Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 319 (2015) 199–216

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Conditioning in Probabilistic Programming

Nils Jansen Benjamin[1] Lucien Kaminski[1] Joost-Pieter Katoen[1]
Federico Olmedo[1]

RWTH Aachen University, Aachen, Germany

Friedrich Gretz[2] Annabelle McIver[2]

Macquarie University, Sydney, Australia

Abstract

In this paper, we investigate the semantic intricacies of conditioning in probabilistic programming, a major feature, e.g., in machine learning. We provide a quantitative weakest pre–condition semantics. In contrast to all other approaches, non–termination is taken into account by our semantics. We also present an operational semantics in terms of Markov models and show that expected rewards coincide with quantitative pre–conditions. A program transformation that entirely eliminates conditioning from programs is given; the correctness is shown using our semantics. Finally, we show that an inductive semantics for conditioning in non–deterministic probabilistic programs cannot exist.

Keywords: Probabilistic Programming, Semantics, Conditional Probabilities, Program Transformation

1 Introduction

In recent years, interest in probabilistic programming has rapidly grown [9,11]. This is due to its wide applicability, for example in machine learning for describing distribution functions; Bayesian inference is pivotal in their analysis. It is used in security for describing both cryptographic constructions such as randomized encryption and experiments defining security properties [4]. Probabilistic programs, being extensions of familiar notions, render these fields accessible to programming communities. A rich palette of probabilistic programming languages exists including `Church` [8] as well as modern approaches like probabilistic `C` [23], `Tabular` [10] and `R2` [22].

Probabilistic programs are sequential programs having two main features: (1) the ability to *draw values at random* from probability distributions, and (2) the

[⋆] This work was supported by the Excellence Initiative of the German federal and state government.

[1] Emails: {nils.jansen, benjamin.kaminski, katoen, federico.olmedo}@informatik.rwth-aachen.de

[2] Emails: annabelle.mciver@mq.edu.au, friedrich.gretz@students.mq.edu.au

http://dx.doi.org/10.1016/j.entcs.2015.12.013
1571-0661/

ability to *condition the value of variables* in a program through so–called *observations*. The semantics of languages without conditioning is well–understood: In his seminal work, Kozen [19] considered denotational semantics for probabilistic programs without non–determinism or observations. One of these semantics—the expectation transformer semantics—was adopted by McIver and Morgan [21], who added support for non–determinism; a corresponding operational semantics is given in [13]. Other relevant works include probabilistic power–domains [17], semantics of constraint probabilistic programming languages [15,14], and semantics for stochastic λ–calculi [26].

Semantic intricacies. The difficulties that arise when program variables are conditioned through observations is less well–understood. This gap is filled in this paper. Previous work on semantics for programs with `observe` statements [22,16] do neither consider the possibility of *non–termination* nor the powerful feature of *non–determinism*. In contrast, we thoroughly study a more general setting which accounts for non–termination by means of a very simple yet powerful probabilistic programming language supporting non–determinism and observations. Let us first analyze a few examples illustrating the different problems. We start with the problem of non–termination; consider the two program snippets

$$x \coloneqq 2 \qquad \text{and} \qquad \{x \coloneqq 2\}\ [1/2]\ \{\texttt{abort}\}\ .$$

The program on the left just assigns the value 2 to the program variable x, while the program on the right tosses a fair coin—which is modeled through a *probabilistic choice*—and depending on the outcome either performs the same variable assignment or diverges due to the `abort` instruction. The semantics given in [22,16] does not distinguish these two programs and is only sensible in the context of terminating programs. A programmer writing only terminating programs is already unrealistic in the non–probabilistic setting. Our semantics does not rely on the assumption that programs always terminate and is able to distinguish these two programs.

To discuss *observations*, consider the program snippet P_{obs_1}

$$\{x \coloneqq 0\}\ [1/2]\ \{x \coloneqq 1\};\ \texttt{observe}\ (x{=}1),$$

which assigns zero to the variable x with probability $1/2$ while x is assigned one with the same likelihood, after which we condition to the outcome of x being one. The `observe` statement blocks all invalid runs violating its condition and renormalizes the probabilities of the remaining valid runs. This differs, e.g., from program annotations like (probabilistic) *assertions* [25] as we will see later. The interpretation of the program is the expected outcome conditioned on the valid runs. For P_{obs_1}, this yields the outcome $1 \cdot 1$—there is one valid run that happens with probability one, with x being one.

More involved problems arise when programs are *infeasible* meaning all runs are blocked. Consider a slight variant of the program above, called P_{obs_2}:

$$\{x \coloneqq 0;\ \texttt{observe}\ (x{=}1)\}\ [1/2]\ \{x \coloneqq 1;\ \texttt{observe}\ (x{=}1)\}$$