Semantics of Probabilistic Programs: A Weak Limit Approach

Alessandra Di Pierro[1] and Herbert Wiklicky[2]

[1] Dipartimento di Informatica, Università di Verona, Italy
[2] Department of Computing, Imperial College London, UK

Abstract. For a simple probabilistic language we present a semantics based on linear operators on infinite dimensional Hilbert spaces. We show the equivalence of this semantics with a standard operational one and we discuss its relationship with the well-known denotational semantics introduced by Kozen. For probabilistic programs, it is typical to use Banach spaces and their norm topology to model the properties to be analysed (observables). We discuss the advantages in considering instead Hilbert spaces as denotational domains, and we present a weak limit construction of the semantics of probabilistic programs which is based on the inner product structure of this space, i.e. the duality between states and observables.

1 Introduction

The formal analysis of probabilistic systems is gaining increasing importance for its recognised benefits in various areas such as distributed systems, where randomised schemes are used to enhance efficiency, and in general to the design of systems with unreliable and unpredictable behaviour, where probability provides a means to make predictions based on the evaluation of performance characteristics (see e.g. [1] and the references therein). A recent trend in system design is highlighting the need for formal analysis techniques that are able to provide quantitative estimates of a system property and mathematical tools for cost optimisation. Several of our own recent works have shown how probabilistic static analysis can serve this purpose (see e.g. [2, 3]).

In order to have a sound basis for such an analysis we need a formal semantics of probabilistic programs. A popular choice for this is the denotational semantics introduced by Kozen in [4]. Despite its mathematical simplicity and clarity this semantics presents some limitations when used for program analysis. One problem is that it is mainly concerned with I/O behaviours, i.e. it only takes into account the final results of a program execution. This implies the identification of a number of behaviours and consequently a loss of precision of any static analyses based on it. Another limitation is that it does not provide a good basis for a relational analysis as correlations between program variables and properties are not made explicit.

We will investigate in this paper an alternative approach to probabilistic semantics which we argue is better suited for probabilistic program analysis. It essentially constructs the generator of a Discrete Time Markov Chain (DTMC) in a syntax-driven way similarly to the collecting semantics in classical program analysis [5]. The topological aspects of the resulting so-called Linear Operator Semantics (LOS) stem from the theory of infinite-dimensional Hilbert spaces. The choice of Hilbert spaces instead of the Banach spaces used in [4] is mainly motivated by the presence of an inner product. More precisely, the notion of an observable as a functional in the dual space of the state space coincides here with the notion of a state (i.e. a probability distribution) as Hilbert spaces are self-dual. We will show that this allows us to define a semantics based on a notion of equivalence which is finer than I/O semantics.

An additional aspect of our construction is an explicit consideration of program labels which are used on one hand to identify particular intermediate execution points but which also allow us to investigate the control flow within a program explicitly. The importance of considering labels has been discussed recently, for example, in the context of program obfuscation [6]. The removal of label information makes it more difficult to de-obfuscate programs via static program analysis, even if one can develop ways to reconstruct such information later. We compare in Section 4 the LOS semantics with Kozen's approach which does not consider program labels.

The main drawback of the construction of semantical operators on infinite-dimensional Hilbert spaces is the fact that even for simple programs, e.g. a constant assignment, it leads to unbounded operators, making it problematic the construction of a well-defined semantics for the program. Ideally, for semantical purposes one would like to consider only operators which are bounded as this requirement is equivalent to continuity of set-theoretical structures.

In order to overcome this problem we replace the notion of norm limit used in [4] by a weaker one, namely the *weak limit*, where convergence is defined directly in terms of inner product. Using weak limit constructions we can approximate the object we are interested in (i.e. the semantics $\mathbf{T}(P)$ of a program P) even though it is not in the semantical domain by considering the effects of its finite dimensional (and thus bounded) approximations on the state space. The classical concept which this approach resembles is the theory of generalised function (developed by Schwartz, Sobolev, et.al.), in particular Dirac's δ "function" which is not a function and yet can be modelled as the weak, more precisely weak$-^*$, limit of functions [7, 8].

As already mentioned, the weak limit semantics we introduce in this paper is intended to provide a sound mathematical basis for program analysis and in particular for probabilistic abstract interpretation [9–11]. This technique allows us to obtain a simplified semantics via an abstraction $\mathbf{A}$ and its corresponding concretisation $\mathbf{A}^\dagger$ defined by the so-called *Moore-Penrose pseudo-inverse*. The abstract semantics for a program P is then obtained as $\mathbf{T}^\#(P) = \mathbf{A}^\dagger\mathbf{T}(P)\mathbf{A}$ and can be constructed compositionally thanks to the properties of the tensor product operation and of the particular notion of generalised inverse we use for the abstraction on infinite-dimensional Hilbert space. In fact, these properties allow us to construct both concrete and abstract semantics by combining (via the tensor product) the effects of individual statements and their local effects.

Mathematical Background and Notation. For the mathematical notions and notation used in this paper we refer to the standard literature and, in particular, to the recent monograph by Kubrusly [12] for the functional analytical and operator algebraic concepts and to the presentation in [13] for the measure theoretic notions.

2 The Language

We will discuss our approach by referring to a probabilistic language which is a simplified version of the language in [10] and essentially the same as the one used [4]. In this section we introduce both the syntax and the operational semantics for this language, which we call **pWhile**.

Syntax. In a style typical of static analysis [5], we introduce labels in the syntax of the language. Labels are used to identify the programs points and are crucial for defining a formal semantics that is suitable for static analysis.

$$S \texttt{::=} [\texttt{skip}]^\ell \mid [x \texttt{ := } e]^\ell \mid [x \texttt{ ?= } \rho]^\ell \mid S_1 \texttt{;} S_2 \mid \texttt{if } [b]^\ell \texttt{ then } S_1 \texttt{ else } S_2 \texttt{ fi} \mid \texttt{while } [b]^\ell \texttt{ do } S \texttt{ od}$$

We denote by **Stmt** the set of all **pWhile** statements S and assume a unique labelling (by numbers $\ell \in \mathbf{Lab}$).

The statement `skip` does not have any operational effect but can be used, for example, as a placeholder in conditional statements. We have the usual (deterministic) assignment $x \texttt{ := } e$, sometimes also in the form $x \texttt{ := } f(x_1, \ldots, x_n)$.

In the random assignment $x \texttt{ ?= } \rho$, the value of a variable x is set to a value according to some random distribution ρ. In [4] it is left open how to define or specify distributions ρ in detail. We will use occasionally an ad-hoc notation as sets of tuples $\{(v_i, p_i)\}$ expressing the fact that value v_i will be selected with probability p_i; or just as a set $\{v_i\}$ assuming a uniform distribution on the values v_i. It might be useful to assume that the random number generator or scheduler which implements this construct can only implement choices over finite ranges, but in principle we can also use distributions with infinite support.

For the rest we have the usual sequential composition, conditional statement and loop. We leave the detailed syntax of functions f or expressions e open as well as for boolean expressions or test b in conditionals and loop statements.

SOS Semantics. The operational semantics of **pWhile** is defined in the SOS style [14] by means of a probabilistic transition system on the set **Conf** of configurations $\langle S, s \rangle$, where S is a **pWhile** program and s a classical state $s : \mathbf{Var} \to \mathbf{Value}$. The transition rules are given in Table 1. We assume an evaluation function $\mathcal{E} : \mathbf{Expr} \to (\mathbf{State} \to \mathbf{Value})$ for expressions defined in the usual way (assuming that **Value** contains e.g. integers as well as booleans `true` and `false`).

R0	$\langle \texttt{stop}, s\rangle \longrightarrow_1 \langle \texttt{stop}, s\rangle$	
R1	$\langle \texttt{skip}, s\rangle \longrightarrow_1 \langle \texttt{stop}, s\rangle$	
R2	$\langle v \texttt{ := } e, s\rangle \longrightarrow_1 \langle \texttt{stop}, s[v \mapsto \mathcal{E}(e)s]\rangle$	
R3	$\langle v \texttt{ ?= } \rho, s\rangle \longrightarrow_{\rho(r)} \langle \texttt{stop}, s[v \mapsto r]\rangle$	
R4$_1$	$\dfrac{\langle S_1, s\rangle \longrightarrow_p \langle S_1', s'\rangle}{\langle S_1; S_2, s\rangle \longrightarrow_p \langle S_1'; S_2, s'\rangle}$	
R4$_2$	$\dfrac{\langle S_1, s\rangle \longrightarrow_p \langle \texttt{stop}, s'\rangle}{\langle S_1; S_2, s\rangle \longrightarrow_p \langle S_2, s'\rangle}$	
R5$_1$	$\langle \texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2 \texttt{ fi}, s\rangle \longrightarrow_1 \langle S_1, s\rangle$	if $\mathcal{E}(b)s = \texttt{true}$
R5$_2$	$\langle \texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2 \texttt{ fi}, s\rangle \longrightarrow_1 \langle S_2, s\rangle$	if $\mathcal{E}(b)s = \texttt{false}$
R6$_1$	$\langle \texttt{while } b \texttt{ do } S \texttt{ od}, s\rangle \longrightarrow_1 \langle S; \texttt{ while } b \texttt{ do } S \texttt{ od}, s\rangle$	if $\mathcal{E}(b)s = \texttt{true}$
R6$_2$	$\langle \texttt{while } b \texttt{ do } S \texttt{ od}, s\rangle \longrightarrow_1 \langle \texttt{stop}, s\rangle$	if $\mathcal{E}(b)s = \texttt{false}$

Table 1. The rules of the SOS semantics of **pWhile**

Our aim is to identify the execution (process) of a program P according to the SOS rules as the realisation of a Discrete Time Markov Chain (DTMC) [15]. Markov chains are essentially transition systems where the successor state of a state is chosen according to a probability distribution. This probability distribution only depends on the current state, so that the system evolution is independent of the history. This is known as the memoryless property. The name Discrete Time Markov Chain refers to the fact that Markov chains are used as a time-abstract model (like transition systems): each transition is assumed to take a single time unit.

DTMC are non-terminating processes: it is assumed that there is always a next state and the process goes on forever. In order to reflect this property in our semantics, we introduce a terminal statement `stop` which indicates successful termination. Then the termination with a state s in the classical setting is represented here by reaching the final configuration $\langle \texttt{stop}, s\rangle$ which then 'loops' forever after. This means that we implicitly extend a statement S to construct full programs of the form $P \equiv S;\ [\texttt{stop}]^{\ell^*}$.

The probabilistic transition system defined in Table 1 is indeed describing a DTMC, as we obviously have a memoryless process: the transitions in Rules **R0** to **R6** depend only on the current configuration and not on the sequence of the configurations that preceded it. It is well-known that the matrix of transition probabilities of a DTMC on a countable state space is a stochastic matrix, i.e. a square (possibly infinite) matrix $\mathbf{P} = (p_{ij})$ whose elements are real numbers in the closed interval $[0, 1]$, for which $\sum_j p_{ij} = 1$ for all i [15, 16]. We can therefore represent the SOS semantics for a **pWhile** program P by the stochastic matrix on the vector space over the set **Conf** of all configurations of a program P defined by the rules in Table 1.

3 Linear Operator Semantics

The SOS semantics introduced in Section 2 specifies effectively the generator of a DTMC representing all executions of the program. However, the representation of this operator as a single unstructured matrix is not a convenient one for a denotational approach as it is not *compositional* (it stems from the SOS).

The labelled version of the syntax introduced in Section 2 allows us to use labels as a kind of program counter. Labels in **Lab** can therefore be used as delimiters of those relevant parts of the program that effectively correspond to the application of each language constructor. Moreover, they allow us to track the computational progress through the program execution, so yielding a semantics which is not only concerned with the I/O behaviour of the program but can also capture some finer notions of observables.

In the following we will present a semantics that we call Linear Operator Semantics (LOS), as it is the composition of different linear operators on the Hilbert space $\ell_2(\mathbf{Conf})$ over configurations, each expressing a particular operation, and contributing to the overall behaviour of the program. More precisely, the LOS is constructed compositionally by means of the operators representing each block of the program. The resulting operator $\mathbf{T}(P)$ is represented by an infinite matrix which we will show to be equivalent to the SOS matrix from Section 2. Moreover, we will show how this can be constructed as the weak limit of a sequence of finite approximations.

States and Observables. We assume that variables occurring in a **pWhile** program can take values in some countable set $\mathbb{X}$ that might be finite (e.g. Booleans) or infinite (typically $\mathbb{Z}$ or $\mathbb{N}$). We will refer to an implicitly given enumeration $\xi : \mathbb{X} \to \mathbb{N}$ of $\mathbb{X}$ (e.g. $\xi = \mathrm{id}$ for $\mathbb{X} = \mathbb{N}$). For high-order languages or languages with e.g. real-valued variables one might need to work with uncountable sets (e.g. $\mathbb{X} = \mathbb{Z} \to \mathbb{Z}$ or $\mathbb{X} = \mathbb{R}$). However, for imperative languages like the one we consider in this paper a finite or countable infinite value space will do. We can nevertheless extend our framework also to deal with the uncountable case.

The *classical state* is defined as a map $s : \mathbf{Var} \to \mathbb{X}$. For a set of v variables $\mathbf{Var} = \{x_1, \dots, x_v\}$ we can identify the classical state space with the v-fold cartesian product $\mathbb{X}^v = \mathbb{X} \times \dots \times \mathbb{X}$. Concretely, the classical state $[x_1 \mapsto v_1, x_2 \mapsto v_2, \dots x_v \mapsto v_v]$ corresponds to the v-tuple $(v_1, v_2, \dots, v_v)$.

In our model *probabilistic states* σ are vectors of a *Hilbert space*, i.e. an inner product space that is complete under the metric induced by the inner product, [17, 12]. We recall that an inner product space $\mathcal{H}$ (over the reals $\mathbb{R}$) is a vector space $\mathcal{H}$ together with an inner product $\langle \cdot, \cdot \rangle : \mathcal{H} \times \mathcal{H} \to \mathbb{R}$, that is a function that is linear and continuous in both its arguments. An inner product induces a norm on $\mathcal{H}$ defined by $\|x\| = \sqrt{\langle x, x \rangle}$.

The concrete Hilbert space we will consider is $\ell_2(\mathbb{X})$, i.e. the space of all sequences $x = (x_i)_{i \in \mathbb{X}}$ for which $\sum_i |x_i|^2 < \infty$ holds [18, Def 1.14]. In fact, all Hilbert spaces with a countable infinite base are isomorphic to $\ell_2(\mathbb{N}) = \ell_2$ [17, Thm 2.2.12]. With the 2-norm $\|x\|_2 = \|(x_i)\|_2 = \left(\sum_{i \in \mathbb{X}} |x_i|^2\right)^{\frac{1}{2}}$ this is a Hilbert space as its norm $\|x\|_2 = \sqrt{\langle x, x \rangle}$ is induced by the inner product $\langle (x_i), (y_i) \rangle = \sum_i x_i y_i$. It contains as a dense sub-space the Banach space $\ell_1(\mathbb{X})$ which is equipped with the 1-norm $\|x\|_1 = \|(x_i)\|_1 = \sum_{i \in \mathbb{X}} |x_i| < \infty$ [18, Exercise 1.14]. We can represent the state of several variables $x_1, \dots, x_v$ by a vector in the tensor product $\ell_2(\mathbb{X}) \otimes \dots \otimes \ell_2(\mathbb{X}) = \ell_2(\mathbb{X}^v)$.

$$\mathit{flow}([\mathtt{skip}]^\ell) = \mathit{flow}([v\ \mathtt{:=}\ e]^\ell) = \mathit{flow}([v\ \mathtt{?=}\ e]^\ell) = \emptyset$$
$$\mathit{flow}(S_1; S_2) = \mathit{flow}(S_1) \cup \mathit{flow}(S_2) \cup \{(\ell, \mathit{init}(S_2)) \mid \ell \in \mathit{final}(S_1)\}$$
$$\mathit{flow}(\mathtt{if}\ [b]^\ell\ \mathtt{then}\ S_1\ \mathtt{else}\ S_2\ \mathtt{fi}) = \mathit{flow}(S_1) \cup \mathit{flow}(S_2) \cup \{(\ell, \underline{\mathit{init}(S_1))}, (\ell, \mathit{init}(S_2))\}$$
$$\mathit{flow}(\mathtt{while}\ [b]^\ell\ \mathtt{do}\ S\ \mathtt{od}) = \mathit{flow}(S) \cup \{(\ell, \underline{\mathit{init}(S)})\} \cup \{(\ell', \ell) \mid \ell' \in \mathit{final}(S)\}$$

Table 2. The control flow

The tensor product is an essential element of the description of probabilistic states. This tensor product – more precisely, the Kronecker product, i.e. the coordinate based version of the abstract concept of a tensor product – of two vectors $(x_1, \ldots, x_n)$ and $(y_1, \ldots, y_m)$ is given by $(x_1y_1, \ldots, x_1y_m, \ldots, x_ny_1, \ldots, x_ny_m)$ an nm dimensional vector. For an $n \times m$ matrix $\mathbf{A} = (\mathbf{A}_{ij})$ and an $n' \times m'$ matrix $\mathbf{B} = (\mathbf{B}_{kl})$ we construct similarly an $nn' \times mm'$ matrix $\mathbf{A} \otimes \mathbf{B} = (\mathbf{A}_{ij}\mathbf{B})$, i.e. each entry $\mathbf{A}_{ij}$ in $\mathbf{A}$ is multiplied with a copy of the matrix or block $\mathbf{B}$. The tensor product of two vector spaces $\mathcal{V} \otimes \mathcal{W}$ can be defined as the formal linear combinations of the tensor products $v_i \otimes w_j$ with v_i and w_j base vectors in $\mathcal{V}$ and $\mathcal{W}$, respectively. For further details we refer e.g to [19, Chap. 14] and for a detailed discussion of tensor products of Hilbert spaces to [17, Sect.2.6].

The notions of semantic states and observables – which are typically both identified with the subsets of some appropriate cpo in the standard approach to nondeterministic semantics – are in the probabilistic case two distinct geometrical aspects that are *dual* to each other in the sense that they belong to *dual spaces*. The dual space of a normed space $\mathcal{X}$, denoted $\mathcal{X}^*$, is the normed space of all continuous linear functionals on $\mathcal{X}$. If $\mathcal{X}$ is a Hilbert spaces then its dual is again a Hilbert space. Thus, as $\ell_2(\mathbb{X})^* = \ell_2(\mathbb{X})$, we have for states $y \in \ell_2(\mathbb{X})$ that observables x are also in $\ell_2(\mathbb{X})$. They are related to each other by the notion of expected value, $\mathbf{E}(x, y)$, which represents the probability that we will observe a certain property x when the state of the system is described by y. In $\ell_2(\mathbb{X})$ we can take $\mathbf{E}(x, y) = \langle x, y \rangle$. Duality is more involved for general Banach spaces, where for example the dual, $\ell_1(\mathbb{X})^*$, of the space $\ell_1(\mathbb{X})$ is $\ell_\infty(\mathbb{X})$, i.e. the space of all sequences with $\|(x_i)\|_\infty = \sup(x_i) < \infty$.

The Control Flow. For the definition of the control flow of a program we follow the presentation in [5]. It is based on two auxiliary operations $\mathit{init} : \mathbf{Stmt} \to \mathbf{Lab}$ and $\mathit{final} : \mathbf{Stmt} \to \mathcal{P}(\mathbf{Lab})$ which return the initial and the final labels of a statement. The control flow in a statement S is then defined by the function $\mathit{flow} : \mathbf{Stmt} \to \mathcal{P}(\mathbf{Lab} \times \mathbf{Lab})$ which maps statements to sets of pairs which represent the control flow graph. It is defined in Table 2. This only records that a certain control flow step is possible. For tests b in conditionals and loops we indicate the branch corresponding to the case when the test succeeds by underlining it. As our semantics is ultimately modelling the semantics of a program via the generator of a DTMC we are also confronted with the fact that such processes never terminate. This can be fixed by adding an additional label ℓ^* to the set of labels and define the flow of a program P as $\mathcal{F}(P) = \mathit{flow}(P) \cup \{(\ell, \ell^*) \mid \ell \in \mathit{final}(P)\} \cup \{(\ell^*, \ell^*)\}$.

Infinite Generator Matrix. Given a program P, our aim is to define compositionally an infinite matrix representing the program behaviour as a DTMC. The domain of the associated linear operator $\mathbf{T}(P)$ is the space of *probabilistic configurations*, that is distributions over classical configurations, defined by $\mathbf{Dist}(\mathbf{Conf}) = \mathbf{Dist}(\mathbb{X}^v \times \mathbf{Lab}) \subseteq \ell_2(\mathbb{X}^v \times \mathbf{Lab})$, where we identify a statement with its label or, more precisely, an SOS configuration $\langle S, s\rangle \in \mathbf{Conf}$ with the pair $\langle s, \mathit{init}(S)\rangle \in \mathbb{X}^v \times \mathbf{Lab}$.

Among the building blocks of the construction of $\mathbf{T}(P)$ are the *identity matrix* $\mathbf{I}$ and the *matrix units* $\mathbf{E}_{ij}$ containing only a single non zero entry $(\mathbf{E}_{ij})_{ij} = 1$ and zero otherwise. We denote by e_i the unit vector with $(e_i)_i = 1$ and zero otherwise. As we represent distributions by row vectors we use post-multiplication, i.e. $\mathbf{T}(x) = x \cdot \mathbf{T}$.

A basic operator is the *update matrix* $\mathbf{U}(c)$ which implements state changes. The intention is that from an initial probabilistic state σ, e.g. a distribution over classical states, we get a new probabilistic state σ' by the product $\sigma' = \sigma \cdot \mathbf{U}$. The matrix $\mathbf{U}(c)$ implements the deterministic update of a variable to a constant c via $(\mathbf{U}(c))_{ij} = 1$ if $\xi(c) = j$ and 0 otherwise, with $\xi : \mathbb{X} \to \mathbb{N}$ the underlying enumeration of values in $\mathbb{X}$. In other words, this is a (possibly infinite) matrix which has only one column (corresponding to c) containing 1s while all other entries are 0. Whatever the value of a variable is, after applying $\mathbf{U}(c)$ to the state vector describing the current situation we get a *point* distribution expressing the fact that the value of our variable is now c.

We also define for any Boolean expression b on $\mathbb{X}$ a diagonal *projection matrix* $\mathbf{P}$ with $(\mathbf{P}(b))_{ii} = 1$ if $b(c)$ holds and $\xi(c) = i$ and 0 otherwise. The purpose of this diagonal matrix is to "filter out" only those states which fulfil the condition b. If we want to apply an operator with matrix representation $\mathbf{T}$ only if a certain condition b is fulfilled then pre-multiplying this $\mathbf{P}(b) \cdot \mathbf{T}$ achieves this effect.

In Table 3 we first define a multi-variable versions of the test matrices and the update matrices via the tensor product '$\otimes$'. We define with $\mathbf{P}(s)$ an operator which tests if the current state is the same as the (classical) state s: Given the state $s = [\mathtt{x}_i \mapsto s(\mathtt{x}_i)]$ we test for each variable $\mathtt{x}_i$ with $i = 1, \ldots, v$ if it has the same value as specified in s by applying $\mathbf{P}(s)$ in each factor of the tensor product, i.e. $\mathbf{P}(s(\mathtt{x}_i)) = \mathbf{P}(\mathtt{x}_i = s(\mathtt{x}_i))$. If we apply $\mathbf{P}(s)$ to a probabilistic state σ then $\mathbf{P}(s)$ filters out the probabilities that each variable has exactly the value specified by the state s. The operator $\mathbf{P}(e = c)$ tests in a similar way if the current state is such that the expression e evaluates to the constant c. In order to accommodate for general expressions e (not just constants) we collect (sum up) the matrices for which $\mathcal{E}(e)s = e$. The update operator $\mathbf{U}(\mathtt{x}_k \leftarrow c)$ assigns a definitive constant value c to variable $\mathtt{x}_k$, all other variables remain unchanged (which is expressed by the fact that the factors corresponding to the other variables in the tensor product are all the identity $\mathbf{I}$). Finally, the operator $\mathbf{U}(\mathtt{x}_k \leftarrow e)$ assigns the value

$$\mathbf{P}(s) = \bigotimes_{i=1}^{v} \mathbf{P}(s(\mathtt{x}_i)) \qquad \mathbf{U}(\mathtt{x}_k \leftarrow c) = \bigotimes_{i=1}^{k-1} \mathbf{I} \otimes \mathbf{U}(c) \otimes \bigotimes_{i=k+1}^{v} \mathbf{I}$$
$$\mathbf{P}(e = c) = \sum_{\mathcal{E}(e)s=c} \mathbf{P}(s) \qquad \mathbf{U}(\mathtt{x}_k \leftarrow e) = \sum_{c} \mathbf{P}(e = c)\mathbf{U}(\mathtt{x}_k \leftarrow c)$$

Table 3. Elementary Operators

$$[\![[x \texttt{ := } e]^{\ell]}]\!] = \mathbf{U}(x \leftarrow e) \qquad [\![[v \texttt{ ?= } \rho]^{\ell}]\!] = \textstyle\sum_{c \in \mathbb{X}} \rho(c)\mathbf{U}(x \leftarrow c)$$
$$[\![[b]^{\ell}]\!] = \mathbf{P}(b = \texttt{false}) \qquad \underline{[\![[b]^{\ell}]\!]} = \mathbf{P}(b = \texttt{true})$$
$$[\![[\texttt{skip}]^{\ell}]\!] = \underline{[\![[\texttt{skip}]^{\ell}]\!]} = \underline{[\![[x \texttt{ := } e]^{\ell]}]\!]} = \underline{[\![[v \texttt{ ?= } \rho]^{\ell}]\!]} = \mathbf{I}$$

Table 4. Elements of the LOS

of an expression e to $\mathtt{x}_k$. This is achieved by testing whether in the current state e evaluates to any of the possible constants c, and if so to assign c to $\mathtt{x}_k$.

With the help of the auxiliary matrices we can now define for every program P the matrix $\mathbf{T}(P)$ of the DTMC representing the program executions as the sum of the effects of the individual control flow steps. For each individual control flow step it is of the form $[\![[B]^{\ell}]\!] \otimes \mathbf{E}_{\ell,\ell'}$ or $\underline{[\![[B]^{\ell}]\!]} \otimes \mathbf{E}_{\ell,\ell'}$, where (ℓ, ℓ') or $(\ell, \underline{\ell'}) \in \mathcal{F}(P)$ and $[\![[B]^{\ell}]\!]$ represents the semantics of the block B labelled by ℓ. The matrix $\mathbf{E}_{\ell,\ell'}$ represents the control flow from label ℓ to ℓ'; it is a finite $l \times l$ matrix, where l is the number of (unique) distinct labels in P.

The definitions of $[\![[B]^{\ell}]\!]$ and $\underline{[\![[B]^{\ell}]\!]}$ are given in Table 4. The semantics of an assignment block is obviously given by $\mathbf{U}(\mathtt{x} \leftarrow e)$. For the random assignment we simply take the linear combination of assignments to all possible values, weighted by the corresponding probability given by the distribution ρ. The semantics of a test block $[b]^{\ell}$ is given by its positive and its negative part, both are test operators $\mathbf{P}(b = \texttt{true})$ and $\mathbf{P}(b = \texttt{false})$ as described before. The meaning of $\underline{[\![[B]^{\ell}]\!]}$ is non-trivial only for tests b while it is the identity for all the other blocks. The positive and negative semantics of all blocks is independent of the context and can be studied and analysed in isolation from the rest of the program P.

Based on the local (forward) semantics of each labelled block, i.e. $[\![[B]^{\ell}]\!]$ and $\underline{[\![[B]^{\ell}]\!]}$, in P we can define the LOS semantics of P as:

$$\mathbf{T}(P) = \sum_{(\ell,\ell') \in \mathcal{F}(P)} [\![[B]^{\ell}]\!] \otimes \mathbf{E}_{\ell,\ell'} + \sum_{(\ell,\underline{\ell'}) \in \mathcal{F}(P)} \underline{[\![[B]^{\ell}]\!]} \otimes \mathbf{E}_{\ell,\ell'}$$

A minor adjustment is required to make our semantics conform to the DTMC model. As paths in a DTMC are *maximal* (i.e. infinite) in the underlying directed graph, we will add a single final loop via a virtual label ℓ^* as discussed in Section 2. This corresponds to adding to $\mathbf{T}(P)$ the factor $\mathbf{I} \otimes \mathbf{E}_{\ell^*,\ell^*}$.

Correspondence between SOS and LOS. As $\mathbf{T}(P)$ operates on $\mathbf{Dist}(\mathbf{Conf})$, we can index the entries in its matrix representation by pairs of classical states

s and program labels ℓ. We can show that these entries are in a one-to-one correspondence with the generator matrix of the operational semantics in Table 1.

Proposition 1. *Let P be a* **pWhile** *program and* $\mathbf{T}(P)$ *its LOS operator. We have that if* $\mathit{init}(P) = \ell$ *and* $\mathit{init}(P') = \ell'$ *and* s, s' *are classical states, then* $\langle P, s\rangle \longrightarrow_p \langle P', s'\rangle$*if and only if* $(\mathbf{T}(P))_{(s,\ell)(s',\ell')} = p$.

The Weak Limit of $\mathbf{T}(P)$. In the standard denotational approach to the semantics of programming languages continuity is an essential requirement for the semantical functions: it guarantees the existence of fixpoints and therefore that the semantics is well-defined. For linear operators the concept of continuity is equivalent to the concept of *boundedness*. This is a basic result in functional analysis and operator theory (see, e.g. [12, Thm. 4.14]). We recall that a linear operator $\mathbf{T} : \mathcal{X} \rightarrow \mathcal{Y}$ between two normed vector spaces $\mathcal{X}$ and $\mathcal{Y}$ is bounded if $\|\mathbf{T}\| = \sup \|\mathbf{T}(x)\|/\|x\| < \infty$.

One feature of Markov chains is that, due to their memoryless property, we can obtain the future of an initial situation $x = x(0)$ (a given distribution) by iterating the generator matrix. The distribution at time t is simply $x(t) = x(0)\mathbf{T}^t$.This can be extended to infinity, i.e. we can compute the limit state distribution as $x(\infty) = \lim_{t\rightarrow\infty} x(0)\mathbf{T}^t$. The question is therefore: does this limit exist for $\mathbf{T}(P)$ for all P and input $x(0) \in \ell_2(\mathbb{X})$? The answer obviously depends on the notion of limit we have in mind. If we refer to the norm limit then the question boils down to whether $\mathbf{T}(P)$ is a bounded operator on $\ell_2(\mathbb{X})$ and the answer is negative as the following example shows.

Example 1. The operator represented by the matrix $\mathbf{U}(x \leftarrow e)$ is in general not bounded on $\ell_2(\mathbb{X})$. To see this, consider $x = (x_i)_{i=0}^{\infty} = (1, \frac{1}{2}, \frac{1}{3}, \ldots)$. Then calculating the 2-norm, $\|x\|_2$, gives rise to a well-known convergent series[3], whereas $\|\mathbf{U}(x \leftarrow 1)\|_2 = \|x\|_1$ corresponds to the harmonic series which is a well-known divergent infinite series.

However, in our setting it makes sense to consider a particular notion of limit, namely the *weak limit*, which allows us to look at the computation as the physical process of *measuring* an observable by means of successive approximations each constructed as the inner product between the observable and an approximation of its dual state.

Definition 1 (Weak Limit [12, Sect 5.11]). *A sequence of vectors* $\{x_n\}_n$ *in a Hilbert space* $\mathcal{H}$ converges weakly *to* $x \in \mathcal{H}$*, denoted by* $x_n \overset{w}{\rightarrow} x$ *or* $w\text{-}\lim_n x_n = x$*, iff for all* $y \in \mathcal{H}$ *we have* $\lim_{n\rightarrow\infty} \langle x_n, y\rangle = \langle x, y\rangle$.

With respect to this notion of limit we can show that the LOS operator $\mathbf{T}(P)$ converges weakly for any initial state and any observable (specified as vector distributions in $\ell_2(\mathbf{Conf})$).

[3] The problem of finding the closed form of the infinite series $1+\frac{1}{2^2}+\frac{1}{3^2}+\frac{1}{4^2}+\ldots$, aka the Basel problem, was solved by Euler who showed that the series is approximately equal to 1.644934.

Definition 2 (Weak Limit of Operators[12, Sect 5.11]). *A sequence of linear operators* $\mathbf{A}_n$ *on a Hilbert space* $\mathcal{H}$ *is said to* converge weakly *to a linear operator* $\mathbf{A}$*, denoted by* $\mathbf{A}_n \overset{w}{\to} \mathbf{A}$ *iff for all* $x \in \mathcal{H}$ *we have* $\mathbf{A}_n(x) \overset{w}{\to} \mathbf{A}(x)$*.*

We first need to introduce the following definition of finite approximations (or sections) of a matrix. Let $\mathbf{P}_n$ be the orthogonal projection onto the spaces spanned by the first n base vectors; then its matrix representation is given by $\mathbf{P}_n = \mathrm{diag}(1, \ldots, 1, 0, 0, \ldots)$, i.e. a diagonal matrix with only the first n diagonal entries being one and the rest all zero. For any infinite matrix $\mathbf{T}$ representing a bounded or unbounded operator, we can define its finite approximations as $\mathbf{T}_n = \mathbf{P}_n\mathbf{T}\mathbf{P}_n$, that is the effect of $\mathbf{T}$ only on the sub-space spanned by the the first n dimensions.

We will show that the numerical series obtained by calculating the inner products between the n-th approximation vector $x \cdot (\mathbf{T}(P))_n$ and an observable y always converges in $\mathbb{R}$, and we will take this limit to define $\langle x \cdot \mathbf{T}(P), y\rangle$.

Proposition 2. *Given a program* P *and its LOS operator* $\mathbf{T}(P)$*, we have that for all* $x, y \in \mathbf{Dist}(\mathbf{Conf}) \subset \ell_2(\mathbf{Conf})$ *converges,* $\lim_{n\to\infty} \langle x \cdot \mathbf{T}(P)_n, y\rangle < \infty$*.*

Proof. Let $x_n = x \cdot \mathbf{T}(P)_n$ and $y \in \mathbf{Dist}(\mathbf{Conf}) \subset \ell_1(\mathbf{Conf}) \subset \ell_2(\mathbf{Conf})$. We need to show that $\langle x_n, y\rangle = \sum_{k=1}^{\infty}(x_n)_k \cdot y_k$ converges in $\mathbb{R}$. Since the $\mathbf{T}(P)_n$ are (sub-)stochastic matrices and x is a distribution with $\|x\|_1 = 1$, we have that $\|x_n\|_1 \leq 1$ and $\langle x_n, y\rangle \leq \langle x_{n+1}, y\rangle$, i.e. monotone. Moreover, by the Cauchy-Schwarz inequality (e.g. [17, Prop. 2.1.1]) we have $\langle x_n, y\rangle \leq \|x_n\|_2\|y\|_2$, Thus, as in general $\|v\|_2 \leq \|v\|_1$ holds for all v(cf. [18, Exercise 1.14]), we have $\langle x_n, y\rangle \leq \|x_n\|_2\|y\|_2 \leq \|x_n\|_1\|y\|_1 \leq 1$, i.e. $\langle x_n, y\rangle$ is a bounded, monotone sequence of real numbers. □

Example 2. Consider $\mathbf{T} = \mathbf{U}(x \leftarrow 1)$, a distribution $s = (s_i)$ as input and observables represented by the base vectors e_i. Then we have

$$\lim_{n\to\infty} \langle s \cdot \mathbf{T}_n, e_i\rangle = \begin{cases} 1 \text{ for } i = 1 \\ 0 \text{ otherwise} \end{cases}$$

In fact, we have that $\lim_{n\to\infty} \langle s \cdot \mathbf{T}_n, e_1\rangle = \lim_{n\to\infty} \sum_{i=1}^{n} s_i = 1$, while for e_i with $i \neq 1$ it is either always zero or converges towards zero. The probability of observing $[x \mapsto 1]$ after executing x **:=** 1 is 1 and 0 for all other possible values.

Based on the weak limit we can also assert the adequacy of the LOS.

Proposition 3. *Given programs* P *and* P' *with init*$(P) = \ell$ *and init*$(P') = \ell'$*, if* $\langle P, s\rangle \longrightarrow_p \langle P', s'\rangle$ *then* $\lim_{n\to\infty} \langle (s \otimes e_\ell) \cdot \mathbf{T}_n, (s' \otimes e_{\ell'})\rangle = p$*.*

The weak limit construction also allows us to work with measures on $\mathbb{X}$ which are not representable by distributions. This is important as it is a well known problem that not all semantically interesting probabilistic behaviours (even on countable infinite spaces) can be described by distributions.

$[\![\texttt{skip}]\!]$	$= \mathbf{I}$
$[\![x \texttt{ := } f(x_1,\ldots,x_n)]\!]$	$= \mathbf{U}(x \leftarrow f(x_1,\ldots,x_n))$
$[\![x \texttt{ ?= } \rho]\!]$	$= (\sum_c \rho(c)\mathbf{U}(x \leftarrow c))$
$[\![S_1 \texttt{;} S_2]\!]$	$= ([\![S_1]\!][\![S_2]\!])$
$[\![\texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2 \texttt{ fi}]\!]$	$= (\mathbf{P}(b)[\![S_1]\!] + \mathbf{P}(\neg b)[\![S_2]\!])$
$[\![\texttt{while } b \texttt{ do } S \texttt{ od}]\!]$	$= (\mathbf{P}(b)[\![S]\!][\![\texttt{while } b \texttt{ do } S \texttt{ od}]\!] + \mathbf{P}(\neg b))$

Table 5. Kozen's semantics

Example 3. Consider the program fragment $P \equiv x$ `:=` $2x$. Its LOS operator is given by a bounded operator $\mathbf{T}(P) = \mathbf{U}(x \leftarrow 2x)$. If we are interested in the probability of obtaining any even number as the result of executing P on an initial distribution x then we can test it on an elementary observable, i.e. a test which can return only 'yes' or 'no'. This implements a kind of *uniform measure* over all even numbers. Strictly speaking, no measure can exists on $\mathbb{Z}$ which would give equal probability to each even number and $\frac{1}{2}$ to the set of all evens. Such a *uniform measure* μ_{ev} cannot be represented by a distribution. However, we can approximate it by considering the distributions ev_n over the first n even numbers, i.e. $ev_1 = (1, 0, 0, \ldots), ev_2 = (\frac{1}{2}, 0, \frac{1}{2}, 0, \ldots), \ldots$. Then we get $\lim_{n\rightarrow\infty} \langle x \cdot \mathbf{T}(P), ev_n\rangle = 1$ for any initial distribution x – as expected. In other words, $\mu_{ev} = w\text{-}\lim ev_n$.

By measure on $\mathbb{Z}$ we usually mean a measure based on the σ-algebra $\mathcal{P}(\mathbb{Z})$ of all subsets of $\mathbb{Z}$. On this σ-algebra it is obviously impossible to define an atomic measure – i.e. one which is generated by the point measures $\mu(\{n\})$ of singletons $n \in \mathbb{Z}$ – which reflects the fact that half of all numbers are even and half are odd. However, it is possible to define such a measure on the (non-standard) σ-algebra $\{\emptyset, E, O, \mathbb{Z}\}$ with E and O the set of all even and odd numbers, respectively. In fact, on this σ-algebra we can introduce the measure $\mu(E) = \mu(O) = \frac{1}{2}$. Thus, the weak limit construction can simulate this measure. This appears to be consistent with classical results in measure theory like the Portemanteau theorem, e.g. [13, Thm 13.16], which allows the representation of certain measures as weak limits.

4 Comparison with Kozen's Semantics

In this section we develop a comparison with the well-known probabilistic semantics defined in [4]. We consider here the formulation which in the original paper is referred to as Semantics 2 and which is based on an iterative construction of the fixed point in the style of Knaster-Tarski [20]. Contrary to the LOS we introduced before, which describes the stepwise behaviour of a program, Kozen's semantics captures the I/O behaviour of a program by means of the probability measure reached after termination (possibly after an infinite number of steps).

Kozen's semantics 2 is defined as the fixed point of a bounded operator on a Banach space which fulfils the recursive equations in Table 5.

There are several features of Kozen's semantics which are in striking contrast with LOS. The first one consists in the fact that all probabilistic choices are assumed to be made before the program execution starts rather than during the execution as in the LOS. This seems to prevent any (non-terminating) program with infinitely many probabilistic assignments (e.g. `while true do` x `?=` $\{0,1\}$ `od`) from ever starting. A second issue is the fact that it treats all execution paths which do not terminate in the same way, namely as the zero operator Not least in the context of program analysis this seems to be rather imprecise as it might well be interesting what happens during an infinite execution path, e.g. if a non-terminating program such as an operating system will cause a variable overflow or not, c.f. `while true do` x `:=` $x+1$ `od`. Another difference is related to the fact that Kozen's semantics does not explicitly refer to relational aspects, e.g. the fact that the values of two variables might be correlated. The LOS semantics on the other hand is essentially constructed using a tensor product which models conditional probabilities in a compositional way.

It might be worth noting that in later work [21] Kozen also presents a backward semantics [21, p165] which is concerned with how measurable functions, which represent observables, need to be transformed in order to define the semantics of a program S. This is intuitively the reverse of the measure transformer semantics in [4]. This backward semantics also appears to be strongly related to the weakest pre-condition calculus. In our self-dual setting the backward semantics is easily identified as the *adjoint* operator of $[\![S]\!]$ via the conditions

$$\langle x \cdot [\![S]\!], f\rangle = \langle x, f \cdot [\![S]\!]^*\rangle$$

for a state x and an observable f in $\ell_2(\mathbf{Conf})$.

Recovering Kozen's Semantics. We can use the LOS to reconstruct the semantics of Kozen by simply taking the limit of $\mathbf{T}(S)^n(s_0)$ for $n \to \infty$ for all initial states s_0. The limit state $\mathbf{T}(S)^n(s_0)$ still contains too much information in relation to Kozen's semantics; in fact we only need the probability distributions on the possible values of the variables at the final label.

In order to extract information about the probability that variables have certain values at a certain label, i.e. program point ℓ, we can use the operator $\mathbf{I}\otimes\ldots\otimes\mathbf{I}\otimes\mathbf{E}_{\ell,\ell}$. In particular, for extracting the information about a probabilistic state we will use $\mathbf{S}_\ell = \mathbf{I}\otimes\ldots\otimes\mathbf{I}\otimes e_\ell$ which forgets about all distributions at other labels than ℓ. In particular we use $\mathbf{S}_f = \mathbf{S}_{\ell^*}$ for the final looping `stop` statement and $\mathbf{S}_i$ for the initial label $init(P)$ of the program. We denote by e_0 the base vector in $\mathbb{R}^l$ which expresses the fact that we are in the initial label, i.e. $e_0 = e_{init(P)}$.

Proposition 4. *Given a program P and an initial state s_0 as a distribution over the program variables, then $(s_0\otimes e_0)\mathbf{T}(P)^n\mathbf{S}_f$ corresponds to the distribution over all states on which P terminates in n or less computational steps.*

We can now show that the effect of the operator we obtain as solution to Kozen's fixed-point equations agrees with the "output" $\lim_{n\to\infty}(s_0\otimes e_0)\mathbf{T}^n\mathbf{S}_f$

we get via the LOS. Essentially, both semantics define the same I/O operator, provided we supply them with the appropriate input. However, the LOS also provides information about internal labels and reflects the relation between different variables via the tensor product representation. It is therefore not possible to reconstruct the LOS from Kozen's semantics.

Proposition 5. *Given a program P and an initial probabilistic state s_0 as a distribution over the program variables, let $[\![P]\!]$ be Kozen's semantics of P and $\mathbf{T}(P)$ the LOS. Then $(s_0 \otimes e_0)(\lim_{n\rightarrow\infty} \mathbf{T}^n)\mathbf{S}_f = s_0[\![P]\!]$.*

The proof follows by induction.

5 Semantics-based Analysis

Although the LOS semantics we presented here is of interest in itself its main motivation is to provide a basis for a semantics based program analysis. Classically the correctness of a program analysis is asserted with respect to the semantics in terms of a correctness relation. The theory of Abstract Interpretation allows for constructing analyses that are automatically correct without having to prove it a posteriori [22, 23]. The main applications of this theory are for the analysis of safety-critical systems as it guarantees correct answers at the cost of precision. For probabilistic systems or the probabilistic analysis of (non-)deterministic ones, the theory of Probabilistic Abstract Interpretation (PAI) allows for the construction of analyses that are possibly unsafe but maximally precise [9, 11]. Its main applications are therefore in fields like speculative optimisation and the analysis of trade-offs. PAI has been used for the definition of various analyses based on the LOS (see e.g. [3, 24, 25] and all involving finite-dimensional spaces.

PAI relies on the notion of generalised (or pseudo-)inverse. This notion is well-established in mathematics where it is used for finding approximate, so-called least-square solutions (cf. e.g. [26]).

Definition 3. *Let $\mathcal{H}_1$ and $\mathcal{H}_2$ be two Hilbert spaces and $\mathbf{A} : \mathcal{H}_1 \mapsto \mathcal{H}_2$ a linear map between them. A linear map $\mathbf{A}^\dagger = \mathbf{G} : \mathcal{H}_2 \mapsto \mathcal{H}_1$ is the* Moore-Penrose pseudo-inverse *of $\mathbf{A}$ iff $\mathbf{A} \circ \mathbf{G} = \mathbf{P_A}$ 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 $\mathbf{G}$.*

A linear operator $\mathbf{P} : \mathcal{H} \rightarrow \mathcal{H}$ is an *orthogonal projection* if $\mathbf{P}^* = \mathbf{P}^2 = \mathbf{P}$, where $(.)^*$ denotes the *adjoint*. The adjoint is defined implicitly via the condition $\langle x \cdot \mathbf{P}, y\rangle = \langle x, y \cdot \mathbf{P}^*\rangle$ for all $x, y \in \mathcal{H}$. For real matrices the adjoint correspond simply to the transpose matrix $\mathbf{P}^* = \mathbf{P}^t$ [19, Ch 10].

If $\mathcal{C}$ an $\mathcal{D}$ are two Hilbert spaces, and $\mathbf{A} : \mathcal{C} \rightarrow \mathcal{D}$ and $\mathbf{G} : \mathcal{D} \rightarrow \mathcal{C}$ are linear operators between $\mathcal{C}$ and $\mathcal{D}$, such that $\mathbf{G}$ is the Moore-Penrose pseudo-inverse of $\mathbf{A}$, then we say that $(\mathcal{C}, \mathbf{A}, \mathcal{D}, \mathbf{G})$ forms a *probabilistic abstract interpretation*, with $\mathcal{C}$ the concrete domain and $\mathcal{D}$ the abstract one.

Important for the applicability of PAI is the fact that it possesses some nice compositionality properties. These allow us to construct the abstract semantics $\mathbf{T}(P)^\#$ by abstracting the single blocks of the concrete semantics $\mathbf{T}(P)$ as follows:

$$\mathbf{T}(P)^{\#} = \mathbf{A}^{\dagger}\mathbf{T}(P)\mathbf{A} = \mathbf{A}^{\dagger}(\sum_{(\ell,\underline{\ell}')\in\mathcal{F}(P)} [\![[B]^{\ell}]\!] \otimes \mathbf{E}_{\ell\ell'})\mathbf{A} =$$
$$= \sum_{(\ell,\underline{\ell}')\in\mathcal{F}(P)} (\mathbf{A}^{\dagger}[\![[B]^{\ell}]\!]\mathbf{A}) \otimes \mathbf{E}_{\ell\ell'} = \sum_{(\ell,\underline{\ell}')\in\mathcal{F}(P)} [\![[B]^{\ell}]\!]^{\#} \otimes \mathbf{E}_{\ell\ell'},$$

where, for simplicity, we do not distinguish between the positive and negative semantics of blocks, and we assume that $\mathbf{A}$ does not abstract $\mathbf{E}_{\ell\ell'}$. The fact that we can work with the abstract semantics of individual blocks instead of the full operator obviously reduces the complexity of the analysis substantially.

Another important fact is that the Moore-Penrose pseudo-inverse of a tensor product can be computed as $(\mathbf{A}_1 \otimes \mathbf{A}_2 \otimes \ldots \otimes \mathbf{A}_v)^{\dagger} = \mathbf{A}_1^{\dagger} \otimes \mathbf{A}_2^{\dagger} \otimes \ldots \otimes \mathbf{A}_v^{\dagger}$ [26, 2.1,Ex 3]. We can therefore abstract properties of individual variables and then combine them in the global abstraction. This is also made possible by the definition of the concrete LOS semantics which is heavily based on the use of tensor product. Typically we have $[\![[B]^{\ell}]\!] = (\bigotimes_{i=1}^{v} \mathbf{T}_{i\ell}) \otimes \mathbf{E}_{\ell\ell'}$ or a sum of a few of such terms. The $\mathbf{T}_{i\ell}$ represents the effect of $\mathbf{T}(S)$, and in particular of $[\![[B]^{\ell}]\!]$, on variable i at label ℓ (both labels and variables only form a finite set). For example, we can define an abstraction $\mathbf{A}$ for one variable and apply it individually to all variables (e.g. extracting their even/odd property), or use different abstractions for different variables (maybe even forgetting about some of them by using $\mathbf{A}_f = (1, 1, \ldots)^t$) and define $\mathbf{A} = \bigotimes_{i=1}^{v} \mathbf{A}_i$ such that $\mathbf{A}^{\dagger} = \bigotimes_{i=1}^{v} \mathbf{A}_i^{\dagger}$ in order to get an analysis on the full state space.

Clearly, for (countably) infinite value spaces $\mathbb{X}$ the abstraction maps which we use in the construction of Probabilistic Abstract Interpretations are often also represented by unbounded operators (similar to the $\mathbf{U}(c)$ of Example 1). The use of weak limits will again help us in order to construct the Moore-Penrose pseudo-inverse $\mathbf{A}^{\dagger}$. Fortunately, the approximations by finite dimensional abstractions $\mathbf{A}_n$ and $\mathbf{A}_n^{\dagger}$ converge weakly for closed operators and in particular if the range of the abstraction is finite dimensional, i.e. if $\mathbf{A} : \ell_2(\mathbb{X}) \to \mathbb{R}^n$. Various general results of operator theory and linear algebra as found, for example in [27–29] offer a rigorous support for extending PAI to infinite-dimensional Hilbert spaces. We have dedicated a companion paper to a full treatment of this infinite case [30].

6 Conclusions

We have introduced a linear operator semantics (LOS) for probabilistic programs based on infinite-dimensional Hilbert space. We have shown how weak limits can be used to guarantee the existence of observable program properties, even for unbounded operators. In contrast with the norm limit on Banach spaces used in the work by Kozen, we are able to capture properties of the intermediate states of a program execution. This is important for program analysis. In fact, the aim of the work presented here is to provide a mathematically sound framework for probabilistic program analysis. The two main elements for this are (i) a compositionally defined semantics, i.e. LOS, and (ii) a way to reduce the concrete semantics in order to obtain a more manageable abstract one via PAI. The concepts of a linear operator semantics and probabilistic abstract interpretation have been used before in the setting of *finite* domains in [3, 24, 31, 25] for the analysis of programs and security properties. This paper extends LOS to infinite (concrete and abstract) domains and informally shows how PAI can be extended accordingly.

The LOS is closely related to various models used in Performance Analysis, like Stochastic Automata Networks (SAN) [32, 33]. As performance models are often based on Continuous Time Markov Chains (CTMC) it would be interesting to develop a continuous time version of the LOS which might help to establish a bridge between performance and program analysis.

From a semantical point of view one important feature of LOS is the notion of duality between states and observables and a weak limit construction to overcome problems with unboundedness. For a more direct approach it would be interesting to investigate which programs lead directly to bounded operators in the LOS approach. It seems that this issue is related to reversibility and finite branching of the reverse computation: for infinite (un-oriented) graphs it is known that the adjacency operator represents a bounded operator on ℓ_2 if and only if it is finitely branching [34, Thm. 3.1]. Similarly, it is also possible to model reversible Markov Chains, e.g. [16], via bounded Hilbert space operators. We aim to explore these aspects further in future work.

References

1. Baier, C., Haverkort, B.R., Hermanns, H., Katoen, J.P.: Performance evaluation and model checking join forces. Commun. ACM **53**(9) (2010) 76–85
2. Di Pierro, A., Hankin, C., Wiklicky, H.: Probabilistic timing covert channels: To close or not to close? Int. J. of Inf. Security **10**(2) (2011) 83–106
3. Di Pierro, A., Sotin, P., Wiklicky, H.: Relational analysis and precision via probabilistic abstract interpretation. In: QAPL'08. Volume 220(3) of ENTCS., Elsevier (2008) 23–42
4. Kozen, D.: Semantics of probabilistic programs. J. Comp. Sys. Sci. **22**(3) (1981) 328–350
5. Nielson, F., Nielson, H.R., Hankin, C.: Principles of Program Analysis. Springer (1999)
6. Kinder, J.: Towards static analysis of virtualization-obfuscated binaries. In: 19th Working Conference on Reverse Engineering, WCRE'12, IEEE (2012) 61–70
7. Lighthill, M.: Introduction to Fourier Analysis and Generalised Functions. Cambridge University Press (1958)
8. Lax, P.D.: Functional Analysis. John Wiley & Sons (2002)
9. Di Pierro, A., Wiklicky, H.: Concurrent Constraint Programming: Towards Probabilistic Abstract Interpretation. In: PPDP'00, ACM (2000) 127–138
10. Di Pierro, A., Hankin, C., Wiklicky, H.: Probabilistic semantics and analysis. In: Formal Methods for Quantitative Aspects of Programming Languages. Volume 6155 of LNCS. Springer (2010) 1–42
11. Di Pierro, A., Wiklicky, H.: Measuring the precision of abstract interpretations. In: LOPSTR'00. Volume 2042 of LNCS., Springer Verlag (2001) 147–164

12. Kubrusly, C.S.: The Elements of Operator Theory. second edn. Birkhäuser (2011)
13. Klenke, A.: Probability Theory - A Comprehensive Course. Springer Verlag (2006)
14. Plotkin, G.: A structured approach to operational semantics. Technical Report DAIMI FN-19, Computer Science Department, Aarhus University (1981)
15. Seneta, E.: Non-negative Matrices and Markov Chains. Springer Verlag (1981)
16. Woess, W.: Denumerable Markov Chains. EMS (2009)
17. Kadison, R., Ringrose, J.: Fundamentals of the Theory of Operator Algebras: Elementary Theory. AMS (1997) reprint from Academic Press edition 1983.
18. Fabian, M., Habala, P., Hájek, P., Montesinos, V., Zizler, V.: Banach Space Theory – The Basis for Linear and Nonlinear Analysis. Springer (2011)
19. Roman, S.: Advanced Linear Algebra. second edn. Springer (2005)
20. Davey, B., Priestley, H.: Introduction to Lattices and Order. Cambridge University Press, Cambridge (1990)
21. Kozen, D.: A probabilistic PDL. J. Comp. Sys. Sci. **30**(2) (1985) 162–178
22. Cousot, P., Cousot, R.: Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In: POPL'77. (1977) 238–252
23. Cousot, P., Cousot, R.: Systematic Design of Program Analysis Frameworks. In: POPL'79. (1979) 269–282
24. Di Pierro, A., Hankin, C., Wiklicky, H.: A systematic approach to probabilistic pointer analysis. In: APLAS'07. Volume 4807 of LNCS., Springer (2007) 335–350
25. Di Pierro, A., Hankin, C., Wiklicky, H.: Probabilistic timing covert channels: to close or not to close? Int. Journal of Inform. Security **10**(2) (2011) 83–106
26. Ben-Israel, A., Greville, T.N.E.: Gereralized Inverses – Theory and Applications. second edn. CMS Books in Mathematics. Springer, New York (2003)
27. Groetsch, C.W.: Stable Approximate Evaluation of Unbounded Operators. Volume 1894 of Lecture Notes in Mathematics. Springer (2007)
28. Du, N.: Finite-dimensional approximation settings for infinite-dimensional Moore-Penrose inverses. SIAM Journal of Numerical Analysis **46**(3) (2008) 1454–1482
29. Kulkarni, S., Ramesh, G.: Projection methods for computing Moore-Penrose inverses of unbounded operators. Indian Journal of Pure and Applied Mathematics **41**(5) (2010) 647–662
30. Di Pierro, A., Wiklicky, H.: Probabilistic analysis of programs: A weak limit approach. Available at http://fopara2013.cs.unibo.it/Proceedings.pdf (2013) [Online Informal Pre-proceedings].
31. Di Pierro, A., Hankin, C., Wiklicky, H.: Measuring the confinement of probabilistic systems. Theoretical Computer Science **340**(1) (2005) 3–56
32. Plateau, B., Atif, K.: Stochastic automata network of modeling parallel systems. IEEE Trans. Softw. Eng. **17**(10) (1991) 1093–1108
33. Fourneau, J.M., Plateau, B., Stewart, W.: Product form for stochastic automata networks. In: Proceedings of ValueTools '07, ICST (2007) 32:1–32:10
34. Mohar, B., Woess, W.: A survey on spectra of infinite graphs. Bulletin of the London Mathematical Society **21** (1988) 209–234