MODEL CHECKING TEMPORAL PROPERTIES OF RECURSIVE PROBABILISTIC PROGRAMS

TOBIAS WINKLER ☢, CHRISTINA GEHNEN ☢, AND JOOST-PIETER KATOEN ☢

RWTH Aachen University, Aachen, Germany
e-mail address: tobias.winkler@cs.rwth-aachen.de, christina.gehnen@rwth-aachen.de, katoen@cs.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 for model checking pPDA against temporal properties 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. These logical formulae allow specifying 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 such as total and partial correctness.

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 [BKOB13] 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 [GHNR14]. Probabilistic programs have many applications [vdMPYW18]. 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 the declaration of procedures—small independent code blocks—and the ability to call procedures from one another, possibly in 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 [PK00, Jae01, CIRW08, SG12]. 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 (pPDA) [EKM04] 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.

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 [ABE18]). 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

```plaintext
proc void infectYoung() :
y := uniform(0,3)
repeat y times :
infectYoung()
e := uniform(0,2)
repeat e times :
f := infectElder()
return

proc bool infectElder() :
y := uniform(0,1)
repeat y times :
infectYoung()
e := uniform(0,4)
repeat e times :
f := bernoulli(0.01)
return f
```

Figure 1: Recursive probabilistic program modeling the outbreak of an infectious disease.

uniform(a, b) stands for the discrete uniform distribution on [a, b].

| Y | E |
|---|---|
| Y | 1.5 | 1 |
| E | 0.5 | 2 |

Figure 2: Example infection rates by age groups.
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) are decidable for finite pPDA (see [BEKK13] 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 $f = true$ 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 naturally expressed in the logic CaRet [AEM04], an extension of LTL:

$$\Box^g (\Box p_Y \land p_Y \land \Diamond^a f).$$

Here, $p_Y$ is an atomic proposition that holds at states which correspond to being in procedure infectYoung, and $f$ indicates that $f = true$. 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 ($\Box p_Y \land p_Y$), 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 [AEM04].

**Technical Contribution.** The contribution of this paper is a solution to the following problem: We are given a (finite) pPDA $\Delta$ and a CaRet formula $\varphi$ and we are interested in determining the probability that a random trajectory of $\Delta$ satisfies $\varphi$.

We briefly outline our solution. In order for the problem to be decidable [DBB12], we need to impose a mild visibility restriction on $\Delta$, yielding a probabilistic visibly pushdown automaton (pVPA). Just like several previous works on model checking pPDA against $\omega$-regular specifications [EKM04, BKS05, KEM06], we follow the automata-based approach (see Figure 3). More specifically, we first translate $\varphi$ into an equivalent non-deterministic Büchi visibly pushdown automaton (VPA) [AM04] $A$ and then determinize it using a procedure introduced by Löding et al. [LMS04]. The resulting DVPA $D$ uses a so-called stair-parity [LMS04] acceptance condition that is strictly more expressive than standard parity or Muller DVPA [AM04]. Stair-parity differs from usual parity in that it only considers certain positions—called steps [LMS04]—of an infinite word where the stack height never decreases again. We then construct a standard product $\Delta \times D$. 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 [DBB12]. 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 $\Delta \times D$, 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 was introduced by Esparza el al. [EKM04] who used them to show decidability of the model checking problem against deterministic (non-pushdown) Büchi automata. For constructing the step chain, certain reachability probabilities of the pPDA need to be computed. In general, these probabilities are algebraic numbers that cannot always be expressed by radicals [EY09], 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) [KEM06].

The resulting main contributions of this paper are complexity results, summarized in Figure 4, and algorithms for quantitative model checking of pPDA against ω-visibly pushdown languages (VPL) given in terms of either deterministic or non-deterministic visibly pushdown 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 [DBB12] proved decidability of model checking against deterministic Muller VPA which capture a strict subset of the CaRet-definable languages [AM04]. 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 [EY09]. 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 [WE07]. 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 probabilistic programs. Proof rules for recursion were first provided in [Jon90], and later extended to proof rules in a weakest-precondition reasoning style [MM01, OKKM16]. Olmedo et al. [OKKM16] 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 [AP09]. The Coq approach is based on higher-order logic using a monadic interpretation of programs as probabilistic distributions.
Figure 4: Complexity results of this paper.

Conference version. A preliminary version of this article was published at FoSSaCS 2022 [WGK22]. The present journal version extends the conference paper by the full proofs as well as further examples and explanations.

Organization. This paper is structured as follows. 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

In this section, we summarize some preliminary results on visibly pushdown languages and their corresponding automata models, and we recall the syntax and semantics of CaRet.

We use the following notation for words. Given a non-empty alphabet \( \Sigma \), let \( \Sigma^* \) be the set of all finite words (including the empty word \( \epsilon \)), and let \( \Sigma^\omega \) be the set of all infinite words over \( \Sigma \). 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 \( w \). For sets of finite words \( A \) and \( B \), the concatenation of all words from \( A \) with those from \( B \) is denoted \( A.B \). We also use \( a.B \) and \( A.b \) as shorthands for \( \{a\}.B \) and \( A.\{b\} \), respectively.

2.1. Visibly Pushdown Automata. A finite alphabet \( \Sigma \) is called pushdown alphabet if it is equipped with a partition \( \Sigma = \Sigma_{\text{call}} \uplus \Sigma_{\text{int}} \uplus \Sigma_{\text{ret}} \) into three—possibly empty—subsets of call, internal, and return symbols. A visibly pushdown automaton (VPA) [AM04] over \( \Sigma \) is like a standard pushdown automaton with the additional syntactic restriction that reading a call, internal, or return symbol triggers a push, internal, or pop transition, respectively (an internal transition is one that does not change the stack height). VPA are defined more formally as follows.

**Definition 2.1 (VPA [AM04]).** Let \( \Sigma \) be a pushdown alphabet. A visibly pushdown automaton (VPA) over \( \Sigma \) is a tuple \( A = (S, s_0, \Gamma, \bot, \delta, \Sigma) \) with \( S \) a finite set of states, \( s_0 \in S \) an initial state, \( \Gamma \) a finite stack alphabet, \( \bot \in \Gamma \) a special bottom-of-stack symbol, and \( \delta = (\delta_{\text{call}}, \delta_{\text{int}}, \delta_{\text{ret}}) \) a triple of relations

\[
\delta_{\text{call}} \subseteq (S \times \Sigma_{\text{call}}) \times (S \times \Gamma \setminus \{\bot\}), \quad \delta_{\text{int}} \subseteq (S \times \Sigma_{\text{int}}) \times S, \quad \delta_{\text{ret}} \subseteq (S \times \Sigma_{\text{ret}} \times \Gamma) \times S
\]

where \( \Gamma \setminus \{\bot\} = \Gamma \setminus \{\bot\} \).

For \( s, t \in S, Z \in \Gamma, \) and \( a \in \Sigma \), we use the shorthand notations \( s \xrightarrow{\alpha} tZ, s \xrightarrow{\alpha} t, sZ \xrightarrow{\alpha} t \) to indicate that there exist transitions \( (s, a, t, Z) \in \delta_{\text{call}}, (s, a, t) \in \delta_{\text{int}}, (s, a, Z, t) \in \delta_{\text{ret}} \), respectively.
The semantics of a VPA is defined as usual via configurations and runs. A configuration of VPA $\mathcal{A}$ is a tuple $(s, \gamma) \in S \times \Gamma_s \times \perp$, written more succinctly as $s\gamma$ in the sequel. Intuitively, being in configuration $s\gamma$ means that the automaton is currently in state $s$ and has the word $\gamma$ on its stack. If $s\gamma = sZ\gamma'$ for some $Z \in \Gamma$, then $sZ$ is called the head of $s\gamma$.

A bottom configuration is a configuration with head $\perp$ for some $s \in S$. 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 $s_0\gamma_0 = s_0\perp$ and for all $i \geq 0$, exactly one of the following cases applies:

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

In other words, if $\mathcal{A}$ reads a call (or internal) symbol $a$ while being in configuration $s\gamma$ and there exists a suitable call transition $s \xrightarrow{a} tZ$ (or internal transition $s \xrightarrow{a} t$), then a run of $\mathcal{A}$ may evolve from configuration $s\gamma$ to $tZ\gamma$ (or $t\gamma$, respectively). Similarly, if $\mathcal{A}$ reads a return symbol $a$ in configuration $sZ\gamma$ where $Z \neq \perp$ and there is a transition $sZ \xrightarrow{a} t$, then a run can move from $sZ\gamma$ to $t\gamma$. Note that invoking a return transition in a bottom configuration $s\perp$ does not remove the topmost symbol $\perp$ from the stack.

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 $\omega$-language of words accepted by a Büchi VPA $\mathcal{A}$ is denoted $\mathcal{L}(\mathcal{A}) \subseteq \Sigma^\omega$.

**Definition 2.2** (ω-VPL [AM04]). Let $\Sigma$ be a pushdown alphabet. $L \subseteq \Sigma^\omega$ is an $\omega$-visibly pushdown language (ω-VPL) if $L = \mathcal{L}(\mathcal{A})$ for a Büchi VPA $\mathcal{A}$ over $\Sigma$.

A VPA is deterministic (DVPA) if it has exactly one run on each input word. In this case, $\delta_{\text{call}}, \delta_{\text{int}},$ and $\delta_{\text{ret}}$ 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 [AM04]. A deterministic automaton model for $\omega$-VPL was given in [LMS04]. It uses a so-called stair-parity acceptance condition which we explain in the upcoming Section 2.2.

### 2.2. Steps and Stair-parity Conditions

In the remainder of this section, $\Sigma$ denotes a pushdown alphabet and $\mathcal{A}$ a VPA over $\Sigma$. 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 $sh(\rho(i)) = |\gamma_i| - 1$ (i.e., the bottom symbol $\perp$ does not count to the stack height). The stair-parity condition relies on the notion of steps:

**Definition 2.3** (Step). Let $\rho$ be a run of $\mathcal{A}$. Position $i \geq 0$ is a step of $\rho$ if
\[
\forall n \geq i: \quad sh(\rho(n)) \geq sh(\rho(i)).
\]

Intuitively, a step is a position of a run such that there is no future position where the stack height is strictly smaller. Slightly abusing terminology, we also say that a configuration $s_i\gamma_i$ of a given run $\rho = s_0\gamma_0, s_1\gamma_1, \ldots$ is a step if position $i$ is a step.
Steps play a central role in the rest of the paper. We therefore explain some of their fundamental properties. Suppose that $\rho$ is a run of $A$ on an infinite word $w \in \Sigma^*$.

- If positions $i < j$ are adjacent steps, i.e., there is no step $k$ strictly in between $i$ and $j$, 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.

- Each step $i$ has a next step $j > i$: If the symbol at step $i$ 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 position $j = i + 1$ 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 position $j = i + 1$. 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.

- As a consequence, each infinite run has infinitely many steps. Notice though that the difference between two adjacent step positions may grow unboundedly as in Figure 5.

- The stack height at the steps either grows unboundedly or eventually stabilizes (the latter occurs in Figure 5; the stack heights at the steps induce the sequence $0, 0, 0, 1^\omega$).

**Remark 2.5.** We 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 [LMS04]. 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 $\rho \downarrow_{\text{steps}} = s_0s_1s_2 \ldots \in S^\omega$ where for all $i \geq 0$ the
position $n_i$ is the $i$-th step of $\rho$. In words, $\rho \downarrow_{\text{Steps}}$ is the projection of the run $\rho$ onto the states occurring at its steps. For the example run in Figure 5, $\rho \downarrow_{\text{Steps}} = s_0 s_1 s_0 s_1^\omega$.

**Definition 2.6** (Stair-parity [LMS04]). Let $A$ be a VPA over pushdown alphabet $\Sigma$. A stair-parity acceptance condition for $A$ is defined in terms of a priority function $\Omega: S \to \mathbb{N}_0$. A word $w \in \Sigma^\omega$ is accepted if $A$ has a run $\rho$ on $w$ such that

$$\min \{ k \mid \exists i \geq 0: \Omega(\rho\downarrow_{\text{Steps}}(i)) = k \} \text{ is even.}$$

The language accepted by $A$ is denoted $L(A)$.

Intuitively, $L(A)$ contains all words that have a run $\rho$ on $A$ such that the minimum priority occurring infinitely often at states in the footprint $\rho \downarrow_{\text{Steps}}$ is even.

**Example 2.7.** The DVPA in Figure 5 with $\Omega(s_0) = 1$ and $\Omega(s_1) = 2$ accepts

$$L_{\text{repbdd}} = \{w \in \Sigma^\omega \mid \exists B \geq 0, \exists i \geq 0: \text{sh}(w(i)) = B\},$$

the language of repeatedly bounded words [LMS04], i.e., words whose stack height (cf. Remark 2.5) is infinitely often equal to some constant $B$. The example word from Figure 5 satisfies this property with $B = 1$. To see why the automaton accepts $L_{\text{repbdd}}$, note that a word is repeatedly bounded iff the stack height at the steps stabilizes eventually. The latter occurs iff in just finitely many cases, the transition before reaching a step was a call. The DVPA in Figure 5 detects this behavior; when reading a call symbol, it always moves to state $s_0$ which has odd priority, and it accepts iff $s_0$ is visited finitely often at call positions. It is known that $L_{\text{repbdd}}$ is not expressible through DVPA with usual parity conditions [AM04].

**Theorem 2.1** ([LMS04, Thm. 1]). For every non-deterministic Büchi VPA $A$ there exists a deterministic stair-parity DVPA $D$ with $2^{O(|S|^2)}$ states such that $L(A) = L(D)$. Moreover, $D$ can be constructed in exponential time in the size of $A$.

It was also shown in [LMS04] that stair-parity DVPA characterize exactly the class of $\omega$-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 [AEM04] is an extension of Linear Temporal Logic (LTL) [Pnu77] that can be used to describe $\omega$-VPL. Its syntax is defined as follows:

**Definition 2.8** (CaRet [AEM04]). Let $AP$ be a finite set of atomic propositions. The logic CaRet adheres to the grammar

$$\varphi ::= p \mid \varphi \lor \varphi \mid \neg \varphi \mid \Diamond^b \varphi \mid \varphi U^b \varphi \mid \Diamond^a \varphi \mid \varphi U^a \varphi \mid \Diamond \neg \varphi \mid \varphi U \neg \varphi,$$

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

Other common modalities such as $\Diamond^b$ and $\Box^b$ for $b \in \{g, a, -\}$ are defined as usual via $\Diamond^b \varphi = \text{true} U^b \varphi$ and $\Box^b \varphi = \neg \Diamond^b \neg \varphi$. We now explain the intuitive semantics of CaRet, the formal definition is stated further below. We assume familiarity with LTL. CaRet formulae are interpreted over infinite words from the pushdown alphabet $\Sigma = 2^{AP} \times \{\text{call, int, ret}\}$. $\Diamond^g$ and $U^g$ are the standard next and until modalities from LTL (called global next and until in CaRet). CaRet extends LTL by two key operators, the caller modality $\Diamond^\neg$ and the
The semantics of these operators is visually explained in Figure 6. The caller \( \bigcirc^- \) is a past modality that points 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. position 3 in Figure 6). On the other hand, the abstract successor of a call symbol is its matching return if it exists, or undefined otherwise. The until modalities \( U^- \) and \( U^a \) 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 expressing Hoare-like total correctness of a procedure \( F \) [AEM04]:

\[
\varphi_{\text{total}} = \Box^g (\text{call} \land p \land p_F \rightarrow \bigcirc^a q)
\]

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 2.7; it is described by the formula \( \varphi_{\text{replbd}} = \bigcirc^g \square^g (\text{call} \rightarrow \bigcirc^a \text{ret}) \) which states that all but finitely many calls have a matching return. Further examples are given in [AEM04].

We now define the semantics of CaRet formally. Let \( \Sigma = 2^{AP} \times \{\text{call, int, ret}\} \) be the pushdown alphabet and \( w \in \Sigma^\omega \). If \( w(i) \in \Sigma_{\text{call}} \), then let \( MR_w(i) \) be the position of the matching return of \( w(i) \), or \( \text{undef} \) if there is no matching return. The abstract successor of position \( i \) in word \( w \) is defined as follows:

\[
succ^a_w(i) = \begin{cases} 
MR_w(i) & \text{if } w(i) \in \Sigma_{\text{call}} \\
i + 1 & \text{if } w(i) \notin \Sigma_{\text{call}} \land w(i+1) \notin \Sigma_{\text{ret}} \\
\text{undef} & \text{if } w(i) \notin \Sigma_{\text{call}} \land w(i+1) \in \Sigma_{\text{ret}}
\end{cases}
\]

The caller \( \text{succ}^-_w(i) \) of position \( i \) in word \( w \) is defined as the innermost (“closest”) unmatched call position \( j < i \), or \( \text{undef} \) if there was no previous open call.

**Definition 2.9 (Semantics of CaRet).** Let \( \Sigma = 2^{AP} \times \{\text{call, int, ret}\} \) and \( w \in \Sigma^\omega \). For all CaRet formulas \( \varphi \) and \( i \in \mathbb{N}_0 \), we define \( w(i) \models \varphi \) by induction over the structure of \( \varphi \) as follows:

- \( w(i) \models p \iff w(i) = (X, \text{type}) \) and either \( p \in X \) or \( p = \text{type} \)
- \( w(i) \models \varphi_1 \lor \varphi_2 \iff w(i) \models \varphi_1 \lor w(i) \models \varphi_2 \)
- \( w(i) \models \neg \varphi \iff w(i) \not\models \varphi \)
- \( w(i) \models \bigcirc^g \varphi_1 \iff w(i+1) \models \varphi_1 \)
- \( w(i) \models \bigcirc^a \varphi_1 \iff \text{succ}^a_w(i) = j \in \mathbb{N}_0 \) is defined and \( w(j) \models \varphi_1 \)
- \( w(i) \models \bigcirc^- \varphi_1 \iff \text{succ}^-_w(i) = j \in \mathbb{N}_0 \) is defined and \( w(j) \models \varphi_1 \)
- \( w(i) \models \varphi_1 U^b \varphi_2 \) for \( b \in \{a, g, -\} \) if there exist positions \( i = i_0, i_1, \ldots, i_k, k \in \mathbb{N}_0 \), such that:
  1. \( w(i_k) \models \varphi_2 \), and
  2. for all \( 0 \leq j < k \), \( w(i_j) \models \varphi_1 \) and \( i_{j+1} = \text{succ}^b_w(i_j) \).

Furthermore, we write \( w \models \varphi \) if \( w(0) \models \varphi \). The language of all words satisfying a CaRet formula \( \varphi \) is denoted \( \mathcal{L}(\varphi) = \{ w \in \Sigma^\omega \mid w \models \varphi \} \).

**Theorem 2.2 ([AAB+07, Thm. 5.1]).** CaRet-definable languages are \( \omega \)-VPL: For each CaRet formula \( \varphi \) there exists a (non-deterministic) Büchi VPA \( A \) such that \( \mathcal{L}(\varphi) = \mathcal{L}(A) \), and \( A \) can be constructed in time \( 2^{O(|\varphi|)} \). \( \triangleq \)
The above theorem is well-known in the literature [AAB+07, ABE18] 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 even applies to the logic NWTL\(^+_\), an FO-complete extension of CaRet [AAB+07] which we do not consider here for the sake of simplicity.

Theorems 2.1 and 2.2 together imply that each CaRet formula can be translated to a deterministic stair-parity VPA of doubly-exponential size.

3. Probabilistic Visibly Pushdown Automata

As explained in the introduction, we employ probabilistic pushdown automata [EKM04] (pPDA) as an operational model for procedural probabilistic programs. pPDA thus play a fundamentally different role in this paper than VPA (cf. Definition 2.1): pPDA are used to model the system, while VPA 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 [DBB12], a syntactic visibility restriction 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 3.4 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 \( \text{Dist}(X) = \{ f : X \rightarrow [0,1] \mid \sum_{a \in X} f(a) = 1 \} \) for the set of probability distributions over \( X \).

**Definition 3.1** (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 \( Q = Q_{\text{call}} \uplus Q_{\text{int}} \uplus Q_{\text{ret}} \), \( q_0 \in Q \) is an initial state, \( \Gamma \) is a finite stack alphabet, \( \bot \in \Gamma \) is a special bottom-of-stack
symbol, $P = (P_{\text{call}}, P_{\text{int}}, P_{\text{ret}})$ is a triple of functions

\[ P_{\text{call}} : Q_{\text{call}} \rightarrow \text{Dist}(Q \times \Gamma_{\perp}) , \quad P_{\text{int}} : Q_{\text{int}} \rightarrow \text{Dist}(Q) , \quad P_{\text{ret}} : Q_{\text{ret}} \times \Gamma \rightarrow \text{Dist}(Q) , \]

$\Sigma = \Sigma_{\text{call}} \uplus \Sigma_{\text{int}} \uplus \Sigma_{\text{ret}}$ is a pushdown alphabet, and $\lambda : Q \rightarrow \Sigma$ is a state labeling function consistent with the visibility condition, i.e., for all type $\in \{\text{call, int, ret}\}$ and all $q \in Q$, it is required that $q \in Q_{\text{type}}$ iff $\lambda(q) \in \Sigma_{\text{type}}$.

Similar to VPA, we use the notation $q \xrightarrow{p} rZ$, $q \xrightarrow{p} r$, and $qZ \xrightarrow{p} r$ to indicate that $P_{\text{call}}(q)(r, Z) = p$, $P_{\text{int}}(q)(r) = p$, and $P_{\text{ret}}(q, Z)(r) = p$, respectively.

Intuitively, the behavior of a pVPA $\Delta$ is as follows. If the current state $q$ is a call state, then the probability distribution $P_{\text{call}}(q)$ determines a random successor state and stack symbol to be pushed on the stack ($\perp$ cannot be pushed). Similarly, if the current state $q$ is internal, then $P_{\text{int}}(q)$ is the distribution over possible successor states and no stack operation is performed. Lastly, if the current state $q$ is a return state and symbol $Z \in \Gamma_{\perp}$ is on top of the stack, then $Z$ is popped and $\Delta$ moves to a successor state with probability according to $P_{\text{ret}}(q, Z)$. As in VPA, the special symbol $\perp$ is not popped from the stack if a return transition occurs in a bottom configuration.

The formal semantics of a pVPA is defined in terms of a countably infinite discrete-time Markov chain. A (labeled) Markov chain is essentially the special case of a pVPA where $Q = Q_{\text{int}}$, with the only difference that we allow for countably infinite $Q$ and do not impose the restriction on the labeling function $\lambda$. A Markov chain can thus be specified as a 5-tuple $(Q, q_0, P, \Sigma, \lambda)$, i.e., we omit $\perp$ and $\Gamma$ from the definition of pVPA because a Markov chain does not use a stack. A run of a Markov chain is an infinite sequence $q_0q_1q_2 \ldots \in q_0Q^\omega$ of states. We extend the labeling function $\lambda$ from states to runs in the natural way.

We define the Markov chain generated by a pVPA $\Delta = (Q, q_0, \Gamma, \perp, P, \Sigma, \lambda)$ as

\[ D_\Delta = (Q \times (\Gamma_{\perp})^\ast \perp, q_0 \perp, P_\Delta, \Sigma, \lambda_\Delta) , \]

i.e., the state space of $D_\Delta$ is the set of configurations of $\Delta$ and the transition probability function $P_\Delta$ is defined as follows. $P_\Delta(q\gamma)(r\gamma') = p > 0$ iff exactly one of the following three cases applies:

1. $q \in Q_{\text{call}}$, $\gamma' = Z\gamma$ for some $Z \in \Gamma_{\perp}$ and $q \xrightarrow{2} rZ$;
2. $q \in Q_{\text{int}}$, $\gamma' = \gamma$ and $q \xrightarrow{1} r$; or
3. $q \in Q_{\text{ret}}$, $Z\gamma' = \gamma$ for some $Z \in \Gamma_{\perp}$ and $qZ \xrightarrow{p} q$, or $\gamma' = \gamma = \perp$ and $q \perp \xrightarrow{p} r$.

Moreover, the labeling function of $D_\Delta$ is $\lambda_\Delta(q\gamma) = \lambda(q)$ for all $q\gamma \in Q \times (\Gamma_{\perp})^\ast \perp$.

**Example 3.2.** Figure 7 depicts a pVPA $\Delta$ and a fragment of its generated Markov chain $D_\Delta$. Even though $D_\Delta$ is infinite, many question about pVPA remain decidable, including in particular questions about reachability probabilities which can be characterized as the least solution of a system of polynomial equations [EKM04]. We will use this extensively in Section 4.

We define the set $\text{Runs}_\Delta$ of a pVPA $\Delta$ as the runs of the Markov chain $D_\Delta$, i.e., $\text{Runs}_\Delta = (Q \times (\Gamma_{\perp})^\ast \perp)^\omega$. Steps of pVPA runs are defined as in Definition 2.3. A further example pVPA and its possible runs are depicted in Figure 8 on page 16.

We associate a probability space with $\text{Runs}_\Delta$ in the usual way (see, e.g., [BK08, Ch. 10]). To this end, we define the $\sigma$-algebra $F \subseteq 2^{\text{Runs}_\Delta}$ as the smallest set that contains all the cylinder sets $\rho.\text{Runs}_\Delta$, where $\rho$ is an arbitrary finite prefix of configurations, and that is
closed under complement and countable union. The sets in \( \mathcal{F} \) are called \textit{measurable} and there is a unique probability measure \( \mathbb{P}: \mathcal{F} \rightarrow [0,1] \) satisfying

\[
\mathbb{P}(\rho_0 \ldots \rho_n.\text{Runs}_\Delta) = \begin{cases} 
\prod_{i=0}^{n-1} P(\rho_i, \rho_{i+1}) & \text{if } \rho_0 = q_0 \perp, \\
0 & \text{otherwise,}
\end{cases}
\]

for all finite prefixes \( \rho_0 \ldots \rho_n \) of configurations.

In the following two remarks, we summarize the technical differences between our pVPA model and existing models in the literature.

\textbf{Remark 3.3.} Our pVPA only produce infinite runs and do not simply “terminate” upon reaching the empty stack as in most of the literature on probabilistic pushdown automata (pPDA), e.g., [EKM04]. Indeed, in our pVPA, the stack can never be empty because the special bottom symbol \( \perp \) cannot be popped. We have chosen this semantics for compatibility with CaRet which describes \( \omega \)-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 \( \perp \) for the first time. Another technical difference between our pVPA and the pPDA introduced in [EKM04] is that in pVPA, only pop transitions can read the stack, whereas in pPDA, all types of transitions can read, and possibly exchange, the current topmost stack symbol. We have chosen this definition (which is not a true restriction) for compatibility with VPA as defined in [AM04].

\textbf{Remark 3.4.} The visibility restriction of our pVPA is slightly different from the definition given in [DBB12] which requires all \textit{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’ \textit{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 \textit{internal} statements such as \( x := \text{bernoulli}(0.5) \) or \( x := \text{uniform}(0,3) \) 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 [DBB12] is not necessarily fulfilled for such programs.
We can now formally state our main problem of interest:

**Definition 3.5 (Probabilistic CaRet Model Checking).** Let $AP$ be a finite set of atomic propositions, $\varphi$ be a CaRet formula over $AP$, $\Delta$ be a pVPA with labels from the pushdown alphabet $\Sigma = 2^{AP} \times \{\text{call, int, ret}\}$, and $\theta \in [0, 1] \cap \mathbb{Q}$. The **quantitative CaRet Model Checking problem** is to decide whether

$$P\left(\{\rho \in \text{Runs}_\Delta \mid \lambda(\rho) \in L(\varphi)\} \right) \geq \theta.$$ 

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

The probabilities in Definition 3.5 are well-defined as $\omega$-VPL are measurable [LMS04].

4. **Model Checking pVPA against Stair-parity DVPA**

In this section, we show that model checking pVPA (Definition 3.1) against VPL given in terms of a stair-parity DVPA (Definition 2.6) 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, internal, or pop) are forced to synchronize.

We now define the product formally. An **unlabeled** pVPA is a pVPA where the labeling function $\lambda$ and alphabet $\Sigma$ are omitted.

**Definition 4.1 (Product $\Delta \times D$).** Let $\Delta = (Q, q_0, \Gamma, \bot, P, \Sigma, \lambda)$ be a pVPA, and $D = (S, s_0, \Gamma', \bot, \delta, \Sigma)$ be a DVPA over pushdown alphabet $\Sigma$. The product of $\Delta$ and $D$ is the unlabeled pVPA

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

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

$$(\text{call}) \quad q \xrightarrow{p_{\Delta \times D}} rZ \land s \xrightarrow{\lambda(q)_{D}} tY$$

$$(q, s) \xrightarrow{p_{\Delta \times D}} (r, t)(Z, Y)$$

$$(\text{return}) \quad qZ \xrightarrow{p_{\Delta \times D}} r \land sY \xrightarrow{\lambda(q)_{D}} t$$

$$(q, s) \xrightarrow{P_{\Delta \times D}} (r, t)(Z, Y)$$

$$(\text{internal}) \quad q \xrightarrow{p_{\Delta \times D}} r \land s \xrightarrow{\lambda(q)_{D}} t$$

$$(q, s) \xrightarrow{p_{\Delta \times D}} (r, t).$$

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

It is not difficult to show that $\Delta \times D$ is indeed a well-defined pVPA and moreover satisfies the following property (the proof is standard, see [WGK21]):
Lemma 4.2 (Soundness of $\Delta \times D$). Let $\Delta$ be a pVPA and $D$ be a stair-parity DVPA with priority function $\Omega$, both over pushdown alphabet $\Sigma$. Then the product pVPA $\Delta \times D$ with priority function $\Omega'$ as in Definition 4.1 satisfies

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

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

Remark 4.3. It is not actually important that the product pVPA again satisfies the visibility condition, even though this happens to be the case. All techniques we apply to the product also work for general pPDA.

4.2. Stair-parity Acceptance Probabilities in pVPA. Lemma 4.2 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 $M_\Delta$ that we call the step chain of $\Delta$. Loosely speaking, $M_\Delta$ simulates jumping from one step (see Definition 2.3) of a run of $\Delta$ to the next. A similar idea first appeared in [EKM04]. Our construction, however, differs from the original one in various aspects. We discuss this in detail in Remark 4.15 further below.

4.2.1. Steps as events. For all $n \in \mathbb{N}_0$, we define a random variable $V^{(n)}$ on $\text{Runs}_\Delta$ whose value is either the state $q$ of $\Delta$ 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 $q \bot$, for some $q \in Q$. We denote the set of all such extended states with $Q \bot = \{ q \bot \mid q \in Q \}$. Formally, $V^{(n)} : \text{Runs}_\Delta \to Q \cup Q \bot$ is defined as

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

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

Lemma 4.4. 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.

Proof. This was proved more generally in [EKM04]. Here we give an alternative proof using the fact that all $\omega$-VPL are measurable [LMS04]. We view $Q \cup Q \bot$ as a pushdown alphabet (the partition is induced by the partition on $Q$). We can construct a non-deterministic Büchi VPA that accepts a word $w \in (Q \cup Q \bot)^\omega$ iff the $n$-th step of $w$ is $v$ (the size of this automaton depends on $n$). To this end, the VPA guesses the first $n$ symbols that are steps and goes to an accepting state $s \in F$ if the $n$-th step was $v$. The automaton can verify that it guessed correctly as follows. If it believes a call symbol is a step, it pushes a special marker on the stack; if this marker is ever popped, then the call was no step and the guess was wrong. If it detects a wrong guess in this way, then it leaves the accepting state $s$, otherwise it loops there forever. The claim follows because the function $f : \text{Runs}_\Delta \to (Q \cup Q \bot)^\omega$ that
maps runs to the sequence of their (extended) states is measurable (indeed, the preimage \( f^{-1}(w) \) of every \( w \in (Q \cup Q \perp)^\omega \) is

\[
f^{-1}(w) = \bigcap_{i \geq 0} \bigcup_{\gamma_0, \ldots, \gamma_i \in (\Gamma \perp)^*, \perp} w(0)\gamma_0 \ldots w(i)\gamma_i.\text{Runs}_\Delta
\]

which is a countable intersection of countable unions of basic cylinder sets).

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 only the state \( q \) at any step configuration \( q\gamma \) (with \( \gamma \neq \perp \)) fully determines the future of the run because being a step already implies that no symbol in \( \gamma \) 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 \( q \perp \). Phrased in probability theoretical terms, the process \( V^{(0)}, V^{(1)} \ldots \) has the Markov property, i.e.,

\[
\mathbb{P}(V^{(n)} = v_n \mid V^{(n-1)} = v_{n-1} \wedge \ldots \wedge V^{(0)} = v_0) = \mathbb{P}(V^{(n)} = v_n \mid V^{(n-1)} = v_{n-1}) \quad \text{(4.1)}
\]

holds for all values of \( v_0, \ldots, v_n \) such that the conditional probabilities are well-defined\(^1\). This was proved in detail in [EKM04]. It is also clear that the Markov process is time-homogeneous in the sense that

\[
\mathbb{P}(V^{(n+1)} = v' \mid V^{(n)} = v) = \mathbb{P}(V^{(m+1)} = v' \mid V^{(m)} = v) \quad \text{(4.2)}
\]

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

**Example 4.5.** Consider the pVPA in Figure 8 (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 \( v_n, v_{n-1} \in Q \cup Q \perp \). They are summarized in the Markov chain on the right. For example, \( V^{(0)} = q_0 \perp \) holds with probability 1, and \( V^{(1)} = q_1 \) and \( V^{(1)} = q_3 \perp \) hold with probability \( \frac{1}{2} \) each because the second step occurs either at position 1 with configuration \( q_1 \perp Z \) or at position 3 with configuration \( q_3 \perp \), 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 \( \gamma \neq \perp \) 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 \( \Delta \), the next state cannot be \( q_3 \) because the latter is a return state which would immediately decrease the stack height of \( \gamma \). This shows that, intuitively speaking, conditioning on being a step influences the probabilities of a state’s outgoing transitions.

4.2.2. **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 4.5. It turns out that those can be stated in terms of the return and diverge probabilities of \( \Delta \).

**Definition 4.6.** Let \( q, r \in Q \) be states, and \( Z \in \Gamma \perp \) be a stack symbol of pVPA \( \Delta \). We define the following probabilities:

\(^1\)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 (4.1) 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 \( v_{n-1} \).
Figure 8: Left: An example (unlabeled) pVPA $\Delta$. Recall that call and return states are drawn as squares and rhombs, respectively, whereas internal states are depicted as circles. Middle: Initial fragments of the two possible runs of $\Delta$. Steps are underlined. Right: The step Markov chain $M_\Delta$ (Definition 4.13, page 22).

Figure 9: Next-step probabilities of the step Markov chain. $P_{\text{type}}$ for $\text{type} \in \{\text{call}, \text{int}, \text{ret}\}$ 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 4.6.

- The return probability $[qZ \downarrow r]$ is the probability to reach configuration $r \perp$ from $qZ \perp$ without visiting another bottom configuration $t \perp$ for some $t \neq r$ in between. More formally,

$$[qZ \downarrow r] = \mathbb{P}_{qZ \perp}( \{ q' \gamma \perp \mid q' \in Q, \gamma \in (\Gamma \perp)^+ \} \cup \{ r \perp \})$$

where $\mathbb{P}_{qZ \perp}$ is the probability measure associated with the infinite Markov chain $D_\Delta$ assuming initial state $qZ \perp$, and $\mathcal{U}$ is the standard until operator from LTL.

- The diverge probability $[q^\uparrow] = 1 - \sum_{t \in Q} [qZ \downarrow t]$, i.e., the probability to never pop $Z$ from the stack when starting in $qZ \perp$. Note that $[q^\uparrow]$ is indeed independent of $Z$ because the only way to read $Z$ is by popping it from the stack.

Example 4.7. We give a minimal example illustrating that the return and diverge probabilities are in general non-rational. Consider the following pVDA:
Intuitively, this pVPA either pops the topmost symbol with probability $1/2$, or it pushes two times the symbol $Z$. Note that all return probabilities of the form $[\ldots \downarrow q_i]$ for $i \neq 0$ are equal to zero. In [EKM04] it was shown that the remaining return probabilities are the least non-negative solution of the polynomial system:

\[
\begin{align*}
[q_0 Z \downarrow q_0] &= \frac{1}{2}[q_1 Z \downarrow q_0] + \frac{1}{2}[q_3 Z \downarrow q_0] \\
[q_1 Z \downarrow q_0] &= [q_2 Z \downarrow q_0] \cdot [q_0 Z \downarrow q_0] \\
[q_2 Z \downarrow q_0] &= [q_0 Z \downarrow q_0] \cdot [q_0 Z \downarrow q_0] \\
[q_3 Z \downarrow q_0] &= 1.
\end{align*}
\]

It follows that $[q_0 Z \downarrow q_0]$ must be the least non-negative solution of

\[
[q_0 Z \downarrow q_0] = \frac{1}{2}[q_0 Z \downarrow q_0]^2 + \frac{1}{2}
\]

which is $[q_0 Z \downarrow q_0] = (\sqrt{5} - 1)/2 \approx 0.618$, the reciprocal of the golden ratio. △

The probabilities in Example 4.7 can still be expressed by radicals (square roots, cubic roots, and so on) which allows for certain effective computations. However, in general, the probabilities cannot even be expressed in this way. For example, consider a pVPA that repeats the following steps until emptying its stack or getting stuck in a sink state: (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 $x = (1/6)x^5 + 1/2$, which is an algebraic number not solvable by radicals [EY09, Thm. 3.2(1)].

**Remark 4.8.** The probabilities $[qZ \downarrow r]$ from Definition 4.6 were called *termination probabilities* in previous work [BEKK13]. However, we believe that *return* probability is more appropriate. When modeling procedural probabilistic programs as pVPA, $[qZ \downarrow r]$ is just the probability to eventually return from local state $q$ of the current procedure to local state $r$ of the calling procedure (the return address is stored on the stack in $Z$). We believe that the term *termination* probability is more adequate for referring to the quantity $\sum_{r \in Q}[q_0 Z_0 \downarrow r]$, where $Z_0$ is some initial stack symbol, i.e., the probability that some initial procedure indentified by $Z_0$ returns at all when started in local state $q_0$.

**Lemma 4.9.** The conditional next-step probabilities in Figure 9 are correct in the sense that if $P(V^{(n+1)} = v' \mid V^{(n)} = v)$ 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 $v \rightarrow v'$.

4.2.3. **Proof sketch of Lemma 4.9.** We first explain the trivial entries of the table in Figure 9. Further below, we give a self-contained proof of the two non-trivial expressions in the left-most column of Figure 9. Throughout the whole proof we fix a pVPA $\Delta = (Q, q_0, \Gamma, \bot, P, \Sigma, \lambda)$, with $P = (P_{\text{call}}, P_{\text{int}}, P_{\text{ret}})$ the call, internal, and return transition
functions, respectively. The following items correspond to the trivial entries in Figure 9 and are ordered column-by-column, from left to right:

- The probability $P(V^{(n+1)} = r \mid V^{(n)} = q)$ with $q \in Q_{\text{ret}}$ is never well-defined because it is impossible that steps occur at non-bottom configurations with a return state.
- The probability $P(V^{(n+1)} = r \mid V^{(n)} = q_\bot)$ with $q \in Q_{\text{int}}$ is trivially zero because if $q$ is internal then the next step after a bottom configuration $q_\bot$ is necessarily also a bottom configuration.
- The probability $P(V^{(n+1)} = r \mid V^{(n)} = q_\bot)$ with $q \in Q_{\text{ret}}$ is trivially zero because if $q$ is a return state at a bottom configuration then the next step occurs at the immediate successor configuration which is a bottom configuration as well.
- The probability $P(V^{(n+1)} = r_\bot \mid V^{(n)} = q_\bot)$ with $q \in Q_{\text{int}}$ is straightforward because $q_\bot$ and $r_\bot$ are both steps and the probability that the next state after $q_\bot$ is $r_\bot$ is $P_{\text{int}}(q, r)$.
- The probability $P(V^{(n+1)} = r_\bot \mid V^{(n)} = q_\bot)$ with $q \in Q_{\text{ret}}$ is simply $P_{\text{ret}}(q_\bot, r)$ for the same reason as in the previous case (recall that a return state at a bottom-configuration behaves exactly like an internal one).
- All the remaining probabilities in the rightmost column “$q \rightarrow r_\bot$” are trivially zero or ill-defined because if a step occurs at non-bottom configuration, then the next step can never occur at a bottom configuration.

We now focus on the following non-trivial cases. Let $r \in Q$ and $n \in \mathbb{N}_0$ be arbitrary.

1. If $q \in Q_{\text{int}}$ then,
   \[
P(V^{(n+1)} = r \mid V^{(n)} = q) = P_{\text{int}}(q, r)[r \uparrow] / [q \uparrow].
   \]

2. If $q \in Q_{\text{call}}$ then,
   \[
P(V^{(n+1)} = r \mid V^{(n)} = q) = [r \uparrow] / [q \uparrow] \left( \sum_{r', Z} P_{\text{call}}(q, r'Z)[r' \downarrow] + \sum_{Z} P_{\text{call}}(q, rZ) \right).
   \]

The other two non-trivial cases are essentially subsumed by case (2). For the complete proof we refer to [WGK21]. Before giving the formal proof, we first provide some intuition on the above expressions.

- For (1), 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 $P_{\text{int}}(q, r)[r \uparrow]$ 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 $\{ P_{\text{int}}(q, r)[r \uparrow] \mid r \in Q \}$ do not form a probability distribution in general. Therefore we divide by the normalizing constant $[q \uparrow] = \sum_{r \in Q} P_{\text{int}}(q, r)[r \uparrow]$.

- In (2), the two summands correspond to the following case distinction: Since the $n$-th step occurs at the call state $q$, the $n+1$-st step either (i) occurs at the same stack height as the current step $n$ (which means that the current call has a matching return), or (ii) the stack height at the next step $n+1$ is incremented by 1 compared to the stack height at step $n$. In case (ii), the next step occurs immediately at the next position, which is why the second summand is just the 1-step probability to go from $q$ to $r$. In case (i), the symbols pushed by the outgoing transitions of $q$ must be eventually popped. For instance, if we assume that $q$ takes a transition to an immediate successor $r'$ and pushes
Z on the stack, then the probability that the next step occurs at r is precisely the \textit{return probability} \([rZ \downarrow r]\) (see Definition 4.6). The factor \([r\uparrow]/[q\uparrow]\) is needed for similar reasons as in (1).

We now give the formal proofs. In the following, we often use equations involving conditional probabilities such as \(\mathbb{P}(A \mid B) = \mathbb{P}(C \mid D)\). These conditional probabilities are not necessarily well-defined in all cases. Therefore, the meaning of our equations is that they hold only if all probabilities involved are well-defined. We need some definitions and (simple) lemmas first.

\textbf{Definition 4.10.} Let \(i \in \mathbb{N}_0\) and \(q \in Q\). We introduce the following events:

- \(q@i\) is the set of all runs \(\rho \in \text{Runs}_\Delta\) such that \(\rho(i) = q\gamma\) with \(\gamma \neq \perp\), i.e., the runs whose \(i\)-th configuration has state \(q\) and stack unequal to \(\perp\).
- Similarly, \(q\perp@i\) denotes the set of all runs \(\rho \in \text{Runs}_\Delta\) such that \(\rho(i) = q\perp\), i.e., the runs whose \(i\)-th configuration is a bottom configuration with state \(q\).
- \(\text{step}@i\) denotes the set of all runs such that the \(i\)-th configuration is a step.
- \(\text{sh}@i = \text{sh}(\rho(i)) \in \mathbb{N}_0\), i.e., the stack height of the \(i\)-th configuration. Strictly speaking, \(\text{sh}@i\) is a random variable, not an event.

Note that these events are all measurable. \(\triangle\)

\textbf{Lemma 4.11.} For all \(i \in \mathbb{N}_0\) and \(q \in Q\), the following identities hold:

\[
\mathbb{P}(\text{step}@i \mid q@i) = [q\uparrow] \tag{4.3}
\]

\[
\mathbb{P}(\text{step}@i \mid q\perp@i) = 1 \tag{4.4}
\]

Further, for \(q \in Q_{\text{int}}\) and \(r \in Q\),

\[
\mathbb{P}(r@\!(i+1) \mid q@i) = P_{\text{int}}(q, r) \tag{4.5}
\]

\[
\mathbb{P}(\text{step}@\!(i+1) \land \text{step}@i \mid r@\!(i+1) \land q@i) = \mathbb{P}(\text{step}@\!(i+1) \mid r@\!(i+1)) \tag{4.6}
\]

\textbf{Proof.} The first three equations all follow immediately from the definitions. For (4.6) we have:

\[
\mathbb{P}(\text{step}@\!(i+1) \land \text{step}@i \mid r@\!(i+1) \land q@i) \\
= \mathbb{P}(\text{step}@\!(i+1) \mid r@\!(i+1) \land q@i) \\
= \mathbb{P}(\text{step}@\!(i+1) \mid r@\!(i+1))
\]

The first equation holds because if \(q \in Q_{\text{int}}\) and \(q@i\), then \(\text{step}@\!(i+1) \land \text{step}@i\) is already implied by \(\text{step}@\!(i+1)\), and the second equation holds because the probability that \(\text{step}@\!(i+1)\) depends only on the state at position \(i+1\), not on the state at position \(i\). \(\square\)

To prove case (1) we argue as follows. By time-homogeneity (see (4.2)) and the definition of \(V^{(n)}\), we have for all \(i, n \in \mathbb{N}_0\), \(q \in Q_{\text{int}}\) and \(r \in Q\) that

\[
\mathbb{P}(V^{(n+1)} = r \mid V^{(n)} = q) = \mathbb{P}(r@\!(i+1) \land \text{step}@\!(i+1) \mid q@i \land \text{step}@i) \tag{4.7}
\]

We can now derive the equation for (1) as follows:

\[
\mathbb{P}(V^{(n+1)} = r \mid V^{(n)} = q) \\
= \mathbb{P}(r@\!(i+1) \land \text{step}@\!(i+1) \mid q@i \land \text{step}@i) \tag{by (4.7)} \\
= \frac{\mathbb{P}(r@\!(i+1) \land \text{step}@\!(i+1) \land q@i \land \text{step}@i)}{\mathbb{P}(q@i \land \text{step}@i)} \tag{cond. probability}
\]
For the second summand of the last expression we obtain that:

\[
\mathbb{P}(r \uparrow (i+1) \wedge \text{step} \uparrow (i+1) \wedge \text{step} \uparrow i \mid q @i) \cdot \mathbb{P}(q@i) = \sum_{Z} \mathbb{P}(r \uparrow (i+1) \wedge \text{step} \uparrow (i+1) \mid q @i) \cdot P_{\text{call}}(q, rZ) \cdot P_{\text{int}}(q, r)
\]

by a similar derivation as in case (1).

**Lemma 4.12.** The following equations hold for all \( q \in Q_{\text{call}} \):

\[
\sum_{r', Z} P_{\text{call}}(q, r'Z) [r'Z \downarrow r] = \sum_{k \triangleright i + 1} \mathbb{P}(r \uparrow k \wedge \forall i < j < k : \text{sh} \uparrow j > \text{sh} \uparrow k = \text{sh} \uparrow i \mid q@i) \tag{4.8}
\]

\[
\mathbb{P}(\text{step} \uparrow k \wedge \forall i < j < k : \neg \text{step} \uparrow j \wedge \text{step} \uparrow i \mid q@i)
\]

\[
= \mathbb{P}(\text{step} \uparrow k \wedge \forall i < j < k : \text{sh} \uparrow j > \text{sh} \uparrow i = \text{sh} \uparrow k \mid q@i) \tag{4.9}
\]

\[
\mathbb{P}(\forall i < j < k : \text{sh} \uparrow j > \text{sh} \uparrow i = \text{sh} \uparrow k \mid \text{step} \uparrow k) = \mathbb{P}(\forall i < j < k : \text{sh} \uparrow j > \text{sh} \uparrow i = \text{sh} \uparrow k) \tag{4.10}
\]

**Proof.** (4.8): \( \sum_{r', Z} P_{\text{call}}(q, r'Z) [r'Z \downarrow r] \) describes the probability to go from \( q \) (with empty stack) to a successor state \( r' \), push \( Z \in \Gamma \) and then later reach \( r \) with empty stack within finitely many steps. If we assume that \( q@i \), then this is the same as summing over all positions \( k > i + 1 \) (\( k = i + 1 \) is not possible because it is not possible to push and pop \( Z \) within one transition) such that \( r@k \) and for all \( i < j < k \) the stack height is greater than at position \( i \) and \( k \). Positions \( i \) and \( k \) have the same stack height because in the transition from \( i \) the symbol \( Z \) is pushed and \( k \) is the position directly after \( Z \) is popped. In between those two transitions, the stack below \( Z \) cannot change, so the stack is the same at both positions.

(4.9): 

\[
\mathbb{P}(\text{step} \uparrow k \wedge \forall i < j < k : \neg \text{step} \uparrow j \wedge \text{step} \uparrow i \mid q@i)
\]

\[
= \mathbb{P}(\text{step} \uparrow k \wedge \forall i < j < k : \text{sh} \uparrow j > \text{sh} \uparrow i = \text{sh} \uparrow k \wedge \text{step} \uparrow i \mid q@i)
\]
= P(step@k \land \forall i<j<k : sh@j > sh@i \Rightarrow sh@k | q@i)

The first equality holds because if no position between \(i\) and \(k\) is a step, then the stack at those positions must be higher than at position \(k\). Furthermore, since \(i\) is a step, we have \(sh@i \leq sh@k\); and moreover, since \(i+1\) is not a step and \(q\) is a call state, we even have \(sh@i = sh@k\). The second equality holds because if \(i\) and \(k\) have the same stack height and all positions between them have a higher stack, then \(i\) is a step if and only of \(k\) is a step.

(4.10): The probability \(P(\forall i<j<k : sh@j > sh@i = sh@k)\) is independent of the event \(step@k\) because knowing whether or not there is a position with lower stack height after position \(k\) does not influence the stack height before \(k\). □

For the first summand of (2) we now argue as follows:

\[
\begin{align*}
\mathbb{P}(\exists k > i+1 : \text{step}@k \land r@k \land \forall i<j<k : \neg \text{step}@j \land \text{step}@i | q@i) \\
= \mathbb{P}(\exists k > i+1 : \text{step}@k \land r@k \land \forall i<j<k : \neg \text{step}@j \land \text{step}@i | q@i) \cdot P(q@i) \\
= \sum_{k>i+1} \mathbb{P}(\text{step}@k \land r@k \land \forall i<j<k : \neg \text{step}@j \land \text{step}@i | q@i) \cdot \frac{P(q@i)}{P(step@i | q@i)} \\
= \sum_{k>i+1} \mathbb{P}(\text{step}@k \land r@k \land \forall i<j<k : \neg \text{step}@j \land \text{step}@i | q@i) \cdot \frac{P(q@i)}{P(step@i | q@i)} \\
= \sum_{k>i+1} \mathbb{P}(\text{step}@k \land r@k | q@i) \cdot \frac{P(\exists i < j < k : sh@j > sh@i = sh@k | q@i \land \text{step}@k \land r@k)}{P(step@i | q@i)} \\
= \sum_{k>i+1} \mathbb{P}(\text{step}@k \land r@k | q@i) \cdot \frac{P(\exists i < j < k : sh@j > sh@i = sh@k \land r@k | q@i)}{P(step@i | q@i)} \\
= \sum_{k>i+1} \mathbb{P}(\text{step}@k | q@i \land r@k) \cdot \frac{P(\exists i < j < k : sh@j > sh@i = sh@k \land r@k | q@i)}{P(step@i | q@i)} \\
= \sum_{k>i+1} \mathbb{P}(\text{step}@k | q@i \land r@k) \cdot \frac{P(\exists i < j < k : sh@j > sh@i = sh@k \land r@k | q@i)}{P(step@i | q@i)} \\
= \sum_{k>i+1} \mathbb{P}(\text{step}@k | q@i \land r@k) \cdot \frac{P(\exists i < j < k : sh@j > sh@i = sh@k \land r@k | q@i)}{P(step@i | q@i)} \\
= \sum_{k>i+1} \mathbb{P}(\text{step}@k | q@i \land r@k) \cdot \frac{P(\exists i < j < k : sh@j > sh@i = sh@k \land r@k | q@i)}{P(step@i | q@i)} \\
= \frac{[r \uparrow q]}{[q \uparrow]} \sum_{k>i+1} \mathbb{P}(r@k \land \forall i<j<k : sh@j > sh@i = sh@k | q@i) \\
= \frac{[r \uparrow q]}{[q \uparrow]} \sum_{r', Z} P_{\text{call}}(q, r'Z)[r' Z \downarrow r] \\
& \quad \text{(by (4.3))}
\end{align*}
\]
This concludes the proof of case (2). \hfill \Box

4.2.4. The step chain. Recall from (4.1) that the stochastic process $V^{(0)}, V^{(1)} , \ldots$, where $V^{(i)} \in Q\cup Q\perp$ is the extended state at the $i$th step, has the Markov property. Since $Q\cup Q\perp$ is a finite set, we can now use Lemma 4.9 to construct the underlying finite Markov chain explicitly.

**Definition 4.13** (The Step Chain $M_\Delta$). $M_\Delta$ is the Markov chain with states

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

initial state $q_0\perp$, and for all $v, v' \in M$, the probability of transition $v \rightarrow v'$ is defined according to Figure 9.

Figure 10 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 4.6) are algebraic numbers that are not always rational or even expressible by radicals [EY09] (cf. Example 4.7). 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 $(\mathbb{R}, +, \cdot, \leq)$ [EKM04]. Since the ETR is decidable in PSPACE, many questions about the step chain are in PSPACE as well. We will make use of this in Theorem 4.1 below.

The property of $M_\Delta$ that is most relevant to us is given by the following Lemma 4.14. We call $\rho \downarrow_{\text{steps}} = V^{(0)}(\rho)V^{(1)}(\rho) \ldots$ the extended footprint of run $\rho$.

**Lemma 4.14** (Soundness of $M_\Delta$). Let $\Delta$ be a pVPA with step chain $M_\Delta$. Let $M$ be the states of the step chain and consider a measurable set $R \subseteq M^\omega$. Then

$$\mathbb{P}(\{ \rho \in \text{Runs}_\Delta \mid \rho \downarrow_{\text{steps}} \in R \}) = \mathbb{P}(R).$$

**Proof.** We first show the case where $R$ is a basic cylinder set, i.e., $R = w.M^\omega$ for some $w \in M^*$. Let $|w| = n$. By the Markov property (4.1), we have

$$\mathbb{P}(V^{(n)} = w(n) \land \ldots \land V^{(0)} = w(0))$$

$$= \mathbb{P}(V^{(n)} = w(n) \mid V^{(n-1)} = w(n-1)) \cdot \ldots \cdot \mathbb{P}(V^{(1)} = w(1) \mid V^{(0)} = w(0))$$
where \( P(s, t) \) denotes the 1-step transition probability from states \( s \) to \( t \) in the Markov chain \( M_\Delta \) and the last equality holds by definition of the probability space associated to a Markov chain.

It remains to show the claim for measurable sets \( R \) that are not of the above form \( R = w. M^\omega \). To this end, we make an induction over the levels of the Borel hierarchy, i.e., we show that if the claim holds for a set \( R \), then it also holds for the complement \( M^\omega \setminus R \); and similarly, if it holds for a countable family of pairwise disjoint sets \( R_0, R_1, \ldots \), then it holds for \( \bigcup_{i \in \mathbb{N}_0} R_i \). It is indeed sufficient to show the claim for pairwise disjoint sets, as a countable union \( \bigcup_{i \in \mathbb{N}_0} R_i \) of not necessarily pairwise disjoint \( R_i \) can be rewritten as

\[
\bigcup_{i \in \mathbb{N}_0} R_i = \bigcup_{i \in \mathbb{N}_0} \left( R_i \setminus \bigcup_{j<i} R_j \right).
\]

Complement:

\[
\mathbb{P}(\{ \rho \in \text{Runs}_\Delta \mid \rho \downarrow \text{Steps} \in M^\omega \setminus R \}) = \mathbb{P}(\{ \rho \in \text{Runs}_\Delta \mid \rho \downarrow \text{Steps} \notin R \})
\]

\[
= \mathbb{P}(\text{Runs}_\Delta \setminus \{ \rho \in \text{Runs}_\Delta \mid \rho \downarrow \text{Steps} \in R \})
\]

\[
= 1 - \mathbb{P}(\{ \rho \in \text{Runs}_\Delta \mid \rho \downarrow \text{Steps} \in R \}) \quad \text{(Def. of probability space)}
\]

\[
= 1 - \mathbb{P}(R) \quad \text{(By the I.H.)}
\]

\[
= \mathbb{P}(M^\omega \setminus R) \quad \text{(Def. of probability space)}
\]

Disjoint union:

\[
\mathbb{P}(\{ \rho \in \text{Runs}_\Delta \mid \rho \downarrow \text{Steps} \in \bigcup_{i \in \mathbb{N}_0} R_i \})
\]

\[
= \mathbb{P}(\bigcup_{i \in \mathbb{N}_0} \{ \rho \in \text{Runs}_\Delta \mid \rho \downarrow \text{Steps} \in R_i \})
\]

\[
= \sum_{i \in \mathbb{N}_0} \mathbb{P}(\{ \rho \in \text{Runs}_\Delta \mid \rho \downarrow \text{Steps} \in R_i \}) \quad \text{(Def. of probability space)}
\]

\[
= \sum_{i \in \mathbb{N}_0} \mathbb{P}(R_i) \quad \text{(By the I.H.)}
\]

\[
= \mathbb{P}(\bigcup_{i \in \mathbb{N}_0} R_i) \quad \text{(Def. of probability space)}
\]

\[
= \mathbb{P}(R)
\]

\[\square\]

**Remark 4.15.** The step chain as presented here differs from the original definition in [EKM04] 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 [EKM04], 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 [EKM04], namely to show that general measurable properties defined on steps—this includes stair-parity—can be evaluated on pVPA (Lemma 4.14).

The following is the main result of this section.

**Theorem 4.1.** Let $\Delta$ be a pVPA and $D$ be a stair-parity DVPA, both over the same push-down alphabet $\Sigma$. Then for all $\theta \in [0, 1] \cap \mathbb{Q}$, the problem $P\{(\rho \in \text{Runs}_\Delta \mid \lambda(\rho) \in L(D))\} \geq \theta$ is decidable in PSPACE.

4.2.5. *Proof of Theorem 4.1.* In this section we give a self-contained proof of Theorem 4.1. We first provide a brief overview. The first step is to construct the product $\Delta \times D$ according to Definition 4.1. By Lemma 4.2 we need to compute the stair-parity acceptance probability of $\Delta \times D$. Lemma 4.14 reduces this to computing a usual parity acceptance probability in the step chain $M_{\Delta \times D}$. This can be achieved through finding the bottom strongly connected components (BSCC) of $M_{\Delta \times D}$, classifying them as good (the minimum priority of a BSCC state is even) or otherwise as bad, and running a standard reachability analysis w.r.t. the good BSCCs (see Figure 11 for an example). The remaining technical difficulty is that the transition probabilities of $M_{\Delta \times D}$ are not rational in general. We handle this using the fact that these probabilities are expressible in the ETR [EKM04].

We now present the formal proof. We use the following result about return probabilities of pPDA, which is originally due to [EKM04]:

**Lemma 4.16** (as found in [BEKK13, Thm. 2]). The return probabilities $[pZ \downarrow q]$ are expressible in ETR. More specifically, there exists an FO-formula $\Phi$ over $(\mathbb{R}, +, \cdot, \leq)$ which uses just existential quantifiers and free variables $\langle pZ \downarrow q \rangle_{p,q \in Q, Z \in \Sigma}$ such that $\Phi$ becomes a true FO-sentence iff each free variable $\langle pZ \downarrow q \rangle$ is substituted by the actual return probability $[pZ \downarrow q]$. Moreover, $\Phi$ can be effectively constructed in polynomial space.

**Lemma 4.17.** The next-step probabilities (i.e., the transition probabilities of the step chain) in Figure 9 are expressible in ETR.

*Proof.* With the previous lemma it suffices to note that ETR expressible numbers are closed under addition, multiplication and division. Let $x, y \in \mathbb{R}$ be expressed by ETR formulae $\Phi(x)$ and $\Phi'(y)$, respectively. Then the formula $\Phi''(z) := \exists x, y: \Phi(x) \land \Phi'(y) \land z = x + y$ where $z$ is a fresh variable expresses the sum of $x$ and $y$, and similar for multiplication. For division, we have that $\Phi''(z) := \exists x, y: \Phi(x) \land \Phi'(y) \land z \cdot y = x$ expresses $x/y$ provided that $y \neq 0$ (if $y = 0$ then $\Phi''$ does not express a unique real number as it is then either unsatisfiable or trivial).

We now describe our PSPACE algorithm to prove Theorem 4.1.

**Step 1.** We first construct the product pVPA $\Delta = \Delta \times D$ with priority function $\Omega': Q \rightarrow \mathbb{N}_0$ where $Q$ are the states of $\Delta$ as in Definition 4.1. By Lemma 4.2 it suffices to compute the probability

$$P\{(\pi \in \text{Runs}_\Delta \mid \rho \downarrow_{\text{steps}} \in \text{Parity}_{\Omega'})\} \quad (4.11)$$

that the footprint of a run in the product $\Delta$ satisfies the parity condition induced by $\Omega'$. $\Delta$ can be constructed in polynomial time.
Step 2. We express (4.11) using the step chain \( \tilde{\mathcal{M}} \). Let \( M \subseteq Q \cup Q^\perp \) be the states of the step chain \( \tilde{\mathcal{M}} \). Let \( \Omega'' \colon M \to \mathbb{N}_0 \) be the extension of \( \Omega' \) to the states of \( M \) via \( \Omega''(q) = \Omega''(q^\perp) = \tilde{\Omega}'(q) \) for all \( q \in Q \). That is, \( \Omega'' \) induces a parity acceptance set \( \text{Parity}_{\Omega''} \subseteq M^\omega \) which is \( \omega \)-regular and thus measurable. Let \( \rho \in \text{Runs}_\tilde{\mathcal{M}} \). Clearly, \( \rho|_{\text{Steps}} \in \text{Parity}_{\Omega'} \) iff \( \rho|_{\text{Steps}} \in \text{Parity}_{\Omega''} \). Thus by Lemma 4.14, (4.11) is equal to
\[
\mathbb{P}(\{ \pi \in \text{Runs}_\tilde{\mathcal{M}} \mid \rho|_{\text{Steps}} \in \text{Parity}_{\Omega'} \}) = \mathbb{P}(\text{Parity}_{\Omega''})
\]
where the right hand side is the probability that a run of the step chain \( \tilde{\mathcal{M}} \) satisfies the parity condition induced by \( \Omega'' \). We have thus reduced the problem of computing a stair-parity acceptance probability in the product pPDA \( \tilde{\Delta} \) to computing a standard parity acceptance probability in the finite Markov chain \( \tilde{\mathcal{M}} \).

The rest of the proof mainly uses standard techniques and is similar to the proof of [EKM04, Thm. 5.15]. The main technical difficulty is that the transition probabilities of \( \tilde{\mathcal{M}} \) cannot be written in an explicit numerical form since they are in general algebraic numbers.

Step 3. We construct the underlying graph \( G_\tilde{\Delta} \) of the step chain \( \tilde{\mathcal{M}} \), i.e., we determine the set \( M \) of states and include a directed edge between states \( v, v' \in M \) iff the probability of transition \( v \to v' \) is greater than zero. Recall that \( q \in Q \) is in \( M \) iff \( [q^\uparrow] > 0 \). The latter can be reduced to checking the truth of an ETR formula (an existentially quantified FO-formula over \((\mathbb{R}, +, \cdot, \leq)\)) which can be decided in \text{PSPACE}. A similar observation applies to checking whether the transition probabilities in Figure 9 are positive.

Step 4. We determine the bottom strongly connected components (BSCC) of \( G_\tilde{\Delta} \) from the previous step by a standard (efficient) graph analysis. We mark the BSCCs \( B \subseteq M \) such that \( \min_{v \in B} \Omega''(v) \) is even as “good”, the others as “bad”. It is well-known that in the finite Markov chain \( \tilde{\mathcal{M}} \) it holds that
\[
\mathbb{P}(\rho \in \text{Parity}_{\Omega''} \mid \rho \text{ reaches a good BSCC}) = 1
\]
due to the Long-Run Theorem of finite Markov chains: Each state of a BSCC is visited infinitely often almost-surely, provided that this BSCC is reached at all. Moreover, if a run \( \rho \) reaches a bad BSCC, than the probability to satisfy the parity condition is zero and thus
\[
\mathbb{P}(\rho \in \text{Parity}_{\Omega''}) = \mathbb{P}(\rho \in \text{Parity}_{\Omega''} \mid \rho \text{ reaches a good BSCC}) \cdot \mathbb{P}(\rho \text{ reaches a good BSCC})
\]
Thus it only remains to compute the probability to reach a good BSCC in \( \tilde{\mathcal{M}} \).

Step 5. We use the previous step to classify the states \( M \) of the step chain \( \tilde{\mathcal{M}} \) into three categories: \( M_{=0} \) contains all states from which no good BSCC is reachable in the graph \( G_\tilde{\Delta} \), \( M_{=1} \) contains all good BSCCs, and \( M_{?} \) contains all other states. We recall from e.g. [BK08, Ch. 10] that the probabilities to reach \( M_{=1} \) are the unique solution of the linear equation system
\[
x_v = 0 \quad \text{if } v \in M_{=0} \\
x_v = 1 \quad \text{if } v \in M_{=1} \\
x_v = \sum_{v \xrightarrow{\Delta} v'} px_{v'} \quad \text{if } v \in M_{?} .
\]
We can treat the vectors of probabilities \( \vec{p} \) and the variables \( \vec{x} \) in this equation system as free variables of an ETR formula \( \mathcal{R}(\vec{p}, \vec{x}) \). By Lemma 4.17, there is an ETR formula \( \Phi(\vec{p}) \)
expressing $\vec{p}$. The ETR formula

$$\exists \vec{p}, \vec{x}: \Phi(\vec{p}) \land R(\vec{p}, \vec{x}) \land x_{v_0} \geq \theta$$

is thus true if the probability to reach a good BSCC from initial state $v_0 \in M$ is at least $\theta$. Truth of this formula can be decided in PSPACE.

4.3. Implications for Probabilistic One-counter Automata. A probabilistic visibly one-counter automaton (pVOC) is the special case of a pVPA with unary stack alphabet, i.e., $|\Gamma| = 1$. For example, the pVPA in Figure 10 (left) is a pVOC. For many problems, better complexity bounds are known for pVOC than for the general case. In particular, $|[p]| > 0$, i.e., the question whether a pVOC started in state $p$ never reaches counter value (or stack height) zero with positive probability, can be decided in $\mathbb{P}$ [BEKK13, Thm. 4]. We can exploit this to improve Theorem 4.1 in the pVOC case:

**Corollary 4.18.** Let $\Delta$ be a pVOC and $D$ be a stair-parity DVPA over pushdown alphabet $\Sigma$. The problem $\mathbb{P}\{\{\rho \in \text{Runs}_\Delta \mid \lambda(\rho) \in L(D)\} = \emptyset\}$ is decidable in $\mathbb{P}$.

**Proof.** The key observation is that, since we can efficiently decide $|[p]| > 0$, we can efficiently (in polynomial time) construct the underlying graph $G_{\Delta \times D}$ of the step chain of $\Delta \times D$ (as in the proof of Theorem 4.1), and then apply polynomial-time graph analysis algorithms to check if only good BSCCs are reachable in $G_{\Delta \times D}$. \qed

Corollary 4.18 implies that there exist efficient algorithms for many properties of pVOC-expressible random walks on $\mathbb{N}_0$. In fact, almost-sure satisfaction of each fixed visibly-pushdown property can be decided in $\mathbb{P}$. For instance, using the DVPA from Figure 5 we can decide if a random walk is repeatedly bounded with probability 1.
5. Model Checking against Büchi VPA and CaRet

With Theorems 2.1 and 4.1 it follows immediately that quantitative model checking of pVPA against non-deterministic Büchi VPA is decidable in EXPSPACE. We can improve the complexity in the qualitative case:

**Theorem 5.1.** Let \( \Delta \) be a pVPA and \( \mathcal{A} \) be a (non-deterministic) Büchi VPA over the same pushdown alphabet. The problem \( \mathbb{P}(\{ \rho \in \text{Runs}_\Delta \mid \lambda(\rho) \in \mathcal{L}(\mathcal{A}) \}) = 1 \) is EXPTIME-complete.

**Proof.** The lower bound is due to [EY05, Thm. 8] and already holds for non-pushdown Büchi automata. We now describe an EXPTIME decision procedure:

• We first determinize \( \mathcal{A} \) using Theorem 2.1 which is possible in time exponential in \( |\mathcal{A}| \). Let \( \mathcal{D} \) be the resulting stair-parity DVPA and consider the product \( \Delta \times \mathcal{D} \) (Definition 4.1). Note that the product can be constructed in polynomial time in \( |\mathcal{D}| \) and \( |\Delta| \), and thus in exponential time in the overall size of the input.

• The crucial observation for the next step is that \( [q] = [(q, s)] \) for all states \( (q, s) \) of \( \Delta \times \mathcal{D} \). This holds because by definition of the product, \( \mathcal{D} \) merely observes the runs of \( \Delta \), and thus the diverge probabilities of \( \Delta \times \mathcal{D} \) and \( \Delta \) are essentially the same. We compute the set \( \text{Div}_\Delta = \{ q \mid [q] > 0 \} \subseteq Q \), \( Q \) the states of \( \Delta \), which can be done in exponential time in \( |\Delta| \) using an efficient (i.e. PSPACE) decision procedure for the ETR [EKM04]. Note that computing \( \text{Div}_{\Delta \times \mathcal{D}} = \{ (q, s) \mid [(q, s)] > 0 \} \) directly would take doubly-exponential time in \( |\mathcal{A}| \); the proposed “optimization” is thus essential for obtaining the EXPTIME upper bound.

• We now determine the set of triples \( \text{Ret}_{\Delta \times \mathcal{D}} = \{ (q, Z, p) \mid [qZ ] > 0 \} \) in \( \Delta \times \mathcal{D} \). Unlike the diverge probabilities, this set can be computed in polynomial time in the size of \( \Delta \times \mathcal{D} \) (hence exponential in the size of the input) because we may disregard the exact transition probabilities and conduct a standard reachability analysis in a non-deterministic pushdown automaton [ABE18], also see [BEKK13, p. 136].

• The next step is to construct the underlying graph \( G_{\Delta \times \mathcal{D}} \) of the step chain \( \mathcal{M}_{\Delta \times \mathcal{D}} \), i.e., the directed graph that has the same vertices as \( \mathcal{M}_{\Delta \times \mathcal{D}} \) and includes an edge \( (u, v) \) iff the 1-step transition probability from \( u \) to \( v \) is positive in the Markov chain. This can be done in polynomial time in \( |\Delta \times \mathcal{D}| \) using the sets \( \text{Div}_\Delta \) and \( \text{Ret}_{\Delta \times \mathcal{D}} \) defined above and Figure 12.

• The final step is, as in Theorem 4.1, to determine the BSCCs of \( G_{\Delta \times \mathcal{D}} \), classify them as good or bad according to whether they satisfy the (standard) parity condition inherited from \( \mathcal{D} \), and then check if there is a bad BSCC reachable from the initial state. All these steps can be done in polynomial time in \( |\Delta \times \mathcal{D}| \).

In the above result, membership in 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 \( \Delta \) but polynomial in the size of \( \mathcal{D} \). EXPTIME-hardness follows from [EY05, Thm. 8]. In fact, qualitative model checking of pPDA against non-pushdown Büchi automata is also EXPTIME-complete [EY05]. With Theorems 2.1, 2.2, 4.1 and 5.1 we immediately obtain the following complexity results for CaRet model checking:

**Theorem 5.2.** The quantitative and qualitative probabilistic CaRet model checking problems (Definition 3.5) are decidable in 2EXPSPACE and 2EXPTIME, respectively.
Both problems are known to be EXPTIME-hard [YE05].

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 \( \omega \)-VPL. We heavily rely on the determinization procedure from [LMS04] and the notion of a step chain used in previous works [EKM04, KEM06]. These two constructions turn out to be a natural match.

We conjecture that our upper complexity bounds are not tight 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 [YE05]. Other topics are to explore to what extent algorithms for probabilistic CTL can be generalized to the branching-time variant of CaReT [GMN18], to consider more expressive logics such as visibly LTL [BS18] or OPTL [CMP20], and to study the interplay of conditioning and recursion [SG12] through the lens of pPDA.

Acknowledgement. The authors thank Christof Löding for his pointer to stair-parity VPA.

References

[AAB+07] Rajeev Alur, Marcelo Arenas, Pablo Barceló, Kousha Etessami, Neil Immerman, and Leonid Libkin. 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, pages 151–160. IEEE Computer Society, 2007.

[ABE18] Rajeev Alur, Ahmed Bouajjani, and Javier Esparza. Model Checking Procedural Programs. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, Handbook of Model Checking, pages 541–572. Springer, 2018.

[AEM04] Rajeev Alur, Kousha Etessami, and P. Madhusudan. 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, volume 2988 of Lecture Notes in Computer Science, pages 467–481. Springer, 2004.

[AM04] Rajeev Alur and P. Madhusudan. Visibly Pushdown Languages. In Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004, pages 202–211. ACM, 2004.
[AP09] Philippe Audebaud and Christine Paulin-Mohring. Proofs of randomized algorithms in Coq. Sci. Comput. Program., 74(8):568–589, 2009.

[BEKK13] Tomás Brázdil, Javier Esparza, Stefan Kiefer, and Antonín Kucera. Analyzing probabilistic pushdown automata. Formal Methods Syst. Des., 43(2):124–163, 2013.

[BK08] Christel Baier and Joost-Pieter Katoen. Principles of Model Checking. MIT Press, 2008.

[BKOB13] Gilles Barthe, Boris Köpf, Federico Olmedo, and Santiago Zanella Béguelin. Probabilistic Relational Reasoning for Differential Privacy. ACM Trans. Program. Lang. Syst., 35(3):9:1–9:49, 2013.

[BKS05] Tomás Brázdil, Antonín Kucera, and Oldřich Stražovský. 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, volume 3404 of Lecture Notes in Computer Science, pages 145–157. Springer, 2005.

[BS18] Laura Bozzelli and César Sánchez. Visibly Linear Temporal Logic. J. Autom. Reason., 60(2):177–220, 2018.

[CIRW08] Lorenzo Casini, Phyllis Mckay Illari, Federica Russo, and Jon Williamson. Recursive Bayesian Networks. Theoria. Revista de Teoria, Historia y Fundamentos de la Ciencia, 26(1):5–33, 2008.

[CMP20] Michele Chiari, Dino Mandrioli, and Matteo Pradella. Operator precedence temporal logic and model checking. Theor. Comput. Sci., 848:47–81, 2020.

[DBB12] Clemens Dubslaff, Christel Baier, and Manuela Berg. Model checking probabilistic systems against pushdown specifications. Inf. Process. Lett., 112(8-9):320–328, 2012.

[EKM04] Javier Esparza, Antonín Kucera, and Richard Mayr. Model Checking Probabilistic Pushdown Automata. In 19th IEEE Symposium on Logic in Computer Science (LICS 2004), 14-17 July 2004, Turku, Finland, Proceedings, pages 12–21. IEEE Computer Society, 2004.

[EY05] Kousha Etessami and Mihalis Yannakakis. 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, volume 3440 of Lecture Notes in Computer Science, pages 253–270. Springer, 2005.

[EY09] Kousha Etessami and Mihalis Yannakakis. Recursive markov chains, stochastic grammars, and monotone systems of nonlinear equations. J. ACM, 56(1):1:1–1:66, 2009.

[GHNR14] Andrew D. Gordon, Thomas A. Henzinger, Aditya V. Nori, and Sriram K. Rajamani. Probabilistic programming. In Proceedings of the on Future of Software Engineering, FOSE 2014, Hyderabad, India, May 31 - June 7, 2014, pages 167–181. ACM, 2014.

[GMN18] Jens Oliver Gutsfeld, Markus Müller-Olm, and Benedikt Nordhoff. A Branching Time Variant of CaRet. In Model Checking Software - 25th International Symposium, SPIN 2018, Malaga, Spain, June 20-22, 2018, Proceedings, volume 10869 of Lecture Notes in Computer Science, pages 153–170. Springer, 2018.

[Jae01] Manfred Jaeger. Complex Probabilistic Modeling with Recursive Relational Bayesian Networks. Ann. Math. Artif. Intell., 32(1-4):179–220, 2001.

[Jon90] Claire Jones. Probabilistic non-determinism. PhD thesis, University of Edinburgh, UK, 1990.

[KEM06] Antonín Kucera, Javier Esparza, and Richard Mayr. Model Checking Probabilistic Pushdown Automata. Log. Methods Comput. Sci., 2(1), 2006.

[LMS04] Christof Löding, P. Madhusudan, and Olivier Serre. Visibly Pushdown Games. In FSTTCS 2004: Foundations of Software Technology and Theoretical Computer Science, 24th International Conference, Chennai, India, December 16-18, 2004, Proceedings, volume 3328 of Lecture Notes in Computer Science, pages 408–420. Springer, 2004.

[MM01] Annabelle McIver and Carroll Morgan. Partial correctness for probabilistic demonic programs. Theor. Comput. Sci., 266(1-2):513–541, 2001.

[OKKM16] Federico Olmedo, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Math-eja. 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, pages 672–681. ACM, 2016.

[PK00] Avi Pfeffer and Daphne Koller. Semantics and Inference for Recursive Probability Models. In Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth
Conference on Innovative Applications of Artificial Intelligence, July 30 - August 3, 2000, Austin, Texas, USA, pages 538–544. AAAI Press / The MIT Press, 2000.

[Pnu77] Amir Pnueli. The temporal logic of programs. In 18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977, pages 46–57. IEEE Computer Society, 1977.

[SG12] Andreas Stuhlmueller and Noah D. Goodman. A Dynamic Programming Algorithm for Inference in Recursive Probabilistic Programs. CoRR, abs/1206.3555, 2012.

[vdMPYW18] Jan-Willem van de Meent, Brooks Paige, Hongseok Yang, and Frank Wood. An Introduction to Probabilistic Programming. CoRR, abs/1809.10756, 2018.

[WE07] Dominik Wojtczak and Kousha Etessami. 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, volume 4424 of Lecture Notes in Computer Science, pages 66–71. Springer, 2007.

[WGK21] Tobias Winkler, Christina Gehnen, and Joost-Pieter Katoen. Model Checking Temporal Properties of Recursive Probabilistic Programs. CoRR, abs/2111.03501, 2021.

[WGK22] Tobias Winkler, Christina Gehnen, and Joost-Pieter Katoen. Model checking temporal properties of recursive probabilistic programs. In Patricia Bouyer and Lutz Schröder, editors, Foundations of Software Science and Computation Structures - 25th International Conference, FOSSACS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, volume 13242 of Lecture Notes in Computer Science, pages 449–469. Springer, 2022.

[YE05] Mihalis Yannakakis and Kousha Etessami. Checking LTL Properties of Recursive Markov Chains. In Second International Conference on the Quantitative Evaluation of Systems (QEST 2005), 19-22 September 2005, Torino, Italy, pages 155–165. IEEE Computer Society, 2005.