Model Checking Temporal Properties of Recursive Probabilistic Programs

Tobias Winkler, Christina Gehnen, and Joost-Pieter Katoen

RWTH Aachen University, Aachen, Germany
{tobias.winkler,katoen}@cs.rwth-aachen.de
christina.gehnen@rwth-aachen.de

Abstract. Probabilistic pushdown automata (pPDA) are a standard operational model for programming languages involving discrete random choices, procedures, and returns. Temporal properties are useful for gaining insight into the chronological order of events during program execution. Existing approaches in the literature have focused mostly on ω-regular and LTL properties. In this paper, we study the model checking problem of pPDA against ω-visibly pushdown languages that can be described by specification logics such as CaRet and are strictly more expressive than ω-regular properties. With these logical formulae, it is possible to specify properties that explicitly take the structured computations arising from procedural programs into account. For example, CaRet is able to match procedure calls with their corresponding future returns, and thus allows to express fundamental program properties like total and partial correctness.

Keywords: Probabilistic Recursive Programs · Model Checking · Probabilistic Pushdown Automata · Visibly Pushdown Languages · CaRet.

1 Introduction

Probabilistic programs extend traditional programs with the ability to flip coins or, more generally, sample values from probability distributions. These programs can be used to encode randomized algorithms and randomized mechanisms in security [7] in a natural way. The interest in probabilistic programs has significantly increased in recent years. To a large extent, this is due to the search in AI for more expressive and succinct languages than probabilistic graphical models for Bayesian inference [17]. Probabilistic programs have many applications [24]. They are used in, amongst others, machine learning, systems biology, security, planning and control, quantum computing, and software–defined networks. Probabilistic variants of many programming languages exist.

Procedural programs allow for declaration of procedures—small independent code blocks—and the ability to *call* procedures from one another, possibly in

This work is supported by the DFG research training group 2236 UnRAVeL and the ERC advanced research grant 787914 FRAPPANT.

```
proc void infectYoung() {              proc bool infectElder() {
  y := uniform(0, 3)                     y := uniform(0, 1)
  repeat y times {                       repeat y times {
    infectYoung()   }                      infectYoung()   }
  e := uniform(0, 2)                     e := uniform(0, 4)
  repeat e times {                       repeat e times {
    f := infectElder()   }                 infectElder()   }
  return   }                             f := bernoulli(0.01); return f   }
```

Fig. 1. Recursive probabilistic program modeling the outbreak of an infectious disease. uniform(a, b) stands for the *discrete* uniform distribution on $[a, b]$.

a recursive fashion. Most common programming languages such as C, Python, or Java support procedures. It is thus not surprising that recursion is a key ingredient in many modern probabilistic programming languages (PPL). In fact, many early approaches to extend Bayesian networks focused on incorporating recursion [26,19,11,27]. Randomized algorithms such as Hoare's quicksort with random pivot selection can be straightforwardly programmed using recursion. Recursion is also a first-class citizen in modeling rule-based dependencies between molecules or populations in systems biology (e.g., modeling reproduction).

This paper studies the automated verification of *probabilistic pushdown automata* [14] (pPDA) as an explicit-state operational model of procedural probabilistic programs against temporal specifications. As a motivating example, let us consider a simple epidemiological model for the outbreak of an infectious disease in a large population where the number of susceptible individuals can be assumed to be infinite. Our example model distinguishes young and elderly persons. Each affected individual infects a uniformly distributed number of others, with varying rates (expected values) according to the age groups (Figure 2). The fatality rate for infected elderly and young persons is 1% and 0%, respectively. Initially, we assume there is a single infected young person, i.e., the overall program is started by calling *infectYoung*(). It is an easy task for any working programmer to specify this model as a discrete probabilistic program with mutually recursive procedures (Figure 1). Note that this program can be easily amended to more realistic models involving, e.g., more age or gender groups, other distributions, hospitalization rate, etc.

	Y	E
Y	1.5	1
E	0.5	2

Fig. 2. Example infection rates by age groups.

The operational behavior of programs such as the one in Figure 1 can be naturally described by pPDA. The technical details of such a translation are beyond the scope of this paper but let us provide some intuition (more details can be found e.g. in [2]). Roughly, the *local* states of the procedures—the valuation of the local variables and the position of the program counter—constitute both the state space and the stack alphabet of the automaton. Procedure calls correspond to push transitions in the automaton in such a way that the program's *procedure* *stack* is simulated by the automaton's pushdown stack, i.e., the current local state is saved on top of the stack. Accordingly, *returning* from a procedure corresponds to taking a pop transition in order to restore the local state of the caller. Returning a value can be handled similarly. Clearly, if the reachable local state spaces of the involved procedures are finite, then the resulting automaton will be finite as well.

A number of relevant questions such as "Will the virus eventually become extinct?" (termination probability) or "What is the expected number of fatalities?" (expected costs) can be decided on finite pPDA (see [9] for a survey). In this work, we focus on *temporal properties*, e.g., questions that involve reasoning about the chronological order of certain events of interest during the epidemic. An example are chains of infection: For instance, we might ask

What is the probability that eventually a young person with only young persons in their chain of infection passes the virus on to an elderly person who then dies?

On the level of the program in Figure 1, this corresponds to the probability of reaching a *global* program configuration where the call stack only contains *infectYoung*() invocations and during execution of the current *infectYoung*(), the local variable f is eventually set to true. This requires reasoning about the nestings of calls and returns of a computation. In fact, in order to decide if $f = \mathit{true}$ in the current procedure, we must "skip" over all calls within it and only consider their local return values. This requirement and many others can be rather naturally expressed in the logic *CaRet* [3], an extension of LTL:

$$\Diamond^g \left(\Box^- p_Y \;\wedge\; p_Y \;\wedge\; \Diamond^a f \right) .$$

Here, p_Y is an atomic proposition that holds at states which correspond to being in procedure *infectYoung*, and f indicates that $f = \mathit{true}$. Intuitively, the above formula states that eventually (outer $\Diamond^g$), the computation reaches a (global) state where only *infectYoung* is on the call stack and the current procedure is *infectYoung* as well ($\Box^- p_Y \wedge p_Y$), and moreover the local—aka *abstract*—path *within* in the current procedure reaches a state where f is true ($\Diamond^a f$). Such properties are in general context-free but not always regular and thus cannot be expressed in LTL [3].

Technical Contribution. We are given a (finite) pPDA Δ and a CaRet formula φ and we are interested in determining the probability that a random trajectory of Δ satisfies φ. In order for this problem to be decidable [13], we need to impose a mild *visibility* restriction on Δ, yielding a probabilistic *visibly* pushdown automaton (pVPA). Just like several previous works on model checking pPDA against ω-regular specifications [14,10,21], we follow the automata-based approach (see Figure 3). More specifically, we first translate φ into an equivalent non-deterministic *Büchi visibly pushdown automaton* [4] (VPA) $\mathcal{A}$ and then determinize it using a result of [22]. The resulting DVPA $\mathcal{D}$ uses a so-called *stair-parity* [22] acceptance condition that is strictly more expressive than standard parity or Muller DVPA [4]. Stair-parity differs from usual parity in that

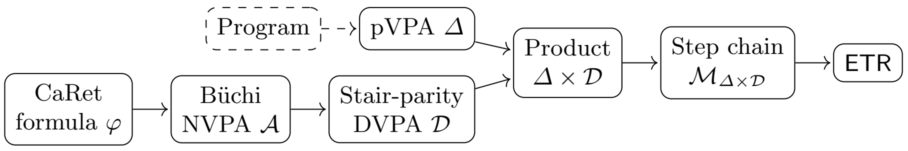

Program
pVPA Δ
CaRet formula φ
Büchi NVPA $\mathcal{A}$
Stair-parity DVPA $\mathcal{D}$
Product $\Delta \times \mathcal{D}$
Step chain $\mathcal{M}_{\Delta \times \mathcal{D}}$
ETR

Fig. 3. Chain of reductions used in this paper. ETR stands for *existential theory of the reals*, i.e., the existentially quantified fragment of the FO-theory over $(\mathbb{R}, +, \cdot, \leq)$.

it only considers certain positions—called *steps* [22]—of an infinite word where the stack height never decreases again. We then construct a standard product $\Delta \times \mathcal{D}$. Here, the visibility conditions ensure that the automata synchronize their stack actions, yielding a product automaton that uses *a single stack* instead of two independent ones, which would lead to undecidability [13]. Finally, we are left with computing a stair-parity acceptance probability in the product, which is itself a pPDA. This is achieved by constructing a specific finite Markov chain associated to $\Delta \times \mathcal{D}$, called *step chain* in this paper. Intuitively, the step chain jumps from one step of a run to the next, and therefore we only need to evaluate *standard parity* rather than stair-parity on the step chain. The idea of step chains is due to [14] where they were used to show decidability against deterministic non-pushdown Büchi automata. For constructing the step chain, certain *termination probabilities* of the pPDA need to be computed. These are in general algebraic numbers that cannot always be expressed by radicals [16], let alone by rationals. However, the relevant problems are still decidable via an encoding in the existential fragment of the FO-theory of the reals (ETR) [21].

The resulting main contributions of this paper are complexity results, summarized in Figure 4, and algorithms for quantitative model checking of pPDA against ω-VPL given in terms of either deterministic automata, non-deterministic automata, or as CaRet formulae. As common in the literature, we consider the special case of qualitative, or *almost-sure* (a.-s.), model checking separately. To the best of our knowledge, none of these problems was known to be decidable before. The work of [13] proved decidability of model checking against deterministic Muller VPA which capture a strict subset of the CaRet-definable languages [4]. As a lemma of independent interest, we show that the step chain can be used for checking all kinds of measurable properties defined on steps, even beyond parity.

Related work. We have already mentioned various works on recursion in probabilistic graphical models (and PPL) as well as on verifying pPDA and the equivalent model of recursive Markov chains [16]. The analysis of these models focuses on reachability probabilities, ω-regular properties or (fragments of) probabilistic CTL, expected costs, and termination probabilities. The computation of termination probabilities in recursive Markov chains and variations thereof with non-determinism is supported by the software tool PReMo [29]. Our paper can be seen as a natural extension from checking pPDA against ω-regular properties to ω-visibly pushdown languages. In contrast to these algorithmic approaches, various deductive reasoning methods have been developed for recursive

ω-VPL given in terms of ...	*qualitative*	*quantitative*
Deterministic stair-parity VPA [Theorem 3]	in PSPACE	in PSPACE
Non-deterministic Büchi VPA [Theorem 4]	EXPTIME-compl.	in EXPSPACE
CaRet formula [Theorem 5]	in 2EXPTIME	in 2EXPSPACE

Fig. 4. Complexity results of this paper.

probabilistic programs. Proof rules for recursion were first provided in [20], and later extended to proof rules in a weakest-precondition reasoning style [23,25]. Olmedo *et al.* [25] also address the connection to pPDA and provide proof rules for expected run-time analysis. A mechanized method for proving properties of randomized algorithms, including recursive ones, for the Coq proof assistant is presented in [5]. The Coq approach is based on higher–order logic using a monadic interpretation of programs as probabilistic distributions.

Organization. We review the basics about VPA and CaRet in Section 2. Section 3 introduces probabilistic *visibly* pushdown automata (pVPA). The stair-parity DVPA model checking procedure is presented in Section 4, and the results for Büchi VPA and CaRet in Section 5. We conclude the paper in Section 6.

2 Visibly Pushdown Languages

We fix some general notation for words first. Given a non-empty alphabet Σ, let Σ^* be the set of finite words (this includes the empty word ϵ), and let Σ^ω be the set of infinite words over Σ. For $i \geq 0$, the i-th symbol of a word $w \in \Sigma^* \cup \Sigma^\omega$ is denoted $w(i)$ if it exists. $|w|$ denotes the length of w.

2.1 Visibly Pushdown Automata

A finite alphabet Σ is called a *pushdown alphabet* if it is equipped with a partition $\Sigma = \Sigma_{\mathsf{call}} \uplus \Sigma_{\mathsf{int}} \uplus \Sigma_{\mathsf{ret}}$ into three—possibly empty—subsets of *call*, *internal*, and *return* symbols. A *visibly pushdown automaton* [4] (VPA) over Σ is like a standard pushdown automaton with the additional syntactic restriction that reading a call or return symbol triggers a push or a pop transition, respectively. Reading an internal symbol, on the other hand, does not affect the stack at all.

Definition 1 (VPA [4]). *Let Σ be a pushdown alphabet. A* visibly pushdown automaton (VPA) *over Σ is a tuple $\mathcal{A} = (S, s_0, \Gamma, \bot, \delta, \Sigma)$ with S a finite set of states, $s_0 \in S$ an initial state, Γ a finite stack alphabet, $\bot \in \Gamma$ a special bottom-of-stack symbol, and $\delta = (\delta_{\mathsf{call}}, \delta_{\mathsf{int}}, \delta_{\mathsf{ret}})$ a triple of relations*

$$\delta_{\mathsf{call}} \subseteq (S \times \Sigma_{\mathsf{call}}) \times (S \times \Gamma_{\text{-}\bot}) \ , \quad \delta_{\mathsf{int}} \subseteq (S \times \Sigma_{\mathsf{int}}) \times S \ , \quad \delta_{\mathsf{ret}} \subseteq (S \times \Sigma_{\mathsf{ret}} \times \Gamma) \times S$$

where $\Gamma_{\text{-}\bot} = \Gamma \setminus \{\bot\}$. For $s, t \in S$, $Z \in \Gamma$, and $a \in \Sigma$, we use the shorthand notations $s \xrightarrow{a} tZ$, $s \xrightarrow{a} t$, $sZ \xrightarrow{a} t$ to indicate that there exist transitions $(s, a, t, Z) \in \delta_{\mathsf{call}}$, $(s, a, t) \in \delta_{\mathsf{int}}$, $(s, a, Z, t) \in \delta_{\mathsf{ret}}$, respectively. Note that e.g. $s \xrightarrow{a} tZ$ implies implicitly that $a \in \Sigma_{\mathsf{call}}$ and $Z \neq \bot$, and similar for internal and return transitions. Intuitively, call transitions push a new symbol Z onto the stack, internal transitions ignore the stack, and return transitions pop the topmost symbol Z from the stack (unless $Z = \bot$, in which case nothing is popped). A *configuration* of VPA $\mathcal{A}$ is a tuple $(s, \gamma) \in S \times \Gamma^*$, written more succinctly as $s\gamma$ in the sequel. Let $w \in \Sigma^\omega$ be an infinite input word. An infinite sequence $\rho = s_0\gamma_0, s_1\gamma_1 \ldots$ of configurations is called a *run* of $\mathcal{A}$ on w if $s_0\gamma_0 = s_0\bot$ and for all $i \geq 0$, exactly one of the following cases applies:

- $w(i) \in \Sigma_{\mathsf{call}}$ and $\gamma_{i+1} = \gamma_i Z$ for some $Z \in \Gamma_{\text{-}\bot}$ such that $s_i \xrightarrow{w(i)} s_{i+1}Z$; or
- $w(i) \in \Sigma_{\mathsf{int}}$ and $\gamma_{i+1} = \gamma_i$ and $s_i \xrightarrow{w(i)} s_{i+1}$; or
- $w(i) \in \Sigma_{\mathsf{ret}}$ and $\gamma_{i+1}Z = \gamma_i$ for some $Z \in \Gamma_{\text{-}\bot}$ such that $s_i Z \xrightarrow{w(i)} s_{i+1}$, or $\gamma_i = \gamma_{i+1} = \bot$ and $s_i\bot \xrightarrow{w(i)} s_{i+1}$.

A *Büchi acceptance condition* for $\mathcal{A}$ is a subset $F \subseteq S$. A VPA equipped with a Büchi condition is called a *Büchi VPA*. An infinite word $w \in \Sigma^\omega$ is accepted by a Büchi VPA if there exists a run $s_0\gamma_0, s_1\gamma_1, \ldots$ of $\mathcal{A}$ on w such that $s_i \in F$ for infinitely many $i \geq 0$. The ω-language of words accepted by a Büchi VPA $\mathcal{A}$ is denoted $\mathcal{L}(\mathcal{A}) \subseteq \Sigma^\omega$.

Definition 2 (ω-VPL [4]). *Let Σ be a pushdown alphabet. $L \subseteq \Sigma^\omega$ is an* ω-visibly pushdown language (ω-VPL) *if $L = \mathcal{L}(\mathcal{A})$ for a Büchi VPA $\mathcal{A}$ over Σ.*

A VPA is *deterministic* (DVPA) if it has exactly one run on each input word. In this case, $\delta_{\mathsf{call}}, \delta_{\mathsf{int}}$, and δ_{ret} can be viewed as (total) functions. As for standard NBA, the class of languages recognized by Büchi DVPA is a strict subset of the languages recognized by non-deterministic Büchi VPA. Unlike in the non-pushdown case, DVPA with Muller or parity conditions are also strictly less expressive than non-deterministic Büchi VPA [4]. A deterministic automaton model for ω-VPL was given in [22]. It uses a so-called *stair-parity* acceptance condition which is the topic of the next subsection.

2.2 Steps and Stair-parity Conditions

Let us fix a pushdown alphabet Σ and a VPA $\mathcal{A}$ over Σ. Consider a run $\rho = s_0\gamma_0, s_1\gamma_1, \ldots$ of $\mathcal{A}$ on an infinite word $w \in \Sigma^\omega$. We define the *stack height* of the i-th configuration as $sh(\rho(i)) = |\gamma_i| - 1$ (the bottom symbol $\bot$ does not count to the stack height). The stair-parity condition relies on the notion of *steps*:

Definition 3 (Step). *Let ρ be a run of $\mathcal{A}$. Position $i \geq 0$ is a* step *of ρ if*

$$\forall n \geq i: \quad sh(\rho(n)) \;\geq\; sh(\rho(i)) \;.$$

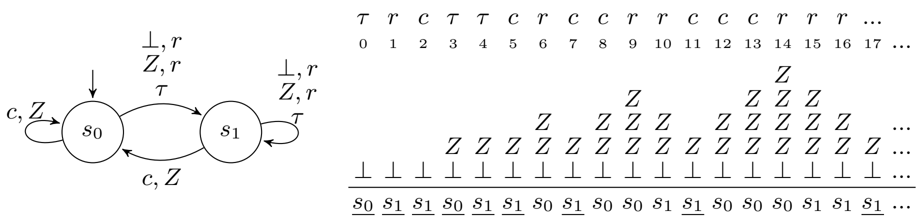

$\bot, r$
Z, r
τ
c, Z
s_0
s_1
$\bot, r$
Z, r
τ
c, Z
$\tau\ r\ c\ \tau\ \tau\ c\ r\ c\ c\ r\ r\ c\ c\ c\ r\ r\ r\ \ldots$
$0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15\ 16\ 17\ \ldots$
$\underline{s_0}\ \underline{s_1}\ \underline{s_1}\ \underline{s_0}\ \underline{s_1}\ \underline{s_1}\ s_0\ \underline{s_1}\ s_0\ s_0\ s_1\ \underline{s_1}\ s_0\ s_0\ s_0\ s_1\ s_1\ \underline{s_1}\ \ldots$

Fig. 5. Left: An example VPA (in fact, a DVPA) with $\Gamma = \{ Z, \bot \}$ over input alphabet $\Sigma = \{ c \} \uplus \{ \tau \} \uplus \{ r \}$. Transitions labeled c, Z are *call transitions* which push Z on the stack, the transitions labeled with τ are *internal* ones that ignore the stack, and those labeled Z, r and $\bot, r$ are *return transitions* that are only enabled if Z ($\bot$, resp.) is on top of the stack; when executing Z, r we also pop Z from the stack. However, the special bottom-of-stack symbol $\bot$ can never be popped (see e.g. pos. 1). Right: The *unique* run of the DVPA on input word $\tau\, r\, c\, \tau\, \tau\, c\, r\, c^2\, r^2\, c^3\, r^3 \ldots$. Steps are underlined.

Abusing terminology, we may also refer to the configurations at the step positions of a run as steps.

Example 1. Figure 5 depicts a DVPA and the initial fragment of its unique run ρ on the input word $\tau\, r\, c\, \tau\, \tau\, c\, r\, c^2\, r^2\, c^3\, r^3 \ldots$. The step positions are underlined, i.e., positions 0-5, 7, 11, and 17 are steps. Note that if $\rho(i) = s\bot$ for some $s \in S$ then i is a step, i.e., *bottom configurations* are always steps.

Steps play a central role in the rest of the paper. We therefore explain some of their fundamental properties.

– If positions $i < j$ are adjacent steps, then $sh(\rho(j)) - sh(\rho(i)) \in \{ 0, 1 \}$, i.e., the stack height from one step to the next increases by either zero or one. More precisely, if the symbol at step position i is internal (e.g. $i = 0, 3, 4$ in Figure 5) or a return (e.g. $i = 1$) then the next step is simply the next configuration $j = i + 1$ and the stack height does not increase. If the symbol at position i is a call, then one of two cases occurs: Either the call has no matching future return (e.g. $i = 2$); in this case, the next step is the next configuration $j = i + 1$. Otherwise the call is eventually matched (e.g. $i = 5, 7, 11$) and the next step $j > i + 1$ occurs after the corresponding matching return is read and has the same stack height.
– Each infinite run has infinitely many steps since the above discussion also implies that each step has a successor. Notice though that the difference between two adjacent step positions may grow unboundedly as in the example.
– As a consequence, the stack height at the steps either grows unboundedly or eventually stabilizes (the latter occurs in Figure 5).

Remark 1. One can also define the steps of a *word* $w \in \Sigma^\omega$ as the positions where a run of *any arbitrary* VPA on w has a step. Due to the visibility restriction, the actual behaviour of the VPA does not influence the step positions [22]. In other words, the step positions are predetermined by the input word. Thus, we can also speak of the stack height $sh(w(i))$ of word w at position i.

We need one last notion before defining stair-parity. The *footprint* of an infinite run $\rho = s_0\gamma_0, s_1\gamma_1, \ldots$ is the infinite sequence $\rho{\downarrow_{Steps}} = s_{n_0}s_{n_1}\ldots \in S^\omega$ where for all $i \geq 0$ the position n_i is the i-th step of ρ. Phrased differently, $\rho{\downarrow_{Steps}}$ is the projection of the run ρ onto the states occurring at its steps. For the example run in Figure 5 (right), $\rho{\downarrow_{Steps}} = s_0s_1s_1s_0s_1^\omega$.

Definition 4 (Stair-parity [22]). *Let $\mathcal{A}$ be a VPA over pushdown alphabet Σ. A* stair-parity *acceptance condition for $\mathcal{A}$ is defined in terms of a priority function $\Omega\colon S \to \mathbb{N}_0$. i.e. A word $w \in \Sigma^\omega$ is accepted if $\mathcal{A}$ has a run ρ on ω s.t.*

$$\min\{\, k \in \mathbb{N}_0 \mid \overset{\infty}{\exists} i\colon\ \Omega(\,\rho{\downarrow_{Steps}}(i)\,) = k \,\}$$

is even. The language accepted by $\mathcal{A}$ is denoted $\mathcal{L}(\mathcal{A})$.

Example 2. The DVPA in Figure 5 with $\Omega(s_0) = 1$ and $\Omega(s_1) = 2$ accepts

$$\mathcal{L}_{repbdd} = \{\, w \in \Sigma^\omega \mid \exists B \geq 0,\ \overset{\infty}{\exists} i \geq 0\colon\ sh(w(i)) \ \leq\ B \,\}\ ,$$

the language of *repeatedly bounded* words [22], i.e., words whose stack height (cf. Remark 1) is infinitely often at most a constant B. It is known that $\mathcal{L}_{repbdd}$ is not expressible by DVPA with usual parity conditions [4].

Theorem 1 ([22, Thm. 1]). *For every non-deterministic Büchi VPA $\mathcal{A}$ there exists a deterministic stair-parity DVPA $\mathcal{D}$ with $2^{\mathcal{O}(|S|^2)}$ states such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{D})$. Moreover, $\mathcal{D}$ can be constructed in exponential time in the size of $\mathcal{A}$.*

It was also shown in [22] that stair-parity DVPA characterize exactly the class of ω-VPL (and are thus not more expressive than non-deterministic Büchi VPA).

2.3 CaRet, a Temporal Logic of Calls and Returns

Specifying requirements directly in terms of automata is tedious in practice. *CaRet* [3] is an extension of Linear Temporal Logic (LTL) that can be used to describe ω-VPL. Its syntax is defined as follows:

Definition 5 (CaRet [3]). *Let AP be a finite set of atomic propositions. The logic CaRet adheres to the grammar*

$$\varphi \ \coloneqq\ p \mid \varphi \vee \varphi \mid \neg\varphi \mid \bigcirc^g\varphi \mid \varphi\,\mathcal{U}^g\varphi \mid \bigcirc^a\varphi \mid \varphi\,\mathcal{U}^a\varphi \mid \bigcirc^-\varphi \mid \varphi\,\mathcal{U}^-\varphi\ ,$$

where $p \in AP \cup \{\,\mathsf{call}, \mathsf{int}, \mathsf{ret}\,\}$.

Other common modalities such as $\Diamond^b$ and $\Box^b$ for $b \in \{\,g, a, -\,\}$ are defined as usual via $\Diamond^b\varphi = \mathit{true}\,\mathcal{U}^b\,\varphi$, and $\Box^b\varphi = \neg\Diamond^b\neg\varphi$. We briefly explain the semantics of CaRet, the formal definition can be found in [3] or the full version [28]. We assume familiarity with LTL. CaRet formulae are interpreted over infinite words from the pushdown alphabet $\Sigma = 2^{AP} \times \{\,\mathsf{call}, \mathsf{int}, \mathsf{ret}\,\}$. $\bigcirc^g$ and $\mathcal{U}^g$ are the standard next and until modalities from LTL (called *global* next and until

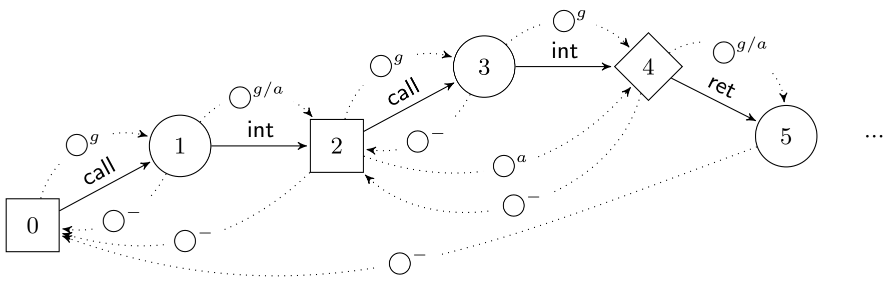


Fig. 6. CaRet's various next modalities applied to the initial fragment of an example word. Call, internal, and return positions are depicted as boxes, circles, and rhombs, resp. Note that $\bigcirc^a$ of position 3 is undefined because $\bigcirc^g$ is a return.

in CaRet). CaRet extends LTL by two key operators, the *caller* modality $\bigcirc^-$ and the *abstract successor* $\bigcirc^a$, see Figure 6. The former is a *past* modality that refers to the position of the last pending call. For internal and return symbols, the abstract successor $\bigcirc^a$ behaves like $\bigcirc^g$ unless the latter is a return, in which case $\bigcirc^a$ is undefined (e.g. pos. 3 in the example). On the other hand, the abstract successor of a call symbol is its *matching return* if it exists, or undefined otherwise. The until modalities $\mathcal{U}^-$ and $\mathcal{U}^a$ are defined over the paths induced by the callers and abstract successors, respectively. Note that the caller path is always finite and the abstract path can be either finite or infinite. A prime application of CaRet is to state Hoare-like total correctness of a procedure F [3]:

$$\varphi_{total} \quad = \quad \Box^g\,(\,\mathsf{call} \wedge p \wedge p_F \;\rightarrow\; \bigcirc^a q\,)$$

where p and q are atomic propositions that hold at the states where the pre- and post-condition is satisfied, respectively, and p_F is an atomic proposition marking the calls to F. Another example is the language of repeatedly bounded words from Example 2; it is $\mathcal{L}_{repbdd} = \mathcal{L}(\Diamond^g\Box^g(\mathsf{call} \rightarrow \bigcirc^a\mathsf{ret}))$. Further examples are given in [3]. The language defined by a CaRet formula φ is denoted $\mathcal{L}(\varphi)$.

Theorem 2 ([1, Thm. 5.1]). *CaRet-definable languages are ω-VPL: For each CaRet formula φ there exists a (non-deterministic) Büchi VPA $\mathcal{A}$ such that $\mathcal{L}(\varphi) = \mathcal{L}(\mathcal{A})$, and $\mathcal{A}$ can be constructed in time $2^{\mathcal{O}(|\varphi|)}$.*

The above theorem is well-known in the literature [1,2] even though it is usually stated for *Nested Word Automata* (NWA) which are equivalent to VPA, and it is more common to state a space bound on $\mathcal{A}$ rather than a time bound for the construction. The theorem also applies to more expressive extensions of CaRet [1] which we do not consider here for the sake of simplicity.

3 Probabilistic Visibly Pushdown Automata

As explained in the introductory section, we employ *probabilistic pushdown automata* [14] (pPDA) as an operational model for procedural probabilistic programs. pPDA thus play a fundamentally different role in this paper than VPA (cf. Definition 1): While the former are used to model the *system*, the latter encode the *specification*. Consequently, our pPDA do *not* read an input word like VPA do, but instead take their transitions randomly, according to fixed probability distributions. In this way, they define a probability space over their possible traces, i.e., runs projected on their labeling sequence. These traces constitute the input words of the VPA. In order for the model checking problems to be decidable [13], a syntactic visibility restriction related—but not exactly analogous—to the one required by VPA needs to be imposed on pPDA. In a nutshell, the condition is that each state only has outgoing transitions of one *type*, i.e., push, internal, or pop. This means that the stack operation is *visible in the states* (recall that for VPA, the stack operation is visible in the input symbol). This restriction is not severe in the context of modeling programs (see Remark 2 further below) and leads to our notion of probabilistic *visibly* pushdown automata (pVPA) which we now define formally.

Given a finite set X, we write $\mathfrak{D}(X) = \{ f \colon X \to [0,1] \mid \sum_{a \in X} f(a) = 1 \}$ for the set of probability distributions on X.

Definition 6 (pVPA). *A* probabilistic visibly pushdown automaton *(pVPA) is a tuple $\Delta = (Q,\, q_0,\, \Gamma,\, \bot,\, P,\, \Sigma,\, \lambda)$ where Q is a finite set of* states *partitioned into $Q = Q_{\mathsf{call}} \uplus Q_{\mathsf{int}} \uplus Q_{\mathsf{ret}}$, $q_0 \in Q$ is an* initial state*, Γ is a finite* stack alphabet*, $\bot \in \Gamma$ is a special* bottom-of-stack symbol*, $P = (P_{\mathsf{call}}, P_{\mathsf{int}}, P_{\mathsf{ret}})$ is a triple of functions with signature*

$$P_{\mathsf{call}} \colon Q_{\mathsf{call}} \to \mathfrak{D}(Q \times \Gamma_{\text{-}\bot}) \ , \quad P_{\mathsf{int}} \colon Q_{\mathsf{int}} \to \mathfrak{D}(Q) \ , \quad P_{\mathsf{ret}} \colon Q_{\mathsf{ret}} \times \Gamma \to \mathfrak{D}(Q) \ ,$$

$\Sigma = \Sigma_{\mathsf{call}} \uplus \Sigma_{\mathsf{int}} \uplus \Sigma_{\mathsf{ret}}$ is a pushdown alphabet, and $\lambda \colon Q \to \Sigma$ is a state labeling function consistent with the visibility condition, i.e., for all $\mathsf{type} \in \{\mathsf{call}, \mathsf{int}, \mathsf{ret}\}$ *and all $q \in Q$, we have that $q \in Q_{\mathsf{type}}$ iff $\lambda(q) \in \Sigma_{\mathsf{type}}$.*

Intuitively, the behavior of a pVPA Δ is as follows. If the current state q is a call state, then the probability distribution $P_{\mathsf{call}}(q)$ determines a random successor state and stack symbol to be pushed on the stack ($\bot$ cannot be pushed). Similarly, if the current state is internal, then $P_{\mathsf{int}}(q)$ is the distribution over possible successor states and the stack is ignored completely. Lastly, if the current state is a return state and symbol $Z \in \Gamma$ is on top of the stack, then $P_{\mathsf{ret}}(q, Z)$ once again determines the probability distribution of successor states, and additionally Z is removed from the stack. Similar to VPA, the bottom symbol $\bot$ is the only exception to this rule, it can never be removed. Thus, pVPA are a generalization of labeled *Markov chains*, which correspond to the special case $Q = Q_{\mathsf{int}}$.

We now define the semantics of pVPA more formally. For $q, r \in Q$, $Z \in \Gamma$ and $p > 0$ we use the shorthand notations $q \xrightarrow{p} rZ$, $q \xrightarrow{p} r$, and $qZ \xrightarrow{p} r$ to indicate that $P_{\mathsf{call}}(q)(r, Z) = p$, $P_{\mathsf{int}}(q)(r) = p$, and $P_{\mathsf{ret}}(q, Z)(r) = p$, respectively. As for VPA, a *configuration* of a pVPA is an element $q\gamma \in Q \times \Gamma^*$. An (infinite) *run* of a pVPA is a sequence of configurations $\rho = q_0\gamma_0, q_1\gamma_1, \ldots$ such that $q_0\gamma_0 = q_0\bot$ and for all $i \geq 0$ we have that either

1. $q_i \in Q_{\mathsf{call}}$, $\gamma_{i+1} = \gamma_i Z$ for some $Z \in \Gamma_{\text{-}\bot}$ and $q_i \xrightarrow{p} q_{i+1}Z$;

2. $q_i \in Q_{\mathsf{int}}$, $\gamma_{i+1} = \gamma_i$ and $q_i \xrightarrow{p} q_{i+1}$; or
3. $q_i \in Q_{\mathsf{ret}}$, $\gamma_{i+1} Z = \gamma_i$ for some $Z \in \Gamma_{\text{-}\bot}$ and $q_i Z \xrightarrow{p} q_{i+1}$, or $\gamma_{i+1} = \gamma_i$ and $q_i \bot \xrightarrow{p} q_{i+1}$ (because the bottom symbol $\bot$ is never popped).

Note that our pVPA only produce infinite runs and do not simply "terminate" upon reaching the empty stack as in e.g. [14]. In fact, in our case the stack cannot be empty due to the special bottom symbol $\bot$ that can never be popped. We have chosen to avoid finite pVPA runs for compatibility with CaRet which describes ω-languages per definition. Nonetheless, terminating behavior can be easily simulated in our framework by moving to a dedicated sink state once the pVPA attempts to pop $\bot$ for the first time.

The set of all runs of a pVPA Δ is denoted $Runs_\Delta$. We extend Δ's labeling function λ to runs $\rho \in Runs_\Delta$ by applying it to each state along ρ individually, yielding a word $\lambda(\rho) \in \Sigma^\omega$. Steps of pVPA runs are defined as in Definition 3. An example pVPA and its possible runs are depicted in Figure 7 on page 14.

We can view the set of all configurations $Q \times \Gamma^*$ as the (infinite) state space of a discrete-time Markov chain. In this way, we obtain a probability space $(Runs_\Delta, \mathcal{F}, \mathbb{P})$ via the usual cylinder set construction [6, Ch. 10].

Remark 2. The visibility restriction of our pVPA is slightly different from the definition given in [13] which requires all *incoming* transitions to a state to be of the same type, i.e., call, internal, or return. Our definition, on the other hand, imposes the same requirement on the states' *outgoing* transitions. We believe that our condition is more natural for pVPA obtained from procedural programs, such as the one in Figure 1. In fact, programs where randomness is restricted to *internal* statements such as $x := \texttt{bernoulli}(0.5)$ or $x := \texttt{uniform}(0,3)$ naturally comply with our visibility condition because all call and return states of such programs are deterministic and thus cannot violate visibility. However, the alternative condition of [13] is not necessarily fulfilled for such programs.

We can now formally state our main problem of interest:

Definition 7 (Probabilistic CaRet Model Checking). *Let AP be a finite set of atomic propositions, φ be a CaRet formula over AP, Δ be a pVPA with labels from the pushdown alphabet $\Sigma = 2^{AP} \times \{\mathsf{call}, \mathsf{int}, \mathsf{ret}\}$, and $\theta \in [0,1] \cap \mathbb{Q}$. The* quantitative CaRet Model Checking problem *is to decide whether*

$$\mathbb{P}(\{\, \rho \in Runs_\Delta \mid \lambda(\rho) \in \mathcal{L}(\varphi) \,\}) \;\geq_?\; \theta \;.$$

The qualitative *CaRet Model Checking problem is the special case where $\theta = 1$.*

The probabilities in Definition 7 are well-defined as ω-VPL are measurable [22].

4 Model Checking against Stair-parity DVPA

In this section, we show that model checking pVPA (Definition 6) against VPL given in terms of a stair-parity DVPA (Definition 4) is decidable. This is achieved by first computing an automata-theoretic product of the pVPA and the DVPA and then evaluating the acceptance condition in the product automaton.

4.1 Products of Visibly Pushdown Automata

In general, pushdown automata are not closed under taking products as this would require *two* independent stacks. However, the visibility conditions on VPA and pVPA ensure that their product is again an automaton with just a single stack because the stack operations (push, nop, or pop) are forced to synchronize.

We now define the product formally. An *unlabeled* pVPA is a pVPA where the labeling function λ and alphabet Σ are omitted.

Definition 8 (Product $\Delta \times \mathcal{D}$). *Let $\Delta = (Q,\, q_0,\, \Gamma,\, \bot,\, P,\, \Sigma,\, \lambda)$ be a pVPA, and $\mathcal{D} = (S,\, s_0,\, \Gamma',\, \bot,\, \delta,\, \Sigma)$ be a DVPA over pushdown alphabet Σ. The product of Δ and $\mathcal{D}$ is the unlabeled pVPA*

$$\Delta \times \mathcal{D} \;=\; (\, Q{\times}S,\; (q_0, s_0),\; \Gamma{\times}\Gamma',\; \langle \bot, \bot \rangle,\; P_{\Delta\times\mathcal{D}}\,)\;,$$

where $P_{\Delta\times\mathcal{D}}$ is the smallest set of transitions satisfying the following rules for all $q, r \in Q$, $Z \in \Gamma$, $s, t \in S$, and $Y \in \Gamma'$:

$$\underset{\textbf{\textit{(call)}}}{\frac{q \xrightarrow{p}_{\Delta} rZ \;\wedge\; s \xrightarrow{\lambda(q)}_{\mathcal{D}} tY}{(q,s) \xrightarrow{p}_{\Delta\times\mathcal{D}} (r,t)\langle Z, Y\rangle}} \qquad \underset{\textbf{\textit{(internal)}}}{\frac{q \xrightarrow{p}_{\Delta} r \;\wedge\; s \xrightarrow{\lambda(q)}_{\mathcal{D}} t}{(q,s) \xrightarrow{p}_{\Delta\times\mathcal{D}} (r,t)}} \qquad \underset{\textbf{\textit{(return)}}}{\frac{qZ \xrightarrow{p}_{\Delta} r \;\wedge\; sY \xrightarrow{\lambda(q)}_{\mathcal{D}} t}{(q,s)\langle Z, Y\rangle \xrightarrow{p}_{\Delta\times\mathcal{D}} (r,t)}}\;.$$

If the DVPA $\mathcal{D}$ is equipped with a priority function $\Omega\colon S \to \mathbb{N}_0$, then we extend Ω to $\Omega'\colon Q \times S \to \mathbb{N}_0$ via $\Omega'(q,s) = \Omega(s)$.

It is not difficult to show that $\Delta\times\mathcal{D}$ is indeed a well-defined pVPA and moreover satisfies the following property (the proof is standard, see [28]):

Lemma 1 (Soundness of $\Delta \times \mathcal{D}$). *Let Δ be a pVPA and $\mathcal{D}$ be a stair-parity DVPA with priority function Ω, both over pushdown alphabet Σ. Then the product pVPA $\Delta \times \mathcal{D}$ with priority function Ω' as in Definition 8 satisfies*

$$\mathbb{P}(\{\, \rho \in \mathit{Runs}_{\Delta} \mid \lambda(\rho) \in \mathcal{L}(\mathcal{D}) \,\}) \;=\; \mathbb{P}(\{\, \rho \in \mathit{Runs}_{\Delta\times\mathcal{D}} \mid \rho{\downarrow}_{\mathit{Steps}} \in \mathit{Parity}_{\Omega'} \,\}),$$

where $\mathit{Parity}_{\Omega'}$ denotes the set of words in $(Q{\times}S)^{\omega}$ satisfying the standard *parity condition defined by Ω'. Moreover, $\Delta\times\mathcal{D}$ can be constructed in polynomial time.*

Remark 3. It is *not* actually important that the product satisfies the visibility condition. All techniques we apply to the product also work for general pPDA.

4.2 Stair-parity Acceptance Probabilities in pVPA

Lemma 1 effectively reduces model checking pVPA against stair-parity DVPA to computing stair-parity acceptance in the product, which is again an (unlabeled) pVPA. We therefore focus on pVPA in this section and do not consider DVPA.

Throughout the rest of this section, let $\Delta = (\,Q,\, q_0,\, \Gamma,\, \bot,\, P\,)$ be an unlabeled pVPA. On the next pages we describe the construction of a finite Markov chain $\mathcal{M}_{\Delta}$ that we call the *step chain* of Δ. Loosely speaking, $\mathcal{M}_{\Delta}$ simulates jumping from one *step* (see Definition 3) of a run of Δ to the next. A similar idea first appeared in [14]. Our construction, however, differs from the original one in various aspects. We discuss this in detail in Remark 5 further below.

Steps as events. For all $n \in \mathbb{N}_0$, we define a *random* variable $V^{(n)}$ on Runs_Δ whose value is either the state q of Δ at the n-th step, or the *extended state* $q\bot$ in the special case where the n-th step occurs at a bottom configuration of the form $q\bot$, for some $q \in Q$. We denote the set of all such extended states with $Q\bot = \{\, q\bot \mid q \in Q \,\}$. Formally, $V^{(n)} \colon \mathit{Runs}_\Delta \to Q \cup Q\bot$ is defined as

$$V^{(n)}(\rho) \quad = \quad \begin{cases} q & \text{if } \mathit{step}_n(\rho) = q\gamma \text{ and } \gamma \neq \bot \\ q\bot & \text{if } \mathit{step}_n(\rho) = q\bot \ , \end{cases}$$

where $\mathit{step}_n(\rho)$ denotes the configuration at the n-th step of ρ. Note that $V^{(0)} = q_0\bot$ because the first position of a run is always a step.

Lemma 2. *For all $n \in \mathbb{N}_0$ and $v \in Q \cup Q\bot$, the event $V^{(n)} = v$ is measurable, and thus $V^{(n)}$ is a well-defined random variable.*

We can view the sequence $V^{(0)}, V^{(1)} \ldots$ of random variables as a *stochastic process*. It is intuitively clear that for all $n \in \mathbb{N}_0$, the value of $V^{(n+1)}$ depends only on $V^{(n)}$, but not on $V^{(i)}$ for $i < n$. This is due to the more general observation that the state q at any step configuration $q\gamma$ (with $\gamma \neq \bot$) *fully determines the future of the run* because being a step already implies that no symbol in γ can ever be read as reading it implies popping it from the stack. In particular, q determines the probability distribution over possible next steps. A similar observation applies to bottom configurations of the form $q\bot$. Phrased in probability theoretical terms, the process $V^{(0)}, V^{(1)} \ldots$ has the *Markov property*, i.e.,

$$\mathbb{P}(V^{(n)}{=}v_n \mid V^{(n-1)}{=}v_{n-1} \wedge \ldots \wedge V^{(0)}{=}v_0) \;=\; \mathbb{P}(V^{(n)}{=}v_n \mid V^{(n-1)}{=}v_{n-1}) \quad (1)$$

holds for all values of $v_0, \ldots, v_n$ such that the above conditional probabilities are well-defined [1]. This was proved in detail in [14]. It is also clear that the Markov process is time-homogeneous in the sense that

$$\mathbb{P}(V^{(n+1)} = v \mid V^{(n)} = v') \quad = \quad \mathbb{P}(V^{(n'+1)} = v \mid V^{(n')} = v')$$

holds for all $n, n' \in \mathbb{N}_0$ for which the two conditional probabilities are well-defined. The following example provides some intuition on these facts.

Example 3. Consider the pVPA in Figure 7 (left). The initial fragments of its two equiprobable runs are depicted in the middle. In this example, it is easy to read off the next-step probabilities $\mathbb{P}(V^{(n)} = v_n \mid V^{(n-1)} = v_{n-1})$ for all $n \in \mathbb{N}_0$ and $v_n, v_{n-1} \in Q \cup Q\bot$. They are summarized in the Markov chain on the right. For example, $V^{(0)} = q_0\bot$ holds with probability 1, and $V^{(1)} = q_1$ and $V^{(1)} = q_3\bot$ hold with probability $1/2$ each because the second step occurs either at position 1 with configuration $q_1\bot Z$ or at position 3 with configuration $q_3\bot$,

[1] A conditional probability is well-defined if the *condition*, i.e., the event on the right hand side of the vertical bar, has positive probability. Expressions like the one in (1) are thus not necessarily well-defined because the probability that $V^{(n-1)} = v_{n-1}$ might be zero for certain values of n and v_{n-1}.

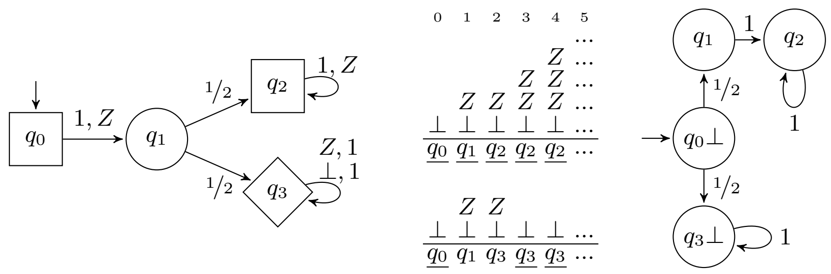


Fig. 7. Left: An example (unlabeled) pVPA Δ. Call, internal, and return states are depicted as squares, circles, and rhombs, respectively. The format of the transition labels is analogous to Figure 5 (left). Middle: Initial fragments of the two possible runs of Δ. Steps are underlined. Right: Its step Markov chain $\mathcal{M}_\Delta$ (Definition 10, page 15).

	$q \to r$	$q\bot \to r$	$q\bot \to r\bot$	$q \to r\bot$
$q \in Q_{\mathsf{call}}$	$\frac{[r\uparrow]}{[q\uparrow]}\Big(\sum_{r',Z} P_{\mathsf{call}}(q, r'Z)[r'Z{\downarrow}r] + \sum_{Z} P_{\mathsf{call}}(q, rZ)\Big)$	$\sum_{Z} P_{\mathsf{call}}(q, rZ)[r\uparrow]$	$\sum_{r',Z} P_{\mathsf{call}}(q, r'Z)[r'Z{\downarrow}r]$	0
$q \in Q_{\mathsf{int}}$	$\frac{[r\uparrow]}{[q\uparrow]} P_{\mathsf{int}}(q, r)$	0	$P_{\mathsf{int}}(q, r)$	0
$q \in Q_{\mathsf{ret}}$	n/a	0	$P_{\mathsf{ret}}(q\bot, r)$	n/a

Fig. 8. Next-step probabilities of the step Markov chain. P_{type} for $\mathsf{type} \in \{\, \mathsf{call}, \mathsf{int}, \mathsf{ret} \,\}$ are the probabilities of the pVPA's call, internal, and return transitions, respectively. The values $[r'Z{\downarrow}r]$ and $[q\uparrow]$ are the return and diverge probabilities from Definition 9.

and both options are equally likely. The case $\mathbb{P}(V^{(2)} = q_2 \mid V^{(1)} = q_1) = 1$ is slightly more interesting: Given that a configuration $q_1\gamma$ with $\gamma \neq \bot$ is a step, we know that the next state must be q_2 (which is then also a step). Even though there is a transition from q_1 to q_3 in Δ, the next state cannot be q_3 because the latter is a return state which would immediately decrease the stack height of γ. This shows that, intuitively speaking, *conditioning on being a step influences the probabilities of a state's outgoing transitions.*

Probabilities of next steps, returns, and diverges. Our next goal is to provide expressions for the next-step probabilities $\mathbb{P}(V^{(n+1)} = v' \mid V^{(n)} = v)$ as we did in Example 3. It turns out that those can be stated in terms of the *return* and *diverge* probabilities of Δ.

Definition 9. *Let $p, q \in Q$, $Z \in \Gamma$, and $\gamma \in \Gamma^*$. We define*

- *the* return *probability $[pZ{\downarrow}q]$ as the probability to reach configuration $q\gamma$ from $p\gamma Z$ without visiting another configuration of the form $r\gamma$ for some $r \in Q$ in between; and*
- *the* diverge *probability $[p\uparrow]$ as the probability to never decrease the stack height below $|\gamma Z|$ when starting in $p\gamma Z$, i.e., $[p\uparrow] = 1 - \sum_{q \in Q}[pZ{\downarrow}q]$.*

Note that $[p{\uparrow}]$ is indeed independent of Z because the only way to read Z is by popping it from the stack which decreases the stack height. The diverge probabilities are closely related to steps. Indeed, the probability that a configuration $p\gamma$ with $\gamma \neq \bot$ is a step is equal to $[p{\uparrow}]$. For example, in the pVPA in Figure 7 the configuration $q_1 \bot Z$ is a step with probability $[q_1{\uparrow}] = 1/2$.

It is known that the return and diverge probabilities are in general *non-rational*. As a minimal example, consider a pVPA that repeats the following steps until emptying its stack or getting stuck: (i) It pushes four symbols with probability 1/6, or (ii) pops one symbol with probability 1/2, or (iii) gets stuck otherwise. The resulting return probability is the least solution of $x = (1/6)x^5 + 1/2$, a non-rational number that is not even solvable by radicals [16, Thm. 3.2(1)].

Remark 4. The terms *return* and *diverge* are natural. When modeling procedural probabilistic programs as pVPA, $[pZ{\downarrow}q]$ is just the probability to eventually *return* from local state p of the current procedure to local state q of the calling procedure (the return address is stored on the stack in Z). Similarly, $[p{\uparrow}]$ is the probability that the current procedure diverges, i.e., it never returns to the calling context. Clearly, this is independent of the return address.

Lemma 3. *The conditional next-step probabilities in Figure 8 are correct in the sense that if* $\mathbb{P}(V^{(n+1)} = v' \mid V^{(n)} = v)$ *is defined for* $n \in \mathbb{N}_0$ *and* $v, v' \in Q \cup Q\bot$ *then it is equal to the probability in the respective column "*$v \to v'$*".*

Proof sketch. We only provide some intuition for two important cases; formal derivations are in [28]. Let $r \in Q$ be arbitrary.

- If $q \in Q_{\mathsf{int}}$ then $\mathbb{P}(V^{(n+1)} = r \mid V^{(n)} = q) = P_{\mathsf{int}}(q,r)[r{\uparrow}]/[q{\uparrow}]$: Suppose that the n-th step takes place at position i of the run. Since the n-th step occurs at an *internal* state q, the n+1-st step must necessarily occur immediately at position i+1. The factor $P(q,r)[r{\uparrow}]$ is proportional to the probability to take an (internal) transition from q to r and then diverge in r, which is necessary in order for the next configuration to be a step. However, the values $\{ P(q,r)[r{\uparrow}] \mid r \in Q \}$ do not form a probability distribution in general. This justifies the division by the normalizing constant $[q{\uparrow}] = \sum_{r \in Q} P(q,r)[r{\uparrow}]$.
- If $q \in Q_{\mathsf{call}}$ then $\mathbb{P}(V^{(n+1)} = r\bot \mid V^{(n)} = q\bot) = \sum_{r',Z} P_{\mathsf{call}}(q, r'Z)[r'Z{\downarrow}r]$: If the n-th step occurs at bottom configuration $q\bot$, then the n+1-st step can only occur at *bottom* configuration $r\bot$ if the symbols pushed by q's outgoing transitions are eventually popped. The expression in the sum equals the probability to take a push-transition from q to r' that pushes Z onto the stack multiplied by the probability to *return* from r' (with Z on top) to r.

The step chain. It is convenient to view the stochastic process $V^{(0)}, V^{(1)} \ldots$ as an explicit (graphical) Markov chain.

Definition 10 (The Step Chain $\mathcal{M}_\Delta$). *$\mathcal{M}_\Delta$ is the Markov chain with states*

$$M \quad = \quad \{ q \in Q_{\mathsf{call}} \cup Q_{\mathsf{int}} \mid [q{\uparrow}] > 0 \} \ \cup \ Q\bot \ ,$$

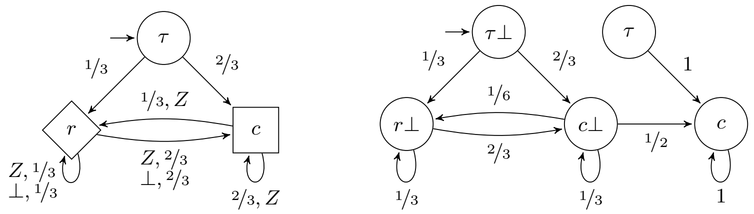


Fig. 9. Left: Example pVPA with the following return-diverge probabilities: $[cZ{\downarrow}c] = {}^1\!/_6$, $[cZ{\downarrow}r] = {}^1\!/_{12}$, $[rZ{\downarrow}r] = {}^1\!/_3$, $[rZ{\downarrow}c] = {}^2\!/_3$, and $[c{\uparrow}] = {}^3\!/_4$, $[\tau{\uparrow}] = {}^1\!/_2$, $[r{\uparrow}] = 0$. Even though it is the case here, these probabilities are not always rational [16]. Right: Its step Markov chain according to Definition 10. The transition probabilities can be computed using the return and diverge probabilities and Figure 8.

initial state $q_0\bot$, and for all $v, v' \in M$, the probability of transition $v \to v'$ is defined according to Figure 8.

Figure 9 depicts a non-trivial pVPA and its step chain. In this example, all return and diverge probabilities are rational. In general, however, the return and diverge probabilities (Definition 9) are algebraic numbers that are not always rational or even expressible by radicals [16]. As a consequence, one cannot easily perform numerical computations on the step chain. However, the probabilities can be encoded implicitly as the unique solution of an *existential theory of the reals* (ETR) formula, i.e. an existentially quantified FO-formula over $(\mathbb{R}, +, \cdot, \leq)$ [14]. Since the ETR is decidable, many questions about the step chain are still decidable as well. We will make use of this in Theorem 3 below.

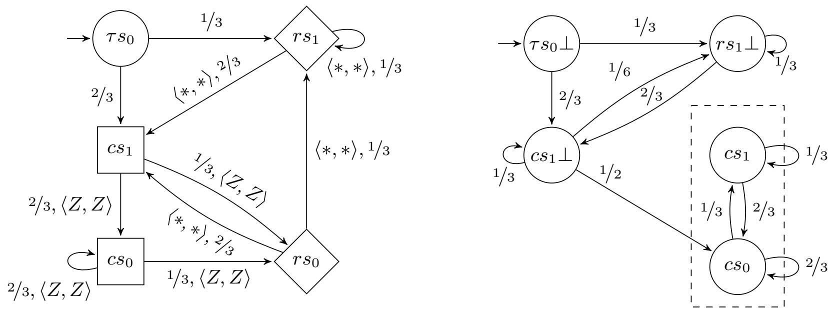
The property of $\mathcal{M}_\Delta$ that is most relevant to us is given by the following Lemma 4. We call $\rho{\Downarrow}_{Steps} = V^{(0)}(\rho)V^{(1)}(\rho)\dots$ the *extended footprint* of run ρ.

Lemma 4 (Soundness of $\mathcal{M}_\Delta$). *Let Δ be a pVPA with step chain $\mathcal{M}_\Delta$. Let M be the states of the step chain and consider a measurable set $R \subseteq M^\omega$. Then*

$$\mathbb{P}(\{\, \rho \in Runs_\Delta \mid \rho{\Downarrow}_{Steps} \in R \,\}) \quad = \quad \mathbb{P}(R) \ .$$

Proof sketch. For basic cylinder sets of the form $R = w \cdot M^\omega$ for some $w \in M^*$, the claim follows from the Markov property (1) together with the correctness of the transition probabilities of $\mathcal{M}_\Delta$ according to Lemma 3. For other measurable sets, it can be shown by induction over the levels of the Borel hierarchy [28].

Remark 5. The step chain as presented here differs from the original definition in [14] in at least two important aspects. First, we have to take the semantics of our special bottom symbol $\bot$ into account. This is why our chain uses a subset of $Q \cup Q\bot$ as states—it must distinguish whether a step occurs at a bottom configuration. The pPDA in [14], on the other hand, may have both finite and infinite runs, and this needs to be handled differently in the step chain. Second, we use step chains for a different purpose than [14], namely to show that general measurable properties defined on steps—this includes stair-parity—can be evaluated on pVPA (Lemma 4).



τs_0
rs_1
cs_1
cs_0
rs_0
$1/3$
$2/3$
$\langle *, * \rangle, 2/3$
$\langle *, * \rangle, 1/3$
$1/3, \langle Z, Z \rangle$
$\langle *, * \rangle, 2/3$
$2/3, \langle Z, Z \rangle$
$2/3, \langle Z, Z \rangle$
$1/3, \langle Z, Z \rangle$
$\tau s_0 \bot$
$rs_1 \bot$
$cs_1 \bot$
cs_1
cs_0
$1/3$
$1/6$
$2/3$
$1/2$

Fig. 10. Left: The product of the pVPA from Figure 9 (left) and the DVPA from Figure 5 (left) on page 7. Right: Its step chain according to Definition 10. The dashed region is the only BSCC. It violates the parity condition $\Omega(s_0) = 1$ and $\Omega(s_1) = 2$ inherited from the DVPA (see Example 2 on page 8) since every run reaching the BSCC visits cs_0 infinitely often with probability 1. Only reachable states are depicted.

Putting it all together. We can now prove the main result of this section.

Theorem 3. *Let Δ be a pVPA and let $\mathcal{D}$ be a stair-parity DVPA, both over the same pushdown alphabet Σ. Then for all $\theta \in [0,1] \cap \mathbb{Q}$, the problem $\mathbb{P}(\{\rho \in Runs_\Delta \mid \lambda(\rho) \in \mathcal{L}(\mathcal{D})\}) \geq_? \theta$ is decidable in* PSPACE.

Proof sketch. We first construct the product $\Delta \times \mathcal{D}$ according to Definition 8. By Lemma 1 we need to compute the stair-parity acceptance probability of $\Delta \times \mathcal{D}$. Lemma 4 reduces this to computing a usual parity acceptance probability in the step chain $\mathcal{M}_{\Delta\times\mathcal{D}}$. This can be achieved through finding the bottom strongly connected components (BSCC) of $\mathcal{M}_{\Delta\times\mathcal{D}}$, classifying them as *good* (the minimum priority of a BSCC state is even) or otherwise *bad*, and running a standard reachability analysis wrt. the good states. See Figure 10 for an example. The remaining technical difficulty is that the transition probabilities of $\mathcal{M}_{\Delta\times\mathcal{D}}$ are not rational in general. However, this can be dealt with using the fact that these probabilities are expressible in the ETR [14] (see [28] for the details).

4.3 Probabilistic One-counter Automata

A probabilistic visibly *one-counter automaton* (pVOC) is the special case of a pVPA with unary stack alphabet, i.e., $|\Gamma_{\text{-}\bot}| = 1$. For example, the pVPA in Figure 9 (left) is a pVOC. For many problems, better complexity bounds are known for pVOC than for the general case. In particular, $[p\uparrow] > 0$ can be decided in P [9, Thm. 4]. We can exploit this to improve Theorem 3 in the pVOC case:

Corollary 1. *Let Δ be a pVOC and $\mathcal{D}$ be a stair-parity DVPA over pushdown alphabet Σ. The problem $\mathbb{P}(\{\rho \in Runs_\Delta \mid \lambda(\rho) \in \mathcal{L}(\mathcal{D})\}) =_? 1$ is decidable in* P.

Corollary 1 implies that there exist efficient algorithms for many properties of pVOC-expressible random walks on $\mathbb{N}_0$. In fact, a.-s. satisfaction of each *fixed* visibly-pushdown property can be decided in P. For instance, using the DVPA from Figure 5 we can decide if a random walk is a.-s. repeatedly bounded.

5 Model Checking against Büchi VPA and CaRet

With Theorems 1 and 3 it follows immediately that quantitative model checking of pVPA against non-deterministic Büchi VPA is decidable in $\mathsf{EXPSPACE}$. We can improve the complexity in the qualitative case:

Theorem 4. *Let Δ be a pVPA and $\mathcal{A}$ be a (non-deterministic) Büchi VPA over the same pushdown alphabet. The problem $\mathbb{P}(\{\rho \in Runs_\Delta \mid \lambda(\rho) \in \mathcal{L}(\mathcal{A})\}) =_? 1$ is $\mathsf{EXPTIME}$-complete.*

In the above result, membership in $\mathsf{EXPTIME}$ relies on the fact that one can construct the underlying *graph* of a step chain $\mathcal{M}_{\Delta\times\mathcal{D}}$ in time exponential in the size of Δ but *polynomial* in the size of $\mathcal{D}$; see [28]. $\mathsf{EXPTIME}$-hardness follows from [15, Thm. 8]. In fact, qualitative model checking of pPDA against *non-pushdown* Büchi automata is also $\mathsf{EXPTIME}$-complete [15]. With Theorems 1 to 4 we immediately obtain the following complexity results for CaRet model checking:

Theorem 5. *The quantitative and qualitative probabilistic CaRet model checking problems (Def. 7) are decidable in $\mathsf{2EXPSPACE}$ and $\mathsf{2EXPTIME}$, respectively.*

Both problems are known to be $\mathsf{EXPTIME}$-hard [30].

6 Conclusion

We have presented the first decidability result for model checking pPDA—an operational model of procedural discrete probabilistic programs—against CaRet, or more generally, against the class of ω-VPL. We heavily rely on the determinization procedure from [22] and the notion of a step chain used in previous works. These two constructions turn our to be natural match.

We conjecture that our complexity bounds are not the best possible which is often the case in purely automata-based model checking. Future work is thus to investigate whether the doubly-exponential complexity can be lowered to singly-exponential, e.g. by generalizing the automata-less algorithm from [30]. Other topics are to explore to what extent algorithms for probabilistic CTL can be generalized to the branching-time variant of CaReT [18], to consider more expressive logics such as visibly LTL [8] or OPTL [12], and to study the interplay of *conditioning* and recursion [27] through the lens of pPDA.

Acknowledgement. The authors thank Christof Löding for his pointer to stair-parity VPA, and the anonymous reviewers for their constructive feedback.

References

1. Alur, R., Arenas, M., Barceló, P., Etessami, K., Immerman, N., Libkin, L.: First-Order and Temporal Logics for Nested Words. In: 22nd IEEE Symposium on Logic in Computer Science (LICS 2007), 10-12 July 2007, Wroclaw, Poland, Proceedings. pp. 151–160. IEEE Computer Society (2007). https://doi.org/10.1109/LICS.2007.19
2. Alur, R., Bouajjani, A., Esparza, J.: Model Checking Procedural Programs. In: Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R. (eds.) Handbook of Model Checking, pp. 541–572. Springer (2018). https://doi.org/10.1007/978-3-319-10575-8_17
3. Alur, R., Etessami, K., Madhusudan, P.: A Temporal Logic of Nested Calls and Returns. In: Tools and Algorithms for the Construction and Analysis of Systems, 10th International Conference, TACAS 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, March 29 - April 2, 2004, Proceedings. Lecture Notes in Computer Science, vol. 2988, pp. 467–481. Springer (2004). https://doi.org/10.1007/978-3-540-24730-2_35
4. Alur, R., Madhusudan, P.: Visibly Pushdown Languages. In: Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004. pp. 202–211. ACM (2004). https://doi.org/10.1145/1007352.1007390
5. Audebaud, P., Paulin-Mohring, C.: Proofs of randomized algorithms in Coq. Sci. Comput. Program. **74**(8), 568–589 (2009). https://doi.org/10.1016/j.scico.2007.09.002
6. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press (2008)
7. Barthe, G., Köpf, B., Olmedo, F., Béguelin, S.Z.: Probabilistic Relational Reasoning for Differential Privacy. ACM Trans. Program. Lang. Syst. **35**(3), 9:1–9:49 (2013). https://doi.org/10.1145/2492061
8. Bozzelli, L., Sánchez, C.: Visibly Linear Temporal Logic. J. Autom. Reason. **60**(2), 177–220 (2018). https://doi.org/10.1007/s10817-017-9410-z
9. Brázdil, T., Esparza, J., Kiefer, S., Kucera, A.: Analyzing probabilistic pushdown automata. Formal Methods Syst. Des. **43**(2), 124–163 (2013). https://doi.org/10.1007/s10703-012-0166-0
10. Brázdil, T., Kucera, A., Strazovský, O.: On the Decidability of Temporal Properties of Probabilistic Pushdown Automata. In: STACS 2005, 22nd Annual Symposium on Theoretical Aspects of Computer Science, Stuttgart, Germany, February 24-26, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3404, pp. 145–157. Springer (2005). https://doi.org/10.1007/978-3-540-31856-9_12
11. Casini, L., Illari, P.M., Russo, F., Williamson, J.: Recursive Bayesian Networks. Theoria. Revista de Teoria, Historia y Fundamentos de la Ciencia **26**(1), 5–33 (2008)
12. Chiari, M., Mandrioli, D., Pradella, M.: Operator precedence temporal logic and model checking. Theor. Comput. Sci. **848**, 47–81 (2020). https://doi.org/10.1016/j.tcs.2020.08.034
13. Dubslaff, C., Baier, C., Berg, M.: Model checking probabilistic systems against pushdown specifications. Inf. Process. Lett. **112**(8-9), 320–328 (2012). https://doi.org/10.1016/j.ipl.2012.01.006
14. Esparza, J., Kucera, A., Mayr, R.: Model Checking Probabilistic Pushdown Automata. In: 19th IEEE Symposium on Logic in Computer Science (LICS 2004), 14-17 July 2004, Turku, Finland, Proceedings. pp. 12–21. IEEE Computer Society (2004). https://doi.org/10.1109/LICS.2004.1319596

15. Etessami, K., Yannakakis, M.: Algorithmic Verification of Recursive Probabilistic State Machines. In: Tools and Algorithms for the Construction and Analysis of Systems, 11th International Conference, TACAS 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3440, pp. 253–270. Springer (2005). https://doi.org/10.1007/978-3-540-31980-1_17
16. Etessami, K., Yannakakis, M.: Recursive markov chains, stochastic grammars, and monotone systems of nonlinear equations. J. ACM **56**(1), 1:1–1:66 (2009). https://doi.org/10.1145/1462153.1462154
17. Gordon, A.D., Henzinger, T.A., Nori, A.V., Rajamani, S.K.: Probabilistic programming. In: Proceedings of the on Future of Software Engineering, FOSE 2014, Hyderabad, India, May 31 - June 7, 2014. pp. 167–181. ACM (2014). https://doi.org/10.1145/2593882.2593900
18. Gutsfeld, J.O., Müller-Olm, M., Nordhoff, B.: A Branching Time Variant of CaRet. In: Model Checking Software - 25th International Symposium, SPIN 2018, Malaga, Spain, June 20-22, 2018, Proceedings. Lecture Notes in Computer Science, vol. 10869, pp. 153–170. Springer (2018). https://doi.org/10.1007/978-3-319-94111-0_9
19. Jaeger, M.: Complex Probabilistic Modeling with Recursive Relational Bayesian Networks. Ann. Math. Artif. Intell. **32**(1-4), 179–220 (2001). https://doi.org/10.1023/A:1016713501153
20. Jones, C.: Probabilistic non-determinism. Ph.D. thesis, University of Edinburgh, UK (1990), http://hdl.handle.net/1842/413
21. Kucera, A., Esparza, J., Mayr, R.: Model Checking Probabilistic Pushdown Automata. Log. Methods Comput. Sci. **2**(1) (2006). https://doi.org/10.2168/LMCS-2(1:2)2006
22. Löding, C., Madhusudan, P., Serre, O.: Visibly Pushdown Games. In: FSTTCS 2004: Foundations of Software Technology and Theoretical Computer Science, 24th International Conference, Chennai, India, December 16-18, 2004, Proceedings. Lecture Notes in Computer Science, vol. 3328, pp. 408–420. Springer (2004). https://doi.org/10.1007/978-3-540-30538-5_34
23. McIver, A., Morgan, C.: Partial correctness for probabilistic demonic programs. Theor. Comput. Sci. **266**(1-2), 513–541 (2001). https://doi.org/10.1016/S0304-3975(00)00208-5
24. van de Meent, J., Paige, B., Yang, H., Wood, F.: An Introduction to Probabilistic Programming. CoRR **abs/1809.10756** (2018), http://arxiv.org/abs/1809.10756
25. Olmedo, F., Kaminski, B.L., Katoen, J., Matheja, C.: Reasoning about Recursive Probabilistic Programs. In: Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016. pp. 672–681. ACM (2016). https://doi.org/10.1145/2933575.2935317
26. Pfeffer, A., Koller, D.: Semantics and Inference for Recursive Probability Models. In: Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence, July 30 - August 3, 2000, Austin, Texas, USA. pp. 538–544. AAAI Press / The MIT Press (2000), http://www.aaai.org/Library/AAAI/2000/aaai00-082.php
27. Stuhlmüller, A., Goodman, N.D.: A Dynamic Programming Algorithm for Inference in Recursive Probabilistic Programs. CoRR **abs/1206.3555** (2012), http://arxiv.org/abs/1206.3555
28. Winkler, T., Gehnen, C., Katoen, J.: Model Checking Temporal Properties of Recursive Probabilistic Programs. CoRR **abs/2111.03501** (2021), https://arxiv.org/abs/2111.03501

29. Wojtczak, D., Etessami, K.: PReMo : An Analyzer for Probabilistic Recursive Models. In: Tools and Algorithms for the Construction and Analysis of Systems, 13th International Conference, TACAS 2007, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2007 Braga, Portugal, March 24 - April 1, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4424, pp. 66–71. Springer (2007). https://doi.org/10.1007/978-3-540-71209-1_7
30. Yannakakis, M., Etessami, K.: Checking LTL Properties of Recursive Markov Chains. In: Second International Conference on the Quantitative Evaluaiton of Systems (QEST 2005), 19-22 September 2005, Torino, Italy. pp. 155–165. IEEE Computer Society (2005). https://doi.org/10.1109/QEST.2005.8