Analyzing Expected Outcomes and (Positive) Almost–Sure Termination of Probabilistic Programs is Hard*

Benjamin Lucien Kaminski**

Software Modeling and Verification Group
RWTH Aachen University
benjamin.kaminski@cs.rwth-aachen.de

Probabilistic programs [1] are imperative sequential programs with the ability to toss a biased coin and proceed their computations depending on the outcome of the coin toss. They are used in security to describe cryptographic constructions (such as randomized encryption) and security experiments [2], in machine learning to describe distribution functions that are analyzed using Bayesian inference [3], and in randomized algorithms. They are typically just a small number of lines, but hard to understand and analyze, let alone algorithmically. We consider two major analysis problems for probabilistic programs:

1. Computing expected outcomes: Is the expected outcome of a program (variable) smaller, equal, or larger than a given rational number?
2. Deciding (positive) almost–sure termination [4]: Does a program terminate with probability one (in an expected finite number of computation steps)?

Regarding almost–sure termination, the majority of the literature does either not consider the hardness of the problem or states that it must be somewhat harder to decide than the classical termination problem, as arithmetical instead of topological reasoning is needed (see e.g. [5, 6]).

In our work, we strive to give a *precise* classification of the computational and arithmetical complexity of solving the aforementioned analysis problems. Most of the results summarized here together with proofs can be found in [7].

Preliminaries: Our classifications will be in terms of levels in the arithmetical hierarchy [8]—a concept which we first briefly recall: For any $n \in \mathbb{N}$ the *class* Σ^0_n is defined as $\Sigma^0_n = \{\mathcal{A} \mid \mathcal{A} = \{\boldsymbol{x} \mid \exists y_1\ \forall y_2\ \exists y_3\ \cdots\ \exists/\forall y_n\colon\ (\boldsymbol{x}, y_1, y_2, y_3, \ldots, y_n) \in \mathcal{R}\},\ \mathcal{R} \text{ is a decidable relation}\}\}$, the *class* Π^0_n is defined as $\Pi^0_n = \{\mathcal{A} \mid \mathcal{A} = \{\boldsymbol{x} \mid \forall y_1\ \exists y_2\ \forall y_3\ \cdots\ \exists/\forall y_n\colon\ (\boldsymbol{x}, y_1, y_2, y_3, \ldots, y_n) \in \mathcal{R}\},\ \mathcal{R} \text{ is a decidable relation}\}\}$, and the *class* Δ^0_n is defined as $\Delta^0_n = \Sigma^0_n \cap \Pi^0_n$. We implicitly always quantify over $\mathbb{Q}^+$ and by the boldface $\boldsymbol{x}$'s we mean tuples over $\mathbb{Q}^+$. In a formula, multiple consecutive quantifiers of the same type can be contracted to *one* quantifier of that type, so n really refers to the number of *quantifier*

* This research is funded by the Excellence Initiative of the German federal and state governments and by the EU FP7 MEALS project.

** I thank Joost-Pieter Katoen and Luis María Ferrer Fioriti for many valuable discussions and comments.

alternations rather than to the number of quantifiers actually used in a formula. A set $\mathcal{A}$ is called *arithmetical*, iff $\mathcal{A} \in \Gamma_n^0$, for some $\Gamma \in \{\Sigma, \Pi, \Delta\}$ and some $n \in \mathbb{N}$. The arithmetical sets form a strict hierarchy, i.e. $\Delta_n^0 \subsetneq \Sigma_n^0, \Pi_n^0 \subsetneq \Delta_{n+1}^0$ for all $n \geq 1$. We have that $\Sigma_0^0 = \Pi_0^0 = \Delta_0^0 = \Delta_1^0$ is precisely the class of decidable sets and Σ_1^0 is precisely the class of recursively enumerable sets.

Next we recall the concept of many–one reducibility and the concept of completeness [8]: Let $\mathcal{A}$, $\mathcal{B}$ be arithmetical sets and let X be some appropriate universe, such that $\mathcal{A}, \mathcal{B} \subseteq X$. $\mathcal{B}$ is called *many–one–reducible* to $\mathcal{A}$, denoted $\mathcal{B} \leq_{\mathrm{m}} \mathcal{A}$, iff there exists a computable function $f\colon X \to X$, such that $\forall \boldsymbol{x} \in X\colon \big(\boldsymbol{x} \in \mathcal{B} \iff f(\boldsymbol{x}) \in \mathcal{A}\big)$. Intuitively this means that it is computationally at least as hard to solve problem $\mathcal{A}$ as it is to solve problem $\mathcal{B}$. $\mathcal{A}$ is called Γ_n^0*–complete*, for $\Gamma \in \{\Sigma, \Pi, \Delta\}$, iff both $\mathcal{A} \in \Gamma_n^0$ and $\mathcal{A}$ is Γ_n^0*–hard*, meaning $\mathcal{B} \leq_{\mathrm{m}} \mathcal{A}$, for any set $\mathcal{B} \in \Gamma_n^0$. The universal halting problem (does a classical, i.e. non–probabilistic, program terminate on any given input?), denoted $\mathcal{UH}$, for instance, is Π_2^0–complete whereas its complement, denoted $\overline{\mathcal{UH}}$, is Σ_2^0–complete [9].

Notice that if $\mathcal{B}$ is Γ_n^0–complete and $\mathcal{B} \leq_{\mathrm{m}} \mathcal{A}$ then $\mathcal{A}$ is Γ_n^0–hard.Another important fact about Σ_n^0– and Π_n^0–complete sets is that they are in some sense the most complicated sets in Σ_n^0 and Π_n^0, respectively. Formally, this can be expressed as follows: 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 $\mathcal{A} \in \Pi_n^0 \setminus \Sigma_n^0$. This implies in particular that if $\mathcal{A}$ is Σ_n^0–complete or Π_n^0–complete then $\mathcal{A} \notin \Delta_n^0$.

Probabilistic Programs: We study analysis problems for pGCL–like probabilistic programs à la McIver & Morgan [10]—an extension of Dijkstra's GCL programs [11]. If v stands for a program variable, e for an arithmetical expression over program variables, and $p \in [0, 1] \subset \mathbb{Q}$ for a probability, then the programs that we consider adhere to the following grammar:

$$P \quad \longrightarrow \quad v := e \;\mid\; P\,;\, P \;\mid\; \{P\}\,[p]\,\{P\} \;\mid\; \texttt{WHILE}\,(b)\,\{P\}$$

While assignment, concatenation, and the while–loop are classical programming language constructs, the command $\{P_1\}\,[p]\,\{P_2\}$ represents a probabilistic choice between the two programs P_1 and P_2. With probability p the program P_1 is executed, while with probability $1-p$ the program P_2 is executed. While classical programs upon termination yield a variable valuation, probabilistic programs instead yield a subdistribution over variable valuations where the missing probability mass is the probability of non–termination. For example, the program

$$x := 0\,;\, \{c := 0\}\,[1/2]\,\{c := 1\};\; \texttt{WHILE}\,(c{=}1)\,\big\{x := x{+}1\,;\, \{c := 0\}\,[1/2]\,\{c := 1\}\big\}$$

terminates with probability 1 and establishes for c the valuation 0 with probability 1 and for x a geometric distribution over the valuations 0, 1, 2, ...

Expected Outcomes: The first analysis problem we consider is to approximate expected outcomes. By the expected outcome of a variable v we mean the value we expect v to have after the program has been executed. In the above example

program, for instance, the expected outcome of x is 1 as $\frac{1}{2}\cdot 0+\frac{1}{4}\cdot 1+\frac{1}{8}\cdot 2+\cdots = 1$. Formally, the expected outcome of a variable v after executing the program P starting with variable valuation η, denoted $\mathrm{E}_{P,\eta}(v)$, can be expressed by

$$\mathrm{E}_{P,\eta}(v) \;=\; \sum_{i=1}^{\infty}\sum_{j=0}^{\infty} W\big(P,\,\eta,\,v,\,i,\,h(j)\big)\;,$$

where W is a computable function that simulates the program P on η for i computation steps while resolving all probabilistic choices that occur during simulation according to a sequence $h(j) \in \{L,\,R\}^*$ which encodes whether to take the *L*eft or the *R*ight branch when the simulation encounters a probabilistic choice. Note that a computable enumeration h of all words over $\{L,\,R\}$ exists. If P terminates after exactly i computation steps with exactly $|h(j)|$ probabilistic choices, then W returns the resulting valuation of the variable v multiplied with the probability with which the probabilistic choices encoded in $h(j)$ would have been made, otherwise 0. Such a function W can be obtained by a slight adaption of the Kleene Normal Form Theorem to probabilistic programs [12]. With the above formula we sum over all possible execution steps and all possible resolutions of the probabilistic choices and sum up all resulting valuations of v weighted with their according probabilities.

For approximations of expected outcomes from below, we define the set $\mathcal{LEXP}$ by $(P,\,\eta,\,v,\,q) \in \mathcal{LEXP}$ iff $q < \mathrm{E}_{P,\eta}(v)$. A tuple $(P,\,\eta,\,v,\,q)$ is in $\mathcal{LEXP}$ if the expected outcome of variable v after executing P starting in η is strictly larger than the rational q. This set can be defined by a Σ_1^0–formula, namely

$$(P,\,\eta,\,v,\,q) \in \mathcal{LEXP} \iff \exists\,\hat{\imath}\,\exists\,\hat{\jmath}\colon q \;<\; \sum_{i=1}^{\hat{\imath}}\sum_{j=0}^{\hat{\jmath}} W\big(P,\,\eta,\,v,\,i,\,h(j)\big)\;,$$

and is therefore recursively enumerable. This means that arbitrarily close rational approximations from below can effectively be enumerated.

For approximations from above the set $\mathcal{REXP}$ is defined by $(P,\,\eta,\,v,\,q) \in \mathcal{REXP}$ iff $q > \mathrm{E}_{P,\eta}(v)$. If $\mathcal{REXP}$ was recursively enumerable, arbitrarily close rational approximations from below *and from above* could effectively be enumerated and expected outcomes would therefore fall into the class of computable reals. We can, however, prove that $\mathcal{REXP}$ is not only in Σ_2^0, defined by

$$(P,\,\eta,\,v,\,q) \in \mathcal{REXP} \iff \exists\,\delta\;\forall\,\hat{\imath}\;\forall\,\hat{\jmath}\colon q-\delta \;>\; \sum_{i=1}^{\hat{\imath}}\sum_{j=0}^{\hat{\jmath}} W\big(P,\,\eta,\,v,\,i,\,h(j)\big)\;,$$

but is also Σ_2^0–complete. We can prove the Σ_2^0–hardness of $\mathcal{REXP}$ by showing $\overline{\mathcal{UH}} \leq_{\mathrm{m}} \mathcal{REXP}$. The set $\mathcal{REXP}$ would be recursively enumerable, if there was access to an oracle for the halting problem (does a classical program halt on a certain fixed input?) [8].

Deciding whether some rational q *equals* the expected outcome of v is even harder: The according set, denoted $\mathcal{EXP}$, is defined as $(P,\,\eta,\,v,\,q) \in \mathcal{EXP}$ iff

$q = \mathrm{E}_{P,\eta}(v)$. This set is in Π^0_2. To see this, consider that $(P, \eta, v, q) \in \mathcal{EXP}$ means that both $(P, \eta, v, q) \notin \mathcal{LEXP}$ and $(P, \eta, v, q) \notin \mathcal{REXP}$. By negating the defining formulas for $\mathcal{LEXP}$ and $\mathcal{REXP}$ we obtain a Π^0_1– and a Π^0_2–formula, respectively. The conjunction of these two formulas gives a Π^0_2–formula defining $\mathcal{EXP}$. By showing $\mathcal{UH} \leq_{\mathrm{m}} \mathcal{EXP}$ we can also prove the Π^0_2–hardness of $\mathcal{EXP}$ and therefore we can establish that $\mathcal{EXP}$ is Π^0_2–complete. This means that determining exact expected outcomes is *not semi–decidable, even if there was access to an oracle for the halting problem.*

Almost–Sure Termination: The probability that P terminates on input η, denoted $\mathrm{Pr}_{P,\eta}(\downarrow)$, can be expressed by

$$\mathrm{Pr}_{P,\eta}(\downarrow) \;=\; \sum_{i=1}^{\infty} \sum_{j=0}^{\infty} T\big(P, \eta, i, h(j)\big) \;,$$

where T is a computable function similar to W, but instead of returning a variable valuation multiplied with the probability of the probabilistic choices encoded in $h(j)$, T returns only those probabilities. We say that a program P *terminates almost–surely* on input η iff $\mathrm{Pr}_{P,\eta}(\downarrow) = 1$. The according set $\mathcal{AST}$ is defined as $(P, \eta) \in \mathcal{AST}$ iff $\mathrm{Pr}_{P,\eta}(\downarrow) = 1$. By showing $\mathcal{AST} \leq_{\mathrm{m}} \mathcal{EXP}$ we can prove that $\mathcal{AST} \in \Pi^0_2$ and by showing $\mathcal{UH} \leq_{\mathrm{m}} \mathcal{AST}$ we can establish that $\mathcal{AST}$ is also Π^0_2–hard and hence Π^0_2–complete.

For ordinary programs, we have that the universal halting problem is Π^0_2–complete, whereas the non–universal version of the halting problem is only Σ^0_1–complete, thus computationally less hard to solve. Interestingly, for almost–sure termination this is not the case: The set of all universally almost–surely terminating programs, denoted $\mathcal{UAST}$, is defined by $P \in \mathcal{UAST}$ iff $\forall \eta\colon \mathrm{Pr}_{P,\eta}(\downarrow) = 1$. By showing $\mathcal{AST} \leq_{\mathrm{m}} \mathcal{UAST}$ we can prove that $\mathcal{UAST}$ is Π^0_2–hard. Furthermore, we can express $\mathcal{UAST}$ using $\mathcal{AST}$, namely by

$$P \in \mathcal{UAST} \iff \forall \eta\colon (P, \eta) \in \mathcal{AST} \;,$$

thus $\mathcal{UAST}$ is in the universal closure of $\mathcal{AST}$. As Π^0_n is closed under universal quantification for any $n \geq 1$ and $\mathcal{AST} \in \Pi^0_2$, we have that $\mathcal{UAST} \in \Pi^0_2$ and hence $\mathcal{UAST}$ is also Π^0_2–complete.

Positive Almost–Sure Termination: Along the lines of [13], we can express the expected termination time of P on input η, denoted $\mathrm{E}_{P,\eta}(\downarrow)$, by

$$\mathrm{E}_{P,\eta}(\downarrow) \;=\; \sum_{k=1}^{\infty} \left(1 - \sum_{i=1}^{k-1} \sum_{j=0}^{k-1} T\big(P, \eta, i, h(j)\big) \right) \;.$$

We say that a program P *terminates positively almost–surely* on input η iff $\mathrm{E}_{P,\eta}(\downarrow) < \infty$, i.e. the expected number of steps until termination is finite. The according set, denoted $\mathcal{PAST}$, is defined as $(P, \eta) \in \mathcal{PAST}$ iff $\mathrm{E}_{P,\eta}(\downarrow) < \infty$. Notice that if the expected number of steps until termination is finite, then the

program also terminates almost–surely. However, there exist programs that *do* terminate almost–surely but *not* positively, i.e. we have that $\mathcal{PAST} \subsetneq \mathcal{AST}$. $\mathcal{PAST}$ is computationally more benign than $\mathcal{AST}$, as it can be defined by

$$(P,\,\eta) \in \mathcal{PAST} \iff \exists\, c\, \forall\, \hat{k}\colon \sum_{k=1}^{\hat{k}} \left(1 - \sum_{i=1}^{k-1} \sum_{j=0}^{k-1} T\big(P,\,\eta,\,i,\,h(j)\big) \right) \;<\; c\ ,$$

and we thus have $\mathcal{PAST} \in \Sigma_2^0$. We can furthermore show $\overline{\mathcal{UH}} \leq \mathcal{PAST}$ and therefore establish that $\mathcal{PAST}$ is Σ_2^0–complete. The result of this reduction is somewhat counterintuitive as each classical program that *does not* terminate on some input is transformed by a computable function f into a probabilistic program that *does* terminate in an expected finite amount of steps, whereas the same f transforms a program that *does* terminate on all inputs into a program that *does not* terminate in an expected finite amount of steps.

The set corresponding to the universal version of $\mathcal{PAST}$, denoted $\mathcal{UPAST}$, is defined by $P \in \mathcal{UPAST}$ iff $\forall \eta\colon (P,\,\eta) \in \mathcal{PAST}$. As the universal closure of a set in Σ_n^0 is in Π_{n+1}^0 for any $n \geq 1$ and $\mathcal{PAST} \in \Sigma_2^0$, we have that $\mathcal{UPAST} \in \Pi_3^0$. Furthermore consider the Π_3^0–complete set $\overline{\mathcal{COF}}$ of *classical* programs defined by $P \in \overline{\mathcal{COF}}$ iff P does not terminate on infinitely many inputs [9]. By showing $\overline{\mathcal{COF}} \leq_{\mathrm{m}} \mathcal{UPAST}$, we can prove the Π_3^0–completeness of $\mathcal{UPAST}$ and thus $\mathcal{UPAST}$ is even harder to solve than $\mathcal{AST}$.

References

1. Kozen, D.: Semantics of Probabilistic Programs. J. Comput. Syst. Sci. **22**(3) (1981) 328–350
2. Barthe, G., Köpf, B., Olmedo, F., Béguelin, S.Z.: Probabilistic relational reasoning for differential privacy. ACM Trans. Program. Lang. Syst. **35**(3) (2013) 9
3. Borgström, J., Gordon, A., Greenberg, M., Margetson, J., van Gael, J.: Measure Transformer Semantics for Bayesian Machine Learning. LMCS **9**(3) (2013)
4. Hart, S., Sharir, M., Pnueli, A.: Termination of Probabilistic Concurrent Programs. TOPLAS **5**(3) (1983) 356–380
5. Morgan, C.: Proof Rules for Probabilistic Loops. In: Proceedings of the BCS-FACS 7th Conference on Refinement. FAC-RW'96, Swinton, UK, British Computer Society (1996) 10
6. Esparza, J., Gaiser, A., Kiefer, S.: Proving Termination of Probabilistic Programs Using Patterns. In: CAV. Volume 7358 of LNCS., Springer (2012) 123–138
7. Kaminski, B.L., Katoen, J.P.: Analyzing Expected Outcomes and Almost-Sure Termination of Probabilistic Programs is Hard. ArXiv e-prints (October 2014)
8. Odifreddi, P.: Classical Recursion Theory. Elsevier (1992)
9. Odifreddi, P.: Classical Recursion Theory, Volume II. Elsevier (1999)
10. McIver, A., Morgan, C.: Abstraction, Refinement And Proof For Probabilistic Systems. Springer (2004)
11. Dijkstra, E.W.: A Discipline of Programming. Prentice Hall (1976)
12. Kleene, S.C.: Recursive Predicates and Quantifiers. Trans. of the AMS **53**(1) (1943) 41 – 73
13. Ferrer Fioriti, L.M., Hermanns, H.: Probabilistic Termination: Soundness, Completeness, and Compositionality. In: Proc. of POPL 2015, ACM (2015) 489–501