Higher-Order Probabilistic Programming

A Tutorial at POPL 2019

Ugo Dal Lago*

Abstract

This tutorial is meant to be an introduction to the principles of randomized and bayesian higher-order programming languages. We will start by giving some simple examples of probabilistic higher-order programs, written in generic or domain specific functional programming languages. Particular attention will be given in highlighting *why* sampling and conditioning can be useful in programming, and *how* the metatheory of higher-order probabilistic programming differs from the one of its deterministic sibling.

1 Tutorial Description

Functional programming languagues are well known to enable modularity and code reuse. Moreover, functional programs are easy to reason about, thanks to the simplicity and cleanliness of the underlying semantics. This is particularly useful in application domains in which symbolic manipulation or complex mathematical modeling are needed. An essential ingredient in many functional programming idioms are *higher-order functions*, namely functions which can take other functions as input and produce functions in output. This mechanism allows for modularity and code reuse, e.g., through so-called higher-order combinators like `fold`, `map`, or `filter`.

This tutorial is meant to be an introduction to the principles of *randomized* and *bayesian* higher-order programming languages. The tutorial is structured into four parts:

1. **Motivating Examples**. We will first introduce some examples of classic *randomized* algorithms implemented in the OCAML functional programming langaage. Then, we will look at how probabilistic graphical models can be written as programs in functional programming languages like HANSEI, giving rise to so-called *bayesian* programming. This part will also serve as a gentle introduction to probabilistic programming for those among the attendants who are not familiar with these concepts.

2. **A λ-Calculus Foundation**. We will introduce a hierarchy of two increasingly expressive typed λ-calculi in which forms of sampling and conditioning operators are available, this way allowing for general enough forms of randomized and bayesian programming. Both calculi will be based on Plotkin's **PCF**.

3. **Operational and Denotational Semantics**. We will endow the languages introduced in the Second Part with two forms of operational semantics, the first of them capturing program evaluation through distributions, the second one capturing *learning* through a notion of sampling. A notion of contextual equivalence will be given together with a refinement in the form of a pseudometric. We will also describe the challenges one faces when generalizing classic domain theory to probabilistic programming.

4. **Resource-Bound Verification through Type Systems**. The main reason for the adoption of randomized algorithms lies in their efficiency: in many computational problems, the most time-efficient algorithms, practically if not theoretically, are randomized. In Bayesian

*University of Bologna & INRIA Sophia Antipolis, ugo.dallago@unibo.it

programming, termination is crucial for learning. We will describe how termination and complexity analysis can be carried out by type systems when done in probabilistic programming languages. We will focus on two aspects, namely on how a proof of *soundness* for the introduced type systems can be carried out, and on the *expressive power* of the introduced type systems.

Prerequisites for this tutorial will be kept to a minimum, being limited to some familiarity with basic probability theory and with the theory and practice of functional programming. Slides will be available in the tutorial's webpage[1].

2 Biography

Ugo Dal Lago is Associate Professor of Computer Science at the University of Bologna, and the vice-head of the FoCUS research group at INRIA Sophia Antipolis. Along the last twenty years, he has worked on complexity analysis of functional programming languages, on relational reasoning about probabilistic programming. He recently applied the above to λ-calculi for machine learning and for the verification of cryptographic primitives.

[1] `http://www.cs.unibo.it/~dallago/HOPP/`