Inferring Covariances for Probabilistic Programs⋆

Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja

Software Modelling and Verification Group
RWTH Aachen University
{benjamin.kaminski,katoen,matheja}@cs.rwth-aachen.de

Abstract. We study weakest precondition reasoning about the (co)variance of outcomes and the variance of run–times of probabilistic programs with conditioning. For outcomes, we show that approximating (co)variances is computationally more difficult than approximating expected values. In particular, we prove that computing both lower and upper bounds for (co)variances is Σ_2^0–complete. As a consequence, neither lower nor upper bounds are computably enumerable. We therefore present invariant–based techniques that *do* enable enumeration of both upper and lower bounds, once appropriate invariants are found. Finally, we extend this approach to reasoning about run–time variances.

Keywords: probabilistic programs · covariance · run–time

1 Introduction

Probabilistic programs describe manipulations on uncertain data in a succinct way. They are normal–looking programs describing how to obtain a distribution over the outputs. Using mostly standard programming language constructs, a probabilistic program transforms a prior distribution into a posterior distribution. Probabilistic programs provide a structured means to describe e.g., Bayesian networks (from AI), random encryption (from security), or predator–prey models (from biology) [5] succinctly.

The posterior distribution of a program is mostly determined by approximate means such as Markov Chain Monte Carlo (MCMC) sampling using (variants of) the well–known Metropolis–Hasting approach. This yields estimates for various measures of interest, such as expected values, second moments, variances, covariances, and the like. Such estimates typically come with weak guarantees in the form of confidence intervals, asserting that with a certain confidence the measure has a certain value. In contrast to these weak guarantees, we aim at the *exact* inference of such measures and their bounds. We hereby focus both on correctness and on run–time analysis of probabilistic programs. Put shortly, we are interested in obtaining *quantitative* statements about the possible outcomes of programs well as their run times.

⋆ This work was supported by the Excellence Initiative of the German federal and state government.

This paper studies reasoning about the (co)variance of outcomes and the variance of run–times of probabilistic programs. Our programs support sampling from discrete probability distributions, conditioning on the outcomes of experiments by observations [5], and unbounded while–loops[1]. In the first part of the paper, we study the *theoretical complexity* of obtaining (co)variances on outcomes. We show that obtaining bounds on (co)variances is computationally more difficult than for expected values. In particular, we prove that computing both upper *and lower* bounds for (co)variances of program outcomes is Σ_2^0–complete, thus *not recursively enumerable*. This contrasts the case for expected values where lower bounds *are recursively enumerable*, while only upper bounds are Σ_2^0–complete [7]. We also show that determining the precise values of (co)variances as well as checking whether the (co)variance is infinite are both Π_2^0–complete. These results rule out analysis techniques based on finite loop–unrollings as complete approaches for reasoning about the covariances of outcomes of probabilistic programs.

In the second part of the paper, we therefore develop a weakest precondition reasoning technique for obtaining covariances on outcomes and variances on run–times. As with deductive reasoning for ordinary sequential programs, the crux is to find suitable loop–invariants. We present a couple of invariant–based proof rules that provide a sound and complete method to computably enumerate both upper and lower bounds on covariances, once appropriate invariants are found. We establish similar results for variances of the run–time of programs. The results of this paper extend McIver and Morgans approach for obtaining expectations of probabilistic programs [11], recent techniques for expected run–time analysis [9], and complement results on termination analysis [7,4].

Some proofs had to be omitted due to lack of space. They can be found in an extended version of this paper [8].

2 Preliminaries

We study approximating the covariance of two random variables (ranging over program states) after successful termination of a probabilistic program on a given input state. Our development builds upon the *conditional probabilistic guarded command language (cpGCL)* [6]—an extension of Dijkstra's guarded command language [3] endowed with probabilistic choice and conditioning constructs.

Definition 1 (cpGCL [6]). *Let* $\mathbb{V}$ *be a finite* set of program variables[2]*. Then the* set of programs in cpGCL*, denoted* $\mathbb{P}$*, adheres to the grammar*

$$\begin{aligned} \mathbb{P} \quad ::= \quad & \texttt{skip} \;\big|\; \texttt{empty} \;\big|\; \texttt{diverge} \;\big|\; \texttt{halt} \;\big|\; x \coloneqq E \;\big|\; \mathbb{P}\texttt{;}\ \mathbb{P} \;\big|\; \texttt{if}\ (B)\ \{\mathbb{P}\}\ \texttt{else}\ \{\mathbb{P}\} \\ & \big|\; \{\mathbb{P}\}\ [p]\ \{\mathbb{P}\} \;\big|\; \texttt{while}\ (B)\ \{\mathbb{P}\} \;\big|\; \texttt{observe}\ B\ , \end{aligned}$$

[1] This contrasts MCMC–based analysis, as this is restricted to bounded programs.

[2] We restrict ourselves to a finite set of program variables for reasons of cleanness of the presentation. In principle, a countable set of program variables could be allowed.

where $x \in \mathbb{V}$*,* E *is an arithmetical expression over* $\mathbb{V}$*,* $p \in [0,\, 1] \cap \mathbb{Q}$ *is a rational probability, and* B *is a Boolean expression over arithmetic expressions over* $\mathbb{V}$*.*

If a program C *contains neither a probabilistic choice* $\{C'\}\ [p]\ \{C''\}$ *nor an* `observe`*–statement, we say that* C *is* non–probabilistic.

We briefly go over the meaning of the language constructs. Furthermore, we assign each statement an execution time in order to reason about the *run–time* of programs. `skip` (`empty`) does nothing—i.e. does not alter the current variable valuations—and consumes one (no) unit of time. `diverge` is syntactic sugar for the certainly non–terminating program `while` (`true`) $\{$`skip`$\}$. `halt` consumes no unit of time and halts program execution immediately (even when encountered inside a loop). It represents an *improper* termination of the program. $x \coloneqq E$, C_1`;` C_2, `if` (B) $\{C_1\}$ `else` $\{C_2\}$, and `while` (B) $\{C'\}$ are standard variable assignment, sequential composition, conditional choice, and while–loop constructs. Assignments and guard evaluations consume one unit of time.

$\{C_1\}\ [p]\ \{C_2\}$ is a probabilistic choice construct: With probability p the program C_1 is executed and with probability $1 - p$ the program C_2 is executed. Flipping the p–coin itself consumes one unit of time. `observe` B is the conditioning construct. Whenever in the execution of a program, an `observe` B is encountered, such that the current variable valuation satisfies the guard B, nothing happens except that one unit of time is being consumed. If, however, an `observe` B is encountered along an execution trace that occurs with probability q, such that B is *not* satisfied, this trace is blocked as it is considered an *undesired execution*. The probabilities of the remaining execution traces are then conditioned to the fact that this undesired trace was not encountered, i.e. the probabilities of the remaining execution traces are renormalized by $1 - q$. We refer to encountering such an undesired execution as an *observation violation*. For more details on conditioning and its semantics, see [6].

Notice that we do not include non–deterministic choice constructs (as opposed to probabilistic choice construct) in our language, as we would then run into similar problems as in [6, Section 6] in the presence of conditioning.

Example 1 (Conditioning inside a Loop). Consider the following loop:

`while` $(c = 1)\{\ \{c \coloneqq 0\}$ `[0.5]` $\{x \coloneqq x + 1\}$`;` `observe` $c = 1 \ \vee\ x$ is odd $\}$

Without the `observe`–statement, this loop would generate a geometric distribution on x. By considering the `observe`–statement, this distribution is conditioned to the fact that after termination x is odd. △

Given a probabilistic program C, an initial state σ, and a random variable f mapping program states to positive reals, we could now ask: What is the *conditional* expected value of f after proper termination of program C on input σ, *given that no observation was violated during the execution*? An answer to this question is given by the conditional weakest pre–expectation calculus introduced in [6]. For summarizing this calculus, we first formally characterize the random variables f, commonly called *expectations* [11]:

C	$\mathsf{wp}\,[C]\,(f)$	$\mathsf{rt}\,[C]\,(t)$
`skip`	f	$t[\tau/\tau+1]$
`empty`	f	t
`diverge`	$\mathbf{0}$	∞
`halt`	$\mathbf{0}$	$\mathbf{0}$
$x := E$	$f[x/E]$	$t[x/E,\, \tau/\tau+1]$
C_1 `;` C_2	$\mathsf{wp}[C_1] \circ \mathsf{wp}\,[C_2]\,(f)$	$\mathsf{rt}[C_1] \circ \mathsf{rt}\,[C_2]\,(t)$
`if` (B) $\{C_1\}$ `else` $\{C_2\}$	$[B] \cdot \mathsf{wp}\,[C_1]\,(f)$ $+[\neg B] \cdot \mathsf{wp}\,[C_2]\,(f)$	$\big([B] \cdot \mathsf{rt}\,[C_1]\,(t)$ $+[\neg B] \cdot \mathsf{rt}\,[C_2]\,(t)\big)[\tau/\tau+1]$
$\{C_1\}$ $[p]$ $\{C_2\}$	$p \cdot \mathsf{wp}\,[C_1]\,(f)$ $+(1-p) \cdot \mathsf{wp}\,[C_2]\,(f)$	$\big(p \cdot \mathsf{rt}\,[C_1]\,(t)$ $+(1-p) \cdot \mathsf{rt}\,[C_2]\,(t)\big)[\tau/\tau+1]$
`while` (B) $\{C'\}$	$\mathsf{lfp}\, X.\ [\neg B] \cdot f$ $+[B] \cdot \mathsf{wp}\,[C']\,(X)$	$\mathsf{lfp}\, X.\ \big([\neg B] \cdot t$ $+[B] \cdot \mathsf{rt}\,[C']\,(X)\big)[\tau/\tau+1]$
`observe` B	$[B] \cdot f$	$[B] \cdot t[\tau/\tau+1]$
C	$\mathsf{wlp}\,[C]\,(f)$	
`diverge`	$\mathbf{1}$	
`halt`	$\mathbf{1}$	
`while` (B) $\{C'\}$	$\mathsf{gfp}\, X.\ [\neg B] \cdot f + [B] \cdot \mathsf{wlp}\,[C']\,(X)$	

Table 1. Definition of wp, wlp, and rt. $[x/E]$ is a syntactic replacement with $f[x/E]\,(\sigma) = f(\sigma[x \mapsto \sigma(E)])$. $[B]$ is the indicator function of B with $[B](\sigma) = 1$ if $\sigma \models B$, and $[B](\sigma) = 0$ otherwise. $F \circ H(f)$ is the functional composition of F and H applied to f. $\mathsf{lfp}\, X.\ F(X)$ ($\mathsf{gfp}\, X.\ F(X)$) is the least (greatest) fixed point of F with respect to $\preceq$. Definitions of wlp for the other language constructs are as for wp and thus omitted.

Definition 2 (Expectations [11,6]). *Let* $\mathbb{S} = \{\sigma \mid \sigma \colon \mathbb{V} \to \mathbb{Q}\}$, *where* $\mathbb{Q}$ *is the set of rational numbers, be the* set of program states.[3] *Then the* set of expectations *is defined as* $\mathbb{E} = \big\{f \bigm| f \colon \mathbb{S} \to \mathbb{R}^{\infty}_{\geq 0}\big\}$, *and the* set of bounded expectations *is defined as* $\mathbb{E}_{\leq 1} = \{f \mid f \colon \mathbb{S} \to [0,\, 1]\}$. *A* complete partial order $\preceq$ *on both* $\mathbb{E}$ *and* $\mathbb{E}_{\leq 1}$ *is given by* $f_1 \preceq f_2$ *iff* $\forall\, \sigma \in \mathbb{S}\colon f_1(\sigma) \leq f_2(\sigma)$.

The *weakest (liberal) pre–expectation transformer* $\mathsf{wp}\colon \mathbb{P} \to (\mathbb{E} \to \mathbb{E})$ ($\mathsf{wlp}\colon \mathbb{P} \to (\mathbb{E}_{\leq 1} \to \mathbb{E}_{\leq 1})$) is defined according to Table 1 (middle column). By means of these two transformers, we can give an answer to the question posed above: Namely, the fraction $\mathsf{wp}[C](f)(\sigma)/\mathsf{wlp}[C](\mathbf{1})(\sigma)$ is indeed the conditional expected value of f after termination of C on input σ, given that no observation was violated during C's execution [6]. Consequently, we define:

Definition 3 (Conditional Expected Values [6]). *Let* $C \in \mathbb{P}$, $\sigma \in \mathbb{S}$, *and* $f \in \mathbb{E}$. *Then the* conditional expected value *of* f *after executing* C *on input* σ

[3] Notice that $\mathbb{S}$ is countable and computably enumerable as $\mathbb{V}$ is finite.

given that no observation was violated is defined as[4]

$$\mathsf{E}_{[\![C]\!](\sigma)}\left(f\right) \quad = \quad \frac{\mathsf{wp}\left[C\right]\left(f\right)\left(\sigma\right)}{\mathsf{wlp}\left[C\right]\left(\mathbf{1}\right)\left(\sigma\right)} \ .$$

Having the definition for conditional expected values readily available, we can now turn towards defining the conditional (co)variance of a (two) random variables. We simply translate the textbook definition to our setting:

Definition 4 (Conditional (Co)variances). *Let $C \in \mathbb{P}$, $\sigma \in \mathbb{S}$, and $f, g \in \mathbb{E}$. Then the* conditional covariance *of the two random variables f and g after executing C on input σ, given that no observation was violated is defined as*

$$\mathsf{Cov}_{[\![C]\!](\sigma)}\left(f,\, g\right) \quad = \quad \mathsf{E}_{[\![C]\!](\sigma)}\left(f \cdot g\right) - \mathsf{E}_{[\![C]\!](\sigma)}\left(f\right) \cdot \mathsf{E}_{[\![C]\!](\sigma)}\left(g\right) \ .$$

The conditional variance *of the single random variable f after executing C on input σ, given that no observation was violated is defined as the conditional covariance of f with itself, i.e.* $\mathsf{Var}_{[\![C]\!](\sigma)}\left(f\right) = \mathsf{Cov}_{[\![C]\!](\sigma)}\left(f,\, f\right)$.

3 Computational Hardness of Computing (Co)variances

In this section, we will investigate the computational hardness of computing upper and lower bounds for conditional (co)variances. The results will be stated in terms of levels in the arithmetical hierarchy—a concept we first briefly recall:

Definition 5 (The Arithmetical Hierarchy [10,12]). *For every $n \in \mathbb{N}$, the* class Σ_n^0 *is defined as* $\Sigma_n^0 = \big\{\mathcal{A} \;\big|\; \mathcal{A} = \big\{x \;\big|\; \exists y_1\, \forall y_2\, \exists y_3 \cdots \exists/\forall y_n \colon\; (x,\, y_1,\, y_2,\, y_3,\, \ldots,\, y_n) \in \mathcal{R}\big\},\, \mathcal{R}$ is a decidable relation$\big\}$ *and the* class Π_n^0 *is defined as* $\Pi_n^0 = \big\{\mathcal{A} \;\big|\; \mathcal{A} = \big\{x \;\big|\; \forall y_1\, \exists y_2\, \forall y_3 \cdots \exists/\forall y_n \colon\; (x,\, y_1,\, y_2,\, y_3,\, \ldots,\, y_n) \in \mathcal{R}\big\},\, \mathcal{R}$ is a decidable relation$\big\}$. *Note that we require the values of variables to be drawn from a computable domain.* Multiple consecutive quantifiers of the same type *can be contracted to* one *quantifier of that type, so the number n really refers to the number of necessary* quantifier alternations. *A set $\mathcal{A}$ is called* arithmetical, *iff $\mathcal{A} \in \Gamma_n^0$, for $\Gamma \in \{\Sigma,\, \Pi\}$ and $n \in \mathbb{N}$. The arithmetical sets form a strict hierarchy, i.e. $\Gamma_n^0 \subset \Gamma_{n+1}^0$ holds for $\Gamma \in \{\Sigma,\, \Pi\}$ and $n \geq 0$. Furthermore, note that $\Sigma_0^0 = \Pi_0^0$ is exactly the class of the decidable sets and Σ_1^0 is exactly the class of the computably enumerable sets.*

Next, we recall the concept of many–one reducibility and completeness:

Definition 6 (Many–One Reducibility and Completeness [12,14,2]). *Let $\mathcal{A}$, $\mathcal{B}$ be arithmetical sets and let X be some appropriate universe such that*

[4] We make use of the convention that $\frac{0}{0} = 0$. Note that since our probabilistic choice is a discrete choice and our language does not support sampling from continuous distributions, the problematic case of "$\frac{0}{0}$" can only occur if executing C on input σ will result in a violation of an observation with probability 1.

$\mathcal{A}, \mathcal{B} \subseteq X$. $\mathcal{A}$ is called many–one reducible *(or simply* reducible*) to $\mathcal{B}$, denoted $\mathcal{A} \leq_{\mathrm{m}} \mathcal{B}$, iff there exists a computable function $r \colon X \to X$, such that $\forall\, x \in X \colon \big(x \in \mathcal{A} \iff r(x) \in \mathcal{B}\big)$. If r is a function such that r reduces $\mathcal{A}$ to $\mathcal{B}$, we denote this by $r \colon \mathcal{A} \leq_{\mathrm{m}} \mathcal{B}$. Note that $\leq_{\mathrm{m}}$ is transitive.*

$\mathcal{A}$ is called Γ_n^0–complete*, for $\Gamma \in \{\Sigma, \Pi\}$, iff both $\mathcal{A} \in \Gamma_n^0$ and $\mathcal{A}$ is Γ_n^0*–hard*, meaning $\mathcal{C} \leq_{\mathrm{m}} \mathcal{A}$, for any set $\mathcal{C} \in \Gamma_n^0$. Note that if $\mathcal{B} \in \Gamma_n^0$ and $\mathcal{A} \leq_{\mathrm{m}} \mathcal{B}$, then $\mathcal{A} \in \Gamma_n^0$, too. Furthermore, note that if $\mathcal{A}$ is Γ_n^0–complete and $\mathcal{A} \leq_{\mathrm{m}} \mathcal{B}$, then $\mathcal{B}$ is necessarily Γ_n^0–hard. Lastly, note that if $\mathcal{A}$ is Σ_n^0–complete, then $\mathcal{A} \in \Sigma_n^0 \setminus \Pi_n^0$. Analogously, if $\mathcal{A}$ is Π_n^0–complete, then $\mathcal{A} \in \Pi_n^0 \setminus \Sigma_n^0$.*

In the following, we study the hardness of obtaining covariance approximations both from above and from below. Furthermore, we are interested in exact values of covariances as well as in deciding whether the covariance is infinite. In order to formally investigate the arithmetical complexity of these problems, we define four problem sets which relate to upper and lower bounds for covariances and to the question whether the covariance is infinite:

Definition 7 (Approximation Problems for Covariances). *We define the following decision problems:*

$$
\begin{aligned}
(C,\, \sigma,\, f,\, g,\, q) \in \mathcal{LCOVAR} &\iff \mathsf{Cov}_{[\![C]\!](\sigma)}\,(f,\, g) > q \\
(C,\, \sigma,\, f,\, g,\, q) \in \mathcal{RCOVAR} &\iff \mathsf{Cov}_{[\![C]\!](\sigma)}\,(f,\, g) < q \\
(C,\, \sigma,\, f,\, g,\, q) \in \mathcal{COVAR} &\iff \mathsf{Cov}_{[\![C]\!](\sigma)}\,(f,\, g) = q \\
(C,\, \sigma,\, f,\, g) \in {}^{\infty}\mathcal{COVAR} &\iff \mathsf{Cov}_{[\![C]\!](\sigma)}\,(f,\, g) \in \{-\infty,\, +\infty\}
\end{aligned}
$$

where $C \in \mathbb{P}$, $\sigma \in \mathbb{S}$, $f, g \in \mathbb{E}$, and $q \in \mathbb{Q}$.[5]

The first fact we establish about the hardness of computing upper and lower bounds of covariances is that this is at most Σ_2^0–hard, thus not harder than deciding whether a non–probabilistic program, i.e. a program without observations and probabilistic choice, does *not* terminate on all inputs, or deciding whether a probabilistic program terminates after an expected finite number of steps [13,7]. Formally, we establish the following results:

Lemma 1. *$\mathcal{LCOVAR}$ and $\mathcal{RCOVAR}$ are both in Σ_2^0.*

For proving Lemma 1, we revert to a fact established in [7]: All lower bounds for expected outcomes are computably enumerable. As a consequence, there exists a computable function $\mathsf{wp}^k\,[C]\,(f)\,(\sigma)$ that is ascending in k, such that for given $C \in \mathbb{P}$, $\sigma \in \mathbb{S}$, and $f \in \mathbb{E}$, we have

$$
\forall\, k \in \mathbb{N} \colon \mathsf{wp}^k\,[C]\,(f)\,(\sigma) \;\leq\; \mathsf{wp}\,[C]\,(f)\,(\sigma), \quad \text{and}
$$
$$
\sup_{k \in \mathbb{N}} \mathsf{wp}^k\,[C]\,(f)\,(\sigma) \;=\; \mathsf{wp}\,[C]\,(f)\,(\sigma)\;.
$$

Intuitively, for every $k \in \mathbb{N}$ the function $\mathsf{wp}^k\,[C]\,(f)\,(\sigma)$ outputs a lower bound of $\mathsf{wp}\,[C]\,(f)\,(\sigma)$ in ascending order.

[5] Note that, for obvious reasons, we restrict to *computable* expectations f, g only.

Similarly, lower bounds for $\mathsf{wlp}\,[C]\,(\mathbf{1})\,(\sigma)$ can be enumerated. To see this, note that $\mathsf{wp}\,[C]\,(\mathbf{1})\,(\sigma) = 1$ for any `observe`–free program C and any state σ. $\mathsf{wp}\,[C]\,(\mathbf{1})\,(\sigma)$ can only be decreased by violation of an observation. Informally,

$$\mathsf{wp}\,[C]\,(\mathbf{1})\,(\sigma) \;=\; 1 - \text{“Probability of } C \text{ violating an observation”}\ .$$

Lower bounds for the latter probability can be enumerated by successively exploring the computation tree of C on input σ and accumulating the probability mass of all execution traces that lead to a violation of an observation. As a consequence, there must exist a computable function $\mathsf{wlp}^k\,[C]\,(\mathbf{1})\,(\sigma)$ that is descending in k, such that for given $C \in \mathbb{P}$ and $\sigma \in \mathbb{S}$,

$$\begin{aligned} \forall\, k \in \mathbb{N}\colon \mathsf{wlp}\,[C]\,(\mathbf{1})\,(\sigma) \;&\leq\; \mathsf{wlp}^k\,[C]\,(\mathbf{1})\,(\sigma), \quad \text{and} \\ \mathsf{wlp}\,[C]\,(\mathbf{1})\,(\sigma) \;&=\; \inf_{k\in\mathbb{N}} \mathsf{wlp}^k\,[C]\,(\mathbf{1})\,(\sigma)\ . \end{aligned}$$

Since $\mathsf{wp}^k\,[C]\,(f)\,(\sigma)$ is ascending and $\mathsf{wlp}^k\,[C]\,(\mathbf{1})\,(\sigma)$ is descending in k, the quotient ${}^{\mathsf{wp}^k[C](f)(\sigma)}/{}_{\mathsf{wlp}^k[C](\mathbf{1})(\sigma)}$ is ascending in k. We can now prove Lemma 1:

Proof (Lemma 1). For $\mathcal{LCOVAR} \in \Sigma_2^0$, consider $(C,\, \sigma,\, f,\, g,\, q) \in \mathcal{LCOVAR}$ iff

$$\exists\, k\ \forall\, \ell\colon\ \frac{\mathsf{wp}^k\,[C]\,(f\cdot g)\,(\sigma)}{\mathsf{wlp}^k\,[C]\,(\mathbf{1})\,(\sigma)} - \frac{\mathsf{wp}^\ell\,[C]\,(f)\,(\sigma)\cdot \mathsf{wp}^\ell\,[C]\,(g)\,(\sigma)}{\mathsf{wlp}^\ell\,[C]\,(\mathbf{1})\,(\sigma)^2} \;>\; q\ .$$

For the proof for $\mathcal{RCOVAR}$, see [8] □

Regarding the hardness of deciding whether a given rational is equal to the covariance and the hardness of deciding non–finiteness of covariances, we establish that this is at most Π_2^0–hard, thus not harder than deciding whether a non–probabilistic program terminates on all inputs, or deciding whether a probabilistic program does *not* terminate after an expected finite number of steps [13,7]. Formally, we establish the following:

Lemma 2. *$\mathcal{COVAR}$ and ${}^\infty\mathcal{COVAR}$ are both in Π_2^0.*

So far we provided upper bounds for the computational hardness of solving approximation problems for covariances. We now show that these bounds are tight in the sense that these problems are *complete* for their respective level of the arithmetical hierarchy. For that we need a Σ_2^0– and a Π_2^0–hard problem in order to perform the necessary reductions for proving the hardness results. Adequate problems are the problem of almost–sure termination and its complement:

Theorem 1 (Hardness of the Almost–Sure Termination Problem [7]). *Let $C \in \mathbb{P}$ be* `observe`*–free. Then C* terminates almost–surely *on input $\sigma \in \mathbb{S}$, iff it does so with probability* 1. *The problem set $\mathcal{AST}$ is defined as $(C,\, \sigma) \in \mathcal{AST}$ iff C terminates almost–surely on input σ. We denote the complement of $\mathcal{AST}$ by $\overline{\mathcal{AST}}$.*[6] *$\mathcal{AST}$ is Π_2^0–complete and $\overline{\mathcal{AST}}$ is Σ_2^0–complete.*

[6] Note that by “complement” we mean not exactly a set theoretic complement but rather all pairs $(C,\, \sigma)$ such that C does not terminate almost–surely on σ.

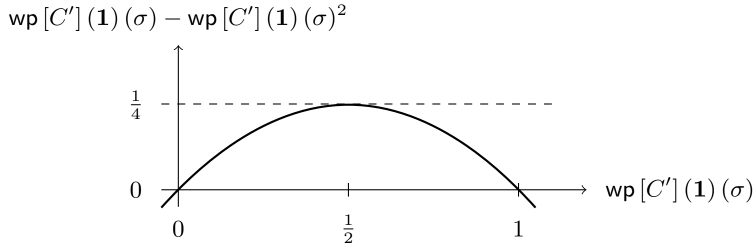

Fig. 1. Plot of the termination probability of a program against the resulting variance.

By reduction from $\overline{\mathcal{AST}}$ we now establish the following hardness results:

Lemma 3. *$\mathcal{LCOVAR}$ and $\mathcal{RCOVAR}$ are both Σ_2^0–hard.*

Proof. For proving the Σ_2^0–hardness of $\mathcal{LCOVAR}$, consider the reduction function $r_{\mathcal{L}}(C, \sigma) = (C', \sigma, v, v, 0)$[7], with $C' = v \coloneqq 0\texttt{;}\ \{\texttt{skip}\}\,[1/2]\,\{C\}\texttt{;}\ v \coloneqq 1$, where variable v does not occur in C. Now consider the following:

$$\begin{aligned}
\mathsf{Cov}_{[\![C']\!](\sigma)}\,(v,\,v) &= \frac{\mathsf{wp}\,[C']\,\big(v^2\big)\,(\sigma)}{\mathsf{wlp}\,[C']\,(\mathbf{1})\,(\sigma)} - \frac{\mathsf{wp}\,[C']\,(v)\,(\sigma)^2}{\mathsf{wlp}\,[C']\,(\mathbf{1})\,(\sigma)^2} \\
&= \frac{\mathsf{wp}\,[C']\,\big(v^2\big)\,(\sigma)}{1} - \frac{\mathsf{wp}\,[C']\,(v)\,(\sigma)^2}{1^2} \qquad (C' \text{ is } \texttt{observe}\text{–free}) \\
&= \mathsf{wp}\,[C']\,\big(v^2\big)\,(\sigma) - \mathsf{wp}\,[C']\,(v)\,(\sigma)^2
\end{aligned}$$

Since v does not occur in C and v is set from 0 to 1 if and only if C' has terminated, this is equal to:

$$\begin{aligned}
&= \mathsf{wp}\,[C']\,\big(\mathbf{1}^2\big)\,(\sigma) - \mathsf{wp}\,[C']\,(\mathbf{1})\,(\sigma)^2 \\
&= \mathsf{wp}\,[C']\,(\mathbf{1})\,(\sigma) - \mathsf{wp}\,[C']\,(\mathbf{1})\,(\sigma)^2
\end{aligned}$$

Note that $\mathsf{wp}\,[C']\,(\mathbf{1})\,(\sigma)$ is exactly the probability of C' terminating on input σ. A plot of this termination probability against the resulting variance is given in Figure 1. We observe that $\mathsf{Cov}_{[\![C']\!](\sigma)}\,(v,\,v) = \mathsf{wp}\,[C']\,(\mathbf{1})\,(\sigma) - \mathsf{wp}\,[C']\,(\mathbf{1})\,(\sigma)^2 > 0$ iff C' terminates *neither* with probability 0 *nor* with probability 1. Since, however, C' terminates by construction *at least* with probability $1/2$, we obtain that $\mathsf{Cov}_{[\![C']\!](\sigma)}\,(v,\,v) > 0$ iff C' terminates with probability less than 1, which is the case iff C terminates with probability less than 1. Thus $r_{\mathcal{L}}(C, \sigma) = (C', \sigma, v, v, 0) \in \mathcal{LCOVAR}$ iff $(C, \sigma) \in \overline{\mathcal{AST}}$. Thus, $r_{\mathcal{L}}\colon \overline{\mathcal{AST}} \leq_{\mathrm{m}} \mathcal{LCOVAR}$. Since $\overline{\mathcal{AST}}$ is Σ_2^0–complete, if follows that $\mathcal{LCOVAR}$ is Σ_2^0–hard.

For the the proof for $\mathcal{RCOVAR}$, see [8]. □

A hardness results for $\mathcal{COVAR}$ is obtained by reduction from $\mathcal{AST}$.

Lemma 4. *$\mathcal{COVAR}$ is Π_2^0–hard.*

[7] We write v for the expectation that in state σ returns $\sigma(v)$.

Proof. Similar to Lemma 3 using $r_{\mathcal{V}}(C, \sigma) = \big(C', \sigma, v, v, \frac{1}{4}\big)$, with $C' = v \coloneqq 0$; $\{\texttt{diverge}\}$ $[1/2]$ $\{C\}$; $v \coloneqq 1$. For details, see [8]. □

For a hardness result on ${}^{\infty}\mathcal{COVAR}$ we use the universal halting problem for non–probabilistic programs.

Theorem 2 (Hardness of the Universal Halting Problem [13]). *Let C be a non–probabilistic program. The* universal halting problem *is the problem of deciding whether C terminates on all inputs. Let $\mathcal{UH}$ denote the* problem set*, defined as $C \in \mathcal{UH}$ iff $\forall \sigma \in \mathbb{S}\colon C$ terminates on input σ. $\mathcal{UH}$ is Π^0_2–complete.*

We now establish by reduction from $\mathcal{UH}$ the remaining hardness result:

Lemma 5. *${}^{\infty}\mathcal{COVAR}$ is Π^0_2–hard.*

Proof. For proving the Π^0_2–hardness of ${}^{\infty}\mathcal{COVAR}$ we use the reduction function $r_\infty(C) = (C', \sigma, v, v)$, where σ is arbitrary but fixed and C' is the program

$c \coloneqq 1$; $i \coloneqq 0$; $x \coloneqq 0$; $v \coloneqq 0$; $\mathit{term} \coloneqq 0$; InitC;
`while` $(c \neq 0)\{$
  StepC; `if` $(\mathit{term} = 1)\{$ $v \coloneqq 2^x$; $i \coloneqq i+1$; $\mathit{term} \coloneqq 0$; InitC $\}$;
  $\{c \coloneqq 0\}$ `[0.5]` $\{c \coloneqq 1\}$; $x \coloneqq x+1$ $\}$,

where *InitC* is a non–probabilistic program that initializes a simulation of the program C on input $e(i)$ (where $e\colon \mathbb{N} \to \mathbb{S}$ is some computable enumeration of $\mathbb{S}$), and *StepC* is a non–probabilistic program that does one single (further) step of that simulation and sets *term* to 1 if that step has led to termination of C.

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

The variable x keeps track of the number of loop iterations (starting from 1 as the first loop iteration will definitely take place), which equals the number of coin tosses. The x–th loop iteration takes place with probability $1/2^x$. The expected value $\mathsf{E}_{[\![C']\!](\sigma)}(v)$ is thus given by a series of the form $S = \sum_{i=1}^{\infty} v_i/2^i$, where $v_i = 2^j$ for some $j \in \mathbb{N}$. Two cases arise:

(1) $C \in \mathcal{UH}$, i.e. C terminates on every input. In that case, v will infinitely often be updated to 2^x. Therefore, summands of the form $2^i/2^i$ will appear infinitely often in S and so S diverges. Hence, the expected value of v is infinity and therefore, the variance of v must be infinite as well. Thus, $(C', \sigma, v, v) \in {}^{\infty}\mathcal{COVAR}$.

(2) $C \notin \mathcal{UH}$, i.e. there exists some input σ' with minimal $i \in \mathbb{N}$ such that $e(i) = \sigma'$ on which C does not terminate. In that case, the numerator of all summands of S is upper bounded by some constant 2^j and thus S converges. Boundedness of the v_i's implies that the series $\sum_{i=1}^{\infty} {v_i}^2/2^i = \mathsf{E}_{[\![C']\!](\sigma)}\big(v^2\big)$ also converges. Hence, the variance of v is finite and $(C', \sigma, v, v) \notin {}^{\infty}\mathcal{COVAR}$. □

Lemmas 1 to 5 together directly yield the following completeness results:

Theorem 3 (The Hardness of Approximating Covariances).

1. *$\mathcal{LCOVAR}$ and $\mathcal{RCOVAR}$ are both Σ_2^0–complete.*
2. *$\mathcal{COVAR}$ and ${}^{\infty}\mathcal{COVAR}$ are both Π_2^0–complete.*

Remark 1 (The Hardness of Approximating Variances). It can be shown that *variance* approximation is not easier than covariance approximation: exactly the same completeness results as in Theorem 3 hold for analogous variance approximation problems. In fact, we have always reduced to approximating a variance for obtaining our hardness results on covariances. △

As an immediate consequence of Theorem 3, computing both upper and lower bounds for covariances is equally difficult. This is *contrary to the case for expected values*: While computing upper bounds for expected values is also Σ_2^0–complete, computing lower bounds is Σ_1^0–complete, thus lower bounds are computably enumerable [7]. Therefore we can computably enumerate an ascending sequence that converges to the sought–after expected value. By Theorem 3 this is *not possible* for a covariance as Σ_2^0–sets are in general not computably enumerable.

Theorem 3 rules out techniques based on finite loop–unrollings as *complete* approaches for reasoning about the covariances of outcomes of probabilistic programs. As this is a rather sobering insight, in the next section we will investigate invariant–aided techniques that are complete and can be applied to tackle these approximation problems.

4 Invariant–Aided Reasoning on Outcome Covariances

For straight–line (i.e. loop–free) programs, upper and lower bounds for covariances are obviously computable, e.g. by using the decompositions from Definitions 3 and 4, and the inference rules from Table 1. Problems do arise, however, for loops. We have seen in the previous section that neither upper nor lower bounds are computably enumerable. In this section we therefore present an invariant–aided approach for enumerating bounds on covariances of loops. The underlying principle of such techniques is quite commonly a result due to Park:

Theorem 4 (Park's Lemma [15]). *Let $(D, \sqsubseteq)$ be a complete partial order and $F \colon D \to D$ be continuous. Then, for all $d \in D$, it holds that $F(d) \sqsubseteq d$ implies $\mathsf{lfp}\, F \sqsubseteq d$, and $d \sqsubseteq F(d)$ implies $d \sqsubseteq \mathsf{gfp}\, F$.*

Using this theorem, we can verify in a relatively easy fashion that some element is an over–approximation of the least fixed point or an under–approximation of the greatest fixed point of a continuous mapping on a complete partial order. In the following, let $C = \texttt{while}\ (B)\ \{C'\}$. In order to exploit Park's Lemma for enumerating bounds on covariances for this while–loop, recall

$$\mathsf{Cov}_{\llbracket C \rrbracket(\sigma)}\,(f,\, g) \quad = \quad \mathsf{E}_{\llbracket C \rrbracket(\sigma)}\,(f \cdot g) - \mathsf{E}_{\llbracket C \rrbracket(\sigma)}\,(f) \cdot \mathsf{E}_{\llbracket C \rrbracket(\sigma)}\,(g)$$

$$= \frac{\mathsf{wp}\,[C]\,(f\cdot g)\,(\sigma)}{\mathsf{wlp}\,[C]\,(\mathbf{1})\,(\sigma)} - \frac{\mathsf{wp}\,[C]\,(f)\,(\sigma)\cdot \mathsf{wp}\,[C]\,(g)\,(\sigma)}{\mathsf{wlp}\,[C]\,(\mathbf{1})\,(\sigma)^2}\;.$$

By inspection of the last line, we can see that for obtaining an over–approximation of $\mathsf{Cov}_{[\![C]\!](\sigma)}\,(f,\,g)$, it suffices to over–approximate ${}^{\mathsf{wp}[C'](f\cdot g)(\sigma)}/_{\mathsf{wlp}[C'](\mathbf{1})(\sigma)}$, which can be done by over–approximating $\mathsf{wp}\,[C']\,(f\cdot g)\,(\sigma)$ and under–approximating $\mathsf{wlp}\,[C']\,(\mathbf{1})\,(\sigma)$. Since wp (wlp) of a loop is defined in terms of a least (greatest) fixed point, we can apply Park's Lemma for over–approximating this fraction. This leads us to the following proof rule:

Theorem 5 (Invariant–Aided Over–Approximation of Covariances). *Let $C = \mathtt{while}\ (B)\ \{C'\}$, $\sigma \in \mathbb{S}$, $f, g \in \mathbb{E}$, $F_h(X) = [\neg B]\cdot h + [B]\cdot \mathsf{wp}\,[C']\,(X)$, for any $h \in \mathbb{E}$, and $G(Y) = [\neg B] + [B]\cdot \mathsf{wlp}\,[C']\,(Y)$. Furthermore, let $\widehat{X} \in \mathbb{E}$ and $\widehat{Y} \in \mathbb{E}_{\leq 1}$, such that $F_{f\cdot g}\big(\widehat{X}\big) \preceq \widehat{X}$, $\widehat{Y} \preceq G\big(\widehat{Y}\big)$, and $\widehat{Y}(\sigma) > 0$. Then for all $k \in \mathbb{N}$ it holds that*[8]

$$\mathsf{Cov}_{[\![C]\!](\sigma)}\,(f,\,g) \;\;\leq\;\; \frac{\widehat{X}(\sigma)}{\widehat{Y}(\sigma)} - \frac{F_f^k(\mathbf{0})(\sigma)\cdot F_g^k(\mathbf{0})(\sigma)}{G^k(\mathbf{1})(\sigma)^2}\;.$$

By this method we can computably enumerate upper bounds for covariances once appropriate invariants are found. The catch is that if we choose the invariants, such that $F_{f\cdot g}\big(\widehat{X}\big)(\sigma) < \widehat{X}(\sigma)$ or $\widehat{Y}(\sigma) < G\big(\widehat{Y}\big)(\sigma)$, then the enumeration will *not* get arbitrarily close to the actual covariance. Note, however, that our method is complete since we could have chosen $\widehat{X} = \mathsf{lfp}\,F_{f\cdot g}$ and $\widehat{Y} = \mathsf{gfp}\,G$:

Corollary 1 (Completeness of Theorem 5). *Let $C = \mathtt{while}\ (B)\ \{C'\}$, $\sigma \in \mathbb{S}$, $f, g \in \mathbb{E}$. Then there exist $\widehat{X} \in \mathbb{E}$ and $\widehat{Y} \in \mathbb{E}_{\leq 1}$, such that*

$$\inf_{k\in\mathbb{N}}\; \frac{\widehat{X}(\sigma)}{\widehat{Y}(\sigma)} - \frac{F_f^k(\mathbf{0})(\sigma)\cdot F_g^k(\mathbf{0})(\sigma)}{G^k(\mathbf{1})(\sigma)^2} \;\;=\;\; \mathsf{Cov}_{[\![C]\!](\sigma)}\,(f,\,g)\,.$$

By considerations analogous to the ones above, we can formulate dual results for lower bounds. For details, see [8].

Example 2 (Application of Theorem 5). Reconsider the loop from Example 1. For reasoning about the variance of x, we pick the invariants

$$\begin{aligned}\widehat{X} \;&=\; [c\neq 0]\cdot x^2 + [c=1]\cdot \big([x\text{ is even}]\cdot {}^1/_{27}\left(9x^2+30x+41\right) \\ &\qquad\qquad + [x\text{ is odd}]\cdot {}^2/_{27}\left(9x^2+12x+20\right)\big)\,, \quad \text{and} \\ \widehat{Y} \;&=\; [c\neq 0] + [c=1]\cdot \left([x\text{ is even}]\cdot {}^1/_3 + [x\text{ is odd}]\cdot {}^2/_3\right)\;,\end{aligned}$$

which satisfy the preconditions of Theorem 5. If we enter the loop in a state σ with $\sigma(c) = 1$ and $\sigma(x) = 0$, we have ${}^{\widehat{X}(\sigma)}/_{\widehat{Y}(\sigma)} = {}^{41}/_9$ which is our first upper bound. We can now enumerate further upper bounds by doing fixed point iteration on $F_x(X) = [c \neq 1]\cdot x + [c = 1]\cdot \mathsf{wp}\,[\textit{loop body}]\,(X) = [c \neq 1]\,\cdot$

[8] Here $F_h^k(X)$ stands for k–fold application of F_h to X.

$x + [c = 1] \cdot \frac{1}{2}\big([x \text{ is odd}] \cdot X[c/0] + X[x/x+1]\big)$ and $G(Y) = [c \neq 1] + [c = 1] \cdot \mathsf{wlp}\,[\mathit{loop\ body}]\,(Y) = [c \neq 1] + [c = 1] \cdot \frac{1}{2}\big([x \text{ is odd}] \cdot Y[c/0] + Y[x/x+1]\big)$:

$$\frac{41}{9} - \frac{F_x^1(\mathbf{0})(\sigma)^2}{G^1(\mathbf{1})(\sigma)^2} = \frac{41}{9} - \frac{F_x^2(\mathbf{0})(\sigma)^2}{G^2(\mathbf{1})(\sigma)^2} = \frac{41}{9}, \qquad \frac{41}{9} - \frac{F_x^3(\mathbf{0})(\sigma)^2}{G^3(\mathbf{1})(\sigma)^2} = \frac{37}{9}, \qquad \ldots$$

Finally, this sequence converges to $41/9 - 25/9 = 16/9$ as the variance of x. $\triangle$

5 Reasoning about Run–Time Variances

In addition to the (co)variance of outcomes we are interested in the variance of the program's *run–time*. Intuitively, the run–time of a program corresponds to its number of executed operations, where each operation is weighted according to some run–time model. For simplicity, our run–time model assumes `skip`, guard evaluations and assignments to consume one unit of time. Other statements are assumed to consume no time at all. More elaborated run–time models, e.g. in which the run–time of assignments depends on the size of a given expression, are possible design choices that can easily be integrated in our formalization.

We describe the run–time variance in terms of an operational model Markov Chain (MC) with rewards. The model is similar to the ones studied in [6,9], but additionally keeps track of the run–time in a dedicated variable τ which is *not accessible by the program*, but may occur in expectations.

Definition 8 (Run–Time Expectations). *Let* $\mathbb{S}_\tau = \{\sigma \mid \mathbb{V} \uplus \{\tau\} \to \mathbb{Q}\}$. *The* set of run–time expectations *is then defined as* $\mathbb{E}_\tau = \big\{t \bigm| t\colon \mathbb{S}_\tau \to \mathbb{R}_{\geq 0}^{\infty}\big\}$.

A corresponding wp–style calculus to reason about expected run–times and variances of probabilistic programs is presented afterwards.

We first briefly recall some necessary notions about MCs and refer to [1, Ch. 10] for a comprehensive introduction. A *Markov Chain* is a tuple $\mathcal{M} = (\mathcal{S}, \mathbf{P}, s_I, \mathit{rew})$, where $\mathcal{S}$ is a countable set of *states*, $s_I \in \mathcal{S}$ is the initial state, $\mathbf{P} : \mathcal{S} \times \mathcal{S} \to [0,1]$ is the *transition probability function* such that for each state $s \in \mathcal{S}$, $\sum_{s' \in \mathcal{S}} \mathbf{P}(s, s') \in \{0,1\}$, and $\mathit{rew} : \mathcal{S} \to \mathbb{R}_{\geq 0}$ is a *reward function*. Instead of $\mathbf{P}(s,s') = p$, we often write $s \xrightarrow{p} s'$. A *path* in $\mathcal{M}$ is a finite or infinite sequence $\pi = s_0 s_1 \ldots$ such that $s_i \in S$ and $\mathbf{P}(s_i, s_{i+1}) > 0$ for each $i \geq 0$ (where we tacitly assume $\mathbf{P}(s_i, s_{i+1}) = 0$ if π is a finite path of length n and $i \geq n$). The *cumulative reward* and the probability of a finite path $\hat{\pi} = s_0 \ldots s_n$ are given by $\mathit{rew}(\hat{\pi}) = \sum_{k=0}^{n-1} \mathit{rew}(s_k)$ and $\Pr^{\mathcal{M}}\{\hat{\pi}\} = \prod_{k=0}^{n-1} \mathbf{P}(s_k, s_{k+1})$. These notions are lifted to infinite paths by the standard cylinder set construction (cf. [1]).

Given a set of target states $T \subseteq \mathcal{S}$, $\Diamond T$ denotes the set of all paths in $\mathcal{M}$ reaching a state in T from initial state s_I. Analogously, all paths starting in s_I that never reach a state in T are denoted by $\neg\Diamond T$. The *expected reward* that $\mathcal{M}$ eventually reaches T from a state $s \in \mathcal{S}$ is defined as follows:

$$\mathsf{ExpRew}^{\mathcal{M}}\,(\Diamond T) \;=\; \begin{cases} \sum_{\pi \in \Diamond T} \Pr^{\mathcal{M}}\{\pi\} \cdot \mathit{rew}(\pi) & \text{if } \sum_{\pi \in \Diamond T} \Pr^{\mathcal{M}}\{\pi\} = 1 \\ \infty & \text{if } \sum_{\pi \in \Diamond T} \Pr^{\mathcal{M}}\{\pi\} < 1. \end{cases}$$

Moreover, the *conditional expected reward* of $\mathcal{M}$ reaching T from s under the condition that a set of undesired states $U \subseteq \mathcal{S}$ is never reached is given by[9]

$$\mathsf{CExpRew}^{\mathcal{M}}(\Diamond T \mid \neg \Diamond U) \;=\; \frac{\mathsf{ExpRew}^{\mathcal{M}}(\Diamond T \cap \neg \Diamond U)}{\Pr^{\mathcal{M}}\{\neg \Diamond U\}}.$$

We are now in a position to define an operational model for our probabilistic programming language $\mathbb{P}$. Let $\downarrow$ and $\lightning$ be two special symbols denoting successful termination of a program and failure of an observation, respectively.

Definition 9 (The Operational MC of a $\mathbb{P}$–Program). *Given a program $C \in \mathbb{P}$, an initial program state $\sigma_0 \in \mathbb{S}_\tau$ and a post–run–time $t \in \mathbb{E}$, the according* MC *is given by $\mathcal{M}^{t}_{\sigma_0}[C] = (\mathcal{S}, \mathbf{P}, s_I, rew)$, where*

- $\mathcal{S} = ((\mathbb{P} \cup \{\downarrow\} \cup \{\downarrow; C \mid C \in \mathbb{P}\}) \times \mathbb{S}_\tau) \;\cup\; \{\langle \mathrm{sink} \rangle, \langle \lightning \rangle\}$,
- *the transition probability function* $\mathbf{P}$ *is given by the rules in Figure 2,*
- $s_I = \langle C, \sigma_0 \rangle$, *and*
- $rew : \mathcal{S} \to \mathbb{R}_{\geq 0}$ *is the reward function defined by* $rew(s) = t(\sigma)$ *if* $s = \langle \downarrow, \sigma \rangle$ *for some* $\sigma \in \mathbb{S}_\tau$ *and* $rew(s) = 0$, *otherwise.*

In this construction, $\sigma_0(\tau)$ represents the *post–execution time* of a program, i.e. the run–time that is added after a program finishes its execution. Hence, τ precisely captures the run–time of a program if $\sigma_0(\tau) = 0$. The rules presented in Figure 2 defining the transition probability function are mostly self–explanatory. Since we assume guard evaluations, probabilistic choices, assignments and the statement `skip` to consume one unit of time. Hence, τ is incremented accordingly for each of these statements and remains untouched otherwise.

Figure 3 sketches the structure of the operational MC $\mathcal{M}^{t}_{\sigma}[C]$. Here, clouds represent a set of states and squiggly arrows indicate that a set of states is reachable by one or more paths. Each run either terminates successfully (i.e. it visits some state $\langle \downarrow, \sigma' \rangle$), or violates an observation (i.e. it visits $\langle \lightning \rangle$), or diverges. In the first two cases each run eventually ends up in the $\langle \mathrm{sink} \rangle$ state. Note that states of the form $\langle \downarrow, \sigma' \rangle$ are the only ones that may have a positive reward. Furthermore, each of the auxiliary states of the form $\langle \downarrow, \sigma' \rangle$, $\langle \lightning \rangle$ and $\langle \mathrm{sink} \rangle$ is needed to properly deal with `diverge`, `halt` and `observe` B.

Since τ precisely captures the run–time of a program if τ is initially set to 0, the *expected run–time* of executing $C \in \mathbb{P}$ on input $\sigma \in \mathbb{S}_\tau$ with $\sigma(\tau) = 0$ is given by the conditional expected reward of $\mathcal{M}^{\tau}_{\sigma}[C]$ reaching $\langle \mathrm{sink} \rangle$, given that no observation fails, i.e. $\mathsf{E}_{[\![C]\!](\sigma)}(\tau) = \mathsf{CExpRew}^{\mathcal{M}^{\tau}_{\sigma}[C]}(\Diamond \langle \mathrm{sink} \rangle \mid \neg \Diamond \langle \lightning \rangle)$. Then, in compliance with Definition 4, the *run–time variance* $\mathsf{RTVar}_{[\![C]\!](\sigma)}$ of $C \in \mathbb{P}$ in state $\sigma \in \mathbb{S}_\tau$ with $\sigma(\tau) = 0$ is given by $\mathsf{E}_{[\![C]\!](\sigma)}\left(\tau^2\right) \;-\; \left(\mathsf{E}_{[\![C]\!](\sigma)}(\tau)\right)^2$ which is

$$\mathsf{CExpRew}^{\mathcal{M}^{\tau^2}_{\sigma}[C]}(\Diamond \langle \mathrm{sink} \rangle \mid \neg \Diamond \langle \lightning \rangle) - \left(\mathsf{CExpRew}^{\mathcal{M}^{\tau}_{\sigma}[C]}(\Diamond \langle \mathrm{sink} \rangle \mid \neg \Diamond \langle \lightning \rangle)\right)^2 \;.$$

In the following we provide a corresponding wp–style calculus to reason about expected run–times and run–time variances of probabilistic programs. A formal

[9] Again, we stick to the convention that $\frac{0}{0} = 0$.

$$\frac{}{\langle \downarrow, \sigma \rangle \xrightarrow{1} \langle \mathrm{sink} \rangle}\ \textsf{[terminated]} \qquad \frac{}{\langle \mathrm{sink} \rangle \xrightarrow{1} \langle \mathrm{sink} \rangle}\ \textsf{[sink]}$$

$$\frac{}{\langle \texttt{empty}, \sigma \rangle \xrightarrow{1} \langle \downarrow, \sigma \rangle}\ \textsf{[empty]} \qquad \frac{}{\langle \texttt{skip}, \sigma \rangle \xrightarrow{1} \langle \downarrow, \sigma[\tau/\tau+1] \rangle}\ \textsf{[skip]}$$

$$\frac{}{\langle \texttt{halt}, \sigma \rangle \xrightarrow{1} \langle \mathrm{sink} \rangle}\ \textsf{[halt]} \qquad \frac{}{\langle x := E, \sigma \rangle \xrightarrow{1} \langle \downarrow, \sigma[x/E,\ \tau/\tau+1] \rangle}\ \textsf{[assgn]}$$

$$\frac{\langle C_1, \sigma \rangle \xrightarrow{p} \langle C_1', \sigma' \rangle \quad 0 < p \leq 1}{\langle C_1\,\texttt{;}\ C_2, \sigma \rangle \xrightarrow{p} \langle C_1'\,\texttt{;}\ C_2, \sigma' \rangle}\ \textsf{[seq-1]} \qquad \frac{}{\langle \downarrow\texttt{;}\ C_2, \sigma \rangle \xrightarrow{1} \langle C_2, \sigma \rangle}\ \textsf{[seq-2]}$$

$$\frac{}{\langle \{C_1\}\ [p]\ \{C_2\}, \sigma \rangle \xrightarrow{p} \langle C_1, \sigma[\tau/\tau+1] \rangle}\ \textsf{[pc-1]}$$

$$\frac{}{\langle \{C_1\}\ [p]\ \{C_2\}, \sigma \rangle \xrightarrow{1-p} \langle C_2, \sigma[\tau/\tau+1] \rangle}\ \textsf{[pc-2]}$$

$$\frac{[B](\sigma) = 1}{\langle \texttt{if}\ (B)\ \{C_1\}\ \texttt{else}\ \{C_2\}, \sigma \rangle \xrightarrow{1} \langle C_1, \sigma[\tau/\tau+1] \rangle}\ \textsf{[if-true]}$$

$$\frac{[B](\sigma) = 0}{\langle \texttt{if}\ (B)\ \{C_1\}\ \texttt{else}\ \{C_2\}, \sigma \rangle \xrightarrow{1} \langle C_2, \sigma[\tau/\tau+1] \rangle}\ \textsf{[if-false]}$$

$$\frac{}{\langle \texttt{while}\ (B)\ \{C\}, \sigma \rangle \xrightarrow{1} \langle \texttt{if}\ (B)\ \{C\texttt{;}\ \texttt{while}\ (B)\ \{C\}\}\ \texttt{else}\ \{\texttt{empty}\}, \sigma \rangle}\ \textsf{[while]}$$

$$\frac{}{\langle \texttt{diverge}, \sigma \rangle \xrightarrow{1} \langle \texttt{diverge}, \sigma \rangle}\ \textsf{[diverge]}$$

$$\frac{[B](\sigma) = 1}{\langle \texttt{observe}\ B, \sigma \rangle \xrightarrow{1} \langle \downarrow, \sigma[\tau/\tau+1] \rangle}\ \textsf{[observe-true]}$$

$$\frac{[B](\sigma) = 0}{\langle \texttt{observe}\ B, \sigma \rangle \xrightarrow{1} \langle \lightning \rangle}\ \textsf{[observe-false]} \qquad \frac{}{\langle \lightning \rangle \xrightarrow{1} \langle \mathrm{sink} \rangle}\ \textsf{[observe-failed]}$$

Fig. 2. Rules for defining the transition probability function of the MC of a $\mathbb{P}$–program.

definition of the *run–time transformer* $\mathsf{rt}\colon \mathbb{P} \to (\mathbb{E}_\tau \to \mathbb{E}_\tau)$ is provided in Table 1 (rightmost column). Intuitively, it behaves like wp except that a *dedicated run–time variable* τ is updated accordingly for each program statement that consumes time. In [9], a transformer for expected run–times without the need for an additional variable τ is studied. However, this approach fails when reasoning about run–time variances since it fails to capture expected squared run–times. The run–time transformer rt precisely captures the notion of expected run–time of our operational model.

Theorem 6 (Operational–Denotational Correspondence). *Let $C \in \mathbb{P}$, $t \in \mathbb{E}_\tau$, and $\sigma \in \mathbb{S}_\tau$. Then*

$$\mathsf{CExpRew}^{\mathcal{M}_\sigma^t[C]}\left(\Diamond\langle \mathrm{sink} \rangle \mid \neg\Diamond\langle \lightning \rangle\right) \;=\; \frac{\mathsf{rt}\,[C]\,(t)\,(\sigma)}{\mathsf{wlp}\,[C]\,(1)\,(\sigma)}.$$

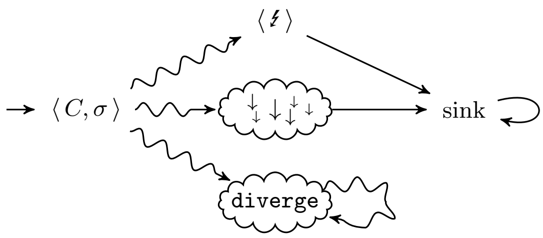

Fig. 3. Schematic depiction of the structure of the operational MC $\mathcal{M}_\sigma^t\,[C]$.

As a result of Theorem 6 we immediately obtain a formal definition of the run–time variance of probabilistic programs in terms of rt and wlp. Formally, the *run–time variance* of $C \in \mathbb{P}$ in state $\sigma \in \mathbb{S}_\tau$ with $\sigma(\tau) = 0$ is given by

$$\begin{aligned} \mathsf{RTVar}_{[\![C]\!](\sigma)} \;&=\; \mathsf{CExpRew}^{\mathcal{M}_\sigma^{\tau^2}[C]}\left(\Diamond\langle\,\mathrm{sink}\,\rangle \mid \neg\Diamond\langle\,\lightning\,\rangle\right) \\ &\qquad - \left(\mathsf{CExpRew}^{\mathcal{M}_\sigma^{\tau}[C]}\left(\Diamond\langle\,\mathrm{sink}\,\rangle \mid \neg\Diamond\langle\,\lightning\,\rangle\right)\right)^2 \\ &=\; \frac{\mathsf{rt}\,[C]\left(\tau^2\right)(\sigma)}{\mathsf{wlp}\,[C]\,(\mathbf{1})\,(\sigma)} \;-\; \frac{\left(\mathsf{rt}\,[C]\,(\tau)\,(\sigma)\right)^2}{\left(\mathsf{wlp}\,[C]\,(\mathbf{1})\,(\sigma)\right)^2}. \end{aligned}$$

Since rt is continuous (cf. [8] for a formal proof), the invariant–aided approach based on Park's Lemma (Theorem 4) presented in Section 4 is applicable to approximate run–time variances as well. We present the result for approximating upper bounds only. The dual result for lower bounds is obtained analogously.

Theorem 7 (Invariant–Aided Over–Approximation of Run–Time Variances). *Let* $C = \texttt{while}\,(B)\,\{C'\}$ *and* $\sigma \in \mathbb{S}_\tau$ *with* $\sigma(\tau) = 0$. *Moreover, let* $F_h(X) = [\neg B]\cdot h + [B]\cdot \mathsf{rt}\,[C']\,(X)$, *and* $G(Y) = [\neg B] + [B]\cdot \mathsf{wlp}\,[C']\,(Y)$. *Furthermore, let* $\widehat{X} \in \mathbb{E}_\tau$ *and* $\widehat{Y} \in \mathbb{E}_{\leq 1}$, *such that* $F_{\tau^2}\big(\widehat{X}\big) \preceq \widehat{X}$, $\widehat{Y} \preceq G\big(\widehat{Y}\big)$, *and* $\widehat{Y}(\sigma) > 0$. *Then for each* $k \in \mathbb{N}$, *it holds*

$$\mathsf{RTVar}_{[\![C]\!](\sigma)} \;\leq\; \frac{\widehat{X}(\sigma)}{\widehat{Y}(\sigma)} \;-\; \left(\frac{F_\tau^k(\mathbf{0})(\sigma)}{G^k(\mathbf{1})(\sigma)}\right)^2.$$

The proof of Theorem 7 is analogous to the proof of Theorem 5. Again, since it is always possible to choose $\widehat{X} = \mathsf{lfp}\,F_{\tau^2}$ and $\widehat{Y} = \mathsf{gfp}\,G$, Theorem 7 is complete, i.e. there exist $\widehat{X} \in \mathbb{E}_\tau$ and $\widehat{Y} \in \mathbb{E}_{\leq 1}$ such that

$$\inf_{k\in\mathbb{N}} \frac{\widehat{X}(\sigma)}{\widehat{Y}(\sigma)} - \left(\frac{F_\tau^k(\mathbf{0})(\sigma)}{G^k(\mathbf{1})(\sigma)}\right)^2 \;=\; \mathsf{RTVar}_{[\![C]\!](\sigma)}.$$

6 Conclusion

We have studied the computational hardness of obtaining both upper and lower bounds on (co)variance of outcomes and established that this is Σ_2^0–complete.

Thus neither upper nor lower bounds are computably enumerable. Furthermore, we have established that deciding whether the (co)variance equals a given rational and deciding whether the covariance is infinite is Π_2^0–complete.

In the second part of the paper, we continued by presenting a sound and complete invariant–aided approach which allows to computably enumerate upper and lower bounds on (co)variances of while–loops, once appropriate loop–invariants are found. Finally, we have shown how this approach can be extended to reason about the variance of run–times.

References

1. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press (2008)
2. Davis, M.D.: Computability, Complexity, and Languages: Fundamentals of Theoretical Computer Science. Academic Press (1994)
3. Dijkstra, E.W.: A Discipline of Programming. Prentice-Hall Englewood Cliffs (1976)
4. Fioriti, L.M.F., Hermanns, H.: Probabilistic termination: Soundness, completeness, and compositionality. In: POPL 2015. pp. 489–501. ACM (2015)
5. 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)
6. Jansen, N., Kaminski, B.L., Katoen, J.P., Olmedo, F., Gretz, F., McIver, A.: Conditioning in probabilistic programming. ENTCS 319, 199–216 (2015)
7. 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)
8. Kaminski, B.L., Katoen, J.P., Christoph, M.: Inferring Covariances for Probabilistic Programs. ArXiv e-prints (Jun 2016)
9. Kaminski, B.L., Katoen, J.P., Matheja, C., Olmedo, F.: Weakest precondition reasoning for expected run–times of probabilistic programs. In: Proc. of ESOP 2016 (2016), (to appear)
10. Kleene, S.C.: Recursive Predicates and Quantifiers. Trans. of the AMS 53(1), 41 – 73 (1943)
11. McIver, A., Morgan, C.: Abstraction, Refinement and Proof for Probabilistic Systems. Springer (2004)
12. Odifreddi, P.: Classical Recursion Theory: The Theory of Functions and Sets of Natural Numbers. Elsevier (1992)
13. Odifreddi, P.: Classical Recursion Theory, Volume II. Elsevier (1999)
14. Post, E.L.: Recursively Enumerable Sets of Positive Integers and their Decision Problems. Bulletin of the AMS 50(5), 284–316 (1944)
15. Wechler, W.: Universal Algebra for Computer Scientists, EATCS Monographs on Theoretical Computer Science, vol. 25. Springer (1992)