Bisimulations for Probabilistic Linear Lambda Calculi

Yuxin Deng* and Yuan Feng†
*Shanghai Key Laboratory of Trustworthy Computing, East China Normal University
†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 of [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 achive 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 equivalence, the state-based one is too strong and only the distribution-based one is complete. Technically speaking, the

978-1-5386-1925-4/17/$31.00

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 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 $\mathcal{C}^a[M]$, where $\mathcal{C}^a$ 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 enviromental 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 typed language, types are given by the following grammar:

$$\begin{array}{rl} & A, B, C \\ ::= & 1 \mid A \multimap B \mid !(A \multimap B) \mid A \otimes B \mid A \oplus B. \end{array}$$

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	Terms
$::=$	x	Variables
$\mid$	$\lambda x^A \,.\, M$	Abstraction
$\mid$	$M\, N$	Application
$\mid$	$M \;{}_p\!\oplus\; N$	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
$\mid$	$\mathtt{match}\ P\ \mathtt{with}\ (x^A : M \mid y^B : N)$	Matches
$\mid$	$\mathtt{letrec}\ f^{A \multimap B} x = M\ \mathtt{in}\ N$	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 $(M \;{}_p\!\oplus\; N)$ 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 `let`-binder, the `match`-binder, and the `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 $M\{N/x\}$ 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 $\mathsf{Bool} = 1 \oplus 1$, $\mathtt{tt} = \mathtt{in}_r\, \mathtt{skip}$, 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 $\mathcal{P}rog(A) = \{M \mid \emptyset \vdash M : A\}$ for the set of all closed programs of type A and $\mathcal{P}rog$ 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 (Δ, 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 $M_i \in \mathcal{P}rog(A_i)$. If $\mathcal{R}$ is a relation on terms M with $fv(M) = \emptyset$, its *open extension*, $\mathcal{R}^\circ$, is the least relation between proved expressions such that $(\Delta, M, A)\ \mathcal{R}^\circ\ (\Delta, N, A)$ if and only if $M\{\tilde{P}/\tilde{x}\}\ \mathcal{R}\ N\{\tilde{P}/\tilde{x}\}$ for any Δ-closure $\{\tilde{P}/\tilde{x}\}$. We will write $\Delta \vdash M\ \mathcal{R}\ N : A$ to mean that $(\Delta, M, A)\ \mathcal{R}^\circ\ (\Delta, N, A)$. Sometimes we omit the type information if it is not important and simply write $\Delta \vdash M\ \mathcal{R}\ N$.

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

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

where $\mathcal{E}$ is any *evaluation context* generated by the grammar

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

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 $\mu : S \to [0, 1]$ with $\sum_{s \in S} \mu(s) \leq 1$. It is a *(full) distribution* if $\sum_{s \in S} \mu(s) = 1$. Let $\lceil \mu \rceil := \{s \in S \mid \mu(s) > 0\}$ be the *support* and $|\mu| := \sum_{s \in \lceil \mu \rceil} \mu(s)$ the *size* of subdistribution μ. 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 $(\mathcal{P}rog, \to)$, where $\mathcal{P}rog$ is the set of all closed programs and $\to \subseteq \mathcal{P}rog \times \mathcal{D}(\mathcal{P}rog)$ is the transition relation with $M \to \mu$ satisfying $\mu(N) = \sum\{\!|p_i \mid M \overset{p_i}{\rightsquigarrow} N|\!\}$. Here $\{\!|p_i \mid M \overset{p_i}{\rightsquigarrow} N|\!\}$ is a multiset if $M \equiv N$, 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 $\to$ to be a transition relation between subdistributions: $\mu \to \nu$ if $\nu = \sum_{M \in \lceil \mu \rceil} \mu(M) \cdot \mu_M$ and $M \to \mu_M$ for each M in $\lceil \mu \rceil$.

Definition 1 (Extreme derivative [13]): Suppose we have subdistributions μ, $\mu_k^{\rightarrow}$, $\mu_k^{\times}$ for $k \geq 0$ with the following properties:

$$\mu = \mu_0^{\rightarrow} + \mu_0^{\times}; \qquad \forall k \geq 0.\ \mu_k^{\rightarrow} \to \mu_{k+1}^{\rightarrow} + \mu_{k+1}^{\times};$$

and each $\mu_k^{\times}$ is stable in the sense that $M \not\to$, for all $M \in \lceil \mu_k^{\times} \rceil$. Then we call $\rho := \sum_{k=0}^{\infty} \mu_k^{\times}$ an *extreme derivative* of μ, and write $\mu \Rightarrow \rho$. The symbol $\mu_k^{\rightarrow}$ represents a subdistribution that can reduce, and $\mu_k^{\times}$ a subdistribution that cannot reduce.

Let M be a program in the Markov chain $(\mathcal{P}rog, \to)$. 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: $[\![M]\!] = \sup\{\mu \mid M \Downarrow \mu\}$, where the supremum of subdistributions are computed component-wisely.

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

$$\Omega\ :=\ \mathtt{letrec}\ f^{\mathsf{Bool} \multimap \mathsf{Bool}} x = fx\ \mathtt{in}\ f\,\mathtt{tt}\ .$$

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

$$\Omega_A\ :=\ \mathtt{letrec}\ f^{A \multimap A} x = fx\ \mathtt{in}\ f\ .$$

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

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

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

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

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

$$\frac{!\Delta, \Delta' \vdash M : A}{!\Delta, \Delta' \vdash \mathtt{in}_l\, M : A \oplus B} \qquad \frac{!\Delta, \Delta' \vdash M : B}{!\Delta, \Delta' \vdash \mathtt{in}_r\, M : A \oplus B}$$

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

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

Fig. 1. Typing Rules.

$$(\lambda x^A.M)V \overset{1}{\rightsquigarrow} M\{V/x\}$$
$$\mathtt{let}\ x^A \otimes y^B = V \otimes W\ \mathtt{in}\ N \overset{1}{\rightsquigarrow} N\{V/x, W/y\}$$
$$\mathtt{skip}; N \overset{1}{\rightsquigarrow} N$$
$$\mathtt{match}\ \mathtt{in}_l\, V\ \mathtt{with}\ (x^A : M \mid y^B : N) \overset{1}{\rightsquigarrow} M\{V/x\}$$
$$\mathtt{match}\ \mathtt{in}_r\, V\ \mathtt{with}\ (x^A : M \mid y^B : N) \overset{1}{\rightsquigarrow} N\{V/y\}$$
$$\mathtt{letrec}\ f^{A \multimap B} x = M\ \mathtt{in}\ N \overset{1}{\rightsquigarrow} N\{(\lambda x^A.\mathtt{letrec}\ f^{A \multimap B} x = M\ \mathtt{in}\ M)/f\}$$
$$M \ {}_p\oplus \ N \overset{p}{\rightsquigarrow} M$$
$$M \ {}_p\oplus \ N \overset{1-p}{\rightsquigarrow} N$$

Fig. 2. Small-Step Axioms.

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 $x : A \vdash \mathcal{C}_{x:A} : B$, 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 λ-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 $\sqsubseteq$ defined as follows: $M \sqsubseteq N$ for $M, N \in \mathcal{P}rog(A)$ if $|[\![\mathcal{C}_{x:A}\{M/x\}]\!]| \leq |[\![\mathcal{C}_{x:A}\{N/x\}]\!]|$ for all linear context $\mathcal{C}_{x:A}$. *Linear contextual equivalence* $\simeq$ is defined as the symmetrization of $\sqsubseteq$, that is, $M \simeq N$ just when $M \sqsubseteq N$ and $N \sqsubseteq M$.

C. A Probabilistic Labeled Transition System

In [17], Gordon defines explicitly a labeled transition system in order to illustrate the bisimulation technique in PCF. 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 $M \xrightarrow{a} \mu$, where M is a program, μ is a subdistribution over programs, and a is an action. If μ 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 `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 β-reduction. On the contrary, the action of $\mathtt{in}_l\, V$ 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 $eval$. 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 [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.

$$\text{(BR1)}\ \frac{}{M \Downarrow \varepsilon} \qquad \text{(BR2)}\ \frac{}{V \Downarrow \overline{V}} \qquad \text{(BR3)}\ \frac{M\{V/x\} \Downarrow \mu}{(\lambda x^A.M)V \Downarrow \mu}$$

$$\text{(BR4)}\ \frac{M_1 \Downarrow \sum_{k\in K} p_k \cdot \overline{W_k} \qquad M_2 \Downarrow \sum_{j\in J} p_j \cdot \overline{V_j} \qquad \{W_k V_j \Downarrow \rho_{kj}\}_{k\in K, j\in J}}{M_1 M_2 \Downarrow \sum_{k\in K, j\in J} p_k p_j \rho_{kj}}$$

$$\text{(BR5)}\ \frac{M_1 \Downarrow p \cdot \overline{\mathtt{skip}} \quad M_2 \Downarrow \mu}{M_1; M_2 \Downarrow p \cdot \mu} \qquad \text{(BR6)}\ \frac{M_1 \Downarrow \sum_{k\in K} p_k \cdot \overline{V_k} \quad M_2 \Downarrow \sum_{j\in J} p_j \cdot \overline{W_j}}{M_1 \otimes M_2 \Downarrow \sum_{k\in K}\sum_{j\in J} p_k p_j \cdot \overline{V_k \otimes W_j}}$$

$$\text{(BR7l)}\ \frac{M \Downarrow \mu}{\mathtt{in}_l\, M \Downarrow \mathtt{in}_l\, \mu} \qquad \text{(BR7r)}\ \frac{M \Downarrow \mu}{\mathtt{in}_r\, M \Downarrow \mathtt{in}_r\, \mu} \qquad \text{(BR8)}\ \frac{M_1 \Downarrow \mu_1 \quad M_2 \Downarrow \mu_2}{(M_1\ {}_p\!\oplus\ M_2) \Downarrow p \cdot \mu_1 + (1-p) \cdot \mu_2}$$

$$\text{(BR9)}\ \frac{M_1 \Downarrow \sum_{k\in K} p_k \cdot \overline{V_k \otimes W_k} \quad \{(M_2\{V_k/x, W_k/y\}) \Downarrow \mu_k\}_{k\in K}}{(\mathtt{let}\ x^A \otimes y^B = M_1\ \mathtt{in}\ M_2) \Downarrow \sum_{k\in K} p_k \cdot \mu_k}$$

$$\text{(BR10)}\ \frac{M_1 \Downarrow \sum_{k\in K} p_k \cdot \overline{V_k} \quad \{M_2\{W_k/x\} \Downarrow \rho_k\}_{V_k = \mathtt{in}_l\, W_k} \quad \{M_3\{W_k/y\} \Downarrow \xi_k\}_{V_k = \mathtt{in}_r\, W_k}}{(\mathtt{match}\ M_1\ \mathtt{with}\ (x^A : M_2 \mid y^B : M_3)) \Downarrow \sum_{V_k = \mathtt{in}_l\, W_k} p_k \rho_k + \sum_{V_k = \mathtt{in}_r\, W_k} p_k \xi_k}$$

$$\text{(BR11)}\ \frac{(N\{(\lambda x^A.\mathtt{letrec}\ f^{A\multimap B} x = M\ \mathtt{in}\ M)/f\}) \Downarrow \mu}{(\mathtt{letrec}\ f^{A\multimap B} x = M\ \mathtt{in}\ N) \Downarrow \mu}$$

Fig. 3. Big-Step Semantics Rules.

$$\frac{}{\mathtt{skip} \xrightarrow{\mathtt{skip}} \varepsilon} \qquad \frac{\emptyset \vdash \lambda x^A.M : A \multimap B \quad \emptyset \vdash V : A}{\lambda x^A.M \xrightarrow{@V} [\![M\{V/x\}]\!]}$$

$$\frac{\emptyset \vdash \mathtt{in}_l\, V : A \oplus B \quad x : A \vdash M : C}{\mathtt{in}_l\, V \xrightarrow{\mathtt{l}\, M} [\![M\{V/x\}]\!]}$$

$$\frac{\emptyset \vdash \mathtt{in}_r\, V : A \oplus B \quad y : B \vdash M : C}{\mathtt{in}_r\, V \xrightarrow{\mathtt{r}\, M} [\![M\{V/y\}]\!]}$$

$$\frac{\emptyset \vdash V \otimes W : A \otimes B \qquad x : A, y : B \vdash M : C}{V \otimes W \xrightarrow{\otimes M} [\![M\{V/x, W/y\}]\!]}$$

$$\frac{}{M \xrightarrow{eval} [\![M]\!]}$$

Fig. 4. Labeled Transition Rules.

The set of programs $\mathcal{P}rog$ together with the transition rules in Figure 4 yields a probabilistic labeled transition system (pLTS). It is in fact a reactive system [33] in the sense that if $M \xrightarrow{a} \mu_1$ and $M \xrightarrow{a} \mu_2$ then $\mu_1 = \mu_2$ for all $M \in \mathcal{P}rog$, that is, no two outgoing transitions leaving a program are labeled by the same action.

Below we recall Larsen and Skou's probabilistic bisimulation [23]. We first review a way of lifting a binary relation $\mathcal{R}$ over a set S to be a relation $\mathcal{R}^\dagger$ over the set of subdistributions on S given in [15][1].

Definition 4: Let S, T be two countable sets and $\mathcal{R} \subseteq S \times T$ be a binary relation. For any $X \subseteq S$, we write $\mathcal{R}(X)$ for the set $\{t \in T \mid \exists s \in X.\ s\ \mathcal{R}\ t\}$ and $\mu(X)$ for the accumulation probability $\sum_{s\in X} \mu(s)$. The lifted relation $\mathcal{R}^\dagger \subseteq \mathcal{D}(S) \times \mathcal{D}(T)$ is defined by letting $\mu\ \mathcal{R}^\dagger\ \nu$ iff $\mu(X) \leq \nu(\mathcal{R}(X))$ for all $X \subseteq S$.

Definition 5: A *state-based simulation* is a preorder $\mathcal{R}$ on programs such that whenever $M\ \mathcal{R}\ N$ we have that

(i) $[\![M]\!]\ \mathcal{R}^\dagger\ [\![N]\!]$ and
(ii) if M, N are values then $M \xrightarrow{a} \mu$ implies $N \xrightarrow{a} \nu$ with $\mu\ \mathcal{R}^\dagger\ \nu$.

A *state-based bisimulation* is a relation $\mathcal{R}$ such that both $\mathcal{R}$ and $\mathcal{R}^{-1}$ are state-based simulations. Let $\preceq_s$ and $\sim_s$ be the largest state-based simulation and bisimulation, called state-based similarity and bisimilarity, respectively. They are relations on programs, but can be generalized to open terms.

For reactive pLTSs, the kernel of probabilistic similarity is probabilistic bisimilarity. That is, $\sim_s = \preceq_s \cap \preceq_s^{-1}$. This of course also applies to the specific pLTS we are working with.

[1]There are several different but equivalent formulations of the lifting operation; see e.g. [29], [11] for more detailed discussion.

The probabilistic bisimilarity defined above is a binary relation between states. Alternatively, it is possible to directly define a (sub)distribution-based bisimilarity by comparing actions emitted from subdistributions. In order to do so, we first define a transition relation between subdistributions.

Definition 6: We write $\mu \xrightarrow{a} \rho$ if $\rho = \sum_{s \in \lceil \mu \rceil} \mu(s) \cdot \mu_s$, where μ_s is determined as follows:

- either $s \xrightarrow{a} \mu_s$
- or there is no ν with $s \xrightarrow{a} \nu$, and in this case we set $\mu_s = \varepsilon$.

Note that this is a weaker notion of transition relation between subdistributions, compared with that defined on Page 3. If $\mu \xrightarrow{a} \mu'$ then some (not necessarily all) states in the support of μ can perform action a. Let μ be a subdistribution over a reactive pLTS. After performing any action, it can reach a unique subdistribution. That is, if $\mu \xrightarrow{a} \nu$ and $\mu \xrightarrow{a} \rho$ then $\nu = \rho$.

In [16], several notions of distribution-based bisimulations are compared. As we will see, it is the weakest one that precisely captures the linear contextual equivalence of probabilistic programs.

Definition 7: A *distribution-based simulation* is a binary relation $\mathcal{R}$ on subdistributions such that $\mu \,\mathcal{R}\, \nu$ implies

(i) $|\mu| \leq |\nu|$;
(ii) if $\mu \xrightarrow{a} \rho$ then $\nu \xrightarrow{a} \xi$ for some ξ with $\rho \,\mathcal{R}\, \xi$.

If both $\mathcal{R}$ and $\mathcal{R}^{-1}$ are distribution-based simulations, then $\mathcal{R}$ is a *distribution-based bisimulation*. Let $\preceq_d$ and $\sim_d$ be the largest distribution-based simulation and bisimulation, respectively. Suppose $\emptyset \vdash M, N : A$. We write $\emptyset \vdash M \preceq_d N : A$ if $[\![M]\!] \preceq_d [\![N]\!]$; similarly for the symbol $\sim_d$.

The first clause of Definition 7 says that if μ is simulated by ν then the size of the former should not exceed that of the latter. The intuition underline this requirement is that divergent terms exhibit less observable behavior than convergent terms. It is not difficult to see that, for any two programs M and N, if $M \sim_s N$ then $M \sim_d N$, but the converse does not hold in general.

Example 2: Consider the following two terms:

$$\begin{array}{rcl} M & := & \lambda x^1.(\mathtt{in}_l\, x \,{}_{\frac{1}{2}}\!\oplus\, \mathtt{in}_r\, x) \\ N & := & (\lambda x^1.\mathtt{in}_l\, x) \,{}_{\frac{1}{2}}\!\oplus\, (\lambda x^1.\mathtt{in}_r\, x) \end{array}$$

We see that $M \not\sim_s N$ because to satisfy $[\![M]\!]\ (\sim_s)^\dagger\ [\![N]\!]$ would require $M \sim_s \lambda x^1.\mathtt{in}_l\, x$ and $M \sim_s \lambda x^1.\mathtt{in}_r\, x$, which is clearly impossible. However, we have $M \sim_d N$ because both $[\![M]\!]$ and $[\![N]\!]$ can perform the visible action $@\mathtt{skip}$ to reach the same distribution $\frac{1}{2}\overline{\mathtt{in}_l\ \mathtt{skip}} + \frac{1}{2}\overline{\mathtt{in}_r\ \mathtt{skip}}$.

III. Soundness of state-based bisimilarity

In this section we show that $\sim_s$ is sound for $\simeq$, which relies on the congruence property of $\sim_s$. The basic idea of the congruence proof is to make use of Howe's method [20], [26], which requires to start from an initial relation $\mathcal{R}$, define a precongruence candidate $\mathcal{R}^H$, a precongruence relation by construction, and then to show the coincidence of that relation with the initial relation. The congruence property of $\sim_d$ will be proved in Section IV.

Definition 8: Let $\mathcal{R}$ be a typed relation on terms. Its *compatible refinement*, $\hat{\mathcal{R}}$, is defined by the following rules:

1) (Comp1) $\dfrac{\Delta \vdash c : A \quad c \in \{x, \mathtt{skip}\}}{\Delta \vdash c\ \hat{\mathcal{R}}\ c}$

2) (Comp2) $\dfrac{\Delta, x : A \vdash \mathcal{C} \qquad \Delta' \vdash M\ \mathcal{R}\ N \quad \mathcal{C} \text{ is a linear context}}{\Delta, \Delta' \vdash \mathcal{C}\{M/x\}\ \hat{\mathcal{R}}\ \mathcal{C}\{N/x\}}$

A relation $\mathcal{R}$ is a *precongruence* iff it contains its own compatible refinement, that is $\hat{\mathcal{R}} \subseteq \mathcal{R}$. Let a *congruence* be an equivalence relation that is a precongruence.

Let $\mathcal{R}$ be a typed relation on terms. The typed relation $\mathcal{R}^H$ is defined by the rules in Table 5. Note that all the rules except for rule (How1) follow the same pattern as the following rule.

$$(\mathsf{HowC})\ \frac{\Delta, x : A \vdash \mathcal{C} \quad \Delta' \vdash M\ \mathcal{R}^H\ N \qquad \Delta, \Delta' \vdash \mathcal{C}\{N/x\}\ \mathcal{R}\ P \quad \mathcal{C} \text{ is a linear context}}{\Delta, \Delta' \vdash \mathcal{C}\{M/x\}\ \mathcal{R}^H\ P}$$

Sometimes, we do not need to analyze each specific context $\mathcal{C}$, so we will use rule (HowC) in places of other rules except for (How1). Below is a list of several fundamental properties of the relation $\mathcal{R}^H$.

Lemma 3: Let $\mathcal{R}$ be a typed relation.

1) If $\mathcal{R}$ is reflexive then so is $\mathcal{R}^H$;
2) If $\mathcal{R}$ is reflexive then $\mathcal{R} \subseteq \mathcal{R}^H$;
3) If $\mathcal{R}$ is reflexive then $\mathcal{R}^H$ is a precongruence relation;
4) If $\mathcal{R}$ is transitive then $\mathcal{R}^H \circ \mathcal{R} \subseteq \mathcal{R}^H$.

In the sequel, we write $\preceq_s^H$ for the more accurate notation $(\preceq_s)^H$.

Lemma 4:
1) If $\Delta, x : A \vdash M \preceq_s^H N$ and $\Delta' \vdash P : A$, then $\Delta, \Delta' \vdash M\{P/x\} \preceq_s^H N\{P/x\}$.
2) Suppose $\Delta, x : A \vdash M$ and $\Delta' \vdash P \preceq_s^H Q : A$. Then $\Delta, \Delta' \vdash M\{P/x\} \preceq_s^H M\{Q/x\}$.

By Lemma 3, if $\mathcal{R}$ is reflexive then $\mathcal{R} \subseteq \mathcal{R}^H$ and $\mathcal{R}^H$ is a precongruence. Therefore, in order to show that $\mathcal{R}$ is also a precongruence, it suffices to establish $\mathcal{R}^H \subseteq \mathcal{R}$ because we then have the coincidence of $\mathcal{R}$ with $\mathcal{R}^H$. In order to show $\preceq_s^H \subseteq \preceq_s$, we need the following two technical lemmas.

Lemma 5: If $\emptyset \vdash M_1 \preceq_s^H M_2$ then $[\![M_1]\!]\ (\preceq_s^H)^\dagger\ [\![M_2]\!]$.

Proof: (*Sketch*) By Definition 4 we need to show that

$$[\![M_1]\!](X) \leq [\![M_2]\!](\preceq_s^H (X))$$

for any set of terms X. For that purpose, it suffices to prove that, for any $\delta > 0$, if $\emptyset \vdash M_1 \preceq_s^H M_2$ and $M_1 \Downarrow \mu$ then there exists some ν with $M_2 \Downarrow \nu$ and

$$\mu(X) \ < \ \nu(\preceq_s^H (X)) + \delta$$

for any set of terms X. This can be proved by induction on the derivation of $M_1 \Downarrow \mu$, nested by an induction on the derivation of $\emptyset \vdash M_1 \preceq_s^H M_2$. ∎

Lemma 6: If $\emptyset \vdash V \preceq_s^H W$ then we have that $V \xrightarrow{a} \mu$ implies $W \xrightarrow{a} \nu$ and $\mu\ (\preceq_s^H)^\dagger\ \nu$.

$$(\mathsf{How1})\ \frac{\Delta \vdash c\ \mathcal{R}\ M \quad c \in \{x, \mathtt{skip}\}}{\Delta \vdash c\ \mathcal{R}^H\ M} \qquad (\mathsf{How2})\ \frac{\Delta, x : A \vdash M\ \mathcal{R}^H\ N \quad \Delta \vdash \lambda x^A.N\ \ \mathcal{R}\ \ P}{\Delta \vdash \lambda x^A.M\ \mathcal{R}^H\ P}$$

$$(\mathsf{How3l})\ \frac{!\Delta, \Delta' \vdash M\ \mathcal{R}^H\ N \quad !\Delta, \Delta'' \vdash P \quad !\Delta, \Delta', \Delta'' \vdash NP\ \ \mathcal{R}\ \ Q}{!\Delta, \Delta', \Delta'' \vdash MP\ \mathcal{R}^H\ Q}$$

$$(\mathsf{How3r})\ \frac{!\Delta, \Delta' \vdash M\ \mathcal{R}^H\ N \quad !\Delta, \Delta'' \vdash P \quad !\Delta, \Delta', \Delta'' \vdash PN\ \ \mathcal{R}\ \ Q}{!\Delta, \Delta', \Delta'' \vdash PM\ \mathcal{R}^H\ Q}$$

$$(\mathsf{How4l})\ \frac{!\Delta, \Delta' \vdash M\ \mathcal{R}^H\ N \quad !\Delta, \Delta'' \vdash P \quad !\Delta, \Delta', \Delta'' \vdash (N\ {}_p\!\oplus\ P)\ \ \mathcal{R}\ \ Q}{!\Delta, \Delta', \Delta'' \vdash (M\ {}_p\!\oplus\ P)\ \mathcal{R}^H\ Q}$$

$$(\mathsf{How4r})\ \frac{!\Delta, \Delta' \vdash M\ \mathcal{R}^H\ N \quad !\Delta, \Delta'' \vdash P \quad !\Delta, \Delta', \Delta'' \vdash (P\ {}_p\!\oplus\ N)\ \ \mathcal{R}\ \ Q}{!\Delta, \Delta', \Delta'' \vdash (P\ {}_p\!\oplus\ M)\ \mathcal{R}^H\ Q}$$

$$(\mathsf{How5l})\ \frac{\begin{array}{l}!\Delta, f :!(A \multimap B), x : A \vdash M\ \mathcal{R}^H\ N \quad !\Delta, \Delta', f :!(A \multimap B) \vdash P \\ !\Delta, \Delta' \vdash (\mathtt{letrec}\ f^{A \multimap B} x = N\ \mathtt{in}\ P)\ \mathcal{R}\ Q\end{array}}{!\Delta, \Delta' \vdash (\mathtt{letrec}\ f^{A \multimap B} x = M\ \mathtt{in}\ P)\ \mathcal{R}^H\ Q}$$

$$(\mathsf{How5r})\ \frac{\begin{array}{l}!\Delta, f :!(A \multimap B), x : A \vdash M \quad !\Delta, \Delta', f :!(A \multimap B) \vdash L\ \mathcal{R}^H\ P \\ !\Delta, \Delta' \vdash (\mathtt{letrec}\ f^{A \multimap B} x = M\ \mathtt{in}\ P)\ \mathcal{R}\ Q\end{array}}{!\Delta, \Delta' \vdash (\mathtt{letrec}\ f^{A \multimap B} x = M\ \mathtt{in}\ L)\ \mathcal{R}^H\ Q}$$

Fig. 5. Howe's Construction Rules — Selection.

Consequently, we can establish the coincidence of $\preceq_s$ with $\preceq_s^H$, from which it is easy to show that $\sim_s$ is a congruence.

Theorem 1: $\preceq_s$ is a precongruence included in $\sqsubseteq$ and $\sim_s$ is a congruence included in $\simeq$.

IV. Full abstraction of distribution-based bisimilarity

In order to show that $\sim_d$ is sound for $\simeq$, we would also like to use Howe's method. But it seems that we have to define Howe's construction rules for subdistributions, which would be very complicated. As a matter of fact, there is an indirect solution to avoid this by still using the rules in Figure 5. The idea is to represent a subdistribution μ over terms as just term M in the sense that $\mu = [\![M]\!]$. In fact, we only need to consider such kind of subdistributions thanks to the following proposition.

Proposition 1:
1) If $M \xrightarrow{a} \mu$ then there exists a term M_μ such that $\mu = [\![M_\mu]\!]$.
2) If $[\![M]\!] \xrightarrow{a} \mu$, then there exists a linear context $\mathcal{C}_z^a$ such that $\mu = [\![\mathcal{C}_z^a\{M/z\}]\!]$. Furthermore, $\mathcal{C}_z^a$ depends on M only through the type; that is, for any term N with the same type of M, we have $[\![N]\!] \xrightarrow{a} [\![\mathcal{C}_z^a\{N/z\}]\!]$.

Let us write $\preceq_d^H$ for Howe's extension $(\preceq_d)^H$ of $\preceq_d$. In analogy to Lemma 5, we have the next property.

Lemma 7: If $\emptyset \vdash M \preceq_d^H N$ then $|[\![M]\!]| \leq |[\![N]\!]|$.

The following property about the matching of transitions is analogous to Lemma 6.

Lemma 8: Let V be a value and P be some term. If $\emptyset \vdash V \preceq_d^H P$ and $[\![V]\!] \xrightarrow{a} \mu$ for some subdistribution μ, then there exist terms M and N such that $\mu = [\![M]\!]$, $[\![P]\!] \xrightarrow{a} [\![N]\!]$ and $M \preceq_d^H N$.

Lemma 8 can be generalized so as to allow V to be an ordinary term that is not necessarily a value.

Lemma 9: Let M and N be two terms. If $\emptyset \vdash M \preceq_d^H N$ and $[\![M]\!] \xrightarrow{a} \mu$ for some subdistribution μ, then there exist terms M' and N' such that $\mu = [\![M']\!]$, $[\![N]\!] \xrightarrow{a} [\![N']\!]$ and $M' \preceq_d^H N'$.

Proof: Suppose $\emptyset \vdash M \preceq_d^H N$ and $[\![M]\!] \xrightarrow{a} \mu$ for some subdistribution μ. Note that M and N have the same type. By Proposition 1(2), there exists a linear context $\mathcal{C}_z^a$ such that $\mu = [\![\mathcal{C}_z^a\{M/z\}]\!]$ and for term N there is a similar transition $[\![N]\!] \xrightarrow{a} [\![\mathcal{C}_z^a\{N/z\}]\!]$. By using the rule (HowC), we derive that $\mathcal{C}_z^a\{M/z\} \preceq_d^H \mathcal{C}_z^a\{N/z\}$. ∎

From Lemmas 7 and 9, we can infer that $\preceq_d^H$ is a distribution-based simulation, i.e. $\preceq_d^H \subseteq \preceq_d$. On the other hand, by the reflexivity of $\preceq_d$ and Lemma 3(2), we see that $\preceq_d \subseteq \preceq_d^H$. Therefore, we obtain the coincidence of $\preceq_d$ with $\preceq_d^H$, and then the following theorem easily follows.

Theorem 2: As a relation on terms, $\preceq_d$ is a precongruence included in $\sqsubseteq$ and $\sim_d$ is a congruence included in $\simeq$.

By exploiting Proposition 1(2) again, we can show that distribution-based bisimilarity is complete for linear contextual equivalence, thus we obtain the full abstraction result.

Theorem 3 (Full Abstraction): As relations on terms, $\simeq$ coincides with $\sim_d$.

Proof: The soundness property is stated in Theorem 2. For the completeness, it suffices to show that the relation

$$\mathcal{R} \ = \ \{([\![\mathcal{C}_x\{M/x\}]\!], [\![\mathcal{C}_x\{N/x\}]\!]) \mid M \simeq N \text{ and } \mathcal{C}_x \text{ a linear context}\}$$

is a distribution-based bisimulation. Consider any pair $(\llbracket \mathcal{C}_x\{M/x\} \rrbracket, \llbracket \mathcal{C}_x\{N/x\} \rrbracket) \in \mathcal{R}$. Since $M \simeq N$, it is easy to see that $\mathcal{C}_x\{M/x\} \simeq \mathcal{C}_x\{N/x\}$, and thus we infer that $|\llbracket \mathcal{C}_x\{M/x\} \rrbracket| = |\llbracket \mathcal{C}_x\{N/x\} \rrbracket|$.

Now suppose $\llbracket \mathcal{C}_x\{M/x\} \rrbracket \xrightarrow{a} \mu$. By Proposition 1(2) there exists a linear context $\mathcal{C}_y^a$ such that $\mu = \llbracket \mathcal{C}_y^a\{\mathcal{C}_x\{M/x\}/y\} \rrbracket$ and for the term $\mathcal{C}_x\{N/x\}$ there is a similar transition $\llbracket \mathcal{C}_x\{N/x\} \rrbracket \xrightarrow{a} \nu$ with $\nu = \llbracket \mathcal{C}_y^a\{\mathcal{C}_x\{N/x\}/y\} \rrbracket$. Furthermore, it is easy to check that $(\mu, \nu) \in \mathcal{R}$ as desired. ■

In the above proof of the completeness of distribution-based bisimilarity with respect to linear contextual equivalence, we directly construct linear contexts from transitions, which differs from [12] by avoiding the intermediate step of characterizing the bisimilarity as a testing equivalence.

V. Concluding Remarks

We have presented two notions of bisimilarity for reasoning about the equivalence of higher-order probabilistic programs in linear contexts, based on an appropriate labeled transition system for specifying the operational behaviour of programs. Both bisimilarities are sound with respect to the linear contextual equivalence, but only the distribution-based one turns out to be complete. This implies, together with the coincidence of linear contextual equivalence with trace equivalence [4], that the distribution-based bisimilarity captures trace equivalence.

Recall that in the nonprobabilistic setting the classical bisimilarity defined on a deterministic labeled transition system coincides with trace equivalence. The model of labeled Markov chains is the probabilistic analogue of deterministic transition systems. We have seen that there is a clear distinction between state-based and distribution-based bisimilarity; the former takes the probabilistic branching structure of each state into account while the latter does not and only observes the overall effect caused by each transition of a distribution, which results in a relation as coarse as trace equivalence. In some applications, trace equivalence is very helpful. For example, it can be used to describe the important concept of computational indistinguishability in complexity-theoretic cryptography [4]. The coinductive proof technique offerred by distribution-based bisimilarity could serve as a useful reasoning tool there.

Acknowledgment We thank the anonymous referees for the helpful comments. Deng would like to acknowledge the support of the National Natural Science Foundation of China (Grant No. 61672229, 61261130589), Shanghai Municipal Natural Science Foundation (16ZR1409100), and ANR 12IS02001 PACE.

References

[1] S. Abramsky. The lazy lambda calculus. In *Research Topics in Functional Programming*, pages 65–116. Addison-Wesley, 1990.

[2] G. Bierman. Program equivalence in a linear functional language. *Journal of Functional Programming*, 10(2):167–190, 2000.

[3] M. Bundgaard, T. Hildebrandt, and J. Godskesen. Typing linear and non-linear higher-order mobile embedded resources with local names. Technical Report TR-2007-97, IT University of Copenhagen, 2007.

[4] A. Cappai and U. D. Lago. On equivalences, metrics, and polynomial time. In *Proc. FCT'15*, volume 9210 of *LNCS*, pages 311–323. Springer, 2015.

[5] I. Cervesato and F. Pfenning. A linear logical framework. *Information and Computation*, 179(1):19–75, 2002.

[6] R. Crubillé and U. Dal Lago. On probabilistic applicative bisimulation and call-by-value λ-calculi. In *ESOP'14*, volume 8410 of *LNCS*, pages 209–228. Springer, 2014.

[7] R. Crubillé and U. Dal Lago. On probabilistic applicative bisimulation and call-by-value λ-calculi (long version). *CoRR*, abs/1401.3766, 2014.

[8] R. Crubillé and U. D. Lago. Metric reasoning about λ-terms: The affine case. In *Proc. LICS'15*, pages 633–644. IEEE Computer Society, 2015.

[9] R. Crubillé, U. D. Lago, D. Sangiorgi, and V. Vignudelli. On applicative similarity, sequentiality, and full abstraction. In *Proc. Correct System Design 2015*, volume 9360 of *LNCS*, pages 65–82. Springer, 2015.

[10] U. Dal Lago, D. Sangiorgi, and M. Alberti. On coinductive equivalences for higher-order probabilistic functional programs. In *POPL'14*, pages 297–308. ACM, 2014.

[11] Y. Deng. *Semantics of Probabilistic Processes: An Operational Approach*. Springer, 2014.

[12] Y. Deng, Y. Feng, and U. D. Lago. On coinduction and quantum lambda calculi. In *Proc. CONCUR'15*, volume 42 of *LIPIcs*, pages 427–440, 2015.

[13] Y. Deng, R. van Glabbeek, M. Hennessy, and C. Morgan. Testing finitary probabilistic processes (extended abstract). In *CONCUR'09*, volume 5710 of *LNCS*, pages 274–288. Springer, 2009.

[14] Y. Deng and Y. Zhang. Program equivalence in linear contexts. *Theoretical Computer Science*, 585:71–90, 2015.

[15] J. Desharnais. *Labelled Markov Processes*. PhD thesis, McGill University, 1999.

[16] W. Du, Y. Deng, and D. Gebler. Behavioural pseudometrics for nondeterministic probabilistic systems. In *Proc. SETTA'16*, volume 9984 of *LNCS*, pages 67–84. Springer, 2016.

[17] A. M. Gordon. A tutorial on co-induction and functional programming. In *Glasgow Workshop on Functional Programming*, pages 78–95. Springer, 1995.

[18] M. Hofmann. A type system for bounded space and functional in-place update. *Nord. J. Comput.*, 7(4):258–289, 2000.

[19] M. Hofmann. Linear types and non-size increasing polynomial time computation. *Information and Computation*, 183(1):57–85, 2003.

[20] D. J. Howe. Proving congruence of bisimulation in functional programming languages. *Information and Computation*, 124(2):103–112, 1996.

[21] C. Jones. *Probabilistic nondeterminism*. PhD thesis, University of Edinburgh, 1990.

[22] N. Kobayashi, B. Pierce, and D. Turne. Linearity and the pi-calculus. *ACM Transactions on Programming Languages and Systems*, 21(5):914–947, 1999.

[23] K. G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, 1991.

[24] J. H. Morris. *Lambda Calculus Models of Programming Languages*. PhD thesis, MIT, 1969.

[25] M. Pagani, P. Selinger, and B. Valiron. Applying quantitative semantics to higher-order quantum computing. In *POPL'14*, pages 647–658. ACM, 2014.

[26] A. M. Pitts. Howe's method for higher-order languages. In *Advanced Topics in Bisimulation and Coinduction*, pages 197–232. Cambridge University Press, 2011.

[27] G. D. Plotkin. Lambda definability in the full type hierarchy. In *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*. Academic Press, 1980.

[28] A. Rioli. *Coinductive techniques on a linear quantum λ-calculus*. PhD thesis, University of Bologna, 2016.

[29] J. Sack and L. Zhang. A general framework for probabilistic characterizing formulae. In *VMCAI'12*, volume 7148 of *LNCS*, pages 396–411. Springer, 2012.

[30] D. Sangiorgi and V. Vignudelli. Environmental bisimulations for probabilistic higher-order languages. In *POPL'16*, pages 595–607. ACM, 2016.

[31] R. Statman. Logical relations and the typed lambda-calculus. *Information and Control*, 65(2/3):85–97, 1985.

[32] F. van Breugel, M. W. Mislove, J. Ouaknine, and J. Worrell. Domain theory, testing and simulation for labelled Markov processes. *Theoretical Computer Science*, 333(1-2):171–197, 2005.

[33] R. J. van Glabbeek, S. A. Smolka, and B. Steffen. Reactive, generative and stratified models of probabilistic processes. *Information and Computation*, 121(1):59–80, 1995.

[34] D. Waler and K. Watkins. On regions and linear types. In *Proc. ICFP'01*. ACM, 2001.
[35] N. Yoshida, K. Honda, and M. Berger. Linearity and bisimulation. *Journal of Logic and Algebraic Programming*, 72(2):207–238, 2007.