Available online at www.sciencedirect.com

SCIENCE DIRECT®

Electronic Notes in Theoretical Computer Science

Electronic Notes in Theoretical Computer Science 112 (2005) 5–18

www.elsevier.com/locate/entcs

λ-calculus and Quantitative Program Analysis (Extended Abstract)

Chris Hankin

Department of Computing
Imperial College London
London, UK

Herbert Wiklicky

Department of Computing
Imperial College London
London, UK

Abstract

In this paper we show how the framework of probabilistic abstract interpretation can be applied to statically analyse a probabilistic λ-calculus. We start by reviewing the classical framework of abstract interpretation. We choose to use (first-order) strictness analysis as our running example. We present the definition of probabilistic abstract interpretation and use it to construct a probabilistic strictness analysis.

Keywords: Probabilistic λ-calculus, strictness analysis, probabilistic abstract interpretation.

1 Introduction

In this paper we aim to show how probabilistic abstract interpretation [8,9] can be used to analyse terms in a probabilistic λ-calculus. Our running example will be a simple strictness analysis [14,2]. 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

doi:10.1016/j.entcs.2004.01.016

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 [4]), which is related to economics and in particular to risk management.

Example 1.1 [Random Walk] A company starts with initial capital of Cap_0, at each time step its income is In_i and its outlay to meet claims is 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 0 and jumps $Step_n = In_n - Out_n$:

$$Cap_n = \begin{cases} Cap_{n-1} + Step_n & \text{if } Cap_{n-1} > 0 \text{ and } Cap_{n-1} + Step_n > 0 \\ 0 & \text{otherwise} \end{cases}$$

Qualitatively, we can analyse the random walk and just conclude that 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 rest of this paper is organised as follows. We start by introducing the probabilistic λ-calculus. 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 present our approach to semantics based on linear operators and describe probabilistic abstract interpretation [8,9]. The final main section returns to the problem of strictness analysis in the probabilistic λ-calculus.

2 Probabilistic λ-calculus

A number of authors have introduced probabilistic features into the λ-calculus, see [17] and [16] for recent examples. Danos and Harmer, [6], show that most forms of probabilistic behaviour can be encoded using a coin flip. We follow this minimalist programme and simply extend the λ-calculus with the ability to make a binary, probabilistic choice between terms. We define, Λ_P, the class of probabilistic λ-terms to be the least class defined by:

- Each variable x is a term.
- Each constant, including $\perp$, is a term.
- For $M, N \in \Lambda_P$, $(MN) \in \Lambda_P$ and $(\lambda x.M) \in \Lambda_P$.
- For $M, N \in \Lambda_P$, $(M \oplus_p N)$ for some probability p.

This is the usual λ-terms plus terms of the form $M_1 \oplus_p M_2$ (indicating a probabilistic choice, the right hand summand being chosen with probability p).

We assume a leftmost reduction strategy. We write $e_1 \rightarrow_p e_2$ to mean that e_1 reduces to e_2 with probability p. Configurations, e, are a pair of a term and an environment; ρ maps *Var* to Λ_P. The semantics of terms are given by the following reduction system:

$$(\textbf{var}) \qquad (x, \rho) \rightarrow_1 \rho(x)$$

$$(\textbf{app}) \qquad \frac{(M, \rho) \rightarrow_p (P, \rho')}{((MN), \rho) \rightarrow_p ((PN), \rho')}$$

$$(\beta) \qquad ((\lambda x.M)N, \rho) \rightarrow_1^{\beta} (M, \rho[x := N])$$

$$(\delta_1) \qquad ((M \oplus_p N), \rho) \rightarrow_{(1-p)}^{\delta} (M, \rho)$$

$$(\delta_2) \qquad ((M \oplus_p N), \rho) \rightarrow_{p}^{\delta} (N, \rho)$$

Terminal configurations have a term which is a λ-term or a constant. The **app** and β rules together enforce the leftmost reduction strategy.

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 [15]. 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 (see below).

We start by sketching the classical approach to semantics-based program analysis: *abstract interpretation* [3,15]. 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*: $R : V \times L \to \{\mathit{true}, \mathit{false}\}$.

The intention is that $v\ R\ l$ formalises our claim that the value v is described by the property l.

To be useful one has to prove that the correctness relation R is preserved under computation: if the relation holds between the initial value and the initial property then it also holds between the final value and the final property. This may be formulated as the implication

$$v_1\ R\ l_1\ \wedge\ f \vdash v_1 \longrightarrow v_2\ \wedge\ f \vdash l_1 \rhd l_2 \Rightarrow v_2\ R\ l_2$$

The most common scenario in abstract interpretation is when both V and L are complete lattices. We then impose the following relationship between R and L:

(1) $\quad v\ R\ l_1\ \wedge\ l_1 \sqsubseteq l_2 \Rightarrow v\ R\ l_2$

(2) $\quad (\forall l \in L' \subseteq L : v\ R\ l) \Rightarrow v\ R\ \left(\bigsqcap L'\right)$

The first of these concerns *safety* [15]: if we have a property which correctly describes a value, then any larger property is also a safe description. The second concerns the existence of *best* descriptions: if we have a set of properties that correctly describe a value then their meet will also be a correct description and is more accurate.

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: $F_{\mathrm{abs}} \sqsupseteq \alpha \circ F \circ \gamma$.

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 [3].

3.1 Strictness Analysis

Strictness analysis [14,2] aims to answer for some function *f*: Does $f\ \bot = \bot$? 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. We will restrict ourselves to a first order functional language with integers as the only data type.

We can construct a Galois connection $(\mathcal{P}_H(\mathbf{Z}_\bot), \alpha, \gamma, \mathbf{Two})$ where $\mathcal{P}_H$ is the *Hoare Powerdomain* construction and **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: i.e. every set contains $\bot$.

We define:

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

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. We could just apply the classical framework to this new setting [12,13]; instead we will apply the techniques of *Probabilistic Abstract Interpretation* [8,9].

4 Linear Representations

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

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

The semantics of terms in our extended calculus can be represented by a probabilistic *reduction graph* where edges are labelled with probabilities. We associate to each quantitative relation $R \subseteq X \times \mathbb{W} \times X$ a matrix, i.e. a linear operator $\mathbf{M}_R$ on $\mathcal{V}(X)$ defined by:

$$(\mathbf{M}_R)_{ij} = \begin{cases} w \text{ iff } \Sigma_{R(x_i,w',x_j)} w' = w \\ 0 \ \text{ otherwise} \end{cases}$$

For probabilistic relations $\mathbb{W} = [0,1]$ and this gives a *Stochastic* matrix.

4.1 Linear Semantics for the Probabilistic λ-calculus

We start by defining three operators which represent transitions corresponding to β-reduction, δ-reduction (for the probabilistic choice) and idling (for terms in normal form) respectively. Each operator is of type $\mathcal{V}(\Lambda_P) \to \mathcal{V}(\Lambda_P)$.

Before defining the operator for one-step β-reduction, we define an appropriate notion of *active context* – these are λ-terms with a single hole which determines where the next redex to be reduced is to be found. Such contexts are used in the definition of *compatible closure* in the standard construction of one-step reductions [1]; since we are interested in call-by-name evaluation, we do not reduce redexes in the argument or under λs. Given this intuition, $C[\,]$, the class of active contexts with a single hole is the least class such that:

- $[\,] \in C[\,]$.
- $C_1[\,]N \in C[\,]$ for any $C_1[\,] \in C[\,]$ and $N \in \Lambda_P$.

The operator $\mathbf{B}$ (for β reduction) has the following matrix representation for each bound variable x and term $N \in \Lambda_P$:

$$\mathbf{B}(x,N)_{t_1,t_2} = \begin{cases} 1 \text{ if } t_1 \equiv C[(\lambda x.M)N], t_2 \equiv C[M[x := N]] \\ 0 \text{ otherwise} \end{cases}$$

The operator $\mathbf{C}$ for the probabilistic choice operator has the following matrix representation:

$$\mathbf{C}_{t_1,t_2} = \begin{cases} (1-p) & \text{if } t_1 \equiv C[M \oplus_p N],\ t_2 \equiv C[M] \\ p & \text{if } t_1 \equiv C[M \oplus_p N],\ t_2 \equiv C[N] \\ 0 & \text{otherwise} \end{cases}$$

Finally, the idling operator, $\mathbf{N}$ (for normal forms), has the following matrix representation:

$$\mathbf{N}_{t_1,t_2} = \begin{cases} 1 \text{ if } t_1 \equiv t_2,\ t_1 \text{ is a } \beta\delta\text{nf} \\ 0 \text{ otherwise} \end{cases}$$

The operator, $\mathbf{T}$, which describes the transitions available from a term is then defined as:

$$\mathbf{T} = \mathbf{C} + \mathbf{N} + \Sigma_{x,N}\mathbf{B}(x, N)$$

In practice we will restrict to the reachable terms from some given term, M:

$$\mathcal{R}(M) = \{N \mid M \rightarrow^*_{\beta\delta} N\}$$

and work with the restricted transition operator:

$$\overline{\mathbf{T}} = \pi_M \mathbf{T} \pi_M$$

where π_M is the projection on to $\mathcal{V}(\mathcal{R}(M))$.

The semantics of a term is then recovered by iterated application of $\mathbf{T}$ to the vector $\vec{M}$ representing the initial term M:

$$\lim_{i \rightarrow \infty} \mathbf{T}^i \vec{M}.$$

If we have only finitely many reachable terms, e.g. if the reduction of a given term M terminates after a finite number of reduction steps, the reduced operator $\overline{\mathbf{T}}$ operates (in effect) on only a finite dimensional sub-vector space of $\mathcal{V}(\Lambda_P)$. It is well known that all topologies on finite dimensional vector spaces which are compatible with the algebraic structure are equivalent[10, Section 7.18]. This means that for finite reductions the re-construction of the operational semantics will lead to the same limit, independently of the topology considered.

It is also possible to extend the construction to the case of non-terminating reductions, or more generally the case of infinitely many reachable terms. However, this requires a more careful consideration of the topological structure used to construct the limit. In order to keep our presentation succinct we will omit here a detailed investigations of this situation as it requires more elaborate concepts from functional analysis and operator theory.

Example 4.1 Consider the term:

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

An enumeration of the reachable terms is:

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

and we have:

$$\overline{\mathbf{T}} = \begin{pmatrix} 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{3}{4} \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

5 Probabilistic Abstract Interpretation

Given two probabilistic domains (i.e. Hilbert spaces over λ-terms), $\mathcal{C}$ and $\mathcal{D}$, a *probabilistic abstract interpretation* [8,9] 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

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

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:

(i) $\mathbf{A}\mathbf{A}^\dagger\mathbf{A} = \mathbf{A}$,

(ii) $\mathbf{A}^\dagger\mathbf{A}\mathbf{A}^\dagger = \mathbf{A}^\dagger$,

(iii) $(\mathbf{A}\mathbf{A}^\dagger)^* = \mathbf{A}\mathbf{A}^\dagger$,

(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 $\gamma \circ \alpha \circ \gamma = \gamma$.

A simple method to construct a probabilistic abstract interpretation is as follows: Given a linear operator Φ on some vector space $\mathcal{V}$ expressing the probabilistic semantics of a concrete system, and a linear abstraction function $\mathbf{A} : \mathcal{V} \mapsto \mathcal{W}$ from the concrete domain into an abstract domain $\mathcal{W}$, we compute the (unique) Moore-Penrose pseudo-inverse $\mathbf{G} = \mathbf{A}^\dagger$ of $\mathbf{A}$. The 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 is guaranteed to be *correct*. In our quantitative setting the induced abstract semantics is the one *closest* to the concrete semantics. This is due to the relation between the Moore-Penrose pseudo-inverse and so-called the least-square approximation, cf e.g. [7,5].

To be more precise: Let $\mathcal{C}$ 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 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 (there is always some loss of precision) 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.

6 Probabilistic Strictness Analysis

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 operator: a classification operator.

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 (these coincide for real $\mathbf{K}$).

$$\mathbf{K}^\dagger = \mathcal{N}(\mathbf{K}^T).$$

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 a more informative analysis – the defined value means *definitely* terminating. This abstraction is achieved by a classification operator.

Example 6.1 A suitable classification matrix for our running example is

$$\mathbf{K} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

which has Moore-Penrose pseudo-inverse

$$\mathbf{K}^\dagger = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{4} & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} \end{pmatrix}$$

The abstract semantics of our original program is

$$\mathbf{K}^\dagger \overline{\mathbf{T}} \mathbf{K} = \begin{pmatrix} 1 & 0 & 0 \\ \frac{1}{16} & \frac{1}{2} & \frac{7}{16} \\ 0 & 0 & 1 \end{pmatrix}$$

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 three iterations we have:

$$(\mathbf{K}^\dagger \overline{\mathbf{T}} \mathbf{K})^3 = \begin{pmatrix} 1 & 0 & 0 \\ \frac{7}{64} & \frac{1}{8} & \frac{49}{64} \\ 0 & 0 & 1 \end{pmatrix}$$

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 interpreting relations as linear operators allows us to 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*:

- $\|\vec{x}\| \geq 0$
- $\|\vec{x}\| = 0 \leftrightarrow \vec{x} = \vec{o}$
- $\|\alpha\vec{x}\| = |\alpha|\|\vec{x}\|$
- $\|\vec{x} + \vec{y}\| \leq \|\vec{x}\| + \|\vec{y}\|$

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 6.2 An accurate abstraction of the original program, computed from the reduction graph, is:

$$\mathbf{T}^{\#} = \begin{pmatrix} 1 & 0 & 0 \\ \frac{1}{8} & 0 & \frac{7}{8} \\ 0 & 0 & 1 \end{pmatrix}$$

Considering the difference between this and our first abstraction we get:

$$\|\mathbf{T}^{\#} - \mathbf{K}^{\dagger}\overline{\mathbf{T}}\mathbf{K}\|_{\infty} = \frac{1}{2}$$

whilst

$$\|\mathbf{T}^{\#} - (\mathbf{K}^{\dagger}\overline{\mathbf{T}}\mathbf{K})^{3}\|_{\infty} = \frac{1}{8}$$

We have abstracted $\overline{\mathbf{T}}$ but we could also iterate this operator.

Example 6.3 We find that:

$$\lim_{i\to\infty}\overline{\mathbf{T}}^{i} = \overline{\mathbf{T}}^{3} = \begin{pmatrix} 0 & 0 & 0 & \frac{1}{2} & 0 & \frac{1}{8} & \frac{3}{8} \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{3}{4} \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{3}{4} \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

The abstraction of this is:

$$(\mathbf{K}^\dagger \overline{\mathbf{T}}^3 \mathbf{K}) = \begin{pmatrix} 1 & 0 & 0 \\ \frac{5}{32} & 0 & \frac{27}{32} \\ 0 & 0 & 1 \end{pmatrix}$$

and

$$\|\mathbf{T}^\# - \mathbf{K}^\dagger \overline{\mathbf{T}}^3 \mathbf{K}\|_\infty = \frac{1}{32}$$

Finally, it should also be noted that:

$$\|\mathbf{K}^\dagger \overline{\mathbf{T}}^3 \mathbf{K} - (\mathbf{K}^\dagger \overline{\mathbf{T}} \mathbf{K})^3\|_\infty = \frac{1}{8}$$

7 Conclusions

We have reviewed the classic approach to abstract interpretation and also shown that Di Pierro and Wiklicky's notion of probabilistic abstract interpretation is a natural analogue of the classical framework. We have illustrated the approach for the λ-calculus in the context of a simple strictness analysis. A present shortcoming of our work is that neither the linear semantics nor the strictness analysis are defined in a compositional way. If we were able to give a compositional linear semantics, we would expect that compositional strictness analysis would be straightforward. Unfortunately this has to remain work for the future but it is possible that earlier work on the relationship between λ-calculus and operator algebras [11] could help in this endeavour.

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] 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.

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

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

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

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

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

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

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

[11] 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.

[12] D. Monniaux. An abstract Monte-Carlo method for the analysis of probabilistic programs (extended abstract). In *Proceedings of POPL*, pages 93–101, 2001. ACM.

[13] D. Monniaux. Backwards abstract interpretation of probabilistic programs. In *Proceedings of ESOP '01*, number 2028 in LNCS. Springer Verlag, 2001.

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

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

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

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