Probabilistic Program Inference With Abstractions

Steven Holtzen
UCLA
sholtzen@cs.ucla.edu

Guy Van den Broeck
UCLA
guyvdb@cs.ucla.edu

Todd Millstein
UCLA
todd@cs.ucla.edu

1 Introduction

Abstraction is a fundamental tool in the analysis and verification of programs. Typically, a program abstraction selectively models particular aspects of the original program while utilizing non-determinism to conservatively account for other behaviors [2]. However, non-deterministic abstractions do not directly apply to the analysis of probabilistic programs. We recently introduced *probabilistic program abstractions*, which explicitly quantify the non-determinism found in a typical over-approximate program abstraction by using a probabilistic choice [5]. These probabilistic program abstractions are themselves probabilistic programs.

Here we illustrate probabilistic program abstractions by example in the context of *predicate abstraction* [1] and describe their application to probabilistic program inference. There is no universal solution to inference: every probabilistic program has subtle properties (for example, sparsity, continuity, conjugacy, submodularity, and discreteness) that require different inference strategies (for example, sampling, message passing, knowledge compilation, or path analysis). We propose to utilize probabilistic program abstractions to automatically *decompose* probabilistic program inference into several simpler inference problems. This general mechanism for breaking a complex query into sub-queries will allow the use of heterogeneous inference algorithms for different concrete sub-queries, and the abstraction will make precise how these sub-queries together can be used to produce the answer to the original inference query.

2 Predicate Abstractions

A *predicate abstraction* is a well-studied program abstraction whose abstract domain is a predicate domain [1]. Predicate abstractions are known as *Boolean programs*. Given a concrete program and a set of predicates $(p_1, \ldots, p_n)$ over the concrete domain, the goal of the predicate abstraction process is to construct an abstract Boolean program that forms a sound over-approximation of the concrete program and is as precise as possible relative to the given predicates.

As an example of the predicate abstraction procedure, consider Figure 1. Figure 1a shows an example concrete probabilistic program, while Figure 1b shows a non-deterministic abstraction of this concrete program with a single predicate. The assignment statement `x=0` in Figure 1a is abstracted to an assignment `{x<3}=T`. More interestingly, consider the assignment statement `x=x+1`. If `x<3` is false before the assignment, then we can be sure that `x<3` is false afterward. If `x<3` is true before the assignment, then the abstraction does not have enough information to know the value of `x<3` after the assignment. Hence in the Boolean program `{x<3}` is assigned to `{x<3}`$\wedge *$, where $*$ is a non-deterministic choice necessary for the abstraction to remain an over-approximation.

```
1 x = unif[-5, 5] // Inclusive uniform
2 if(x<0) { x = 0 } else { x = x + 1 }
```

(a) A simple concrete probabilistic program over integer `x`.

```
1 {x<3} = *
2 if({x<3} ∧ *) { {x<3} = T }
3 else { {x<3} = {x<3} ∧ * }
```

(b) A predicate abstraction of the program in Figure 1a induced by the predicate `x<3`, where $*$ denotes a non-deterministic choice, and `{p}` denotes the Boolean variable associated with the predicate `p`.

```
1 {x<3} = flip(θ1)
2 if({x<3} ∧ flip(θ2)) { {x<3} = T }
3 else { {x<3} = {x<3} ∧ flip(θ3) }
```

(c) A probabilistic predicate abstraction of the program in Figure 1a induced by the predicate `x<3`, where `flip`(θ) denotes a probabilistic choice which is true with probability θ.

Figure 1. A concrete program, its non-deterministic abstraction, and its probabilistic abstraction.

3 Probabilistic Predicate Abstractions

Predicate abstractions may be generalized to generate probabilistic program abstractions [5]. Figure 1c shows a probabilistic version of the non-deterministic abstraction from Figure 1b. In the probabilistic version, each use of the non-deterministic choice operator $*$ is replaced by a probabilistic `flip`(θ) operator, which is true with probability θ. For example, in the case when `x` is less than 3, the "then" branch of the `if`-statement is taken with probability θ_1.

Notably, the probabilistic abstraction is itself a *probabilistic program*: its semantics defines a probability distribution over the states of the predicates. Every probabilistic predicate abstraction is in fact a *family* of abstractions, induced by the parameters of the program. By choosing the parameters of the abstraction in a particular way, different relationships between the probabilistic program abstraction and the concrete program may be established. For example, by choosing $\theta_1 = 8/11$, we see that the predicate `{x<3}` is true with the same probability as in the concrete program on Line 1.

4 Decomposition via Abstraction

Approaches to probabilistic program inference can roughly be clustered into four categories: (1) performing inference directly on the program using sampling including Markov chain Monte-Carlo techniques (2) compiling the program to a tractable representation and performing inference on that new representation, (3) path-based inference using symbolic analysis, and (4) variational or message-passing approximations. Despite the enormous progress in general probabilistic program inference, all of the inference approaches above place strong requirements on the types of programs for which they are effective. However, we desire inference algorithms that flexibly apply a variety of inference approaches.

We propose to utilize probabilistic program abstractions to automatically *decompose* an inference task into several simpler ones. The simplest form of decomposition is based on *independence*; a small example is depicted in Figure 2. The concrete probabilistic program in Figure 2a calls a function f that uses discrete random variables, encodes logical dependencies, and has a complex control flow. It also calls function g which models a large number of correlated continuous variables with a smooth yet multi-modal probability landscape. Suppose that our goal is to compute $\Pr(x < 1)$. We believe that there does not exist a single algorithm that can perform this inference computation. However, inference in the function f is readily performed using the compilation approach or graphical model inference, and inference in function g is perfectly suitable for Hamiltonian Monte-Carlo sampling.

This observation is exploited by the probabilistic program abstraction in Figure 2b. Each θ parameter in the abstraction corresponds to a distinct inference sub-problem. In the example, we can respectively set the correct probabilities for each θ_i by performing inference on f and g separately, using the algorithm that is most effective for each. The last line of the abstraction then makes precise how the results of these inference queries combine to compute $\Pr(x < 1)$ exactly.

```
y = f() // discrete probabilistic program
z = g() // continuous probabilistic program
x = y * ⌊z⌋
```

(a) A hybrid concrete probabilistic program which combines the results of two complex functions f and g. The type of y is a nonnegative integer, the type of z is nonnegative real. The symbol $\lfloor\cdot\rfloor$ is the floor function.

```
{y<1} = flip(θ1)
{z<1} = flip(θ2)
{x<1} = {z<1} ∨ {y<1}
```

(b) A probabilistic predicate abstraction of 2a with predicates {x<1}, {y<1}, {z<1}.

Figure 2. A hybrid concrete probabilistic program that is difficult for inference, and a probabilistic predicate abstraction that allows for the decomposition of the inference query asking $\Pr(x < 1)$.

5 Future Work

In order to utilize probabilistic program abstractions to perform inference, we must produce abstractions that decompose a program into subprograms, each of which is amenable to a particular form of inference. We plan to formalize this decomposition process and identify criteria under which the abstraction can answer inference queries about the concrete program *exactly*, with no loss of precision. This work will leverage the connection to probabilistic programs and the existing body of work on their semantics.

6 Related Work

For context, we provide several strands of related prior work and compare them with our proposal. Probabilistic abstract interpretation [3] is a general framework for probabilistic verification of programs, and prior works that place upper bounds on the probability of a particular path [7] or construct Monte Carlo methods [8] can be viewed as instantiations of this framework. However, this line of work does not explore the connections between abstractions and probabilistic programs. Probabilistic program abstractions have been applied to a limited subset of program inference tasks, in particular probabilistic reachability in probabilistic timed automata [4, 6]. However, these tools do not extend to more general Bayesian inference queries involving evidence, which is necessary for supporting more sophisticated verification queries such as conditional reachability.

References

[1] Thomas Ball, Rupak Majumdar, Todd Millstein, and Sriram K. Rajamani. 2001. Automatic predicate abstraction of C programs. In *Proc. of PLDI*. 203–213.

[2] Patrick Cousot and Radhia Cousot. 1977. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *Proc. of POPL*. 238–252. https://doi.org/10.1145/512950.512973

[3] Patrick Cousot and Michael Monerau. 2012. Probabilistic abstract interpretation. In *Proc. of ESOP*. 169–193. https://doi.org/10.1007/978-3-642-28869-2_9

[4] Holger Hermanns, Björn Wachter, and Lijun Zhang. 2008. *Probabilistic CEGAR*. Springer Berlin Heidelberg, Berlin, Heidelberg, 162–175. https://doi.org/10.1007/978-3-540-70545-1_16

[5] Steven Holtzen, Todd Millstein, and Guy Van den Broeck. 2017. Probabilistic Program Abstractions. In *Proc. of UAI*.

[6] Marta Kwiatkowska, Gethin Norman, and David Parker. 2011. PRISM 4.0: Verification of Probabilistic Real-time Systems. In *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV'11)*. Springer-Verlag, Berlin, Heidelberg, 585–591.

[7] David Monniaux. 2000. Abstract Interpretation of Probabilistic Semantics. In *International Symposium on Static Analysis*. 322–339.

[8] David Monniaux. 2001. An Abstract Monte-Carlo Method for the Analysis of Probabilistic Programs. *SIGPLAN Not.* 36, 3 (Jan. 2001), 93–101. https://doi.org/10.1145/373243.360211