Probabilistic λ-calculus and Quantitative Program Analysis

Alessandra Di Pierro Chris Hankin Herbert Wiklicky

May 31, 2004

Abstract

We show how the framework of probabilistic abstract interpretation can be applied to statically analyse a probabilistic version of the λ-calculus. The resulting analysis allows for a more speculative use of its outcomes based on the consideration of statistically defined quantities. After introducing a linear operator based semantics for our probabilistic λ-calculus Λ_P, and reviewing the framework of abstract interpretation and strictness analysis, we demonstrate our technique by constructing a probabilistic (first-order) strictness analysis for Λ_P.

1 Introduction

In this paper we aim to show how probabilistic abstract interpretation [12, 13] can be used to analyse terms in a probabilistic λ-calculus. Our running example will be a simple strictness analysis [17, 2]. This analysis has been used in the non-probabilistic setting to optimise lazy functional languages by allowing lazy evaluation to be replaced by eager evaluation without compromising the semantics. We suggest that, in the probabilistic setting, strictness analysis might be used to perform a more speculative optimisation which replaces lazy by eager evaluation as long as the risk of introducing non-termination is sufficiently low.

In order to illustrate how quantitative elements change classical analysis, we will present an example borrowed from the theory of stochastic processes (see Example 2.1 of [6]), which is related to economics and in particular to risk management. Consider the situation in which a company starts with initial capital of $\mathbf{Cap}_0$, at each time step its income is $\mathbf{In}_i$ and its outlay to meet claims is $\mathbf{Out}_i$; the sequence of incomes and outlays are modelled by mutually independent and identically distributed variables. The fortunes of the company are modelled by a simple random walk with an absorbing barrier at zero and jumps $\mathbf{Step}_n = \mathbf{In}_n - \mathbf{Out}_n$:

$$\mathbf{Cap}_n = \begin{cases} \mathbf{Cap}_{n-1} + \mathbf{Step}_n & \text{if } \mathbf{Cap}_{n-1} > 0 \text{ and } \mathbf{Cap}_{n-1} + \mathbf{Step}_n > 0 \\ 0 & \text{otherwise} \end{cases}$$

Qualitatively, we can analyse the random walk and just conclude that $\mathbf{Cap}$ ranges over the interval $[0, \infty)$; quantitatively, we can ask the more interesting question: *What is the probability of bankruptcy for a given statistical behaviour of claims and income?* Obviously, one can ask similar questions also with respect to computational processes which in one way or another use limited computational resources.

The probabilistic version of the λ-calculus we introduce is the base for the definition of a quantitative approach to a semantics based program analysis. We will define a probabilistic semantics for the λ-calculus which reflects the spirit of the example above by associating to a λ-program a linear operator specifying the dynamic of a Markov chain (a generalisation of a random walk). We will then show how this linear operator semantics can be used to apply probabilistic abstract interpretation techniques and obtain a quantitative version of the classical strictness analysis of the λ-calculus.

The rest of this paper is organised as follows. We start by introducing the probabilistic λ-calculus and its semantics. In the next Section we review the main features of classical abstract interpretation and show how the framework may be applied to produce a strictness analysis for a first-order fragment of an applied λ-calculus. The paper [2] shows how these ideas can be extended to the higher-order case. We then describe probabilistic abstract interpretation [12, 13]. The final main section returns to the problem of strictness analysis in the probabilistic λ-calculus.

2 Probabilistic λ-calculus

We introduce a probabilistic version of the untyped λ-calculus, that is the formal theory of functions where each expression can be considered as both a function and a function argument. This theory can be seen as the simplest programming language with higher-order objects. An extension of this basic theory with probabilistic features is at the basis of the quantitative approach to static program analysis we will introduce in Section 5.

Other works have introduced probabilistic features into the λ-calculus, mainly motivated by the design and implementation of probabilistic languages (see [21] and [20] for recent examples). In this work, we aim to introduce a probabilistic semantics for the λ-calculus by extending the classical theory with a notion of *probabilistic term.* This corresponds essentially to a probability distribution over classical λ-terms. We recall that the set Λ of classical λ-terms, is the set of all expressions generated by the syntax:

$$M ::= x \;\Big|\; \lambda x.M \;\Big|\; MN$$

Formally, we define the class, Λ_P, of probabilistic λ-terms as follows. Let x range over a set Var of variables. Then the class of probabilistic λ-expressions (ranged over by P) conforms to the syntax:

$$P ::= x \;\Big|\; \lambda x.P \;\Big|\; PP' \;\Big|\; \bigoplus_{i=1}^{n} p_i : P_i$$

We require that for each $\bigoplus$-construct the associated p_i are real numbers in the interval $[0, 1]$, i.e. $p_i \in [0, 1]$, and that they sum up to one[1].

Danos and Harmer, [8], show that most forms of probabilistic behaviour can be encoded using a coin flip, i.e. that a binary choice — which we also denote by $P_1 \oplus_p P_2$ instead of $(1-p) : P_1 \oplus p : P_2$ — is sufficient. This minimalist programme however introduces a number of technical complications (such as distributivity in a nested binary sum) which tend to obfuscate the basic structures. We therefore opted for an n-ary probabilistic choice.

Clearly, every classical λ-term can be also seen as a probabilistic λ-term; to be precise we will write M if we consider it as a classical λ-term and $\tilde{M}$ when we treat it as a probabilistic term. This can be interpreted as the probability distribution, which we denote by $\overline{M}$, on Λ, i.e. a map $\Lambda \to [0, 1]$ which assigns 1 to term M and 0 to all the other terms.

Although the recursive definition of P makes it not immediately evident, we can give a similar interpretation to *every* probabilistic λ-term. In fact, we can define for each $P \in \Lambda_P$, its "flattened" version

$$\tilde{P} = \bigoplus_{i=1}^{n} p_i : \tilde{M}_i$$

which corresponds one-to-one to a probability distribution on Λ, i.e. a vector $\overline{P} \in \mathcal{V}(\Lambda)$:

$$\overline{P} = \sum_{i=1}^{n} p_i \cdot \overline{M_i}$$

We recall that the *vector space* $\mathcal{V}(X, \mathbb{W})$ over a set X is the space of formal (potentially infinite) linear combinations of elements in X with coefficients in some field $\mathbb{W}$ (e.g. $\mathbb{W} = \mathbb{R}$), i.e.

$$\mathcal{V}(X) = \left\{ \sum c_x \vec{x} \mid c_x \in \mathbb{W},\; x \in X \right\}.$$

Thus, we can interpret a probability distribution on Λ as a vector in $\mathcal{V}(\Lambda) = \mathcal{V}(\Lambda, \mathbb{R})$.

[1]Alternatively, one could also allow for any $p_i \in [0, \infty)$ and statically normalise all p_i to $\tilde{p}_i = p_i / \sum_j p_j$.

It will be convenient to assume the existence of a (Gödel) enumeration of all classical λ-terms, i.e. a function: $\sharp. : \Lambda \to \mathbb{N}$ which is bijective, and its inverse function $\flat. : \mathbb{N} \to \Lambda$.

Then we can denote a probability distribution over the set Λ of classical λ-terms by $\sum_i p_i \overline{\flat(i)}$ (or, alternatively as a vector of real numbers $(p_i)_i$), where $p_i \in [0,1]$ for all i and $\sum_i p_i = 1$; p_i is intended to represent the probability associated to term $\flat(i) \in \Lambda$. It will also be convenient to introduce the following operations on vector distributions on Λ:

$$\lambda x. \left(\sum_i p_i \overline{M_i} \right) = \sum_i p_i \overline{(\lambda x. M_i)}$$

and

$$\left(\sum_i p_i \overline{M_i} \right) \otimes \left(\sum_j q_j \overline{N_j} \right) = \sum_{i,j} (p_i p_j \cdot \overline{M_i N_j})$$

Definition 1 *The flattened version $\overline{P}$ of a probabilistic λ-term $P \in \Lambda_P$ is defined by:*

$$\begin{array}{rcl} P = x & \mapsto & \overline{P} = \overline{x} \\ P = \lambda x.P' & \mapsto & \overline{P} = \lambda x.\overline{P'} \\ P = P_1 P_2 & \mapsto & \overline{P} = \overline{P_1} \otimes \overline{P_2} \\ P = \bigoplus_{i=1}^{n} p_i : P_i & \mapsto & \overline{P} = \sum_{i=1}^{n} p_i \cdot \overline{P_i} \end{array}$$

Proposition 1 *For all $P \in \Lambda_P$, $\overline{P}$ is a probability distribution on Λ, i.e. $\overline{P} = \sum_i p_i \cdot \overline{M_i}$, $M_i \in \Lambda$ for all i, and $\sum_i p_i = 1$.*

Proof: By structural induction and the fact the the two operations λx and $\otimes$ transform distributions into distributions. $\square$

2.1 Probabilistic β-Reduction

The notion of *β-reduction* expresses the computational aspect of classical λ-calculus. Informally, it consists in the evaluation of the β-normal form of a term by means of the application of the β-rule

$$(\lambda x.M)N \to_\beta M[x := N].$$

If this process is identified with function application in programming languages, then β-reduction is actually the execution of λ-term programs.

For our probabilistic λ-calculus, we need a notion of *probabilistic β-reduction* which is able to take into account the different alternatives of a probabilistic choice represented by a probabilistic λ-term. We will introduce two different versions of an operational semantics for our probabilistic λ-calculus: the first one uses distributions over classical λ-terms as operational states and a non-probabilistic transition relation $\Rightarrow_\beta$ on them which can be seen as a 'lifting' of the classical β-reduction $\to_\beta$; the second one is based on a probabilistic transition relation $\to^p$ on probabilistic λ-terms defined in the SOS style, typically adopted in the semantics of programming languages.

2.1.1 Distribution-Based Semantics

Formally, we introduce the probabilistic one-step β-reduction $\Rightarrow_\beta$ for probabilistic λ-terms as an extension of the classical one-step β-reduction relation $\to_\beta$ on the classical terms Λ, which we will briefly recall in the following.

Definition 2 *The binary relation $\to_\beta$ on Λ is defined by:*

$$(\lambda x.M)N \to_\beta M[x := N];$$

$$\frac{M \to_\beta N}{MZ \to_\beta NZ}$$

$$\frac{M \to_\beta N}{\lambda x.M \to_\beta \lambda x.N}$$

Note that the the first two rules together enforce a leftmost reduction strategy making the reduction process deterministic.

Definition 3 *The binary relation $\Rightarrow_\beta$ on Λ_P is defined by:*

$$\frac{M \to_\beta N}{\overline{M} \Rightarrow_\beta \overline{N}}$$

$$\frac{M \not\to_\beta}{\overline{M} \Rightarrow_\beta \overline{M}}$$

$$\frac{\overline{M_i} \Rightarrow_\beta \overline{N_i}}{\overline{\bigoplus_i p_i : M_i} \Rightarrow_\beta \overline{\bigoplus_i p_i : N_i}}$$

Note that since β-reduction with a leftmost strategy is deterministic, there is no ambiguity in the second rule.

Classically, we say that term M β-reduces to term N if (M, N) is in the reflexive, transitive closure,$\to^*_\beta$, of $\to_\beta$. Analogously, we define the relation $\Rightarrow^*_\beta$ on Λ_P as the reflexive, transitive closure of $\Rightarrow_\beta$.

Definition 4 *The binary relation $\Rightarrow^*_\beta$ on Λ_P is defined by:*

$$P \Rightarrow^*_\beta P$$

$$\frac{P \Rightarrow_\beta P'}{P \Rightarrow^*_\beta P'}$$

$$\frac{P_1 \Rightarrow_\beta P_2, P_2 \Rightarrow_\beta P_3}{P_1 \Rightarrow^*_\beta P_3}$$

The computational idea behind the classical β-reduction is to calculate for a term M its β-*normal form* (if there any), that is a term N with no sub-expressions of the form $(\lambda x.N')N''$. In fact, if M has a β-normal form N, then this is unique and $M \to^*_\beta N$. In the case where no reduction strategy is assumed, this is a consequence of the Church-Rosser theorem [1].

The probabilistic β-reduction allows us to compute β-normal forms for classical terms together with an information on the probability of actually achieving them by β-reduction.

Definition 5 *A probabilistic λ-term P is a probabilistic β-normal form if it is a probability distribution on basic λ-terms which are β-normal forms, i.e. $\overline{P} = \sum_i p_i \cdot \overline{M_i}$ and M_i are β-normal forms for all i such that $p_i \neq 0$.*

Proposition 2 *For a probabilistic λ-term P there is at most one probabilistic term P_β such that P_β is a probabilistic β-normal form and $P \Rightarrow^*_\beta P_\beta$.*

Proof: The transition relation $\Rightarrow_\beta$ is deterministic, i.e. for every probabilistic term P with $P \Rightarrow_\beta P_1$ and $P \Rightarrow_\beta P_2$ it follows that $P_1 = P_2$. This follows from the fact that we use a deterministic reduction strategy (left-most). □

2.1.2 Term-Based Semantics

A more programming language oriented definition of the reduction of probabilistic λ-terms, is based on the definition of a probabilistic transition relation $\rightarrow^p \subseteq \Lambda_p \times [0,1] \times \Lambda_p$ on probabilistic λ terms Λ_p. This will subsume the classical non-probabilistic β-reduction $\rightarrow_\beta$.

Definition 6 *The probabilistic transition relation $\rightarrow^p \subseteq \Lambda_p \times [0,1] \times \Lambda_p$ is defined inductively by the following rules:*

$$\begin{array}{ll}
(\textbf{app}) & \dfrac{P \rightarrow^p P'}{(PQ) \rightarrow^p (P'Q)} \\[2ex]
(\beta) & (\lambda x.P)Q \rightarrow^1_\beta P[x := Q] \\[1ex]
(\delta) & (\bigoplus_i p_i : P_i) \rightarrow^{p_i}_\delta P_i
\end{array}$$

where $P, Q \in \Lambda_p$ are probabilistic terms.

We write $P_1 \rightarrow^p P_2$ to mean that P_1 reduces to P_2 with probability p in one step, i.e. by application of one of the above reduction rules. If the reduction process is finite, then it terminates with a term which is a λ-term or a constant. The **app** and β rules together enforce the leftmost reduction strategy.

Example 1 *The possible reductions of the probabilistic λ-term $P \equiv ((\lambda x.0) \oplus_{\frac{1}{2}} (\lambda x.x))(\perp \oplus_{\frac{3}{4}} 42)$ can be depicted as follows:*

$$\begin{tikzcd}
 & ((\lambda x.0) \oplus_{\frac{1}{2}} (\lambda x.x))(\perp \oplus_{\frac{3}{4}} 42) \arrow[dl, "\frac{1}{2}", "\delta"'] \arrow[dr, "\frac{1}{2}"', "\delta"] & \\
(\lambda x.0)(\perp \oplus_{\frac{3}{4}} 42) \arrow[d, "1"', "\beta"] & & (\lambda x.x)(\perp \oplus_{\frac{3}{4}} 42) \arrow[d, "\beta"', "1"] \\
0 & & \perp \oplus_{\frac{3}{4}} 42 \arrow[dl, "\frac{1}{4}", "\delta"'] \arrow[d, "\delta"', "\frac{3}{4}"] \\
 & \perp & 42
\end{tikzcd}$$

*The original term is of the form QQ', thus we have to apply the rule (**app**), i.e. reduce the first term Q first. This is of the form $P_1 \oplus_{\frac{1}{2}} P_2$, so we have to apply the (δ) rule and choose P_1 and P_2 with their respective probability $\frac{1}{2}$. After that we end up with two terms which are both of the form $(\lambda x.T)T'$; thus we can apply with probability one the (β) rule which gives us either the constant 0 or the choice $\perp \oplus_{\frac{3}{4}} 42$. The latter then reduces via the (δ) rule either to $\perp$ or 42 with the indicated probabilities.*

The transitive closure $\rightarrow^{*p}$ of $\rightarrow^p$ is defined in the usual way, but with some care for the interpretation of the label p; this is calculated by taking the product of probabilities along the paths and then summing up the probabilities along different paths which reach the same probabilistic λ-term (see e.g. [10]). As an example consider the slightly modified version of the term P in the example above, $P' \equiv ((\lambda x.0) \oplus_{\frac{1}{2}} (\lambda x.x))(\perp \oplus_{\frac{3}{4}} 0)$. By simply replacing 42 by 0 in the reduction tree for P, we see that there are two paths leading from P' to 0 with probability $\frac{1}{2} \cdot 1$ and $\frac{1}{2} \cdot 1 \cdot \frac{3}{4}$ respectively, so that we have $P' \rightarrow^{*p} 0$ with $p = \frac{1}{2} + \frac{3}{8} = \frac{7}{8}$.

While in the classical case we can identify a unique "normal form" for every λ-term (in case we do not assume a leftmost reduction strategy this is guaranteed by the Church-Rosser theorem), probabilistic λ-terms can be reduced to a number of "final terms". It is easy to see that these are always classical terms and in particular classical β-normal forms. This inspires the following definition of the semantics of a probabilistic λ-term as a distribution over classical β-normal forms, that is as a vector in $\mathcal{V}(\Lambda)$.

Definition 7 *We define the observables of a probabilistic λ-term $P \in \Lambda_p$ as:*

$$\mathcal{O}(P) = \{\langle M_i, p_i\rangle \mid P \to^{*p_i} M_i \ \wedge \ M_i \not\to_\beta\}$$

Note that the definition of $\to^{*p}$ (and in particular of the label p in this relation) guarantees that

$$\sum\{p_i \mid \langle M_i, p_i\rangle \in \mathcal{O}(P)\} = 1,$$

that is $\mathcal{O}(P) \in \mathcal{V}(\Lambda)$ is a probability distribution for every $P \in \Lambda_p$.

Example 2 *The observables of the term P in Example 1 are*

$$\mathcal{O}(P) = \{\left\langle 0, \frac{1}{2}\right\rangle, \left\langle 42, \frac{3}{8}\right\rangle, \left\langle \perp, \frac{1}{8}\right\rangle\}.$$

2.1.3 Correspondence

We show that the probabilistic observables defined via the probabilistic transition system in Section 2.1.2 correspond to the probabilistic β-normal forms that can be calculated via the probabilistic β- reduction rule introduced in Section 2.1.1

Proposition 3 *Let $P \in \Lambda_p$ be a probabilistic λ-term which admits a probabilistic β-normal form. Then we have:*

$$\mathcal{O}(P) = P_\beta$$

Proof: By induction on the depth of probabilistic choices.

We say that a probabilistic λ-term is of *choice depth* 0 if it is a classical λ-term; it is of choice depth $n+1$ when it is of the form $\bigoplus_i p_i : P_i$ where the P_i are probabilistic λ-terms of choice depth less or equal to n and at least one of the P_i's is exactly of choice depth n.

By hypothesis we have that for every classical λ-term M involved in a (probabilistic) reduction, there exists a classical λ-term N such that $M \to^*_\beta N$, that is N is the normal form of M.

Choice depth 0 or 1: Terms of the form $\bigoplus_{i=1}^n p_i : M_i$ with $n \geq 1$ and M_i classical terms are identical to their flattened form. By the third rule of Definition 3 of the relation $\Rightarrow_\beta$ and the definition of its transitive closure $\Rightarrow^*_\beta$, we see that each of the classical terms M_i reduces independently of the others to its normal form N_i. We thus obtain a distribution of the normal forms $\sum_i p_i \cdot \overline{N_i}$ using $\Rightarrow^*_\beta$. We end up with the same distribution if we (i) select a term M_i with probability p_i according to the (δ)-rule in Definition 6 and then reduce it to its (classical) normal form using the (probabilistic) β-reduction for (probabilistic) terms.

Choice depth $n > 1$: By structural induction.

$p = x$ This case cannot occur.

$P = \lambda x.P'$ Since by the inductive hypothesis $P'_\beta = \mathcal{O}(P')$ where $P'_\beta = \sum_i p_i \cdot M_i$ for some M_i in β-normal form, we have that $\overline{P} = \bigoplus_i p_i : \lambda x.M_i$. Since each M_i is already a classical normal normal form, i.e. $M_i \to^*_\beta M_i$, we have $\sum_i p_i \cdot \overline{\lambda x.M_i} \Rightarrow^*_\beta \sum_i p_i \cdot \overline{\lambda x.M_i}$. On the other hand it is obvious that we have $\mathcal{O}(P) = \{\langle P, 1\rangle\}$ as we have $\lambda x.M_i \not\to_\beta$.

$P = P_1P_2$ By the inductive hypothesis we can again assume that $P_1 = \bigoplus_i p_{1i} : M_{1i}$ and $P_2 = \bigoplus_j p_{2j} : M_{2j}$ with M_{1i} and M_{2j} β-normal forms. Again the transition relation $\to^p_\beta$ and $\Rightarrow_\beta$ result in the combination of normal forms $M_{1i}M_{2j} \to_\beta N_{ij}$; the probabilities are exactly given by $p_{1i}p_{2j}$ for reaching the normal term N_{ij}.

$P = \bigoplus_i p : P_i$**:** Without loss of generality we can assume that $P = p_1 : P_1 \oplus p_2 : P_2$, where P_1 and P_2 are probabilistic λ-terms of choice depth $n' \leq n$. By the induction hypothesis we have $P_1 = \bigoplus_i p_{1i} : M_{1i}$ and $P_2 = \bigoplus_j p_{2j} : M_{2j}$. The flattened version of P is thus of the form $\bigoplus_i p_1p_{1i}M_{1i} \oplus \bigoplus_j p_2p_{2j}M_{2j}$. Again it is obvious that $\to^p_\beta$ and $\Rightarrow_\beta$ lead to the same linear combination of normal terms.

□

2.2 Linear Representation

In this section we introduce a linear operator on the vector space $\mathcal{V}(\Lambda)$ as a representation of the the semantics $\mathcal{O}(P)$ for a probabilistic λ-term P which is more suitable as a base for a probabilistic analysis. For this purpose we will use the distribution-based semantics introduced in Section 2.1.1 as an intermediate step, and define a linear operator representing the one-step reduction relation $\Rightarrow_\beta$. We first define a linear operator for the β-reduction of the classical λ-calculus, i.e. for terms $M, N \in \Lambda$. This is given by:

$$\mathbf{T} : \mathcal{V}(\Lambda) \to \mathcal{V}(\Lambda)$$

with the following entries:

$$(\mathbf{T})_{MN} = \begin{cases} 1 & \text{if } M \to_\beta N \\ 1 & \text{if } M \not\to_\beta \;\wedge\; M = N \\ 0 & \text{otherwise} \end{cases}$$

and then the restriction for each classical λ-term $M \in \Lambda$:

$$\mathbf{T}_M = \pi_{\mathcal{R}(M)} \mathbf{T} \pi_{\mathcal{R}(M)},$$

where $\pi_{\mathcal{R}(M)}$ is the projection onto the reachable states $\mathcal{R}(M)$ i.e. the sub-vector space $\mathcal{V}(\mathcal{R}(M))$ of $\mathcal{V}(\Lambda)$.

For a probabilistic term $P \in \Lambda_p$ we consider its linear representation, i.e. its flattened form $\overline{P} \in \mathcal{V}(\Lambda)$:

$$\overline{P} = \sum_i p_i \cdot \overline{M_i}$$

and the restriction of $\mathbf{T}$ to the union of reachable states, i.e.

$$\mathbf{T}_P = \pi_{\bigcup_i \mathcal{R}(M_i)} \mathbf{T} \pi_{\bigcup_i \mathcal{R}(M_i)}.$$

Now we can define the semantics of P via the iteration of the operator $\mathbf{T}_P$, as follows.

Definition 8 *For a probabilistic λ-term $P \in \Lambda_p$ we define its semantics as:*

$$[\![P]\!] = \overline{P} \cdot \lim_{i \to \infty} \mathbf{T}_P^i$$

Proposition 4 *For every probabilistic term $P \in \Lambda_p$ which admits a probabilistic β-normal form we have:*

$$[\![P]\!] = \mathcal{O}(P) = P_\beta$$

Proof: Obvious from the construction of the operator $\mathbf{T}_P$ and the correspondence shown in Proposition 3. □

Example 3 *Consider again the probabilistic λ-term from Example 1:*

$$P \equiv ((\lambda x.0) \oplus_{\frac{1}{2}} (\lambda x.x))(\perp \oplus_{\frac{3}{4}} 42)$$

After the flattening procedure we get the probability distribution

$$\begin{aligned} \overline{P} &= (\frac{1}{2}\overline{\lambda x.0} + \frac{1}{2}\overline{\lambda x.x}) \otimes (\frac{1}{4}\overline{\perp} + \frac{3}{4}\overline{42}) \\ &= \frac{1}{8}\overline{(\lambda x.0)\perp} + \frac{3}{8}\overline{(\lambda x.0)42} + \frac{1}{8}\overline{(\lambda x.x)\perp} + \frac{3}{8}\overline{(\lambda x.x)42} \end{aligned}$$

An enumeration of the reachable terms is:

1	$(\lambda x.0)\perp$	**4**	$(\lambda x.x)\perp$	**7**	0
2	$(\lambda x.0)42$	**5**	$(\lambda x.x)42$	**8**	$\perp$
3	$(\lambda x.0)0$	**6**	$(\lambda x.x)0$	**9**	42.

According to this enumeration we can represent $\overline{P}$ by the vector

$$\overline{P} = \left(\begin{array}{ccccccccc} \frac{1}{8} & \frac{3}{8} & 0 & \frac{1}{8} & \frac{3}{8} & 0 & 0 & 0 & 0 \end{array} \right),$$

and we have:

$$\mathbf{T}_P = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} = \lim_{i \to \infty} \mathbf{T}_P^i$$

Thus we get:

$$[\![P]\!] = \overline{P} \cdot \mathbf{T}_P = \left(\begin{array}{ccccccccc} 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{8} & \frac{3}{8} \end{array} \right).$$

3 Classical Abstract Interpretation

Program analysis aims to determine some property of a program without running it. A classical example is *Reaching Definitions* analysis which determines, for each node in a flowchart, which definitions (assignments) reach it [18]. The results of this analysis might be used to perform a constant folding transformation of the program. Such transformations should be semantics preserving and it is therefore important that the analysis gives correct information about the program. Often the properties that we are interested in are undecidable and so correctness is replaced by some approximation notion.

We start by sketching the classical approach to semantics-based program analysis: *abstract interpretation* [4, 18]. The *semantics* of a program f identifies some set V of values and specifies how the program transforms one value v_1 to another v_2: $f \vdash v_1 \longrightarrow v_2$.

In a similar way, a *program analysis* identifies the set L of properties and specifies how a program f transforms one property l_1 to another l_2: $f \vdash l_1 \rhd l_2$.

As we have seen, every program analysis should be correct with respect to the semantics. For first-order program analyses, i.e. those that abstract properties of values, this is established by directly relating properties to values using a *correctness relation*: $R : V \times L \to \{true, false\}$. The intention is that $v \; R \; l$ formalises our claim that the value v is described by the property l.

The correctness relation is often achieved via a *Galois connection*: (V, α, γ, L) is a Galois connection between the complete lattices $(V, \sqsubseteq)$ and $(L, \sqsubseteq)$ if and only if $\alpha : V \to L$ and $\gamma : L \to V$ are monotone functions that satisfy: $\gamma \circ \alpha \sqsupseteq \lambda v.v$ and $\alpha \circ \gamma \sqsubseteq \lambda l.l$.

Having defined a suitable "set" of properties we then define suitable interpretations of program operations. The framework of abstract interpretation guarantees that the analysis will be safe as long as we use an interpretation, F_{abs}, of each language operator, F, that satisfies: $F_{\text{abs}} \sqsupseteq \alpha \circ F \circ \gamma$.

Since interesting languages involve iteration or recursion we also have to construct efficient implementations; a generic solution to this problem is the theory of widenings and narrowings [4].

3.1 Strictness Analysis

Strictness analysis [17, 2] aims to answer for some function *f*: Does $f \perp = \perp$? If the function has this property then it either uses its argument or is the bottom function. In either case, an affirmative answer would mean that arguments can be passed by value rather than using (the more costly) lazy evaluation. For illustration, we will restrict ourselves to a first-order applied λ-calculus with integers as the only data type.

We can construct a Galois connection $(\mathcal{P}_H(\mathbf{Z}_\perp), \alpha, \gamma, \mathbf{Two})$ where $\mathcal{P}_H$ is the *Hoare Powerdomain* construction on the integers $\mathbf{Z}$ and $\mathbf{Two}$ is $\{0, 1\}$ ordered by $0 \sqsubseteq 1$. The elements of the

Hoare Powerdomain in this case are just down-closed sets ordered by subset inclusion, so that every set contains $\perp$.

We define:

$$\alpha(Z) = \begin{cases} 0 & \text{if } Z = \{\perp\} \\ 1 & \text{otherwise} \end{cases} \qquad \gamma(S) = \begin{cases} \{\perp\} & \text{if } S = 0 \\ \mathbf{Z}_\perp & \text{if } S = 1 \end{cases}$$

We can construct the induced operations that correspond to the operations in this first-order applied λ-calculus:

Concrete operation	Induced operation
base type constants	1
if x then y else z	$x \sqcap (y \sqcup z)$
x *op* y	$x \sqcap y$

The conditional takes three arguments (x, y, z); the predicate must be defined and then the result is at least as defined as either of the branches. Thus the abstract interpretation of

$$(\lambda\ x.\mathit{if}\ x = 0\ \mathit{then}\ 15\ \mathit{else}\ 42)$$

is $(\lambda\ x.(x \sqcap 1) \sqcap (1 \sqcup 1)) \equiv \lambda x.x$. Since $(\lambda\ x.\ x)\ 0 = 0$, this tells us that our original function is strict. We now extend this approach to a probabilistic λ-calculus by applying techniques of *Probabilistic Abstract Interpretation* [12, 13]. This will allow us to solve one important problem that despite decades of research into strictness analysis methods based on abstract interpretation has remained open, namely the precise and formal characterisation of the information loss.

4 Probabilistic Abstract Interpretation

Probabilistic Abstract Interpretation (PAI) was introduced in [12, 13] as a method for approximating the semantics of probabilistic programs which re-formulates the classical theory of Abstract Interpretation [4] in a setting suitable for a quantitative program analysis.

The PAI framework is based on the notion of *probabilistic domains* which we will identify with Hilbert spaces over some set representing the computational states. Given two probabilistic domains, $\mathcal{C}$ and $\mathcal{D}$, a *probabilistic abstract interpretation* is a pair of linear maps, $\mathbf{A} : \mathcal{C} \mapsto \mathcal{D}$ and $\mathbf{G} : \mathcal{D} \mapsto \mathcal{C}$, between the concrete domain $\mathcal{C}$ and the abstract domain $\mathcal{D}$, such that $\mathbf{G}$ is the *Moore-Penrose pseudo-inverse* of $\mathbf{A}$, and vice versa. Let $\mathcal{C}$ and $\mathcal{D}$ be two Hilbert spaces and $\mathbf{A} : \mathcal{C} \mapsto \mathcal{D}$ a bounded linear map between them. A bounded linear map $\mathbf{A}^\dagger = \mathbf{G} : \mathcal{D} \mapsto \mathcal{C}$ is the Moore-Penrose pseudo-inverse of $\mathbf{A}$ iff

$$\mathbf{A} \circ \mathbf{G} = \mathbf{P}_A \quad \text{and} \quad \mathbf{G} \circ \mathbf{A} = \mathbf{P}_G$$

where $\mathbf{P}_A$ and $\mathbf{P}_G$ denote orthogonal projections onto the ranges of $\mathbf{A}$ and $\mathbf{G}$.

Alternatively, if $\mathbf{A}$ is Moore-Penrose invertible, its Moore-Penrose pseudo-inverse, $\mathbf{A}^\dagger$ satisfies the following:

$$\textbf{(i)}\ \mathbf{A}\mathbf{A}^\dagger\mathbf{A} = \mathbf{A}, \qquad \textbf{(iii)}\ (\mathbf{A}\mathbf{A}^\dagger)^* = \mathbf{A}\mathbf{A}^\dagger,$$
$$\textbf{(ii)}\ \mathbf{A}^\dagger\mathbf{A}\mathbf{A}^\dagger = \mathbf{A}^\dagger, \qquad \textbf{(iv)}\ (\mathbf{A}^\dagger\mathbf{A})^* = \mathbf{A}^\dagger\mathbf{A}.$$

where $\mathbf{M}^*$ is the adjoint of $\mathbf{M}$. It is instructive to compare these equations with the classical setting. For example, if (α, γ) is a Galois insertion: $\alpha \circ \gamma \circ \alpha = \alpha$ and $\gamma \circ \alpha \circ \gamma = \gamma$.

As in the classical framework, given a concrete semantics we can always construct a *best correct approximation* for this semantics, although the notions of correctness and optimality assume in our linear setting a different connotation as explained in the following.

If Φ is a linear operator on some vector space $\mathcal{V}$ expressing the probabilistic semantics of a concrete system, and $\mathbf{A} : \mathcal{V} \mapsto \mathcal{W}$ is a linear abstraction function from the concrete domain into an abstract domain $\mathcal{W}$, we can compute the (unique) Moore-Penrose pseudo-inverse $\mathbf{G} = \mathbf{A}^\dagger$ of $\mathbf{A}$. An abstract semantics can then be defined as the linear operator on the abstract domain $\mathcal{W}$:

$$\Psi = \mathbf{A} \circ \Phi \circ \mathbf{G} = \mathbf{G}\Phi\mathbf{A}.$$

In the case of classical abstract interpretation the abstract semantics constructed in this way (called the *induced semantics in* [5]) is guaranteed to be the *best correct approximation* of the concrete semantics, meaning that it is the most precise among all correct approximation (the relative precision being left unquantified). In the linear space based setting of PAI where the order of the classical domains is replaced by some notion of metric distance, the induced abstract semantics is the *closest* one to the concrete semantics. This "closeness" property expresses both the "safety" of the approximation and its optimality, which comes from the following properties of the Moore-Penrose pseudo-inverse. The theory of the least-square approximation [9, 7] tells us that if $\mathcal{C}$ and $\mathcal{D}$ be two finite dimensional vector spaces, $\mathbf{A} : \mathcal{C} \mapsto \mathcal{D}$ a linear map between them, and $\mathbf{A}^\dagger = \mathbf{G} : \mathcal{D} \mapsto \mathcal{C}$ its Moore-Penrose pseudo-inverse, then the vector $x_0 = y\mathbf{G}$ is the one minimising the distance between $x\mathbf{A}$ for any vector x in $\mathcal{C}$ and y, i.e.

$$\inf_{x \in \mathcal{C}} \|x\mathbf{A} - y\| = \|x_0\mathbf{A} - y\|.$$

In other words, if we consider the equation $x\mathbf{A} = y$ we can identify a (exact) solution x_* as a vector for which $\|x_*\mathbf{A} - y\| = 0$. In particular in the case that no such solution vector x_* exists we can generalise the concept of a exact solution to that of a "pseudo-solution", i.e. we can look for a x_0 such that $x_0\mathbf{A}$ is the *closest* vector to y we can construct. This closest approximation to the exact solution is now constructed using the Moore-Penrose pseudo-inverse, i.e. take $x_0 = y\mathbf{A}^\dagger$.

Returning to our program analysis setting, suppose that we have an operator Φ and a vector x. We can apply Φ to x and abstract the result giving $x\Phi\mathbf{A}$ or we can apply the abstract operator to an abstract vector giving $x\mathbf{A}\mathbf{A}^\dagger\Phi\mathbf{A}$. Ideally, we would like these to be equal. If $\mathbf{A}$ is invertible then its Moore-Penrose pseudo-inverse is identical to the inverse and we are done. In program analysis $\mathbf{A}$ is never a square matrix and thus $\mathbf{A}\mathbf{A}^\dagger$ in $x\mathbf{A}\mathbf{A}^\dagger\Phi\mathbf{A}$ will lead to some loss of precision. The Moore-Penrose pseudo-inverse is as close as possible to an inverse if the matrix is not invertible and thus for the particular choice of $\mathbf{A}$, $\mathbf{A}^\dagger\Phi\mathbf{A}$ is the best approximation of Φ that we can have. Moreover, by choosing an appropriate notion of distance we can measure this closeness to get a quantitative estimate of the information lost in the abstraction.

5 Probabilistic Strictness Analysis

In this section we will use the PAI technique previously introduced to define a probabilistic abstraction of the linear operator expressing the semantics of a probabilistic λ-term (see Section 2.2). This will allow us to construct a probabilistic strictness analysis from which we will be able to extract quantitative information such as the probability that a given reduction will not terminate.

In many cases, and particularly in strictness analysis, the abstraction is a surjective function. An alternative view of abstraction in this case is that it maps concrete values to equivalence classes. Equivalence relations can be represented by a particular kind of operators, namely classification operators.

We call an $n \times m$-matrix $\mathbf{K}$ a *classification* matrix if it is a 0/1-matrix, where every row has exactly one non-zero entry and columns have at least one non-zero entry. Classification matrices are thus particular kinds of stochastic matrices. We denote by $\mathcal{K}(n, m)$ the set of all $n \times m$-classification matrices ($m \leq n$). Let $X = \{x_1, \ldots, x_n\}$ be a finite set. Then for each equivalence relation $\approx$ on X with $|X/_\approx| = m$, there exists a classification matrix $\mathbf{K} \in \mathcal{K}(n, m)$ and vice versa. Each column in the classification matrix represents a (non-empty) equivalence class.

The pseudo-inverse of a classification matrix $\mathbf{K} \in \mathcal{K}(n, m)$ corresponds to its normalised transpose or adjoint $\mathbf{K}^\dagger = \mathcal{N}(\mathbf{K}^T)$ (these coincide for real $\mathbf{K}$), where the *normalisation* operation $\mathcal{N}$ is defined for a matrix $\mathbf{A}$ by:

$$\mathcal{N}(\mathbf{A})_{ij} = \begin{cases} \frac{\mathbf{A}_{ij}}{a_i} & \text{if } a_i = \sum_j \mathbf{A}_{ij} \neq 0 \\ 0 & \text{otherwise.} \end{cases}$$

A suitable abstraction for probabilistic strictness analysis classifies terms as undefined, don't know or defined. We abstract every term in the enumeration to one of these values. The don't know value is used to classify terms whose "definedness" is not yet determined. Classically 0 represents definite non-termination whilst 1 represents *possible* termination. The use of three values here allows for a more informative analysis – the defined value means *definitely* terminating. This abstraction is achieved by the classification operator corresponding to the partition of the set of terms into the three classes represented by 0, ? (unknown) and 1. We will demonstrate the method by the following examples.

Example 4 *Coming back to our running Example 1:*

$$P \equiv ((\lambda x.0) \oplus_{\frac{1}{2}} (\lambda x.x))(\bot \oplus_{\frac{3}{4}} 42)$$

and its flattened version $P \mapsto \tilde{P}$:

$$\tilde{P} \equiv \frac{1}{8} : ((\lambda x.0)\bot) \oplus \frac{3}{8} : ((\lambda x.0)42) \oplus \frac{1}{8} : ((\lambda x.x)\bot) \oplus \frac{3}{8} : ((\lambda x.x)42)$$

Using the following abstraction operator and its pseudo-inverse

$$\mathbf{K} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad \mathbf{K}^\dagger = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} \end{pmatrix}$$

we get the abstract (induced) semantical operator $\mathbf{T}_P^\# : \mathcal{V}(\{0, ?, 1\}) \to \mathcal{V}(\{0, ?, 1\})$:

$$\mathbf{T}_P^\# = \mathbf{K}^\dagger \mathbf{T}_P \mathbf{K} = \begin{pmatrix} 1 & 0 & 0 \\ \frac{1}{6} & 0 & \frac{5}{6} \\ 0 & 0 & 1 \end{pmatrix}$$

As the flattened version $\tilde{P}$ (of this program) is constructed only of classical λ-terms which can be reduced in a single step to their normal form the abstract semantics turns out to be an exact abstraction in the sense that we have:

$$\overline{P} \cdot \mathbf{T}_P \cdot \mathbf{K} = \begin{pmatrix} \frac{1}{8} & 0 & \frac{7}{8} \end{pmatrix} = \overline{P} \cdot \mathbf{K} \cdot \mathbf{T}_P^\#$$

i.e. the abstracted concrete semantics coincides with the abstract semantics.

The last example is optimal as the abstraction does not loose any information, however in general we will encounter a loss of precision when we move from the concrete to the abstract semantics, as the following example illustrates:

Example 5 *Consider the probabilistic term*

$$\begin{aligned} P \quad \equiv \quad & \frac{4}{12} : (\lambda x.x)(\lambda x.x)(42) \ \oplus \ \frac{2}{12} : (\lambda x.x)(\bot) \ \oplus \\ & \frac{3}{12} : (\lambda x.x)(\lambda x.x)(\bot) \ \oplus \ \frac{3}{12} : (\lambda x.x)(\lambda x.0)(\bot) \end{aligned}$$

and the enumeration of the reachable(classical) λ terms:

1 $(\lambda x.x)(\lambda x.x)(42)$

2 $(\lambda x.x)(\bot)$

3 $(\lambda x.x)(\lambda x.x)(\bot)$

4 $(\lambda x.x)(\lambda x.0)(\bot)$

5 $(\lambda x.x)(42)$

6 $\bot$

7 $(\lambda x.x)(\bot)$

8 $(\lambda x.0)(\bot)$

9 42

10 0

Then we can construct the restricted one-step reduction operator for P and a suitable classification matrix for this program:

$$\mathbf{T}_P = \begin{pmatrix} 0&0&0&0&1&0&0&0&0&0\\ 0&0&0&0&0&1&0&0&0&0\\ 0&0&0&0&0&0&1&0&0&0\\ 0&0&0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&0&0&1&0\\ 0&0&0&0&0&1&0&0&0&0\\ 0&0&0&0&0&1&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&1\\ 0&0&0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&0&0&1 \end{pmatrix} \qquad \mathbf{K} = \begin{pmatrix} 0&1&0\\ 0&1&0\\ 0&1&0\\ 0&1&0\\ 0&1&0\\ 1&0&0\\ 0&1&0\\ 0&1&0\\ 0&0&1\\ 0&0&1 \end{pmatrix}.$$

We can then construct the abstract one-step reduction operator:

$$\mathbf{T}_P^{\#} = \mathbf{K}^{\dagger}\mathbf{T}_P\mathbf{K} = \begin{pmatrix} 1 & 0 & 0\\ 0.29 & 0.43 & 0.29\\ 0 & 0 & 1 \end{pmatrix}$$

The middle row and column represent the don't know value. The value in the middle row, middle column gives a bound on how much the other two values in that row might change when we iterate – in this sense, it gives a measure of the precision of the current abstract operator. Iterating this abstract operator causes the probability of a transition from don't know to don't know to decrease rapidly; for example after two iterations we have:

$$(\mathbf{T}_P^{\#})^2 = \begin{pmatrix} 1 & 0 & 0\\ 0.41 & 0.18 & 0.41\\ 0 & 0 & 1 \end{pmatrix}$$

Achieving a defined outcome becomes more and more likely. This result could be used to support the decision to speculatively evaluate the argument.

One advantage of the use of linear operators is that we can measure them. The standard way to measure the "size" of a linear operator is via an *operator norm* which in turn may have its origins in a vector *norm*:

(i) $\|\vec{x}\| \geq 0$

(ii) $\|\vec{x}\| = 0 \leftrightarrow \vec{x} = \vec{o}$

(iii) $\|\alpha\vec{x}\| = |\alpha|\|\vec{x}\|$

(iv) $\|\vec{x}+\vec{y}\| \leq \|\vec{x}\| + \|\vec{y}\|$

For example, we could use the 1-norm (sum of absolute values), euclidean norm (square root of the sum of squares of absolute values) or the supremum norm (supremum of absolute values).

Example 6 *For the example above, an accurate abstraction of the original program can be achieved by taking the limit of the iterations of the abstract operator (numerically computed as $\mathbf{S}^{100}$):*

$$\lim_{i\rightarrow\infty}(\mathbf{T}_P^{\#})^i = \begin{pmatrix} 1 & 0 & 0\\ 0.50 & 0 & 0.50\\ 0 & 0 & 1 \end{pmatrix}.$$

The limit of T_P gives instead the concrete semantics of P:

$$\lim_{i\to\infty} \mathbf{T}_P^i = \mathbf{T}_P^2 = \begin{pmatrix} 0&0&0&0&0&0&0&0&1&0\\ 0&0&0&0&0&1&0&0&0&0\\ 0&0&0&0&0&1&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&1\\ 0&0&0&0&0&0&0&0&1&0\\ 0&0&0&0&0&1&0&0&0&0\\ 0&0&0&0&0&1&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&1\\ 0&0&0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&0&0&1 \end{pmatrix}$$

from which we can directly construct the abstract semantical operator

$$\mathbf{K}^\dagger(\lim_{i\to\infty} \mathbf{T}_P^i)\mathbf{K} = \mathbf{K}^\dagger\mathbf{T}_P^2\mathbf{K} = \begin{pmatrix} 1&0&0\\ 0.43&0&0.57\\ 0&0&1 \end{pmatrix}.$$

By comparing the two operators we get $\mathbf{K}^\dagger[\![P]\!]\mathbf{K}$ *and* $\lim \mathbf{S}$ *we get*

$$\|\mathbf{K}^\dagger(\lim_{i\to\infty} \mathbf{T}_P^i)\mathbf{K} - \lim_{i\to\infty}(\mathbf{K}^\dagger\mathbf{T}_P\mathbf{K})^i\|_\infty = 0.143$$

and when we compare the abstract and concrete semantics after two iterations (when the concrete semantics converges to its limit) we have:

$$\|\mathbf{K}^\dagger\mathbf{T}_P^2\mathbf{K} - (\mathbf{T}_P^\#)^2\|_\infty = 0.367.$$

These norm differences measure the "precision" of the abstract semantics. In the first case we see that the abstract semantics $\lim_{i\to\infty}(\mathbf{T}_P^\#)^i$ *and the abstracted semantics* $\mathbf{K}^\dagger(\lim_{i\to\infty}\mathbf{T}_P^i)\mathbf{K}$ *differ by about* 14%, *i.e. the true chance of obtaining* $\perp$ *or a constant and the estimated probabilities are within* ±7%; *and indeed we see that, for example, the true chance of getting* $\perp$ *is about* 43%, *while the abstract semantics forecasts a* 50% *chance of obtaining this result.*

6 Conclusions

In this paper we introduced a probabilistic version of the λ-calculus and provided it with three different semantics. One is based on a *probabilistic* transition relation on *probabilistic* λ-terms which subsumes the classical, i.e. non-probabilistic, β-reduction; a second one is defined via a *classical* transition relation based on probability distributions on *classical* terms; and a third one is a *linear operator* semantics which specifies the computational dynamics in the form of so-called Markov chains, e.g. [19]. All the three semantics are equivalent in the sense that they assign the same meaning to a probabilistic λ-term P. This is essentially a distribution on the β-normal forms to which the classical sub-terms of P can be β-reduced. The second semantics can be seen as an intermediate step between the first one and the linear operator semantics, which is the most appropriate as a base for quantitative analysis.

We have also reviewed the classic approach to abstract interpretation and strictness analysis and shown how Di Pierro and Wiklicky's notion of probabilistic abstract interpretation [12, 13] is a natural analogue of the classical framework in a probabilistic setting. We have illustrated the approach for the probabilistic λ-calculus in the context of a simple strictness analysis. This quantitative version of strictness analysis is based on the linear operator semantics for the λ-calculus and aims in constructing an abstract semantics which is as close as possible to the abstraction of the concrete semantics. The final outcome of this analysis are quantitative estimates on the probability that a probabilistic term reduces to a constant normal form or to $\perp$.

In our formal treatment of the semantics and analysis of the probabilistic λ-calculus we assumed that the λ-terms we consider all have normal forms, i.e. essentially finite, terminating reduction sequences. This allows us to use straight forward constructions from linear algebra. In related work we have shown that in principle it is possible to accommodate also infinite reduction sequences, e.g. [11]. However, this requires a recasting of our framework in a functional analytical setting, using results and concepts from the theory of Hilbert spaces and C*-algebras (see e.g. [3, 15]), which would deserve a separate treatment.

A present shortcoming of our work is that the strictness analysis is not defined in a compositional way. However, the intermediate, distribution-based semantics can be seen as a kind of 'lifting' of the classical β-rule and it seems therefore possible to import results from the semantics of the classical α-calculus and classical strictness analysis in order to define a compositional linear semantics and to refine the simple probabilistic strictness analysis presented in this paper.

References

[1] H. P. Barendregt. The Lambda Calculus: Its Syntax and Semantics. North-Holland, 1981.

[2] G. Burn, C. Hankin and S. Abramsky. The Theory and Practice of Higher-order Strictness Analysis. *Science of Computer Programming*, 1986.

[3] J.B. Conway. *A Course in Functional Analysis*, volume 96 of *Garduate Texts in Mathematics.* Springer Verlag, New York, second edition, 1990.

[4] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixedpoints. In *Proceedings of POPL*, pages 238–252, 1977. ACM.

[5] P. Cousot and R. Cousot. Systematic Design of Program Analysis Frameworks. In *Proceedings of POPL*, pages 269–282, 1979. ACM.

[6] D. Cox and H. R. Miller. The Theory of Stochastic Processes. Chapman and Hall. 1965.

[7] Ben-Israel, A., Greville, T.: Generalised Inverses — Theory and Applications. second edn. Springer Verlag, New York — Berlin (2003)

[8] V. Danos and R. Harmer. Probabilistic Game Semantics. *ACM Transactions on Computational Logic*, 2001.

[9] F. Deutsch. *Best Approximation in Inner Product Spaces*, volume 7 of *CMS Books in Mathematics.* Springer Verlag, New York — Berlin, 2001.

[10] A. Di Pierro and H. Wiklicky. Quantitative observables and averages in Probabilistic Concurrent Constraint Programming. In K.R. Apt, T. Kakas, E. Monfroy, and F. Rossi, editors, *New Trends in Constraints*, number 1865 in Lecture Notes in Computer Science, Springer Verlag, New York — Berlin, 2000.

[11] A. Di Pierro and H. Wiklicky, *Linear structures for concurrency in probabilistic programming languages*, in: *Proceedings of MFCSIT00*, Electronic Notes in Theoretical Computer Science 40 (2001).

[12] A. Di Pierro and H. Wiklicky. Concurrent Constraint Programming: Towards Probabilistic Abstract Interpretation. In *Proceedings of PPDP'00*, 2000. ACM.

[13] A. Di Pierro and H. Wiklicky. Measuring the precision of abstract interpretations. In *Proceedings of LOPSTR'00*, LNCS 2042, 2001. Springer Verlag.

[14] W.H. Greub. *Linear Algebra*, volume 97 of *Grundlehren der mathematischen Wissenschaften.* Springer Verlag, New York, third edition, 1967.

[15] R.V. Kadison and J.R. Ringrose. *Fundamentals of the Theory of Operator Algebras: Volume I & II*, volume 15 of *Graduate Studies in Mathematics.* AMS, Providence, Rhode Island, 1997.

[16] P. Malacaria and L. Regnier. Some results on the Interpretation of λ-calculus in Operator Algebra. In *Proceedings of LICS*, pages 63–72, 1991, IEEE Press.

[17] A. Mycroft. Abstract Interpretation and Optimising Transformations for Applicative Programs. Ph.D. Thesis, University of Edinburgh, 1981.

[18] F. Nielson, H. Riis Nielson and C. Hankin. *Principles of Program Analysis.* Springer Verlag, 1999.

[19] J.R. Norris. *Markov Chains. Cambidge Series in Statistical and Probabilistic Mathematics.* Cambridge University Press, Cambridge, 1997.

[20] Sungwoo Park. *A calculus for probabilistic languages.* ACM SIGPLAN Notices, 37(9), pages 206–217, 2002. ACM.

[21] N. Ramsey and A. Pfeffer. *Stochastic lambda calculus and monads of probability distributions.* In *Proceedings of POPL*, pages 154–165, 2002. ACM.