Acta Informatica manuscript No.
(will be inserted by the editor)

On the Hardness of Analyzing Probabilistic Programs

Benjamin Lucien Kaminski · Joost-Pieter Katoen · Christoph Matheja

Received: September 15 2016 / Accepted: date

Abstract We study the hardness of deciding probabilistic termination as well as the hardness of approximating expected values (e.g. of program variables) and (co)variances for probabilistic programs.

Termination: We distinguish two notions of probabilistic termination: Given a program P and an input σ...

1. ... does P terminate with probability 1 on input σ? (*almost–sure termination*)
2. ... is the expected time until P terminates on input σ finite? (*positive almost–sure termination*)

For both of these notions, we also consider their universal variant, i.e. given a program P, does P terminate on *all* inputs?

We show that deciding almost–sure termination as well as deciding its universal variant is Π^0_2–complete in the arithmetical hierarchy. Deciding positive almost–sure termination is shown to be Σ^0_2–complete, whereas its universal variant is Π^0_3–complete.

Expected values: Given a probabilistic program P and a random variable f mapping program states to rationals, we show that computing lower and upper bounds on the expected value of f after executing P is Σ^0_1– and Σ^0_2–complete,

This research is funded by the Excellence Initiative of the German federal and state governments, by the RTG 2236 UnRAVeL, and by the CDZ project CAP (GZ 1023).

B.L. Kaminski
Tel.: +49 (0)241 80 21208
E-mail: benjamin.kaminski@cs.rwth-aachen.de

J.-P. Katoen
Tel.: +49 (0)241 80 21200
E-mail: katoen@cs.rwth-aachen.de

C. Matheja
Tel.: +49 (0)241 80 21210
E-mail: matheja@cs.rwth-aachen.de

Software Modeling and Verification Group
RWTH Aachen University
52056 Aachen, Germany

respectively. Deciding whether the expected value equals a given rational value is shown to be Π^0_2–complete.

Covariances: We show that computing upper and lower bounds on the covariance of two random variables is both Σ^0_2–complete. Deciding whether the covariance equals a given rational value is shown to be in Δ^0_3. In addition, this problem is shown to be Σ^0_2–hard as well as Π^0_2–hard and thus a "proper" Δ^0_3–problem. All hardness results on covariances apply to variances as well.

Keywords probabilistic programs · expected value · weakest preexpectation · almost–sure termination · positive almost–sure termination · computational hardness · covariance

1 Introduction

Probabilistic programs [28] are imperative programs with the ability to toss a (possibly biased) coin and proceed their execution depending on the outcome of the coin toss. Whereas an ordinary program maps inputs to outputs, a probabilistic program maps inputs to a (posterior) distribution over outputs. Probabilistic programs are used to describe randomized algorithms and Bayesian networks [17]. Other application areas include, amongst others, machine learning, systems biology, security [3], planning and control, quantum computing [44], and software–defined networks [15]. This paper focuses on the computational hardness of the following program analyses: two notions of probabilistic termination, approximating preexpectations [29,31], and determining covariances on preexpectations.

Program Termination. Probabilistic programs are normal–looking programs, but reasoning about their correctness is intricate. The key property of program termination exemplifies this. Whereas a classical program either *certainly* terminates on a given input or not, this is no longer true for probabilistic programs. The program

```
i := 0;
c := 1;
while (c ≠ 0){
    i := i + 1;
    {c := 0} [1/3] {c := 1}        // flip a biased coin[1]
}
```

computes a geometric distribution with parameter $1/3$, as the probability that i equals $N > 0$ on termination is $(2/3)^{(N-1)} \cdot 1/3$. This program does not always terminate as it admits an infinite run in which the variable c stays one forever. The probability of this single non–terminating run, however, is zero. The program thus does not terminate certainly (meaning that every single of its runs terminates), but it terminates with probability one. This is referred to as *almost–sure* termination [41].

If a classical, deterministic program terminates, it reaches a unique final state in finitely many steps. This is not always true for probabilistic programs: It is possible to reach a final state with different probabilities and with different numbers of steps. For probabilistic programs, we are thus interested in the expected (or

[1] The left branch is executed with probability $1/3$ and the right branch with probability $2/3$.

average) number of steps required until termination. As the expected value of a geometric distribution is finite, the above program terminates almost surely and needs—on average—finitely many steps to do so.

Probabilistic programs that terminate in expected finitely many steps are called *positively almost–surely* terminating—a terminology introduced by Bournez and Garnier [5]. Their inspiration for the name "positive" comes from Markov chain theory, more specifically from the distinction between *positively recurrent* states (a state is revisited with probability one and the expected time until a revisit is finite) and *null recurrent* states (a state is revisited with probability one but the expected time to revisit is infinite) [39, Section A.2, p. 588].

The difference between almost-sure termination and positive almost-sure termination is highlighted by the following program:

```
x := 10;
while (x > 0){
    {x := x - 1 } [1/2] {x := x + 1}        // flip a fair coin
}
```

This program describes a left–bounded one–dimensional random walk where the particle starts at position 10. With probability $1/2$, the variable x is decremented by one (the particle moves to the left); with the same likelihood x is incremented by one (the particle moves to the right). This program terminates almost–surely [41], but on average requires infinitely many computation steps until it terminates [23, Section 7.1]. It is thus not positively almost–surely terminating.

These two sample programs show that there are several nuances when considering termination of probabilistic programs. Reasoning about positive almost–sure termination is not trivial. For instance, positive almost–sure termination is not preserved under sequential composition. Consider program P, given by

```
x := 1;
c := 1;
while (c ≠ 0){
    {c := 0 } [1/2] {c := 1};        // flip a fair coin
    x := 2 · x
}
```

which is positively almost–surely terminating for the same reason as the geometric distribution program. The simple program Q:

```
while (x > 0){
    x := x - 1
}
```

terminates certainly on any input, and does so in a finite number of steps. The program $P;Q$ however is not positively almost–surely terminating as it requires on average an infinite number of steps until termination. This is due to the fact that the variable x grows exponentially in program P, whereas P's termination probability decays exponentially and program Q requires k steps when x equals k.

Computational Hardness of Program Termination. Almost–sure termination is an active field of research [14,9] with seminal papers by Hart, Sharir, and Pnueli [19, 41] going back to 1983. A lot of work has been done towards automated reasoning

for almost–sure termination. For instance, [42] gives an overview of some particularly interesting examples of probabilistic logic programs and the according intuition for proving almost–sure termination. Arons *et al.* [1] reduce almost–sure termination to termination of nondeterministic programs by means of a planner. This idea has been further exploited and refined into a pattern–based approach with prototypical tool support [13]. Chakarov and Sankaranarayanan use ranking supermartingales to reason about positive almost–sure termination [7]. A related approach for reasoning about expected runtimes is presented in [24]. A technique for synthesizing ranking supermartingales is presented by Chatterjee *et al.* [8]. Dal Lago and Grellois have presented a completely different approach for proving almost–sure termination by means of a type system [30].

Despite the existence of several (sometimes automated) approaches to tackle almost–sure termination, most authors claim that it must intuitively more difficult than the termination problem for ordinary programs. Esparza *et al.* [13], for instance, claim that almost–sure termination is more involved to decide than ordinary termination since for the latter a topological argument suffices while for the former arithmetical reasoning is needed. Hence, one cannot use standard termination provers that have been developed for ordinary programs.

There are some results on computational hardness in connection with probabilistic programs, like non–recursive–enumerability results for probabilistic rewriting logic [6] and decidability results for restricted probabilistic programming languages [34]. The computational hardness of almost–sure termination has, however, received scant attention. As a notable exception, [43] establishes that deciding almost–sure termination of certain concurrent probabilistic programs is in Π^0_2; the second level of the arithmetical hierarchy.

In this paper, we study the computational hardness of almost–sure termination and positive almost–sure termination. We obtain that deciding almost–sure termination of probabilistic program P on a given input σ is Π^0_2–complete. While for ordinary programs we have a complexity leap when moving from the non–universal to the universal halting problem, we establish that *this is not the case for probabilistic programs*: Deciding universal almost–sure termination turns out to be Π^0_2–complete too. Our hardness results are established by reductions from the universal halting problem for ordinary, i.e. non–probabilistic, programs. The case for *positive* almost–sure termination is different, however: While deciding (non–universal) positive almost–sure termination is Σ^0_2–complete, we show that universal positive almost–sure termination is Π^0_3–complete.

Preexpectations. Establishing the correctness of probabilistic programs needs—due to their intricate behavior even more so than ordinary programs—formal reasoning. Weakest–precondition calculi à la Dijkstra [12] provide an important apparatus to enable formal reasoning. To develop such calculi, one takes into account that due to its random nature, the final state of a program on termination is not unique. Thus, rather than a mapping from inputs to outputs (as in Dijkstra's approach), probabilistic programs map an initial state to a *distribution* on possible final states. More precisely, we have *sub*–distributions where the "missing" probability mass represents the likelihood of divergence. Given a random variable f and an initial distribution, a key issue now is to determine f's expected value on the probabilistic program's termination. Kozen's seminal work on a probabilistic variant of propositional dynamic logic [29] focuses on this. McIver and Morgan [31]

extended Kozen's approach with (amongst others) nondeterminism. In their jargon, f is called an expectation and — à la Dijkstra — pre- and postexpectations are the quantitative counterparts of pre- and postconditions. To illustrate preexpectations, consider the program:

```
{x := 0 } [1/2] {x := 1};        // flip a fair coin
{y := 0 } [1/3] {y := 1};        // flip a biased coin
```

that flips a pair of coins, one being fair and one being biased. The preexpectation of the random variable $x + y$ (i.e. the expected value of $x + y$ after termination of the above program) equals $1/2 \cdot 0 + 1/2 \cdot 1 + 1/3 \cdot 0 + 2/3 \cdot 1 = 7/6$. We can also capture probabilities of events, using indicator functions. For instance, we denote by $[x + y = 0]$ the indicator function of the event that x plus y equals 0. The preexpectation of the random variable $[x + y = 0]$ equals $1/6$, i.e. the probability that after termination x plus y equals 0 is $1/6$. As another example, the preexpectation of $[x = y]$ equals $1/2$. As with ordinary programs, the difficult part of the program analysis is to deal with loops; for example by finding appropriate loop invariants [25].

Computational Hardness of Computing Preexpectations and Covariances. This paper deals with establishing the computational hardness of approximating preexpectations of probabilistic programs. Morgan [33] remarks that while partial correctness for small–scale examples is not harder to prove than for ordinary programs, the case for total correctness of a probabilistic loop must be harder to analyze. This paper gives precise classifications of the level of arithmetical reasoning that is needed to compute preexpectations by establishing the following results: We first show that computing lower bounds on the expected value of a random variable f after executing a probabilistic program P on a given input σ is Σ^0_1–complete and therefore arbitrarily close approximations from below are computably enumerable. Computing upper bounds, on the other hand, is shown to be Σ^0_2–complete, thus arbitrarily close approximations from above are not computably enumerable in general. We also show that deciding whether the expected value is at all finite is Σ^0_2–complete. Deciding whether an expected value equals some rational is shown to be Π^0_2–complete.

We complement these results by also considering obtaining bounds on (co)-variances of probabilistic programs. We show that obtaining bounds on (co)variances is computationally more difficult than for expected values. In particular, we prove that computing upper *and lower* bounds for (co)variances of preexpectations is both Σ^0_2–complete, thus *not computably enumerable*. We also show that determining the precise values of (co)variances is in Δ^0_3 and both Σ^0_2– and Π^0_2–hard. The covariance problem is thus a problem that lies "properly" in Δ^0_3. These results rule out analysis techniques based on finite loop–unrollings as complete approaches for reasoning about covariances.

Organization of the Paper. Section 2 introduces probabilistic programs and computational and arithmetical complexity. Section 3 formally defines the various decision problems for preexpectations and determines their hardness. We treat preexpectations before termination, since our hardness results on termination make use of the results on preexpectations. Section 4 defines (positive) almost–sure termination and presents detailed proofs of their computational hardness. Our hardness

results on probabilistic termination are then applied in Section 5 to study the hardness of determining covariances. Section 6 concludes the paper. This paper is based on the conference papers [21] and [22], provides a unified treatment of these results, sharpens some of the results, and presents detailed proofs.

2 Preliminaries

2.1 Syntax and Semantics of Probabilistic Programs

Our development builds upon a simple programming language called *probabilistic guarded command language (pGCL)* [28,31]—a variant of Dijkstra's guarded command language [12] endowed with probabilistic choice constructs. Its syntax is given as follows:

Definition 1 (Syntax of pGCL [28,31]) Let Var be the (countable) **set of program variables**. The **set Prog of pGCL programs** adheres to the grammar

$$\begin{aligned}\mathsf{Prog} \quad \longrightarrow \quad & \texttt{skip} \mid \texttt{diverge} \mid v \coloneqq e \mid \mathsf{Prog}\texttt{;}\ \mathsf{Prog} \mid \texttt{if}\ (b)\ \{\mathsf{Prog}\}\ \texttt{else}\ \{\mathsf{Prog}\} \\ & \mid \texttt{while}\ (b)\ \{\mathsf{Prog}\} \mid \{\mathsf{Prog}\}\ [p]\ \{\mathsf{Prog}\}\ ,\end{aligned}$$

where $v \in \mathsf{Var}$, e is an arithmetical expression over Var, $p \in [0,1] \cap \mathbb{Q}$, and b is a Boolean expression over arithmetical expressions over Var. We call the set of programs that *do not* contain any probabilistic choices the **set of ordinary programs** and denote this set by $\mathsf{ordProg}$. △

Let us shortly go over the language constructs: `skip` does nothing. `diverge` is a program that certainly enters an infinite loop.[2] $v \coloneqq e$ assigns the value of expression e evaluated in the current variable valuation to variable v. $P_1\texttt{;}\ P_2$ is the sequential composition of P_1 and P_2. $\texttt{if}\ (b)\ \{P_1\}\ \texttt{else}\ \{P_2\}$ is a conditional construct guarded by a Boolean expression b. $\texttt{while}\ (b)\ \{P'\}$ is a loop construct also guarded by a Boolean expression b. $\{P_1\}\ [p]\ \{P_2\}$ denotes a probabilistic choice between the program P_1 (which is executed with probability p) and the program P_2 (which is executed with probability $1-p$). A simple operational semantics for pGCL is given in the following. We first define what a program configuration is.

Definition 2 (Program States and Configurations) Let

$$\mathbb{S} = \{\sigma \mid \sigma\colon V \to \mathbb{Q},\ V \subset \mathsf{Var},\ V \text{ finite}\}\ .$$

be the **set of program states**. Notice that $\mathbb{S}$ is countable since the domain V of any state $\sigma\colon V \to \mathbb{Q}$ is finite. A state σ is considered a **valid input** for a given program P, iff all program variables that occur in P are in the domain of σ. Notice that the domain of σ may contain more variables than occur in P. For each program $P \in \mathsf{Prog}$, let

$$\mathbb{S}_P = \{\sigma \in \mathbb{S} \mid \sigma \text{ is a valid input for } P\}$$

denote the set of valid inputs for P.[3]

[2] i.e. `diverge` is syntactic sugar for $\texttt{while}\ (\texttt{true})\ \{\texttt{skip}\}$.

[3] The notion of valid inputs is needed due to our restriction that program states have finite domains. If we drop this restriction, the set of all program states becomes uncountable. Moreover, note that it is clearly decidable whether a program state is valid for a given program.

For $\sigma : V \to \mathbb{Q}$, we denote by $\sigma[x \mapsto v]$ the state

$$\sigma[x \mapsto v]\colon \quad V \cup \{x\} \to \mathbb{Q},\ y \mapsto \begin{cases} \sigma(y), & \text{if } x \neq y \\ v, & \text{if } x = y \end{cases},$$

i.e. we enlarge the domain of σ if necessary and let variable x evaluate to value v.

Let $\mathsf{Prog}_\downarrow = \mathsf{Prog} \cup \{\downarrow\} \cup \{\downarrow;\ P \mid P \in \mathsf{Prog}\}$. For $P \in \mathsf{Prog}_\downarrow$, we define $\mathbb{S}_P$ analogously to the case for $P \in \mathsf{Prog}$. The **set of program configurations** (or simply **configurations**) is given by

$$\mathbb{K} \ = \ \{\langle P, \sigma, a, \Theta\rangle \in \mathsf{Prog}_\downarrow \times \mathbb{S} \times \big([0,1] \cap \mathbb{Q}\big) \times \{L, R\}^* \mid \sigma \in \mathbb{S}_P\}\ .$$

Notice that the configurations in $\mathbb{K}$ couple only those states σ with programs P, where σ is a valid input for P. Notice furthermore, that $\mathbb{K}$ is countable since it is a cartesian product of countable sets. The last two components of a configuration are used for bookkeeping of probabilistic choices as explained later. A configuration is a **terminal configuration** if it is of the form $\langle\downarrow, \sigma, a, \Theta\rangle$. For a program $P \in \mathsf{Prog}$ and an initial state $\sigma \in \mathbb{S}_P$, we denote by $\gamma_{P,\sigma}$ the **initial configuration** $\langle P, \sigma, 1, \varepsilon\rangle$. △

Using the notion of configurations, we can now define our operational semantics in terms of a configuration transition relation:

Definition 3 (Operational Semantics of pGCL) Let $[\![e]\!]_\sigma \in \mathbb{Q}$ and $[\![b]\!]_\sigma \in \{\mathsf{true}, \mathsf{false}\}$ be the evaluation of an arithmetical expression e and a Boolean expression b in state σ, respectively. Then the **semantics of pGCL** is given by the least relation $\vdash \subseteq \mathbb{K} \times \mathbb{K}$ satisfying the following inference rules:

$$\text{(skip)}\ \frac{}{\langle \texttt{skip}, \sigma, a, \theta\rangle \ \vdash\ \langle\downarrow, \sigma, a, \theta\rangle}$$

$$\text{(diverge)}\ \frac{}{\langle \texttt{diverge}, \sigma, a, \theta\rangle \ \vdash\ \langle \texttt{diverge}, \sigma, a, \theta\rangle}$$

$$\text{(assign)}\ \frac{}{\langle v\ \texttt{:=}\ e, \sigma, a, \theta\rangle \ \vdash\ \langle\downarrow, \sigma[v \mapsto [\![e]\!]_\sigma], a, \theta\rangle}$$

$$\text{(seq-1)}\ \frac{\langle P_1, \sigma, a, \theta\rangle \ \vdash\ \langle P_1', \sigma', a', \theta'\rangle}{\langle P_1\,\texttt{;}\ P_2, \sigma, a, \theta\rangle \ \vdash\ \langle P_1'\,\texttt{;}\ P_2, \sigma', a', \theta'\rangle}$$

$$\text{(seq-2)}\ \frac{}{\langle\downarrow\texttt{;}\ P_2, \sigma, a, \theta\rangle \ \vdash\ \langle P_2, \sigma, a, \theta\rangle}$$

$$\text{(if-t)}\ \frac{[\![b]\!]_\sigma = \mathsf{true}}{\langle \texttt{if}\ (b)\ \{P_1\}\ \texttt{else}\ \{P_2\}, \sigma, a, \theta\rangle \ \vdash\ \langle P_1, \sigma, a, \theta\rangle}$$

$$\text{(if-f)}\ \frac{[\![b]\!]_\sigma = \mathsf{false}}{\langle \texttt{if}\ (b)\ \{P_1\}\ \texttt{else}\ \{P_2\}, \sigma, a, \theta\rangle \ \vdash\ \langle P_2, \sigma, a, \theta\rangle}$$

$$\text{(while-t)}\ \frac{[\![b]\!]_\sigma = \mathsf{true}}{\langle \texttt{while}\ (b)\ \{P'\}, \sigma, a, \theta\rangle \ \vdash\ \langle P'\,\texttt{;}\ \texttt{while}\ (b)\ \{P'\}, \sigma, a, \theta\rangle}$$

$$\text{(while-f)}\ \frac{[\![b]\!]_\sigma = \mathsf{false}}{\langle \texttt{while}\ (b)\ \{P'\}, \sigma, a, \theta\rangle \ \vdash\ \langle\downarrow, \sigma, a, \theta\rangle}$$

$$\text{(prob-L)}\ \frac{}{\left\langle \{P_1\}\,[p]\,\{P_2\},\, \sigma,\, a,\, \theta \right\rangle \;\vdash\; \langle P_1,\, \sigma,\, a \cdot p,\, \theta \cdot L \rangle}$$

$$\text{(prob-R)}\ \frac{}{\left\langle \{P_1\}\,[p]\,\{P_2\},\, \sigma,\, a,\, \theta \right\rangle \;\vdash\; \langle P_2,\, \sigma,\, a \cdot (1-p),\, \theta \cdot R \rangle}\ ,$$

where

$$\sigma[v \mapsto [\![e]\!]_\sigma]\left(v'\right) \;=\; \begin{cases} [\![e]\!]_\sigma, & \text{if } v' = v, \quad \text{and} \\ \sigma\left(v'\right), & \text{otherwise.} \end{cases}$$

We use $\gamma \vdash^k \gamma'$ in the usual sense, i.e. there exist $k-1$ configurations $\gamma_1, \ldots, \gamma_k \in \mathbb{K}$, such that $\gamma \vdash \gamma_1 \vdash \cdots \vdash \gamma_{k-1} \vdash \gamma'$. △

The semantics given by the $\vdash$–relation is mostly a straightforward operational semantics except for two features: in addition to the program that is to be executed next and the current variable valuation, each configuration also stores a sequence θ over the alphabet $\{L, R\}$ that encodes a history of the branches (Left or Right) that were chosen at the probabilistic choices as well as the probability a that all those branches were chosen. It can easily be seen that the graph that is spanned by the $\vdash$–relation is just an unfolding of the Markov decision process semantics for pGCL provided in [18]. An equivalent semantics has already been provided in Kozen's seminal work on probabilistic propositional dynamic logic (PPDL), where each pGCL construct is shown to correspond to a PPDL formula [29].

It is a well–known result due to Kleene that for any ordinary program $Q \in \mathsf{ordProg}$ and an associated configuration $\gamma = \langle Q, \sigma, a, \theta \rangle$ the k-th successor with respect to $\vdash$ is unique and computable. If, however, program P contains probabilistic choices, the k-th successor of a configuration is not necessarily unique, because at various points of the execution the program must choose a left or a right branch with some probability. However, if we resolve those choices by providing a sequence of symbols w over the alphabet $\{L, R\}$ that encodes for all probabilistic choices which occur whether the Left or the Right branch shall be chosen at a branching point, we can construct a computable function that computes a unique k-th successor with respect to sequence w. Notice that for this purpose a sequence of finite length is sufficient. We obtain the following:

Proposition 1 (The Successor Configuration Function) *Let* $\mathbb{K}_\perp = \mathbb{K} \cup \{\perp\}$. *There exists a computable total function* $\mathrm{T} \colon \mathbb{N} \times \mathbb{K}_\perp \times \{L, R\}^* \to \mathbb{K}_\perp$, *such that*

$$\mathrm{T}_0(\gamma, w) \;=\; \begin{cases} \gamma, & \text{if } w = \varepsilon, \\ \perp, & \text{otherwise,} \end{cases}$$

$$\mathrm{T}_{k+1}(\gamma, w) \;=\; \begin{cases} T_k(\gamma', w'), & \text{if } \gamma = \langle P, \sigma, a, \theta \rangle \vdash \langle P', \sigma', a', \theta \cdot b \rangle = \gamma', \\ & \quad \text{with } w = b \cdot w' \text{ and } b \in \{L, R, \varepsilon\}, \\ \perp, & \text{otherwise.} \end{cases}$$

Hence $\mathrm{T}_k(\gamma, w)$ returns a successor configuration $\gamma' \neq \perp$, if $\gamma \vdash^k \gamma'$ and the inference rules for probabilistic choices (prob-L) and (prob-R) have been applied exactly $|w|$ times to resolve all probabilistic choices in γ according to w. Otherwise $\mathrm{T}_k(\gamma, w)$ returns $\perp$. Note in particular that for both the inference of a terminal configuration $\langle \downarrow, \sigma, a, \theta \rangle$ within less than k steps as well as the inference of a terminal configuration through less or more than $|w|$ probabilistic choices, $\mathrm{T}_k(\gamma, w)$ results in $\perp$.

2.2 Computational and Arithmetical Complexity

Our hardness results will be stated in terms of levels in the arithmetical hierarchy—a notion for classifying sets according to the complexity required to define them in the language of first–order Peano arithmetic.[4] We first briefly recall this concept:

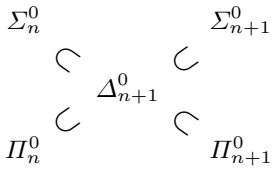
Definition 4 (Arithmetical Hierarchy [26,35]) The class Σ_n^0 is defined as[5]

$$\Sigma_n^0 \;=\; \big\{\mathcal{A} \;\big|\; \mathcal{A} = \big\{x \;\big|\; \exists y_1\, \forall y_2\, \exists y_3\, \cdots\, \exists/\forall y_n \colon\; (x,\, y_1,\, y_2,\, y_3,\, \ldots,\, y_n) \in \mathcal{R}\big\},\; \mathcal{R} \text{ is a decidable relation}\big\}\;,$$

the class Π_n^0 is defined as

$$\Pi_n^0 \;=\; \big\{\mathcal{A} \;\big|\; \mathcal{A} = \big\{x \;\big|\; \forall y_1\, \exists y_2\, \forall y_3\, \cdots\, \exists/\forall y_n \colon\; (x,\, y_1,\, y_2,\, y_3,\, \ldots,\, y_n) \in \mathcal{R}\big\},\; \mathcal{R} \text{ is a decidable relation}\big\}\;,$$

and the class Δ_n^0 is defined as $\Delta_n^0 = \Sigma_n^0 \cap \Pi_n^0$, for every $n \in \mathbb{N}$. *Multiple consecutive quantifiers of the same type* can be contracted into *one* quantifier of that type, so the number n refers to the number of necessary *quantifier alternations* rather than to the actual number of quantifiers occurring in a defining formula. A set $\mathcal{A}$ is called **arithmetical**, iff $\mathcal{A} \in \Gamma_n^0$, for some $\Gamma \in \{\Sigma, \Pi, \Delta\}$ and $n \in \mathbb{N}$. The inclusion diagram



with $\Sigma_n^0 \neq \Pi_n^0$ holds for every $n \geq 1$, thus the arithmetical sets form a strict hierarchy. Furthermore, note that $\Sigma_0^0 = \Pi_0^0 = \Delta_0^0 = \Delta_1^0$ is the class of the decidable sets and Σ_1^0 is the class of the computably enumerable sets. △

The notion of the arithmetical hierarchy is of utter utility: Besides establishing a strong connection between computability and logic, stating precisely at which level in the arithmetical hierarchy a decision problem must be placed amounts to giving a measure of just "how unsolvable" the decision problem is [10].

In order to establish results of the form *"problem $\mathcal{A}$ is at least as hard to solve as problem $\mathcal{B}$"*, we make use of two notions called *many–one reducibility* and *many–one completeness*.

Definition 5 (Many–One Reducibility and Completeness [35,38,11]) Let $\mathcal{A}$ and $\mathcal{B}$ be arithmetical sets and let X and Y be some appropriate universes such that $\mathcal{A} \subseteq X$ and $\mathcal{B} \subseteq Y$. $\mathcal{A}$ is called **many–one reducible** (or simply **reducible**) to $\mathcal{B}$, denoted

$$\mathcal{A} \;\leq_{\mathrm{m}}\; \mathcal{B}\;,$$

[4] Note that we allow the values of the quantified variables to be drawn from a computable domain other than $\mathbb{N}$ that could be encoded in the natural numbers such as $\mathbb{Q}$, the set of syntactically correct programs, etc.

[5] The last quantifier is universal if n is even and existential if n is odd.

iff there exists a computable function $f\colon X \to Y$, such that

$$\forall\, x \in X\colon \quad x \in \mathcal{A} \iff f(x) \in \mathcal{B}\ .$$

If f is a function such that f reduces $\mathcal{A}$ to $\mathcal{B}$, we denote this by $f\colon \mathcal{A} \leq_{\mathrm{m}} \mathcal{B}$. Note that $\leq_{\mathrm{m}}$ is transitive.

Given $\Gamma \in \{\Sigma, \Pi, \Delta\}$, a set $\mathcal{A}$ is called **many–one complete for $\boldsymbol{\Gamma^0_n}$** (or simply **$\boldsymbol{\Gamma^0_n}$–complete**) iff both $\mathcal{A}$ is a member of Γ^0_n and $\mathcal{A}$ is **$\boldsymbol{\Gamma^0_n}$–hard**, meaning $\mathcal{C} \leq_{\mathrm{m}} \mathcal{A}$, for any set $\mathcal{C} \in \Gamma^0_n$. Note that if $\mathcal{A}$ is Γ^0_n–complete and $\mathcal{A} \leq_{\mathrm{m}} \mathcal{B}$, then $\mathcal{B}$ is necessarily Γ^0_n–hard. Furthermore, note that if $\mathcal{A}$ is Σ^0_n–complete, then $\mathcal{A} \in \Sigma^0_n \setminus \Pi^0_n$. Analogously if $\mathcal{A}$ is Π^0_n–complete, then $\mathcal{A} \in \Pi^0_n \setminus \Sigma^0_n$. △

Many well–known and natural problems are complete for some level of the arithmetic hierarchy. Arguably one of the most prominent problems is the halting problem for ordinary programs:

Theorem 1 (The Halting Problem [36]) *The **halting problem** is the problem whether an ordinary program terminates on a given valid input. The according problem set $\mathcal{H} \subset \mathsf{ordProg} \times \mathbb{S}$ is defined as*

$$(P, \sigma) \in \mathcal{H} \quad \text{iff} \quad \sigma \in \mathbb{S}_P \text{ and } \exists\, k \in \mathbb{N}\ \exists\, \sigma' \in \mathbb{S}\colon \mathrm{T}_k(\gamma_{P,\sigma}, \varepsilon) = \langle \downarrow, \sigma', 1, \varepsilon \rangle\ .$$

*The **complement of the halting problem** is the problem whether a program does* not *terminate on a given valid input. It is given by*

$$(P, \sigma) \in \overline{\mathcal{H}} \quad \text{iff} \quad \sigma \in \mathbb{S}_P \text{ and } (P, \sigma) \notin \mathcal{H}\ .$$

$\mathcal{H}$ is Σ^0_1–complete and $\overline{\mathcal{H}}$ is Π^0_1–complete.

The halting problem is the problem of whether a given program terminates on a given valid input. Its *universal* version is the problem of whether a given program terminates on all valid inputs.

Theorem 2 (The Universal Halting Problem [36]) *The **universal halting problem** is the problem whether an ordinary program terminates on all possible valid inputs. The according problem set $\mathcal{UH} \subset \mathsf{ordProg}$ is defined as*

$$P \in \mathcal{UH} \quad \text{iff} \quad \forall\, \sigma \in \mathbb{S}_P\colon (P, \sigma) \in \mathcal{H}\ .$$

*The **complement of** $\mathcal{UH}$ is given by $\overline{\mathcal{UH}} = \mathsf{ordProg} \setminus \mathcal{UH}$.*

$\mathcal{UH}$ is Π^0_2–complete and $\overline{\mathcal{UH}}$ is Σ^0_2–complete.

We observe that we have a complexity jump from $\mathcal{H}$ to $\mathcal{UH}$, namely from Σ^0_1 to Π^0_2, i.e. a jump one level up and to the "other side" of the hierarchy.

Since we go as high as the third level of the arithmetical hierarchy, we introduce a third Σ^0_3–complete problem: the problem of whether the set of valid inputs on which a given ordinary program does not terminate is finite.

Theorem 3 (The Cofiniteness Problem [36]) *The **cofiniteness problem** is the problem of deciding whether the set of valid inputs on which an ordinary program P terminates is cofinite.*[6] *The according problem set $\mathcal{COF} \subset \mathsf{ordProg}$ is given by*

$$P \in \mathcal{COF} \quad \text{iff} \quad \{\sigma \in \mathbb{S}_P \mid (P, \sigma) \in \mathcal{H}\} \text{ is cofinite.}^{7}$$

[6] In this context, a set is cofinite iff its *relative complement*, i.e. its complement with respect to some appropriate universe, is finite.

Let $\overline{\mathcal{COF}} = \mathsf{ordProg} \setminus \mathcal{COF}$ *denote the* ***complement of*** $\mathcal{COF}$*.*
$\mathcal{COF}$ *is* Σ_3^0*–complete and* $\overline{\mathcal{COF}}$ *is* Π_3^0*–complete.*

3 The Hardness of Approximating Preexpectations

In this section we investigate the computational hardness of approximating expected values of random variables assigning numbers to program states. For such random variables we want to know the expected value after program execution. We work with a certain class of random variables commonly called expectations [31]:

Definition 6 (Expectations [31, 20]) The **set of expectations** is defined as

$$\mathbb{E} \;=\; \{f \mid f\colon \mathbb{S} \to \mathbb{Q}_{\geq 0},\ f \text{ computable}\}\ .$$

Notice that $\mathbb{E}$ is countable as there are only countably many computable functions. △

Given a probabilistic program P, initial state σ, and expectation f, we would like to answer the question:

"What is the *expected value of* f after termination of P on input σ?"

For this problem, f is referred to as a *postexpectation* and the aforementioned expected value of f is referred to as the *preexpectation*[8]. We have restricted to *computable* postexpectations since this (a) makes the set $\mathbb{E}$ countable and (b) there is no hope of determining the value of a non–computable postexpectation in a final state.

In order to express preexpectations in our formalism, we need—in addition to the successor configuration function T—another computable operation:

Proposition 2 *There exists a total computable function* $\wp\colon \mathbb{K}_\perp \times \mathbb{E} \to \mathbb{Q}_{\geq 0}$*, with*

$$\wp(\gamma,\, f) \;=\; \begin{cases} f(\sigma) \cdot a, & \text{if } \gamma = \langle\downarrow,\, \sigma,\, a,\, \_\rangle, \\ 0, & \text{otherwise,} \end{cases}$$

where _ *represents an arbitrary value in* $\{L,\, R\}^*$*.*

$\wp$ takes a configuration γ and an expectation f and returns the probability of reaching γ multiplied with the value of f in the configuration γ (this is where computability of f is needed). It does so *only if* the provided configuration γ is a *terminal configuration*. Otherwise it returns 0.

We can now define preexpectations using T and $\wp$ as follows:

Definition 7 (Preexpectations) Let $f \in \mathbb{E}$ and $A^{\leq k} = \bigcup_{i=0}^{k} A^i$ for a finite alphabet A. Then the **preexpectation of P with respect to postexpectation f in initial state $\sigma \in \mathbb{S}_P$** is given by

$$\mathrm{E}_{P,\sigma}\,(f) \;=\; \sum_{k=0}^{\infty} \sum_{w \in \{L,\, R\}^{\leq k}} \wp\big(\mathrm{T}_k(\gamma_{P,\sigma},\, w),\, f\big)\ .$$

[7] i.e. iff $\mathbb{S}_P \setminus \{\sigma \in \mathbb{S}_P \mid (P,\, \sigma) \in \mathcal{H}\}$ is finite.
[8] This is because we ask for an *a–priori* expected value with respect to an *initial state*.

For later use, we also define the notation $\mathrm{E}^k_{P,\sigma}(f)$ for the k-th summand of the outer sum of $\mathrm{E}_{P,\sigma}(f)$:

$$\mathrm{E}^k_{P,\sigma}(f) \;=\; \sum_{w \in \{L,\,R\}^{\leq k}} \wp\big(\mathrm{T}_k(\gamma_{P,\sigma},\, w),\, f\big)\;. \qquad \triangle$$

The preexpectation $\mathrm{E}_{P,\sigma}(f)$ as defined here coincides with the weakest preexpectation $wp.P.f\,(\sigma)$ à la McIver and Morgan [31] for fully probabilistic programs, which in turn coincides with Kozen's [29] eventuality operation $\langle P\rangle f$ in probabilistic propositional dynamic logic. In the above definition for $\mathrm{E}_{P,\sigma}(f)$, we sum over all possible numbers of inference step lengths k and sum over all possible probabilistic–choice–resolving sequences from length 0 up to length k. Using $\wp$ we filter out the terminal configurations γ and sum up the values of $\wp(\gamma,\, f)$.

In order to investigate the complexity of approximating $\mathrm{E}_{P,\sigma}(f)$, we define three sets: $\mathcal{LEXP}$, which relates to the set of rational lower bounds of $\mathrm{E}_{P,\sigma}(f)$, $\mathcal{REXP}$, which relates to the set of rational upper bounds, and $\mathcal{EXP}$ which relates to the value of $\mathrm{E}_{P,\sigma}(f)$ itself:

Definition 8 (Approximation Problems for Preexpectations) The problem sets $\mathcal{LEXP}, \mathcal{REXP}, \mathcal{EXP} \subset \mathsf{Prog} \times \mathbb{S} \times \mathbb{E} \times \mathbb{Q}_{\geq 0}$ are defined as

$$\begin{aligned}
(P,\,\sigma,\,f,\,q) \in \mathcal{LEXP} \quad &\text{iff} \quad \sigma \in \mathbb{S}_P \text{ and } q < \mathrm{E}_{P,\sigma}(f)\;, \\
(P,\,\sigma,\,f,\,q) \in \mathcal{REXP} \quad &\text{iff} \quad \sigma \in \mathbb{S}_P \text{ and } q > \mathrm{E}_{P,\sigma}(f)\;, \quad \text{and} \\
(P,\,\sigma,\,f,\,q) \in \mathcal{EXP} \quad &\text{iff} \quad \sigma \in \mathbb{S}_P \text{ and } q = \mathrm{E}_{P,\sigma}(f)\;.
\end{aligned} \qquad \triangle$$

The computational hardness of approximating preexpectations coincides with the hardness of deciding these problem sets. The first hardness result we establish is the Σ^0_1–completeness of $\mathcal{LEXP}$. For that, we show that $\mathcal{LEXP}$ is a Σ^0_1 problem and then show by a reduction from the (non–universal) halting problem for ordinary programs that $\mathcal{LEXP}$ is Σ^0_1–hard.

Theorem 4 *$\mathcal{LEXP}$ is Σ^0_1–complete.*

Proof For proving the membership $\mathcal{LEXP} \in \Sigma^0_1$, first notice that $\sigma \in \mathbb{S}_P$ is clearly decidable and then consider the following:

$$\begin{aligned}
& (P,\,\sigma,\,f,\,q) \in \mathcal{LEXP} \\
\iff\; & \sigma \in \mathbb{S}_P \;\wedge\; q < \mathrm{E}_{P,\sigma}(f) && \text{(Definition 8)} \\
\iff\; & \sigma \in \mathbb{S}_P \;\wedge\; q < \sum_{k=0}^{\infty} \mathrm{E}^k_{P,\sigma}(f) && \text{(Definition 7)} \\
\iff\; & \sigma \in \mathbb{S}_P \;\wedge\; \exists\, y\colon\, q < \sum_{k=0}^{y} \mathrm{E}^k_{P,\sigma}(f) && \text{(all summands are positive)} \\
\implies\; & \mathcal{LEXP} \in \Sigma^0_1 && \text{(the above is a } \Sigma^0_1\text{–formula)}
\end{aligned}$$

Figure 1 (left) gives an intuition on the resulting Σ^0_1–formula: With increasing maximum computation length y more and more probability mass of the expected value can be accumulated until eventually a probability mass strictly larger than q has been accumulated.

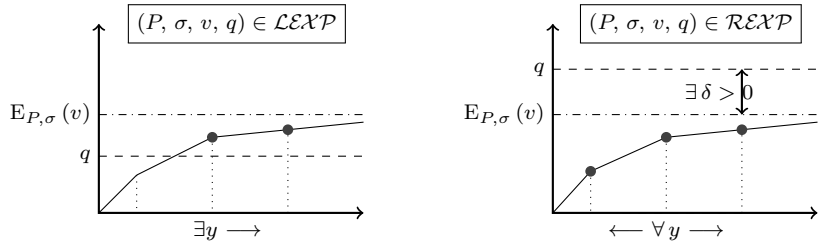


Fig. 1 Schematic depiction of the formulae defining $\mathcal{LEXP}$ (left diagram) and $\mathcal{REXP}$ (right diagram), respectively. In each diagram, the solid line represents the monotonically increasing graph of $\sum_{k=0}^{y} \sum_{w \in \{L,\, R\}^{\leq k}} \wp\big(\mathrm{T}_k(\gamma_{P,\sigma},\, w),\, v\big)$ plotted over increasing y.

It remains to show that $\mathcal{LEXP}$ is Σ_1^0–hard by constructing a computable function r such that $r\colon \mathcal{H} \leq_{\mathrm{m}} \mathcal{LEXP}$. This function r takes an ordinary program $Q \in \mathsf{ordProg}$ and variable valuation σ as its input and computes $(P,\, \sigma,\, 1,\, 1/2)$, where P is the following probabilistic program:

$$\{\texttt{skip}\}\ [1/2]\ \{Q\}$$

Correctness of the reduction: There are two cases: (1) Q terminates on input σ. Then P terminates with probability 1 and the expected value of 1 [9] after executing the program P on input σ is thus 1. As $1/2 < 1$, we have that $(P,\, \sigma,\, 1,\, 1/2) \in \mathcal{LEXP}$.

(2) Q does not terminate on input σ. Then P terminates with probability $1/2$ and the expected value of 1 after executing the program P on input σ is thus $1/2 \cdot 1 = 1/2$ since the right branch contributes 0 to the expected value of 1. As $1/2 \not< 1/2$, we have that $(P,\, \sigma,\, 1,\, 1/2) \notin \mathcal{LEXP}$. □

As an immediate consequence of Theorem 4, $\mathcal{LEXP}$ is computably enumerable (cf. Definition 5). This means that all lower bounds for preexpectations can be effectively enumerated by some algorithm. Now, if upper bounds were computably enumerable as well, then preexpectations would correspond to computable reals. However, we show that the contrary holds, because $\mathcal{REXP}$ is Σ_2^0–complete. Thus $\mathcal{REXP} \notin \Sigma_1^0$, i.e. upper bounds are not computably enumerable.

We now establish the Σ_2^0–hardness of $\mathcal{REXP}$ reduction from $\overline{\mathcal{UH}}$, i.e. the complement of the universal halting problem for ordinary programs (see Theorem 2).

To simplify notation, we define the following abbreviation (function $\wp$ is introduced in Proposition 2):

$$\alpha\colon \quad \mathbb{K}_{\perp} \to \mathbb{Q}_{\geq 0},\ \gamma \mapsto \wp(\gamma, 1)\ .$$

Intuitively, α can be used to express termination probabilities, as we will see later.

Overall we get the following hardness result for approximating preexpectations from above:

Theorem 5 *$\mathcal{REXP}$ is Σ_2^0–complete.*

Proof For proving the membership $\mathcal{REXP} \in \Sigma_2^0$, consider the following:

$$(P,\, \sigma,\, f,\, q) \in \mathcal{REXP}$$

[9] I.e. the postexpectation that maps every program state to constantly 1.

$$\iff \sigma \in \mathbb{S}_P \wedge q > \mathrm{E}_{P,\sigma}(f) \qquad \text{(Definition 8)}$$

$$\iff \sigma \in \mathbb{S}_P \wedge q > \sum_{k=0}^{\infty} \mathrm{E}^k_{P,\sigma}(f) \qquad \text{(Definition 7)}$$

$$\iff \sigma \in \mathbb{S}_P \wedge \exists\, \delta > 0\, \forall\, y\colon q - \delta > \sum_{k=0}^{y} \mathrm{E}^k_{P,\sigma}(f)$$

$$\implies \mathcal{REXP} \in \Sigma^0_2 \qquad \text{(the above is a } \Sigma^0_2\text{–formula)}$$

Figure 1 (right) gives an intuition on the resulting Σ^0_2–formula: No matter what maximum computation length y we allow and thereby no matter how much probability mass of the actual expected value we accumulate, this probability mass is strictly smaller than q (ensured by the safety margin δ).

It remains to show that $\mathcal{REXP}$ is Σ^0_2–hard: We do this by constructing a function $r\colon \overline{\mathcal{UH}} \leq_{\mathrm{m}} \mathcal{REXP}$: This function r takes an ordinary program $Q \in \mathsf{ordProg}$ as its input and computes the tuple $(P, \sigma[v \mapsto 0], v, 1)$, where σ is an arbitrary but fixed input, and $P \in \mathsf{Prog}$ is the following probabilistic program:

```
i := 0;
{c := 0} [1/2] {c := 1};

while (c ≠ 0){                  // compute geometric distribution of i
    i := i + 1;                   // with parameter 0.5
    {c := 0} [1/2] {c := 1}
};

k := 0;
{c := 0} [1/2] {c := 1};

while (c ≠ 0){                   // compute geometric distribution of k
    k := k + 1;                  // with parameter 0.5
    {c := 0} [1/2] {c := 1}
};

v := 0;
TQ ,
```

where TQ is an ordinary program that computes[10] $\alpha\big(\mathrm{T}_k\big(\gamma_{Q,g_Q(i)}, \varepsilon\big)\big) \cdot 2^{k+1}$ and stores the result in the variable v, and $g_Q\colon \mathbb{N} \to \mathbb{S}_Q$ is some computable enumeration of valid inputs for Q. Thus $\alpha\big(\mathrm{T}_k\big(\gamma_{Q,g_Q(i)}, \varepsilon\big)\big) \cdot 2^{k+1}$ returns 2^{k+1} if and only if Q terminates on input $g_Q(i)$ after exactly k steps (otherwise it returns 0).

Correctness of the reduction: The two while–loops generate independent geometric distributions with parameter $1/2$ on i and k, respectively, so the probability of generating exactly the numbers i and k is $1/2^{i+1} \cdot 1/2^{k+1} = 1/2^{i+k+2}$. The expected valuation of v after executing the program P is hence independent of the input σ and given by

$$\sum_{i=0}^{\infty} \sum_{k=0}^{\infty} \frac{1}{2^{i+k+2}} \cdot \alpha\Big(\mathrm{T}_k\Big(\gamma_{Q,g_Q(i)}, \varepsilon\Big)\Big) \cdot 2^{k+1} \ .$$

[10] The ε in the $\mathrm{T}_k(\ldots, \varepsilon)$ comes from the fact that T_k is supposed to simulate k steps of an *ordinary* program. The ε thus stands for an empty sequence of resolutions of probabilistic choices.

Since for each input, the number of steps until termination of Q is either unique or does not exist, the formula for the expected outcome reduces to $\sum_{i=0}^{\infty} 1/2^{i+1} = 1$ if and only if Q terminates on every input after some finite number of steps. Thus if there exists an input on which Q *does not* eventually terminate, then $(P, \sigma[v \mapsto 0], v, 1) \in \mathcal{REXP}$ as then the preexpectation of v is strictly less than 1. If, on the other hand, Q *does* terminate on every input, then this preexpectation is exactly 1 and hence $(P, \sigma[v \mapsto 0], v, 1) \notin \mathcal{REXP}$. □

As mentioned before, a consequence of Theorem 4 and Theorem 5 for approximating preexpectations is that upper bounds are not computable at all whereas lower bounds are at least computably enumerable. Upper bounds would be computably enumerable if we had access to an oracle for the (non–universal) halting problem $\mathcal{H}$ for ordinary programs. Given a rational q it would then be semi–decidable whether q is an upper bound when provided access to an oracle for $\mathcal{H}$. Next, we establish that this is not the case for the problem of deciding whether q *equals* exactly the value of the preexpectation. Formally, we establish the following hardness result:

Theorem 6 *$\mathcal{EXP}$ is Π_2^0–complete.*

Proof For proving the membership $\mathcal{EXP} \in \Pi_2^0$ consider the following: By Theorem 5 there exists a decidable relation $\mathcal{R}$, such that

$$(P, \sigma, v, q) \in \mathcal{REXP} \quad \text{iff} \quad \exists\, r_1\, \forall\, r_2 \colon (r_1, r_2, P, \sigma, v, q) \in \mathcal{R}\ .$$

Furthermore, by Theorem 4 there exists a decidable relation $\mathcal{L}$, such that

$$(P, \sigma, v, q) \in \mathcal{LEXP} \quad \text{iff} \quad \exists\, \ell \colon (\ell, P, \sigma, v, q) \in \mathcal{L}\ .$$

Let $\neg\mathcal{R}$ and $\neg\mathcal{L}$ be the (decidable) negations of $\mathcal{R}$ and $\mathcal{L}$, respectively. Then:

$$\begin{aligned}
& (P, \sigma, v, q) \in \mathcal{EXP} \\
\iff\ & \sigma \in \mathbb{S}_P \wedge q = \mathrm{E}_{P,\sigma}(v) && \text{(Definition 8)} \\
\iff\ & \sigma \in \mathbb{S}_P \wedge q \leq \mathrm{E}_{P,\sigma}(v) \wedge q \geq \mathrm{E}_{P,\sigma}(v) \\
\iff\ & \sigma \in \mathbb{S}_P \wedge \neg\big(q > \mathrm{E}_{P,\sigma}(v)\big) \wedge \neg\big(q < \mathrm{E}_{P,\sigma}(v)\big) \\
\iff\ & \sigma \in \mathbb{S}_P \wedge \neg\big(\exists\, r_1\, \forall\, r_2 \colon (r_1, r_2, P, \sigma, v, q) \in \mathcal{R}\big) \wedge \neg\big(\exists\, \ell \colon (\ell, P, \sigma, v, q) \in \mathcal{L}\big) \\
\iff\ & \sigma \in \mathbb{S}_P \wedge \big(\forall\, r_1\, \exists\, r_2 \colon (r_1, r_2, P, \sigma, v, q) \in \neg\mathcal{R}\big) \wedge \big(\forall\, \ell \colon (\ell, P, \sigma, v, q) \in \neg\mathcal{L}\big) \\
\iff\ & \sigma \in \mathbb{S}_P \wedge \forall\, r_1\, \forall\, \ell\, \exists\, r_2 \colon (r_1, r_2, P, \sigma, v, q) \in \neg\mathcal{R} \wedge (\ell, P, \sigma, v, q) \in \neg\mathcal{L} \\
\implies\ & \mathcal{EXP} \in \Pi_2^0 && \text{(the above is a } \Pi_2^0\text{–formula)}
\end{aligned}$$

It remains to show that $\mathcal{EXP}$ is Π_2^0–hard. We do this by proving $\mathcal{UH} \leq_{\mathrm{m}} \mathcal{EXP}$. Reconsider the reduction function r from the proof of Theorem 5: Given an ordinary program Q, r computes the tuple $(P, \sigma[v \mapsto 0], v, 1)$, where P is a probabilistic program with $\mathrm{E}_{P,\sigma}(\mathbf{1}) = 1$ if and only if Q terminates on all inputs. Thus $Q \in \mathcal{UH}$ iff $(P, \sigma[v \mapsto 0], v, 1) \in \mathcal{EXP}$ and therefore $r \colon \mathcal{UH} \leq_{\mathrm{m}} \mathcal{EXP}$. □

Apart from approximating preexpectations, we consider the question whether a preexpectation is finite. This is formalized by the problem set $\mathcal{FEXP}$:

Definition 9 The problem set $\mathcal{FEXP} \subset \mathsf{Prog} \times \mathbb{S} \times \mathbb{E}$ is defined as

$$(P, \sigma, f) \in \mathcal{FEXP} \quad \text{iff} \quad \sigma \in \mathbb{S}_P \quad \text{and} \quad \mathrm{E}_{P,\sigma}(f) < \infty. \qquad \triangle$$

Since deciding whether a given rational number is an upper bound of a pre-expectation is Σ_2^0–complete (cf. Theorem 5), it is not surprising that deciding finiteness of preexpectations is also in Σ_2^0. It is in fact Σ_2^0–complete as well.

Theorem 7 *$\mathcal{FEXP}$ is Σ_2^0–complete.*

Proof For proving the membership $\mathcal{FEXP} \in \Sigma_2^0$, consider the following:

$$
\begin{aligned}
& (P,\sigma,f) \in \mathcal{FEXP} \\
\iff\ & \sigma \in \mathbb{S}_P \wedge \mathrm{E}_{P,\sigma}(f) \ < \ \infty \\
\iff\ & \sigma \in \mathbb{S}_P \wedge \exists\, q\colon \mathrm{E}_{P,\sigma}(f) \ < \ q \\
\iff\ & \exists\, q\colon \sigma \in \mathbb{S}_P \wedge \mathrm{E}_{P,\sigma}(f) \ < \ q \\
\iff\ & \exists\, q\colon (P,\sigma,f,q) \in \mathcal{REXP} && \text{(Definition 8)} \\
\implies\ & \mathcal{FEXP} \in \Sigma_2^0 && \text{(Theorem 5, the above is a } \Sigma_2^0\text{–formula)}
\end{aligned}
$$

The proof that $\mathcal{FEXP}$ is Σ_2^0–hard is deferred to Lemma 1, because we use a reduction from the positive almost–sure termination problem (Definition 11), which is studied in detail in the next section. □

This concludes our study of the computational hardness of approximating pre-expectations. In the next section, we study computational aspects of analyzing the termination behavior of probabilistic programs.

4 The Hardness of Deciding Probabilistic Termination

We now turn towards the computational hardness of analyzing the termination behavior of probabilistic programs. We are interested in two notions: The termination probability and finiteness of the expected runtime of the program. Note that these two notions *do not* coincide but the latter implies the former. Using the function α as defined earlier, we can capture the termination probability and the expected runtime of a probabilistic program on a given input as follows: Recall $\gamma_{P,\sigma} = \langle P,\, \sigma,\, 1,\, \varepsilon \rangle$ (see Definition 2) and $A^{\leq k} = \bigcup_{i=0}^{k} A^i$ (see Definition 7). Then we define the following:

Definition 10 (Termination Probabilities and Expected Runtimes) Let $P \in \mathsf{Prog}$ and $\sigma \in \mathbb{S}_P$. Then

1. the **probability that P terminates on σ** is given by
$$\Pr_{P,\sigma}(\downarrow) \ = \ \mathrm{E}_{P,\sigma}(1) \ ,$$
2. the **expected runtime of P on σ** is given by
$$\mathrm{E}_{P,\sigma}(\downarrow) \ = \ \sum_{k=0}^{\infty} \left(1 - \sum_{w \in \{L,\,R\}^{\leq k}} \alpha\big(\mathrm{T}_k(\gamma_{P,\sigma},\, w)\big) \right) \ . \qquad \triangle$$

For the termination probability $\Pr_P(\downarrow)$, we sum up the probabilities of all reachable final configurations. This coincides with computing the expected value of the

constant function $\lambda\sigma\,.\,1$. As for $\mathrm{E}_{P,\sigma}\,(\downarrow)$, it is pointed out in [14] that the expected runtime of P on σ can be expressed as

$$\begin{aligned}\mathrm{E}_{P,\sigma}\,(\downarrow) &= \sum_{k=0}^{\infty} \Pr(\text{“}P \text{ runs for more than } k \text{ steps on } \sigma\text{”}) \\ &= \sum_{k=0}^{\infty} \big(1 - \Pr(\text{“}P \text{ terminates within } k \text{ steps on } \sigma\text{”})\big)\ .\end{aligned}$$

Regarding the termination probability of a probabilistic program, the case of almost–sure termination is of special interest: A program P *terminates almost–surely* on input σ iff P terminates on σ with probability 1. Furthermore, we say that P *terminates positively almost–surely* on σ iff the expected runtime of P on σ is finite. Lastly, we say that P terminates *universally* (positively) almost–surely, if it does so on all possible inputs σ. Notice that almost–sure termination *does not* imply positive almost–sure termination. The reversed implication, however, is true. Thus positive almost–sure termination is the stronger notion.

For analyzing the hardness of deciding (universal) (positive) almost–sure termination, we formally define the according problem sets:

Definition 11 (Probabilistic Termination Problem Sets) The sets $\mathcal{AST}$, $\mathcal{PAST}$, $\mathcal{UAST}$, and $\mathcal{PAST}$ are defined as follows:

$$\begin{aligned}(P,\sigma) \in \mathcal{AST} \quad &\text{iff} \quad \sigma \in \mathbb{S}_P \text{ and } \Pr_{P,\sigma}(\downarrow) = 1 \\ (P,\sigma) \in \mathcal{PAST} \quad &\text{iff} \quad \sigma \in \mathbb{S}_P \text{ and } \mathrm{E}_{P,\sigma}\,(\downarrow) < \infty \\ \\ P \in \mathcal{UAST} \quad &\text{iff} \quad \forall\, \sigma \in \mathbb{S}_P \colon (P,\sigma) \in \mathcal{AST} \\ P \in \mathcal{UPAST} \quad &\text{iff} \quad \forall\, \sigma \in \mathbb{S}_P \colon (P,\sigma) \in \mathcal{PAST}\end{aligned}$$

Notice that both $\mathcal{PAST} \subset \mathcal{AST}$ and $\mathcal{UPAST} \subset \mathcal{UAST}$ hold. △

The problem of (universal) almost–sure termination is often considered as the probabilistic counterpart to the (universal) halting problem for ordinary programs. Supported by our hardness results, we will, however, argue why perhaps (universal) positive almost–sure termination is a more suitable probabilistic analog.

As a first hardness result, we establish that deciding almost–sure termination of a program on a given input is Π^0_2–complete:

Theorem 8 *$\mathcal{AST}$ is Π^0_2–complete.*

Proof For proving $\mathcal{AST} \in \Pi^0_2$, we show $\mathcal{AST} \leq_{\mathrm{m}} \mathcal{EXP}$. For that, consider the following function r which takes a probabilistic program Q and a state σ as its input and computes the tuple $(Q, \sigma, 1, 1)$.

Correctness of the reduction: The expected value of postexpectation 1 after executing P on input σ is exactly the termination probability, see Definition 10. Thus $(P, \sigma, 1, 1) \in \mathcal{EXP}$ iff $(Q, \sigma) \in \mathcal{AST}$ and therefore $r \colon \mathcal{AST} \leq_{\mathrm{m}} \mathcal{EXP}$. Since $\mathcal{EXP}$ is Π^0_2–complete by Theorem 6, it follows that $\mathcal{AST} \in \Pi^0_2$.

It remains to show that $\mathcal{AST}$ is Π^0_2–hard. For that we reduce the Π^0_2–complete universal halting problem to $\mathcal{AST}$. Our reduction function r' takes an ordinary program Q as its input and computes the pair (P', σ), where σ is some arbitrary input in $\mathbb{S}_{P'}$ and P' is the probabilistic program

```
i := 0;
{c := 0} [1/2] {c := 1};

while (c ≠ 0){              // compute geometric distribution of i
    i := i + 1;             // with parameter 0.5
    {c := 0} [1/2] {c := 1}
};

SQ(i) ,
```

where $SQ(i)$ is an ordinary program that regardless of its input σ simulates the program Q on input $g_Q(i)$, and $g_Q\colon \mathbb{N} \to \mathbb{S}_Q$ is some computable enumeration of valid inputs for Q.

Correctness of the reduction: The while–loop in P' establishes a geometric distribution with parameter $1/2$ on i and hence a geometric distribution on all possible inputs for Q. After the while–loop, the program Q is simulated on the input generated probabilistically in the while–loop. Obviously then the entire program P' terminates with probability 1 on any arbitrary input σ, i.e. terminates almost–surely on σ, if and only if the simulation of Q terminates on every input. Thus $Q \in \mathcal{UH}$ if and only if $(P', \sigma) \in \mathcal{AST}$. □

While for ordinary programs there is a complexity gap between the halting problem for some given input and the universal halting problem (Σ_1^0–complete vs. Π_2^0–complete), we establish that *there is no such gap for almost–sure termination*, i.e. $\mathcal{UAST}$ is exactly as hard to decide as $\mathcal{AST}$:

Theorem 9 *$\mathcal{UAST}$ is Π_2^0–complete.*

Proof For proving $\mathcal{UAST} \in \Pi_2^0$, consider that, by Theorem 8, there exists a decidable relation $\mathcal{R}$, such that

$$(P, \sigma) \in \mathcal{AST} \quad \text{iff} \quad \forall\, y_1\, \exists\, y_2 \colon (y_1, y_2, P, \sigma) \in \mathcal{R} \ .$$

By that we have that

$$P \in \mathcal{UAST} \quad \text{iff} \quad \forall\, \sigma \in \mathbb{S}_P\, \forall\, y_1\, \exists\, y_2 \colon (y_1, y_2, P, \sigma) \in \mathcal{R}$$

which is a Π_2^0–formula and therefore $\mathcal{UAST} \in \Pi_2^0$.

It remains to show that $\mathcal{UAST}$ is Π_2^0–hard. This can be done by proving $\mathcal{AST} \leq_{\mathrm{m}} \mathcal{UAST}$ as follows: On input (Q, σ) the reduction function $r\colon \mathcal{AST} \leq_{\mathrm{m}} \mathcal{UAST}$ computes a probabilistic program P that first initializes all variables according to σ and then executes Q. This reduction is clearly correct. □

As mentioned above, this result is sort of opposed to the result for ordinary programs. There a Σ_1^0–formula expressing termination on a certain input is prepended with a universal quantifier over all possible inputs yielding a Π_2^0–formula. The reason for the missing complexity gap between non–universal and universal *almost–sure* termination is that non–universal almost–sure termination is already a Π_2^0–property, basically due to the inherent universal quantification over all resolutions of probabilistic choices. Prepending this Π_2^0–formula with another universal

quantifier over all possible inputs does not increase the complexity, as two universal quantifiers can computably be contracted to a single one yielding again a Π_2^0–formula.

We now investigate the computational hardness of deciding *positive* almost–sure termination: It turns out that deciding $\mathcal{PAST}$ is Σ_2^0–complete. Thus, $\mathcal{PAST}$ becomes semi–decidable when given access to an $\mathcal{H}$–oracle whereas $\mathcal{AST}$ does not. We establish Σ_2^0–hardness by a reduction from $\overline{\mathcal{UH}}$. The implications of this reduction are counterintuitive as it means that the reduction function *effectively* transforms each ordinary program that *does not terminate* on all inputs into a probabilistic program that *does terminate* within an expected finite number of steps.

Theorem 10 *$\mathcal{PAST}$ is Σ_2^0–complete.*

Proof For proving $\mathcal{PAST} \in \Sigma_2^0$, consider the following:

$$
\begin{aligned}
& (P, \sigma) \in \mathcal{PAST} \\
\iff\ & \sigma \in \mathbb{S}_P \wedge \infty > \mathrm{E}_{P,\sigma}(\downarrow) && \text{(Definition 11)} \\
\iff\ & \sigma \in \mathbb{S}_P \wedge \exists\, c\colon c > \mathrm{E}_{P,\sigma}(\downarrow) \\
\iff\ & \sigma \in \mathbb{S}_P \wedge \exists\, c\colon c > \sum_{k=0}^{\infty} \left(1 - \sum_{w \in \{L,R\}^{\leq k}} \alpha\big(\mathrm{T}_k(\sigma_{P,\sigma}, w)\big)\right) && \text{(Definition 10)} \\
\iff\ & \sigma \in \mathbb{S}_P \wedge \exists\, c\, \forall\, \ell\colon c > \sum_{k=0}^{\ell} \left(1 - \sum_{w \in \{L,R\}^{\leq k}} \alpha\big(\mathrm{T}_k(\sigma_{P,\sigma}, w)\big)\right) \\
\implies\ & \mathcal{PAST} \in \Sigma_2^0 && \text{(the above is a } \Sigma_2^0\text{–formula)}
\end{aligned}
$$

It remains to show that $\mathcal{PAST}$ is Σ_2^0–hard. For that, we use a reduction function $r\colon \overline{\mathcal{UH}} \leq_{\mathrm{m}} \mathcal{PAST}$ with $r(Q) = (P, \sigma)$, where σ is an arbitrary valid input for P and P is the probabilistic program

```
c := 1; i := 0; x := 0; term := 0;
InitQ(i);

while (c ≠ 0){
    StepQ(i);

    if (term = 1){
        Cheer(x);
        i := i + 1; term := 0;
        InitQ(i)
    } else {skip};

    x := x + 1;
    {c := 0} [1/2] {c := 1}
} ,
```

where $\mathit{InitQ}(i)$ is an ordinary program that initializes a simulation of the program Q on input $g_Q(i)$ (recall the enumeration g_Q from Theorem 5), $\mathit{StepQ}(i)$ is an

ordinary program that does one single (further) step of that simulation and sets *term* to 1 if that step has led to termination of Q, and $Cheer(x)$ is an ordinary program that executes 2^x many effectless computation steps. We refer to this as "cheering"[11].

Correctness of the reduction: Intuitively, the program P starts by simulating Q on input $g_Q(0)$. During the simulation, it—figuratively speaking—gradually looses interest in further simulating Q by tossing a coin after each simulation step to decide whether to continue the simulation or not. If eventually P finds that Q has terminated on input $g_Q(0)$, it "cheers" for a number of steps exponential in the number of coin tosses that were made so far, namely for 2^x steps. P then continues with the same procedure for the next input $g_Q(1)$, and so on.

The variable x keeps track of the number of loop iterations (starting from 0), which equals the number of coin tosses. The x–th loop iteration takes place with probability $1/2^x$.

Notice that the simulation of a single step of the ordinary program Q, i.e. the program $StepQ(i)$, requires (in our runtime model) at most a number linear in the number of instructions in program Q. Since the size of program Q is fixed in the construction of program P, we consider the time required to execute $StepQ(i)$ to be constant.[12]

One loop iteration then consists of a constant number c_1 of steps in case Q did not terminate on input $g_Q(i)$ in the current simulation step. Such an iteration therefore contributes $c_1/2^x$ to the expected runtime of P. In case Q did terminate, a loop iteration takes a constant number c_2 of steps plus 2^x additional "cheering" steps. Such an iteration therefore contributes $c_2+2^x/2^x = c_2/2^x + 1 > 1$ to the expected runtime. Overall, the expected runtime $\mathrm{E}_{P,\sigma}(\downarrow)$ roughly resembles a geometric series with exponentially decreasing summands. However, for each time the program Q terminates on an input, a summand of the form $c_2/2^x + 1$ appears in this series. There are now two cases:

(1) $Q \in \overline{\mathcal{UH}}$, so there exists some valid input σ with minimal i such that $g_Q(i) = \sigma$ on which Q does not terminate. In that case, summands of the form $c_2/2^x+1$ appear only $i-1$ times in the series and therefore, the series converges—the expected runtime is finite, so $(P, \sigma) \in \mathcal{PAST}$.

(2) $Q \notin \overline{\mathcal{UH}}$, so Q terminates on every input. In that case, summands of the form $c_2/2^x + 1$ appear infinitely often in the series and therefore, the series diverges—the expected runtime is infinite, so $(P, \sigma) \notin \mathcal{PAST}$. □

We are now in a position to present the missing part of the proof of Theorem 7: We show that deciding $\mathcal{FEXP}$, i.e. the question whether a preexpectation computed by a probabilistic program for a given input is finite, is Σ_2^0–hard by reduction from the positive almost–sure termination problem.

Lemma 1 *$\mathcal{FEXP}$ is Σ_2^0–hard.*

[11] The program P cheers as it was able to prove the termination of Q on input $g_Q(i)$.

[12] The runtime of a program corresponds to the number of execution steps in our operational semantics of pGCL (Definition 3). If more fine–grained runtime models are considered that take, for instance, the size of numbers into account, a single program step can be simulated in at most polynomial time on a Turing machine. To this end, we first translate pGCL programs to programs on a random access machine and translate the resulting program to Turing machines (cf. [37, Theorem 2.5]). In particular, our reduction remains valid for such more fine–grained runtime models.

Proof We use a reduction function $r\colon \mathcal{PAST} \leq_{\mathrm{m}} \mathcal{FEXP}$ that is given by $r(P, \sigma) = (P', \sigma', v)$, where σ' is an arbitrary valid input for P' and P' is the probabilistic program

```
c := 1; k := 0;

while (c ≠ 0){
    k := k + 1;
    {c := 0} [1/2] {c := 1}
}

v := 0;
TP(k) ,
```

where $TP(k)$ is an ordinary program that computes

$$\Pr(\text{“}P \text{ terminates after exactly } k \text{ steps on input } \sigma\text{”}) \cdot k \cdot 2^k \qquad (\dagger)$$

and stores this value in variable v.

Correctness of the reduction: Regardless of its input σ', the while loop of program P' establishes a geometric distribution on variable k such that the probability that $k = i$ is given by $1/2^i$. This while loop terminates almost–surely. Thereafter, the program computes (†). Due to the geometric distribution on k, the program in effect stores $\Pr(\text{“}P \text{ terminates after exactly } i \text{ steps on input } \sigma\text{”}) \cdot i \cdot 2^i$ in variable v with probability $1/2^i$. The expected value of variable v is thus given by

$$\begin{aligned}
\mathrm{E}_{P,\sigma}(v) &= \sum_i^{\infty} \frac{\Pr(\text{“}P \text{ terminates after exactly } i \text{ steps on input } \sigma\text{”}) \cdot i \cdot 2^i}{2^i} \\
&= \sum_i^{\infty} \Pr(\text{“}P \text{ terminates after exactly } i \text{ steps on input } \sigma\text{”}) \cdot i \\
&= \mathrm{E}_{P,\sigma}(\downarrow) \ .
\end{aligned}$$

We see that the expected value of v after executing P' on an arbitrary input equals exactly the expected runtime of P on input σ and this expected value is infinite if and only if the expected runtime is infinite. Thus, we have $(P, \sigma) \in \mathcal{PAST}$ iff $r(P, \sigma) = (P', \sigma', v) \in \mathcal{FEXP}$ and hence $r\colon \mathcal{PAST} \leq_{\mathrm{m}} \mathcal{FEXP}$. □

It is noteworthy that we cannot just annotate the given program with a runtime counter and determine the expected value of that runtime counter. In order to see that, consider the program `while (true) {skip}` and its annotated version

```
counter := 0;
while (true){
    counter := counter + 1;
    skip
} .
```

The expected value of variable *counter* is 0, since the program terminates with probability 0 and there is no probability mass whatsoever that could contribute to the expected value of *counter*. On the other hand, the expected runtime of `while (true) {skip}` is clearly ∞. The expected value of the runtime counter is

thus unequal to the expected runtime and therefore we need the more involved construction presented in the proof of Lemma 1.

Coming back to termination problems, the last problem we study is *universal* positive almost–sure termination. In contrast to the non–positive version, we *do* have a complexity leap when moving from non–universal to universal positive almost–sure termination. We will establish that $\mathcal{UPAST}$ is Π^0_3–complete and thus strictly harder to decide than $\mathcal{UAST}$. We do this by a reduction from $\overline{\mathcal{COF}}$, the complement of the cofiniteness problem (see Theorem 3):

Theorem 11 *$\mathcal{UPAST}$ is Π^0_3–complete.*

Proof By Theorem 10, there exists a decidable relation $\mathcal{R}$, such that

$$(P, \sigma) \in \mathcal{PAST} \quad \text{iff} \quad \exists\, y_1\, \forall\, y_2 \colon (y_1, y_2, P, \sigma) \in \mathcal{R}\ .$$

Therefore $\mathcal{UPAST}$ is definable by

$$P \in \mathcal{UPAST} \quad \text{iff} \quad \forall\, \sigma \in \mathbb{S}_P\ \exists\, y_1\, \forall\, y_2 \colon (y_1, y_2, P, \sigma) \in \mathcal{R}\ ,$$

which is a Π^0_3–formula and therefore $\mathcal{UPAST} \in \Pi^0_3$.

It remains to show that $\mathcal{UPAST}$ is Π^0_3–hard. For that we reduce the Π^0_3–complete complement of the cofiniteness problem to $\mathcal{UPAST}$ using the following function f: f takes an ordinary program Q as its input and computes the probabilistic program P given by

```
c := 1; i := max{⌈i⌉, 0}; x := 0; term := 0;
InitQ(i);

while (c ≠ 0){
    StepQ(i);

    if (term = 1){
        Cheer(x);
        i := i + 1; term := 0;
        InitQ(i)
    } else {skip};

    x := x + 1;
    {c := 0} [1/2] {c := 1}
},
```

where $\mathit{InitQ}(i)$ is an ordinary program that initializes a simulation of the program Q on input $g_Q(i)$ (recall the enumeration g_Q from Theorem 5), $\mathit{StepQ}(i)$ is an ordinary program that does one single (further) step of that simulation and sets *term* to 1 if that step has led to termination of Q, and $\mathit{Cheer}(x)$ is an ordinary program that executes 2^x many effectless computation steps.

Note that program P is the same program as in the proof of Theorem 10 with one exception: The variable i is not initialized with 0, but rounded up to the next natural number, say ℓ, of the initial value of i in a given program state.[13] Thus,

[13] Rounding up the value of i to a natural number, i.e. computing $\max\{\lceil i \rceil, 0\}$, is a technical necessity: We assume that variable valuations range over $\mathbb{Q}$ but the domain of g_Q is $\mathbb{N}$.

for every input σ, the program P skips all inputs (in the order given by g_Q) up to the ℓ-th input, where $\ell = \max\{\lceil \sigma(i) \rceil, 0\}$. After that, program P simulates Q on all remaining inputs starting with input $g_Q(\ell)$.

This ability to skip any number of inputs for some input state σ is crucial for the correctness of the reduction. Intuitively, program P terminates in finite expected time on all input states, i.e. $P \in \mathcal{UPAST}$, if it is impossible to find an input state σ (and thus a value $\ell \in \mathbb{N}$ determined by $\sigma(i)$) such that executing P on σ skips all input states on which Q does not terminate. Otherwise, P (when executed on such an input state σ) keeps simulating terminating runs of Q and thus "cheers" infinitely often. In this case, the expected runtime of P on input state σ becomes infinite, i.e. $P \notin \mathcal{UPAST}$.

Correctness of the reduction: The problem $\overline{\mathcal{COF}}$ can alternatively be defined as

$$Q \in \overline{\mathcal{COF}} \quad \text{iff} \quad \{\sigma \in \mathbb{S}_Q \mid (Q, \sigma) \in \overline{\mathcal{H}}\} \text{ is infinite.}$$

There are now two cases:

(1) $Q \notin \overline{\mathcal{COF}}$. Then there are only *finitely* many inputs on which Q does not terminate. Say $\ell \in \mathbb{N}$ is a minimal value such that Q does not terminate on input $g_Q(\ell)$, i.e. the program Q terminates on all input states $g_Q(j)$ with $j > \ell$. Now, consider the execution of program P on some input σ with $\sigma(i) > \ell$. Then the "cheering" steps in the `if`–branch of the while–loop of P are executed infinitely often. Consequently, the runtime of P on that input σ is infinite (analogously to the proof of Theorem 10). Hence, $P \notin \mathcal{UPAST}$.

(2) $Q \in \overline{\mathcal{COF}}$. Then there are *infinitely* many inputs on which Q does not terminate. For every input state σ of P (and thus regardless of the number of skipped input states of Q, i.e. the value $\ell = \max\{\lceil \sigma(i) \rceil, 0\} \in \mathbb{N}$ that is initially assigned to variable i), the variable i will eventually be incremented to some value $j > \ell$ such that Q does not terminate on input $g_Q(j)$. From this point on, the "cheering" steps in the `if`–branch of the while–loop of P are *not* executed anymore. Consequently, for every input state σ, the expected time until termination of P in input state σ is finite. Hence, $P \in \mathcal{UPAST}$. □

This concludes our study of the computational hardness of analyzing probabilistic termination. We have seen that there is a complexity leap when moving from positive almost–sure termination to universal positive almost–sure termination, namely from Σ^0_2–complete (i.e. $\exists \forall \cdots$) to Π^0_3–complete (i.e. $\forall \exists \forall \cdots$).

Considering the quantifier ordering and the type of objects that are quantified, we believe that the problem of (universal) positive almost–sure termination is maybe a more natural probabilistic analog to the (universal) halting problem for ordinary programs: For ordinary programs, we can define the halting problem by $\exists$–quantifying over a witness computation length and then running the (unique) computation of at most that length on a given input. For the universal halting problem, we additionally prepend a $\forall$–quantification over all possible inputs.

Somewhat analogously, for probabilistic programs, we can define the problem of positive almost–sure termination by $\exists$–quantifying over a witness expected computation length and then running all computations (captured by a $\forall$–quantifier) of at most that expected length on a given input. For the universal version of the problem, we additionally prepend a $\forall$–quantifier over all possible inputs.

Another argument stems more from a user perspective. For a user, the expected runtime of an algorithm might be more relevant than its termination probability,

because an algorithm whose runtime can at least be estimated to some finite value is probably more useful in practice than an algorithm for which one has to expect to wait forever until the algorithm finishes its computation (even if it does so with probability 1). Along this line of thought, the expected runtime of an algorithm is also a key notion in defining probabilistic complexity classes such as ZPP—the class of decision problems that can be decided by a probabilistic program that always gives the correct answer *within expected polynomial time* [16].

5 Hardness of Approximating (Co)variances

The last group of analysis problems we study concerns the approximation of variances and covariances. More precisely, we are given a probabilistic program P, an initial state σ and two expectations, i.e. random variables, $f, g \in \mathbb{E}$. Furthermore, let μ denote the final distribution of program states that is generated by running P on input σ. What is then the covariance of f and g (the variance of f or the variance of g) under the final distribution μ? Having the operator $\mathrm{E}_{P,\sigma}(\,\cdot\,)$ for obtaining preexpectations, i.e. expected values of random variables, readily available, the textbook definition of variances and covariances of expectations is as follows (cf. [2, Definition 4.10.10, Lemma 4.10.6] or [27, Definition 5.1]):

Definition 12 Let $P \in \mathsf{Prog}$, $\sigma \in \mathbb{S}_P$, and $f, g \in \mathbb{E}$. Then

1. the **covariance of f and g after executing P on σ** is given by

$$\begin{aligned}\mathrm{Cov}_{P,\sigma}(f,\,g) &= \mathrm{E}_{P,\sigma}\left((f-\mathrm{E}_{P,\sigma}(f))\cdot(g-\mathrm{E}_{P,\sigma}(g))\right)\\ &= \mathrm{E}_{P,\sigma}(f\cdot g)-\mathrm{E}_{P,\sigma}(f)\cdot\mathrm{E}_{P,\sigma}(g)\,.\end{aligned}$$

   if $\mathrm{E}_{P,\sigma}(f)\,,\mathrm{E}_{P,\sigma}(g)$ and $\mathrm{E}_{P,\sigma}(f\cdot g)$ are finite; otherwise it is undefined,
2. the **variance of f after executing P on σ** is given by

$$\mathrm{Var}_{P,\sigma}(f) \;=\; \mathrm{E}_{P,\sigma}\left(f^2\right)-\left(\mathrm{E}_{P,\sigma}(f)\right)^2$$

   if $\mathrm{E}_{P,\sigma}(f)$ is finite; otherwise it is undefined. In particular, if $\mathrm{E}_{P,\sigma}\left(f^2\right)$ is also finite, then $\mathrm{Var}_{P,\sigma}(f) \;=\; \mathrm{Cov}_{P,\sigma}(f,\,f)$. △

Since definedness of covariances is not always guaranteed, we first address the question whether a covariance is defined. According to Definition 12, the covariance $\mathrm{Cov}_{P,\sigma}(f,\,g)$ is defined if and only if

$$\mathrm{E}_{P,\sigma}(f) \;<\; \infty \quad\text{and}\quad \mathrm{E}_{P,\sigma}(g) \;<\; \infty \quad\text{and}\quad \mathrm{E}_{P,\sigma}(f\cdot g) \;<\; \infty.$$

Hence, we are concerned with the following problem set:

Definition 13 The problem set $\mathcal{DCOVAR} \subset \mathsf{Prog}\times\mathbb{S}\times\mathbb{E}\times\mathbb{E}$ is defined as

$$\begin{aligned}(P,\sigma,f,g)\in\mathcal{DCOVAR} \quad\text{iff}\quad &\sigma\in\mathbb{S}_P \text{ and } \mathrm{E}_{P,\sigma}(f) \;<\; \infty \text{ and } \mathrm{E}_{P,\sigma}(g) \;<\; \infty\\ &\text{and } \mathrm{E}_{P,\sigma}(f\cdot g) \;<\; \infty.\end{aligned}$$ △

Apart from the usual condition that σ is a valid input state for P, each of these conditions can be reduced to the question whether the expected value of a random variable is finite. As a consequence of Definition 13 and Theorem 7, we obtain the following theorem on the hardness of deciding definedness of covariances:

Theorem 12 *$\mathcal{DCOVAR}$ is Σ_2^0–complete.*

Proof By Theorem 7 there is a decidable relation $\mathcal{F}$ such that

$$(P,\,\sigma,\,f) \in \mathcal{FEXP} \quad \text{iff} \quad \mathrm{E}_{P,\sigma}\,(f) \;<\; \infty \quad \text{iff} \quad \exists\, y_1\, \forall\, y_2 \colon (y_1,\, y_2,\, P,\, \sigma,\, f) \in \mathcal{F}$$

By the above we have the following:

$$\begin{aligned}
& (P,\,\sigma,\,f,\,g) \in \mathcal{DCOVAR} \\
\iff\; & \sigma \in \mathbb{S}_P \,\wedge\, \mathrm{E}_{P,\sigma}\,(f) \;<\; \infty \,\wedge\, \mathrm{E}_{P,\sigma}\,(g) \;<\; \infty \,\wedge\, \mathrm{E}_{P,\sigma}\,(f\cdot g) \;<\; \infty \\
\iff\; & \sigma \in \mathbb{S}_P \,\wedge\, \exists\, y_1\, \forall\, y_2 \colon (y_1,\, y_2,\, P,\, \sigma,\, f) \in \mathcal{F} \\
& \quad \wedge\, \exists\, y_1'\, \forall\, y_2' \colon (y_1',\, y_2',\, P,\, \sigma,\, g) \in \mathcal{F} \,\wedge\, \exists\, y_1''\, \forall\, y_2'' \colon (y_1'',\, y_2'',\, P,\, \sigma,\, f\cdot g) \in \mathcal{F} \\
\implies\; & \mathcal{DCOVAR} \in \Sigma_2^0
\end{aligned}$$

It remains to show that $\mathcal{DCOVAR}$ is Σ_2^0–hard. For that, we reduce the Σ_2^0–complete problem $\mathcal{FEXP}$ to $\mathcal{DCOVAR}$ by $r(P,\,\sigma,\,f) = (P,\,\sigma,\,f,\,0)$.

Correctness of the reduction: Consider the following:

$$\begin{aligned}
& (P,\,\sigma,\,f) \in \mathcal{FEXP} \\
\iff\; & \mathrm{E}_{P,\sigma}\,(f) < \infty \\
\iff\; & \mathrm{E}_{P,\sigma}\,(f) < \infty \,\wedge\, 0 < \infty \,\wedge\, 0 < \infty \\
\iff\; & \mathrm{E}_{P,\sigma}\,(f) < \infty \,\wedge\, \mathrm{E}_{P,\sigma}\,(0) < \infty \,\wedge\, \mathrm{E}_{P,\sigma}\,(0) < \infty \\
\iff\; & \mathrm{E}_{P,\sigma}\,(f) < \infty \,\wedge\, \mathrm{E}_{P,\sigma}\,(0) < \infty \,\wedge\, \mathrm{E}_{P,\sigma}\,(f\cdot 0) < \infty \\
\iff\; & (P,\,\sigma,\,f,\,0) \in \mathcal{DCOVAR}
\end{aligned}$$

Thus, we have that $(P,\,\sigma,\,f) \in \mathcal{FEXP}$ if and only if $r(P,\,\sigma,\,f) \in \mathcal{DCOVAR}$ and hence $r \colon \mathcal{FEXP} \leq_{\mathrm{m}} \mathcal{DCOVAR}$. □

We now turn towards the hardness of approximating covariances. Analogously to our studies on the hardness of approximating preexpectations, we define three problem sets: one for lower bounds, one for upper bounds, and one for the exact value of $\mathrm{Cov}_{P,\sigma}(f,\,g)$.

Definition 14 (Approximation Problems for Covariances) The problem sets $\mathcal{LCOVAR}, \mathcal{RCOVAR}, \mathcal{COVAR} \subset \mathsf{Prog} \times \mathbb{S} \times \mathbb{E} \times \mathbb{E} \times \mathbb{Q}$ are defined as

$$\begin{aligned}
& (P,\,\sigma,\,f,\,g,\,q) \in \mathcal{LCOVAR} \\
& \qquad \text{iff} \quad \sigma \in \mathbb{S}_P \text{ and } (P,\,\sigma,\,f,\,g) \in \mathcal{DCOVAR} \text{ and } q < \mathrm{Cov}_{P,\sigma}(f,\,g)\ ,
\end{aligned}$$

$$\begin{aligned}
& (P,\,\sigma,\,f,\,g,\,q) \in \mathcal{RCOVAR} \\
& \qquad \text{iff} \quad \sigma \in \mathbb{S}_P \text{ and } (P,\,\sigma,\,f,\,g) \in \mathcal{DCOVAR} \text{ and } q > \mathrm{Cov}_{P,\sigma}(f,\,g)\ ,
\end{aligned}$$

$$\begin{aligned}
& (P,\,\sigma,\,f,\,g,\,q) \in \mathcal{COVAR} \\
& \qquad \text{iff} \quad \sigma \in \mathbb{S}_P \text{ and } (P,\,\sigma,\,f,\,g) \in \mathcal{DCOVAR} \text{ and } q = \mathrm{Cov}_{P,\sigma}(f,\,g)\ .
\end{aligned}$$

The first fact we establish on computational hardness of approximating covariances is that approximating lower bounds of covariances is Σ_2^0–complete:

Theorem 13 *$\mathcal{LCOVAR}$ is Σ_2^0–complete.*

Proof For proving the membership $\mathcal{LCOVAR} \in \Sigma_2^0$, consider the following:

$$\begin{aligned}
& (P,\sigma,f,g,q) \in \mathcal{LCOVAR} \\
\iff\ & \sigma \in \mathbb{S}_P \ \wedge\ (P,\sigma,f,g) \in \mathcal{DCOVAR} \ \wedge\ q < \mathrm{Cov}_{P,\sigma}(f,\,g) && \text{(Definition 14)}
\end{aligned}$$

By Theorem 12 ($\mathcal{DCOVAR}$ is in Σ_2^0), there is a *decidable* relation $\mathcal{D}$ such that

$$\begin{aligned}
& (P,\sigma,f,g,q) \in \mathcal{LCOVAR} \\
\iff\ & \sigma \in \mathbb{S}_P \ \wedge\ \exists y_1\,\forall\, y_2\colon (P,\sigma,f,g) \in \mathcal{D} \ \wedge\ q < \mathrm{Cov}_{P,\sigma}(f,\,g) \\
\iff\ & \sigma \in \mathbb{S}_P \ \wedge\ \exists y_1\,\forall\, y_2\colon (P,\sigma,f,g) \in \mathcal{D} && \text{(Definition 12)} \\
& \wedge\ q < \mathrm{E}_{P,\sigma}(f\cdot g) - \mathrm{E}_{P,\sigma}(f)\cdot \mathrm{E}_{P,\sigma}(g) \\
\iff\ & \sigma \in \mathbb{S}_P \ \wedge\ \exists y_1\,\forall\, y_2\colon (P,\sigma,f,g) \in \mathcal{D} && \text{(Definition 7)} \\
& \wedge\ q < \sum_{k=0}^{\infty} \mathrm{E}^k_{P,\sigma}(f\cdot g) - \sum_{k=0}^{\infty} \mathrm{E}^k_{P,\sigma}(f)\cdot \sum_{k=0}^{\infty} \mathrm{E}^k_{P,\sigma}(g) \\
\iff\ & \sigma \in \mathbb{S}_P \ \wedge\ \exists y_1\,\forall\, y_2\colon (P,\sigma,f,g) \in \mathcal{D} \\
& \wedge\ \exists\, z_1\colon q < \sum_{k=0}^{z_1} \mathrm{E}^k_{P,\sigma}(f\cdot g) - \sum_{k=0}^{\infty} \mathrm{E}^k_{P,\sigma}(f)\cdot \sum_{k=0}^{\infty} \mathrm{E}^k_{P,\sigma}(g) \\
\iff\ & \sigma \in \mathbb{S}_P \ \wedge\ \exists y_1\,\forall\, y_2\colon (P,\sigma,f,g) \in \mathcal{D} \\
& \wedge\ \exists\, z_1 \forall\, z_2\colon q < \sum_{k=0}^{z_1} \mathrm{E}^k_{P,\sigma}(f\cdot g) - \sum_{k=0}^{z_2} \mathrm{E}^k_{P,\sigma}(f)\cdot \sum_{k=0}^{z_2} \mathrm{E}^k_{P,\sigma}(g) \\
\iff\ & \sigma \in \mathbb{S}_P \ \wedge\ \exists y_1 \exists\, z_1\,\forall\, y_2 \forall\, z_2\colon (P,\sigma,f,g) \in \mathcal{D} \\
& \wedge\ q < \sum_{k=0}^{z_1} \mathrm{E}^k_{P,\sigma}(f\cdot g) - \sum_{k=0}^{z_2} \mathrm{E}^k_{P,\sigma}(f)\cdot \sum_{k=0}^{z_2} \mathrm{E}^k_{P,\sigma}(g) \\
\implies\ & \mathcal{LCOVAR} \in \Sigma_2^0 && \text{(the above is a } \Sigma_2^0\text{–formula)}
\end{aligned}$$

For proving the Σ_2^0–hardness of $\mathcal{LCOVAR}$ we reduce $\overline{\mathcal{AST}}$ to $\mathcal{LCOVAR}$. Consider the reduction function $r\colon \overline{\mathcal{AST}} \leq_{\mathrm{m}} \mathcal{LCOVAR}$ with $r(P,\,\sigma) = (P',\,\sigma,\,1,\,1,\,0)$, where P' is given by

$$\{\texttt{skip}\}\ [1/2]\ \{P\}\ .$$

Correctness of the reduction: First observe that expected value $\mathrm{E}_{P',\sigma}(1)$ ranges from 0 to 1. In particular, this means that the expected values of 1 as well as 1^2 are finite, i.e. the covariance $\mathrm{Cov}_{P',\sigma}(1,\,1)$ is defined. Then

$$\begin{aligned}
\mathrm{Cov}_{P',\sigma}(1,\,1) &= \mathrm{E}_{P',\sigma}\left(1^2\right) - \mathrm{E}_{P',\sigma}(1)^2 && \text{(Definition 12)} \\
&= \mathrm{E}_{P',\sigma}(1) - \mathrm{E}_{P',\sigma}(1)^2\ .
\end{aligned}$$

Note that $\mathrm{E}_{P',\sigma}(1)$ is exactly the probability of P' terminating on input σ. A plot of this termination probability against the resulting variance is given in Figure 2. We observe that $\mathrm{Cov}_{P',\sigma}(1,\,1) = \mathrm{E}_{P',\sigma}(1) - \mathrm{E}_{P',\sigma}(1)^2 > 0$ holds iff P' terminates *neither* with probability 0 *nor* with probability 1. Since, however, P' terminates by construction *at least* with probability $1/2$, we obtain that $\mathrm{Cov}_{P',\sigma}(1,\,1) > 0$ iff

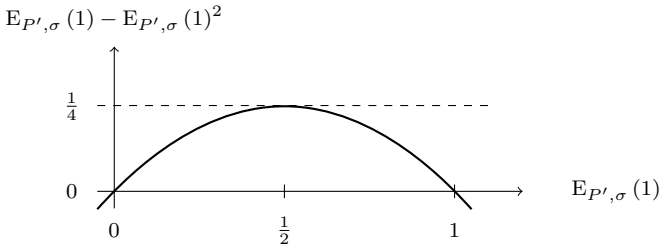


Fig. 2 Plot of the termination probability of a program P' on input σ against the resulting variance. The curve is the one of the polynomial $x - x^2$.

P' terminates with probability less than 1, which is the case iff P terminates with probability less than 1. Thus $r(P, \sigma) = (P', \sigma, 1, 1, 0) \in \mathcal{LCOVAR}$ iff $(P, \sigma) \in \overline{\mathcal{AST}}$ and therefore $r \colon \overline{\mathcal{AST}} \leq_{\mathrm{m}} \mathcal{LCOVAR}$. Since $\overline{\mathcal{AST}}$ is Σ_2^0–complete, if follows that $\mathcal{LCOVAR}$ is Σ_2^0–hard. □

Next, we show that approximating upper bounds for covariances is exactly as hard as approximating lower bounds, namely Σ_2^0–complete:

Theorem 14 *$\mathcal{RCOVAR}$ is Σ_2^0–complete.*

Proof For proving membership $\mathcal{RCOVAR} \in \Sigma_2^0$, consider the following:

$$
\begin{aligned}
& (P, \sigma, f, g, q) \in \mathcal{RCOVAR} \\
\iff\ & \sigma \in \mathbb{S}_P \ \wedge\ (P, \sigma, f, g) \in \mathcal{DCOVAR} \ \wedge\ q > \mathrm{Cov}_{P,\sigma}(f, g) && \text{(Definition 14)}
\end{aligned}
$$

Now, by Theorem 12 ($\mathcal{DCOVAR}$ is in Σ_2^0) there is a *decidable* relation $\mathcal{D}$ such that

$$
\begin{aligned}
& (P, \sigma, f, g, q) \in \mathcal{RCOVAR} \\
\iff\ & \sigma \in \mathbb{S}_P \ \wedge\ \exists\, y_1 \forall\, y_2 \colon (P, \sigma, f, g) \in \mathcal{D} \ \wedge\ q > \mathrm{Cov}_{P,\sigma}(f, g) \\
\iff\ & \sigma \in \mathbb{S}_P \wedge\ \exists\, y_1 \forall\, y_2 \colon (P, \sigma, f, g) \in \mathcal{D} \ \wedge\ \exists\, \delta > 0 \colon q - \delta > \mathrm{Cov}_{P,\sigma}(f, g) \\
\iff\ & \sigma \in \mathbb{S}_P \wedge\ \exists\, y_1 \forall\, y_2 \colon (P, \sigma, f, g) \in \mathcal{D} && \text{(Definition 12)} \\
& \wedge\ \exists\, \delta > 0 \colon q - \delta > \mathrm{E}_{P,\sigma}(f \cdot g) - \mathrm{E}_{P,\sigma}(f) \cdot \mathrm{E}_{P,\sigma}(g) \\
\iff\ & \sigma \in \mathbb{S}_P \wedge\ \exists\, y_1 \forall\, y_2 \colon (P, \sigma, f, g) \in \mathcal{D} && \text{(Definition 7)} \\
& \wedge\ \exists\, \delta > 0 \colon q - \delta > \sum_{k=0}^{\infty} \mathrm{E}_{P,\sigma}^k(f \cdot g) - \sum_{k=0}^{\infty} \mathrm{E}_{P,\sigma}^k(f) \cdot \sum_{k=0}^{\infty} \mathrm{E}_{P,\sigma}^k(g) \\
\iff\ & \sigma \in \mathbb{S}_P \wedge\ \exists\, y_1 \forall\, y_2 \colon (P, \sigma, f, g) \in \mathcal{D} \\
& \wedge\ \exists\, \delta > 0 \exists\, z_2 \colon q - \delta > \sum_{k=0}^{\infty} \mathrm{E}_{P,\sigma}^k(f \cdot g) - \sum_{k=0}^{z_2} \mathrm{E}_{P,\sigma}^k(f) \cdot \sum_{k=0}^{z_2} \mathrm{E}_{P,\sigma}^k(g) \\
\iff\ & \sigma \in \mathbb{S}_P \wedge\ \exists\, y_1 \forall\, y_2 \colon (P, \sigma, f, g) \in \mathcal{D} \\
& \wedge\ \exists\, \delta > 0 \exists\, z_2 \forall\, z_1 \colon q - \delta > \sum_{k=0}^{z_1} \mathrm{E}_{P,\sigma}^k(f \cdot g) - \sum_{k=0}^{z_2} \mathrm{E}_{P,\sigma}^k(f) \cdot \sum_{k=0}^{z_2} \mathrm{E}_{P,\sigma}^k(g) \\
\iff\ & \sigma \in \mathbb{S}_P \wedge\ \exists\, y_1 \exists\, \delta > 0 \exists\, z_2 \forall\, y_2 \forall\, z_1 \colon (P, \sigma, f, g) \in \mathcal{D} \\
& \wedge\ q - \delta > \sum_{k=0}^{z_1} \mathrm{E}_{P,\sigma}^k(f \cdot g) - \sum_{k=0}^{z_2} \mathrm{E}_{P,\sigma}^k(f) \cdot \sum_{k=0}^{z_2} \mathrm{E}_{P,\sigma}^k(g)
\end{aligned}
$$

$$\implies \mathcal{RCOVAR} \in \Sigma_2^0 \qquad \text{(the above is a } \Sigma_2^0\text{–formula)}$$

For proving the Σ_2^0–hardness of $\mathcal{RCOVAR}$, we reduce the Σ_2^0–complete $\overline{\mathcal{AST}}$ to $\mathcal{RCOVAR}$. Let (P, σ) be an instance of $\overline{\mathcal{AST}}$. Consider the reduction function $r(P, \sigma) = \big(P', \sigma, 1, 1, 1/4\big)$, with P' being the program

$$\{\texttt{diverge}\}\ [1/2]\ \{P\}\ .$$

Analogously to the proof of Theorem 13, $\mathrm{Cov}_{P',\sigma}(1, 1)$ is defined and we have

$$\mathrm{Cov}_{P',\sigma}(1, 1) \ = \ \mathrm{E}_{P',\sigma}\,(1) - \mathrm{E}_{P',\sigma}\,(1)^2\ .$$

Recall that $\mathrm{E}_{P',\sigma}\,(1)$ is exactly the probability of P' terminating on input σ. By reconsidering Figure 2, we can see that $\mathrm{Cov}_{P',\sigma}(1, 1) = \mathrm{E}_{P',\sigma}\,(1) - \mathrm{E}_{P',\sigma}\,(1)^2 < 1/4$ holds iff P' does not terminate with probability $1/2$. Since by construction P' terminates with a probability of at most $1/2$, it follows that $\mathrm{Cov}_{P',\sigma}(v, v) < 1/4$ holds iff P' terminates with probability less than $1/2$, which is the case iff P terminates with probability less than 1. Thus $r(P, \sigma) = \big(P', \sigma, 1, 1, 1/4\big) \in \mathcal{RCOVAR}$ iff $(P, \sigma) \in \overline{\mathcal{AST}}$ and therefore we have $r\colon \overline{\mathcal{AST}} \leq_{\mathrm{m}} \mathcal{RCOVAR}$. Since $\overline{\mathcal{AST}}$ is Σ_2^0–complete, if follows that $\mathcal{RCOVAR}$ is Σ_2^0–hard. □

Regarding the hardness of deciding whether a given rational is equal to the covariance we establish $\mathcal{COVAR}$ can be solved in $\Delta_3^0 = \Sigma_3^0 \cap \Pi_3^0$, $\mathcal{COVAR}$ is Π_2^0–hard, and $\mathcal{COVAR}$ is Σ_2^0–hard. $\mathcal{COVAR}$ is therefore at least as hard as deciding whether a non–probabilistic program terminates on all inputs or deciding whether a probabilistic program terminates positively almost–surely.

Theorem 15 *$\mathcal{COVAR}$ is in Δ_3^0.*

Proof To prove $\mathcal{COVAR} \in \Delta_3^0 = \Sigma_3^0 \cap \Pi_3^0$, consider that $(P, \sigma, f, g, q) \in \mathcal{COVAR}$ if and only if

$$\begin{aligned} &\sigma \in \mathbb{S}_P \ \wedge\ (P, \sigma, f, g) \in \mathcal{DCOVAR} \\ \wedge\ &(P, \sigma, f, g, q) \notin \mathcal{LCOVAR} \ \wedge\ (P, \sigma, f, g, q) \notin \mathcal{RCOVAR}. \end{aligned}$$

Now, by Theorem 12, Theorem 13, and Theorem 14 there exist *decidable* relations $\mathcal{D}$, $\mathcal{L}$, $\mathcal{R}$ such that

$$\begin{aligned} &(P, \sigma, f, g, q) \in \mathcal{COVAR} \\ \iff\ &\sigma \in \mathbb{S}_P \ \wedge\ \exists x_1 \forall x_2\colon (P, \sigma, f, g) \in \mathcal{D} \\ &\qquad \wedge\ \neg\exists\, y_1 \forall\, y_2\colon (P, \sigma, f, g, q) \in \mathcal{L} \ \wedge\ \neg\exists\, z_1 \forall\, z_2\colon (P, \sigma, f, g, q) \in \mathcal{R} \\ \iff\ &\sigma \in \mathbb{S}_P \ \wedge\ \exists x_1 \forall x_2\colon (P, \sigma, f, g) \in \mathcal{D} \\ &\qquad \wedge\ \forall\, y_1 \exists\, y_2\colon (P, \sigma, f, g, q) \notin \mathcal{L} \ \wedge\ \forall\, z_1 \exists\, z_2\colon (P, \sigma, f, g, q) \notin \mathcal{R} \\ \iff\ &\sigma \in \mathbb{S}_P \ \wedge\ \exists x_1 \forall x_2 \forall\, y_1 \forall\, z_1 \exists\, y_2 \exists\, z_2\colon (P, \sigma, f, g) \in \mathcal{D} \ \text{ (this is a } \Sigma_3^0\text{–formula)} \\ &\qquad \wedge\ (P, \sigma, f, g, q) \notin \mathcal{L} \ \wedge\ (P, \sigma, f, g, q) \notin \mathcal{R} \\ \iff\ &\sigma \in \mathbb{S}_P \ \wedge\ \forall\, y_1 \forall\, z_1 \exists\, y_2 \exists\, z_2 \exists x_1 \forall x_2\colon (P, \sigma, f, g) \in \mathcal{D} \ \text{ (this is a } \Pi_3^0\text{–formula)} \\ &\qquad \wedge\ (P, \sigma, f, g, q) \notin \mathcal{L} \ \wedge\ (P, \sigma, f, g, q) \notin \mathcal{R} \end{aligned}$$

Hence, there exists a Σ_3^0–formula and a Π_3^0–formula both defining $\mathcal{COVAR}$ and therefore $\mathcal{COVAR} \in \Sigma_3^0 \cap \Pi_3^0 = \Delta_3^0$. □

Theorem 16 *$\mathcal{COVAR}$ is Π_2^0–hard.*

Proof We reduce the Π_2^0–complete $\mathcal{AST}$ to $\mathcal{COVAR}$. Let (P, σ) be an instance of $\mathcal{AST}$. Consider the reduction function $r(P, \sigma) = \big(P', \sigma, 1, 1, 1/4\big)$, with P' being the program

$$\{\texttt{diverge}\}\ [1/2]\ \{P\}\ .$$

Again, the covariance $\mathrm{Cov}_{P',\sigma}(1, 1)$ is defined and we have

$$\mathrm{Cov}_{P',\sigma}(1, 1) \ = \ \mathrm{E}_{P',\sigma}(1) - \mathrm{E}_{P',\sigma}(1)^2$$

of which the plot is depicted in Figure 2. Recall that $\mathrm{E}_{P',\sigma}(1)$ is exactly the probability of P' terminating on input σ. We can see that $\mathrm{Cov}_{P',\sigma}(1, 1) = \mathrm{E}_{P',\sigma}(1) - \mathrm{E}_{P',\sigma}(1)^2 = \frac{1}{4}$ iff P' terminates with probability $1/2$. Since P' terminates at most with probability $1/2$, we obtain that $\mathrm{Cov}_{P',\sigma}(1, 1) = \frac{1}{4}$ iff P' terminates with probability $1/2$, which is the case iff P terminates almost–surely. Thus $r(P, \sigma) = \big(P', \sigma, 1, 1, \frac{1}{4}\big) \in \mathcal{COVAR}$ iff $(P, \sigma) \in \mathcal{AST}$ and therefore $r\colon \mathcal{AST} \leq_{\mathrm{m}} \mathcal{COVAR}$. Since $\mathcal{AST}$ is Π_2^0–complete, we obtain that $\mathcal{COVAR}$ is Π_2^0–hard. □

Theorem 17 *$\mathcal{COVAR}$ is Σ_2^0–hard.*

Proof We reduce the Σ_2^0–complete $\mathcal{FEXP}$ to $\mathcal{COVAR}$. For that, consider the reduction function $r(P, \sigma, f) = (P, \sigma, f, 0, 0)$.

Correctness of the reduction: The covariance of f and 0 is defined if and only if the expected value of f is finite, the expected value of 0 is finite (which is trivially satisfied), and the expected value of $f \cdot 0 = 0$ is finite (which is again trivially satisfied). In case that this covariance is defined, it is equal to 0 by definition of the covariance of any random variable f and 0. Thus, the covariance is defined and its value is 0 if and only if the expected value of f after executing P on input σ is finite. Hence, $r\colon \mathcal{FEXP} \leq_{\mathrm{m}} \mathcal{COVAR}$. □

Remark 1 (The Hardness of Approximating Variances) Variance approximation is *not easier* than covariance approximation, i.e. the same hardness results as in Theorems 13 through 16 hold for analogous variance approximation problems. In fact, we have always reduced to approximating a variance — the variance of termination — for obtaining our hardness results on covariances. The only exception is Theorem 17, where two different expectations are used.

It can furthermore be seen that variance approximation is *not harder* than covariance approximation The proofs are analogous to the corresponding proofs for covariances presented in this section. The main difference is that we additionally have to consider the case that $\mathrm{E}_{P,\sigma}\big(f^2\big)$ is infinite (otherwise, we have $\mathrm{Var}_{P,\sigma}(f) = \mathrm{Cov}_{P,\sigma}(f, f)$). In this case it suffices to drop the finiteness check for $\mathrm{E}_{P,\sigma}\big(f^2\big)$ from $\mathcal{DCOVAR}$ when approximating lower bounds of variances. This does not change the complexity, because we still have to check that the expected value of f is finite. When approximating upper bounds no change is required: If $\mathrm{E}_{P,\sigma}\big(f^2\big)$ is infinite, so is the variance and no a priori fixed constant q is an upper bound of the variance. For computing exact variances, we proceed as in Theorem 15. △

As an immediate consequence of Theorems 13 and 14, computing both upper and lower bounds for covariances are equally difficult. This is *contrary to the case for expected values*: While computing upper bounds for expected values is Σ^0_2–complete, we have seen that computing lower bounds is "only" Σ^0_1–complete, thus lower bounds are computably enumerable. By this, we can even computably enumerate an ascending sequence that monotonically converges to the sought–after expected value. By Theorems 13 and 14 this is *not possible* for a covariance as Σ^0_2–sets are in general not computably enumerable.

For deciding whether the covariance equals a given rational, we have that this is properly in Δ^0_3 (see Theorem 15) since this problem is both Σ^0_2–hard (Theorem 17) and Π^0_2–hard (Theorem 16). Since there exist *no complete problems in* Δ^0_n*, for* $n \geq 2$ (in the sense of many–one–reducibility) [40, Exercise 14–14, p. 332], it is impossible to establish a completeness result similar to our other hardness results for $\mathcal{COVAR}$.

Our hardness results rule out techniques based on finite loop–unrollings as *complete* approaches for reasoning about the covariances of values of probabilistic programs. An invariant-based approach to overcome this problem is outlined in [22]. This approach also extends to reasoning on runtime variances.

6 Conclusion and Discussion

We have studied the computational complexity of solving a variety of natural problems which appear in the analysis of probabilistic programs: Approximating lower bounds, upper bounds, and exact preexpectations ($\mathcal{LEXP}$, $\mathcal{REXP}$, and $\mathcal{EXP}$), deciding whether a preexpectation is finite ($\mathcal{FEXP}$), deciding non–universal and universal almost–sure termination ($\mathcal{AST}$ and $\mathcal{UAST}$), deciding non–universal and universal positive almost–sure termination ($\mathcal{PAST}$ and $\mathcal{UPAST}$), and approximating lower bounds, upper bounds, and exact covariances ($\mathcal{LCOVAR}$, $\mathcal{RCOVAR}$, and $\mathcal{COVAR}$). Our complexity results are summarized in Figure 3. Lower bounds of preexpectations are computably enumerable. All other problems are strictly more difficult. Each of the examined problems — except for $\mathcal{COVAR}$ — is complete for their respective level of the arithmetical hierarchy. For $\mathcal{COVAR}$ we have established that it is both Σ^0_2– and Π^0_2–hard but in Δ^0_3.

An interesting issue that is raised by our results is the following: We see that universal positive almost–sure termination ($\mathcal{UPAST}$) is computationally strictly harder to decide than universal almost–sure termination ($\mathcal{UAST}$), namely Π^0_3–complete as opposed to Π^0_2–complete. Yet for $\mathcal{UPAST}$ there exist several complete approaches based on supermartingales [7,24]. These approaches seem to work quite well in practice [8] and moreover, they are conceptually quite easy. Techniques for proving $\mathcal{UAST}$, on the other hand, seem much more involved [32]. So while $\mathcal{UPAST}$ is harder to verify than $\mathcal{UAST}$ in theory, it appears to be more approachable in practice.

This issue is not new to us: While $\mathcal{UPAST}$ requires proving an *upper bound* on the expected runtime, $\mathcal{UAST}$ requires proving a (non–strict) *lower bound* (namely 1) on the termination probability. We have encountered a very similar phenomenon in [24]: While proving upper bounds on the expected runtime is conceptually easy, proving lower bounds is much more involved. The same discrepancy seems to apply to lower and upper bounds of expected values: Proving upper bounds is

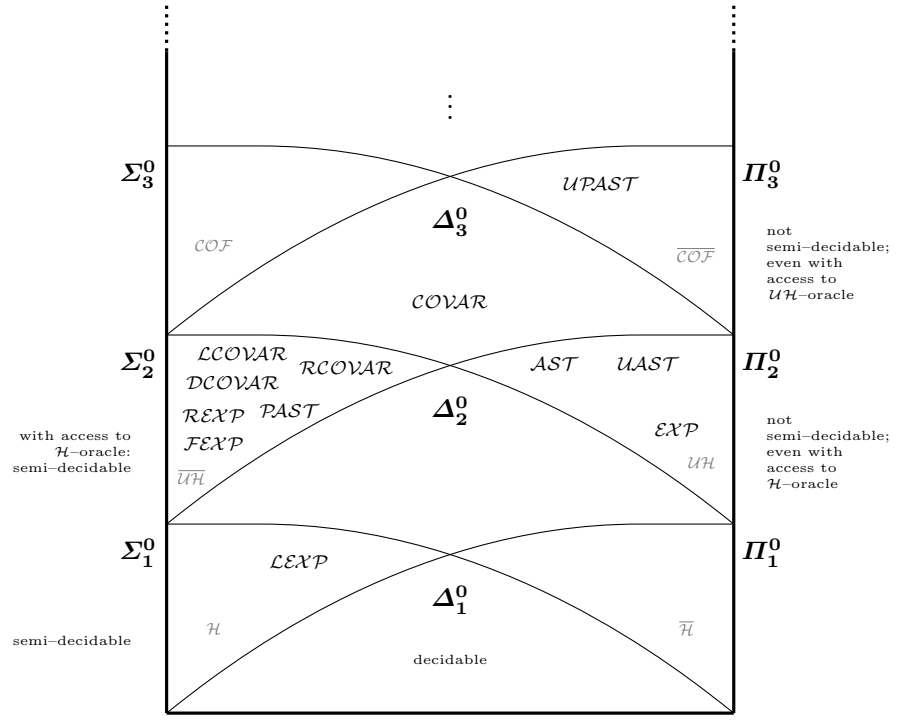


Fig. 3 The complexity landscape of analysis problems for probabilistic programs. All analysis problems–except from the Σ_2^0–hard and Π_2^0–hard problem $\mathcal{COVAR}$–are complete for the respective level in the arithmetical hierarchy.

conceptually easier than proving lower bounds, whereas in theory proving upper bounds should be computationally harder.

While we have no solution to this "paradox", we do believe that the issue is worth investigating. An interesting direction could be to perform for instance for both $\mathcal{UPAST}$ and $\mathcal{UAST}$ a *smoothed analysis of the problem*[14] [4] and see whether this gives valuable insights on this paradox.

Another direction for future research is to investigate the interplay of nondeterministic and probabilistic behavior. We conjecture that for demonic nondetermism, all our results remain valid. For angelic nondetermism, there is evidence that the problems might become harder, i.e. the complexity classes may change [9].

Acknowledgements The authors would like to thank Luis María Ferrer Fioriti (Saarland University), Federico Olmedo (University of Chile), and Wolfgang Thomas (RWTH Aachen University) for the fruitful discussions on the topics of this paper. Furthermore, we acknowledge the valuable and very constructive comments we received from the anonymous referees that lead to substantial improvements of this paper.

[14] As opposed to a smoothed analysis of an algorithm.

References

1. Arons, T., Pnueli, A., Zuck, L.D.: Parameterized Verification by Probabilistic Abstraction. In: FoSSaCS, *LNCS*, vol. 2620, pp. 87–102. Springer (2003)
2. Ash, R.B., Doleans-Dade, C.: Probability and Measure Theory. Academic Press (2000)
3. Barthe, G., Köpf, B., Olmedo, F., Béguelin, S.Z.: Probabilistic Relational Reasoning for Differential Privacy. ACM Trans. Program. Lang. Syst. **35**(3), 9 (2013)
4. Bläser, M., Manthey, B.: Smoothed complexity theory. TOCT **7**(2), 6:1–6:21 (2015)
5. Bournez, O., Garnier, F.: Proving positive almost-sure termination. In: Term Rewriting and Applications, 16th International Conference, RTA 2005, Nara, Japan, April 19-21, 2005, Proceedings, *LNCS*, vol. 3467, pp. 323–337. Springer (2005)
6. Bournez, O., Hoyrup, M.: Rewriting Logic and Probabilities. In: RTA, *LNCS*, vol. 2706, pp. 61–75. Springer (2003)
7. Chakarov, A., Sankaranarayanan, S.: Probabilistic program analysis with martingales. In: CAV, *LNCS*, vol. 8044, pp. 511–526. Springer (2013)
8. Chatterjee, K., Fu, H., Goharshady, A.K.: Termination analysis of probabilistic programs through positivstellensatz's. In: CAV (1), *LNCS*, vol. 9779, pp. 3–22. Springer (2016)
9. Chatterjee, K., Fu, H., Novotný, P., Hasheminezhad, R.: Algorithmic analysis of qualitative and quantitative termination problems for affine probabilistic programs. In: POPL, pp. 327–342. ACM (2016)
10. Davis, M.D.: Computability and Unsolvability. McGraw-Hill Series in Information Processing and Computers. McGraw-Hill (1958)
11. Davis, M.D.: Computability, Complexity, and Languages: Fundamentals of Theoretical Computer Science. Academic Press (1994)
12. Dijkstra, E.W.: A Discipline of Programming. Prentice-Hall Englewood Cliffs (1976)
13. Esparza, J., Gaiser, A., Kiefer, S.: Proving Termination of Probabilistic Programs Using Patterns. In: CAV, *LNCS*, vol. 7358, pp. 123–138. Springer (2012)
14. Fioriti, L.M.F., Hermanns, H.: Probabilistic Termination: Soundness, Completeness, and Compositionality. In: POPL 2015, pp. 489–501. ACM (2015)
15. Foster, N., Kozen, D., Mamouras, K., Reitblatt, M., Silva, A.: Probabilistic NetKAT. In: ESOP, *LNCS*, vol. 9632, pp. 282–309. Springer (2016)
16. Gill, J.: Computational Complexity of Probabilistic Turing Machines. SIAM J. Comput. **6**(4), 675–695 (1977)
17. Gordon, A.D., Henzinger, T.A., Nori, A.V., Rajamani, S.K.: Probabilistic Programming. In: Future of Software Engineering (FOSE), pp. 167–181. ACM (2014)
18. Gretz, F., Katoen, J.P., McIver, A.: Operational versus Weakest Pre–Expectation Semantics for the Probabilistic Guarded Command Language. Performance Evaluation **73**, 110–132 (2014)
19. Hart, S., Sharir, M., Pnueli, A.: Termination of Probabilistic Concurrent Programs. TOPLAS **5**(3), 356–380 (1983)
20. Jansen, N., Kaminski, B.L., Katoen, J.P., Olmedo, F., Gretz, F., McIver, A.: Conditioning in Probabilistic Programming. ENTCS **319**, 199–216 (2015)
21. Kaminski, B.L., Katoen, J.P.: On the Hardness of Almost–Sure Termination. In: Proc. of MFCS 2015, Part I, *LNCS*, vol. 9234, pp. 307–318. Springer (2015)
22. Kaminski, B.L., Katoen, J.P., Matheja, C.: Inferring Covariances for Probabilistic Programs. In: QEST 2016, pp. 191–206 (2016)
23. Kaminski, B.L., Katoen, J.P., Matheja, C., Olmedo, F.: Weakest Precondition Reasoning for Expected Run-Times of Probabilistic Programs. CoRR **abs/1601.01001** (2016). URL http://arxiv.org/abs/1601.01001
24. Kaminski, B.L., Katoen, J.P., Matheja, C., Olmedo, F.: Weakest Precondition Reasoning for Expected Run-Times of Probabilistic Programs. In: ESOP, *LNCS*, vol. 9632, pp. 364–389. Springer (2016)
25. Katoen, J., McIver, A., Meinicke, L., Morgan, C.C.: Linear-Invariant Generation for Probabilistic Programs: Automated Support for Proof-Based Methods. In: SAS, *LNCS*, vol. 6337, pp. 390–406. Springer (2010)
26. Kleene, S.C.: Recursive Predicates and Quantifiers. Trans. of the AMS **53**(1), 41 – 73 (1943)
27. Klenke, A.: Probability Theory: A Comprehensive Course. Springer Science & Business Media (2013)
28. Kozen, D.: Semantics of Probabilistic Programs. J. Comput. Syst. Sci. **22**(3), 328–350 (1981)

29. Kozen, D.: A probabilistic PDL. J. Comput. Syst. Sci. **30**(2), 162–178 (1985)
30. dal Lago, U., Grellois, C.: Probabilistic termination by monadic affine sized typing. In: ESOP, *LNCS*, vol. 10201, pp. 393–419. Springer (2017)
31. McIver, A., Morgan, C.: Abstraction, Refinement and Proof for Probabilistic Systems. Springer (2004)
32. McIver, A., Morgan, C., Kaminski, B.L., Katoen, J.P.: A New Proof Rule for Almost-Sure Termination. In: POPL [to appear] (2018)
33. Morgan, C.: Proof Rules for Probabilistic Loops. In: Proc. of the BCS-FACS 7th Refinement Workshop, Workshops in Computing, p. 7. Springer Verlag (1996)
34. Murawski, A., Ouaknine, J.: On Probabilistic Program Equivalence and Refinement. In: CONCUR, *LNCS*, vol. 3653, pp. 156–170. Springer (2005)
35. Odifreddi, P.: Classical Recursion Theory: The Theory of Functions and Sets of Natural Numbers. Elsevier (1992)
36. Odifreddi, P.: Classical Recursion Theory, Volume II. Elsevier (1999)
37. Papadimitriou, C.H.: Computational complexity. Addison-Wesley (1994)
38. Post, E.L.: Recursively Enumerable Sets of Positive Integers and their Decision Problems. Bulletin of the AMS **50**(5), 284–316 (1944)
39. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley & Sons (2005)
40. Rogers, H.: Theory of Recursive Functions and Effective Computability, vol. 5. McGraw-Hill New York (1967)
41. Sharir, M., Pnueli, A., Hart, S.: Verification of probabilistic programs. SIAM J. Comput. **13**(2), 292–314 (1984). DOI 10.1137/0213021. URL https://doi.org/10.1137/0213021
42. Sneyers, J., de Schreye, D.: Probabilistic Termination of CHRiSM Programs. In: LOPSTR, *LNCS*, vol. 7225, pp. 221–236. Springer (2011)
43. Tiomkin, M.L.: Probabilistic Termination Versus Fair Termination. TCS **66**(3), 333–340 (1989)
44. Ying, M.: Floyd-Hoare Logic for Quantum Programs. ACM Trans. Program. Lang. Syst. **33**(6), 19 (2011)