantitative reachability for pPDA is in **EXPTIME**, and the model-checking problem for pPDA against qPCTL and deterministic Büchi specifications is decidable. The equivalent model of Recursive Markov

Chains (RMCs) was studied independently in [20] (see also [22]), and one of the main results of this work is a polynomial-time algorithm for the qualitative reachability problem for single exit RMCs (which are equivalent to pBPA). The study of pPDA was continued by considering the expected accumulated reward and its variance [16], discounted properties [7], model-checking linear-time properties [19, 23], etc. We refer to [6, 12] for a more comprehensive overview.

More loosely related to this work are papers on recursive Markov decision processes and stochastic games (see, e.g., [8, 10, 11, 21]). In particular, in [21] it is shown that the controller synthesis problem for qualitative LTL and recursive Markov decision processes is undecidable. On the other hand, in [10] it is demonstrated that the model-checking problem for BPA decision processes and qPCTL is in **EXPTIME**.

2. Preliminaries

We assume familiarity with basic notions of probability theory and automata theory such as σ -field, probability space, finite automaton, etc. We also use the standard notation for writing regular expressions.

Definition 2.1. A (discrete) Markov chain is a triple $M = (S, \rightarrow, Prob)$ where S is a finite or countably infinite set of states, $\rightarrow \subseteq S \times S$ is a transition relation such that for every $s \in S$ there is $t \in S$ with $s \rightarrow t$, and $Prob$ is a function which to each transition (s, t) assigns its probability $Prob(s, t) \in (0, 1]$ so that for every $s \in S$ we have that $\sum_{s \rightarrow t} Prob(s, t) = 1$.

We write $s \xrightarrow{x} t$ to indicate that $s \rightarrow t$ and $Prob(s, t) = x$. A path in M is a finite or infinite sequence $w = s_0 s_1 \dots$ of states such that $s_i \rightarrow s_{i+1}$ for every i . A run is an infinite path. We use Run to denote the set of all runs in M , and $Run(w)$ to denote the set of all runs that start with a given finite path w . For a given run w , we use $w(i)$ to denote the state s_i of w , and w_i to denote the run $s_i s_{i+1} \dots$ (note that $w_0 = w$). A state t is *reachable* from a state s if there is a finite path starting in s and ending in t .

To every $s \in S$ we associate the probability space $(Run(s), \mathcal{F}, \mathcal{P})$ where \mathcal{F} is the σ -field generated by all *basic cylinders* $Run(w)$ where w is a finite path initiated in s , and $\mathcal{P} : \mathcal{F} \rightarrow [0, 1]$ is the unique probability measure (see [27, 28]) such that $\mathcal{P}(Run(w)) = \prod_{i=1}^m x_i$ where $w = s_0, \dots, s_m$ and $s_{i-1} \xrightarrow{x_i} s_i$ for every $1 \leq i \leq m$ (the empty product is equal to 1).

2.1. Branching-Time Temporal Logics

We start with a definition of the temporal logic PCTL*. The syntax of PCTL* state and path formulae Φ and φ , resp., is given by the following abstract syntax equations.

$$\begin{aligned}\Phi & ::= a \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \mathcal{P}^{\sim\varrho}\varphi \\ \varphi & ::= \Phi \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \mathcal{X}\varphi \mid \varphi_1 \mathcal{U} \varphi_2\end{aligned}$$

Here a ranges over a countably infinite set Ap of atomic propositions, $\varrho \in [0, 1]$ is a rational constant, and $\sim \in \{\leq, <, \geq, >, =\}$.

We use standard abbreviations tt (always true), \vee (or) and \Rightarrow (implication), and we also define $\Diamond\varphi \equiv \text{tt} \mathcal{U} \varphi$ and $\Box\varphi \equiv \neg\Diamond\neg\varphi$. If the probabilistic operator $\mathcal{P}^{\sim x}(\cdot)$ is applied to a formula of the form $\mathcal{X}\varphi$ or $\varphi_1 \mathcal{U} \varphi_2$, we further simplify our notation by writing $\mathcal{X}^{\sim\varrho}\varphi$ and $\varphi_1 \mathcal{U}^{\sim\varrho} \varphi_2$ instead of $\mathcal{P}^{\sim x}(\mathcal{X}\varphi)$ and $\mathcal{P}^{\sim x}(\varphi_1 \mathcal{U} \varphi_2)$, respectively. The size of a given PCTL* formula τ is denoted by $|\tau|$ (we assume that τ is represented as a tree and that the probability bounds in the probabilistic operator are encoded as fractions of binary numbers). We also use $Cl(\tau)$ to denote the set of all subformulae of τ .

The logic PCTL is a fragment of PCTL* where path formulae are given by the equation $\varphi ::= \mathcal{X}\Phi \mid \Phi_1 \mathcal{U} \Phi_2$. The *qualitative fragments* of PCTL and PCTL*, denoted by qPCTL and qPCTL*, resp., are obtained by restricting the allowed operator/number combinations in $\mathcal{P}^{\sim\varrho}\varphi$ subformulae to ‘=0’ and ‘=1’ (we do not include ‘<1’, ‘>0’ because these are definable from ‘=0’, ‘=1’, and negation). Finally, a path formula φ is an *LTL formula* if all of its state subformulae are atomic propositions.

Now we define the semantics of PCTL*. Let us fix a Markov chain $M = (S, \rightarrow, Prob)$ and a valuation $\nu : Ap \rightarrow 2^S$. State formulae are interpreted over S , and path formulae are interpreted over Run . Hence, for a given $s \in S$ and $w \in Run$ we define

$$\begin{aligned}s \models^\nu a & \quad \text{iff } s \in \nu(a), \\ s \models^\nu \neg\Phi & \quad \text{iff } s \not\models^\nu \Phi, \\ s \models^\nu \Phi_1 \wedge \Phi_2 & \quad \text{iff } s \models^\nu \Phi_1 \text{ and } s \models^\nu \Phi_2, \\ s \models^\nu \mathcal{P}^{\sim\varrho}\varphi & \quad \text{iff } \mathcal{P}(\{w \in Run(s) \mid w \models^\nu \varphi\}) \sim \varrho, \\ w \models^\nu \Phi & \quad \text{iff } w(0) \models^\nu \Phi, \\ w \models^\nu \neg\varphi & \quad \text{iff } w \not\models^\nu \varphi, \\ w \models^\nu \varphi_1 \wedge \varphi_2 & \quad \text{iff } w \models^\nu \varphi_1 \text{ and } w \models^\nu \varphi_2, \\ w \models^\nu \mathcal{X}\varphi & \quad \text{iff } w_1 \models^\nu \varphi, \\ w \models^\nu \varphi_1 \mathcal{U} \varphi_2 & \quad \text{iff there is } j \geq 0 \text{ s.t. } w_j \models^\nu \varphi_2 \text{ and } w_i \models^\nu \varphi_1 \text{ for all } 0 \leq i < j.\end{aligned}$$

For PCTL, the semantics of path formulae can be defined in a simplified (but equivalent) way as follows:

$$\begin{aligned} w \models^\nu \mathcal{X}\Phi & \quad \text{iff } w(1) \models^\nu \Phi, \\ w \models^\nu \Phi_1 \mathcal{U} \Phi_2 & \quad \text{iff there is } j \geq 0 \text{ s.t. } w(j) \models^\nu \Phi_2 \text{ and } w(i) \models^\nu \Phi_1 \text{ for all } 0 \leq i < j. \end{aligned}$$

2.2. Probabilistic pushdown automata

Probabilistic pushdown automata [15] are a fully probabilistic variant of the classical model deeply studied in automata theory (see, e.g., [25]). They are expressively equivalent to Recursive Markov Chains (RMC), and there are linear-time translations between the two models [20].

Definition 2.2. A probabilistic pushdown automaton (pPDA) is a tuple $\Delta = (Q, \Gamma, \delta, \text{Prob})$ where Q is a finite set of control states, Γ is a finite stack alphabet, $\delta \subseteq Q \times \Gamma \times Q \times \Gamma^{\leq 2}$ (where $\Gamma^{\leq 2} = \{\alpha \in \Gamma^*, |\alpha| \leq 2\}$) is a set of transition rules (we write $pX \rightarrow q\alpha$ instead of $(p, X, q, \alpha) \in \delta$), and Prob is a function which to each transition rule $pX \rightarrow q\alpha$ assigns a rational probability $\text{Prob}(pX \rightarrow q\alpha) \in (0, 1]$ so that for all $p \in Q$ and $X \in \Gamma$ we have that $\sum_{pX \rightarrow q\alpha} \text{Prob}(pX \rightarrow q\alpha) = 1$.

We assume that for each $pX \in Q \times \Gamma$ there is at least one $q\alpha$ such that $pX \rightarrow q\alpha$, and we write $pX \xrightarrow{x} q\alpha$ to indicate that $pX \rightarrow q\alpha$ and $\text{Prob}(pX \rightarrow q\alpha) = x$. The set $Q \times \Gamma^*$ of all configurations of Δ is denoted by $\mathcal{C}(\Delta)$. The *head* of a configuration $p\gamma$ is either pX or p , depending on whether $\gamma = X\alpha$ or $\gamma = \varepsilon$, respectively (where ε denotes the empty word).

An important subclass of pPDA are *stateless* pPDA, where the set of control states is a singleton. For simplicity, the configurations of stateless pPDA are written without the only control state (for example, we write just XY instead of pXY). In the rest of this paper, the subclass of stateless pPDA is denoted by pBPA². The pBPA subclass exactly corresponds to 1-exit RMC [20].

To Δ we associate a Markov chain M_Δ where $\mathcal{C}(\Delta)$ is the set of states and the transitions are determined as follows:

- $p\varepsilon \xrightarrow{1} p\varepsilon$ for each $p \in Q$;
- $pX\beta \xrightarrow{x} q\alpha\beta$ is a transition of M_Δ iff $pX \xrightarrow{x} q\alpha$ is a transition rule of Δ .

²The BPA acronym stands for ‘‘Basic Process Algebra’’ and it is used mainly for historical reasons.

The logics PCTL and PCTL* can be interpreted over M_Δ only after fixing some valuation $\nu : Ap \rightarrow 2^{\mathcal{C}(\Delta)}$. General valuations are not apt for algorithmic analysis, because one can easily encode undecidable problems for PDA into suitable valuations. Therefore, we restrict our attention to *regular* valuations that can be encoded by finite-state automata. More precisely, let \mathcal{A} be a deterministic finite-state automaton³ (DFA) over the alphabet $Q \cup \Gamma$. The set of all configurations *encoded* by \mathcal{A} , denoted by $\mathcal{C}(\mathcal{A})$, consists of all $p\alpha \in \mathcal{C}(\Delta)$ such that the *reverse* of $p\alpha$ is in $\mathcal{L}(\mathcal{A})$. In other words, \mathcal{A} reads the stack from bottom up. Note that we do not require $\mathcal{L}(\mathcal{A}) \subseteq \Gamma^*Q$.

Definition 2.3. *Let $\Delta = (Q, \Gamma, \delta, Prob)$ be a pPDA, and $\nu : Ap \rightarrow 2^{\mathcal{C}(\Delta)}$ a valuation. We say that ν is regular if for each $a \in Ap$ there is a DFA \mathcal{A}_a such that $\nu(a) = \mathcal{C}(\mathcal{A}_a)$. Further, we say that ν is simple if for each $a \in Ap$ there is a subset of heads $H_a \subseteq Q \cup (Q \times \Gamma)$ such that $p\alpha \in \nu(a)$ iff the head of $p\alpha$ is in H_a .*

Obviously, every simple valuation is regular, and we formally consider simple valuations as a special type of regular valuations (in particular, we use DFA to encode simple valuations).

An instance of the *model-checking* problem for pPDA and PCTL/PCTL* is given by a pPDA Δ , a configuration $p\alpha \in \mathcal{C}(\Delta)$, a PCTL/PCTL* state formula Φ , and a regular valuation ν . The question is whether $p\alpha \models^\nu \Phi$. The size of Δ and Φ is the length of corresponding binary encoding, where the (rational) probabilities and probability constraints are encoded as fractions of binary numbers. The regular valuation ν is represented by a finite collection of all \mathcal{A}_a such that a occurs in Φ (see Definition 2.3), and the size of ν is the total size of all these DFA.

3. Undecidability of Quantitative Branching-time Model-checking

In this section we prove the undecidability results given in Fig. 1.

Theorem 3.1. *There is a fixed PCTL formula Φ such that the model-checking problem for pPDA and Φ is undecidable. Further, there is a fixed PCTL* formula Ψ such that the model-checking problem for pBPA and Ψ is undecidable.*

³For purposes of this paper, the transition function in DFA is formally defined as a *total* function. This is convenient in, e.g., Definition 4.7 where we simulate a tuple of DFA on-the-fly in the stack, and the totality of transition functions in these DFA guarantees that the simulation cannot get stuck.

Both parts of Theorem 1 are proved by reduction from (a slightly modified version of) the Post’s correspondence problem (PCP). An instance of PCP consists of two sequences x_1, \dots, x_n and y_1, \dots, y_n of words over the alphabet $\Sigma = \{A, B, \bullet\}$ such that all x_i and y_j have the same length m . The question is whether there is a finite sequence i_1, \dots, i_k of indexes such that $x_{i_1} \cdots x_{i_k}$ and $y_{i_1} \cdots y_{i_k}$ are the same words after erasing all occurrences of “ \bullet ”.

3.1. pPDA and PCTL model-checking

In this subsection we prove the first part of Theorem 3.1. Let x_1, \dots, x_n and y_1, \dots, y_n be an instance of PCP, where all x_i and y_j have the same length m . We construct a pPDA Δ where

- the number of control states is $\mathcal{O}(m \cdot n)$, and there are always three distinguished control states g, c , and t ;
- the stack alphabet of Δ is fixed and contains the symbols Z, Y , and nine symbols of the form (z, z') where $z, z' \in \Sigma$.

Further, $\Phi \equiv \diamond^{>0}(cZ \wedge \diamond^{=1/2}tY)$ where the atomic propositions cZ and tY are valid in exactly all configurations with head cZ and tY , respectively. Our construction ensures that the given PCP instance has a solution iff $gZ \models^\nu \Phi$, where ν is the simple valuation described above (from now on, we omit the ν superscript in \models^ν).

Now we describe the pPDA Δ in greater detail. To simplify our presentation, we specify only the “relevant” transition rules of Δ . Note that according to Definition 2.2, there should be at least one transition rule for every $pX \in Q \times \Gamma$, even if no configuration with head pX is reachable from the initial configuration gZ (which makes the rules for pX irrelevant). Formally, if we do not give any explicit rule for pX in our construction below, we assume the only default rule $pX \xrightarrow{1} pX$. We also use $\mathcal{P}(p\alpha, \varphi)$ to denote the probability of all runs initiated in $p\alpha$ satisfying the path formula φ .

From the initial configuration gZ , the pPDA Δ tries to “guess” a solution to our PCP instance by storing pairs of words (x_i, y_i) successively on the stack. Since x_i and y_i have the same length m , this is implemented by pushing pairs of letters from Σ . For example, if $x_i = AAB$ and $y_i = BA\bullet$, then the pair (x_i, y_i) is stored as a sequence of three stack symbols $(A, B), (A, A), (B, \bullet)$, where (B, \bullet) is on the top of the stack. After storing a chosen pair of words, the automaton can either go on with guessing another pair of words, or enter a *checking* configuration by changing the control state from g to c and pushing the symbol Z on the top of

the stack. The transition probabilities do not matter here, and hence we assume they are distributed uniformly. This “guessing phase” is formalized below. Since transition probabilities are distributed uniformly, we do not write them explicitly. The symbol “|” separates alternatives.

$$\begin{aligned} gX &\rightarrow g_1^1 X \mid \cdots \mid g_n^1 X, \\ g_i^j X &\rightarrow g_i^{j+1}(x_i(j), y_i(j))X, \\ g_i^{m+1} X &\rightarrow cZX \mid gX \end{aligned}$$

Here $1 \leq \ell \leq n$, $1 \leq j \leq m$, $x_i(j)$ and $y_i(j)$ denote the j^{th} letters in x_i and y_i , respectively, and X ranges over the stack alphabet. The following lemma is immediate.

Lemma 3.2. *A configuration of the form $cZ\alpha$ is reachable from gZ iff $\alpha \equiv (z_1, z'_1) \cdots (z_\ell, z'_\ell)Z$ and there is a sequence i_1, \dots, i_k such that $z_\ell \cdots z_1 = x_{i_1} \cdots x_{i_k}$ and $z'_\ell \cdots z'_1 = y_{i_1} \cdots y_{i_k}$.*

The crucial part of the construction is the next phase which verifies that the guess was correct, i.e., that the words stored in the first and the second component of stack symbols are the same when “•” is disregarded. For this we use the following transition rules (again, the transition probabilities are distributed uniformly):

$$\begin{aligned} cZ &\rightarrow f\varepsilon \mid s\varepsilon, & f(A, z) &\rightarrow tY \mid f\varepsilon, & s(z, A) &\rightarrow rY \mid s\varepsilon, & tY &\rightarrow tY, \\ & & f(B, z) &\rightarrow rY \mid f\varepsilon, & s(z, B) &\rightarrow tY \mid s\varepsilon, & rY &\rightarrow rY \\ & & f(\bullet, z) &\rightarrow f\varepsilon, & s(z, \bullet) &\rightarrow s\varepsilon, & & \\ & & fZ &\rightarrow tY \mid rY, & sZ &\rightarrow tY \mid rY, & & \end{aligned}$$

Here z ranges over Σ . We claim that a checking configuration satisfies the formula $\diamond^{=1/2} tY$ iff the previous guess was correct. To get some intuition, let us evaluate the probability of reaching a configuration with the head tY for, e.g., a configuration $cZ(A, A)(A, \bullet)(\bullet, A)(B, B)Z$. First, the symbol Z is erased from the top of the stack and the control state is changed to either f or s , which intuitively means that we are going to read the letters stored either in the first or in the second component of stack symbols, respectively. Then, the stack symbols are popped one by one according to the above transition rules. One can easily confirm that the probability of reaching a configuration with the head tY is equal to

$$\frac{1}{2} \left(\left(1 \cdot \frac{1}{2} + 1 \cdot \frac{1}{2^2} + 0 \cdot \frac{1}{2^3} + 1 \cdot \frac{1}{2^4} \right) + \left(0 \cdot \frac{1}{2} + 0 \cdot \frac{1}{2^2} + 1 \cdot \frac{1}{2^3} + 1 \cdot \frac{1}{2^4} \right) \right)$$

which can be written in binary as follows: $\frac{1}{2}(0.1101 + 0.0011)$. The first three digits after the binary point in 0.1101 and 0.0011 reflect the structure of the words $AA\bullet B$ and $A\bullet AB$ stored in the stack, and the last 1 is due to the Z at the very bottom. Note that the role of A in the two words is dual—in the first case, it generates 1's, and in the second case it generates 0's (similarly for B). The symbol \bullet is just popped from the stack with probability one, which does not influence the probability of reaching a configuration satisfying tY . Note that the probabilities 0.1101 and 0.0011 are “complementary” and their sum is equal to 1. This “complementarity” breaks down iff the words stored in the first and the second component of stack symbols are *not* the same, in which case the sum is different from 1. Now we formalize this intuition.

Definition 3.3. For every $\alpha \in \Sigma^*$, we define the numbers $Fnum(\alpha)$ and $Snum(\alpha)$ inductively as follows:

- $Fnum(\varepsilon) = Snum(\varepsilon) = \frac{1}{2}$;
- $Fnum(X\alpha) = \begin{cases} Fnum(\alpha) & \text{if } X = \bullet, \\ \frac{1}{2} \cdot Fnum(\alpha) & \text{if } X = B, \\ \frac{1}{2} + \frac{1}{2} \cdot Fnum(\alpha) & \text{if } X = A; \end{cases}$
- $Snum(X\alpha) = \begin{cases} Snum(\alpha) & \text{if } X = \bullet, \\ \frac{1}{2} \cdot Snum(\alpha) & \text{if } X = A, \\ \frac{1}{2} + \frac{1}{2} \cdot Snum(\alpha) & \text{if } X = B. \end{cases}$

Given $\alpha, \beta \in \Sigma^*$, we write $\alpha \overset{\circ}{=} \beta$ iff α and β are the same words after erasing all occurrences of “ \bullet ” in α and β .

A proof of the following lemma is straightforward and uses simple properties of binary numbers.

Lemma 3.4. For all $\alpha, \beta \in \Sigma^*$ we have that $\alpha \overset{\circ}{=} \beta$ iff $Fnum(\alpha) + Snum(\beta) = 1$.

Now observe that for a checking configuration $cZ\alpha$, where $\alpha = (z_1, z'_1) \cdots (z_\ell, z'_\ell)Z$, we have that

$$\mathcal{P}(cZ\alpha, \diamond tY) = \frac{1}{2} \cdot (Fnum(z_1 \cdots z_\ell) + Snum(z'_1 \cdots z'_\ell)).$$

By applying Lemma 3.2 and 3.4, we obtain that $gZ \models \diamond^{>0}(cZ \wedge \diamond^{=1/2} tY)$ iff the PCP instance has a solution.

Remark 3.5. *There is nothing really special about the constraint $=\frac{1}{2}$ in the formula Φ . In fact, we could use any rational x strictly between 0 and 1 by making the following change to Δ . If $x < \frac{1}{2}$, we replace the transitions $cZ \xrightarrow{0.5} f\varepsilon$ and $cZ \xrightarrow{0.5} s\varepsilon$ with transitions $cZ \xrightarrow{1-2x} rY$, $cZ \xrightarrow{x} f\varepsilon$, and $cZ \xrightarrow{x} s\varepsilon$, getting*

$$\mathcal{P}(cZ\alpha, \diamond tY) = x \cdot (Fnum(z_1 \cdots z_\ell) + Snum(z'_1 \cdots z'_\ell)).$$

If $x > \frac{1}{2}$, we replace the same transitions with $cZ \xrightarrow{2x-1} tY$, $cZ \xrightarrow{1-x} f\varepsilon$, and $cZ \xrightarrow{1-x} s\varepsilon$, which yields

$$\mathcal{P}(cZ\alpha, \diamond tY) = (2x - 1) + (1 - x) \cdot (Fnum(z_1 \cdots z_\ell) + Snum(z'_1 \cdots z'_\ell)).$$

In both cases, we have that $\mathcal{P}(cZ\alpha, \diamond tY) = x$ iff $Fnum(z_1 \cdots z_\ell) + Snum(z'_1 \cdots z'_\ell) = 1$.

3.2. pBPA and PCTL* model-checking

As in the previous subsection, consider a PCP instance x_1, \dots, x_n and y_1, \dots, y_n , where the length of all x_i and y_j is m . We construct a pBPA Δ with a distinguished stack symbol Z and a PCTL* formula of the form $C \wedge \mathcal{P}^{=1/2}(\varphi_1 \vee \varphi_2)$ such that Z satisfies the formula iff the PCP instance has a solution. The stack alphabet of Δ contains the symbols Z, Z', C, F, S, G_i^j for $1 \leq i \leq n$ and $1 \leq j \leq m + 1$, and $X_{(z,z')}$ and (z, z') for all $z, z' \in \Sigma$. The subformulae φ_1 and φ_2 look as follows:

- $\varphi_1 \equiv (\neg S \wedge \bigwedge_{z \in \Sigma} \neg X_{(B,z)}) \mathcal{U} (\bigvee_{z \in \Sigma} X_{(A,z)})$,
- $\varphi_2 \equiv (\neg F \wedge \bigwedge_{z \in \Sigma} \neg X_{(z,A)}) \mathcal{U} (\bigvee_{z \in \Sigma} X_{(z,B)})$.

The atomic propositions C, F, S , and $X_{(z,z')}$, where $z, z' \in \Sigma$, are valid in exactly all configurations with the respective head.

Similarly to the pPDA constructed in the previous subsection, the pBPA Δ works in two phases. In the first phase, Δ tries to guess a solution to the PCP instance by storing pairs of words (x_i, y_i) in the stack. This is achieved by the following rules (all transition rule probabilities are again distributed uniformly):

$$\begin{aligned} Z &\rightarrow G_1^1 Z' \mid \cdots \mid G_n^1 Z', \\ G_i^j &\rightarrow G_i^{j+1}(x_i(j), y_i(j)), \\ G_i^{m+1} &\rightarrow C \mid G_1^1 \mid \cdots \mid G_n^1. \end{aligned}$$

Here $1 \leq i \leq n$ and $1 \leq j \leq m$. The next lemma says that the above rules work as expected (a proof is trivial).

Lemma 3.6. *A configuration of the form $C\alpha$ is reachable from Z iff $\alpha \equiv (z_1, z'_1) \cdots (z_\ell, z'_\ell)Z'$ where $z_1, \dots, z_\ell, z'_1, \dots, z'_\ell \in \Sigma$ and there is a sequence i_1, \dots, i_k such that $z_\ell \cdots z_1 = x_{i_1} \cdots x_{i_k}$ and $z'_\ell \cdots z'_1 = y_{i_1} \cdots y_{i_k}$.*

The second phase (verifying the guess) differs slightly from the one given in Section 3.1. For pPDA, we could use control states to distinguish between the runs that “read” the first and the second component of stack symbols. This does not work for pBPA, and the runs are distinguished by the subformulae φ_1 and φ_2 given above. We also need to add the following transition rules to Δ , where z and z' range over Σ :

$$\begin{array}{ll} C \rightarrow F \mid S, & (z, z') \rightarrow X_{(z, z')} \mid \varepsilon, \\ V \rightarrow \varepsilon, & Z' \rightarrow X_{(A, B)} \mid X_{(B, A)}, \\ \hat{V} \rightarrow \varepsilon, & X_{(z, z')} \rightarrow \varepsilon \end{array}$$

Let us first re-examine the simple example of Section 3.1. Consider a configuration $C\alpha$ where $\alpha = (A, A)(A, \bullet)(\bullet, A)(B, B)Z'$. Then $\mathcal{P}(C\alpha, \varphi_1)$ is the probability of all runs that start with the transition $C\alpha \xrightarrow{1/2} F\alpha$ and then reach a configuration with the head $X_{(A, \cdot)}$ (here \cdot is an arbitrary symbol) without any prior visit to a configuration with the head $X_{(B, \cdot)}$. In our example, this probability is equal to

$$\frac{1}{2} \left(1 \cdot \frac{1}{2} + 1 \cdot \frac{1}{2^2} + 0 \cdot \frac{1}{2^3} + 1 \cdot \frac{1}{2^4} \right)$$

Similarly, $\mathcal{P}(C\alpha, \varphi_2)$ is the probability of all runs starting with $C\alpha \xrightarrow{1/2} S\alpha$ that reach a configuration with the head $X_{(\cdot, B)}$ without any prior visit to a configuration with the head $X_{(\cdot, A)}$. In our example, $\mathcal{P}(C\alpha, \varphi_2)$ is equal to

$$\frac{1}{2} \left(0 \cdot \frac{1}{2} + 0 \cdot \frac{1}{2^2} + 1 \cdot \frac{1}{2^3} + 1 \cdot \frac{1}{2^4} \right)$$

Clearly, $\mathcal{P}(C\alpha, \varphi_1 \vee \varphi_2) = \mathcal{P}(C\alpha, \varphi_1) + \mathcal{P}(C\alpha, \varphi_2)$. In our example, this yields

$$\frac{1}{2} \left(\left(1 \cdot \frac{1}{2} + 1 \cdot \frac{1}{2^2} + 0 \cdot \frac{1}{2^3} + 1 \cdot \frac{1}{2^4} \right) + \left(0 \cdot \frac{1}{2} + 0 \cdot \frac{1}{2^2} + 1 \cdot \frac{1}{2^3} + 1 \cdot \frac{1}{2^4} \right) \right)$$

which is equal to one half because the probabilities are “complementary”. Similarly to the case of pPDA and PCTL, this complementarity breaks down iff the

first and the second component of stack symbols in α are *not* the same, in which case the probability $\mathcal{P}(C\alpha, \varphi_1 \vee \varphi_2)$ is different from $\frac{1}{2}$. More precisely,

$$\mathcal{P}(C(z_1, z'_1) \cdots (z_\ell, z'_\ell)Z', \varphi_1 \vee \varphi_2) = \frac{1}{2} (Fnum(z_1 \cdots z_\ell) + Snum(z'_1 \cdots z'_\ell)).$$

Due to Lemma 3.4, we have that $C(z_1, z'_1) \cdots (z_\ell, z'_\ell)Z' \models \mathcal{P}^{=1/2}(\varphi_1 \vee \varphi_2)$ iff $z_1 \cdots z_\ell \stackrel{\circ}{=} z'_1 \cdots z'_\ell$. From this and Lemma 3.6 we obtain that $Z \models^\nu \diamond^{>0}(C \wedge \mathcal{P}^{=1/2}(\varphi_1 \vee \varphi_2))$ iff the PCP instance has a solution.

4. Upper Complexity Bounds for Qualitative Model-checking

In this section we prove the upper complexity bounds given in Fig. 1. That is, we aim at proving the following theorem:

Theorem 4.1. *The model-checking problem for pPDA and the logics qPCTL and qPCTL* is in EXPTIME and 2-EXPTIME, respectively. For an arbitrary fixed qPCTL* formula Φ , the model-checking problem for pPDA and pBPA against Φ is in EXPTIME and P, respectively.*

For the rest of this section, we fix a pPDA $\Delta = (Q, \Gamma, \rightarrow, Prob)$. Given a qPCTL* formula Ψ and a valuation ν , we denote by $\mathcal{C}(\Psi, \nu)$ the set of all configurations $p\alpha$ of Δ satisfying $p\alpha \models^\nu \Psi$.

The core of our qPCTL* model-checking algorithm is a global model-checking procedure for qualitative LTL. We show that for every formula of the form $\mathcal{P}^{=1}(\varphi)$, where φ is an LTL formula, the set of all configurations satisfying $\mathcal{P}^{=1}(\varphi)$ can be encoded by an effectively constructible DFA. Here we employ the *local* LTL model-checking algorithm of [23] which decides whether almost all runs initiated in a given configuration $p\alpha$ satisfy a given LTL formula. The global qualitative LTL model-checking procedure is then used to solve the model-checking problem for a qPCTL* formula Φ . For every state subformula Ψ of Φ , the algorithm computes a DFA \mathcal{A}_Ψ such that $\mathcal{C}(\mathcal{A}_\Psi) = \mathcal{C}(\Psi, \nu)$. It starts with the innermost subformulae (i.e., atomic propositions) and then proceeds consistently with the natural structural ordering over the subformulae. Thus, we eventually obtain a DFA \mathcal{A}_Φ such that $\mathcal{C}(\mathcal{A}_\Phi) = \mathcal{C}(\Phi, \nu)$. Theorem 4.1 follows by analyzing the complexity of this algorithm. For the reader's convenience, in Section 4.2 we provide an example of the execution of the algorithm on a simple instance.

We start by formulating the result about global LTL model-checking precisely.

Lemma 4.2. *Let φ be an LTL formula, and let a_1, \dots, a_n be the atomic propositions occurring in φ . Let ν be a regular valuation, and let \mathcal{A}_i be a DFA satisfying $\mathcal{C}(\mathcal{A}_i) = \nu(a_i)$ for all $1 \leq i \leq n$. Furthermore, let $K = \prod_{i=1}^n |K_i|$ where K_i is the set of states of \mathcal{A}_i . Then there is a DFA \mathcal{A} with $2^{|\mathcal{Q}| \cdot 2^{|\varphi|}} \cdot K$ states computable in*

$$2^{|\Delta|^{\mathcal{O}(1)} \cdot 2^{\mathcal{O}(|\varphi|)}} \cdot K^{\mathcal{O}(1)}$$

time such that $\mathcal{C}(\mathcal{P}^=1\varphi, \nu) = \mathcal{C}(\mathcal{A})$. Moreover, if Δ is a pBPA, then \mathcal{A} has $2^{2^{|\varphi|}} \cdot K$ states and it is computable in $2^{2^{\mathcal{O}(|\varphi|)}} \cdot (|\Delta| \cdot K)^{\mathcal{O}(1)}$ time.

A proof of Lemma 4.2 is postponed to Section 4.1. Now we show how to complete the proof of Theorem 4.1 using Lemma 4.2.

Let us fix a qPCTL* formula Φ and a regular valuation ν . Since $\mathcal{P}^=0(\varphi)$ is equivalent to $\mathcal{P}^=1(\neg\varphi)$, we may safely assume that no subformula of Φ takes the form $\mathcal{P}^=0(\varphi)$. For every qPCTL* path formula φ , we define the set $St(\varphi)$ of all “maximal” state subformulae of φ inductively as follows: If φ is a state formula, then $St(\varphi) = \{\varphi\}$. Otherwise, we put

$$St(\varphi) = \begin{cases} St(\varphi_1) & \text{if } \varphi \equiv \neg\varphi_1 \text{ or } \varphi \equiv \mathcal{X}\varphi_1; \\ St(\varphi_1) \cup St(\varphi_2) & \text{if } \varphi \equiv \varphi_1 \wedge \varphi_2 \text{ or } \varphi \equiv \varphi_1 \mathcal{U} \varphi_2. \end{cases}$$

Our algorithm starts by computing a special form of the syntax tree for Φ where the nodes are labeled by state subformulae of Φ . More precisely, for every qPCTL* state formula Ψ we define the associated $Tree(\Psi)$ inductively as follows:

- $Tree(a)$ is a tree with only one node (the root) labeled by a ;
- $Tree(\Psi \wedge \Xi)$ is a tree whose root is labeled by $\Psi \wedge \Xi$ and its only successors are the roots of $Tree(\Psi)$ and $Tree(\Xi)$;
- $Tree(\neg\Psi)$ is a tree whose root is labeled by $\neg\Psi$ and its only successor is the root of $Tree(\Psi)$;
- $Tree(\mathcal{P}^=1\varphi)$ is a tree whose root is labeled by $\mathcal{P}^=1\varphi$ and its successors are the roots of all $Tree(\Psi)$ where $\Psi \in St(\varphi)$.

Then, the algorithm computes for each node u of $Tree(\Phi)$ a DFA \mathcal{A}_u such that $\mathcal{C}(\mathcal{A}_u) = \mathcal{C}(\Psi_u, \nu)$, where Ψ_u is the label of u . This is achieved by processing $Tree(\Phi)$ from its leaves to the root as follows:

- If u is a leaf labeled by $a \in Ap$, then $\mathcal{A}_u = \mathcal{A}_a$.

- If u is labeled by $\Psi \wedge \Xi$, then u has two successors s and t labeled by Ψ and Ξ , respectively. By applying a standard product construction (see, e.g., [25]) to \mathcal{A}_s and \mathcal{A}_t , we yield a DFA \mathcal{A}_u such that $\mathcal{C}(\mathcal{A}_u) = \mathcal{C}(\mathcal{A}_s) \cap \mathcal{C}(\mathcal{A}_t)$. Clearly, $\mathcal{C}(\mathcal{A}_u) = \mathcal{C}(\Psi \wedge \Xi, \nu)$.
- If u is labeled by $\neg\Psi$, then \mathcal{A}_u is obtained just by complementing the automaton \mathcal{A}_s , where s is the only successor of u .
- If u is labeled by $\mathcal{P}^=1\varphi$, then we first construct an LTL formula $\hat{\varphi}$ by replacing each occurrence of every formula $\Psi \in St(\varphi)$ with a fresh atomic proposition b_Ψ , and extend the valuation ν into a (regular) valuation $\hat{\nu}$ by stipulating $\hat{\nu}(b_\Psi) = \mathcal{C}(\mathcal{A}_s)$, where s is the unique successor of u labeled by Ψ . By applying Lemma 4.2, we obtain a DFA \mathcal{A}_u satisfying

$$\mathcal{C}(\mathcal{A}_u) = \mathcal{C}(\mathcal{P}^=1\hat{\varphi}, \hat{\nu}) = \mathcal{C}(\mathcal{P}^=1\varphi, \nu).$$

Note that $\mathcal{C}(\Phi, \nu) = \mathcal{C}(\mathcal{A}_r)$ where r is the root of $Tree(\Phi)$.

Complexity analysis. For every node u of $Tree(\Phi)$, let $Tree(\Phi, u)$ be the subtree of $Tree(\Phi)$ rooted by u . We use $L(u)$ to denote the number of leaves in $Tree(\Phi, u)$, and $P(u)$ to denote the number of nodes labeled by a formula of the form $\mathcal{P}^=1\varphi$ in $Tree(\Phi, u)$. We also use $|\mathcal{A}|$ to denote the maximal $|\mathcal{A}_a|$ where a occurs in Φ , and $|\theta|$ to denote the maximal $|\hat{\xi}|$ where $\mathcal{P}^=1\xi$ is a subformula of Φ . (Recall that $\hat{\xi}$ is the LTL formula obtained from ξ by substituting each occurrence of every $\Psi \in St(\xi)$ with a fresh atomic proposition b_Ψ .)

A straightforward induction on the height of $Tree(\Phi, u)$ reveals that \mathcal{A}_u has at most

$$2^{P(u) \cdot |\mathcal{Q}| \cdot 2^{|\theta|}} \cdot |\mathcal{A}|^{L(u)}$$

states and it is computable in time

$$2^{P(u) \cdot |\Delta|^{\mathcal{O}(1)} \cdot 2^{\mathcal{O}(|\theta|)}} \cdot |\mathcal{A}|^{\mathcal{O}(L(u))}.$$

Let r be the root of $Tree(\Phi)$. Since $|\theta|, P(r), L(r) \leq |\Phi|$, the automaton \mathcal{A}_r is computable in time

$$2^{|\Phi| \cdot |\Delta|^{\mathcal{O}(1)} \cdot 2^{\mathcal{O}(|\Phi|)}} \cdot |\mathcal{A}|^{\mathcal{O}(|\Phi|)}.$$

Hence, the model-checking problem for pPDA and qPCTL* is in **2-EXPTIME**, but for an arbitrary fixed formula Φ , the problem is in **EXPTIME**.

If Δ is a BPA, then $|Q| = 1$ and hence \mathcal{A}_u has at most

$$2^{P(u) \cdot 2^{|\theta|}} \cdot |\mathcal{A}|^{L(u)}$$

states. By applying the ‘‘pBPA part’’ of Lemma 4.2, one can further show that \mathcal{A}_u is computable in time

$$2^{P(u) \cdot 2^{\mathcal{O}(|\theta|)}} \cdot |\Delta|^{\mathcal{O}(1)} \cdot |\mathcal{A}|^{\mathcal{O}(L(u))}.$$

Hence, the model-checking problem for pBPA and qPCTL* is solvable in time

$$2^{|\Phi| \cdot 2^{\mathcal{O}(|\Phi|)}} \cdot |\Delta|^{\mathcal{O}(1)} \cdot |\mathcal{A}|^{\mathcal{O}(|\Phi|)}.$$

In particular, the problem is in **P** for an arbitrary fixed qPCTL* formula Φ .

For qPCTL, every (sub)formula of the form $\mathcal{P}^=1\varphi$ satisfies that φ is either $\mathcal{X}\Phi_1$, $\Phi_1 \mathcal{U} \Phi_2$, $\neg(\mathcal{X}\Phi_1)$, or $\neg(\Phi_1 \mathcal{U} \Phi_2)$. Hence, we have that $|\theta|$ is $\mathcal{O}(1)$, and the complexity bounds given above imply that the model-checking problem for pPDA and qPCTL is in **EXPTIME**.

4.1. A Proof of Lemma 4.2

For the rest of this section, we fix a pPDA $\Delta = (Q, \Gamma, \delta, Prob)$. We start by proving an analogy of Lemma 4.2 for *simple* valuations (see Lemma 4.4 below), using the local LTL model-checking algorithm of [23]. Then, we show how to reduce the global model-checking problem for regular valuations to the one for simple valuations. Here we use a standard technique of simulating a tuple of DFA on-the-fly in the stack (see, e.g., [18]).

To achieve the desired complexity bounds, the solution for simple valuations is decomposed into two steps. First, we compute all *non-terminating heads*, i.e., the set \mathbf{S}_Δ of all configurations $pX \in \mathcal{C}(\Delta)$ such that the probability of reaching a configuration with an empty stack from pX is strictly less than one. By applying the results of [17, 22], the set \mathbf{S}_Δ is computable in polynomial space for general pPDA, and in polynomial time for pBPA. Then, we solve the model-checking problem itself. Here we use the local LTL model-checking algorithm of [23] which runs in time polynomial in the number of stack symbols, assuming that the set of non-terminating heads has already been computed. More precisely, we rely on the following theorem:

Theorem 4.3 ([23]). *Assume that ν is simple and the set \mathbf{S}_Δ has already been computed. Let $pX \in Q \times \Gamma$ and let φ be an LTL formula. Then the problem of whether $pX \models^\nu \mathcal{P}^=1\varphi$ is decidable in time $2^{\mathcal{O}(|\varphi|)} \cdot |\Delta|^{\mathcal{O}(1)}$.*

Now we generalize Theorem 4.3 to a global model-checking algorithm.

Lemma 4.4. *Assume that ν is simple and the set S_Δ has already been computed. Let φ be an LTL formula. Then a DFA \mathcal{B} satisfying $\mathcal{C}(\mathcal{P}^=1\varphi, \nu) = \mathcal{C}(\mathcal{B})$ is computable in time $2^{|Q| \cdot 2^{\mathcal{O}(|\varphi|)}} \cdot |\Delta|^{\mathcal{O}(1)}$ and has $2^{|Q| \cdot 2^{|\varphi|}}$ states.*

Proof. We begin by introducing some notation adopted from [23]. Recall that $Cl(\varphi)$ denotes the set of all subformulae of φ . To every run w of Δ we assign its type $type(w) \subseteq Cl(\varphi)$ such that $\psi \in type(w)$ iff $w \models^\nu \psi$. A type t is *admissible* for $p\alpha \in \mathcal{C}(\Delta)$ if

$$\mathcal{P}(\{w \in Run(p\alpha) \mid type(w) = t\}) > 0.$$

The set of states of \mathcal{B} is $2^Q \times 2^{Cl(\varphi)}$. After reading the reverse of a word $\alpha \in \Gamma^*$, the automaton \mathcal{B} enters a state P such that $(p, t) \in P$ iff t is admissible for $p\alpha$. If the automaton \mathcal{B} reads $p \in Q$ (being in such a state P), it enters an accepting state iff all $(p, t) \in P$ satisfy $\varphi \in t$. Obviously, the latter is equivalent to $p\alpha \models^\nu \mathcal{P}^=1(\varphi)$.

In order to define the transition function of \mathcal{B} , we need the following simple observation about types (a proof is immediate and hence omitted):

Lemma 4.5. *Given a configuration $p\alpha$ and a run w of type t such that $p\alpha \rightarrow w(0)$, the type t' of the run $p\alpha, w$ is uniquely determined by $p\alpha$ and t as follows: For every $\psi \in Cl(\varphi)$ we have that $\psi \in t'$ iff one of the following is true:*

- $\psi \in Ap$ and $p\alpha \in \nu(\psi)$;
- $\psi = \psi_1 \wedge \psi_2$ and $\psi_1 \in t'$ and $\psi_2 \in t'$;
- $\psi = \neg\psi'$ and $\psi' \notin t'$;
- $\psi = \mathcal{X}\psi'$ and $\psi' \in t$;
- $\psi = \psi_1 \mathcal{U} \psi_2$ and either $\psi_2 \in t'$, or $\psi_1 \in t'$ and $\psi \in t$.

Denote by $type(p\alpha, t)$ the type t' determined by $p\alpha$ and t . This extends inductively to paths, so given a finite path v , we denote by $type(v, t)$ the type determined by v and t , i.e., the type of a run of the form v, w' where w' has the type t . In the rest of this proof we also use the following notation. Given a finite path v of length at least one, we denote by \tilde{v} the path obtained from v by removing the last configuration. Further, we say that a run w initiated in a configuration $pX\alpha \in \mathcal{C}(\Delta)$

(where $X \in \Gamma$) is *clean* if w does not visit any configuration of the form $q\alpha$ where $q \in Q$. That is, the stack height along a clean run never decreases below its initial level. The set of all clean runs initiated in $pX\alpha$ is denoted by $Clean(pX\alpha)$.

Now we formally define the automaton $\mathcal{B} = (K, \Sigma, \gamma, P_0, F)$ where

- $K = 2^Q \times 2^{Cl(\varphi)}$ is the set of states.
- $\Sigma = \Gamma \cup Q$ is the input alphabet.
- The transition function γ is defined as follows (note that γ is total):
 - For all $P \in K$ and $X \in \Gamma$, the set $\gamma(P, X)$ consists of all $(p, t) \in Q \times 2^{Cl(\varphi)}$ such that at least one of the following conditions holds:
 - (A) $\mathcal{P}(\{w \in Clean(pX) \mid type(w) = t\}) > 0$.
 - (B) There is $(q, t') \in P$ and a path v from pX to $q\epsilon$ such that $type(\tilde{v}, t') = t$.
 - For all $P \in K$ and $p \in Q$, the set $\gamma(P, p)$ is either empty or equal to $Q \times 2^{Cl(\varphi)}$, depending on whether $\varphi \in t$ for all $(p, t) \in P$ or not, respectively (in the latter case, we could use an arbitrary non-empty subset of $Q \times 2^{Cl(\varphi)}$).
- The initial state $P_0 \subseteq Q \times 2^{Cl(\varphi)}$ consists of all pairs (p, t) where $p \in Q$ and t is the unique type of the run $(p\epsilon)^\omega$.
- $F = \{\emptyset\}$ is the set of accepting states.

The automaton \mathcal{B} is constructed so that after reading the reverse of $\alpha \in \Gamma^*$, a state P is entered such that $(q, t) \in P$ iff t is admissible for $q\alpha$. To get some intuition on how \mathcal{B} works, assume that the above condition holds for a given $\alpha \in \Gamma^*$, and consider admissible types for $pX\alpha$, where $X \in \Gamma$. First, a type t can be admissible for $pX\alpha$ because there is a set $R \subseteq Clean(pX\alpha)$ such that $\mathcal{P}(R) > 0$ and all runs in R have the type t (note that since ν is simple, α is irrelevant). This is captured by Condition (A) above. The second option, captured by Condition (B) above, is that there is a path v from $pX\alpha$ to $q\alpha$ such that the stack is never decreased to α along \tilde{v} , and there is a set R of runs of the form $\tilde{v}w$ such that $\mathcal{P}(R) > 0$ and all runs in R are of type t . Then, there must be a subset of runs $R' \subseteq Run(q\alpha)$ such that $\mathcal{P}(R') > 0$ and all runs in R' are of type t' where $type(\tilde{v}, t') = t$.

Now we give a full correctness proof which formalizes the above intuition.

Lemma 4.6. *For every configuration $p\alpha$ we have that $p\alpha \models^\nu \mathcal{P}^{-1}(\varphi)$ iff $p\alpha \in \mathcal{C}(\mathcal{B})$.*

Proof. Assume that after reading the reverse of $\alpha \in \Gamma^*$ the automaton \mathcal{B} reaches a state P . It suffices to prove that $(p, t) \in P$ iff t is admissible for $p\alpha$. The rest follows immediately from the definition of \mathcal{B} . We proceed by induction on the length of α . If $\alpha = \varepsilon$, there is only one run initiated in $p\varepsilon$ and our lemma follows directly from the definition of P_0 . Now assume that $\alpha = X\beta$. Let t be a type, and let R^t be the set of all runs of type t initiated in $p\alpha$. Given $q \in Q$, we denote by R_q^t the set of all runs in R^t that visit $q\beta$ and no configuration preceding this visit has stack content β . The set $R^t \setminus \cup_{q \in Q} R_q^t$ is denoted by R_{Clean}^t . We distinguish two possibilities:

- $\mathcal{P}(R_{Clean}^t) > 0$. Observe that for every run $w \in R_{Clean}^t$ there is an associated run of $Clean(pX)$ obtained by removing the suffix β from every configuration of w . Clearly, these two runs have the same type (here we need that ν is simple). By applying Condition (A), we can conclude that t is admissible for $pX\beta$ iff $(p, t) \in P$.
- $\mathcal{P}(R_{Clean}^t) = 0$. Let $(p, t) \in P$, and let P' be the state of \mathcal{B} entered after reading the reverse of β . By Condition (B), there is $(q, t') \in P'$ and a path v from pX to $q\varepsilon$ such that $t = type(\tilde{v}, t')$. By induction hypothesis, the type t' is admissible for $q\beta$, i.e., there is a set of runs $R^{t'} \subseteq Run(q\beta)$ of type t' such that $\mathcal{P}(R^{t'}) > 0$. Due to Lemma 4.5, for all $w \in R^{t'}$ we have that $type(\tilde{v}w) = type(\tilde{v}, t') = t$ which implies that $\tilde{v}w \in R^t$, and thus

$$\mathcal{P}(R^t) \geq \mathcal{P}(\{\tilde{v}w \mid w \in R^{t'}\}) = \mathcal{P}(Run(v)) \cdot \mathcal{P}(R^{t'}) > 0$$

Hence, t is admissible for $pX\beta$.

Now suppose that t is admissible for $pX\beta$, i.e., $\mathcal{P}(R^t) > 0$. As $\mathcal{P}(R_{Clean}^t) = 0$ and $\mathcal{P}(R^t) = \mathcal{P}(R_{Clean}^t \cup \cup_q R_q^t)$, there must be $q \in Q$ such that $\mathcal{P}(R_q^t) > 0$. Given a type t' and a path v from $pX\beta$ to $q\beta$ such that \tilde{v} does not visit any configuration with stack content β , we denote by $R_{v,t'}$ the set of all runs of the form $\tilde{v}w$ where w has the type t' . At least one of the sets $R_{v,t'}$ must have a positive probability, and hence t' is admissible for $q\beta$. By induction hypothesis, $(q, t') \in P'$. By applying Lemma 4.5, $type(\tilde{v}, t') = t$ and thus $(p, t) \in P$ by Condition (B). \square

To finish the proof of Lemma 4.4, we need to analyze the complexity of deciding Conditions (A) and (B). Condition (A), i.e., the question whether

$\mathcal{P}(\{w \in \text{Clean}(pX) \mid \text{type}(w) = t\}) > 0$, is equivalent to the question whether $pX \not\models^\nu \mathcal{P}^=1\theta$ where θ is the following LTL formula:

$$\left(\bigvee_{\psi \in t} \neg\psi \right) \vee \left(\bigvee_{\xi \in \text{Cl}(\varphi) \setminus t} \xi \right) \vee \left(\bigvee_{q \in Q} \diamond[q\varepsilon] \right)$$

Here $[q\varepsilon]$ is an atomic proposition which holds only in the configuration $q\varepsilon$. Thus, we can apply Theorem 4.3 and conclude that Condition (A) is decidable in $2^{\mathcal{O}(|\varphi|)} \cdot |\Delta|^{\mathcal{O}(1)}$ time.

Condition (B) is reducible to the reachability problem for non-probabilistic PDA as follows: We construct a PDA Δ' where

- the set of control states is the same as in Δ ;
- the stack alphabet consists of all symbols (t, X, t') where $X \in \Gamma$ and t, t' are types;
- the transitions are defined as follows:
 - $p(t, X, t') \rightarrow q(s, Y, s')(s', Z, t')$ iff $pX \rightarrow qYZ$, $t = \text{type}(pX, s)$, and s' is an arbitrary type,
 - $p(t, X, t') \rightarrow q(s, Y, t')$ iff $pX \rightarrow qY$ and $t = \text{type}(pX, s)$,
 - $p(t, X, t') \rightarrow q\varepsilon$ iff $pX \rightarrow q\varepsilon$ and $t = \text{type}(pX, t')$.

There might be a head $p(t, X, t')$ to which none of the above conditions applies, and in this case we put $p(t, X, t') \rightarrow p(t, X, t')$.

It follows directly from definitions that there is a path v from pX to $q\varepsilon$ such that $\text{type}(\tilde{v}, t') = t$ iff there is a path from $p(t, X, t')$ to $q\varepsilon$. The latter can be decided in time polynomial in $|\Delta'|$, and hence in time $2^{\mathcal{O}(|\varphi|)} \cdot |\Delta|^{\mathcal{O}(1)}$.

Hence, \mathcal{B} is indeed computable in time

$$2^{|Q| \cdot 2^{\mathcal{O}(|\varphi|)}} \cdot |\Delta|^{\mathcal{O}(1)}$$

and it has $2^{|Q| \cdot 2^{|\varphi|}}$ states. □

Now we have all the tools needed to prove Lemma 4.2. So, let φ be an LTL formula with atomic propositions a_1, \dots, a_n , and let ν be a regular valuation such that $\nu(a_i) = \mathcal{C}(\mathcal{A}_i)$ for all $1 \leq i \leq n$, where $\mathcal{A}_i = (K_i, \Gamma \cup Q, \gamma_i, q_i^0, F_i)$ is a DFA. We show that ν can be effectively transformed into a simple valuation by simulating the DFA $\mathcal{A}_1, \dots, \mathcal{A}_n$ on-the-fly in the stack of Δ . This technique is similar to the one used in [15, 18].

Definition 4.7. We define a pPDA $\Delta[\mathcal{A}_1, \dots, \mathcal{A}_n] = (Q, \Gamma', \delta', \text{Prob}')$ where $\Gamma' = \Gamma \times \prod_{i=1}^n K_i$, and the rules of $\Delta[\mathcal{A}_1, \dots, \mathcal{A}_n]$ are determined as follows:

1. $p(X, \vec{s}) \xrightarrow{x} q\varepsilon$ iff $pX \xrightarrow{x} q\varepsilon$;
2. $p(X, \vec{s}) \xrightarrow{x} q(Y, \vec{s})$ iff $pX \xrightarrow{x} qY$;
3. $p(X, \vec{s}) \xrightarrow{x} q(Y, \vec{t})(Z, \vec{s})$ iff $pX \xrightarrow{x} qYZ$ and $\gamma_i(\vec{s}(i), Z) = \vec{t}(i)$ for all $1 \leq i \leq n$;

Intuitively, the pPDA $\Delta[\mathcal{A}_1, \dots, \mathcal{A}_n]$ behaves exactly like Δ but it also simulates the execution of $\mathcal{A}_1, \dots, \mathcal{A}_n$ on-the-fly in its stack. Observe that not every configuration of $\Delta[\mathcal{A}_1, \dots, \mathcal{A}_n]$ corresponds to a correct simulation of $\mathcal{A}_1, \dots, \mathcal{A}_n$. We say that a configuration $p(X_1, \vec{s}_1) \cdots (X_k, \vec{s}_k)$ of $\Delta[\mathcal{A}_1, \dots, \mathcal{A}_n]$ is *consistent* if $\vec{s}_k = (q_1^0, \dots, q_n^0)$ and for all $1 < j \leq k$ and $1 \leq i \leq n$ we have that $\vec{s}_{j-1}(i) = \gamma_i(\vec{s}_j(i), X_j)$. A configuration with an empty stack is also consistent. Given $pX_1 \cdots X_n$, we denote by $\mathcal{K}(pX_1 \cdots X_n)$ the unique consistent configuration of the form $p(X_1, \vec{s}_1) \cdots (X_n, \vec{s}_n)$.

Lemma 4.8. Given a consistent configuration $p(X_1, \vec{s}_1) \cdots (X_k, \vec{s}_k)$, the configuration $pX_1 \cdots X_k$ is in $\mathcal{C}(\mathcal{A}_i)$ iff \mathcal{A}_i initiated in $\vec{s}_1(i)$ accepts the word X_1p .

Proof. For the “ \Rightarrow ” part, observe that \mathcal{A}_i accepts $X_k \cdots X_1p$ by going through the states $\vec{s}_k(i), \dots, \vec{s}_1(i), t, t'$, where t and t' are the states in which \mathcal{A}_i ends after reading $X_k \cdots X_1$ and $X_k \cdots X_1p$, respectively, and t' is accepting. It follows that when initiated in $\vec{s}_1(i)$, \mathcal{A}_i accepts the word X_1p . The other direction is also immediate. \square

Hence, the membership of $pX_1 \cdots X_k$ into $\mathcal{C}(\mathcal{A}_i)$ can be determined just by inspecting the head $p(X_1, \vec{s}_1)$ of the corresponding consistent configuration. Let $\nu' : Ap \rightarrow 2^{\mathcal{C}(\Delta[\mathcal{A}_1, \dots, \mathcal{A}_n])}$ be a simple valuation such that for every a_i , where $1 \leq i \leq n$, the set of heads H_{a_i} (see Definition 2.3) is determined as follows:

- For every $p(X_1, \vec{s}_1) \in Q \times \Gamma'$, we have that $p(X_1, \vec{s}_1) \in H_{a_i}$ iff \mathcal{A}_i initiated in $\vec{s}_1(i)$ accepts the word X_1p ;
- for every $p \in Q$, we have that $p \in H_{a_i}$ iff $p\varepsilon \in \mathcal{C}(\mathcal{A}_i)$.

Now it is easy to prove that

$$p\alpha \models^\nu \mathcal{P}^{-1}\varphi \quad \text{iff} \quad \mathcal{K}(p\alpha) \models^{\nu'} \mathcal{P}^{-1}\varphi$$

By Lemma 4.4, there is an effectively computable DFA $\mathcal{B} = (K, \Gamma \times \prod_{i=1}^n K_i, \gamma, F)$ encoding all configurations $p\alpha \in \mathcal{C}(\Delta[\mathcal{A}_1, \dots, \mathcal{A}_k])$ such that $p\alpha \models^{\nu'} \mathcal{P}^=1\varphi$. Note that the size of K is in $2^{|Q| \cdot 2^{|\varphi|}}$.

Now we aim at constructing a DFA \mathcal{A} such that $\mathcal{C}(\mathcal{B}) = \mathcal{K}(\mathcal{C}(\mathcal{A}))$. Note that \mathcal{A} then encodes all configurations of Δ satisfying $\mathcal{P}^=1\varphi$ because

$$p\alpha \models^{\nu'} \mathcal{P}^=1\varphi \quad \text{iff} \quad \mathcal{K}(p\alpha) \models^{\nu'} \mathcal{P}^=1\varphi \quad \text{iff} \quad \mathcal{K}(p\alpha) \in \mathcal{C}(\mathcal{B}) \quad \text{iff} \quad p\alpha \in \mathcal{C}(\mathcal{A}).$$

Intuitively, \mathcal{A} is essentially the same as \mathcal{B} , but it also simulates the execution of $\mathcal{A}_1, \dots, \mathcal{A}_n$ on-the-fly. This means that if the automaton \mathcal{B} goes through the states q_1, \dots, q_{k+1}, q' when reading $(X_1, \vec{s}_1) \dots (X_k, \vec{s}_k)p$, then the automaton \mathcal{A} goes through the states $(q_1, \vec{s}_1), \dots, (q_k, \vec{s}_k), (q_{k+1}, \vec{s}_{k+1}), (q', \vec{t})$ when reading $X_1 \dots X_k p$, where $\gamma_i(\vec{s}_k(i), X_k) = \vec{s}_{k+1}(i)$ and $\gamma_i(\vec{s}_{k+1}(i), p) = \vec{t}(i)$ for all $1 \leq i \leq n$.

Formally, we put $\mathcal{A} = (K \times \prod_{i=1}^n K_i, \Gamma \cup Q, \gamma', (q_0, \vec{q}_0), F')$ where $\vec{q}_0 = (q_1^0, \dots, q_n^0)$, $F' = F \times \prod_{i=1}^n K_i$, and the transition function γ' is defined as follows:

- For all $X \in \Gamma$ and $(s, \vec{s}) \in K \times \prod_{i=1}^n K_i$, we have that $\gamma'((s, \vec{s}), X) = (t, \vec{t})$ iff $\gamma(s, (X, \vec{s})) = t$ and $\gamma_i(\vec{s}(i), X) = \vec{t}(i)$ for all $1 \leq i \leq n$;
- for all $p \in Q$ and $(s, \vec{s}) \in K \times \prod_{i=1}^n K_i$, we have that $\gamma'((s, \vec{s}), p) = (t, \vec{t})$ iff $\gamma(s, p) = t$ and $\gamma_i(\vec{s}(i), p) = \vec{t}(i)$ for all $1 \leq i \leq n$.

It is straightforward to show that $\mathcal{C}(\mathcal{B}) = \mathcal{K}(\mathcal{C}(\mathcal{A}))$.

Let us analyze the complexity of the above procedure. First, note that if the set \mathbf{S}_Δ is given, then the set $\mathbf{S}_{\Delta[\mathcal{A}_1, \dots, \mathcal{A}_k]}$ is computable in time polynomial in $\Delta[\mathcal{A}_1, \dots, \mathcal{A}_k]$ (it is easy to see that pX terminates with probability one iff $p(X, \vec{s})$ terminates with probability one for all \vec{s}). Since \mathbf{S}_Δ can be computed in exponential/polynomial time for pPDA/pBPA, we obtain that $\mathbf{S}_{\Delta[\mathcal{A}_1, \dots, \mathcal{A}_k]}$ is computable in $2^{|\Delta|^{\mathcal{O}(1)}} \cdot (\prod_{i=1}^n |K_i|)^{\mathcal{O}(1)}$ time for pPDA and in $(|\Delta| \cdot \prod_{i=1}^n |K_i|)^{\mathcal{O}(1)}$ time for pBPA. By Lemma 4.4, the automaton \mathcal{B} satisfying $\mathcal{C}(\mathcal{B}) = \mathcal{C}(\mathcal{B})$ is computable in time polynomial in $|K'| \cdot |\Delta| \cdot n \cdot \prod_{i=1}^n |K_i|$ where $|K'| = 2^{|Q| \cdot 2^{|\varphi|}}$. Hence, the overall time complexity is $2^{|\Delta|^{\mathcal{O}(1)} \cdot 2^{\mathcal{O}(|\varphi|)}} \cdot (\prod_{i=1}^n |K_i|)^{\mathcal{O}(1)}$ for pPDA and $2^{2^{\mathcal{O}(|\varphi|)}} \cdot (|\Delta| \cdot \prod_{i=1}^n |K_i|)^{\mathcal{O}(1)}$ for pBPA.

4.2. Example of the execution of the algorithm

To conclude this section, let us show how the algorithm is executed on a simple pPDA $\Delta = (\{p, q\}, \{X, Y\}, \delta, Prob)$, where δ is given below and the transition

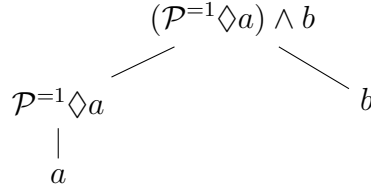
probabilities are distributed uniformly.

$$\begin{array}{ll} pX \rightarrow pYX \mid pX & qX \rightarrow qXX \\ pY \rightarrow qY & qY \rightarrow qYY \mid q\varepsilon \end{array}$$

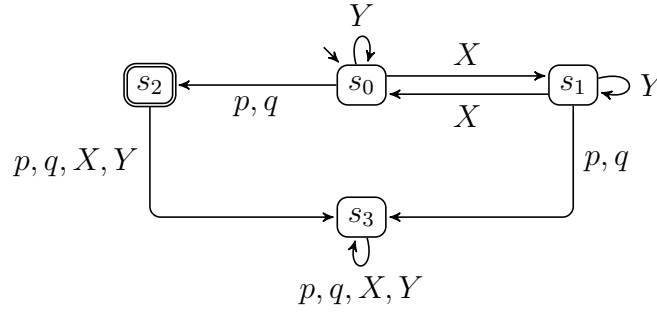
Let $\Phi = (\mathcal{P}^1 \diamond a) \wedge b$, and let ν be a valuation such that

- $\nu(a)$ consists of all configurations with an even number of X 's stored in the stack,
- $\nu(b)$ is the set of all configurations with X on the top of the stack.

We want to determine whether $pX \models^\nu \Phi$. The algorithm starts by computing $Tree(\Phi)$ which looks as follows:



Then, the above tree is processed bottom-up. We start by assigning the following DFA \mathcal{A}_a to the leaf a .



We proceed with the node $\mathcal{P}^1 \diamond a$ of $Tree(\Phi)$. Using Δ and \mathcal{A}_a , we construct a pPDA $\Delta[\mathcal{A}_a] = (\{p, q\}, \{X, Y\} \times \{s_0, s_1\}, \delta', Prob')$ described in Definition 4.7. In particular, δ' contains the following rules (here, $i \in \{0, 1\}$, $j \in \{2, 3\}$, and $Z \in \{X, Y\}$):

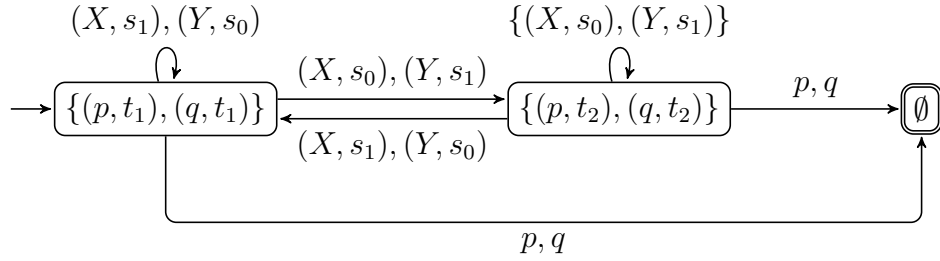
$$\begin{array}{ll} p(X, s_i) \rightarrow p(Y, s_{1-i})(X, s_i) \mid p(X, s_i) & q(X, s_i) \rightarrow q(X, s_{1-i})(X, s_i) \\ p(Y, s_i) \rightarrow q(Y, s_i) & q(Y, s_i) \rightarrow q(Y, s_i)(Y, s_i) \mid q\varepsilon \\ p(X, s_j) \rightarrow p(Y, s_3)(X, s_j) \mid p(X, s_j) & q(X, s_j) \rightarrow q(X, s_3)(X, s_j) \\ p(Y, s_j) \rightarrow q(Y, s_j) & q(Y, s_j) \rightarrow q(Y, s_3)(Y, s_j) \mid q\varepsilon \end{array}$$

Next, we construct a simple valuation ν' where the set H_a (see Definition 2.3) is given by

$$H_a = \{p(X, s_1), q(X, s_1), p(Y, s_0), q(Y, s_0), p, q\}.$$

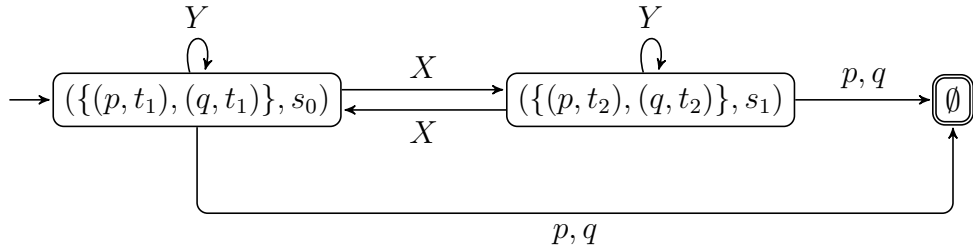
The purpose of the construction of $\Delta[\mathcal{A}_a]$ and ν' is to “simplify” the regular valuation to a simple valuation. Indeed, our proofs give us that $pX \models^\nu \mathcal{P}^{-1}\diamond a$ (here $\nu(a)$ is given by the automaton \mathcal{A}_a) iff $p(X, s_0) \models^{\nu'} \mathcal{P}^{-1}\diamond a$ (here $\nu'(a)$ is given by the set of heads H_a).

Now, we apply the construction given in the proof of Lemma 4.4 and obtain the following DFA \mathcal{B} , where $t_1 = \{\diamond a, a\}$ and $t_2 = \{\diamond a\}$. For simplicity, only the states and transitions contributing to $\mathcal{L}(\mathcal{B})$ are drawn.



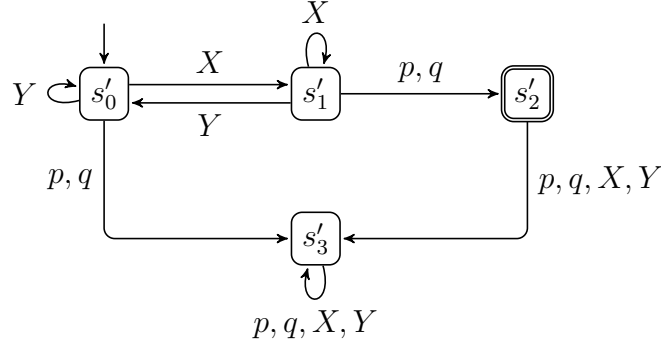
When the automaton \mathcal{B} reads a set of stack symbols of $\Delta[\mathcal{A}_a]$, it “stores” in its state the relevant formulae that will be satisfied if a particular control state of $\Delta[\mathcal{A}_a]$ is read as the next symbol. For example, after reading $(X, s_1)(X, s_0)$ the automaton ends in a state containing $(p, \{\diamond a\})$, which means that $p(X, s_0)(X, s_1)$ satisfies $\mathcal{P}^{-1}\diamond a$, but does not satisfy a .

We are now ready to finish the analysis of the node $\mathcal{P}^{-1}\diamond a$ of $Tree(\Phi)$. From the automaton \mathcal{B} that accepts the (reverse of) configurations of $\Delta[\mathcal{A}_a]$ which satisfy $\mathcal{P}^{-1}\diamond a$ under ν' we construct an automaton accepting the (reverse of) configurations of Δ which satisfy $\mathcal{P}^{-1}\diamond a$ under ν . We employ the construction described at page 22 and obtain the following DFA \mathcal{A} (we again draw only the relevant part of \mathcal{A} and use t_1 and t_2 as before):



Here the intuition is that thanks to the correspondence between configurations of Δ and consistent configurations of $\Delta[\mathcal{A}_a]$, the automaton \mathcal{A} can mimic the behaviour of \mathcal{B} while also simulating the computation of \mathcal{A}_a .

For the leaf b of $Tree(\Phi)$ we have the following automaton \mathcal{A}_b :



We would now apply a standard product construction to get a DFA \mathcal{A}_Φ which accepts the language $\mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{A}_b)$, and then we would verify that the automaton \mathcal{A}_Φ accepts the word Xp , and so $pX \models^\nu \Phi$. We omit this part here as it is trivial.

5. Lower Complexity Bounds for Qualitative Model-checking

In this section we prove the lower complexity bounds for qPCTL and qPCTL* model-checking given in Fig. 1. These results are mostly obtained by simple modifications of the existing results for non-probabilistic PDA. In particular, the **EXPTIME**-hardness proof for model-checking non-probabilistic PDA and CTL* presented in [5] carries over to the probabilistic setting almost immediately. The proof uses a reduction from the acceptance problem for polynomially bounded alternating Turing machines, and can be adapted to pPDA and qPCTL* just by using uniform probability distributions in the constructed PDA and by substituting the A and E path quantifiers with $\mathcal{P}^{=1}$ and $\mathcal{P}^{>0}$ in the constructed CTL* formula Φ , respectively. Although the semantics of A and E is somewhat different from the semantics of $\mathcal{P}^{=1}$ and $\mathcal{P}^{>0}$, respectively, the substitution works in the expected way. This is because

- every occurrence of the universal path quantifier A in Φ is in subformulae of the form $A\mathcal{X}\psi$ or $A\Box\psi$ where ψ is a Boolean combination of state formulae (note that the semantics of $A\mathcal{X}\psi$ and $A\Box\psi$ is essentially the same as the semantics of $\mathcal{P}^{=1}\mathcal{X}\psi$ and $\mathcal{P}^{=1}\Box\psi$, respectively);

- every occurrence of the existential path quantifier E in Φ is in subformulae of the form $E\varphi$, where φ has the following abstract syntax:

$$\varphi ::= \Phi \mid \neg\Phi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \mathcal{X}\varphi \mid \diamond\varphi$$

Here Φ ranges over state formulae. Note that if a run satisfies such a φ , then it has a finite prefix w such that *any* run prefixed by w satisfies φ . Thanks to this property, the semantics of $E\varphi$ is essentially the same as the semantics of $\mathcal{P}^{>0}\varphi$.

It follows that the program complexity of model-checking pPDA and qPCTL* is **EXPTIME**-complete.

Similar arguments can be used to prove **EXPTIME**-hardness of the model-checking problem for pPDA and qPCTL (by modifying the construction of [29]) and **2-EXPTIME**-hardness of the model-checking problem for pPDA and qPCTL* (by modifying another construction of [5]). However, we show that all of these results can be actually strengthened even to BPA and hence also to pBPA. This observation is essentially due to Richard Mayr⁴. We present just the main ideas behind this modification (in a self-contained form) without giving full technical details.

We start by proving **EXPTIME**-hardness of the model-checking problem for pBPA and qPCTL (Theorem 5.1). Then, we indicate how to modify this proof to obtain **2-EXPTIME**-hardness of the model-checking problem for pBPA and qPCTL* (Theorem 5.2).

Theorem 5.1. *The model-checking problem for pBPA and qPCTL is **EXPTIME**-hard.*

Proof Sketch. This proof is a modification of the construction presented in [29] which uses a reduction from the acceptance problem of linearly bounded alternating Turing machines. An alternating Turing machine T is a tuple $(C, \Upsilon, \gamma, q_I, q_A, q_R, \lambda)$, where C is a finite set of control states, Υ is a finite tape alphabet, γ is a transition function, λ is a function that partitions the set of states C into existential and universal states, and q_I, q_A, q_R are the initial, accepting, and rejecting states, respectively. The transition function of T assigns to each pair $(q, X) \in C \times \Upsilon$ an ordered pair of moves, where a move is a triple (q', Y, D) such that $q' \in C$ is the next state, Y is the symbol which rewrites X on the tape, and

⁴Private communication, July 2004.

$D \in \{L, R\}$ specifies the direction in which the head moves. Configurations of T can be written in the form vqw , where $q \in C$ is the current state of T , $vw \in \Upsilon^*$ is the content of the non-empty part of the tape, and the head is positioned over the first letter of w . A configuration is accepting (or rejecting) if its state is q_A (or q_R) and terminal if it is either accepting or rejecting. At the beginning, an input word is written on the tape, the head stays on the leftmost symbol, and the machine is in the initial state q_I . In every step, the machine reads the symbol under the head and moves according to the transition function γ . If the current state is existential, the machine chooses one of the two alternatives. Otherwise, if the current state is universal, the machine continues with both alternatives. The machine stops if either q_A or q_R is reached. This induces a computation tree of T over the input word. (More precisely, nodes of the tree are labeled with configurations, the root is labeled with the initial configuration, leaves are labeled with terminal configurations, inner nodes with non-terminal configurations, successors of an inner node labeled with a configuration c are labeled by one step successors of c in the computation, i.e., existential nodes have one successor and universal nodes have two successors.) The machine accepts the input word iff there is a computation tree for which all its leaves are labeled with accepting configurations.

Let $T = (C, \Upsilon, \gamma, q_I, q_A, q_R, \lambda)$ be an alternating Turing machine using n tape cells on input of size n , and let c be an initial configuration. We construct a pBPA system Δ , a simple valuation ν , and a qPCTL formula Φ such that $G \models^\nu \Phi$ for a distinguished stack symbol G of Δ iff T has an accepting computation from c .

Intuitively, the pBPA Δ simulates a depth-first search for an accepting computation tree on its stack. The formula Φ is used to ensure that the simulation is correct. Formally, we define $\Delta = (\Gamma_\Delta, \delta_\Delta, Prob_\Delta)$ where Γ_Δ contains all symbols of $C \cup \Upsilon$, distinguished symbols E, L, R, G, M, A, F , and additional symbols which we don't define formally and whose role will be explained later. Intuitively, E stands for an existential move, L and R stand for the left and the right elements of a universal move, respectively. The G, M, A, F are special control symbols used for checking whether the simulation is correct. The transition function δ_Δ contains the following rules (the probability function $Prob_\Delta$ is not significant, we only require that it assigns a positive probability to each transition).

$$\begin{array}{lll}
G & \rightarrow & Mc'L \mid Mc'E \mid A \\
M & \rightarrow & G \mid F \\
L & \rightarrow & Mc'R \mid F \\
R & \rightarrow & \varepsilon \\
A & \rightarrow & \varepsilon \\
E & \rightarrow & \varepsilon \\
F & \rightarrow & \varepsilon \\
X & \rightarrow & \varepsilon
\end{array}$$

Here c' stands for a configuration of T (a string of $(\Upsilon \cup C)^{n+1}$), $X \in \Upsilon$, and

$q \in C$. We define $\Phi \equiv \mathcal{P}^{>0}((\neg M) \mathcal{U} (M \wedge \Phi_{Comp}))$ where

$$\Phi_{Comp} \equiv \text{Init} \wedge \mathcal{P}^{>0} \left((\neg F \wedge (A \Rightarrow \text{Accept}) \wedge (M \Rightarrow \text{Move})) \mathcal{U} \varepsilon \right)$$

where the formulae *Init*, *Move* and *Accept*, asserting correctness of the simulation, are described below.

The simulation works as follows: It begins in the configuration G and starts guessing a computation tree of T . Each of the configurations is guessed in n steps, because we need $|\Delta|$ to be polynomial in the size of T (hence, the rules $G \rightarrow Mc'L, G \rightarrow Mc'E, L \rightarrow Mc'R$ are in fact abbreviations for a family of rules that guess a new configuration symbol by symbol, and this is where we use the additional stack symbols). The formula *Init* checks that the first configuration guessed is the initial configuration c . Each time a new configuration is guessed, the run of Δ has to pass through a configuration with M on the top of the stack and the correctness is checked by the formula *Move*. When an accepting configuration is guessed (by putting A on the stack), and checked (by the formula *Accept*), all the symbols up to L are erased from the stack, and guessing of the right branch of the corresponding universal move is started. This continues until the stack becomes empty.

It remains to define the formulae *Init*, *Accept* and *Move* checking correctness of the simulation performed by Δ . We put

$$\text{Init} \equiv \mathcal{P}^{>0} \mathcal{X} \left(F \wedge \bigwedge_{i=1}^{n+1} (\mathcal{P}^{>0} \mathcal{X})^i X_i \right)$$

where X_i is the i -th symbol of c , and

$$\begin{aligned} \text{Accept} &\equiv \mathcal{P}^{>0} (\neg(E \vee L \vee R) \mathcal{U} q_A) \\ \text{Move} &\equiv \mathcal{P}^{>0} \mathcal{X} \left(F \wedge \bigvee_{t \in \gamma} \text{Trans}_t \right) \end{aligned}$$

The Trans_t formulae check consistency of symbols in consecutive configurations according to the transition function γ . They can be constructed using subformulae of the form

$$(\mathcal{P}^{>0} \mathcal{X})^{i+n+2} X \Rightarrow (\mathcal{P}^{>0} \mathcal{X})^i Y \quad (1)$$

which say that the i -th symbol of the current configuration (i.e., the one on the top of the stack) is Y if the i -th symbol of the previous configuration is X . We omit the construction of Trans_t from this sketch; the underlying principles are the same as in the construction presented in [29].

We obtain that if T has an accepting computation from T , then the depth-first search performed on the corresponding computation tree determines a run of Δ initiated in G which ends with the empty stack and on which the formula

$$\neg F \wedge (A \Rightarrow \text{Accept}) \wedge (M \Rightarrow \text{Move})$$

is satisfied in every configuration, and Init is satisfied in the first configuration which has M on the top of the stack. Conversely, if $G \models^\nu \Phi$, then there is a run of Δ ending in ε which uniquely determines an accepting computation tree. Thus, we obtain that $G \models^\nu \Phi$ iff T has an accepting computation from c . \square

Theorem 5.2. *The model-checking problem for pBPA and qPCTL* is 2-EXPTIME-hard.*

Proof. Our proof is similar to the one presented in [5]. It proceeds by reduction from the acceptance problem for exponentially space bounded alternating Turing machines, which is known to be **2-EXPTIME**-complete [14]. We indicate how to modify the construction used in the proof of Theorem 5.1 to obtain **2-EXPTIME**-hardness of the model-checking problem for pBPA and qPCTL*, using the ideas of [5].

As in the proof of Theorem 5.1, we consider an alternating Turing machine $T = (C, \Upsilon, \gamma, q_I, q_A, q_R, \lambda)$. This time, however, we assume that T uses at most $n = 2^m$ tape symbols for an input word of length m . The initial configuration is a word c of size $n = 2^m$ starting with the input word and padded by $2^m - m$ empty-space symbols. For a given T and an input word of length m , we construct a pBPA Δ , a simple valuation ν , and a qPCTL* formula Φ such that $G \models^\nu \Phi$ for a distinguished stack symbol G iff T has an accepting computation from the initial configuration c .

The main obstacle in adapting the construction used in the proof of Theorem 5.1 is that configurations are now of exponential length. Hence, the formulae verifying the correctness of one computational step constructed in the style of (1) would require exponentially many \mathcal{X} operators. Similarly as in [5], this problem is overcome by using an extended representation of configurations of T . A configuration of the form $Y_1 Y_2 \dots Y_k$ is now represented by the string $Y_1[1]Y_2[2] \dots Y_k[k]$ where each $[i]$ is a binary representation of i . We assume that the length of $[i]$ is exactly m , i.e., $[i]$ is the standard binary representation of i filled with an appropriate number of 0's from the left.

Now, we define a pBPA $\Delta = (\Gamma, \delta, \text{Prob})$ where Γ contains the symbols of $C \cup \Upsilon$, fresh symbols $E, L, R, G, M, A, F, 0, 1, V, U$, and some auxiliary counting symbols. The meaning of E, L, R, G, M, A, F is the same as in the proof of

Theorem 5.1. The symbols 0 and 1 are used to encode the binary strings $[i]$ (see above), and the purpose of V and U is explained later. Transition rules of Δ are similar as in the proof of Theorem 5.1 except that the configurations are guessed in the new format, and the symbols of $C \cup \Upsilon$ rewrite either to V or U before the size of the stack is decreased. Thus, we obtain the following:

$$\begin{array}{ll} G & \rightarrow Mc'L \mid Mc'E \mid A & X & \rightarrow V \mid U \\ M & \rightarrow G \mid F & Y & \rightarrow \varepsilon \\ L & \rightarrow Mc'R \mid F \end{array}$$

Here $X \in C \cup \Upsilon$, $Y \in \{R, A, E, F, V, U, 0, 1\}$ and c' is a string of the form $((C \cup \Upsilon) \cdot \{0, 1\}^m)^+$ containing precisely one letter from C . Obviously, such strings can be generated by polynomially many rules using polynomially many auxiliary counting symbols, and hence we can safely use the symbolic rules $G \rightarrow Mc'L \mid Mc'E$ and $L \rightarrow Mc'R$.

The formula Φ is of a similar form as before, i.e.,

$$\Phi \equiv \mathcal{P}^{>0} \left(\text{Init} \wedge \left((\neg F \wedge (A \Rightarrow \text{Accept}) \wedge (M \Rightarrow \text{Move})) \mathcal{U} \varepsilon \right) \right)$$

where Accept is also the same as before, i.e.,

$$\text{Accept} \equiv \mathcal{P}^{>0} (\neg(E \vee L \vee R) \mathcal{U} q_A).$$

However, the formulae Init and Move are slightly different. We define

$$\text{Move} \equiv \mathcal{P}^{>0} \mathcal{X} \left(F \wedge \text{Count} \wedge \bigvee_{t \in \gamma} \text{Trans}_t \right)$$

Here, the formula Count checks the consistency of the $[i]$ indexes, and Trans_t checks whether the topmost configuration is a t -successor of the previous configuration (here $t \in \gamma$ is a transition of T).

The simulation is initiated in G , and proceeds by guessing a computation tree of T . Every configuration is guessed as a string of the form $((C \cup \Upsilon) \cdot \{0, 1\}^m)^+$. As before, each time a new configuration is guessed, the run of Δ has to pass through a configuration with M on the top of the stack and the correctness is checked by the formula Move (here Count checks whether the string on the top of the stack is a correct encoding of a configuration). The formula Init checks that the first guessed configuration is the initial configuration. When an accepting

configuration is pushed on the stack and checked (by the formula *Accept*), all the symbols up to L are erased from the stack, and guessing of the right branch of the corresponding universal move is started. This continues until the stack becomes empty.

Now we define the new formulae more precisely. The formula *Init* is defined as follows:

$$Init \equiv (\neg M) \mathcal{U} \left(M \wedge \mathcal{P}^{>0} \mathcal{X} \left(F \wedge \left(\bigwedge_{i=1}^m \varphi_i \right) \wedge \varphi_{empty} \right) \right)$$

where φ_i checks that $[i]$ is followed by the i -th symbol of the initial configuration c , and φ_{empty} checks that all symbols after the m -th symbol of the encoded configuration are empty-space symbols. These conditions can clearly be expressed in qPCTL*.

A configuration of the form $F X_1 v_1 X_2 v_2 \dots X_n v_n \beta$, where $v_i \in \{0, 1\}^m$ and $X_i \in C \cup \Upsilon$, satisfies the formula *Count* iff every v_i is equal to $[i]$. The formula *Count* can be constructed using subformulae of the following form (where $X \in C \cup \Upsilon$ and k is between 1 and m):

$$(U \vee V) \wedge \bigwedge_{j=1}^{k-1} \left((\mathcal{X}^j 0 \Rightarrow \mathcal{X}^{m+2+j} 0) \wedge (\mathcal{X}^j 1 \Rightarrow \mathcal{X}^{m+2+j} 1) \right) \wedge \\ \left(\mathcal{X}^k 1 \wedge \bigwedge_{i=k+1}^m \mathcal{X}^i 0 \right) \Rightarrow \mathcal{X}^{m+2} \left(\mathcal{X}^k 0 \wedge \bigwedge_{i=k+1}^m \mathcal{X}^i 1 \right)$$

Intuitively, the formula says that if the binary representation of a number at some unspecified position is of the form $v10^{m-k-1}$, then the binary representation of the number just below is of the form $v01^{m-k-1}$, i.e., the two strings encode two consecutive numbers.

A configuration of the form $F c' D c'' \beta$, where $D \in \{L, R, E\}$, satisfies the formula *Trans_t* iff the current configuration of T , encoded by c' , is a t -successor of the previous configuration of T , encoded by c'' . This is achieved in the standard way, i.e., by checking that each triple of three consecutive symbols in the configuration encoded by c'' is *compatible* (w.r.t. the move t) with the triple of three consecutive symbols at the corresponding position in the configuration encoded by c' . This condition can be verified using subformulae that are true in configurations of the form $F c' \beta'$ iff all runs initiated in $F c' \beta'$ that satisfy the conditions A-D given below also satisfy the condition (E) that the six symbols $\bar{X}_1, \bar{X}_2, \bar{X}_3$,

$\hat{X}_1, \hat{X}_2, \hat{X}_3$ of $C \cup \Upsilon$ satisfying the following are compatible w.r.t. t : \bar{X}_1 is visited immediately before entering the first configuration with V on the top of the stack, and \bar{X}_2 and \bar{X}_3 are visited immediately after that; \hat{X}_1 is visited immediately before entering the second configuration with V on the top of the stack, and \hat{X}_2 and \hat{X}_3 are visited immediately after that. The conditions A-D say the following:

- (A) Every symbol is removed as soon as possible from the top of the stack (in particular, L rewrites to F).
- (B) At least two symbols from L, R, E are visited, i.e. the configuration $Fc'\beta'$ is of the form $Fc'Dc''\beta$, where $D \in \{L, R, E\}$, and $c \in (C \cup \Upsilon)^*$.
- (C) The run visits precisely two configurations with V on the top of the stack: the first such configuration is visited when removing the symbols of c' (i.e., the encoding of the current configuration of T) and the second one is visited when removing the symbols of c'' (i.e., the encoding of the previous configuration of T).
- (D) The binary number visited right after a configuration with V on the top of the stack is the same in both cases, and is lower than $n - 1$

The conditions A–D are clearly expressible using some path PCTL* formulae $\varphi_A, \varphi_B, \varphi_C$ and φ_D . The condition E can be expressed using a formula φ_E which is a disjunction (over all compatible tuples $\bar{X}_1, \bar{X}_2, \bar{X}_3, \hat{X}_1, \hat{X}_2, \hat{X}_3$) of the formulae

$$\left((-V) \mathcal{U} \bar{\varphi}(\bar{X}_1, \bar{X}_2, \bar{X}_3) \right) \wedge \left((-V) \mathcal{U} \left(V \wedge \mathcal{X} \left((-V) \mathcal{U} \bar{\varphi}(\hat{X}_1, \hat{X}_2, \hat{X}_3) \right) \right) \right)$$

where

$$\bar{\varphi}(X_1, X_2, X_3) \equiv (\mathcal{X}V) \wedge X_1 \wedge (\mathcal{X}^{m+2}X_2) \wedge (\mathcal{X}^{2\cdot m+4}X_3)$$

We then put

$$Trans_t \equiv (\varphi_A \wedge \varphi_B \wedge \varphi_C \wedge \varphi_D) \Rightarrow \varphi_E$$

Let us now explain the intuition behind $Trans_t$. Thanks to the formula $Count$ and the condition B , we can suppose that the configuration $Fc'\beta'$ is of the form

$$FX_1[1]X_2[2] \dots X_n[n]DX'_1[1]X'_2[2] \dots X'_n[n]\beta,$$

and it is our goal to check that the symbols $X_i, X_{i+1}, X_{i+2}, X'_i, X'_{i+1}, X'_{i+2}$ are compatible for every $1 \leq i \leq n - 2$. If a run initiated in $Fc'\beta'$ satisfies the

conditions A–D above, then there is i such that the following two configurations are visited before visiting the configuration β :

$$\begin{array}{c} X_i[i]X_{i+1}[i+1] \dots X_n[n]DX'_1[1]X'_2[2] \dots X'_n[n]\beta \\ X'_i[i]X'_{i+1}[i+1] \dots X'_n[n]\beta \end{array}$$

and the suffix of the run initiated in the first and second configurations above satisfies $\varphi(X_i, X_{i+1}, X_{i+2})$ and $\varphi(X'_i, X'_{i+1}, X'_{i+2})$, respectively. The opposite also holds, i.e. for every i there is a run satisfying A–D such that the above two configurations are visited. Hence, it indeed suffices to concentrate on these runs and check that the symbols adjacent to the two occurrences of V are compatible. The symbols U and V here are used to “mark” positions in a run to be checked. When elements of $C \cup \Upsilon$ are removed from the stack, they are either overwritten to V (a mark is set), or to U (no mark is set). Due to the probabilistic nature of the pBPA we have no way of controlling where the marks are set, but we can use the conditions A–D to restrict to runs where marks were set correctly, and on these runs we use the condition E to enforce the compatibility of the symbols. This technical step is needed, because we need to compare parts of the stack which are separated by exponential number of symbols, and so we cannot hard-code the comparison using multiple occurrences of the \mathcal{X} operator as we did in the proof of Theorem 5.1.

As in the previous proof we omit the technical details of the construction, because the underlying principles are the same as in [5]. \square

6. Conclusions

We presented tight decidability and complexity results for model-checking probabilistic pushdown automata against branching-time probabilistic logics PCTL, PCTL*, and their qualitative fragments. The only problem left open is the decidability of the model-checking problem for stateless probabilistic pushdown automata and the logic PCTL.

References

References

- [1] *Proceedings of STACS 2005*, volume 3404 of *Lecture Notes in Computer Science*. Springer, 2005.

- [2] P.A. Abdulla, C. Baier, S.P. Iyer, and B. Jonsson. Reasoning about probabilistic channel systems. In *Proceedings of CONCUR 2000*, volume 1877 of *Lecture Notes in Computer Science*, pages 320–330. Springer, 2000.
- [3] C. Baier. *On the Algorithmic Verification of Probabilistic Systems*. Habilitation, Universität Mannheim, 1998.
- [4] C. Baier and B. Engelen. Establishing qualitative properties for probabilistic lossy channel systems: an algorithmic approach. In *Proceedings of 5th International AMAST Workshop on Real-Time and Probabilistic Systems (ARTS'99)*, volume 1601 of *Lecture Notes in Computer Science*, pages 34–52. Springer, 1999.
- [5] L. Bozzelli. Complexity results on branching-time pushdown model checking. *Theoretical Computer Science*, 379(1–2):286–297, 2007.
- [6] T. Brázdil. *Verification of Probabilistic Recursive Sequential Programs*. PhD thesis, Masaryk University, Faculty of Informatics, 2007.
- [7] T. Brázdil, V. Brožek, J. Holeček, and A. Kučera. Discounted properties of probabilistic pushdown automata. In *Proceedings of LPAR 2008*, volume 5330 of *Lecture Notes in Computer Science*, pages 230–242. Springer, 2008.
- [8] T. Brázdil, V. Brožek, K. Etessami, A. Kučera, and D. Wojtczak. One-counter Markov decision processes. In *Proceedings of SODA 2010*, pages 863–874. SIAM, 2010.
- [9] T. Brázdil, V. Brožek, and V. Forejt. Branching-time model-checking of probabilistic pushdown automata. *Electronic Notes in Theoretical Computer Science*, 239:73–83, 2009.
- [10] T. Brázdil, V. Brožek, V. Forejt, and A. Kučera. Reachability in recursive Markov decision processes. *Information and Computation*, 206(5):520–537, 2008.
- [11] T. Brázdil, V. Brožek, A. Kučera, and J. Obdržálek. Qualitative reachability in stochastic BPA games. In *Proceedings of STACS 2009*, volume 3 of *Leibniz International Proceedings in Informatics*, pages 207–218. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2009.

- [12] T. Brázdil, J. Esparza, S. Kiefer, and A. Kučera. Analyzing probabilistic pushdown automata. *Formal Methods in System Design*. DOI 10.1007/s10703-012-0166-0.
- [13] T. Brázdil, A. Kučera, and O. Stražovský. On the decidability of temporal properties of probabilistic pushdown automata. In *Proceedings of STACS 2005* [1], pages 145–157.
- [14] Ashok K. Chandra, Dexter Kozen, and Larry J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981.
- [15] J. Esparza, A. Kučera, and R. Mayr. Model-checking probabilistic pushdown automata. In *Proceedings of LICS 2004*, pages 12–21. IEEE Computer Society Press, 2004.
- [16] J. Esparza, A. Kučera, and R. Mayr. Quantitative analysis of probabilistic pushdown automata: Expectations and variances. In *Proceedings of LICS 2005*, pages 117–126. IEEE Computer Society Press, 2005.
- [17] J. Esparza, A. Kučera, and R. Mayr. Model-checking probabilistic pushdown automata. *Logical Methods in Computer Science*, 2(1:2):1–31, 2006.
- [18] J. Esparza, A. Kučera, and S. Schwoon. Model-checking LTL with regular valuations for pushdown systems. *Information and Computation*, 186(2):355–376, 2003.
- [19] K. Etessami and M. Yannakakis. Algorithmic verification of recursive probabilistic systems. In *Proceedings of TACAS 2005*, volume 3440 of *Lecture Notes in Computer Science*, pages 253–270. Springer, 2005.
- [20] K. Etessami and M. Yannakakis. Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations. In *Proceedings of STACS 2005* [1], pages 340–352.
- [21] K. Etessami and M. Yannakakis. Recursive Markov decision processes and recursive stochastic games. In *Proceedings of ICALP 2005*, volume 3580 of *Lecture Notes in Computer Science*, pages 891–903. Springer, 2005.
- [22] K. Etessami and M. Yannakakis. Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations. *Journal of the Association for Computing Machinery*, 56, 2009.

- [23] K. Etessami and M. Yannakakis. Model checking of recursive probabilistic systems. *ACM Transactions on Computational Logic*, 13, 2012.
- [24] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6:512–535, 1994.
- [25] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.
- [26] S.P. Iyer and M. Narasimha. Probabilistic lossy channel systems. In *Proceedings of TAPSOFT'97*, volume 1214 of *Lecture Notes in Computer Science*, pages 667–681. Springer, 1997.
- [27] J. G. Kemeny and J. L. Snell. *Finite Markov Chains: With a New Appendix "Generalization of a Fundamental Matrix"*. Springer, first edition, 1983.
- [28] J. G. Kemeny, J. L. Snell, A. W. Knapp, and D.S. Griffeath. *Denumerable Markov Chains*. Springer, second edition, 1976.
- [29] I. Walukiewicz. Model checking CTL properties of pushdown systems. In *Proceedings of FST&TCS'2000*, volume 1974 of *Lecture Notes in Computer Science*, pages 127–138. Springer, 2000.