Bisimulations for Probabilistic Linear Lambda Calculi

Yuxin Deng and Yuan Feng[y]
Shanghai Key Laboratory of Trustworthy Computing, MOE International Joint Lab of Trustworthy Software,
and International Research Center of Trustworthy Software, East China Normal University
[y]University of Technology Sydney, Australia

***Abstract*—We investigate a notion of probabilistic program equivalence under linear contexts. We show that both a state-based and a distribution-based bisimilarity are sound coinductive proof techniques for reasoning about higher-order probabilistic programs, but only the distribution-based one is complete for linear contextual equivalence. The completeness proof is novel and directly constructs linear contexts from transitions, rather than the traditional approach of characterizing bisimilarities as testing equivalences.**

***Index Terms*—Probabilistic lambda calculi; Contextual equivalence; Bisimulation; Full abstraction**

I. Introduction

Contextual equivalence in the style of Morris [24] is an important notion of program equivalence that can be used to formalize and reason about many interesting properties of computing systems. It has been widely studied in both sequential and concurrent languages. However, to directly prove that two programs are contextually equivalent is a difficult task because of the universal quantification over all possible contexts. This is especially the case when we consider programs in higher-order languages. Therefore, it is interesting to develop sound techniques that can help to establish contextual equivalence. In this aspect, applicative bisimulation [1], [20] and logical relations [27], [31] are successful in proving contextual equivalence for programs in functional languages. However, it is known that the completeness of logical relations are often hard to achieve, especially for higher-order types. In contrast, characterizations in terms of bisimulations have received a lot of attention.

For probabilistic functional languages, there are also attempts to develop operational techniques for contextual equivalence. For example, Dal Lago et al. [10] have generalized Abramsky's applicative bisimulation [1] to a call-by-name, untyped probabilistic λ-calculus. Probabilistic applicative bisimulation is shown to be sound but incomplete with respect to contextual equivalence. Full abstraction is recovered when pure, deterministic λ-terms are considered, and when an involved notion of coupled logical bisimulation is used instead of applicative bisimulation. Crubillé and Dal Lago [7] have investigated a call-by-value, typed probabilistic λ-calculus. In that setting, applicative bisimilarity turns out to coincide with contextual equivalence. However, the correspondence does not hold if we consider applicative simulation and contextual preorder. Rioli [28] has defined a call-by-value, linearly typed probabilistic λ-calculus, where probabilistic bisimilarity is sound but incomplete with respect to contextual equivalence. All the probabilistic bisimulations mentioned above are state-based in the sense that they are binary relations first defined for states on labeled Markov chains, and then lifted to be a relation on distributions by a lifting operation [23], [15], [11].

In the current work, we are interested in characterizing linear contextual equivalence as proposed in [14], where the programs to be compared are examined in linear contexts and used exactly once. In general, this leads to a coarser relation than the usual notion of contextual equivalence since the discriminating power of contexts is limited when the capacity of duplicating resources is prohibited; see e.g. [14], [2]. Linear contexts are motivated from computational cryptography and are also natural when observing the behavior of quantum programs [12]. To formally define linear contexts, we work in the framework of a probabilistic λ-calculus similar to the language used in [28] except that we allow both linear and non-linear types. Linear types are useful for introducing imperative concepts into functional programming [18], structural complexity theory [19], or analyzing memory allocation [34]. Moreover, the linear λ-calculus can be employed to formalize deductive systems [5]. The study of linearity in concurrent languages such as the π-calculus allows for a fine-grained analysis of process behavior [22], [35], [3]. Non-linear types are also admitted here because pure linear languages have limited expressive power. Cappai and Dal Lago [4] have worked in a similar setting and shown that linear contextual equivalence coincides with a notion of trace equivalence, which generalizes both the result and the proof idea of linear context reduction in [14] from the nonprobabilistic paradigm to the probabilistic framework. They have also proved that applicative bisimilarity is sound but incomplete for linear contextual equivalence. This mismatch persists when going from equivalences to metrics [8].

We show that it is possible to give a coinductive and complete characterization of linear contextual equivalence. The reason that completeness fails in [4], [28] lies in the fact that the bisimulations investigated there are state-based. In order to achieve completeness, the key idea is to shift our focus from state-based bisimulations to distribution-based ones, that is, by directly defining bisimulations as a binary relation on (sub)distributions. More precisely, although both notions of bisimilarities are sound with respect to linear contextual

978-1-5386-1925-4/17/$31.00 © 2017 IEEE

equivalence, the state-based one is too strong and only the distribution-based one is complete. Technically speaking, the soundness part requires to show that both bisimilarities are congruence relations. Since Howe's method [20], [26] is successful in showing the congruence property of applicative bisimilarity in various dialects of the λ-calculi, it is a very robust approach. Thus, we extend it to the probabilistic setting. For the state-based bisimilarity, the use of Howe's method is similar to that in [10], [6], [4], [9], [28]. For the distribution-based bisimilarity, however, there is a subtlety because Howe's lifting inherently demands syntax-directed rules, which are naturally defined on terms but not on distributions. Our solution is to work with distributions that can be viewed as the denotations of terms (intuitively, the denotation of a term is the unique subdistribution obtained from the term by keeping reducing it until no more reduction is possible). If a visible action is performed on a term, its denotation can also perform the same action and evolve into a distribution that can still be regarded as the denotation of another term. This property is also useful for showing the completeness of distribution-based bisimilarity with respect to linear contextual equivalence.

The current work can be considered as a companion of [12] where two coinductive proof techniques for quantum lambda calculi are proposed. The definitions there are complicated due to the use of quantum closures that represent the states of quantum systems. In order to separate general proof ideas that are robust and applicable to broader settings from quantum closures and render them accessible to a wider audience, we now work in a probabilistic functional language. We show that linear contextual equivalence is exactly captured by the distribution-based bisimilarity defined in [16]. Furthermore, a notable improvement over [12] is the proof of the completeness of distribution-based bisimilarity with respect to linear contextual equivalence. In the quantum setting the completeness proof works by introducing a testing equivalence as a stepping stone and then turning each test into a linear context. The same phenomenon appeared in [7], where the completeness proof of state-based bisimilarity for the non-linear contextual equivalence crucially relies on a characterization of state-based bisimilarity as a testing equivalence by van Breugel et al. [32]; the proof of the characterization itself, in turn, involves advanced machinery such as the Lawson topology on probabilistic powerdomains [21] and Banach algebras. Our completeness proof here uses a different strategy and directly links labeled transitions to contexts. More specifically, if the denotation of term M as a subdistribution can perform an action a, then it will result in a subdistribution that can be viewed as the denotation of a term $C^a[M]$, where C^a is a linear context fully determined by action a and the type of M. This key observation leads to a neat completeness proof.

In [10] and [30], coupled logical bisimilarity and environmental bisimilarity are used to give complete characterizations of non-linear contextual equivalence for lambda calculi. Both notions of bisimilarity are based on formal sums, which are syntactic representations of probability subdistributions. However, our distribution-based bisimilarity has a much simpler formulation than both of them and is very close to the classical non-probabilistic bisimilarity (cf. Definition 7).

The rest of the paper is structured as follows. In Section II we introduce the syntax, the reduction semantics, and a labeled transition semantics of a probabilistic λ-calculus. A linear contextual equivalence and two bisimilarities are defined. In Section III we use the Howe's method to show that the state-based bisimilarity is a congruence relation included in linear contextual equivalence. In Section IV we show that the distribution-based bisimilarity is sound and complete for linear contextual equivalence. Finally, we conclude in Section V and discuss some future work.

II. A Probabilistic λ-Calculus

In this section we introduce the syntax and operational semantics of a probabilistic λ-calculus, using similar notation from [25], [12].

A. Syntax

In this language, types are given by the following grammar:

$$A, B, C ::= 1 \mid A \multimap B \mid !(A \multimap B) \mid A \otimes B \mid A \oplus B.$$

Here the unit type is 1. The usual linear function types are in the form $A \multimap B$ and non-linear function types take the form $!(A \multimap B)$. For any arbitrary type A, the type $!A$ can be simulated by $!(1 \multimap A)$. Tensor and sum types are standard.

Terms are built up from constants and variables, using the following constructs:

$M, N, P ::=$	x	Variables
$\mid$	$\lambda x^A.M$	Abstraction
$\mid$	$M\,N$	Application
$\mid$	$M \oplus_p N$	Prob. choice
$\mid$	$\mathtt{skip}$	Skip
$\mid$	$M; N$	Seq. composition
$\mid$	$M \otimes N$	Tensor products
$\mid$	$\mathtt{let}\ x^A \otimes y^B = M\ \mathtt{in}\ N$	Tensor projection
$\mid$	$\mathtt{in}_l M \mid \mathtt{in}_r M$	Sums
$\mid$	$\mathtt{match}\ P\ \mathtt{with}\ (\lambda x^A.M \mid \lambda y^B.N)$	Matches
$\mid$	$\mathtt{letrec}\ f^{A\to B} x = M\ \mathtt{in}\ N$	Recursions

Most of the language constructs are standard. The term $\lambda x^A.M$ represents a linear function that uses the variable x exactly once in M. The term $M \otimes N$ is a pair where both components M and N, are used. The terms $\mathtt{in}_l M$ and $\mathtt{in}_r M$ are injections of term M into the left and right of a sum. Let p be a rational number in the interval $[0, 1]$. The term $(M \oplus_p N)$ describes an unstable system that may evolve into M with probability p and into N with probability $1 - p$.

Variables appearing in the λ-binder, the $\mathtt{let}$-binder, the $\mathtt{match}$-binder, and the $\mathtt{letrec}$-binder are bound variables. We write $fv(M)$ for the set of free variables in term M. We will not distinguish α-*equivalent* terms, which are terms syntactically identical up to a renaming of bound variables. If M and N are terms and x is a variable, then $M\{N/x\}$

$$\frac{A \text{ linear}}{!\Delta, x : A \vdash x : A} \qquad \frac{}{!\Delta, x : !(A \multimap B) \vdash x : A \multimap B} \qquad \frac{!\Delta \vdash V : A \multimap B \quad V \text{ value}}{!\Delta \vdash V : !(A \multimap B)}$$

$$\frac{\Delta, x : A \vdash M : B}{\Delta \vdash \lambda x^{A}.M : A \multimap B} \qquad \frac{!\Delta, \Delta' \vdash M : A \multimap B \quad !\Delta, \Delta'' \vdash N : A}{!\Delta, \Delta', \Delta'' \vdash MN : B}$$

$$\frac{!\Delta, \Delta' \vdash M : A \quad !\Delta, \Delta' \vdash N : A}{!\Delta, \Delta' \vdash M \otimes N : A} \qquad \frac{}{!\Delta \vdash \mathtt{skip} : 1} \qquad \frac{!\Delta, \Delta' \vdash M : 1 \quad !\Delta, \Delta'' \vdash N : A}{!\Delta, \Delta', \Delta'' \vdash M; N : A}$$

$$\frac{!\Delta, \Delta' \vdash M : A \quad !\Delta, \Delta'' \vdash N : B}{!\Delta, \Delta', \Delta'' \vdash M \otimes N : A \otimes B} \qquad \frac{!\Delta, \Delta' \vdash M : A \otimes B \quad !\Delta, \Delta'', x : A, y : B \vdash N : C}{!\Delta, \Delta', \Delta'' \vdash \mathtt{let}\ x^{A} \otimes y^{B} = M\ \mathtt{in}\ N : C}$$

$$\frac{!\Delta, \Delta' \vdash M : A}{!\Delta, \Delta' \vdash \mathtt{in}_{l}\ M : A \oplus B} \qquad \frac{!\Delta, \Delta' \vdash M : B}{!\Delta, \Delta' \vdash \mathtt{in}_{r}\ M : A \oplus B}$$

$$\frac{!\Delta, \Delta' \vdash P : A \oplus B \quad !\Delta, \Delta'', x : A \vdash M : C \quad !\Delta, \Delta'', y : B \vdash N : C}{!\Delta, \Delta', \Delta'' \vdash \mathtt{match}\ P\ \mathtt{with}\ (x^{A} : M \mid y^{B} : N) : C}$$

$$\frac{!\Delta, f : !(A \multimap B), x : A \vdash M : B \quad !\Delta, \Delta', f : !(A \multimap B) \vdash N : C}{!\Delta, \Delta' \vdash \mathtt{letrec}\ f^{A(\ B} x = M\ \mathtt{in}\ N : C}$$

Fig. 1. Typing Rules.

denotes the term resulted from substituting N for all free occurrences of x in M. More generally, given a list $N_1, \ldots, N_n$ of terms and a list $x_1, \ldots, x_n$ of distinct variables, we write $M\{N_1/x_1, \ldots, N_n/x_n\}$ or simply $M\{\tilde{N}/\tilde{x}\}$ for the result of simultaneously substituting each N_i for free occurrences in M of the corresponding variable x_i.

Values are special terms in the following form

$$V, W ::= x \mid \mathtt{skip} \mid \lambda x^{A}.M \mid V \otimes W \mid \mathtt{in}_{l}\ V \mid \mathtt{in}_{r}\ W.$$

As syntactic sugar we write $\mathbb{B}\mathrm{ool} = 1 \oplus 1$, $\mathtt{tt} = \mathtt{in}_{r}\ \mathtt{skip}$, and $\mathtt{ff} = \mathtt{in}_{l}\ \mathtt{skip}$.

A *typing assertion* takes the form $\Delta \vdash M : A$, where Δ is a finite partial function from variables to types, M is a term, and A is a type. We write $dom(\Delta)$ for the domain of Δ. We call Δ *exponential* (resp. *linear*) whenever $\Delta(x)$ is (resp. is not) a $!$-type for each $x \in dom(\Delta)$. We write $!\Delta$ for a context that is exponential. The *type assignment relation* consists of all typing assertions that can be derived from the axioms and rules in Figure 1, where the contexts Δ' and Δ'' are assumed to be linear. The notation $\Delta, x : A$ denotes the partial function which properly extends Δ by mapping x to A, so it is assumed that $x \notin dom(\Delta)$. Similarly, in the notation Δ, Δ' the domains of Δ and Δ' are assumed to be disjoint. All the typing rules are standard for linear functional languages. We write $Prog(A) = \{M \mid \cdot \vdash M : A\}$ for the set of all closed programs of type A and $Prog$ the set of all closed programs.

For any typing assertion $\Delta \vdash M : A$, it is not difficult to see that $fv(M) \subseteq dom(\Delta)$. Let a *proved expression* be a triple $(\Delta; M; A)$ with $\Delta \vdash M : A$. If $\Delta = x_1 : A_1, \ldots, x_n : A_n$, a Δ-*closure* is a substitution $\{\tilde{M}/\tilde{x}\}$ where each $M_i \in Prog(A_i)$. If R is a relation on terms M with $fv(M) = \emptyset$, its *open extension*, R°, is the least relation between proved expressions such that $(\Delta; M; A)\ R^{\circ}\ (\Delta; N; A)$ if and only if $M\{\tilde{P}/\tilde{x}\}\ R\ N\{\tilde{P}/\tilde{x}\}$ for any Δ-closure $\{\tilde{P}/\tilde{x}\}$. We will write $\Delta \vdash M\ R\ N : A$ to mean that $(\Delta; M; A)\ R^{\circ}\ (\Delta; N; A)$. Sometimes we omit the type information if it is not important and simply write $\Delta \vdash M\ R\ N$.

B. The Reduction Semantics

The reduction semantics is defined in terms of Markov chains. We introduce $M \overset{p}{\rightsquigarrow} N$ to mean that term M reduces to N immediately with probability $p \in (0, 1]$. Formally, the one-step reduction $\overset{p}{\rightsquigarrow}$ between terms is the smallest relation including the axioms from Figure 2 together with the structural rule

$$\frac{M \overset{p}{\rightsquigarrow} N}{E\{M/x\} \overset{p}{\rightsquigarrow} E\{N/x\}}$$

where E is any *evaluation context* generated by the grammar

$$\begin{aligned} E ::= {} & [\,] \mid E\ M \mid V\ E \mid E; M \mid E \otimes M \mid V \otimes E \\ & \mid \mathtt{in}_{l}\ E \mid \mathtt{in}_{r}\ E \mid \mathtt{let}\ x^{A} \otimes y^{B} = E\ \mathtt{in}\ M \\ & \mid \mathtt{match}\ E\ \mathtt{with}\ (x^{A} : M \mid y^{B} : N). \end{aligned}$$

The first six axioms are in the form $M \overset{1}{\rightsquigarrow} N$. They represent reductions in classical languages where term M is determined to evolve into term N with probability 1. The reduction semantics defined above employs a *call-by-value* evaluation strategy.

Lemma 1 (Type safety): Let M be a closed term. Then either M is a value or $M \overset{p_i}{\rightsquigarrow} M_i$ with $i = 1; 2$ satisfying $p_1 + p_2 = 1$ and each M_i preserves the type of M.

Let S be a countable set. A *(discrete) probability subdistribution* over S is a function $\mathcal{D} : S \to [0, 1]$ with $\sum_{s \in S} \mathcal{D}(s) \leq 1$. It is a *(full) distribution* if $\sum_{s \in S} \mathcal{D}(s) = 1$. Let $d\mathcal{D}e := \{s \in S \mid \mathcal{D}(s) > 0\}$ be the *support* and $|\mathcal{D}| := \sum_{s \in d\mathcal{D}e} \mathcal{D}(s)$ the *size* of subdistribution $\mathcal{D}$. Let $\mathcal{D}(S)$ denote the set of all subdistributions over S. By Lemma 1 we see that the reduction semantics induces a Markov chain

$$(\lambda x^{A}.M)V \xrightarrow{1} M\{V/x\}$$
$$\texttt{let } x^{A} \otimes y^{B} = V \otimes W \texttt{ in } N \xrightarrow{1} N\{V/x, W/y\}$$
$$\texttt{skip}; N \xrightarrow{1} N$$
$$\texttt{match in}_{l}\ V \texttt{ with } (x^{A}.M \mid y^{B}.N) \xrightarrow{1} M\{V/x\}$$
$$\texttt{match in}_{r}\ V \texttt{ with } (x^{A}.M \mid y^{B}.N) \xrightarrow{1} N\{V/y\}$$
$$\texttt{letrec } f^{A \multimap B} x = M \texttt{ in } N \xrightarrow{1} N\{(\lambda x^{A}.\texttt{letrec } f^{A \multimap B} x = M \texttt{ in } M)/f\}$$
$$M \oplus_{p} N \xrightarrow{p} M$$
$$M \oplus_{p} N \xrightarrow{1-p} N$$

Fig. 2. Small-Step Axioms.

$(Prog, \mapsto)$, where $Prog$ is the set of all closed programs and $\mapsto \subseteq Prog \times \mathcal{D}(Prog)$ is the transition relation with $M \mapsto$ satisfying $\mathcal{D}(N) = \{p_i \cdot N_i \mid M \xrightarrow{p_i} N_i\}$. Here $\{p_i \cdot N_i \mid M \xrightarrow{p_i} N_i\}$ is a multiset if $M \neq N$, e.g. in the reduction of the term $M \oplus_{\frac{1}{2}} M$.

In order to investigate the long-term behaviour of a Markov chain, we introduce the notion of extreme derivative from [13]. We first lift the relation $\mapsto$ to be a transition relation between subdistributions: $\Delta \mapsto$ if $\Delta = \sum_{M \in \lceil \Delta \rceil} \Delta(M) \cdot \Delta_M$ and $M \mapsto \Delta_M$ for each M in $\lceil \Delta \rceil$.

Definition 1 (Extreme derivative [13]): Suppose we have subdistributions Δ, Δ'_k, Δ_k for $k \geq 0$ with the following properties:

$$\Delta = \Delta'_0 + \Delta_0; \qquad \forall k \geq 0.\ \Delta'_k \mapsto \Delta'_{k+1} + \Delta_{k+1};$$

and each Δ_k is stable in the sense that $\Delta_k \not\mapsto$, for all $M \in \lceil \Delta_k \rceil$. Then we call $\Delta^{\infty} := \sum_{k=0}^{\infty} \Delta_k$ an *extreme derivative* of Δ, and write $\Delta \Rightarrow \Delta^{\infty}$. The symbol Δ'_k represents a subdistribution that can reduce, and Δ_k a subdistribution that cannot reduce.

Let M be a program in the Markov chain $(Prog, \mapsto)$. The extreme derivative of the point distribution on M that assigns probability 1 to M, written $\overline{M}$, is unique, and we use it for the denotation of M, indicated as $[\![M]\!]$. So we always have $\overline{M} \Rightarrow [\![M]\!]$. Note that in the presence of divergence $[\![M]\!]$ may be a proper subdistribution.

Extreme derivatives can also be defined by a *big-step* semantics using a binary relation $\Downarrow$ between terms and value distributions. The defining rules are listed in Figure 3, where "ε" stands for the empty subdistribution.

Lemma 2: $[\![M]\!] = \sup\{\Delta \mid M \Downarrow \Delta\}$, where the supremum of subdistributions are computed component-wisely.

Example 1: We define a non-terminating program Ω as follows:

$$\Omega := \texttt{letrec } f^{\mathsf{Bool} \multimap \mathsf{Bool}} x = f x \texttt{ in } f\ \texttt{tt}$$

It is easy to check that $\Omega \xrightarrow{1} \xrightarrow{1} \Omega$ and $[\![\Omega]\!] = \varepsilon$. More generally, for any type A, we define the term Ω_A that will be used later on.

$$\Omega_A := \texttt{letrec } f^{A \multimap A} x = f x \texttt{ in } f$$

Let M be a term of type A. It can be checked that $[\![\Omega_A\, M]\!] = \varepsilon$.

Following [14] we would like to give an alternative characterization of linear contextual equivalence. Intuitively, a linear context is a context where the programs under examination will be evaluated and used *exactly once*.

Definition 2: A *linear context* is a term with a single free variable x such that the typing assertion $x : A \vdash C_{x:A} : \mathsf{B}$, where A is a linear type, can be derived from the rules in Figure 1. We often omit the variable and the type subscription when they are clear from the text or irrelevant.

Contextual equivalence is usually based on some notion of observables. The termination of a term is observable, which is very natural. For the probabilistic-calculus, we require that the observable behaviour of a term M is its probability of convergence $|[\![M]\!]|$.

Definition 3: The *linear contextual preorder* is the binary relation $\preceq$ defined as follows: $M \preceq N$ for $M, N \in Prog(A)$ if $|[\![C_{x:A}\{M/x\}]\!]| \leq |[\![C_{x:A}\{N/x\}]\!]|$ for all linear context $C_{x:A}$. *Linear contextual equivalence* $\simeq$ is defined as the symmetrization of $\preceq$, that is, $M \simeq N$ just when $M \preceq N$ and $N \preceq M$.

C. A Probabilistic Labeled Transition System

In [17], Gordon defines explicitly a labeled transition system in order to illustrate the bisimulation technique in HOCF. We follow this idea to define a *probabilistic* labeled transition system for the probabilistic λ-calculus, upon which we can define probabilistic bisimulations.

Transition rules are listed in Figure 4: we make the typing of terms explicit in the rules as the type system plays an important role in defining the operational semantics of typed terms. A transition takes the form $M \xrightarrow{a} \Delta$, where M is a program, Δ is a subdistribution over programs, and a is an action. If Δ is a point distribution $\overline{N}$, we simply write the transition as $M \xrightarrow{a} N$. Transitions represent the way terms interact with environments (or contexts).

The action of the program $\texttt{skip}$ is to exhibit itself to the environment, and after that it cannot provide any information, hence no external transition can occur any more, which is modeled by the empty subdistribution. The transition

$$\lambda x^{A}.M \xrightarrow{@V} [\![M\{V/x\}]\!]$$

says that a λ-abstraction can "consume" a term, which is supplied by the environment as an argument, and forms a reduction. On the contrary, the action of V is to request a term M from the environment and replace appropriate variables in M by V. The next two rules are similar. The last rule turns the big-step evaluation into the external action τ. Therefore, term reductions are considered as internal transitions that are abstracted away. Compared with the labeled transition semantics for lambda calculi defined in [10] and also used in [6], [9], [8], [28], the terms here are not multisorted as we do not treat values in two different ways (as terms versus as values). Moreover, we require that whatever action a term performs it directly reaches an extreme derivative.

The set of programs $Prog$ together with the transition rules in Figure 4 yields a probabilistic labeled transition system