On Probabilistic Applicative Bisimulation and Call-by-Value λ-Calculi

Raphaëlle Crubillé, Ugo Dal Lago

On Probabilistic Applicative Bisimulation and Call-by-Value λ-Calculi*

Raphaëlle Crubillé[1] and Ugo Dal Lago[2]

[1] ENS-Lyon, raphaelle.crubille@ens-lyon.fr
[2] Università di Bologna & INRIA, dallago@cs.unibo.it

Abstract. Probabilistic applicative bisimulation is a recently introduced coinductive methodology for program equivalence in a probabilistic, higher-order, setting. In this paper, the technique is applied to a typed, call-by-value, lambda-calculus. Surprisingly, the obtained relation coincides with context equivalence, contrary to what happens when call-by-name evaluation is considered. Even more surprisingly, full-abstraction only holds in a symmetric setting.

Keywords: lambda calculus, probabilistic computation, bisimulation, coinduction

1 Introduction

Traditionally, an algorithm is nothing but a finite description of a sequence of deterministic primitive instructions, which solve a computational problem when executed. Along the years, however, this concept has been generalized so as to reflect a broader class of effective procedures and machines. One of the many ways this has been done consists in allowing probabilistic choice as a primitive instruction in algorithms, this way shifting from usual, deterministic computation to a new paradigm, called probabilistic computation. Examples of application areas in which probabilistic computation has proved to be useful include natural language processing [19], robotics [28], computer vision [3], and machine learning [22]. Sometimes, being able to "flip a fair coin" while computing is a *necessity* rather than an alternative, like in computational cryptography (where, e.g., secure public key encryption schemes are bound to be probabilistic [10]).

Any (probabilistic) algorithm can be executed by concrete machines only once it takes the form of a *program*. And indeed, various probabilistic programming languages have been introduced in the last years, from abstract ones [15, 26, 21] to more concrete ones [23, 11]. A quite common scheme consists in endowing any deterministic language with one or more primitives for probabilistic choice, like binary probabilistic choice or primitives for distributions.

Viewing algorithms as functions allows a smooth integration of distributions into the playground, itself nicely reflected at the level of types through monads [12, 26]. As a matter of fact, some existing probabilistic programming languages [23, 11] are designed around the λ-calculus or one of its incarnations, like Scheme. This, in turn has stimulated foundational research about probabilistic λ-calculi, and in particular about

* The authors are partially supported by the ANR project 12IS02001 PACE.

the nature of program equivalence in a probabilistic setting. This has already started to produce some interesting results in the realm of denotational semantics, where adequacy and full-abstraction results have recently appeared [7, 9].

Not much is known about operational techniques for probabilistic program equivalence, and in particular about coinductive methodologies. This is in contrast with what happens for deterministic or nondeterministic programs, when various notions of bisimulation have been introduced and proved to be adequate and, in some cases, fully abstract [1, 20, 18]. A recent paper by Alberti, Sangiorgi and the second author [5] generalizes Abramsky's applicative bisimulation [1] to $\Lambda_{\oplus}$, a call-by-name, untyped λ-calculus endowed with binary, fair, probabilistic choice [6]. Probabilistic applicative bisimulation is shown to be a congruence, thus included in context equivalence. Completeness, however, fails, the counterexample being exactly the one separating bisimulation and context equivalence in a nondeterministic setting. Full abstraction is then recovered when pure, deterministic λ-terms are considered, as well as when another, more involved, notion of bisimulation, called coupled logical bisimulation, takes the place of applicative bisimulation.

In this paper, we proceed with the study of probabilistic applicative bisimulation, analysing its behaviour when instantiated on call-by-value λ-calculi. This investigation brings up some nice, unexpected results. Indeed, not only the non-trivial proof of congruence for applicative bisimulation can be adapted to the call-by-value setting, which is somehow expected, but applicative bisimilarity turns out to precisely characterize context equivalence. This is quite surprising, given that in nondeterministic λ-calculi, both when call-by-name and call-by-value evaluation are considered, applicative bisimilarity is a congruence, but *finer* than context equivalence [18]. There is another, even less expected result: the aforementioned correspondence does not hold anymore if we consider applicative *simulation* and the contextual *preorder*.

Technically, the presented results owe much to a recent series of studies about probabilistic bisimulation for labelled Markov processes [8, 29], i.e., labelled probabilistic transition systems in which the state space is continuous (rather than discrete, as in Larsen and Skou's labelled Markov chains [17]), but time stays discrete. More specifically, the way we prove that context equivalent terms are bisimilar goes by constructively showing how each *test* of a kind characterizing probabilistic bisimulation can be turned into an equivalent *context*. If, as a consequence, two terms are not bisimilar, then any test the two terms satisfy with different probabilities (of which there must be at least one) becomes a context in which the two terms converge with different probabilities. This helps understanding the discrepancies between the probabilistic and nondeterministic settings, since in the latter the class of tests characterizing applicative bisimulation is well-known to be quite large [20]. The mismatch between the symmetric and asymmetric cases is also clarified — again, the language of tests characterizing similarity is strictly more general than the one characterizing bisimilarity [29].

The whole development is done in a probabilistic variation on PCF with lazy lists, called $\mathsf{PCFL}_{\oplus}$. Working with an applied calculus allows to stay closer to concrete programming languages, this way facilitating exemplification, as in Section 2 below. Infinitary data structures are there to show that probabilistic applicative bisimulation works well in a setting where coinduction plays a key rôle.

2 Some Motivating Examples

In this section, we want to show how λ-calculus can naturally express probabilistic programs. More importantly, we will argue that checking the equivalence of some of the presented programs is not only interesting from a purely theoretical perspective, but corresponds to a proof of *perfect security* in the sense of Shannon [27].

Let's start from the following very simple programs:

$$NOT = \lambda x.\mathsf{if}\ x\ \mathsf{then}\ \underline{\mathsf{false}}\ \mathsf{else}\ \underline{\mathsf{true}}\ : \mathbf{bool} \to \mathbf{bool};$$
$$ENC = \lambda x.\lambda y.\mathsf{if}\ x\ \mathsf{then}\ (NOT\ y)\ \mathsf{else}\ y : \mathbf{bool} \to \mathbf{bool} \to \mathbf{bool};$$
$$GEN = \underline{\mathsf{true}} \oplus \underline{\mathsf{false}} : \mathbf{bool}.$$

The function ENC computes exclusive disjunction as a boolean function, but can also be seen as the encryption function of a one-bit version of the so-called One-Time Pad cryptoscheme (OTP in the following). On the other hand, GEN is a term reducing probabilistically to one of the two possible boolean values, each with probability $\frac{1}{2}$, and is meant to be a way to generate a random key for the same scheme.

One of the many ways to define perfect security of an encryption scheme consists is setting up an *experiment* [16]: the adversary generates two messages, of which one is randomly chosen, encrypted, and given back to the adversary who, however, should not be able to guess whether the first or the second message have been chosen (with success probability strictly greater than $\frac{1}{2}$). This can be seen as the problem of proving the following two programs to be context equivalent:

$$EXP = \lambda x.\lambda y.ENC\ (x \oplus y)\ GEN : \mathbf{bool} \to \mathbf{bool} \to \mathbf{bool};$$
$$RND = \lambda x.\lambda y.\underline{\mathsf{true}} \oplus \underline{\mathsf{false}} : \mathbf{bool} \to \mathbf{bool} \to \mathbf{bool};$$

where $\oplus$ is a primitive for fair, probabilistic choice. Analogously, one could verify that any adversary is not able to distinguish an experiment in which the *first* message is chosen from an experiment in which the *second* message is chosen. This, again, can be seen as the task of checking whether the following two terms are context equivalent:

$$EXP_{FST} = \lambda x.\lambda y.ENC\ x\ GEN : \mathbf{bool} \to \mathbf{bool} \to \mathbf{bool};$$
$$EXP_{SND} = \lambda x.\lambda y.ENC\ y\ GEN : \mathbf{bool} \to \mathbf{bool} \to \mathbf{bool}.$$

But how could we actually *prove* two programs to be context equivalent? The universal quantification in its definition, as is well known, turns out to be burdensome in proofs. The task can be made easier by way of various techniques, including context lemmas and logical relations. Later in this paper, we show how the four terms above can be proved equivalent by way of applicative bisimulation, which is proved sound (and complete) with respect to context equivalence in Section 4 below.

Before proceeding, we would like to give examples of terms having the same type, but which are *not* context equivalent. We will do so by again referring to perfect security. The kind of security offered by the OTP is unsatisfactory not only because keys cannot be shorter than messages, but also because it does not hold in presence of multiple encryptions, or when the adversary is *active*, for example by having an access to

an encryption oracle. In the aforementioned scenario, security holds if and only if the following two programs (both of type $\mathbf{bool} \to \mathbf{bool} \to \mathbf{bool} \times (\mathbf{bool} \to \mathbf{bool})$) are context equivalent:

$$EXP^{CPA}_{FST} = \lambda x.\lambda y.(\lambda z.\langle ENC\ x\ z, \lambda w.ENC\ w\ z\rangle)\, GEN;$$
$$EXP^{CPA}_{SND} = \lambda x.\lambda y.(\lambda z.\langle ENC\ y\ z, \lambda w.ENC\ w\ z\rangle)\, GEN.$$

It is very easy, however, to realize that if $C = (\lambda x.(\mathsf{snd}\,(x))(\mathsf{fst}\,(x)))([\cdot]\ \underline{\mathsf{true}}\ \underline{\mathsf{false}})$, then $C[EXP^{CPA}_{FST}]$ reduces to $\underline{\mathsf{true}}$, while $C[EXP^{CPA}_{SND}]$ reduces to $\underline{\mathsf{false}}$, both with probability 1. In other words, the OTP is not secure in presence of active adversaries, and for very good reasons: having access to an oracle for encryption is essentially equivalent to having access to an oracle for *decryption*.

3 Programs and Their Operational Semantics

In this section, we will present the syntax and operational semantics of $\mathsf{PCFL}_\oplus$, the language on which we will define applicative bisimulation. Due to lack of space, we cannot give all the details, which are anyway available in [4]. Moreover, $\mathsf{PCFL}_\oplus$ is identical to Pitts' PCFL [24], except for the presence of a primitive for binary probabilistic choice.

3.1 Terms and Types

The terms of $\mathsf{PCFL}_\oplus$ are built up from constants (for boolean and integer values, and for the empty list) and variables, using the usual constructs from PCF, and binary choice. In the following, $\mathcal{X} = \{x, y, \ldots\}$ is a countable set of variables and $\mathcal{O}$ is a finite set of binary arithmetic operators including at least the symbols $+$, $\leq$, and $=$.

Definition 1. Terms *are expressions generated by the following grammar:*

$$\begin{aligned} M, N\ ::= x \ \Big|\ \underline{n} \ \Big|\ \underline{b} \ \Big|\ \mathsf{nil} \ \Big|\ \langle M, M\rangle \ \Big|\ M :: M \ \Big|\ \lambda x.M \ \Big|\ \mathsf{fix}\, x.\, M \\ \Big|\ M \oplus M \ \Big|\ \mathsf{if}\ M\ \mathsf{then}\ M\ \mathsf{else}\ M \ \Big|\ M\ \mathsf{op}\ M \ \Big|\ \mathsf{fst}\,(M) \ \Big|\ \mathsf{snd}\,(M) \\ \Big|\ \mathsf{case}\ M\ \mathsf{of}\ \{\mathsf{nil} \to M \mid h :: t \to M\} \ \Big|\ M\, M, \end{aligned}$$

where $x, h, t \in \mathcal{X}$, $n \in \mathbb{N}$, $b \in \mathbb{B} = \{\,\mathsf{true}\,, \mathsf{false}\,\}$, *and* $\mathsf{op} \in \mathcal{O}$.

In what follows, we consider terms of $\mathsf{PCFL}_\oplus$ as α-equivalence classes of syntax trees. The set of free variables of a term M is indicated as $FV(M)$. A term M is closed if $FV(M) = \emptyset$. The (capture-avoiding) substitution of N for the free occurrences of x in M is denoted $M[N/x]$.

The constructions from PCF have their usual meanings. The operator $(\cdot :: \cdot)$ is the constructor for lists, nil is the empty list, and $\mathsf{case}\, L\, \mathsf{of}\, \{\mathsf{nil} \to M \mid h :: t \to N\}$ is a list destructor. The construct $M \oplus N$ is a binary choice operator, to be interpreted probabilistically, as in $\Lambda_\oplus$ [6].

Example 1. Relevant examples of terms are $\Omega = (\mathsf{fix}\, x.\, x)\, \underline{0}$, and $I = \lambda x.x$: the first one always diverges, while the second always converges (to itself). In between, one can find terms that converge with probability between 0 and 1, excluded, e.g., $I \oplus \Omega$, and $I \oplus (I \oplus \Omega)$.

We are only interested in well-formed terms, i.e., terms to which one can assign a type.

Definition 2. Types *are given by the following grammar:*

$$\sigma, \tau \ ::= \ \gamma \ \Big| \ \sigma \to \sigma \ \Big| \ \sigma \times \sigma \ \Big| \ [\sigma]; \qquad\qquad \gamma, \delta \ ::= \ \mathbf{bool} \ \Big| \ \mathbf{int}.$$

The set of all types is $\mathcal{Y}$. Please observe that the language of types we consider here coincides with the one of Pitts' PCFL [24]. An alternative typing discipline for probabilistic languages (see, e.g. [26]), views probability as a *monad*, this way reflecting the behaviour of programs in types: if σ is a type, $\Box\sigma$ is the type of probabilistic distributions over σ, and the binary choice operator always produces elements of type $\Box\sigma$.

We assume that all operators from $\mathcal{O}$ take natural numbers as input, and we associate to each operator $\mathsf{op} \in \mathcal{O}$ its *result type* $\gamma_{\mathsf{op}} \in \{\mathbf{bool}, \mathbf{int}\}$ and its semantics $\overline{\mathsf{op}} : \mathbb{N} \times \mathbb{N} \to X$ where X is either $\mathbb{B}$ or $\mathbb{N}$, depending on γ_{op}. A *typing context* Γ is a finite partial function from variables to types. $dom(\Gamma)$ is the domain of the function Γ. If $x \notin dom(\Gamma)$, $(x : \sigma, \Gamma)$ represents the function which extends Γ to $dom(\Gamma) \cup \{x\}$, by associating σ to x.

Definition 3. *A* typing judgement *is an assertion of the form $\Gamma \vdash M : \sigma$, where Γ is a context, M is a term, and σ is a type. Typing rules are standard, and the most interesting ones are in Figure 1.*

$$\frac{\Gamma \vdash M : \mathbf{int} \qquad \Gamma \vdash M : \mathbf{int}}{\Gamma \vdash M\, \mathsf{op}\, M : \gamma_{\mathsf{op}}} \qquad \frac{\Gamma \vdash M : \sigma \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash M \oplus N : \sigma}$$

$$\frac{\Gamma, x : \sigma \to \tau \vdash M : \sigma \to \tau \qquad x \notin dom(\Gamma)}{\Gamma \vdash \mathsf{fix}\, x.\, M : \sigma \to \tau} \qquad \frac{\Gamma \vdash T : [\sigma] \qquad \Gamma \vdash H : \sigma}{\Gamma \vdash H :: T : [\sigma]}$$

$$\frac{\Gamma \vdash L : [\sigma] \qquad \Gamma \vdash M : \tau \qquad \Gamma, h : \sigma, t : [\sigma] \vdash N : \tau}{\Gamma \vdash \mathsf{case}\, L\, \mathsf{of}\, \{\mathsf{nil} \to M \mid h :: t \to N\} : \tau}$$

Fig. 1. Type Assignment in $\mathsf{PCFL}_{\oplus}$ — Rule Selection

Please notice that any term of which we want to form the fixpoint needs to be a function. If σ is a type and Γ is a typing context, then $\mathcal{T}^{\sigma} = \{t \,|\, \emptyset \vdash t : \sigma\}$, $\mathcal{T} = \{t \,|\, \exists \sigma, t \in \mathcal{T}^{\sigma}\}$, $\mathcal{T}^{\sigma}_{\Gamma} = \{t \,|\, \Gamma \vdash t : \sigma\}$. Terms in $\mathcal{T}^{\sigma}$ are said to be the *closed terms* (also called *programs*) of type σ.

3.2 Operational Semantics

Because of the probabilistic nature of choice in $\mathsf{PCFL}_{\oplus}$, a program does not evaluate to a value, but to a probability distribution of values. Therefore, we need the following notions to define an evaluation relation.

Definition 4. Values *are terms of the following form:*

$$V \ ::= \ \underline{n} \ \Big| \ \underline{b} \ \Big| \ \mathsf{nil} \ \Big| \ \lambda x.M \ \Big| \ \mathsf{fix}\, x.\, M \ \Big| \ M :: M \ \Big| \ \langle M, M \rangle.$$

We will call $\mathcal{V}$ the set of values, and we note $\mathcal{V}^\sigma = \mathcal{V} \cap \mathcal{T}^\sigma$. A value distribution *is a function $\mathscr{D} : \mathcal{V} \to [0,1]$, such that $\sum_{V \in \mathcal{V}} \mathscr{D}(V) \leq 1$. Given a value distribution $\mathscr{D}$, we will note $\mathsf{S}(\mathscr{D})$ the set of those values V such that $\mathscr{D}(V) > 0$. A value distribution $\mathscr{D}$ is said* finite *whenever $\mathsf{S}(\mathscr{D})$ has finite cardinality. If V is a value, we note $\{V^1\}$ the value distribution $\mathscr{D}$ such that $\mathscr{D}(W) = 1$ if $W = V$ and $\mathscr{D}(V) = 0$ otherwise. Value distributions can be ordered pointwise.*

We first give an *approximation* semantics, which attributes *finite* probability distributions to terms, and only later define the actual semantics, which will be the least upper bound of all distributions obtained through the approximation semantics. Big-step semantics is given by way of a binary relation $\Downarrow$ between closed terms and value distributions, which is defined by some rules, of which we only give the most interesting ones in Figure 2. This evaluation relation, by the way, is the natural extension to $\mathsf{PCFL}_{\oplus}$

$$\frac{}{M \Downarrow \emptyset} \qquad \frac{}{V \Downarrow \{V^1\}} \qquad \frac{M \Downarrow \mathscr{D} \qquad N \Downarrow \mathscr{E}}{M\ \mathsf{op}\ N \Downarrow \sum_{\underline{n} \in \mathsf{S}(\mathscr{D}), \underline{m} \in \mathsf{S}(\mathscr{E})} \mathscr{D}(\underline{n})\mathscr{E}(\underline{m})\{\underline{\mathsf{op}(m,n)}^1\}}$$

$$\frac{\begin{array}{c} M \Downarrow \mathscr{K} \qquad N \Downarrow \mathscr{F} \qquad \{P[V/x] \Downarrow \mathscr{E}_{P,V}\}_{\lambda x.P \in \mathsf{S}(\mathscr{K}),\, V \in \mathsf{S}(\mathscr{F})} \\ \{Q[\mathsf{fix}\, x.\, Q/x]V \Downarrow \mathscr{G}_{Q,V}\}_{\mathsf{fix}\, x.\, Q \in \mathsf{S}(\mathscr{K}),\, V \in \mathsf{S}(\mathscr{F})} \end{array}}{MN \Downarrow \sum_{V \in \mathsf{S}(\mathscr{F})} \mathscr{F}(V) \left(\sum_{\lambda x.P \in \mathsf{S}(\mathscr{K})} \mathscr{K}(\lambda x.P)\mathscr{E}_{P,V} + \sum_{\mathsf{fix}\, x.\, Q \in \mathsf{S}(\mathscr{K})} \mathscr{K}(\mathsf{fix}\, x.\, Q)\mathscr{G}_{Q,V} \right)}$$

$$\frac{M \Downarrow \mathscr{D} \qquad N \Downarrow \mathscr{E} \qquad L \Downarrow \mathscr{F}}{\mathsf{if}\ M\ \mathsf{then}\ N\ \mathsf{else}\ L \Downarrow \mathscr{D}(\underline{\mathsf{true}})\mathscr{E} + \mathscr{D}(\underline{\mathsf{false}})\mathscr{F}} \qquad \frac{M \Downarrow \mathscr{D} \qquad N \Downarrow \mathscr{E}}{M \oplus N \Downarrow \frac{1}{2}\mathscr{D} + \frac{1}{2}\mathscr{E}}$$

Fig. 2. Evaluation — Rule Selection

of the evaluation relation given in [6] for the untyped probabilistic λ-calculus. Please observe how function arguments are evaluated before being passed to functions. Moreover, $M :: N$ is a value even if M or N are not, which means that lists are *lazy* and potentially infinite.

Proposition 1. *Call-by-value evaluation preserves typing, that is: if $M \Downarrow \mathscr{D}$, and $M \in \mathcal{T}^\sigma$, then for every $V \in \mathsf{S}(\mathscr{D})$, $V \in \mathcal{V}^\sigma$.*

Lemma 1. *For every term M, if $M \Downarrow \mathscr{D}$, and $M \Downarrow \mathscr{E}$, then there exists a distribution $\mathscr{F}$ such that $M \Downarrow \mathscr{F}$ with $\mathscr{D} \leq \mathscr{F}$, and $\mathscr{E} \leq \mathscr{F}$.*

Proof. The proof is by induction on the structure of derivations for $M \Downarrow \mathscr{D}$.

Definition 5. *For any closed term M, we define the* big-steps semantics $[\![M]\!]$ *of M as* $\sup_{M \Downarrow \mathscr{D}} \mathscr{D}$.

Since distributions form an ω-complete partial order, and for every M the set of those distributions $\mathscr{D}$ such that $M \Downarrow \mathscr{D}$ is a countable directed set, this definition is well-posed, and associates a unique value distribution to every term.

The distribution $[\![M]\!]$ can be obtained equivalently by taking the least upper bound of all finite distributions $\mathscr{D}$ for which $M \Rightarrow \mathscr{D}$, where $\Rightarrow$ is a binary relation capturing *small-step* evaluation of terms. More about it can be found in [4].

Example 2. Approximation semantics does not allow to derive any assertion about Ω, and indeed $[\![\Omega]\!] = \emptyset$. Similarly, $[\![I]\!] = \{I^1\}$. Recursion allows to define much more interesting programs, e.g. $M = (\mathsf{fix}\, x.\, (\lambda y.y) \oplus \lambda y.x(y + \underline{1}))\, \underline{0}$. Indeed, $[\![M]\!](\underline{n}) = \frac{1}{2^{n+1}}$ for every $n \in \mathbb{N}$, even if $M \not\Downarrow [\![M]\!]$.

3.3 Relations

A *typed relation* is a family $\mathcal{R} = (\mathcal{R}_\sigma^\Gamma)_{\sigma,\Gamma}$, where each $\mathcal{R}_\sigma^\Gamma$ is a binary relation on $\mathcal{T}_\sigma^\Gamma$. Sometime, $M\ \mathcal{R}_\sigma^\Gamma\ N$ will be noted as $\Gamma \vdash M\ \mathcal{R}_\sigma\ N$ (or as $\Gamma \vdash M\ \mathcal{R}\ N : \sigma$). The notions of symmetry, reflexivity, transitivity and compatibility can all be extended to typed relations in the natural way. Since being compatible can be seen as being reflexive on ground types and stable by the constructors of the language, the following is easy to prove:

Proposition 2. *Let $\mathcal{R}$ be a typed relation. If $\mathcal{R}$ is compatible, then $\mathcal{R}$ is reflexive.*

Any typed relation capturing a notion of equivalence should be a congruence, this way being applicable at any point in the program, possibly many times:

Definition 6. *Let $\mathcal{R}$ be a typed relation. Then $\mathcal{R}$ is said to be a* precongruence relation *if $\mathcal{R}$ is transitive and compatible, and $\mathcal{R}$ is said to be a* congruence relation *if $\mathcal{R}$ is symmetric, transitive and compatible.*

We write $\mathscr{R}$ for the set of type-indexed families $\mathcal{R} = (\mathcal{R}_\sigma)_\sigma$ of binary relations $\mathcal{R}_\sigma$ between the terms in $\mathcal{T}^\sigma$.

3.4 Context Equivalence

The general idea of context equivalence is the following: two terms M and N are equivalent if any occurrence of M in any program L can be replaced with N without changing the observable behaviour of L. The notion of a context allows us to formalize this idea.

Definition 7. *A* context *is a term containing a unique hole $[\cdot]$. Given a context C and a term M, $C[M]$ is the term obtained by substituting the unique hole in C with M.*

When defining context equivalence, we work with *closing* contexts, namely those contexts C such that $C[M]$, and $C[N]$ are closed terms (where M and N are the possibly open terms being compared). In the following, we will use judgements in the form $\Gamma \vdash C(\Delta;\sigma) : \tau$, which informally means that if M is a term of type σ under the typing context Δ, then the hole of C can be filled by M, obtaining a term of type τ under the typing context Γ. Correct assertions of this form can be derived by a formal system, which we cannot present for lack of space, but which can be anyway found in [4].

Example 3. Example of derivable judgments of the just described form are $\emptyset \vdash \lambda x.[\cdot]\,(x : \sigma;\tau) : (\sigma \to \tau)$ and $\emptyset \vdash ((\lambda x.\underline{\mathsf{true}})\,[\cdot])\;(\emptyset;\sigma) : \mathbf{bool}$.

Here, following [7, 5], we consider that the observable behaviour of a program M is its *probability of convergence* $\sum[\![M]\!] = \sum_V [\![M]\!](V)$. We now have all the ingredients necessary to define what context equivalence is:

Definition 8. *The* contextual preorder *is the typed relation $\leq$ given by: for every $M, N \in \mathcal{T}_\tau^\Gamma$, $\Gamma \vdash M \leq N : \tau$ if for every context C such that $\emptyset \vdash C\,(\Gamma;\tau) : \sigma$, it holds that $\sum[\![C[M]]\!] \leq \sum[\![C[N]]\!]$.* Context equivalence *is the typed relation $\equiv$ given by stipulating that $\Gamma \vdash M \equiv N : \sigma$ iff $\Gamma \vdash M \leq N : \sigma$ and $\Gamma \vdash N \leq M : \sigma$.*

Another way to define context equivalence would be to restrain ourselves to contexts of $\mathbf{bool}$ and $\mathbf{int}$ type in the definition of context equivalence: this is the so-called *ground* context equivalence. In a call-by-value setting, however, this gives exactly the same relation, since any non-ground context can be turned into a ground context inducing the same probability of convergence. A similar argument holds for a notion of equivalence in which one observes the obtained (ground) *distribution* rather than merely its sum. The following can be proved in a standard way:

Proposition 3. *$\leq$ is a typed relation, which is reflexive, transitive and compatible.*

Because of the quantification over all contexts, it is usually difficult to show that M and N are two context equivalent terms. In the next sections, we will introduce another notion of equivalence, and we show that it is included in context equivalence.

4 Applicative Bisimulation

In this section, we introduce the notions of similarity and bisimilarity for $\mathsf{PCFL}_\oplus$. We proceed by instantiating probabilistic bisimulation as developed by Larsen and Skou for a generic labelled Markov chain in [17]. A similar use was done for a call-by-name untyped probabilistic λ-calculus $\Lambda_\oplus$ in [5].

4.1 Larsen and Skou's Probabilistic Bisimulation

Preliminary to the notion of (bi)simulation, is the notion of a *labelled Markov chain* (LMC in the following), which is a triple $\mathcal{M} = (\mathcal{S}, \mathcal{L}, \mathcal{P})$, where $\mathcal{S}$ is a countable set of *states*, $\mathcal{L}$ is a set of *labels*, and $\mathcal{P}$ is a *transition probability matrix*, i.e., a function $\mathcal{P} : \mathcal{S} \times \mathcal{L} \times \mathcal{S} \to \mathbb{R}$ such that for every state $s \in \mathcal{S}$ and for every label $l \in \mathcal{L}$, $\sum_{t \in \mathcal{S}} \mathcal{P}(s,l,t) \leq 1$. Following [8], we allow the sum above to be smaller than 1, modelling divergence this way. The following is due to Larsen and Skou [17]:

Definition 9. *Given $(\mathcal{S}, \mathcal{L}, \mathcal{P})$ a labelled Markov chain, a* probabilistic simulation *is a preorder relation R on $\mathcal{S}$ such that $(s,t) \in R$ implies that for every $X \subseteq \mathcal{S}$ and for every $l \in \mathcal{L}$, $\mathcal{P}(s,l,X) \leq \mathcal{P}(t,l,R(X))$, with $R(X) = \{y \mid \exists x \in X \text{ such that } x\ R\ y\}$. Similarly, a* probabilistic bisimulation *is an equivalence relation R on $\mathcal{S}$ such that $(s,t) \in R$ implies that for every equivalence class E modulo R, and for every $l \in \mathcal{L}$, $\mathcal{P}(s,l,E) = \mathcal{P}(t,l,E)$.*

Insisting on bisimulations to be equivalence relations has the potential effect of not allowing them to be formed by just taking unions of other bisimulations. The same can be said about simulations, which are assumed to be partial orders. Nevertheless:

Proposition 4. *If $(R_i)_{i \in I}$ is a collection of probabilistic (bi)simulations, then the reflexive and transitive closure of their union, $(\cup_{i \in I} R_i)^*$, is a (bi)simulation.*

A nice consequence of the result above is that we can define *probabilistic similarity* (noted $\precsim$) simply as the relation $\precsim = \bigcup\{R \mid R \text{ is a probabilistic simulation}\}$. Analogously for the largest probabilistic bisimulation, that we call *probabilistic bisimilarity* (noted $\backsim$), defined as $\backsim = \bigcup\{R \mid R \text{ is a probabilistic bisimulation}\}$. A property of probabilistic bisimulation which does not hold in the usual, nondeterministic, setting, is the following:

Proposition 5. $\backsim = \precsim \cap \precsim^{op}$.

4.2 A Concrete Labelled Markov Chain

Applicative bisimulation will be defined by instantiating Definition 9 on a specific LMC, namely the one modelling evaluation of $\mathsf{PCFL}_\oplus$ programs.

Definition 10. *The labelled Markov chain $\mathcal{M}_\oplus = (\mathcal{S}_\oplus, \mathcal{L}_\oplus, \mathcal{P}_\oplus)$ is given by:*

- *A set of states $\mathcal{S}_\oplus$ defined as follows:*

$$\mathcal{S}_\oplus = \{(M,\sigma) \mid M \in \mathcal{T}^\sigma\} \uplus \{(\hat{V},\sigma) \mid V \in \mathcal{V}^\sigma\},$$

  where terms and values are taken modulo α-equivalence. A value V in the second component of $\mathcal{S}_\oplus$ is distinguished from one in the first by using the notation $\hat{V}$.
- *A set of labels $\mathcal{L}_\oplus$ defined as follows:*

$$\mathcal{V} \uplus \mathcal{Y} \uplus \mathbb{N} \uplus \mathbb{B} \uplus \{nil, hd, tl\} \uplus \{fst, snd\} \uplus \{eval\},$$

  where, again, terms are taken modulo α-equivalence, and $\mathcal{Y}$ is the set of types.
- *A transition probability matrix $\mathcal{P}_\oplus$ such that:*
  - *For every $M \in \mathcal{T}^\sigma$, $\mathcal{P}_\oplus\left((M,\sigma),\sigma,(M,\sigma)\right) = 1$, and similarly for values.*
  - *For every $M \in \mathcal{T}^\sigma$, and any value $V \in \mathsf{S}([\![M]\!])$, $\mathcal{P}_\oplus\left((M,\sigma), eval, (\hat{V},\sigma)\right) = [\![M]\!](V)$.*
  - *If $V \in \mathcal{V}^\sigma$ then certain actions from $\mathcal{L}_\oplus$ are enabled and produce the natural outcomes depending on the shape of σ. As a an example, if $\sigma = \tau \to \theta$, and $V = \lambda x.M$, then for each $W \in \mathcal{V}^\tau$, $\mathcal{P}_\oplus\left((\hat{V}, \tau \to \theta), W, (M[W/x], \theta)\right) = 1$. As another example, if $\sigma = \mathbf{int}$, then there is $k \in \mathbb{N}$ such that $V = \underline{k}$ and $\mathcal{P}_\oplus\left((\hat{V}, \mathbf{int}), k, (\hat{V}, \mathbf{int})\right) = 1$. The other cases are similar, and more details are in [4].*

For all s, l, t such that $\mathcal{P}_\oplus(s,l,t)$ is not defined above, we have $\mathcal{P}_\oplus(s,l,t) = 0$.

Please observe that if $V \in \mathcal{V}^\sigma$, both (V,σ) and $(\hat{V},\sigma)$ are states of the Markov chain $\mathcal{M}_\oplus$. A similar Markov chain was used in [5] to define bisimilarity for the untyped probabilistic λ-calculus $\Lambda_\oplus$. We use here in the same way actions which apply a term to a value, and an action which models term evaluation, namely $eval$.

4.3 The Definition

We would like to see any simulation (or bisimulation) on the LMC $\mathcal{M}_\oplus$ as a family in $\mathscr{R}$. As can be easily realized, indeed, any (bi)simulation on $\mathcal{M}_\oplus$ cannot put in correspondence states (M,σ) and (N,τ) where $\sigma \neq \tau$, since each such pair exposes its second component as an action. Moreover, $(\hat{V},\sigma)$ is (bi)similar to $(\hat{W},\sigma)$ iff (V,σ) is (bi)similar to (W,σ). This then justifies the following:

Definition 11. *A* probabilistic applicative simulation *(a PAS in the following), is a family $(\mathcal{R}_\sigma) \in \mathscr{R}$ such that there exists a probabilistic simulation R on the LMC $\mathcal{M}_\oplus$ such that for every type σ, and for every $M, N \in \mathcal{T}^\sigma$ it holds that $M\ \mathcal{R}_\sigma\ N \Leftrightarrow (M,\sigma)\ R\ (N,\sigma)$. A* probabilistic applicative bisimulation *(PAB in the following) is defined similarly, requiring R to be a bisimulation rather than a simulation.*

The greatest simulation and the greatest bisimulation on $\mathcal{M}_\oplus$ are indicated with $\precsim$, and $\backsim$, respectively. In other words, $\precsim_\sigma$ is the relation $\{(M,N) \mid (M,\sigma) \precsim (N,\sigma)\}$, while $\backsim_\sigma$ is the relation $\{(M,N) \mid (M,\sigma) \backsim (N,\sigma)\}$. Terms having the same semantics need to be bisimilar:

Lemma 2. *Let $(\mathcal{R}_\sigma) \in \mathscr{R}$ be defined as follows: $M\ \mathcal{R}_\sigma\ N \Leftrightarrow M, N \in \mathcal{T}^\sigma \wedge [\![M]\!] = [\![N]\!]$. Then $(\mathcal{R}_\sigma)$ is a PAB.*

As a consequence, if $M, N \in \mathcal{T}^\sigma$ are such that $[\![M]\!] = [\![N]\!]$, then $M \backsim_\sigma N$.

Example 4. For all σ, M, N such that $\emptyset \vdash M, N : \sigma$ and $[\![N]\!] = \emptyset$, we have that $M \precsim_\sigma N$ implies $[\![M]\!] = \emptyset$. For every terms M, N such that $x : \tau \vdash M : \sigma$, and $\emptyset \vdash N : \tau$, we have, as a consequence of Lemma 2, that $(\lambda x.M)N \backsim_\sigma M[N/x]$.

We have just defined applicative (bi)simulation as a family $(\mathcal{R}_\sigma)_\sigma$, each $\mathcal{R}_\sigma$ being a relation on closed terms of type σ. We can extend it to a *typed* relation, by the usual open extension:

Definition 12. *1. If $\Gamma = x_1 : \tau_1, \ldots, x_n : \tau_n$ is a context, a* Γ-closure *makes each variable x_i to correspond to a value $V_i \in \mathcal{V}^{\tau_i}$ (where $1 \leq i \leq n$). The set of Γ-closures is CC_Γ. For every term $\Gamma \vdash M : \sigma$ and for every Γ-closure ξ, $M\xi$ is the term in $\mathcal{T}^\sigma$ obtained by substituting the variables in Γ with the corresponding values from ξ.*

2. Let be $\mathcal{R} = (\mathcal{R}_\sigma) \in \mathscr{R}$. We define the open extension *of $(\mathcal{R}_\sigma)$ as the typed relation $\mathcal{R}_\circ = (\mathcal{P}^\Gamma_\sigma)$ where $\mathcal{P}^\Gamma_\sigma \subseteq \mathcal{T}^\Gamma_\sigma \times \mathcal{T}^\Gamma_\sigma$ is defined by stipulating that $M\ \mathcal{P}^\Gamma_\sigma\ N$ iff for every $\xi \in CC_\Gamma$, $(M\xi)\ \mathcal{R}_\sigma\ (N\xi)$.*

Definition 13 (Simulation Preorder and Bisimulation Equivalence). *The typed relation $\precsim_\circ$ is said to be the* simulation preorder. *The typed relation $\backsim_\circ$ is said to be* bisimulation equivalence.

4.4 Bisimulation Equivalence is a Congruence

In this section, we want to show that $\backsim_\circ$ is actually a congruence, and that $\precsim_\circ$ is a precongruence. In view of Proposition 5, it is enough to show that the typed relation $\precsim_\circ$ is a precongruence, since $\backsim_\circ$ is the intersection of $\precsim_\circ$ and the opposite relation of $\precsim_\circ$. The key step consists in showing that $\precsim_\circ$ is compatible. This will be carried out by the Howe's Method, which is a general method for establishing such congruence properties [14].

The main idea of Howe's method consists in defining an auxiliary relation $\precsim_\circ^H$, such that it is easy to see that it is compatible, and then prove that $\precsim_\circ = (\precsim_\circ^H)^+$.

Definition 14. *Let $\mathcal{R}$ be a typed relation. The relation $\mathcal{R}^H$ is defined by a set of rules, of which we report a selection in Figure 3. The others can be found in [4], and are anyway identical to the analogous ones from [24].*

$$\frac{\Gamma, x:\sigma \vdash x\ \mathcal{R}\ M : \sigma}{\Gamma, x:\sigma \vdash x\ \mathcal{R}^H\ M : \sigma} \qquad \frac{\Gamma \vdash \underline{n}\ \mathcal{R}\ M : \mathbf{int}}{\Gamma \vdash \underline{n}\ \mathcal{R}^H\ M : \mathbf{int}}$$

$$\frac{\Gamma \vdash M\ \mathcal{R}^H\ N : \mathbf{int} \qquad \Gamma \vdash L\ \mathcal{R}^H\ P : \mathbf{int} \qquad \Gamma \vdash (N\ \mathsf{op}\ P)\ \mathcal{R}\ R : \gamma_{\mathsf{op}}}{\Gamma \vdash (M\ \mathsf{op}\ L)\ \mathcal{R}^H\ R : \gamma_{\mathsf{op}}}$$

$$\frac{\Gamma, x:\sigma \vdash M\ \mathcal{R}^H\ N : \tau \qquad \Gamma \vdash (\lambda x.N)\ \mathcal{R}\ L : \sigma \to \tau}{\Gamma \vdash (\lambda x.M)\ \mathcal{R}^H\ L : \sigma \to \tau}$$

$$\frac{\Gamma, x:\sigma \vdash M\ \mathcal{R}^H\ N : \sigma \qquad \Gamma \vdash (\mathsf{fix}\, x.\, N)\ \mathcal{R}\ L : \sigma}{\Gamma \vdash (\mathsf{fix}\, x.\, M)\ \mathcal{R}^H\ L : \sigma}$$

$$\frac{\Gamma \vdash M\ \mathcal{R}^H\ N : \sigma \to \tau \qquad \Gamma \vdash L\ \mathcal{R}^H\ P : \sigma \qquad \Gamma \vdash (NP)\ \mathcal{R}\ R : \tau}{\Gamma \vdash (ML)\ \mathcal{R}^H\ R : \tau}$$

Fig. 3. Howe's Construction — Rule Selection

We are now going to show, that if the relation $\mathcal{R}$ we start from satisfies minimal requirements, namely that it is reflexive and transitive, then the transitive closure $(\mathcal{R}^H)^+$ of the Howe's lifting is guaranteed to be a precongruence which contains $\mathcal{R}$. This is a direct consequence of the following results, whose proofs are standard inductions (see [4] for some more details):

- Let $\mathcal{R}$ be a reflexive typed relation. Then $\mathcal{R}^H$ is compatible.
- Let $\mathcal{R}$ be transitive. Then:

$$\big(\Gamma \vdash M\ \mathcal{R}^H\ N : \sigma\big) \wedge (\Gamma \vdash N\ \mathcal{R}\ L : \sigma) \Rightarrow \big(\Gamma \vdash M\ \mathcal{R}^H\ L : \sigma\big)$$

- If $\mathcal{R}$ is reflexive and $\Gamma \vdash M\ \mathcal{R}\ N : \sigma$, then $\Gamma \vdash M\ \mathcal{R}^H\ N : \sigma$.
- If $\mathcal{R}$ is compatible, then so is $\mathcal{R}^+$.

We can now apply the Howe's construction to $\precsim_\circ$, since it is clearly reflexive and transitive. The points above then tell us that $\precsim_\circ^H$, and $(\precsim_\circ^H)^+$ are both compatible. What we are left with, then, is proving that $(\precsim_\circ^H)^+$ is also a simulation. The following is a crucial step towards proving it:

Lemma 3 (Key Lemma). *For every terms M, N, the following hold:*

- *If $\emptyset \vdash M \ \precsim_\circ^H \ N : \sigma \to \tau$, then for every $X_1 \subseteq \mathcal{T}^\tau_{x:\sigma}$ and $X_2 \subseteq \mathcal{T}^{\sigma\to\tau}_{x:\sigma\to\tau}$, it holds that $[\![M]\!]\,(\lambda x.X_1 \bigcup \mathsf{fix}\, x.\, X_2) \leq [\![N]\!](\precsim_\circ (\lambda x.Y_1 \bigcup \mathsf{fix}\, x.\, Y_2))$, where $Y_1 = \{L \in \mathcal{T}^\tau_{x:\sigma} \mid \exists P \in X_1.x : \sigma \vdash P \precsim_\circ^H L : \tau\}$ and $Y_2 = \{L \in \mathcal{T}^{\sigma\to\tau}_{x:\sigma\to\tau} \mid \exists P \in X_2.x : \sigma \to \tau \vdash P \precsim_\circ^H L : \sigma \to \tau\}$.*
- *If $\emptyset \vdash M \ \precsim_\circ^H \ N : \sigma \times \tau$, then for every $X \subseteq \mathcal{V}^{\sigma\times\tau}$ we have: $[\![M]\!](X) \leq [\![N]\!](\precsim_\circ(Y))$, where $Y = \{\langle L, P\rangle \mid \exists \langle R, T\rangle \in X \wedge \emptyset \vdash R \precsim_\circ^H L : \sigma \wedge \emptyset \vdash T \precsim_\circ^H P : \tau\}$.*
- *If $\big(\emptyset \vdash M \ \precsim_\circ^H \ N : [\sigma]\big)$ then it holds that $[\![M]\!](\mathsf{nil}) \leq [\![N]\!](\mathsf{nil})$ and for every $X \subseteq \mathcal{V}^{[\sigma]}$, $[\![M]\!](X) \leq [\![N]\!](\precsim_\circ(Y))$ where Y is the set of those $K :: L$ such that there are H, T with $H :: T \in X$, $\emptyset \vdash H \precsim_\circ^H K : \sigma$, and $\emptyset \vdash T \ \precsim_\circ^H \ L : [\sigma]$.*
- *$\emptyset \vdash M \ \precsim_\circ^H \ N : \mathbf{int} \Rightarrow \forall k \in \mathbb{N}, [\![M]\!](\underline{k}) \leq [\![N]\!](\underline{k})$.*
- *$\emptyset \vdash M \ \precsim_\circ^H \ N : \mathbf{bool} \Rightarrow \forall b \in \mathbb{B}, [\![M]\!](\underline{b}) \leq [\![N]\!](\underline{b})$.*

The Key Lemma can be proved with tools very similar to the ones employed in [5] for an analogous result in an untyped call-by-name setting. Details can be found in [4]. A careful look at its statement reveals that, indeed, what it says is that $\precsim_\circ^H$ satisfies the axioms of a simulation when instantiated on the concrete LMC $\mathcal{M}_\oplus$.

A consequence of the Key Lemma, then, is that $(\precsim_\circ^H)^+$ is an applicative bisimulation, thus included in the largest one, namely $\precsim_\circ$. Since the latter is itself included in $\precsim_\circ^H$, we obtain that $\precsim_\circ = (\precsim_\circ^H)^+$. But $(\precsim_\circ^H)^+$ is a precongruence, and we get the main result of this section: $\precsim_\circ$ is a precongruence.

Theorem 1 (Soundness). *The typed relation $\precsim_\circ$ is a precongruence relation included in $\leq$. Analogously, $\backsim_\circ$ is a congruence relation included in $\equiv$.*

4.5 Back to Our Examples

We now have all the necessary tools to prove that the example programs from Section 2 are indeed context equivalent. As an example, let us consider again the following terms:

$$EXP_{FST} = \lambda x.\lambda y.ENC\ x\ GEN : \mathbf{bool} \to \mathbf{bool} \to \mathbf{bool};$$
$$EXP_{SND} = \lambda x.\lambda y.ENC\ y\ GEN : \mathbf{bool} \to \mathbf{bool} \to \mathbf{bool}.$$

One can define the relations $\mathcal{R}_{\mathbf{bool}}, \mathcal{R}_{\mathbf{bool}\to\mathbf{bool}}, \mathcal{R}_{\mathbf{bool}\to\mathbf{bool}\to\mathbf{bool}}$ by stipulating that $\mathcal{R}_\sigma = X_\sigma \times X_\sigma \cup ID_\sigma$ where

$$X_{\mathbf{bool}} = \{(ENC\ \underline{\mathsf{true}}\ GEN), (ENC\ \underline{\mathsf{false}}\ GEN)\};$$
$$X_{\mathbf{bool}\to\mathbf{bool}} = \{(\lambda y.ENC\ y\ GEN), (\lambda y.ENC\ \underline{\mathsf{true}}\ GEN), (\lambda y.ENC\ \underline{\mathsf{false}}\ GEN)\};$$
$$X_{\mathbf{bool}\to\mathbf{bool}\to\mathbf{bool}} = \{EXP_{FST}, EXP_{SND}\};$$

and for every type σ, ID_σ is the identity on $\mathcal{T}^\sigma$. When σ is not one of the types above, $\mathcal{R}_\sigma$ can be set to be just ID_σ. This way, the family $(\mathcal{R}_\sigma)$ can be seen as a relation R on the state space of $\mathcal{M}_\oplus$ (since any state in the form $(\hat{V}, \sigma)$ can be treated as (V, σ)). But R is easily seen to be a bisimulation. Indeed:

- All pairs of terms in $\mathcal{R}_{\mathbf{bool}}$ have the same semantics, since $[\![ENC\ \underline{\mathsf{true}}\ GEN]\!]$ and $[\![ENC\ \underline{\mathsf{false}}\ GEN]\!]$ are both the uniform distribution on the set of boolean values.
- The elements of $X_{\mathbf{bool}\to\mathbf{bool}}$ are values, and if we apply any two of them to a fixed boolean value, we end up with two terms $\mathcal{R}_{\mathbf{bool}}$ puts in relation.
- Similarly for $X_{\mathbf{bool}\to\mathbf{bool}\to\mathbf{bool}}$: applying any two elements of it to a boolean value yields two elements which are put in relations by $X_{\mathbf{bool}\to\mathbf{bool}}$.

Being an applicative bisimulation, $(\mathcal{R}_\sigma)_\sigma$ is included in $\sim$. And, by Theorem 1, we can conclude that $EXP_{FST} \equiv EXP_{SND}$. Analogously, one can verify that $EXP \equiv RND$.

5 Full Abstraction

Theorem 1 tells us that applicative bisimilarity is a sound way to prove that certain terms are context equivalent. Moreover, applicative bisimilarity is a congruence, and can then be applied in any context yielding bisimilar terms. In this section, we ask ourselves *how close* bisimilarity and context equivalence really are. Is it that the two coincide?

5.1 LMPs, Bisimulation, and Testing

The concept of probabilistic bisimulation has been generalized to the continuous case by Edalat, Desharnais and Panangaden, more than ten years ago [8]. Similarity and bisimilarity as defined in the aforementioned paper were later shown to exactly correspond to appropriate, and relatively simple, notions of *testing* [29]. We will make essential use of this characterization when proving that context equivalence is included in bisimulation. And this section is devoted to giving a brief but necessary introduction to the relevant theory. For more details, please refer to [29] and to [4].

In the rest of this section, $\mathscr{A}$ is a fixed set of labels. The first step consists in giving a generalization of LMCs in which the set of states is not restricted to be countable:

Definition 15. *A* labelled Markov process *(LMP in the following) is a triple $\mathcal{C} = (\mathcal{X}, \Sigma, \mu)$, consisting of a set $\mathcal{X}$ of states, a σ-field Σ on $\mathcal{X}$, and a transition probability function $\mu : \mathcal{X} \times \mathscr{A} \times \Sigma \to [0,1]$, such that:*

- *for all $x \in \mathcal{X}$, and $a \in Act$, the naturally defined function $\mu_{x,a}(\cdot) : \Sigma \to [0,1]$ is a subprobability measure;*
- *for all $a \in Act$, and $A \in \Sigma$, the naturally defined function $\mu_{(\cdot),a}(A) : \mathcal{X} \to [0,1]$ is measurable.*

The notion of (bi)simulation can be smoothly generalized to the continuous case:

Definition 16. *Let $(\mathcal{X}, \Sigma, \mu)$ be a LMP, and let R be a reflexive relation on $\mathcal{X}$. We say that R is a* simulation *if it satisfies Condition 1 below, and we say that R is a* bisimulation *if it satisfies both conditions 1 and 2:*

1. *If $x\ R\ y$, then for every $a \in \mathscr{A}$ and for every $A \in \Sigma$ such that $A = R(A)$, it holds that $\mu_{x,a}(A) \leq \mu_{y,a}(A)$.*
2. *If $x\ R\ y$, then for every $a \in \mathscr{A}$ and for every $A \in \Sigma$, $\mu_{x,a}(\mathcal{X}) = \mu_{y,a}(\mathcal{X})$.*

We say that two states are bisimilar *if they are related by some bisimulation.*

We will soon see that there is a natural way to turn any LMC into a LMP, in such a way that (bi)similarity stays the same. Before doing so, however, let us introduce the notion of a *test*:

Definition 17. *The* test language $\mathscr{T}$ *is given by the grammar* $t ::= \omega \mid a \cdot t \mid \langle t, t \rangle$, *where* $a \in \mathscr{A}$.

Please observe that tests are *finite* objects, and that there isn't any disjunctive nor any negative test in $\mathscr{T}$. Intuitively, ω is the test which always succeeds, while $\langle t, s \rangle$ corresponds to making two copies of the underlying state, testing them independently according to t and s and succeeding iff *both* tests succeed. The test $a \cdot t$ consists in performing the action a, and in case of success performing the test t. This can be formalized as follows:

Definition 18. *Given a labelled Markov Process* $\mathcal{C} = (\mathcal{X}, \Sigma, \mu)$, *we define an indexed family* $\{P_{\mathcal{C}}(\cdot, t)\}_{t \in \mathscr{T}}$ *(such that* $P_{\mathcal{C}}(\cdot, t) : \mathcal{X} \to \mathbb{R}$*) by induction on the structure of* t*:*

$$P_{\mathcal{C}}(x, \omega) = 1; \qquad P_{\mathcal{C}}(x, a \cdot t) = \int P_{\mathcal{C}}(\cdot, t) d\mu_{x,a}; \qquad P_{\mathcal{C}}(x, \langle t, s \rangle) = P_{\mathcal{C}}(x, t) \cdot P_{\mathcal{C}}(x, s).$$

From our point of view, the key result is the following one:

Theorem 2 ([29]). *Let* $\mathcal{C} = (\mathcal{X}, \Sigma, \mu)$ *be a LMP. Then* $x, y \in \mathcal{X}$ *are bisimilar iff* $P_{\mathcal{C}}(x, t) = P_{\mathcal{C}}(y, t)$ *for every test* $t \in \mathscr{T}$.

5.2 From LMPs to LMCs

We are now going to adapt Theorem 2 to LMCs, thus getting an analogous characterization of probabilistic bisimilarity for them.

Let $\mathcal{M} = (\mathcal{X}, \mathscr{A}, \mathcal{P})$ be a LMC. The function $\mu_{\mathcal{M}} : \mathcal{X} \times \mathscr{A} \times \mathscr{P}(\mathcal{X}) \to [0, 1]$ is defined by $\mu_{\mathcal{M}}(s, a, X) = \sum_{x \in X} \mathcal{P}(s, a, x)$. This construction allows us to see any LMC as a LMP:

Lemma 4. *Let* $\mathcal{M} = (\mathcal{X}, \mathscr{A}, \mathcal{P})$ *be a LMC. Then* $(\mathcal{X}, \mathscr{P}(\mathcal{X}), \mu_{\mathcal{M}})$ *is a LMP, that we denote as* $\mathcal{C}_{\mathcal{M}}$.

But how about bisimulation? Do we get the same notion of equivalence this way? The answer is positive:

Lemma 5. *Let* $\mathcal{M} = (\mathcal{X}, \mathscr{A}, \mathcal{P})$ *be a LMC, and let* R *be an equivalence relation over* $\mathcal{X}$. *Then* R *is a bisimulation with respect to* $\mathcal{M}$ *if and only if* R *is a bisimulation with respect to* $\mathcal{C}_{\mathcal{M}}$. *Moreover, two states are bisimilar with respect to* $\mathcal{M}$ *iff they are bisimilar with respect to* $\mathcal{C}_{\mathcal{M}}$.

Let $\mathcal{M} = (\mathcal{X}, \mathscr{A}, \mathcal{P})$ be a LMC. We define an indexed family $\{P_{\mathcal{M}}(\cdot, t)\}_{t \in \mathscr{T}}$ by $P_{\mathcal{M}}(x, t) = P_{\mathcal{C}_{\mathcal{M}}}(x, t)$, the latter being the function from Definition 18 applied to the Markov process $\mathcal{C}_{\mathcal{M}}$. As a consequence of the previous results in this section, we get that:

Theorem 3. *Let* $\mathcal{M} = (\mathcal{X}, \mathscr{A}, \mathcal{P})$ *be a LMC. Then two states* $x, y \in \mathcal{X}$ *are bisimilar if and only if for all tests* $t \in \mathscr{T}$, $P_{\mathcal{M}}(x, t) = P_{\mathcal{M}}(y, t)$.

The last result derives appropriate expressions for the $P_{\mathcal{M}}(\cdot,\cdot)$, which will be extremely useful in the next section:

Proposition 6. *Let $\mathcal{M} = (\mathcal{X}, \mathscr{A}, \mathcal{P})$ be a LMC. For all $x \in \mathcal{X}$, and $t \in \mathscr{T}$, we have:*

$$P_{\mathcal{M}}(x,\omega) = 1; \qquad P_{\mathcal{M}}(x, a\cdot t) = \sum_{s\in\mathcal{X}} \mathcal{P}(x,a,s)\cdot P_{\mathcal{M}}(s,t); \qquad P_{\mathcal{M}}(x,\langle t,s\rangle) = P_{\mathcal{M}}(x,t)\cdot P_{\mathcal{M}}(x,s).$$

5.3 Every Test has an Equivalent Context

We are going to consider the labelled Markov chain $\mathcal{M}_{\oplus}$ defined previously. We know that two programs M and N in $\mathcal{T}^{\sigma}$ are bisimilar if and only if the states (M,σ) and (N,σ) have exactly the same probability to succeed for the tests in $\mathscr{T}$, measured according to $P_{\mathcal{M}}(\cdot,\cdot)$. Proving that context equivalence is included in bisimulation boils down to show that if M and N have exactly the same convergence probability for all contexts, then they have exactly the same success probability for all tests. Or, more precisely, that for a given test t, and a given type σ, there exists a context C, such that for every term M of type σ, the success probability of t on (M,σ) is *exactly* the convergence probability of $C[M]$. However, we should take into account states in the form $(\hat{V},\sigma) \in \mathcal{S}_{\oplus}$, where V is a value. The formalisation of the just described idea is the following Lemma:

Lemma 6. *Let σ be a type, and t a test. Then there are contexts C_t^{σ}, and D_t^{σ} such that $\emptyset \vdash C_t^{\sigma}(\emptyset;\sigma) : \mathbf{bool}$, $\emptyset \vdash D_t^{\sigma}(\emptyset;\sigma) : \mathbf{bool}$, and for every $M \in \mathcal{T}^{\sigma}$ and every $V \in \mathcal{V}^{\sigma}$, it holds that*

$$P_{\mathcal{M}_{\oplus}}((M,\sigma),t) = \sum [\![C_t^{\sigma}[M]]\!]; \qquad P_{\mathcal{M}_{\oplus}}((\hat{V},\sigma),t) = \sum [\![D_t^{\sigma}[V]]\!].$$

The proof of Lemma 6 is by induction on the structure of the test t. If $t = \omega$, we can take $(\lambda x.\underline{\mathsf{true}})(\lambda x.[\cdot])$ for C_t^{σ}, and D_t^{σ}, since it always converges. If $t = \langle t,s\rangle$, we want to have a context which makes two copies of a term M, and applies t to the first copy and s to the second copy; this strategy can of course be implemented. The most delicate case is the one in which $t = a\cdot s$. We consider here only the case where $a = \mathit{eval}$. We take $D_t^{\sigma} = (\lambda x.[\cdot])\Omega$, since eval is aimed to be applied only to states of the form (M,σ). If D_s^{σ} is the context associated to s for values of type σ, we take $C_t^{\sigma} = (\lambda x.D_s^{\sigma}[x])[\cdot]$. Since the evaluation is call-by-value, the reduction of $C_t^{\sigma}[M]$ is done in the following way: first M is evaluated , and then the context D_s^{σ} is applied to the result of the evaluation of M. So the probability of convergence of $C_t^{\sigma}[M]$ is equal to $\sum_{V\in\mathcal{V}^{\sigma}} ([\![M]\!](V)\cdot(\sum [\![D_s^{\sigma}[V]]\!]))$, which is precisely what we wanted. Please observe that it couldn't be done similarly in a call-by-name setting, since $((\lambda x.B[x])[\cdot])\,[M]$ has there the same probability of convergence that $B[M]$.

It follows from Lemma 6 that if two well-typed closed terms are context equivalent, they are bisimilar:

Theorem 4. *Let M, N be terms such that $\emptyset \vdash M \equiv N : \sigma$. Then $\emptyset \vdash M \backsim_{\circ} N : \sigma$.*

Proof. Let t be a test. We have that, since $M \equiv N$,

$$P_{\mathcal{M}_{\oplus}}((M,\sigma),t) = \sum [\![C_t^{\sigma}[M]]\!] = \sum [\![C_t^{\sigma}[N]]\!] = P_{\mathcal{M}_{\oplus}}((N,\sigma),t),$$

where C_t^σ is the context from Lemma 6. By Theorem 3, (M, σ) and (N, σ) are bisimilar. So $\emptyset \vdash M \backsim_\circ N : \sigma$ which is the thesis.

We can now easily extend this result to terms in $\mathcal{T}_\sigma^\Gamma$, which gives us Full Abstraction: bisimilarity and context equivalence indeed coincide.

Theorem 5 (Full Abstraction). *Let M and N be terms in $\mathcal{T}_\sigma^\Gamma$ .Then $\Gamma \vdash M \equiv N : \sigma$ iff $\Gamma \vdash M \backsim_\circ N : \sigma$.*

5.4 The Asymmetric Case

Theorem 5 establishes a precise correspondence between bisimulation and context equivalence. This is definitely not the end of the story — surprisingly enough, indeed, *simulation* and the contextual *preorder* do not coincide, and this section gives a counterexample, namely a pair of terms which can be compared in the context preorder but which are not similar.

Let us fix the following terms: $M = \lambda x.\lambda y.(\Omega \oplus I)$ and $N = \lambda x.(\lambda y.\Omega) \oplus (\lambda y.I)$. Both these terms can be given the type $\sigma = \mathbf{bool} \to \mathbf{bool} \to \mathbf{bool} \to \mathbf{bool}$ in the empty context. The first thing to note is that M and N cannot even be compared in the simulation preorder:

Lemma 7. *It is not the case that $\emptyset \vdash M \precsim_\circ N : \sigma$ nor that $\emptyset \vdash N \precsim_\circ M : \sigma$.*

We now proceed by proving that M and N can be compared in the *contextual* preorder. We will do so by studying their dynamics seen as terms of $\Lambda_\oplus$ [6] (in which the only constructs are variables, abstractions, applications and probabilistic choices, and in which types are absent) rather than terms of $\mathsf{PCFL}_\oplus$. We will later argue why this translates back into a result for $\mathsf{PCFL}_\oplus$. This detour allows to simplify the overall treatment without sacrificing generality. From now on, then M and N are seen as pure terms, where Ω takes the usual form $(\lambda x.xx)(\lambda x.xx)$.

Let us introduce some notation now. First of all, three terms need to be given names as follows: $L = \lambda y.(\Omega \oplus I)$, $L_0 = \lambda y.\Omega$, and $L_1 = \lambda y.I$. If $b = b_1, \ldots, b_n \in \{0,1\}^n$, then L_b denotes the sequence of terms $L_{b_1} \cdots L_{b_n}$. If P is a term, $P \Rightarrow^p$ means that there is distribution $\mathscr{D}$ such that $P \Rightarrow \mathscr{D}$ and $\sum \mathscr{D} = p$ (where $\Rightarrow$ is small-step approximation semantics [6]; see [4] for more details).

The idea, now, is to prove that in any term P, if we replace an occurrence of M by an occurrence of N, we obtain a term R which converges with probability smaller than the one with which P converges. We first need an auxiliary lemma, which proves a similar result for L_0 and L_1.

Lemma 8. *For every term P, if $(P[L_0/x]) \Rightarrow^p$, then there is another real number $q \geq p$ such that $(P[L_1/x]) \Rightarrow^q$.*

Proof. First, we can remark that, for every term P and any variable z which doesn't appear in P, $P[L_0/x] = (P[\lambda y.z/x])\,[\Omega/z]$, and $P[L_1/x] = (P[\lambda y.z/x])\,[I/z]$. It is thus enough to show that for every term R, if $(R[\Omega/x]) \Rightarrow^p$, then there is $q \geq p$ such that $(R[I/x]) \Rightarrow^q$. This is an induction on the proof of $(R[\Omega/x]) \Rightarrow^p$, i.e., an induction on the structure of a derivation of $(R[\Omega/x]) \Rightarrow \mathscr{D}$ where $\sum \mathscr{D} = p$. Some interesting cases:

- If $(R[\Omega/x]) = V$ is a value, then the term $(R[I/x])$ is a value too. So we have $(R[I/x]) \Rightarrow \{(R[I/x])^1\}$, and so $(R[I/x]) \Rightarrow^1$, and the thesis holds.
- Suppose that the derivation looks as follows:

$$\frac{(R[\Omega/x]) \to \overline{T} \qquad T_i \Rightarrow \mathscr{E}_i}{(R[\Omega/x]) \Rightarrow \sum_{1 \leq i \leq k} \frac{1}{k} \cdot \mathscr{E}_i}$$

  Then there are two possible cases :
  - If $R[\Omega/x] \to T_1, \ldots, T_k$, but the involved redex is *not* Ω, then we can easily prove that each T_i can be written in the form $U_i[\Omega/x]$, where

$$R[\Omega/x] \to U_1[\Omega/x], \ldots, U_k[\Omega/x].$$

    Similarly $R[I/x] \to U_1[I/x], \ldots, U_k[I/x]$. We can then apply the induction hypothesis to each of the derivations for $U_i[\Omega/x]$.
  - The interesting case is when the active redex in $R[\Omega/x]$ is Ω. Since we have $\Omega \to \Omega$, we have $R[\Omega/x] \to R[\Omega/x]$, and so $\overline{T} = T_1 = R[\Omega/x]$, and $\mathscr{D} = \mathscr{E}_1$. We can apply the induction hypothesis to $T_1 \Rightarrow \mathscr{E}_1$, and the thesis follows.

This concludes the proof. □

We are now ready to prove the central lemma of this section, which takes a rather complicated form just for the sake of its inductive proof:

Lemma 9. *Suppose that P is a term and suppose that $(P[M, L/x, \overline{y}]) \Rightarrow^p$, where $\overline{y} = y_1, \ldots, y_n$. Then for every $b \in \{0,1\}^n$ there is p_b such that $(P[N, L_b/x, \overline{y}]) \Rightarrow^{p_b}$ and $\sum_b \frac{p_b}{2^n} \geq p$.*

Proof. This is an induction on the proof of $(P[M, L/x, \overline{y}]) \Rightarrow^p$, i.e., an induction on the structure of a derivation of $(P[M, L/x, \overline{y}]) \Rightarrow \mathscr{D}$ where $\sum \mathscr{D} = p$:

- If $P[M, L/x, \overline{y}]$ is a value, then:
  - either $p = 1$, but we can also choose p_b to be 1 for every b, since the term $P[N, L_b/x, \overline{y}]$ is a value, too;
  - or $p = 0$, and in this case we can fix p_b to be 0 for every b.
- If $P[M, L/x, \overline{y}] \to R_1, \ldots, R_k$, but the involved redex has *not* M nor L as functions, then we are done, because one can easily prove in this case that each R_i can be written in the form $T_i[M, L/x, \overline{y}]$, where

$$P[N, L_b/x, \overline{y}] \to T_1[N, L_b/x, \overline{y}], \ldots, T_k[N, L_b/x, \overline{y}].$$

  It suffices, then, to apply the induction hypothesis to each of the derivations for $T_i[M, L/x, \overline{y}]$, easily reaching the thesis;
- The interesting case is when the active redex in $P[M, L/x, \overline{y}]$ has either M or L (or, better, occurrences of them coming from the substitution) in functional position.
  - If M is involved, then there are a term R and a variable z such that

$$P[M, L/x, \overline{y}] \to R[M, L, L/x, \overline{y}, z];$$
$$P[N, L_b/x, \overline{y}] \to R[N, L_b, L_0/x, \overline{y}, z], \to R[N, L_b, L_1/x, \overline{y}, z].$$

    This, in particular, means that we can easily apply the induction hypothesis to $R[M, L, L/x, \overline{y}, z]$.

- If, on the other hand L is involved in the redex, then there are a term R and a variable z such that

$$P[M, L/x, \overline{y}] \to R[M, L, \Omega/x, \overline{y}, z], R[M, L, I/x, \overline{y}, z].$$

Moreover, the space of all sequences b can be partitioned into two classes of the same cardinality 2^{n-1}, call them B_B and B_G; for every $b \in B_B$, we have that $P[N, L_b/x, \overline{y}]$ is diverging, while for every $b \in B_G$, we have that

$$P[N, L_b/x, \overline{y}] \to R[N, L_b, I/x, \overline{y}, z].$$

Observe how for any $b \in B_B$ there is $\hat{b} \in B_G$ such that b and $\hat{b}$ agree on every bit except one, which is 0 in b and 1 in $\hat{b}$. Now, observe that $p = \frac{q}{2}$ where $R[M, L, I/x, \overline{y}, z] \Rightarrow^q$. We can then apply the induction hypothesis and obtain that $q \leq \sum_b \frac{q_b}{2^n}$ where $R[N, L_b, I/x, \overline{y}, z] \Rightarrow^{q_b}$. Due to Lemma 8, we can assume without losing generality that $q_b \leq q_{\hat{b}}$ for every $b \in B_B$. Now, fix $p_b = 0$ if $b \in B_B$ and $p_b = q_b$ if $b \in B_G$. Of course $(P[N, L_b/x, \overline{y}]) \Rightarrow^{p_b}$. But moreover,

$$p = \frac{q}{2} \leq \frac{1}{2} \sum_b \frac{q_b}{2^n} \leq \frac{1}{2} \sum_{b \in B_G} \frac{2 \cdot q_b}{2^n} = \sum_{b \in B_G} \frac{q_b}{2^n} = \sum_b \frac{p_b}{2^n}.$$

This concludes the proof. □

From what we have seen so far, it is already clear that for any context C, it cannot be that $\sum[\![C[M]]\!] > \sum[\![C[N]]\!]$, as this would mean that for a certain term P, $P[M/x]$ would converge to a distribution $\mathscr{D}$ whose sum p is higher than the sum of any distribution to which $P[N/x]$ converges, and this is in contradiction with Lemma 9: simply consider the case where $n = 0$.

But how about $\mathsf{PCFL}_\oplus$? Actually, there is an embedding $\langle\!\langle \cdot \rangle\!\rangle$ of $\mathsf{PCFL}_\oplus$ into $\Lambda_\oplus$ such that for every $P \in \mathcal{T}^\sigma$, it holds that $\sum[\![P]\!] = \sum[\![\langle\!\langle P \rangle\!\rangle]\!]$ (again, see [4] for more details). As a consequence there cannot be any $\mathsf{PCFL}_\oplus$ context contradicting what we have said in the last paragraph. Summing up,

Theorem 6. *The simulation preorder $\precsim_\circ$ is not fully abstract.*

The careful reader may now wonder whether a result akin to Theorem 3 exists for *simulation* and testing. Actually, there *is* such a result [29], but for a different notion of test, which not only, like $\mathscr{T}$, includes conjunctive tests, but also *disjunctive* ones. Now, anybody familiar with the historical developments of the quest for a fully abstract model of PCF [25, 2] would immediately recognize disjunctive tests as something which cannot be easily implemented by terms.

6 A Comparison with Call-by-Name

Actually, $\mathsf{PCFL}_\oplus$ could easily be endowed with call-by-name rather than call-by-value operational semantics. The obtained calculus, then, is amenable to a treatment similar

to the one described in Section 4. Full abstraction, however, holds neither for simulation nor for bisimulation. These results are given in more detail in [4], and are anyway among the major contributions of [5]. The precise correspondence between testing and bisimulation described in Section 5.2 shed some further light on the gap between call-by-value and call-by-name evaluation. In both cases, indeed, bisimulation can be characterized by testing as given in Definition 17. What call-by-name evaluation misses, however, is the capability to copy a term *after* having evaluated it, a feature which is instead available if parameters are passed to function evaluated, as in call-by-value. In a sense, then, the tests corresponding to bisimilarity are the same in call-by-name, but the calculus turns out to be too poor to implement all of them. We conjecture that the subclass of tests which are implementable in a call-by-name setting are those in the form $\langle t_1, \ldots, t_n \rangle$ (where each t_i is in the form $a_i^1 \cdot \ldots \cdot a_i^{m_i} \cdot \omega$), and that full abstraction can be recovered if the language is endowed with an operator for *sequencing*.

7 Conclusions

In this paper, we study probabilistic applicative bisimulation in a call-by-value scenario, in the meantime generalizing it to a typed language akin to Plotkin's PCF. Actually, some of the obtained results turn out to be surprising, highlighting a gap between the symmetric and asymmetric cases, and between call-by-value and call-by-name evaluation. This is a phenomenon which simply does not show up when applicative bisimulation is defined over deterministic [1] nor over nondeterministic [18] λ-calculi. The path towards these results goes through a characterization of bisimilarity by testing which is known from the literature [29]. Noticeably, the latter helps in finding the right place for probabilistic λ-calculi in the coinductive spectrum: the corresponding notion of test is more powerful than plain trace equivalence, but definitely less complex than the infinitary notion of test which characterizes applicative bisimulation in presence of nondeterminism [20].

Further work includes a broader study on (not necessarily coinductive) notions of equivalence for probabilistic λ-calculi. As an example, it would be nice to understand the relations between applicative bisimulation and logical relations (e.g. the ones defined in [13]). Another interesting direction would be the study of notions of *approximate* equivalence for λ-calculi with restricted expressive power. This would be a step forward getting a coinductive characterization of computational indistinguishability, with possibly nice applications for cryptographic protocol verification.

References

1. S. Abramsky. The Lazy λ-Calculus. In D. Turner, editor, *Research Topics in Functional Programming*, pages 65–117. Addison Wesley, 1990.
2. G. Berry and P.-L. Curien. Sequential algorithms on concrete data structures. *Theor. Comput. Sci.*, 20:265–321, 1982.
3. D. Comaniciu, V. Ramesh, and P. Meer. Kernel-based object tracking. *IEEE Trans. on Pattern Analysis and Machine Intelligence,*, 25(5):564–577, 2003.
4. R. Crubille and U. Dal Lago. Probabilistic applicative bisimulation for call-by-value lambda calculi (long version). Available at `http://arxiv.org/abs/1401.3766`, 2014.

5. U. Dal Lago, D. Sangiorgi, and M. Alberti. On coinductive equivalences for higher-order probabilistic functional programs. In *POPL*, pages 297–308, 2014.
6. U. Dal Lago and M. Zorzi. Probabilistic operational semantics for the lambda calculus. *RAIRO - Theor. Inf. and Applic.*, 46(3):413–450, 2012.
7. V. Danos and R. Harmer. Probabilistic game semantics. *ACM Trans. Comput. Log.*, 3(3):359–382, 2002.
8. J. Desharnais, A. Edalat, and P. Panangaden. Bisimulation for labelled markov processes. *Inf. Comput.*, 179(2):163–193, 2002.
9. T. Ehrhard, C. Tasson, and M. Pagani. Probabilistic coherence spaces are fully abstract for probabilistic PCF. In *POPL*, pages 309–320, 2014.
10. S. Goldwasser and S. Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.
11. N. D. Goodman. The principles and practice of probabilistic programming. In *POPL*, pages 399–402, 2013.
12. A. D. Gordon, M. Aizatulin, J. Borgström, G. Claret, T. Graepel, A. V. Nori, S. K. Rajamani, and C. V. Russo. A model-learner pattern for bayesian reasoning. In *POPL*, pages 403–416, 2013.
13. J. Goubault-Larrecq, S. Lasota, and D. Nowak. Logical relations for monadic types. *Mathematical Structures in Computer Science*, 18(6):1169–1217, 2008.
14. D. J. Howe. Proving congruence of bisimulation in functional programming languages. *Inf. Comput.*, 124(2):103–112, 1996.
15. C. Jones and G. D. Plotkin. A probabilistic powerdomain of evaluations. In *LICS*, pages 186–195, 1989.
16. J. Katz and Y. Lindell. *Introduction to Modern Cryptography*. Chapman & Hall Cryptography and Network Security Series. Chapman & Hall, 2007.
17. K. G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Inf. Comput.*, 94(1):1–28, 1991.
18. S. B. Lassen. *Relational Reasoning about Functions and Nondeterminism*. PhD thesis, University of Aarhus, 1998.
19. C. D. Manning and H. Schütze. *Foundations of statistical natural language processing*, volume 999. MIT Press, 1999.
20. C.-H. L. Ong. Non-determinism in a functional setting. In *LICS*, pages 275–286, 1993.
21. S. Park, F. Pfenning, and S. Thrun. A probabilistic language based on sampling functions. *ACM Trans. Program. Lang. Syst.*, 31(1), 2008.
22. J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 1988.
23. A. Pfeffer. IBAL: A probabilistic rational programming language. In *IJCAI*, pages 733–740. Morgan Kaufmann, 2001.
24. A. Pitts. Operationally-based theories of program equivalence. In *Semantics and Logics of Computation*, pages 241–298. Cambridge University Press, 1997.
25. G. D. Plotkin. LCF considered as a programming language. *Theor. Comput. Sci.*, 5(3):223–255, 1977.
26. N. Ramsey and A. Pfeffer. Stochastic lambda calculus and monads of probability distributions. In *POPL*, pages 154–165, 2002.
27. C. Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28:656–715, 1949.
28. S. Thrun. Robotic mapping: A survey. *Exploring artificial intelligence in the new millennium*, pages 1–35, 2002.
29. F. van Breugel, M. W. Mislove, J. Ouaknine, and J. Worrell. Domain theory, testing and simulation for labelled markov processes. *Theor. Comput. Sci.*, 333(1-2):171–197, 2005.