4

On Probabilistic λ-Calculi

Ugo Dal Lago
University of Bologna & INRIA Sophia Antipolis

Abstract: This chapter is meant to be a gentle introduction to probabilistic λ-calculi in their two main variations, namely randomised λ-calculi and Bayesian λ-calculi. We focus our attention on the operational semantics, expressive power and termination properties of randomised λ-calculi, only giving some hints and references about denotational models and Bayesian λ-calculi.

4.1 Introduction

Probabilistic models are more and more pervasive in computer science and are among the most powerful modelling tools in many areas like computer vision (Prince, 2012), machine learning (Pearl, 1988) and natural language processing (Manning and Schütze, 1999). Since the early times of computation theory (De Leeuw et al., 1956), the concept of an algorithm has been generalised from a purely deterministic process to one in which certain elementary computation steps can have a probabilistic outcome, this way enabling efficient solutions to many computational problems (Motwani and Raghavan, 1995). More recently, programs have been employed as means to express probabilistic *models* rather than *algorithms*, with program *evaluation* replaced by a form of *inference* which does not aim at looking for the result of a computation, but rather at the probability of certain events in the model.

How can all this be taken advantage of in programming languages, and in particular in *higher-order* functional programming languages? How is the underlying meta-theory affected? This Chapter is an attempt to give a brief introduction to this topic, presenting some basic notions and results, and pointing to the relevant literature on the subject. Although probabilistic λ-calculi have been known from four decades now (Saheb-Djaromi, 1978; Jones and Plotkin, 1989), their study has been quite scattered until very recently, and a unified view of their theory is, as a consequence, still missing.

[a] From *Foundations of Probabilistic Programming*, edited by Gilles Barthe, Joost-Pieter Katoen and Alexandra Silva published 2020 by Cambridge University Press.

A universally accepted paradigm for functional programs is Church's λ-calculus (Barendregt, 1984), in which the processes of forming functions and of passing parameters to them are modelled by dedicated constructs, namely by the λ-binder and binary application. Probabilistic λ-calculi most often take the form of ordinary λ-calculi in which the language of terms is extended with one or more constructs allowing for a form of probabilistic evolution. There are at least two ways to do that, which give rise to two different styles of λ-calculi, depending on the additional operators they provide.

Randomised λ-calculi. Here, the only new operator provided by the underlying programming language is a form of probabilistic choice, whose evaluation can produce different outcomes, in a probabilistic fashion. Various choice operators can be considered, the simplest one is a form of binary, fair, probabilistic choice. By that, one can form terms such as $M \oplus N$, which evolves like M or N depending on the outcome of a probabilistic process, typically corresponding to the flipping of a coin. The outcome of such a coin flip is thus a probabilistic event and different coin flips are taken as *independent* events. This new operator alone is perfectly sufficient to model randomised algorithms (Motwani and Raghavan, 1995). We call λ-calculi built along these lines *randomised* λ-calculi. As already mentioned, randomised λ-calculi have been investigated since the seventies (Saheb-Djaromi, 1978; Jones and Plotkin, 1989), but scatteredly until very recently, when they have been the object of much work about denotational semantics (Jung and Tix, 1998; Danos and Harmer, 2002; Danos and Ehrhard, 2011), program equivalence (Dal Lago et al., 2014a; Crubillé and Dal Lago, 2014; Bizjak and Birkedal, 2015) and type systems (Dal Lago and Grellois, 2017; Breuvart and Dal Lago, 2018).

Bayesian λ-calculi. In Bayesian λ-calculi, programs are not seen as modelling *algorithms*, like in randomised λ-calculi, but rather as a way to describe a certain kind of probabilistic models, namely *bayesian networks* (Pearl, 1988; Koller and Friedman, 2009), also known as *probabilistic graphical models*. This paradigm has been adopted in concrete programming languages like ANGLICAN (Tolpin et al., 2015) and CHURCH (Goodman et al., 2008) and ultimately consists in endowing the class of terms with two new constructs, the first one modelling *sampling* and thus conceptually similar to the probabilistic choice operator from randomised λ-calculi, and the second one allowing to *condition* the underlying distribution based on external evidence, this way giving rise to both an *a priori* and an *a posteriori* distribution. Bayesian λ-calculi, contrarily to randomised λ-calculi, have been introduced only relatively recently (Borgström et al., 2016), and their metatheory is definitely not as stable as the one of randomised λ-calculi. Most often, but not

always, the sampling and conditioning operators works on real numbers, this way allowing to build continuous probabilistic models.

These two kinds of λ-calculi certainly have some similarities, but deserve to be introduced and described independently. Randomised λ-calculi will be the subject of Section 4.3, while Bayesian λ-calculi will be introduced in Section 4.4. As a prologue to that, we will give an introduction to probabilistic λ-calculi by way of an example, in Section 4.2.

Endowing programs with probabilistic primitives (e.g. an operator which models sampling from a distribution) significantly changes the underlying theory. The reader is however invited to keep in mind that this domain is still under investigation by the programming language and logic in computer science communities, and is thus intrinsically unstable, in particular as for Bayesian λ-calculi. In the following, we give some hints as for the challenges one needs to face when analysing randomised and bayesian λ-calculi.

Operational Semantics and Contextual Equivalence. Formally describing the computational process implicit in a λ-term becomes strictly more challenging when the latter is allowed to flip coins, thus evolving probabilistically rather than deterministically. In particular, capturing the evaluation process in a finitary way, like in ordinary λ-calculus, is impossible. When conditioning is present, the task becomes even more difficult, since computation is replaced by learning. Another difficulty one encounters when dealing with the operational semantics of randomised and bayesian λ-calculi is the necessity of some (admittedly basic) measure theory, this of course only in presence of *continuous* rather than *discrete* distributions.

Expressive Power. Not much is known about the expressive power of *probabilistic* higher-order calculi, as opposed to the extensive literature on the same subject about *deterministic* calculi (see, e.g. (Statman, 1979; Sørensen and Urzyczyn, 2006; Longley and Normann, 2015)). What happens to the class of representable functions if one enriches, say, a deterministic λ-calculus X with certain probabilistic choice primitives? Are the expressive power or the good properties of X somehow preserved? These questions have been given answers in the case in which X is the pure, untyped, probabilistic λ-calculus (Dal Lago and Zorzi, 2012): in that case, Turing-completeness continues to hold, i.e., fair binary probabilistic choice is sufficient to encode of computable distributions. But what if one restricts the underlying calculus, e.g., by way of a type system?

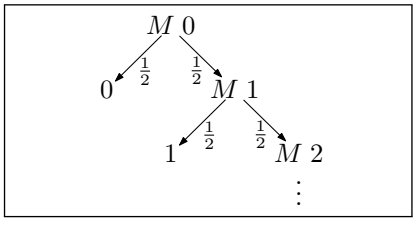
Termination. Termination is a key property already in deterministic functional programs, but how should it be spelled out in probabilistic λ-calculi? There are at least *two* different ways to give an answer to this question, following a pioneering

work on probabilistic λ-calculus (Saheb-Djaromi, 1978) and recent extensive work on probabilistic termination (McIver and Morgan, 2005; Bournez and Garnier, 2005). On the one hand, we can consider *almost sure termination*, by which we mean termination with maximal likelihood. On the other hand, we can go for the stronger *positive almost sure termination*, in which one requires the average number of evaluation steps to be finite. Are there ways to enforce either form of termination in probabilistic λ-calculi, similarly to what has been done in deterministic ones?

Denotational Semantics. Already for a simple, imperative probabilistic programming language, giving a denotational semantics is nontrivial (Kozen, 1981). When languages also have higher-order constructs, everything becomes even harder (Jung and Tix, 1998) to the point of disrupting much of the beautiful theory known in the deterministic case (Barendregt, 1984). This has stimulated research on denotational semantics of higher-order probabilistic programming languages, with some surprising positive results coming out recently (Ehrhard et al., 2014; Heunen et al., 2017).

In the rest of this Chapter, we focus on the first three aspects, leaving the task of delving into the denotational semantics of probabilistic λ-calculi to some future contribution. We are mainly interested in randomised λ-calculi, giving some hints about bayesian λ-calculi in Section 4.4.

4.2 A Bird's Eye View on Probabilistic Lambda Calculi



Consider a λ-term M such that for every real number r, the term $M\ r$ deterministically reduces to $r \oplus (M\ (r+1))$, where $\oplus$ is the new operator for binary probabilistic choice mentioned in the previous section. After evolving deterministically, the term $M\ r$ thus flips a fair coin, and either terminates as r, with probability $\frac{1}{2}$ or proceeds as $M\ (r+1)$, again with probability $\frac{1}{2}$. In Figure 4.2, the overall computational behaviour of the term $M\ 0$ is graphically represented as an infinite binary tree. This tree should not be confused with the reduction tree of an ordinary λ-term, in which branching models the *external* nondeterminism coming from the choice of the next redex to fire, rather than the *internal* nondeterminism coming from $\oplus$.

It should already be clear that some basic questions about the dynamics of term reduction are bound to receiving answers which are fundamentally different from the ones one gets in the ordinary λ-calculus. Let us consider some of them. First of

all, *which value* does $M\ \underline{0}$ evaluate to? Clearly, the result of the evaluation process cannot just be *one* value, and must rather be a *distribution* of values. But which one? One is tempted to say that the distribution to which $M\ \underline{0}$ evaluates is the geometric distribution assigning probability $\frac{1}{2^{n+1}}$ to every natural number n. If this the correct answer, then how could we *prove* that this is the case? Finitary, inductively defined formal systems are bound *not* to be the right tool here, since such derivations can by construction only "prove" distributions having *finite support* to be those to which terms evaluate. Finitary derivations are however used to derive *approximations* to the operational semantics of the underlying term, as we will see in Section 4.3.2 below.

Another question then arises: should we consider $M\ \underline{0}$ as *terminating*? And *why?* Actually, termination becomes a probabilistic event in presence of probabilistic choices, and as such happens *with a certain probability*. The probability that the evaluation of $M\ \underline{0}$ terminates is easily seen to be $\sum_{i=0}^{\infty} \frac{1}{2^{n+1}}$, namely 1. As such, $M\ \underline{0}$ is an *almost surely terminating term*. This is not the end of the story, however: what if we are rather interested in checking that the *expected number of reduction steps* to termination for $M\ \underline{0}$ is finite? Again, the answer is positive, let us see why. The expected number of reduction steps can be computed by counting the number of *internal nodes* of the reduction tree of $M\ \underline{0}$ (again, see Figure 4.2), each node weighted by the probability of reaching it. This way every computation step is taken into account without having to deal directly with infinite traces and their probabilities, which requires measure theory. In the case at hand, internal nodes are those labelled with $M\ \underline{n}$ and each of them has probability $\frac{1}{2^n}$. As a consequence, the expected type to termination is $\sum_{n=0}^{\infty} \frac{1}{2^n} = 2$, and this witnesses the fact that $M\ \underline{0}$ is indeed *positively* almost surely terminating. Is there any relation between the two concepts we have just introduced? We will have something to say about that in Section 4.3.5 below.

One of the most interesting properties of the ordinary λ-calculus is *confluence*: the inherent nondeterminism induced by the presence of multiple redexes is of a very benign form, i.e., if M rewrites to both N and L, then there is P to which both N and L themselves rewrite. As consequence, the normal form of any term M if it exists, is unique. The choice of a strategy does not influence the actual final result of the computation, although not all strategies are guaranteed to lead to a normal form. Unfortunately, this nice picture is not there anymore if one endows the λ-calculus with the $\oplus$ operator. Consider the term M, defined as $(\lambda x.add_2(x, x))\ (\underline{0} \oplus \underline{1})$ where add_2 is a operator computing addition modulo 2, which can be easily defined. Two redexes occur in M namely M itself and $\underline{0} \oplus \underline{1}$. Firing the latter first leads, independently on the outcome of the probabilistic choice, to $\underline{0}$. If M is fired first, instead, one obtains either $\underline{0}$ or $\underline{1}$, each with probability $\frac{1}{2}$. Please observe that the

failure of confluence is *not* merely a consequence of the presence of probabilistic choice, but holds even if considering all possible outcomes.

4.3 A Typed λ-Calculus with Binary Probabilistic Choice

In this section, we introduce randomised λ-calculi in their simplest form, namely one in which the only probabilistic operator is one for binary probabilistic choice. We have chosen to go *typed* rather than *untyped*, because this way examples are easier to delineate. Most of the results we give here remain valid in an untyped setting, e.g. the setting considered in many relevant works on the subject (Ehrhard et al., 2011; Dal Lago and Zorzi, 2012; Dal Lago et al., 2014a). Going typed (Saheb-Djaromi, 1978; Jones and Plotkin, 1989; Danos and Harmer, 2002; Crubillé and Dal Lago, 2014; Bizjak and Birkedal, 2015) has also the advantage of allowing to present calculi and type systems guaranteeing termination without the need to significantly change the underlying notation. Bayesian λ-calculi are, by the way, naturally presented as typed calculi themselves (Culpepper and Cobb, 2017; Heunen et al., 2017; Wand et al., 2018; Vákár et al., 2019), and this is precisely what we are going to do in Section 4.4 below.

The calculus we introduce in the rest of Section 4.3, dubbed $\mathsf{PCF}_{\oplus}$, can be seen as being an extension of Plotkin's PCF in which an operator for binary probabilistic choice is available, while the rest of the system (including types and typing rules) remain essentially unaltered.

4.3.1 Types and Terms

The first notion we need is the one of a *type* which, as we already mentioned, is not different from the one of other typed λ-calculi:

Definition 4.1 ($\mathsf{PCF}_{\oplus}$: Types)**.** The *types* of $\mathsf{PCF}_{\oplus}$ are the expressions derived by way of the following grammar:

$$\textbf{Types} \qquad \tau, \rho \ ::= \ \textsc{Unit} \mid \textsc{Num} \mid \tau \to \rho;$$

There are two type constants Unit and Num, while type constructors only include the arrow, modelling function spaces, and do *not* include, e.g., products or coproducts. Including them in the calculus would be harmless, but would render the underlying metatheory unnecessarily more complicated. The ground type Unit is to be interpreted as the singleton set, while the type of numbers Num is interpreted as a monoid $(\mathbb{M}, +, 0_{\mathbb{M}})$. As an example, $\mathbb{M}$ could be the natural numbers or the real numbers. Taking real numbers as a ground type has the advantage of allowing a smooth integration of sampling from continuous distributions, as we will see in

Terms	$M, N ::= V \mid V\ W \mid \texttt{let}\ M = x\ \texttt{in}\ N \mid M \oplus N$
	$\mid \texttt{if}\ V\ \texttt{then}\ M\ \texttt{else}\ N \mid f_n(V_1, \ldots, V_n)$
Values	$V, W ::= \star \mid x \mid r \mid \lambda x.M \mid \texttt{fix}\ x.V$

Figure 4.1 Terms and Values

Section 4.4 below. All we say in this section holds independently on $\mathbb{M}$. In the following sections, however, sticking to one particular monoid $\mathbb{M}$ will sometimes be necessary. Whenever we want to insist on the underlying monoid to be $\mathbb{M}$, we write $\mathsf{PCF}_{\oplus}^{\mathbb{M}}$ instead of $\mathsf{PCF}_{\oplus}$.

We assume to reader to be familiar with the basic terminology and notation from usual, pure λ-calculus. Good references for that are Barendregt (1984) or Hindley and Seldin (2008), for example.

Definition 4.2 ($\mathsf{PCF}_{\oplus}$: Terms and Values)**.** *Terms* and *values* of $\mathsf{PCF}_{\oplus}$ are both defined in Figure 4.1, where r ranges over $\mathbb{M}$, x ranges over a denumerable sets of variables $\mathcal{V}$, and f_n ranges over a class of function symbols $\mathcal{F}_n$, each of them interpreted as a total function $f_n^* : \mathbb{M}^n \to \mathbb{M}$. Both terms and values, as customary, are taken modulo α-equivalence. The set of all terms (respectively, all values) is indicated as $\mathbb{T}$ (respectively, as $\mathbb{V}$).

The calculus $\mathsf{PCF}_{\oplus}$, is indeed a close relative of ordinary PCF. One difference is the fact that terms are written in so-called *A-normal form*: one cannot form the application MN of two *arbitrary* λ-terms M and N, but only of two *values* V and W. The generic form of an application can however be recovered as follows:

$$M\ N = \texttt{let}\ M = x\ \texttt{in}\ \texttt{let}\ N = y\ \texttt{in}\ (x\ y)$$

where x and y are fresh variables not occurring free in M nor in N. The other main difference, of course, is the presence of a binary choice operator $\oplus$, which models fair binary probabilistic choice. A slightly more general form of binary choice is sometimes used in the literature, namely one in which the left argument is chosen with probability p and the right argument is chosen with probability $1 - p$, the number p being any rational number between 0 and 1. Choice then becomes *biased*, and takes the form of a family of operators $\{\oplus_p\}_{p \in \mathbb{Q}_{[0,1]}}$. Fair binary choice is however perfectly sufficient for our purposes, including the one of guaranteeing the calculus to be universal as for its expressive power.

Value Typing Rules

$$\frac{}{\Gamma \vdash \star : \textsc{Unit}}\ \mathsf{S} \qquad \frac{}{\Gamma, x : \tau \vdash x : \tau}\ \mathsf{V} \qquad \frac{}{\Gamma \vdash r : \textsc{Num}}\ \mathsf{R}$$

$$\frac{\Gamma, x : \tau \vdash M : \rho}{\Gamma \vdash \lambda x.M : \tau \to \rho}\ \lambda \qquad \frac{\Gamma, x : \tau \to \rho \vdash M : \tau \to \rho}{\Gamma \vdash \mathtt{fix}\ x.M : \tau \to \rho}\ \mathsf{X}$$

Term Typing Rules

$$\frac{\Gamma \vdash V : \tau \to \rho \quad \Gamma \vdash W : \tau}{\Gamma \vdash V\ W : \rho}\ @ \qquad \frac{\Gamma \vdash M : \tau \quad \Gamma, x : \tau \vdash N : \rho}{\Gamma \vdash \mathtt{let}\ M = x\ \mathtt{in}\ N : \rho}\ \mathsf{L}$$

$$\frac{\Gamma \vdash M : \tau \quad \Gamma \vdash N : \tau}{\Gamma \vdash M \oplus N : \tau}\ \oplus \qquad \frac{\Gamma \vdash V : \textsc{Num} \quad \Gamma \vdash M : \tau \quad \Gamma \vdash N : \tau}{\Gamma \vdash \mathtt{if}\ V\ \mathtt{then}\ M\ \mathtt{else}\ N : \tau}\ \mathsf{I}$$

$$\frac{\Gamma \vdash V_1 : \textsc{Num} \quad \cdots \quad \Gamma \vdash V_n : \textsc{Num}}{\Gamma \vdash f_n(V_1, \ldots, V_n) : \textsc{Num}}\ \mathsf{F}$$

Figure 4.2 Type System Rules

The notions of free and bound occurrences of variables in terms are the usual ones from ordinary λ-calculus, and allow us to define the subsets $\mathbb{CT}$ and $\mathbb{CV}$ (of $\mathbb{T}$ and $\mathbb{V}$, respectively) of *closed terms* and *closed values*.

Definition 4.3 ($\mathsf{PCF}_\oplus$: Type Judgments and Rules)**.** A *type judgment* is an expression in the form $\Gamma \vdash M : \tau$, where M is a term, τ is a type, and Γ is an *environment*, namely a set $\{x_1 : \rho_1, \cdots, x_m : \rho_m\}$ of assignments of types to variables which is non-ambiguous: $x_i = x_j$ implies $i = j$. As customary, such an environment Γ is indicated as $x_1 : \rho_1, \cdots, x_m : \rho_m$, thus omitting parentheses. The *typing rules* for values and terms are in Figure 4.2.

The typing rules as we have introduced them are standard. Just some quick comments are needed about the rule typing binary choices. The way one types $M \oplus N$ is quite restrictive, but anyway very natural: we require M and N to *both* have type τ in order for $M \oplus N$ to have type τ. This constraint might be relaxed by way of a notion of *monadic* typing, which, together with affinity and sized types, enforces termination (Dal Lago and Grellois, 2017).

Example 4.4. As an example of a term, consider the following expression

$$\mathit{GEO} := (\mathtt{fix}\ f.\lambda x.x \oplus (\mathtt{let}\ \mathit{succ}_1(x) = y\ \mathtt{in}\ f\ y))\ 0$$

Actually, *GEO* is nothing more than a $\mathsf{PCF}_\oplus$ term behaving like the hypothetical term $M\ 0$ we were talking about in Section 4.2. Observe how writing it requires

One-Step Reduction

$$(\lambda x.M)V \to \delta(M[V/x])$$
$$\texttt{let } V = x \texttt{ in } M \to \delta(M[V/x])$$
$$\texttt{if } 0 \texttt{ then } M \texttt{ else } N \to \delta(M)$$
$$\texttt{if } r \texttt{ then } M \texttt{ else } N \to \delta(N) \text{ if } r \neq 0$$
$$M \oplus N \to \left\{M : \frac{1}{2}, N : \frac{1}{2}\right\}$$
$$f(r_1, \cdots, r_n) \to \delta(f^*(r_1 \ldots, r_n))$$

$$\frac{M \to \{L_i : p_i\}_{i \in I}}{\texttt{let } M = x \texttt{ in } N \to \{\texttt{let } L_i = x \texttt{ in } N : p_i\}_{i \in I}}$$

Step-Indexed Reduction

$$\frac{}{M \Rightarrow_0 \emptyset} \qquad \frac{}{V \Rightarrow_1 \delta(V)} \qquad \frac{}{V \Rightarrow_{n+1} \emptyset} \qquad \frac{M \to \mathcal{D} \quad \forall N \in \mathsf{SUPP}(\mathcal{D}).N \Rightarrow_n \mathcal{E}_N}{M \Rightarrow_{n+1} \sum_{N \in \mathsf{SUPP}(\mathcal{D})} \mathcal{D}(N) \cdot \mathcal{E}_N}$$

Figure 4.3 Small-Step Distribution Semantics

the presence of a term $succ_1$ among the functions in $\mathcal{F}_1$. As expected, $succ_1^*$ is the successor function. Another key ingredient for writing *GEO* is of course the fixed-point operator `fix`, without which the essentially infinitary behaviour of *GEO* could not be captured.

Typing induces a family $\{\mathbb{CT}_\tau\}_\tau$, where $\mathbb{CT}_\tau$ is the set of all closed terms which can be assigned type τ in the empty environment, i.e. those terms M such that $\emptyset \vdash M : \tau$. Similarly for $\{\mathbb{CV}_\tau\}_\tau$.

4.3.2 Operational Semantics

We are now going to define the operational semantics of the closed terms of $\mathsf{PCF}_\oplus$. We first define a family of *step-indexed reduction relations*, and then take the *operational semantics* of a term as the sum of all its step-indexed approximations. In turn, the step-indexed reduction relation is obtained by convolution from a *one-step reduction relation*. Both these relations rewrite terms into *subdistributions* of terms, which need to be defined formally:

Definition 4.5 (Distributions)**.** Given any set X, a *distribution* on X is a function $\mathcal{D} : X \to \mathbb{R}_{[0,1]}$ such that $\mathcal{D}(x) > 0$ only for denumerably many elements of X and that $\sum_{x \in X} \mathcal{D}(x) \leq 1$. The *support* of a distribution $\mathcal{D}$ on X is the subset $\mathsf{SUPP}(\mathcal{D})$

of X defined as

$$\mathsf{SUPP}(\mathcal{D}) := \{x \in X \mid \mathcal{D}(x) > 0\}.$$

The set of all distributions over X is indicated as $\mathbf{D}(X)$. We indicate the distribution assigning probability 1 to the element $x \in X$ and 0 to any other element of X, the so-called *Dirac distribution on* x, as $\delta(x)$. The *null distribution* $\emptyset \in \mathbf{D}(X)$ assigns 0 to every element of X. The distribution $\mathcal{D}$ on X mapping x_i to p_i for every $i \in \{1, \ldots, n\}$ (and 0 to any other element of X) is indicated as $\{x_1 : p_1, \cdots, x_n : p_n\}$, similarly for the expression $\{x_i : p_i\}_{i \in I}$, where I is any countable index set. Given a distribution $\mathcal{D}$ on X, its *sum* $\|\mathcal{D}\|$ is simply $\sum_{x \in X} \mathcal{D}(x)$.

The one-step reduction relation $\rightarrow$ is a relation between closed terms and distributions over closed terms, i.e., a subset of $\mathbb{CT} \times \mathbf{D}(\mathbb{CT})$ The step-indexed reduction relations are instead a *family* of relations $\{\Rightarrow_n\}_{n \in \mathbb{N}}$, where each $\Rightarrow_n$ is a subset of $\mathbb{CT} \times \mathbf{D}(\mathbb{CV})$. As customary, we write $M \Rightarrow_n \mathcal{D}$ to indicate that $(M, \mathcal{D}) \in \Rightarrow_n$, and similarly for $\rightarrow$. The rules deriving the one-step and step-indexed reduction relations are given in Figure 4.3, and are to be interpreted inductively. Observe that the only rule for $\rightarrow$ allowing for a distribution *not* in the form $\delta(M)$ in the right-hand side is, expectedly, the one for the binary probabilistic choice $M \oplus N$.

Remark 4.6. A quick inspection at the rules in Figure 4.3 reveals that $\rightarrow$ and each of the $\Rightarrow_n$ are *partial functions*: for every $M \in \mathbb{CT}$ and for every $n \in \mathbb{N}$, there are at most one $\mathcal{D} \in \mathbf{D}(\mathbb{CT})$ such that $M \rightarrow \mathcal{D}$ and one $\mathcal{D} \in \mathbf{D}(\mathbb{CV})$ such that $M \Rightarrow_n \mathcal{D}$. This can be proved formally by easy inductions on the structure of M, and on n.

The following is an easy observation, that will be useful in some of the forthcoming sections:

Lemma 4.7. *If* $M \Rightarrow_n \mathcal{D}$, *then* $\mathsf{SUPP}(\mathcal{D})$ *is a finite set.*

If we consider reduction of *typed* closed terms rather than mere terms, classic results in the theory of λ-calculus continue to hold. On the one hand, reduction can only get stuck at values:

Proposition 4.8 (Progress)**.** *For every* $M \in \mathbb{CT}_\tau$, *either* M *is a value or there is* $\mathcal{D}$ *with* $M \rightarrow \mathcal{D}$.

Proof The proof is by induction on the structure of any type derivation for M, whose conclusion has to be in the form $\emptyset \vdash M : \tau$. □

On the other hand, reduction preserves typing:

Proposition 4.9 (Subject Reduction)**.** *For every* $M \in \mathbb{CT}_\tau$ *and for every* $n \in \mathbb{N}$, *if* $M \rightarrow \mathcal{D}$ *and* $M \Rightarrow_n \mathcal{E}$, *then* $\mathcal{D} \in \mathbf{D}(\mathbb{CT}_\tau)$ *and* $\mathcal{E} \in \mathbf{D}(\mathbb{CV}_\tau)$.

Proof The proof, as usual, consists in first proving a Substitution Lemma, followed by some case analysis on the rules used to derive that $M \to \mathcal{D}$. The generalisation to $\Rightarrow_n$ can be proved by an induction on n. □

Example 4.10. By applying the rules in Figure 4.3, one easily derives that

$$GEO \Rightarrow_3 \left\{0 : \frac{1}{2}\right\}; \qquad GEO \Rightarrow_8 \left\{1 : \frac{1}{4}\right\}; \qquad GEO \Rightarrow_{13} \left\{2 : \frac{1}{8}\right\}.$$

More generally, $GEO \Rightarrow_{3+5n} \left\{n : \frac{1}{2^{n+1}}\right\}$ for every n, while $GEO \Rightarrow_m \emptyset$ whenever m cannot be written as $3 + 5n$.

An interesting consequence of Progress and Subject Reduction is that $\Rightarrow_n$ becomes a total function on closed typable terms:

Corollary 4.11. *For every $M \in \mathbb{CT}_\tau$ and for every $n \in \mathbb{N}$, there is exactly one distribution $\mathcal{D}_n$ such that $M \Rightarrow_n \mathcal{D}_n$*

The distribution $\mathcal{D}_n$ from Corollary 4.11 will sometimes be indicated as $\langle M \rangle_n$. Noticeably, not only $\langle M \rangle_n$ but also $\sum_{m=0}^{n} \langle M \rangle_m$ is a distribution, as can be easily proved by induction on n.

Definition 4.12 (Pointwise Order on Distributions)**.** Given two distributions $\mathcal{D}, \mathcal{E} \in \mathbf{D}(X)$, we write $\mathcal{D} \leq \mathcal{E}$ iff $\mathcal{D}(x) \leq \mathcal{E}(x)$ for every $x \in X$. This relation endows $\mathbf{D}(X)$ with the structure of a partial order, which is actually an ω**CPO**: every ω-chain $\{\mathcal{D}_n\}_{n \in \mathbb{N}}$ of distributions has a least upper bound, which is defined pointwise:

$$\sup\{\mathcal{D}_n\}_{n \in \mathbb{N}} = x \mapsto \sup_{n \in \mathbb{N}} \mathcal{D}_n(x).$$

That this is a good definition ultimately descends from the fact that $\mathbb{R}_{[0,1]}$ is itself and ω**CPO**.

We are now ready to give the most important definition of this section:

Definition 4.13 (Operational Semantics of Closed Terms.)**.** Given a closed term $M \in \mathbb{CT}_\tau$, the *operational semantics* of M is defined to be the distribution $\langle M \rangle \in \mathbf{D}(\mathbb{CV}_\tau)$ defined as $\sum_{n \in \mathbb{N}} \langle M \rangle_n = \sup_{m \in \mathbb{N}} \sum_{n=0}^{m} \langle M \rangle_n$. That this is a well-posed definition is a consequence of $\sum_{n=0}^{m} \langle M \rangle_n$ being an ω-chain in $\mathbf{D}(\mathbb{CV}_\tau)$.

Example 4.14. It is routine to check that

$$\langle GEO \rangle = \sum_{n \in \mathbb{N}} \langle GEO \rangle_n = \left\{n : \frac{1}{2^{n+1}}\right\}_{n \in \mathbb{N}}.$$

In other words, the operational semantics of *GEO* is indeed the geometric distribution on the natural numbers.

Term Contexts	$C_{\mathbb{T}}, D_{\mathbb{T}} ::= C_{\mathbb{V}} \mid [\cdot] \mid C_{\mathbb{V}}\, V \mid V\, C_{\mathbb{V}}$
	$\mid \texttt{let}\ C_{\mathbb{T}} = x\ \texttt{in}\ N$
	$\mid \texttt{let}\ M = x\ \texttt{in}\ C_{\mathbb{T}} \mid C_{\mathbb{T}} \oplus D_{\mathbb{T}}$
	$\mid \texttt{if}\ V\ \texttt{then}\ C_{\mathbb{T}}\ \texttt{else}\ D_{\mathbb{T}}$
Value Contexts	$C_{\mathbb{V}}, D_{\mathbb{V}} ::= \lambda x.C_{\mathbb{T}} \mid \texttt{fix}\ x.C_{\mathbb{V}}$

Figure 4.4 Value and Term Contexts

An easy corollary of Subject Reduction is that $\langle M\rangle \in \mathbf{D}(\mathbb{CV}_\tau)$ whenever $M \in \mathbb{CT}_\tau$. In other words, the operational semantics respects types.

4.3.3 Contextual Equivalence

Once an operational semantics is given, the next step to be taken towards building a metatheory for $\mathsf{PCF}_\oplus$ consists in endowing it with a notion of *program equivalence*: when is it reasonable to dub two programs to be equivalent, i.e., to have the same behaviour? A first answer to the question above consists in stipulating that equivalent programs should behave the same when placed in any context.

Definition 4.15 ($\mathsf{PCF}_\oplus$: Contexts)**.** *Value and term contexts* are defined in Figure 4.4. Given a term context $C_{\mathbb{T}}$ and a term M, the expression $C_{\mathbb{T}}[M]$ stands for the term obtained by substituting the (unique) occurrence of $[\cdot]$ in $C_{\mathbb{T}}$ with M. Similarly for $C_{\mathbb{V}}[M]$, which is by construction a *value*. When we speak of a *context*, what we are referring to is a *term* context, a concept more general than the one of a value context. Metavariables like C or D refer to *term* contexts.

It is easy to generalise the type system as we presented it in Section 4.3 to a formal system capable of deriving judgments in the form

$$\Gamma \vdash C[\Delta \vdash \cdot : \tau] : \rho.$$

The judgment above, when provable, ensures that the ordinary type judgment $\Gamma \vdash C[M] : \rho$ is provable whenever $\Delta \vdash M : \tau$ is itself derivable.

We are now ready to give the most important definition of this section.

Definition 4.16 (Contextual Equivalence)**.** Given two terms M, N such that $\Gamma \vdash M : \tau$ and $\Gamma \vdash N : \tau$, we say that M and N are (Γ, τ)-*equivalent*, and we write $M \equiv^{\tau}_{\Gamma} N$ iff whenever $\emptyset \vdash C[\Gamma \vdash \cdot : \tau] : \textsc{Unit}$, it holds that $\|\langle C[M]\rangle\| = \|\langle C[N]\rangle\|$.

The way we have defined contextual equivalence turns it into a *typed relation*, i.e. a family $\{\mathcal{R}^\tau_\Gamma\}_{\Gamma,\tau}$ of relations such that $M\mathcal{R}^\tau_\Gamma N$ implies $\Gamma \vdash M : \tau$ and $\Gamma \vdash N : \tau$. We say that any typed relation $\{\mathcal{R}^\tau_\Gamma\}_{\Gamma,\tau}$ is:

- A *congruence* iff each $\mathcal{R}^\tau_\Gamma$ is an equivalence and whenever $M\mathcal{R}^\tau_\Gamma N$ and $\Delta \vdash C[\Gamma \vdash \cdot : \tau] : \rho$, it holds that $C[M]\mathcal{R}^\rho_\Delta C[N]$.
- *Adequate* iff whenever $M\mathcal{R}^\tau_\Gamma N$ and $\emptyset \vdash C[\Gamma \vdash \cdot : \tau] : \textsc{Unit}$, it holds that $\|\langle C[M]\rangle\| = \|\langle C[N]\rangle\|$.

In fact contextual equivalence is an adequate congruence, and is the largest such typed relation, as witnessed by the following result, whose proof is easy, and whose deterministic counterpart has been known since the inception of contextual equivalence (Morris, 1969):

Proposition 4.17. *Contextual equivalence is the largest adequate congruence.*

Contextual equivalence is thus a very satisfactory notion of program equivalence. This does not mean, however, that proving pairs of terms to be contextually equivalent is easy: the universal quantification over all contexts Definition 4.16 relies on makes concrete proofs of equivalence hard. This has stimulated many investigations on alternative notions of equivalence, not only in presence of probabilistic choice, but also in the realm of usual, deterministic λ-calculi.

4.3.4 On the Expressive Power of $\mathsf{PCF}^{\mathbb{N}}_{\oplus}$

It is well-known that the class of partial functions on the natural numbers Plotkin's PCF can represent is precisely the class of partial recursive functions (see (Longley and Normann, 2015) for a proof). But how about $\mathsf{PCF}_\oplus$? First of all, is there any analogue to the class of partial recursive functions if the underlying computation model is probabilistic rather than deterministic?

There are two essentially different ways in which one can answer the question above. The first one consists in seeing any probabilistic computational model (e.g. probabilistic Turing machines (De Leeuw et al., 1956; Santos, 1969) as a device meant to solve *ordinary* computational problems seen as functions or languages. If one proceeds this way, one soon realises that, under mild conditions, any probabilistic computational model only decides partial recursive functions, capturing all of them when the underlying deterministic machinery is sufficiently powerful (Santos, 1969).

Another route consists in looking at probabilistic computational models as devices computing functions from the natural numbers to *distributions* of natural numbers, called a *probabilistic function*. This is the route followed by the author in his work with Gabbrielli and Zuppiroli (Dal Lago et al., 2014b), in which a variation on Kleene's function algebra is proved to capture *probabilistic computable functions*,

namely those probabilistic functions which can be represented by probabilistic Turing machines. In the rest of this Section, we give an overview of this result, only taking the definition of a probabilistic Turing machine for granted.

Let us start by defining probabilistic functions and their computability:

Definition 4.18 (Computable Probabilistic Functions)**.** Any function $f : \mathbb{N} \to \mathbf{D}(\mathbb{N})$ is said to be a *probabilistic function*. Such a probabilistic function f is said to be *Turing-computable* (or just *computable*) iff there is a probabilistic Turing machine $\mathscr{M}$ which, when fed with the encoding of $n \in \mathbb{N}$ terminates producing in output an encoding of $m \in \mathbb{N}$ with probability $f(n)(m)$.

A quite similar definition can be obtained by replacing Turing machines with $\mathsf{PCF}_\oplus$:

Definition 4.19. Let M be a $\mathsf{PCF}_\oplus^{\mathbb{N}}$ closed term of type $\textsc{Num} \to \textsc{Num}$. Then M is said to *compute* a probabilistic function $f : \mathbb{N} \to \mathbf{D}(\mathbb{N})$ if and only if for every $n \in \mathbb{N}$, it holds that $\langle M\ n \rangle = f(n)$. Such a probabilistic function f is in this case said to be *computable by* $\mathsf{PCF}_\oplus^{\mathbb{N}}$.

One may wonder how endowing either Turing machines or PCF with an operation for probabilistic choice affects the underlying expressive power. In fact, the obtained computational models remain equivalent:

Theorem 4.20. *The class of computable probabilistic functions coincides with the class of probabilistic functions computable by* $\mathsf{PCF}_\oplus^{\mathbb{N}}$.

Proof Proving that probabilistic functions computable by $\mathsf{PCF}_\oplus^{\mathbb{N}}$ are also Turing computable is straightforward, since the operational semantics of $\mathsf{PCF}_\oplus^{\mathbb{N}}$ is effective, thus implementable by a Turing machine. The converse implication can be easily proved via the already mentioned characterisation of probabilistic computable functions via a slight variation of Kleene partial recursive functions (Dal Lago et al., 2014b), namely one in which a basic function modelling probabilistic choice is present. □

4.3.5 Termination in $\mathsf{PCF}_\oplus$

As we mentioned in the Introduction, techniques ensuring termination of probabilistic programs have already been investigated, and at least *two* distinct notions of termination for probabilistic programs have been introduced, namely *almost sure* termination, and its strengthening *positive* almost sure termination. Let us first of all see how these notions can be formalised in our setting.

Definition 4.21 (Almost Sure Termination)**.** Let M be any closed term. We say

that M is *almost surely terminating* if $\|\langle M\rangle\| = 1$, namely if its probability of convergence is 1.

Example 4.22. An example of an almost surely terminating term is certainly *GEO* from Example 4.4. Not all $\mathsf{PCF}_{\oplus}^{\mathbb{N}}$ terms are almost surely terminating, however. As an example, the term $M = (\mathtt{fix}\ f.f)\ 0$ is clearly not terminating because $M \to \delta(M)$, and as a consequence $\langle M\rangle_n = \emptyset$ for every natural number n. There are subtler examples, like the following variation on *GEO*:

$$\mathit{TWICE} := (\mathtt{fix}\ f.\lambda x.x \oplus (\mathtt{let}\ \mathit{succ}_1(x) = y\ \mathtt{in}\ f\ (f\ y)))\ 0.$$

Please observe how the only difference between *GEO* and *TWICE* lies in the presence of *two* nested recursive calls to f (instead of one) in the right-hand-side of the probabilistic choice operator. The reader is invited to derive a value for $\|\langle \mathit{TWICE}\rangle\|$ as an exercise.

There is nothing in the definition of an almost surely terminating term which ensures the term's expected number of reduction steps to be *finite*. Enforcing it requires an additional, further, constraint. But before introducing it, let us first define the *expected reduction length* of any closed term M. As already mentioned in Section 4.2, a convenient way to define it is as

$$\sum_{m=0}^{\infty} \mathsf{Pr}(T > m),$$

where T is the random variable counting the number of steps M requires to be reduced to a value. Now, how can we define $\mathsf{Pr}(T > m)$ in terms of the operational semantics of M? Actually, an easy way to do it is to observe that $\mathsf{Pr}(T > m)$ is $1 - \sum_{n=0}^{m} \|\langle M\rangle_n\|$: the quantity $\sum_{n=0}^{m} \|\langle M\rangle_n\|$ precisely captures the probability for M to reduce to a value in *at most m* reduction steps. As a consequence, *the average number of reduction steps ExLen*(M) for M is defined as follows:

$$\mathit{ExLen}(M) := \sum_{m=0}^{\infty} \left(1 - \sum_{n=0}^{m} \|\langle M\rangle_n\|\right).$$

Definition 4.23 (Positive Almost Sure Termination)**.** Let M be any closed term. We say that M is *positively almost surely terminating* if $\mathit{ExLen}(M) < +\infty$.

The following is easy to prove, and is a standard result in the theory of Markov chains and processes:

Lemma 4.24. *Every positively almost-surely terminating term is almost-surely terminating.*

Proof A necessary condition for $ExLen(M)$ to be finite is that

$$\lim_{m\to+\infty}\sum_{n=0}^{m}\|\langle M\rangle_n\| = \lim_{m\to+\infty}\left\|\sum_{n=0}^{m}\langle M\rangle_n\right\| = 1,$$

which can hold only if $\|\langle M\rangle\| = 1$, namely only if M is almost surely terminating. □

The converse implication does not hold however, as witnessed by the following example.

Example 4.25. Consider the following $\mathsf{PCF}^{\mathbb{N}}_{\oplus}$ term:

$$RW := (\mathtt{fix}\ f.\lambda x.\mathtt{if}\ gt_2(x,0)\ \mathtt{then}\ (N_{\Uparrow}\oplus N_{\Downarrow})\ \mathtt{else}\ 0)1,$$

where

$$N_{\Uparrow} := \mathtt{let}\ succ_1(x) = y\ \mathtt{in}\ f\ y$$
$$N_{\Downarrow} := \mathtt{let}\ pred_1(x) = y\ \mathtt{in}\ f\ y$$

The recursive function on which the term is based first tests whether the argument x (of type Num) is positive, and in case it is, makes a recursive call with argument either decreased or increased by 1, each with equal probability $\frac{1}{2}$. The term RW, then, can be seen as modelling an unbounded, fair, random walk. As such it is well known to be almost surely terminating, but not positively: the average number of steps which are necessary to reach the base case is *infinite*.

Is there any reasonable way to restrict, e.g., $\mathsf{PCF}^{\mathbb{N}}_{\oplus}$ in such a way as to *enforce* almost-sure termination? The answer is positive. As an example:

- Removing fixpoints from the class of terms, replacing them with primitive recursion, namely with a combinator `rec` having type

  $$(\textsc{Num}\to\tau\to\tau)\to\tau\to\textsc{Num}\to\tau$$

  turns the calculus into a probabilistic variation on Gödel's T, that we call $\mathsf{T}_{\oplus}$. Terms of the calculus are not only positively almost-surely terminating, but satisfy an even stronger constraint: there is a global, uniform, bound on the length of any probabilistic branch, i.e., for every $M\in\mathbb{CT}_\tau$ there is $n\in\mathbb{N}$ such that $\langle M\rangle_m = \emptyset$ for every $m\geq n$. By Lemma 4.7, we can conclude that $\mathsf{SUPP}(\langle M\rangle)$ is finite, i.e. that $\mathsf{T}_{\oplus}$ is an essentially finitary calculus. Notice that such a uniform bound cannot be found for terms like *GEO* which, although being almost surely terminating, can possibly diverge (of course with null probability).
- $\mathsf{T}_{\oplus}$ can be made infinitary by endowing it with a primitive `geo` behaving exactly as *GEO*, and having type Num. Remarkably, this apparently innocuous change has the effect of making the calculus almost surely terminating, but *not* positively so.

4.3.6 Further Reading

In the previous sections, a brief introduction to randomised λ-calculi has been given. This Section is meant to be a repository of pointers to the literature about this class of idioms, without any hope of being comprehensive.

We have specified the operational semantics of $\mathsf{PCF}_{\oplus}$ in small-step style, and by way of an inductively defined set of rules. This is not, however, the only option. Indeed, one could certainly go big-step, but also interpret reduction rules coinductively (Dal Lago and Zorzi, 2012). Another choice we have implicitly made when introducing $\mathsf{PCF}_{\oplus}$ is to consider *call-by-value* rather than *call-by-name* evaluation. This, again, does not mean that the latter is a route which cannot be followed (Dal Lago and Zorzi, 2012; Danos and Ehrhard, 2011). What makes call-by-value more appealing is the possibility of "implicitly memoizing" the result of probabilistic choices by way of sequencing, something which is not available in call-by-name. Consider, as an example, a term M in which some probabilistic choice $(\lambda x.N) \oplus (\lambda x.L)$ occurs. In call-by-value evaluation, this occurrence can be copied unevaluated, and once one copy of it is indeed evaluated (i.e. when it becomes the argument of a `let`), the outcome of the probabilistic choice *can itself be copied*. In call-by-name, at least if sequencing is not available, this is simply impossible: once (a copy of) a subterm is evaluated, the result of its evaluation cannot be spread around the term by way of copying. This, by the way, is the source of some discrepancies between the nature of contextual equivalence in call-by-name and call-by-value evaluation, which shows up when probabilistic choice is available (Crubillé and Dal Lago, 2014).

Contextual equivalence, as we have introduced it, is a very satisfactory definition of equivalence for probabilistic programs, being the largest compatible and adequate (equivalence) relation. Contextual equivalence relying on a universal quantification over all contexts, however, makes concrete proofs of equivalence quite hard. Alternative methodologies have been introduced for the purpose of making proofs of equivalence easier in a probabilistic setting, following the extended body of work about the same problem in the deterministic setting (e.g. (Plotkin, 1973; Abramsky, 1990; Mitchell, 1996)). For example, step-indexed logical relations have been adapted to an higher-order probabilistic λ-calculus, and proved not only sound for contextual equivalence, but also complete (Bizjak and Birkedal, 2015). As an another example, Abramsky's applicative bisimilarity has been itself generalised to randomised λ-calculi (Dal Lago et al., 2014a; Crubillé and Dal Lago, 2014) and proved fully-abstract, the latter holding only when call-by-*value* evaluation is considered. Finally, a variation on the notion of Böhm tree (Barendregt, 1984) has been recently defined for an untyped, randomised λ-calculus (Leventis, 2018); quite interestingly, the classic separability result continues to hold.

Another way of proving terms to be equivalent consists in comparing their meanings in any adequate model, along the lines of so-called denotational semantics. Following this path has proved remarkably hard in randomised λ-calculi. In particular, coming up with a completely satisfactory notion of probabilistic powerdomain, this way modelling probabilistic choice in monadic style has proved to be impossible (Jones and Plotkin, 1989; Jung and Tix, 1998). On the other hand, interpreting $\mathsf{PCF}_{\oplus}$ by denotational models in the style of coherence spaces (Danos and Ehrhard, 2011) is indeed possible, and also leads to *full abstraction* results (Ehrhard et al., 2014). Very recently, another way of giving semantics to randomised λ-calculi and based on Boolean-Valued models has been proposed (Bacci et al., 2018).

The two notions of termination for probabilistic programs we introduced and discussed in this section have been studied in depth in the context of imperative programming languages (McIver and Morgan, 2005), and have been proved to be strictly more difficult than their deterministic counterparts, recursion-theoretically (Kaminski and Katoen, 2015). Various techniques for proving imperative programs to be terminating have been introduced, based on the notion of ranking martingale or Lyapunov function, the natural probabilistic analogues of the so-called ranking function (Bournez and Garnier, 2005). While the same technique has been shown to be applicable to term rewrite systems recently (Avanzini et al., 2018), not much is known about its applicability to *higher-order* functional programs. Up to now, the only works in this directions are based on type systems, and in particular on variations on either sized-types (Dal Lago and Grellois, 2017) or intersection types (Breuvart and Dal Lago, 2018).

4.4 Sampling and Conditioning

In randomised λ-calculi, only one form of probabilistic choice is available, which most often has a discrete nature. As we argued in the last section, this is perfectly adequate to model randomised computation. In recent years, starting from the pioneering works on languages like CHURCH and ANGLICAN, functional programming languages have also been employed as vehicles for the representation of probabilistic *models* rather than *algorithms*. This amounts to a different execution model, in which *inference* takes the place of *evaluation*.

In this Section, we give some hints about how this style of programming can be modelled in an extension of $\mathsf{PCF}_{\oplus}$, that we call $\mathsf{PCF}_{\texttt{sample},\texttt{score}}$. The latter calculus can be derived from the former by:

- Replacing binary probabilistic choice with an operator `sample` which, when evaluated, samples a real number in $[0,1]$ uniformly at random. This implies, in particular, that the underlying monoid $\mathbb{M}$ needs to include $[0,1]$, and is often

Terms $M, N ::= \texttt{sample} \mid \texttt{score}(V).$

Typing Rules

$$\frac{}{\Gamma \vdash \texttt{sample} : \textsc{Num}}\ \mathsf{A} \qquad \frac{\Gamma \vdash V : \textsc{Num}}{\Gamma \vdash \texttt{score}(V) : \textsc{Unit}}\ \mathsf{C}$$

Figure 4.5 Grammar and Typing Rules for `sample` and `score`.

taken as the additive monoid of real numbers. The type of the new term `sample` is Num.

- Endowing the class of terms with another operator, called `score`, which takes a positive real number r as a parameter, and modifies the *weight* of the current probabilistic branch by multiplying it by r. This serves as a way to take *observations* into account, by *conditioning* on them. The type of `score` is Unit, while its argument must of course have type Num.

Formally, the language of terms and the typing rules are extended as shown in Figure 4.5. From a purely syntactical point of view, then, formally defining Bayesian λ-calculi poses absolutely no problem.

What is nontrivial, however, is to give a *meaning* to those calculi, even in the form of an operational semantics. As already mentioned, terms are not meant to model *algorithms* but *probabilistic models*, on which inference is supposed to take place. This can be dealt with by defining a *sampling* operational semantics, following (Borgström et al., 2016), or by a *distribution* semantics which, however, requires some nontrivial measure theory. We will deal with them in the following section.

4.4.1 Operational Semantics

How could we give an operational semantics to $\mathsf{PCF}_{\texttt{sample},\texttt{score}}$? As we already mentioned, there are at least two answers to this questions, which lead to formal systems which are related, although having distinct properties.

One may first of all wonder whether the distribution semantics we introduced for $\mathsf{PCF}_{\oplus}$ could be adapted to $\mathsf{PCF}_{\texttt{sample},\texttt{score}}$. In fact this can be done, at the price of making the whole development nontrivial, due to the underlying measure theory. To understand why this is the case, let us consider how the evaluation rule for `sample` would look like. At the right-hand-side of it, what we expect is a distribution $\mathcal{D}$ on $\mathbb{R}$. The latter's support, however, is $\mathbb{R}_{[0,1]}$, and is thus not countable. As a consequence, one needs to switch to *measures* in the sense of measure theory, and assume the

underlying set, namely $\mathbb{R}$ to have the structure of a measurable space. Adapting the rule for `let`-terms naturally leads to

$$\frac{M \to \mu}{\texttt{let } M = x \texttt{ in } N \to \texttt{let } \mu = x \texttt{ in } N}$$

where $\texttt{let } \mu = x \texttt{ in } N$ should itself be a measure, meaning that not only *real numbers*, but also *terms* should be endowed with the structure of a measurable space. Finally, the last rule in step-indexed reduction needs to be adapted itself, the summation in its conclusion to be replaced by an integral:

$$\frac{M \to \mu \quad \forall N \in \mathsf{SUPP}(\mu).N \Rightarrow_n \sigma_N}{M \Rightarrow_{n+1} A \mapsto \int \sigma_N(A)\mu(dN)}$$

For the integral above to be well defined, the underlying function must be measurable. It turns out that the appropriate invariant is even stronger: each $\Rightarrow_{n+1}$ can be seen as *finite* kernel, and the operational semantics of M thus becomes an s-finite kernel (Staton, 2017).

Measure-theoretic distribution semantics, however, is not the only way a calculus like $\mathsf{PCF}_{\texttt{sample},\texttt{score}}$ can be given a meaning. An alternative consists of going for the so-called *sampling-based* semantics, in which the process of sampling and scoring are made explicit.

Definition 4.26 (Trace)**.** A *trace* is a possibly empty finite sequence of elements from $\mathbb{R}_{[0,1]}$ and is indicated with metavariables like s, t. The trace whose first element is $r \in \mathbb{R}_{[0,1]}$, and whose other elements form a trace t is indicated as $r :: t$. The set of all traces is $\mathbb{X}$.

In sampling-based semantics, *one-step reduction* and *multi-step reduction* are both subsets of $(\mathbb{CT} \times \mathbb{X}) \times \mathbb{R}_+ \times (\mathbb{CT} \times \mathbb{X})$, i.e. ternary rather than binary relations. They are indicated as $\twoheadrightarrow$ and $\Rrightarrow$, respectively. We write $\langle M, s\rangle \overset{r}{\twoheadrightarrow} \langle N, t\rangle$ for $((M, s), r, (N, t)) \in \twoheadrightarrow$. Similarly for $\langle M, s\rangle \overset{r}{\Rrightarrow} \langle N, t\rangle$. Rules for small-step sampling semantics can be found in Figure 4.6. Observe that if $\langle M, s\rangle \overset{r}{\twoheadrightarrow} \langle N, t\rangle$ and $r \neq 1$, then the redex fired in M must be of the form $\texttt{score}(r)$.

The two kinds of semantics can be proved equivalent, in the following sense: for every measurable set of real numbers A, the total probability of observing A when evaluating $\emptyset \vdash M : \textsc{Num}$ in sampling-based and distribution-based are the same (see, e.g., (Borgström et al., 2016) for some more details).

4.4.2 Further Reading

The study of Bayesian λ-calculi is in its infancy , and the underlying metatheory, despite some breakthrough advances in the last ten years, is still underdeveloped compared to the one of randomised λ-calculi.

One-Step Reduction

$$\langle (\lambda x.M)V, s\rangle \xrightarrow{1}\!\!\!\!\rightarrow \langle M[V/x], s\rangle$$

$$\langle \texttt{let } V = x \texttt{ in } M, s\rangle \xrightarrow{1}\!\!\!\!\rightarrow \langle M[V/x], s\rangle$$

$$\langle \texttt{if } 0 \texttt{ then } M \texttt{ else } N, s\rangle \xrightarrow{1}\!\!\!\!\rightarrow \langle M, s\rangle$$

$$\langle \texttt{if } r \texttt{ then } M \texttt{ else } N, s\rangle \xrightarrow{1}\!\!\!\!\rightarrow \langle N, s\rangle \text{ if } r \neq 0$$

$$\langle \texttt{sample}, r :: s\rangle \xrightarrow{1}\!\!\!\!\rightarrow \langle r, s\rangle$$

$$\langle \texttt{score}(r), s\rangle \xrightarrow{r}\!\!\!\!\rightarrow \langle \star, s\rangle$$

$$\langle f(r_1, \cdots, r_n), s\rangle \xrightarrow{1}\!\!\!\!\rightarrow \langle f^*(r_1 \ldots, r_n), s\rangle$$

$$\frac{\langle M, s\rangle \xrightarrow{r}\!\!\!\!\rightarrow \langle L, t\rangle}{\langle \texttt{let } M = x \texttt{ in } N, s\rangle \xrightarrow{r}\!\!\!\!\rightarrow \langle \texttt{let } L = x \texttt{ in } N, s\rangle}$$

Multi-Step Reduction

$$\frac{}{\langle V, s\rangle \overset{1}{\Rightarrow} \langle V, s\rangle} \qquad \frac{\langle M, s\rangle \xrightarrow{r}\!\!\!\!\rightarrow \langle N, t\rangle \quad \langle N, t\rangle \overset{s}{\Rightarrow} \langle L, u\rangle}{\langle M, s\rangle \overset{r\cdot s}{\Rightarrow} \langle L, u\rangle}$$

Figure 4.6 Small-Step Sampling Semantics

Calculi in which continuous distributions and sampling from them are available were introduced by Ramsey and Pfeffer (Ramsey and Pfeffer, 2002) and Park, Pfenning, and Thrun (Park et al., 2005). The first example of a λ-calculus in which these two features are both present is due to the author, together with Börgstrom, Gordon, and Szymczak (Borgström et al., 2016), who introduced trace and distribution semantics for an *untyped* bayesian λ-calculus with primitives for sampling and scoring, together with trace and distribution semantics, both in small-step *and* big-step styles.

Contextual equivalence and logical relations for a typed λ-calculus with sampling and scoring were introduced in Culpepper and Cobb (2017), and adapted to a calculus with full higher-order recursion in Wand et al. (2018).

Giving a satisfactory denotational semantics for bayesian λ-calculi has been proved to be quite challenging. Quasi-borel spaces (Heunen et al., 2017) provide both a closed structure and the machinery necessary to model sampling and conditioning, something which is provably impossible in the category of measurable

spaces. Subsequently, quasi-borel spaces have also been shown to give rise to a category of domains, thus accounting for the presence of recursion in the underlying calculus (Vákár et al., 2019). Generalising probabilistic coherent spaces (Danos and Ehrhard, 2011) to a calculus allowing sampling from continuous distributions has proved to be possible, but highly nontrivial (Ehrhard et al., 2018). At the time of writing, it is not clear whether all this scales to a calculus in which a general form of conditioning (as embodied by the `score` operator) is available.

References

Abramsky, Samson. 1990. The lazy lambda calculus. Pages 65–117 of: Turner, D. (ed), *Research Topics in Functional Programming*. Addison Wesley.

Avanzini, Martin, Lago, Ugo Dal, and Yamada, Akihisa. 2018. On probabilistic term rewriting. Pages 132–148 of: *Proc. of FLOPS 2018*.

Bacci, Giorgio, Furber, Robert, Kozen, Dexter, Mardare, Radu, Panangaden, Prakash, and Scott, Dana. 2018. Boolean-valued semantics for the stochastic λ-calculus. Pages 669–678 of: *Proc. of LICS 2018*.

Barendregt, Henk P. 1984. *The Lambda Calculus, Its Syntax and Semantics*. Elsevier.

Bizjak, Ales, and Birkedal, Lars. 2015. Step-indexed logical relations for probability. Pages 279–294 of: *Proc. of FoSSaCS 2015*.

Borgström, Johannes, Dal Lago, Ugo, Gordon, Andrew D., and Szymczak, Marcin. 2016. A lambda-calculus foundation for universal probabilistic programming. Pages 33–46 of: *Proc. of ICFP 2016*.

Bournez, Olivier, and Garnier, Florent. 2005. Proving positive almost-sure termination. Pages 323–337 of: *Proc. of RTA 2005*.

Breuvart, Flavien, and Dal Lago, Ugo. 2018. On intersection types and probabilistic lambda calculi. Pages 8:1–8:13 of: *Proc. of PPDP 2018*.

Crubillé, Raphaëlle, and Dal Lago, Ugo. 2014. On Probabilistic applicative bisimulation and call-by-value λ-calculi. Pages 209–228 of: *Proc. of ESOP 2014*.

Culpepper, Ryan, and Cobb, Andrew. 2017. Contextual equivalence for probabilistic programs with continuous random variables and scoring. Pages 368–392 of: *Proc. of ESOP 2017*.

Dal Lago, Ugo, and Grellois, Charles. 2017. Probabilistic termination by monadic affine sized typing. Pages 393–419 of: *Proc. of ESOP 2017*.

Dal Lago, Ugo, and Zorzi, Margherita. 2012. Probabilistic operational semantics for the lambda calculus. *RAIRO – Theor. Inf. and Applic.*, **46**(3), 413–450.

Dal Lago, Ugo, Sangiorgi, Davide, and Alberti, Michele. 2014a. On coinductive equivalences for higher-order probabilistic functional programs. Pages 297–308 of: *Proc. of POPL 2014*.

Dal Lago, Ugo, Zuppiroli, Sara, and Gabbrielli, Maurizio. 2014b. Probabilistic recursion theory and implicit computational complexity. *Sci. Ann. Comp. Sci.*, **24**(2), 177–216.

Danos, Vincent, and Ehrhard, Thomas. 2011. Probabilistic coherence spaces as a model of higher-order probabilistic computation. *Inf. Comput.*, **209**(6), 966–991.

Danos, Vincent, and Harmer, Russell. 2002. Probabilistic game semantics. *ACM Trans. Comput. Log.*, **3**(3), 359–382.

De Leeuw, Karel, Moore, Edward F, Shannon, Claude E, and Shapiro, Norman. 1956. Computability by probabilistic machines. *Automata Studies*, **34**, 183–198.

Ehrhard, Thomas, Pagani, Michele, and Tasson, Christine. 2011. The computational meaning of probabilistic coherence spaces. Pages 87–96 of: *Proc. of LICS 2011*.

Ehrhard, Thomas, Pagani, Michele, and Tasson, Christine. 2014. Probabilistic coherence spaces are fully abstract for probabilistic PCF. Pages 309–320 of: *Proc. of POPL 2014*.

Ehrhard, Thomas, Pagani, Michele, and Tasson, Christine. 2018. Measurable cones and stable, measurable functions: a model for probabilistic higher-order programming. Pages 59:1–59:28 of: *Proc. of POPL 2018*.

Goodman, Noah D., Mansinghka, Vikash K., Roy, Daniel M., Bonawitz, Keith, and Tenenbaum, Joshua B. 2008. Church: a language for generative models. Pages 220–229 of: *Proc. of UAI 2008*.

Heunen, Chris, Kammar, Ohad, Staton, Sam, and Yang, Hongseok. 2017. A convenient category for higher-order probability theory. Pages 1–12 of: *Proc. of LICS 2017*.

Hindley, J. Roger, and Seldin, Jonathan P. 2008. *Lambda-Calculus and Combinators: An Introduction*. 2 edn. Cambridge University Press.

Jones, Claire, and Plotkin, Gordon D. 1989. A probabilistic powerdomain of evaluations. Pages 186–195 of: *Proc. of LICS 1989*.

Jung, Achim, and Tix, Regina. 1998. The troublesome probabilistic powerdomain. *Electr. Notes Theor. Comput. Sci.*, **13**, 70–91.

Kaminski, Benjamin Lucien, and Katoen, Joost-Pieter. 2015. On the Hardness of Almost-Sure Termination. Pages 307–318 of: *Proc. of MFCS 2015*.

Koller, Daphne, and Friedman, Nir. 2009. *Probabilistic Graphical Models: Principles and Techniques – Adaptive Computation and Machine Learning*. The MIT Press.

Kozen, Dexter. 1981. Semantics of Probabilistic Programs. *J. Comput. Syst. Sci.*, **22**(3), 328–350.

Leventis, Thomas. 2018. Probabilistic Böhm Trees and Probabilistic Separation. Pages 649–658 of: *Proc. of LICS 2018*.

Longley, John, and Normann, Dag. 2015. *Higher-order Computability*. Theory and Applications of Computability. Springer.

Manning, Christopher D, and Schütze, Hinrich. 1999. *Foundations of Statistical Natural Language Processing*. Vol. 999. MIT Press.

McIver, Annabelle, and Morgan, Carroll. 2005. *Abstraction, Refinement and Proof for Probabilistic Systems*. Monographs in Computer Science. Springer.

Mitchell, John C. 1996. *Foundations of Programming Languages*. Cambridge, MA, USA: MIT Press.

Morris, James. 1969. *Lambda Calculus Models of Programming Languages*. Ph.D. thesis, Massachusetts Institute of Technology, Alfred P. Sloan School of Management.

Motwani, Rajeev, and Raghavan, Prabhakar. 1995. *Randomized Algorithms*. Cambridge University Press.

Park, Sungwoo, Pfenning, Frank, and Thrun, Sebastian. 2005. A probabilistic language based upon sampling functions. Pages 171–182 of: *Proc. of POPL 2005*. ACM.

Pearl, Judea. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann.

Plotkin, Gordon. 1973. *Lambda-definability and logical relations*. Tech. rept. SAI-RM-4. University of Edinburgh.

Prince, Simon J. D. 2012. *Computer Vision: Models, Learning, and Inference*. New York, NY, USA: Cambridge University Press.

Ramsey, Norman, and Pfeffer, Avi. 2002. Stochastic lambda calculus and monads of probability distributions. Pages 154–165 of: *Proc. of POPL 2002*. ACM.

Saheb-Djaromi, N. 1978. Probabilistic LCF. Pages 442–451 of: *Proc. of MFCS 1978*.

Santos, Eugene S. 1969. Probabilistic Turing machines and computability. *Proceedings of the American Mathematical Society*, **22**(3), 704–710.

Sørensen, Morten Heine, and Urzyczyn, Pawel. 2006. *Lectures on the Curry–Howard Isomorphism,*. New York, NY, USA: Elsevier Science Inc.

Statman, Richard. 1979. The typed lambda-calculus is not elementary recursive. *Theor. Comput. Sci.*, **9**, 73–81.

Staton, Sam. 2017. Commutative semantics for probabilistic programming. Pages 855–879 of: *Proc. of ESOP 2017*.

Tolpin, David, van de Meent, Jan-Willem, and Wood, Frank D. 2015. Probabilistic programming in Anglican. Pages 308–311 of: *Proc. of ECML PKDD 2015, Part III*.

Vákár, Matthijs, Kammar, Ohad, and Staton, Sam. 2019. A domain theory for statistical probabilistic programming. Pages 36:1–36:29 of: *Proc. of POPL 2019*.

Wand, Mitchell, Culpepper, Ryan, Giannakopoulos, Theophilos, and Cobb, Andrew. 2018. Contextual equivalence for a probabilistic language with continuous random variables and recursion, arXiv https://arxiv.org/abs/1807.02809.