Finite-depth higher-order abstract syntax trees for reasoning about probabilistic programs [*]

Extended Abstract

Theophilos Giannakopoulos
BAE Systems
theo.giannakopoulos@baesystems.com

Mitchell Wand   Andrew Cobb
Northeastern University
wand@ccs.neu.edu   acobb@ccs.neu.edu

1. Introduction

We define a core calculus for the purpose of investigating reasoning principles of probabilistic programming languages. By using a variation of a technique called higher-order abstract syntax (HOAS), which is common in the implementation of domain-specific languages, the calculus captures the semantics of a stochastic language with observation while being agnostic to the details of its deterministic portions. By remaining agnostic to the non-stochastic portions of the language, this style of semantics enables the discovery of *general* reasoning principles for the principled manipulation of probabilistic program fragments by programmers, compilers, and analysis tools. This generality allows us to reason about probabilistic program fragments without the need to resort to the underlying measure theory in every instance, by instead enabling reasoning in terms of the core calculus in a way that we believe to be applicable to various surface-level languages.

2. The need for formal reasoning

The two program fragments in Figure 1 are written in pseudocode for a probabilistic programming language. They describe programs that seem equivalent: when any color coin is substituted for `x`, the programs denote measures that are equivalent up to a normalization factor (i.e. same-color coins have the same bias). For example,

$$\mathrm{Prob}\llbracket \texttt{p1 Red} \rrbracket(\texttt{Heads}) = \mathrm{Prob}\llbracket \texttt{p2 Red} \rrbracket(\texttt{Heads}) = 1/5.$$

However, when combined with the given prior on coin colors, a careful evaluation of their denotations shows that the two fragments are not equivalent.

$$\mathrm{Prob}\llbracket \texttt{do \{x <- prior; p1 x\}} \rrbracket(\texttt{Heads}) = \frac{33}{85}$$

$$\mathrm{Prob}\llbracket \texttt{do \{x <- prior; p2 x\}} \rrbracket(\texttt{Heads}) = \frac{9}{25}$$

Running an enumeration on the examples in a probabilistic programming language, such as Gamble [2] or webppl [4], yields the same results.

This example demonstrates how seemingly sound reasoning about probabilistic programs can fail if it is not carefully done. In a situation where it is not possible to enumerate the distributions denoted by both programs, e.g. where the purpose of using an equivalent form is to make inference tractable, such informal reasoning will lead to incorrect results.

```
p1 x = do
    d <- discrete [( 1/25, ( Red, Heads))
                  ,( 4/25, ( Red, Tails))
                  ,( 8/25, (Blue, Heads))
                  ,(12/25, (Blue, Tails))]
    observe (fst d == x)
    return (snd d)

p2 x = if x == Red
       then discrete [(1/5,Heads), (4/5,Tails)]
       else discrete [(2/5,Heads), (3/5,Tails)]

prior = discrete [(1/5, Red), (4/5, Blue)]
```

Figure 1. Example of inequivalent program fragments. Given a choice of a red or blue coin, the fragments `p1` and `p2` flip the coin and return the result. The coins have the same bias between the two fragments, but combining the fragments with the same prior does not result in equivalent programs.

EMBED

$$\frac{\mu \text{ is a sub-probability measure on } (A, \Sigma_A)}{\vdash \mathsf{embed}\, \mu : \mathsf{Meas}\ A}$$

BIND

$$\frac{\vdash t : \mathsf{Meas}\ A \qquad f : A \to \mathsf{Meas}\ B}{\vdash t \gg\!= f : \mathsf{Meas}\ B}$$

Figure 2. Constructors and typing rules

3. Finite-depth HOAS trees

The above example can be captured using *finite-depth HOAS trees*. Finite-depth HOAS trees are built from two constructors: embed and $\gg\!=$ (pronounced bind). The typing rules for the constructors can be found in Figure 2. The constructor embed allows for the embedding of arbitrary sub-probability measures into the calculus. The constructor $\gg\!=$ combines a HOAS tree with a (deterministic) mathematical function that produces HOAS trees. Using general mathematics as the host language for the HOAS technique is what makes our calculus language-agnostic.

The denotations assigned to these constructs are the morphisms and monadic bind from Panangaden's category of Markov kernels **SRel** [6], which extends Giry's category of probabilities [3] to sub-probability measures. The denotations are given in Figure 3.

[*] This material is based upon work sponsored by the Air Force Research Laboratory (AFRL) and the Defense Advanced Research Projects Agency (DARPA) under Contract No. FA8750-14-C-0002. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

$$[\![\mathsf{embed}\ \mu]\!](\sigma) = \mu(\sigma)$$

$$[\![t \gg= f]\!](\sigma) = \int [\![f_x]\!](\sigma)[\![t]\!](dx)$$

Figure 3. Finite-depth HOAS tree denotations. HOAS trees only have well-defined denotations when for every $t \gg= f$, $x \mapsto [\![f_x]\!](\sigma)$ is a measurable function for all σ.

Note that the monadic unit (or return) operation is captured by embedding a Dirac measure, which we denote `return` x = `embed` δ_x. The earlier pseudocode is in the do-notation syntactic sugar for monadic expressions. We also introduce `observe` as sugar for

$$\texttt{observe}\ b = \begin{cases} \texttt{return}\ () & b = \texttt{True} \\ \texttt{embed}\ (\sigma \mapsto 0) & \text{otherwise} \end{cases}.$$

In this way, observation failure is represented as missing mass in the sub-probability measure.

Using this calculus, we prove a theorem that describes when probabilistic program fragments (which implicitly denote measure *kernels*) can be reasoned about as normalized distributions, rather than as more general measures. To obtain this theorem, we introduce two forms of equivalence on measure kernels. For two program fragments $f, g : A \to \texttt{Meas}\, B$, we say that f is *point-wise dist-equivalent* to g when for all a, there is some normalizing constant $c \in \mathbb{R}$ such that for all σ, $[\![f_a]\!](\sigma) = c \cdot [\![g_a]\!](\sigma)$. This is the kind of equivalence we used in the attempt to reason about the program fragments in the original example.

When the same c can be used for all a, we have the second form of equivalence, and say that the two program fragments are *uniformly dist-equivalent*.

THEOREM 1. *Let* $t, s : \texttt{Meas}\, A$ *and let* $f, g : A \to \texttt{Meas}\, B$. *Assume there is some* c *such that for all* σ, $[\![t]\!](\sigma) = c \cdot [\![s]\!](\sigma)$ *and that* f *and* g *are uniformly dist-equivalent with normalizing constant* d.

Then, for all σ, $[\![t \gg= f]\!](\sigma) = c \cdot d \cdot [\![s \gg= g]\!](\sigma)$.

This theorem states the requirement that two program fragments be *uniformly* dist-equivalent, rather than simply point-wise dist-equivalent, for them to exhibit equivalence in all contexts. This is the reason why the programs in the original example were not equivalent in the given context.

Simple sufficient criteria for the application of this theorem are that two program kernels be point-wise dist-equivalent and have no observations that depend on the input to the kernels. This is powerful enough to allow a programmer to use observations to define "helper distributions" which can then be re-written (by the programmer or by a sufficiently smart compiler) into more efficient forms, e.g. via equivalences due to conjugacy. For example, a programmer could write

```
b <- beta 1 1
c <- bernoulli b
observe (c == 1)
```

at an arbitrary point in a program, and by the above theorem it could be replaced with the equivalent and presumably more efficient `beta 2 1`.

4. Related work

As mentioned above, we use Panangaden's **SRel** category to give the semantics for HOAS trees [6]. Unlike Panangaden we have no recursion in our language, and so we are using the missing probability mass of the sub-probability measures to represent observation failure instead of non-termination.

In that our calculus does not support recursion and that we use missing probability mass to represent observation failure, we have some similarities to the measure transformers given by Borgström [1]. However, unlike Borgström we prefer to work with measures directly and so do not make use of probability density functions in our semantics.

5. Ongoing work and conclusions

Denotational semantics is useful for reasoning because it lets us focus on the meaning of programs independently of execution strategies for the programs. Because our calculus is agnostic to the deterministic portions of a language, it is even more independent of details unrelated to the reasoning we want to perform.

Using this technique we were able to devise a general theorem about the equivalence of probabilistic program fragments up to normalization. This theorem, which justifies the re-writing of program fragments at arbitrary points in a program, demonstrates that our approach can be used for discovering novel reasoning principles about probabilistic programs.

We are currently working on a second core calculus that captures recursively defined probabilistic programs with strict data structures. The semantics of this calculus are given in a complete partial order (cpo) of measures that is similar to the ones given by Saheb-Djahromi [7] and by Jones & Plotkin [5]. So far we have preliminary results indicating that the interaction between joins in cpos and integration of measures is well behaved. We have also found that when working in a calculus that supports non-termination, it appears to be convenient to distinguish between observation failure and non-termination by abandoning the use of sub-probability measures and instead using distinct points for the different kinds of failure.

We hope to use the second core calculus to show results similar to those for finite-depth HOAS trees. We also plan to extend the semantics to handle non-strict data structures, so that we can use them to reason about constructs like `mem`, which we have observed to be frequently used in the design of probabilistic programs.

Acknowledgments

Thanks to Norman Ramsey, Aleksey Kliger, Ryan Culpepper, Sean Stromsten, and Valentino Crespi.

References

[1] J. Borgström, A. D. Gordon, M. Greenberg, J. Margetson, and J. Van Gael. Measure transformer semantics for Bayesian machine learning. In *Programming Languages and Systems*, pages 77–96. Springer, 2011.

[2] R. Culpepper. Gamble. https://github.com/rmculpepper/gamble, 2015.

[3] M. Giry. A categorical approach to probability theory. In *Categorical aspects of topology and analysis*, pages 68–85. Springer, 1982.

[4] N. D. Goodman and A. Stuhlmüller. The Design and Implementation of Probabilistic Programming Languages. http://dippl.org, 2014. Accessed: 2015-10-12.

[5] C. Jones and G. D. Plotkin. A probabilistic powerdomain of evaluations. In *Logic in Computer Science, 1989. LICS'89, Proceedings., Fourth Annual Symposium on*, pages 186–195. IEEE, 1989.

[6] P. Panangaden. The category of Markov kernels. *Electronic Notes in Theoretical Computer Science*, 22:171–187, 1999.

[7] N. Saheb-Djahromi. CPO's of measures for nondeterminism. *Theoretical Computer Science*, 12(1):19–37, 1980.