Running Probabilistic Programs Backwards

Neil Toronto and Jay McCarthy

PLT @ Brigham Young University, Provo, Utah, USA
`neil.toronto@gmail.com` and `jay@cs.byu.edu`

Abstract. All probabilistic languages are limited in some critical way. Languages created by statisticians for their own use lack semantics, and their implementations almost certainly compute wrong values on large classes of programs. Almost all languages defined by semantics lack probabilistic conditioning, making them useless for probabilistic inference.
We first define an uncomputable semantics that gives all programs a theoretically sound, exact meaning. We then derive and implement a computable, approximate semantics and prove that its approximations approach the values calculated using the exact semantics.
The uncomputable semantics is based on measure theory, so the derived implementation handles probabilistic conditioning naturally, as well as programs for which other implementations compute wrong values.

Keywords: Semantics, Domain-specific languages, Probability theory

1 Introduction

All probabilistic languages to date are limited in some critical way.

Languages defined by statisticians for their own use lack a semantics; there is thus no standard for correct behavior or convergence. Further, because of their implicit reliance on density models, these languages cannot behave correctly in the presence of mixed-type arithmetic, nonlinear probabilistic conditions, and unbounded recursion.

On the other hand, almost all probabilistic languages defined by a semantics lack a critical feature: probabilistic conditioning. While such languages are useful for studying probabilistic algorithms and implementing reliable probabilistic simulations, they are mostly useless for inference.

In short, there are languages for inference that statisticians cannot trust, and reliable languages for simulation that statisticians cannot use.

2 The Preimage Arrow

We interpret probabilistic programs as **measure-theoretic models** so that mixed-type arithmetic, nonlinear conditions, and unbounded recursion can be given meanings consistent with theoretical expectations.

Our measure-theoretic models consist of 1) a universe of first-order values $\mathcal{V}$; 2) a Borel σ-algebra $\mathcal{B}(\mathcal{V})$ containing measurable subsets of $\mathcal{V}$ (a measure-theoretic analogue of a topology); 3) a probability measure $\mathbb{P} : \mathcal{B}(\mathcal{V}) \Rightarrow [0,1]$; and 4) the interpretation of a program, a function $F : \mathcal{V} \Rightarrow \mathcal{V}$. The probability of a set $B \in \mathcal{B}(\mathcal{V})$ of program outputs is $\mathbb{P}(F^{-1}(B))$, where $F^{-1}(B)$ denotes calculating the preimage of B under F.

For $\mathbb{P}(F^{-1}(B))$ to be sensible, $F^{-1}(B)$ must be in $\mathcal{B}(\mathcal{V})$; i.e. F must be $\mathcal{B}(\mathcal{V})$-$\mathcal{B}(\mathcal{V})$ measurable, a property analogous to topological continuity but weaker. In fact, constructing F using the combinators that define the ***measurable function arrow*** and a small selection of lifted primitives is enough to ensure F is measurable. The proofs are structural and follow from the measurability of the combinators and primitives. Further, the measurability proofs give simple formulas for calculating preimages compositionally.

Because these preimage formulas operate on arbitrary uncountable sets, they are generally unimplementable. We therefore define the measurable function arrow in λ_{ZFC} [1], an untyped, super-computational λ-calculus that gives precise meaning to terms that operate on uncountable sets. The goal now is to define a compositional process for computing approximate preimages that are in some sense equivalent—even if only in the limit—to the exact preimages calculated under measurable function arrow instances.

First, we use the preimage formulas to define the ***preimage arrow***, whose instances calculate exact preimages under their corresponding measurable function. That is, if $F : \mathcal{V} \Rightarrow \mathcal{V}$ is a measurable function, and $G : \mathcal{B}(\mathcal{V}) \Rightarrow \mathcal{B}(\mathcal{V})$ is constructed the same way but in the preimage arrow, then $G(B) = F^{-1}(B)$ for all $B \in \mathcal{B}(\mathcal{V})$.

Second, we replace the σ-algebra $\mathcal{B}(\mathcal{V})$ with an algebra $\mathcal{C}(\mathcal{V})$ of finitely representable rectangles. We also relax the requirement that the preimage arrow compute exact preimages, and settle for conservative overapproximations. The result is the ***approximate preimage arrow***. Though it is defined in λ_{ZFC}, it is directly implementable. We have sketched a proof that, as long as its combinators have certain properties, it can be used to generate disjoint preimage coverings that, in the limit, overapproximate exact preimages only by a set of measure 0. Therefore, probabilities computed using the approximate preimage arrow converge to the probabilities calculated using the preimage arrow or measurable function arrow.

We have implemented the approximate preimage arrow and a semantic function that transforms first-order let-calculus terms into arrow instances. We have used this language, along with a novel, self-adjusting, divide-and-conquer sampler that searches for nonempty preimage subsets, to carry out statistical inference. This includes inference on programs with mixed-type arithmetic, nonlinear probabilistic conditions, and unbounded recursion.

References

1. Toronto, N., McCarthy, J.: Computing in Cantor's paradise with λ-ZFC. In: Functional and Logic Programming Symposium (FLOPS). pp. 290–306 (2012)