Probabilistic Analysis of Programs: A Weak Limit Approach

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

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

Abstract. We present an approach to probabilistic analysis which is based on program semantics and exploits the mathematical properties of the semantical operators to ensure a form of optimality for the analysis. As in the algorithmic setting, where the analysis results are used the help the design of efficient algorithms, the purposes of our framework are to offer static analysis techniques usable for resource optimisation.

1 Introduction

In probabilistic analysis we can distinguish two different situations, namely the case in which the program is deterministic and the input data varies according to some probability distribution, and the case in which the program is probabilistic and we evaluate its behaviour in different executions determined by the probability distribution on the same input data.

In this paper we present a general framework for the probabilistic static analysis of quantitative properties, that can cover both aspects and in particular includes the treatment of average case analyses as a special instance of a wider range of applications centred on the quantification of resource consumption. This is similar to the algorithmic setting where probabilistic analysis, or average-case analysis, refers to the estimation of the computational complexity of an algorithm starting from an assumption about the probabilistic distribution on the set of all possible inputs; the results of such an analysis are then used for the design of efficient algorithms. The approach we follow is a semantics based analysis where the program semantics is defined formally so as to express the various resources via the observable behaviour of the program.

As the possible behaviours of a given program are often 'too many', the complexity of the analysis of the possible executions of some code often suffers from the problem of combinatorial explosion (even when un-decidability is not involved). At the centre of numerous approaches to program analysis is therefore the attempt of a "simplification" or "abstraction" of programs and their possible executions, in particular the abstraction of the concrete state space to a substantially simpler one.

In previous work [1–3] we have introduced a framework for probabilistic analysis based on *least square approximation*, which achieves such a simplification. We have called the general methodology Probabilistic Abstract Interpretation

(PAI) for its strong analogy with the theory of Abstract Interpretation [4]. PAI aims in constructing statistical or average estimates of program properties which are usable for resource optimisation, in the same way as in algorithmic complexity a probabilistic analysis might aim at estimating the average-case computational complexity of an algorithm to the purpose of improving its efficiency.

Contrary to the classical abstract interpretation based static analysis, PAI-based analyses are not necessarily safe, i.e. it is not guaranteed that all accepted behaviours conform a given specification. They are rather guaranteed to be as close as possible to the concrete properties. This is because the objective of a PAI based analysis is 'performance' rather than 'correctness', and applications are the optimisation of resource usage rather than the construction of bounds for the (minimal or maximal) probability that something happens. As very often probability bounds tend to be 0 or 1 they are only of limited use in a context where optimisation is the main issue.

In previous work we have studied various aspects of the PAI framework for large but still *finite* state spaces. The central contribution of this paper is to extend PAI to *infinite* state spaces, by facing the mathematical problems which arise in this case and showing how they can be overcome. In particular, we will present a weak limit construction for abstraction operators on infinite domains that will allow us to deal with the problem of the unboundedness of such operators.

Mathematical Background. 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 [5] with regard to functional analytical and operator algebraic concepts. We will only recall here some basic notions that are essential for the comprehension of the results we are going to present.

The concrete Banach and Hilbert spaces we consider here are spaces ℓ_p of infinite sequences of real numbers $(x_i)_{i\in\mathbb{N}}$ for which $(\sum_{i\in\mathbb{N}} |x_i|^p)^{1/p} < \infty$ with $p = 1$ and $p = 2$, respectively. The space ℓ_1 is the standard example of a Banach space; in particular it contains (as the set of positive normalised elements) all probability distributions on $\mathbb{N}$. The space ℓ_2 is the standard example of a real Hilbert space with inner product $\langle (x_i), (y_i)\rangle = \sum_i x_i y_i$.

We also recall the definition of the three main topologies on Hilbert spaces, namely the norm or uniform convergence, denoted $\mathbf{A}_n \to \mathbf{A}$ or $\lim_n \mathbf{A}_n = \mathbf{A}$, the strong operator topology, $\mathbf{A}_n \xrightarrow{s} \mathbf{A}$ or $s\text{-}\lim_n \mathbf{A}_n = \mathbf{A}$ and the weak topology $\mathbf{A}_n \xrightarrow{w} \mathbf{A}$ or $w\text{-}\lim_n \mathbf{A}_n = \mathbf{A}$ defined by (cf. e.g. pag. 378 and [5, Def 4.45]):

$$\begin{aligned}
&\mathbf{A}_n \to \mathbf{A} \text{ iff } \|\mathbf{A}_n - \mathbf{A}\| \to 0\\
&\mathbf{A}_n \xrightarrow{s} \mathbf{A} \text{ iff } \|(\mathbf{A}_n - \mathbf{A})(x)\| \to 0\\
&\mathbf{A}_n \xrightarrow{w} \mathbf{A} \text{ iff } \langle (\mathbf{A}_n - \mathbf{A})(x), y\rangle \to 0
\end{aligned}$$

for all $x, y \in \mathcal{H}$. Note that the norm topology is defined in terms of the operator norm, while the strong topology in terms of the vector norm on $\mathcal{H}$. We have $\mathbf{A}_n \to \mathbf{A}$ implies $\mathbf{A}_n \xrightarrow{s} \mathbf{A}$ and $\mathbf{A}_n \xrightarrow{s} \mathbf{A}$ implies $\mathbf{A}_n \xrightarrow{w} \mathbf{A}$, but not vice versa.

Three definitions regarding linear operators on Hilbert spaces will play an important role in the next, namely those of *normally solvable*, *densely defined*

and *closed* operators, which we will therefore recall here. We will write $\mathcal{L}(\mathcal{X},\mathcal{Y})$ for the set of all linear operators between $\mathcal{X}$ and $\mathcal{Y}$, and for $\mathbf{T} \in \mathcal{L}(\mathcal{X},\mathcal{Y})$, we will denote by $\mathcal{D}(\mathbf{T})$, $\mathcal{N}(\mathbf{T})$ and $\mathcal{R}(\mathbf{T})$ the domain, the null space and the range of $\mathbf{T}$, respectively. A linear operator $\mathbf{T} : \mathcal{X} \to \mathcal{Y}$ between two Hilbert spaces $\mathcal{X}$ and $\mathcal{Y}$ is *bounded* if $\|\mathbf{T}\| = \sup \|\mathbf{T}(x)\|/\|x\| < \infty$. For linear operators the concept of *continuity* is equivalent to the concept of *boundedness*, see, e.g. [5, Thm. 4.14]). A linear map $\mathbf{T} \in \mathcal{L}(\mathcal{X},\mathcal{Y})$ is normally solvable if its range is closed. A necessary and sufficient condition for $\mathbf{T}$ to be normally solvable is that its range coincide with the orthogonal complement of the null space of its linear adjoint $\mathbf{T}^*$. The linear map $\mathbf{T} \in \mathcal{L}(\mathcal{X},\mathcal{Y})$ is called *closed* if its graph $\mathit{\Gamma}_{\mathbf{T}} = \{(x, \mathbf{T}x) \in \mathcal{D}(\mathbf{T}) \times \mathcal{Y} \subset \mathcal{X} \times \mathcal{Y}\}$ is closed in $\mathcal{X} \times \mathcal{Y}$. Equivalently, $\mathbf{T}$ is closed if and only if for any sequence $x_n \in \mathcal{D}(\mathbf{T}) \subseteq \mathcal{X}$, with $x_n \to x$ and $\mathbf{T}(x_n) \to y$ this implies that $y = \mathbf{T}(x)$, cf. [5, 6]. $\mathbf{T}$ is said to be densely defined if $\mathcal{D}(\mathbf{T})$ is dense in $\mathcal{X}$, that is $\overline{\mathcal{D}(\mathbf{T})} = \mathcal{X}$, where the notation $\overline{X}$ indicates the topological closure of the space X.

2 The Language

We will discuss our framework by referring to a simple core language. This is in essence the language used by Kozen in [7]. In this section we introduce both the syntax and the semantics for this language, which we call **pWhile**.

2.1 Syntax

In a style typical of static analysis [8], we introduce a labelled version of the **pWhile** language, where labels are used to identify the programs points that are crucial for defining our formal semantics.

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

We 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 $x := f(x_1, \ldots, x_n)$, sometimes also in the form $x := e$.

Then we have the random assignment $x \texttt{ ?= } \rho$ where the value of a variable is set to a value according to some random distribution ρ. In [7] 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. 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.

2.2 Linear Operator Semantics

We will base our probabilistic analysis on a syntax-based semantics which represents the executions of a program S as a Discrete Time Markov Chain (DTMC). More precisely, we associate to S a linear operator $\mathbf{T}(S)$ corresponding to the generator of the DTMC associated to S. This is a possibly infinite matrix whose domain is the set of *probabilistic configurations*, i.e. distributions on classical configurations $(x_1, \ldots, x_v, \ell)$ as row vectors, which record the value of all variables and the current label, defined by $\mathbf{Dist}(\mathbf{Conf}) = \mathbf{Dist}(\mathbb{X}^v \times \mathbf{Lab}) \subseteq \ell_2(\mathbb{X}^v \times \mathbf{Lab})$. Using the *tensor product* (e.g. [9, Chap. 14] or [10, Chap. 2.6]) we can exploit the fact that $\ell_2(\mathbb{X}^v \times \mathbf{Lab}) = \ell_2(\mathbb{X})^{\otimes v} \otimes \ell_2(\mathbf{Lab})$. We refer to $s \in \mathbf{Var} \to \mathbb{X} = \mathbb{X}^v$ as a *classical state* and to $\sigma \in \mathbf{Dist}(\mathbf{Var} \to \mathbb{X}) \subseteq \ell_2(\mathbb{X})^{\otimes v}$ as *probabilistic state*. In this definition we assume that 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 $\mathbb{N}$).

The labelled version of the syntax introduced in Section 2.1 allows us to use labels as a kind of program counter. The control flow $\mathcal{F}(S)$ in a program S is then defined via a function *flow* : $\mathbf{Stmt} \to \mathcal{P}(\mathbf{Lab} \times \mathbf{Lab})$ which maps statements to sets of pairs which represent the control flow graph, e.g. [8, Sect. 2.1] or [11]. This only records that a certain control flow step is possible. For tests $[b]^\ell$ 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. For this we will add a single final loop via a virtual label ℓ^* at the end of the program.

The construction of $\mathbf{T}(S)$ is done compositionally by using among its building blocks simple operators such as the *identity matrix* $\mathbf{I}$ and the *matrix units* $\mathbf{E}_{ij}$ containing only a single non zero entry $(\mathbf{E}_{ij})_{ij} = 1$. We also define for any Boolean expression b on $\mathbb{X}$ a diagonal *projection matrix* $\mathbf{P}(b)$ with $(\mathbf{P}(b))_{ii} = 1$ if $b(i)$ holds and 0 otherwise. The operator $\mathbf{P}(s)$ tests for a classical state s, i.e. if each variable $\mathtt{x}_i$ has the value $s(\mathtt{x}_i)$, and $\mathbf{P}(e = c)$ whether an expression e evaluates to a constant c. The *update* operator $\mathbf{U}$ implements state changes. The matrix $\mathbf{U}(c)$ implements the deterministic update of a variable to a constant c via $(\mathbf{U}(c))_{nm} = 1$ if $m = c$ and 0 otherwise. The operator $\mathbf{U}(x_k \leftarrow e)$ makes sure that the k^{th} variable $\mathtt{x}_k$ is assigned the value of the expression e

The matrix $\mathbf{T}(S)$ of the DTMC representing the program's executions is then defined as the sum of the effects of the individual control flow steps, i.e. the computational effect of each (labelled) block $[B]^\ell$ – with $B = \mathtt{skip}$, a test b or a (random) assignment – and a control flow step of the form $\mathbf{E}_{\ell\ell'}$.

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

If we also consider the final loop we have to add the term $\mathbf{I} \otimes \mathbf{E}_{\ell^*,\ell^*}$. The definition of the semantics of the individual blocks is given in Table 2.2.

Although the operator $\mathbf{T}(S)$ is in general not bounded (e.g. consider just $\mathbf{U}(\mathtt{x} \leftarrow 1)$), we can guarantee that it converges *weakly* for any initial state and

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

$$\mathbf{P}(e = c) = \sum_{\mathcal{E}(e)s=c} \mathbf{P}(s) \qquad \mathbf{U}(\mathtt{x}_k \leftarrow e) = \sum_{c} \mathbf{P}(e = c)\mathbf{U}(\mathtt{x}_k \leftarrow c)$$

$$[\![[x \ \mathtt{:=}\ e]^{\ell}]\!] = \mathbf{U}(x \leftarrow e) \qquad [\![[v \ \mathtt{?=}\ \rho]^{\ell}]\!] = \textstyle\sum_{c \in \mathbb{X}} \rho(c)\mathbf{U}(x \leftarrow c)$$

$$[\![[b]^{\ell}]\!] = \mathbf{P}(b = \mathtt{false}) \qquad \underline{[\![[b]^{\ell}]\!]} = \mathbf{P}(b = \mathtt{true})$$

$$[\![[\mathtt{skip}]^{\ell}]\!] = \underline{[\![[\mathtt{skip}]^{\ell}]\!]} = \underline{[\![[x \ \mathtt{:=}\ e]^{\ell}]\!]} = \underline{[\![[v \ \mathtt{?=}\ \rho]^{\ell}]\!]} = \mathbf{I}$$

Table 1. Elements of the LOS

any observable (specified as vector distributions in $\ell_1 \subseteq \ell_2$). We have shown this in [11], to which we refer for a full treatment of the linear operational semantics (LOS).

3 Probabilistic Abstract Interpretation

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. This is possible because of the conditions defining a Galois connection (the mathematical structure underlying abstract interpretation), which guarantee that we do not lose safety by going back and forth between the two lattices of the concrete and the analysis domains – although we may lose precision [4, 8].

Probabilistic Abstract Interpretation was introduces in [1, 12] as an alternative theory leading to analyses that are possibly not safe but that are guaranteed to be as close as possible to the concrete properties. It is concerned with probabilistic analysis, i.e. an analysis intended to return quantitative answers about a program property, rather than a 'yes/no' answer.

We have shown in a number of papers how the PAI technique can be used for performing the probabilistic analysis of many problems typical of classical program analysis, such as data-flow and pointer analyses [2], and of security problems [13]. The treatment in this previous work is restricted to the special case of finite state spaces. In this paper we generalise the theory to infinite-dimensional Hilbert spaces, where it is necessary the consideration of some appropriate topological notions in order to correctly deal with possibly infinite abstraction operators (and their generalised inverses).

Both classical abstract interpretation and PAI are usually based on state abstraction, i.e. on the abstraction of the concrete semantical domain. If we can make sure that the final state contains information about the resource(s) we are interested in, then we can use static analysis techniques in order to perform a quantitative analysis of resources and resource consumption. In some

situations the computational state already describes the resources needed; for example, if we have a loop index which determines the number of iterations, then the final state(s) will automatically encode the time complexity of the program in question. If this is not the case, then we can always introduce additional variables (e.g. counters) which extend the state so as this information becomes available for a state based analysis via classical abstract interpretation (e.g. for a worst case analysis) or PAI (e.g. for an average case analysis). As an example, we have developed a 'language-based' complexity analysis of quicksort, showing how the time behaviour of a program implementing this sorting algorithm can be analysed with PAI (see Section 4.2).

3.1 Basic Definitions

The theory of Probabilistic Abstract Interpretation relies on the notion of generalised (or pseudo-)inverse. This notion is well-known in mathematics where it is used for finding approximate solutions to integral and differential equations [14, Chap 9]. The abstract concept of a generalised or pseudo-inverse was introduced by Moore in the 1920s and was then rediscovered by Penrose in the 1950s.

Definition 1. *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 $\mathbf{G}$.*

It is also possible to define the Moore-Penrose pseudo-inverse $\mathbf{A}^\dagger$ of an operator $\mathbf{A}$ without direct reference to orthogonal projections (cf. e.g. [15, Section 4.7]). For this we need the notion of the adjoint $\mathbf{A}^*$ of an operator $\mathbf{A} : \mathcal{H} \to \mathcal{H}$ on a Hilbert space; this is defined via the condition $\langle \mathbf{A}(x), y\rangle = \langle x, \mathbf{A}^*(y)\rangle$. Then the Moore-Penrose pseudo-inverse can be defined via the following four conditions: $\mathbf{A}\mathbf{A}^\dagger\mathbf{A} = \mathbf{A}$, $\mathbf{A}^\dagger\mathbf{A}\mathbf{A}^\dagger = \mathbf{A}^\dagger$, $(\mathbf{A}^\dagger\mathbf{A})^* = \mathbf{A}^\dagger\mathbf{A}$, and $(\mathbf{A}\mathbf{A}^\dagger)^* = \mathbf{A}\mathbf{A}^\dagger$. In this form it is also obvious that the Moore-Penrose pseudo-inverse (like a Galois connection) is indeed a pseudo-inverse.

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

3.2 Correctness vs Best Solution

The Moore-Penrose pseudo-inverse plays a similar role in Probabilistic Abstract Interpretation as Galois connections in the theory of classical Abstract Interpretation in helping with the problem of combinatory explosion: it allows us to simplify the semantics $\mathbf{T}(S)$ of a program S by considering the abstract semantics $\mathbf{T}^\# = \mathbf{A}^\dagger\mathbf{T}(S)\mathbf{A}$ instead of the concrete one.

However, the properties of the Moore-Penrose pseudo-inverse (e.g. [16, 14]) guarantee a different form of optimality of the abstractions (abstract semantics) we can construct via PAI. In particular, these properties ensure that PAI

abstractions are the *closest* ones to the concrete semantics one can construct. Here closeness is defined via the distance induced by the norm on the Hilbert space, thus the name of '*least square approximation*' which is often used to refer to this approximation notion. As a consequence, in our probabilistic setting the main aim of the analysis is to reduce the error margin rather than to 'err on the safe side', which would lead to *safe* abstractions in the usual sense, i.e. over- or under-approximations of the concrete semantics.

The choice of the Hilbert space ℓ_2 as the domain for the LOS semantics is partly motivated by the symmetry between observables and states (we recall that ℓ_2 is self-dual) but also by the fact that it provides a simple and mathematically well understood theory of best approximations by Moore-Penrose pseudo-inverses. A similar well-behaved theory does not exist in general Banach spaces like ℓ_1 which do not have an inner product and a notion of an adjoint operator $\mathbf{A}^*$. As a result the theory of approximations in Banach spaces can, in general, not even guarantee the existence of unique optima (e.g. [17]).

3.3 Compositionality of PAI

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

$$\mathbf{T}(S)^{\#} = \mathbf{A}^{\dagger}\mathbf{T}(S)\mathbf{A} = \mathbf{A}^{\dagger}\left(\sum [\![[B]^{\ell}]\!]\right)\mathbf{A} = \sum \left(\mathbf{A}^{\dagger}[\![[B]^{\ell}]\!]\mathbf{A}\right) = \sum [\![[B]^{\ell}]\!]^{\#}$$

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: $(\mathbf{A}_1 \otimes \mathbf{A}_2 \otimes \ldots \otimes \mathbf{A}_v)^{\dagger} = \mathbf{A}_1^{\dagger} \otimes \mathbf{A}_2^{\dagger} \otimes \ldots \otimes \mathbf{A}_v^{\dagger}$, cf [14, 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 (cf. Section 2.2) 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.

Usually we do not abstract the control step $\mathbf{E}_{\ell\ell'}$, although this would also be possible. Very often most of the $\mathbf{T}_{i\ell}$'s are just the identity matrix – expressing the fact that the variable in question is not involved in whatever happens at label ℓ. In this case we have $\mathbf{A}_i^{\dagger}\mathbf{I}\mathbf{A}_i = \mathbf{I}$, where the two identity operators are of the appropriate dimensions (maybe an infinite matrix on the concrete space, but a finite matrix on the abstract property space).

The abstractions of tests $[b]^\ell$, i.e. $\mathbf{P}(b)$, provide the basis for analysing *abstract branching probabilities* [18]. This can be done by constructing $\mathbf{A}^\dagger[\![[b]^\ell]\!]\mathbf{A}$ and $\mathbf{A}^\dagger[\![[\neg b]^\ell]\!]\mathbf{A}$. As these operators are projections, we can exploit the fact that $[\![[b]^\ell]\!]^\# = \mathbf{I} - [\![[\neg b]^\ell]\!]$ and compute in practice only one of them. The abstract operator $[\![[\neg b]^\ell]\!]^\#$ in effect estimates the chance of the test b succeeding in terms of the abstract properties. For example, with a parity analysis we obtain a (least square) estimate for the chance that in an conditional statement, for an even number we take the `then` branch rather than the `else` branch (cf. Example in Section 4). If an analytical (statistical) construction of $[\![[b]^\ell]\!]^\#$ is infeasible one might consider profiling information as a replacement of the exact solution.

3.4 Abstraction Operators

In classical program analysis [8] an extraction function $\alpha : C \to D$ is a function which associates to each element c of a concrete domain C an abstract description or property $d \in D$. In our framework for probabilistic analysis, we will consider Hilbert spaces for both concrete and abstract domains $\mathcal{C}$ and $\mathcal{D}$, respectively. In particular, for an extraction function $\alpha : C \to D$ we will construct the Hilbert spaces $\mathcal{C} = \ell_2(C)$ and $\mathcal{D} = \ell_2(D)$ generated by the base vectors $\{\boldsymbol{c}\}_{c\in C}$ and $\{\boldsymbol{d}\}_{d\in D}$. For finite C and D we can identify the two spaces with the finite dimensional vector spaces $\mathbb{R}^{|C|}$ and $\mathbb{R}^{|D|}$. We can then define a linear operator $\mathbf{A}_\alpha = \mathbf{A} : \ell_2(C) \to \ell_2(D)$ by mapping every $\boldsymbol{x} = \sum_c x_c$ to $\mathbf{A}(\boldsymbol{x}) = \mathbf{A}(\sum_c x_c\boldsymbol{c}) = \sum_c x_c\mathbf{A}(\boldsymbol{c})$, with $(\mathbf{A}(\boldsymbol{c}))_{\alpha(c)} = 1$ and $(\mathbf{A}(\boldsymbol{c}))_d = 0$ for all $d \neq \alpha(c)$. In effect, this means that we construct a matrix $\mathbf{A}$, which we refer to as *classification operator*, with rows enumerated by elements in C and columns enumerated by elements in D such that $(\mathbf{A})_{cd} = 1$ iff $\alpha(c) = d$ and 0 otherwise.

The following theorem shows that a necessary and sufficient condition for the existence of the Moore-Penrose inverse for a *bounded* linear operator $\mathbf{A} : \mathcal{H} \to \mathcal{H}$ on a Hilbert space $\mathcal{H}$ is that $\mathbf{A}$ is *normally solvable*, i.e. its range $R(\mathbf{A}) = \{\mathbf{A}x \mid x \in \mathcal{H}\}$ is closed. This implies that in particular all operators on a finite dimensional Hilbert space are Moore-Penrose invertible.

Proposition 1. *[15, Thm 4.24] A bounded operator $\mathbf{A}\in\mathcal{B}(\mathcal{H}))$ is Moore-Penrose pseudo-invertible, i.e. a unique $\mathbf{A}^\dagger$ exists, if and only if it is normally solvable. In this case $(\mathbf{A}^*\mathbf{A} + \mathbf{P})$ is invertible and $\mathbf{A}^\dagger = (\mathbf{A}^*\mathbf{A} + \mathbf{P})^{-1}\mathbf{A}^*$ where $\mathbf{P}$ is the orthogonal projection of $\mathcal{H}$ onto $N(\mathbf{A}) = \{x \mid \mathbf{A}(x) = o\}$ with o being the null vector.*

Typically D will be finite, but C could be countably infinite. In this case we are faced with the problem that $\mathbf{A}$ might not be bounded. Thus – though $\mathbf{A}$ represents a closed map because D is finite – we need to be more careful when we construct the Moore-Penrose pseudo-inverse.

Example 1 (Forgetful Abstraction). Let us consider the abstraction into an abstract domain containing only a single property $*$, i.e. $D = \{*\}$ and $\alpha(s) = *$ for all concrete values s. Essentially, this abstraction only tests for the existence

of the system. It might seem rather pointless, but it turns out to be useful in various contexts, for example to analyse only a particular variables property ignoring other program variables, (see e.g. [3]).

The matrix $\mathbf{A}_f$ representing the forgetful abstraction in the PAI framework is a single column matrix containing only 1s. This corresponds to the map $x \mapsto (\|x\|_1) \in \mathbb{R}^1 = \ell_2(\{*\})$. This matrix represents clearly an unbounded map $\ell_2 \to \mathbb{R}$: the vector $x = (x_i)_{i=0}^{\infty} = (1, \frac{1}{2}, \frac{1}{3}, \ldots)$ is in ℓ_2 but $\mathbf{A}_f(x) = (\|x\|_1) = (\infty)$.

However, one can easily check that the operator $\mathbf{A}_f$ of Example 1 is densely defined and closed. More concretely the following holds:

- $\mathbf{A}_f$ is not bounded on ℓ_2. To see this, take the vector $x = (x_i)_{i=0}^{\infty} = (1, \frac{1}{2}, \frac{1}{3}, \ldots)$ which is in ℓ_2 but for which we have $\mathbf{A}_f(x) = (\|x\|_1) = (\infty)$.
- $\mathbf{A}_f$ is defined on a dense subspace of ℓ_2, namely ℓ_1, since for all $x \in \ell_1$ the 1-dim vector $\mathbf{A}_f(x) = (\|x\|_1) \in \mathbb{R}^1$ is defined.
- $\mathbf{A}_f$ is closed (on ℓ_1). If we have a sequence of vectors $x_n \to x$ then $\mathbf{A}_f(x_n) = (\|x_n\|_1)$ converges to $\|x\|_1$ as ℓ_1 is complete and the 1-norm is continuous.

In general, the following theorem holds, which allows us to establish the existence of a Moore-Penrose pseudo-inverse also in the case of unbounded abstraction operators provided that certain conditions are satisfied.

Theorem 1. *[6, Thm2.12] If $\mathbf{A} : \mathcal{D} \subseteq \mathcal{H}_1 \to \mathcal{H}_2$ is a closed densely defined linear operator then $\mathbf{A}^\dagger : \mathcal{D}(\mathbf{A}^\dagger) = R(\mathbf{A}) + R(\mathbf{A})^\perp \to \mathcal{D}(\mathbf{A}) \cap N(\mathbf{A})^\perp$ is closed and densely defined. Moreover,*

(i) *$\mathbf{A}^\dagger$ is bounded if and only if $R(\mathbf{A})$ is closed.*
(ii) *$\mathbf{A}\mathbf{A}^\dagger(y) = \mathbf{P}_{\overline{R(\mathbf{A})}}(y)$ for all $y \in \mathcal{D}(\mathbf{A}^\dagger)$.*
(iii) *$\mathbf{A}^\dagger\mathbf{A}(y) = \mathbf{P}_{N(\mathbf{A})^\perp}(y)$ for all $y \in \mathcal{D}(\mathbf{A})$.*

Fortunately, for classification operators, i.e. the abstraction operators of the PAI framework, we can guarantee the existence of a unique Moore-Penrose pseudo-inverse. In fact, the following two results generalise the properties of the forgetful abstraction in Example 1 to any probabilistic abstraction.

Proposition 2. *For any countable set C and finite set D the classification operator $\mathbf{A} : \ell_2(C) \to \ell_2(D)$ corresponding to the extraction function $\alpha : C \to D$ is a densely defined closed linear operator.*

Proof. The representation of the abstraction operator $\mathbf{A}$ for any extraction function α is a (possibly infinite) *stochastic* matrix, i.e. the row sums are all 1. This matrix represents therefore a bounded operator on ℓ_1 with respect to the 1-norm on ℓ_1,. This is because stochastic matrices preserve the 1-norm $\|\mathbf{A}(x)\|_1 = \|x\|_1$.

Clearly, $\mathbf{A}$ is in general not defined for all vectors in ℓ_2 (cf. the operator $\mathbf{A}_f(x)$ in Example 1). However, all abstractions $\mathbf{A}$ are well-defined on ℓ_1 which is dense in ℓ_2 (e.g. [5]). In order to show that $\mathbf{A}$ is closed, we recall that $\|x\|_2 \leq \|x\|_1$ holds, cf. [19, Exercise 1.15]. Therefore, if we assume convergence of a sequence $\{x_n\}$ in the 1-norm, i.e. $\|x_n - x\|_1 \to 0$ then this implies that also $\|x_n - x\|_2 \to 0$

holds. Now, if $\mathbf{A}(x_n) \to y$ in $\mathbb{R}^d \subseteq \ell_2(D)$, because by hypothesis $\mathbf{A} \in \mathcal{B}(\ell_1, \mathbb{R}^d)$ is continuous with respect to the 1-norm, we have that $\|\mathbf{A}(x_n) - \mathbf{A}(x)\|_1 \to 0$, i.e. $\mathbf{A}(x_n) \to \mathbf{A}(x)$. Because $\mathbb{R}^d$ is a finite-dimensional Hilbert space, all norms are equivalent on it, and therefore $\mathbf{A}(x) = y$ must hold. □

Proposition 3. *For any countable set C and finite set D and any extraction map $\alpha : C \to D$, the corresponding probabilistic abstraction $\mathbf{A} : \ell_2(C) \to \ell_2(D)$ has a Moore-Penrose pseudo-inverse $\mathbf{A}^\dagger : \mathcal{D} \subseteq \ell_2(D) \to \ell_2(C)$.*

Proof. This follows from the fact that $\mathbf{A}$ is densely defined and closed (from Proposition 2) and Theorem 1. □

3.5 Construction of Infinite-dimensional Abstractions

For an abstraction operator $\mathbf{A}$, i.e. a classification matrix corresponding to an extraction function α on a finite dimensional space $\ell_2(C) = \mathbb{R}^{|C|}$, we can construct the Moore-Penrose pseudo-inverse by just transposing $\mathbf{A}$ and row-normalising the resulting transposed matrix. However, we cannot use the same construction for infinite abstractions even if by Proposition 3 we are guaranteed that $\mathbf{A}^\dagger$ does exist. In Example 1, it is clear that with a concrete finite space $C = \{c_1, \ldots, c_n\}$ we can construct

$$\mathbf{A}_f = (1, 1, \ldots, 1)^t \quad \text{and} \quad \mathbf{A}_f^\dagger = (\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n}).$$

However, if we extend this to the case of a countable infinite concrete space, e.g. $C = \mathbb{Z}$, then clearly the row-normalised transpose of an infinite version of $\mathbf{A}_f$ must be the zero (single-row) matrix.

Example 2 (Parity Abstraction). Consider as abstract and concrete domains $\mathcal{C} = \ell_2(\{0, \ldots, n\})$ and $\mathcal{D} = \ell_2(\{\text{even}, \text{odd}\})$. The abstraction operator $\mathbf{A}_p$ and its concretisation operator $\mathbf{G}_p = \mathbf{A}_p^\dagger$ corresponding to a *parity analysis* are represented by the following $(n+1) \times 2$ and $2 \times (n+1)$ matrices (assuming w.l.o.g. that n is odd)

$$\mathbf{A}_p = \begin{pmatrix} 1\,0\,1\,0\,\ldots\,0 \\ 0\,1\,0\,1\,\ldots\,1 \end{pmatrix}^t \qquad \mathbf{A}_p^\dagger = \begin{pmatrix} \frac{2}{n+1} & 0 & \frac{2}{n+1} & 0 & \ldots & 0 \\ 0 & \frac{2}{n+1} & 0 & \frac{2}{n+1} & \ldots & \frac{2}{n+1} \end{pmatrix}$$

The concretisation operator $\mathbf{A}_p^\dagger$ represents uniform distributions over the $\frac{n}{2}$ even numbers in the range $0, \ldots, n$ (as the first row) and the n odd numbers in the same range (in the second row). Clearly, if we increase the dimension n we encounter the same problems as with $\mathbf{A}_f$.

In the following, we address the problem of effectively constructing $\mathbf{A}^\dagger$ for any probabilistic abstraction $\mathbf{A}$. This would be essential for practical and computational purposes, not least the implementation of our analyses.

In numerical mathematics [20, Sect 12.1], there is a general theory related to the so-called *finite sections* or *projection* methods which aims in approximating

infinite dimensional operators on Hilbert spaces by means of finite-dimensional approximations. A general approximating setting for projection methods is defined in [21] for bounded linear operators $\mathbf{T} : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{X}$ and $\mathcal{Y}$ are Hilbert spaces over the same field (that, for our purposes, we can assume to be $\mathbb{R}$). In this setting, let $\{X_n\}$ and $\{Y_n\}$ be sequences of closed subspaces of $\mathcal{X}$ and $\mathcal{Y}$, respectively, which satisfy $\mathbf{P}_n \xrightarrow{s} \mathbf{I}_X$ and $\mathbf{Q}_n \xrightarrow{s} \mathbf{I}_Y$, where $\mathbf{P}_n = \mathbf{P}_{X_n}$ and $\mathbf{Q}_n = \mathbf{Q}_{Y_n}$ denote the orthogonal projections from $\mathcal{X}$ and $\mathcal{Y}$ onto X_n and Y_n, respectively. A generalisation of this setting was introduced in [22] where projection methods are defined that apply to densely defined and closed operators (thus to our abstraction operators). The method consists in constructing $\{Y_n\}_{n\in\mathbb{N}}$ as an increasing sequence of subspaces of $\mathcal{R}(\mathbf{T})$ such that $\overline{\cup_{n=1}^{\infty} Y_n} = \mathcal{R}(\mathbf{T})$, and $\{X_n\}_{n\in\mathbb{N}}$ as an increasing sequence of subspaces of the orthogonal complement $\mathcal{N}(\mathbf{T})^\dagger$ of the null space of $\mathbf{T}$ such that $\overline{\cup_{n=1}^{\infty} X_n} = \mathcal{N}(\mathbf{T})^\dagger$. These two sequences must be constructed so as to satisfy some particular properties that are defined in [22, Def. 3.2] and called *admissibility* of the sequences. If we now define $\mathbf{T}_n = \mathbf{P}_n\mathbf{T}\mathbf{Q}_n$ and $\hat{\mathbf{T}}_n = \mathbf{T}_{n|_{X_n}}$, then the method is said to be convergent wrt to a given admissible pair $\{\mathbf{P}_n, \mathbf{Q}_n\}$ if for all $y \in \mathcal{Y}$ (using post-multiplication):

$$y \cdot \mathbf{P}_n \hat{\mathbf{T}}_n^\dagger \longrightarrow y \cdot \mathbf{T}^\dagger.$$

By applying Theorem 3.4 in [22], we can show that for any abstraction operator $\mathbf{A}$, we can construct $\mathbf{A}^\dagger$ by applying the generalised projection method described above. In fact, the following proposition holds:

Proposition 4. *For any countable set C and finite set D, let $\mathbf{A} : \ell_2(C) \to \ell_2(D)$ be the classification operator corresponding to the extraction function $\alpha : C \to D$. Then the generalised projection method for $\mathbf{A}$ is convergent wrt the sequence $\{\mathbf{P}_n, \mathbf{Q}_n\}$ of projections corresponding to the subspaces $X_n = \mathcal{N}(\mathbf{A})_n^\dagger$ and $Y_n = \overline{\mathcal{R}(\mathbf{A})}_n$, respectively.*

The fact that for the abstraction operators constituting the PAI framework a Moore-Penrose pseudo-inverse is always defined and that it can be constructed efficiently (e.g. [23–25]), makes the PAI theory consistent, and guarantees a solid mathematical basis to the program analysis techniques based on it.

However, for the purposes of our analysis we are actually not really interested in the limit object $\mathbf{A}^\dagger$ itself; we are rather interested in the effect of the finite approximations of the abstract semantics defined via $\mathbf{A}$ and $\mathbf{A}^\dagger$ on the abstract state of a given program with respect to the property under consideration. This is captured by the notion of weak limit: if $\mathbf{T}(S)$ is the concrete (possibly unbounded) linear operator semantics for a given program S, x is an element (distribution) in the domain of the abstract operator $\mathbf{T}^\#(S) = \mathbf{A}^\dagger\mathbf{T}(S)\mathbf{A}$, and y is an abstract property (distribution) in the same domain, we are interested in analysing the behaviour of S by observing the abstract sequence of the inner products between the n-th approximation vector $x\cdot(\mathbf{T}^\#(S))_n$ and the observable y, i.e. the weak limit of the sequence with respect to that observable. We will show in the following that this sequence always converges in $\mathbb{R}$, and we will take this limit to define the *effect* of property y on state $x \cdot \mathbf{T}^\#(S)$.

For a program S and its LOS, $\mathbf{T}(S)$, we can define the finite approximations of $\mathbf{T}(S)$ as the operators $\mathbf{T}(S)_n = \mathbf{P}_n\mathbf{T}(S)\mathbf{P}_n$, where for all $n \in \mathbb{N}$, $\mathbf{P}_n$ is a diagonal matrix with all entries zero except for the first n diagonal entries that are all equal to 1. Then, for a given abstraction operator $\mathbf{A}$, we define the finite approximations $\mathbf{T}^{\#}(S)_n$ of the abstract semantics $\mathbf{T}^{\#}(S) = \mathbf{A}^{\dagger}\mathbf{T}(S)\mathbf{A}$ as $\mathbf{A}_n^{\dagger}\mathbf{T}(S)_n\mathbf{A}_n$, where $\mathbf{A}_n$ is the finite section of $\mathbf{A}$ defined in the projection method, for which we can easily construct $\mathbf{A}_n^{\dagger}$ (it is a finite matrix).

Proposition 5. *For any countable set C and finite set D and any classification map $\alpha : C \to D$, let $\mathbf{A}$ be the corresponding linear map $\mathbf{A} : \ell_2(C) \to \ell_2(D)$. Then for any program S and its LOS operator $\mathbf{T}(S)$ and for all distributions $x, y \in \mathbf{Dist}(\mathbf{Conf}) \subset \ell_2(D)$, we have that $\lim_{n\to\infty} \left\langle x \cdot \mathbf{A}_n^{\dagger}\mathbf{T}(S)_n\mathbf{A}_n, y \right\rangle < \infty$.*

Proof. Let $x_n = x \cdot \mathbf{T}^{\#}(S)_n$ 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 $\mathbf{T}^{\#}(S)_n$ are (sub-)stochastic matrices and $\|x\|_1 = 1$ (because x is a distribution), we have that $\|x_n\|_1 \leq 1$ and $\langle x_n, y\rangle \leq \langle x_{n+1}, y\rangle$, i.e. the sequence of the inner products is monotone. Moreover, by the Cauchy-Schwarz inequality (e.g. [10, Prop. 2.1.1]) we have that $\langle x_n, y\rangle \leq \|x_n\|_2\|y\|_2$. Thus, as in general $\|v\|_2 \leq \|v\|_1$ holds for all v (cf. [19, Exercise 1.14]), we have that $\langle x_n, y\rangle \leq \|x_n\|_2\|y\|_2 \leq \|x_n\|_1\|y\|_1 \leq 1$, i.e. $\langle x_n, y\rangle$ is a bounded, monotone sequence of reals, which thus converges. □

Example 3 (Approximation of $\mathbf{A}_f^{\dagger}$). If we construct the weak limit of finite approximations to $\mathbf{A}_f$ and consider its effect only in the context of a concrete state σ then we get a useful result even if we do not effectively construct the norm or uniform limit of $(\mathbf{A}_f)_n$. For any probabilistic state $\sigma \in \ell_1(C)$ with $\sigma_i \geq 0$ and $\|\sigma\|_1 = 1$, i.e. a probability distribution, we get for all $x \in \ell_2(C)$

$$\lim_{n\to\infty} \langle (\mathbf{A}_f)_n(\sigma), x\rangle = \mathbf{E}(x, \sigma) \quad \text{and} \quad \lim_{n\to\infty} \left\langle (\mathbf{A}_f^{\dagger})_n(*), x \right\rangle = \mathbf{E}(x, \nu),$$

where $\mathbf{E}(x, d)$ is the expectation value of x with respect to the distribution d and ν is the uniform distribution. In other words, the weak limit $\mathbf{A}_f^{\dagger} = w\text{-}\lim(\mathbf{A}_f)_n$ represents the effect of an eventually non-existing "uniform measure" on all of C, even when C is an infinite set. In measure theoretic terms this means that we represent a uniform measure (on all of C) which per se cannot be expressed as a distribution as the limit of distributions.

4 Examples

We conclude by discussing in detail an example which illustrates how probabilistic abstraction allows us to analyse the properties of programs. We also demonstrate how the PAI framework could be used for average case complexity analysis by re-phrasing the well-known probabilistic analysis of the Quicksort algorithm in our program analysis setting. We will also discuss the efficiency of the analysis, i.e. how PAI can be deployed in order to beat the combinatorial explosion or the curse of dimensionality.

4.1 Factorial

It is easy to observe that the factorial function $n!$ "almost always" returns an even number (except for 0! and 1!). If we perform a classical abstraction we cannot justify this intuition as in oder to be safe we can only obtain a guarantee that the result may be *even* or *odd*. In order to provide a formal analysis of $n!$ let us first consider the concrete semantics of the program F using labelling:

$$[m := 1]^1;\ \texttt{while}\ [n > 1]^2\ \texttt{do}\ [m := m \times n]^3;\ [n := n - 1]^4\ \texttt{od};\ [\texttt{skip}]^5$$

The flow is $\mathcal{F}(F) = \{(1,2),(2,\underline{3}),(3,4),(4,2),(2,5),(5,5)\}$ which includes a looping on the final `skip` statement. We then have: $\mathbf{T}(F) = \mathbf{U}(\texttt{m} \leftarrow \texttt{1}) \otimes \mathbf{E}_{1,2} + \mathbf{P}(\texttt{(n > 1)}) \otimes \mathbf{E}_{2,3} + \mathbf{U}(\texttt{m} \leftarrow \texttt{(m * n)}) \otimes \mathbf{E}_{3,4} + \mathbf{U}(\texttt{n} \leftarrow \texttt{(n - 1)}) \otimes \mathbf{E}_{4,2} + \mathbf{P}(\texttt{(n <= 1)}) \otimes \mathbf{E}_{2,5} + \mathbf{I} \otimes \mathbf{E}_{5,5}$.

If we just consider the factorials 0!, 1! and 2! then we can restrict ourselves to values $m, n \in \{0,1,2\}$. In this case the semantics of each block is given by a $3 \cdot 3 \times 3 \cdot 3 = 9 \times 9$ matrix.

For the updates in label 3 and 4 we have "empty rows", i.e. rows where we have no non-zero entries. These correspond to over- and under-flows as we are dealing only with finite values in $\mathbb{Z}$, e.g. the product for $m = 2$ and $n = 2$ is not in $\{0,1,2\}$. We could clarify the situation in various ways, e.g. by introducing an additional value $\perp$ for undefined (concrete) values of variables, or by introducing an `error` configuration. In the analysis we present here these over- and under-flows do not play any relevant role and we therefore leave things as they are.

The full operator representing the LOS semantics of the factorial program is given by a $(3 \cdot 3 \cdot 5) \times (3 \cdot 3 \cdot 5) = 45 \times 45$ matrix.

We can construct an abstract version $\mathbf{T}^\#(F) = \mathbf{A}^\dagger \mathbf{T}(F) \mathbf{A}$ of $\mathbf{T}(F)$ by recording only the parity of m as even and odd. We will not abstract n nor the labels defining the current configuration during the execution. We thus get the $(2 \cdot 3 \cdot 5) \times (2 \cdot 3 \cdot 5) = 30 \times 30$ matrix $\mathbf{T}^\#(F) = (\mathbf{A}_p \otimes \mathbf{I} \otimes \mathbf{I})^\dagger \mathbf{T}(F)(\mathbf{A}_p \otimes \mathbf{I} \otimes \mathbf{I})$. Though this abstract semantics does have some interesting properties, it appears to be only a minor improvement with regard to the concrete semantics: We managed to reduce the dimension only from 45 to 30. However, the simplification becomes substantially more dramatic once we increase the possible values of `m` and `n`, and combinatorial explosion really takes a hold. If we allow `n` to take values between 0 and n then we must allow for `m` values between 0 and $n!$. Concrete values of the dimensions of $\mathbf{T}(F)$ and $\mathbf{T}^\#(F)$ for $n = 1, 2, \ldots, 9$ are $\dim(\mathbf{T}(F)) = 45, 140, 625, 3630, 25235, 201640, 1814445, 18144050$ but for the abstract semantics only $\dim(\mathbf{T}^\#(F)) = 30, 40, 50, 60, 70, 80, 90, 100$.

The problem is that the size of $\mathbf{T}(F)$ explodes so quickly that it is impossible to simulate it for values of n much larger than 5 on a normal PC. If we want to analyse the abstract semantics, things remain much smaller. Importantly, we can construct the abstract semantics in the same way as the concrete one, just using "smaller" matrices: $\mathbf{T}^\#(F) = \mathbf{U}^\#(\texttt{m} \leftarrow \texttt{1}) \otimes \mathbf{E}_{1,2} + \mathbf{P}^\#(\texttt{(n > 1)}) \otimes \mathbf{E}_{2,3} + \mathbf{U}^\#(\texttt{m} \leftarrow \texttt{(m * n)}) \otimes \mathbf{E}_{3,4} + \mathbf{U}^\#(\texttt{n} \leftarrow \texttt{(n - 1)}) \otimes \mathbf{E}_{4,2} + \mathbf{P}^\#(\texttt{(n <= 1)}) \otimes \mathbf{E}_{2,5} + \mathbf{I}^\# \otimes \mathbf{E}_{5,5}$. Fortunately, most of the operators $\mathbf{T}^\#(\ell, \ell')$ are very easy to construct. These matrices are $2 \cdot (n+1) \cdot 5 \times 2 \cdot (n+1) \cdot 5 = 10(n+1) \times 10(n+1)$ matrices if we

consider the control transfer, and only $2(n+1) \times 2(n+1)$ matrices if we deal only with the update of the current state.

Except for label 3 only either `m` or `n` but never both are involved in each statement: We thus can express the $\mathbf{T}^{\#}(\ell, \ell')$'s as tensor products of a 2×2 and a $(n+1) \times (n+1)$ matrix. Finally, we need to construct the update for label 3. It is easy to see that for even `m` the result is again even and for odd `m` the parity of `n` determines the parity of the resulting `m`. We can thus write this update at label 3 as:

$$\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \otimes \mathbf{I} + \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \otimes \mathbf{P}(\mathbf{even}) + \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \otimes \mathbf{P}(\mathbf{odd})$$

where $\mathbf{I}$ is the identity matrix, and $\mathbf{P}(\mathbf{even})$ and $\mathbf{P}(\mathbf{odd})$ are the indicator projections for even and odd, e.g. $(\mathbf{P}(\mathbf{even}))_{ij} = 1$ if $i = j$ is even and 0 otherwise (with $i, j \in \{0, 1, \ldots\}$).

With this we can now approximate the probabilistic properties of the factorial function. In particular, if we look at the terminal configurations with the initial abstract configuration: $\boldsymbol{x}_0 = \left(\frac{1}{2}\ \frac{1}{2}\right) \otimes \left(\frac{1}{n+1} \cdots \frac{1}{n+1}\right) \otimes \left(1\,0\,0\,0\,0\right)$ which corresponds to a uniform distribution over all possible abstract values for our variables `m` and `n` (in fact, the part describing `m` could be any other distribution), then we get as final probabilistic configuration: $\boldsymbol{x} = \left(\frac{n-1}{n+1}\ \frac{2}{n+1}\right) \otimes \left(\frac{1}{n+1}\ \frac{n}{n+1}\ 0 \ldots 0\right) \otimes \left(0\,0\,0\,0\,1\right)$. This expresses the fact that indeed in most cases (with probability $\frac{n-1}{n+1}$) we get an even factorial – only in two cases out of $n+1$ (for 0 and 1) we get an odd result (namely 1). The final value of `n` is nearly always 1 except when we start with 0 and we always reach the final statement with label 5.

If we start with the abstract initial state $\boldsymbol{x}_0$ above and execute $\mathbf{T}^{\#}(F)$ until we get a fixpoint $\boldsymbol{x}$ we can use abstractions not to simplify the semantics but instead in order to extract the relevant information. Concretely we use: $\mathbf{A} = \mathbf{I} \otimes \mathbf{A}_f \otimes \mathbf{A}_f$, i.e. once we reached the terminal configuration (of the abstract execution) we ignore the value of n and the final label ℓ and only concentrate on the abstract, i.e. parity, values of m. Concretely we have to compute: $(\lim_{i\to\infty} \boldsymbol{x}_0 \cdot (\mathbf{T}^{\#}(F))^i) \cdot \mathbf{A}$. Note that we always reach the fixpoint after a finite number of iterations (namely at most n) so this can be computed in finite time. The concrete probabilities we get for various n are:

n	**even**	**odd**
10	0.81818	0.18182
100	0.98019	0.019802

n	**even**	**odd**
1000	0.99800	0.0019980
10000	0.99980	0.00019998

We can easily compute the final distribution on $\{\mathbf{even}, \mathbf{odd}\}$ for quite large n despite the fact that, as said, it is virtually impossible to compute the explicit representation of the concrete semantics $\mathbf{T}(F)$ already for $n = 6$.

Considering the factorial only for a limited number of inputs – here for n less than 10, 100, 1000, and 10000 – means in effect considering the finite approximations for the full factorial function F with any input $n \in \mathbb{Z}$. We are essentially computing

$$(\lim_{i\to\infty} \boldsymbol{x}_0 \cdot (\mathbf{T}^{\#}_n(F))^i) \cdot \mathbf{A} = (\lim_{i\to\infty} \boldsymbol{x}_0 \cdot (\mathbf{P}_n \mathbf{T}(F) \mathbf{P}_n)^i) \cdot \mathbf{A}$$

for $n = 10$, 100, 1000, and 10000. Numerically the last table of results shows that for $n \to \infty$ we obviously get $P(\mathbf{even}) = 1$, and $P(\mathbf{odd}) = 0$.

We see that with a uniform distribution for n we get a vanishing probability $P(\mathbf{odd}) = 0$ for m. But we could also execute the abstract program with different initial distributions to get different estimates for $P(\mathbf{odd})$. If we take, for example, as a distribution for n something like $(\frac{1}{2}, \frac{1}{2}, 0, 0, \ldots)$ – which would mean that we only need to compute 0! and 1! – we obviously get the result $P(\mathbf{even}) = 0$ for m. A worst-case, safe analysis would thus result in $0 \leq P(\mathbf{even}) \leq 1$ – which is certainly correct but trivial – while by an average case analysis providing instead optimal estimates (for different initial distributions) we can achieve more useful results.

4.2 Quicksort

QuickSort (due to C.A.R. Hoare) is one of the best known sorting algorithms. Its complexity theoretic properties are studied in practically all mongraphs on algorithms and complexity. It is well-known that its worst case time complexity is $O(n^2)$ and average case time complexity is $O(n \log n)$, where n is the length of the list to be sorted and it is assumed a uniform distribution over all permutations. We show here how our approach to probabilistic program analysis can be used to analyse the QuickSort algorithm. Our aim is not to provide a new complexity theoretic analysis of this algorithm but to illustrate how PAI can be used to gain some understanding of the average running time of QuickSort. In order to do this we consider the following simple recursive pseudo-code for QuickSort:

```
QuickSort (list,left,right)
  if (left<right) then pivot := Partition(list,left,right);
  Quicksort(list,left,pivot-1); Quicksort(list,pivot+1,right)
```

For the purposes of the analysis we concentrate on the partitioning step. Clearly, the number of calls to `Partition` determines the overall running time of `QuickSort`. To simplify the discussion we analyse the behaviour of a function `Split` which takes as input a list and splits it in two parts based on a given element p of the list (the pivot). In our implementation we take as pivot always the first element in the list. The effect of the splitting is a partition of the original list into a list of elements strictly smaller than p and a list of elements strictly larger than p. Function `Split` always returns the longest of the two lists.

We define the concrete semantics of `Split` via the corresponding transition matrix $\mathbf{T}$ on $\mathcal{V}(\bigcup_{j=1}^{n} S_j)$ where $S_j = \{\pi_i\}_{i=1}^{j!}$ is the set of all permutations of $\{1, 2, \ldots, j\}$ enumerated in some way. We denote by π_i^j the i-th permutation of length j. We observe that `Split` can produce (as longer list) either a list π which is a permutation of all elements smaller than the pivot p – i.e. an element in S_{p-1} – or a permutation of all elements larger than p, which we can also identify with an element in S_{n-p} by dropping p from all elements in π; we denote the resulting (maybe p 'shifted') list by $\lfloor \pi \rfloor$.

Let us denote by $n!! = \sum_{i=1}^{n} i!$. Given the enumeration of elements in any S_i, we enumerate the $n!!$ permutations in $\bigcup_{i=1}^{n} S_i$ by $\iota(\pi_i^k) = (k-1)!! + i$ for $\pi_i^k \in S_k$, where we set $\iota(\pi_1^1) = 1$ for the single permutation in S_1. `Split` can now be implemented as the reduction which takes a permutation $\pi_i^k \in S_k$ to a permutation π_j^l in S_l (with $l < n$). Formally, we define this reduction via an operator on $\mathbb{R}^{n!!} = \mathcal{V}(\{\bigcup_{i=1}^{n} S_i\})$ represented by an $n!! \times n!!$ matrix with entries (in row i and column j): $(\mathbf{P}(\pi))_{i,j} = 1$ if $\pi' = \lfloor \texttt{Split}(\pi) \rfloor$ with $i = \iota(\pi)$ and $j = \iota(\pi')$, and 0 otherwise. The operator $\mathbf{T}$, which encodes all reduction steps for lists of at most n elements under `Split`, is the linear operator $\mathbf{T} : \mathbb{R}^{n!!} \to \mathbb{R}^{n!!}$

$$\mathbf{T} = \sum_{k=2}^{n} \sum_{i=1}^{k!} \mathbf{P}(\pi_i^k) + \mathbf{E}_{1,1},$$

where the second term expresses looping on terminal states, i.e. the states where the list is reduced to a singleton, in order to stay within the framework of DTMCs. The concrete semantics of the `Split` function quickly becomes very large and difficult to understand. In order to perform our analysis we nevertheless just need to concentrate on the length of the longer list produced in the splitting process, and abstract away all the rest. This can be achieved by abstracting $\mathbf{T}$ via the operator $\mathbf{A} : \mathcal{V}(\{\bigcup_{i=1}^{n} S_n\}) \to \mathcal{V}(\{1, \ldots, n\})$ or $\mathbf{A} : \mathbb{R}^{n!!} \to \mathbb{R}^n$, represented by the $n!! \times n$ matrix with entries $(\mathbf{A})_{ij} = 1$ if $i = \iota(\pi)$ and $\pi \in S_j$, and 0 otherwise. This keeps the length and ignore the concrete permutation we get after splitting. The abstract semantics is therefore: $\mathbf{T}^{\#} = \mathbf{A}^{\dagger}\mathbf{T}\mathbf{A}$. Note that if we allow for lists of any length then $\mathbf{T}^{\#}$ is an infinite dimensional operator; its finite sections are given by restricting the space according to the maximal length of the list to which it is applied. More precisely, the n-th section of $\mathbf{T}^{\#}$ is the upper left sub-matrix of $\mathbf{T}^{\#}$ defined on $\mathbb{R}^n$. We can therefore express the average behaviour of `Split`, that is the average running time of `QickSort`, in terms of the expectation value $\langle e_n \mathbf{T}^{\#}, e_1 \rangle$, where $e_n = (0, 0, \ldots, 0, 1) \in \mathbb{R}^n$ represents all lists of length at most n and $e_1 = (1, 0, \ldots, 0)$ is the (final) point distribution representing all singleton lists reached at the end of the splitting process.

If we define $p_{\leq l} = \langle e_n (\mathbf{T}^{\#})^l, e_1 \rangle = \langle e_n (\mathbf{T}_n^{\#})^l, e_1 \rangle$ as the probability that `Split` terminates in l steps or less, then $p_l = p_{\leq l} - p_{\leq l-1}$ is the probability that `Split` terminates in exactly l steps. The average running time for lists of length n is thus $\sum_{l=0}^{\infty} l \cdot p_l = \lim_{n\to\infty} \sum_{l=0}^{n} l \cdot p_l = \lim_{n\to\infty} \sum_{l=0}^{n} l \langle e_n (\mathbf{T}^{\#})^{l-1} (\mathbf{T}^{\#} - \mathbf{I}), e_1 \rangle$ ..

From the numerical experiments it is easy to conjecture that a general form for the abstract operator $\mathbf{T}^{\#}$ is (verified with `octave` for $n = 1, .., 9$):

$$(\mathbf{T}^{\#})_{i,j} = \begin{cases} 1 \text{ for } i = j = 1 \\ \frac{2}{i} \text{ for } i = 2k \text{ and } k \leq j < i \text{ or for } i = 2k+1 \text{ and } k < j < i \\ \frac{1}{i} \text{ for } i = 2k+1 \text{ and } j = k \\ 0 \text{ otherwise} \end{cases}$$

Based on $\mathbf{T}^{\#}$ – more precisely its nth sections $\mathbf{T}_n^{\#}$ – we can numerically calculate the average running time (until repeated calls of `Split` terminate) for lists of length $n = 1 \ldots, 100$. It is easy to see from these figures that the

average running time increases roughly logarithmically. As we have simplified the problem by considering only the longer list, the results obtained this way do, of course, only indicate the actual complexity theoretic behaviour.

5 Related Work and Conclusions

The aim of this work is to provide a mathematically sound framework for probabilistic program analysis. The two main elements for this are (i) a compositionally defined semantics, called 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, 2, 13] for the analysis of programs and security properties. This paper extends PAI to infinite abstract domains.

Our LOS is closely related to a number of models which are popular in, for example, performance analysis, like Stochastic Automata Networks (SAN) [26, 27]. The idea of reducing the complexity of dynamical systems via least square approximations, which is at the core of our PAI approach, can also be found in various approaches ranging from Kalman filters, to model order reduction [28] and aggregation of Markov models [29]. However, to the best of our knowledge most of these models are finite dimensional.

While the theoretical framework of generalised inverses for finite-dimensional and bounded operators is well understood and relatively straight forward (e.g. [16, 14]), it is less well developed for the infinite dimensional unbounded case (see e.g. [6]). In the area of program analysis the well-known semantics for probabilistic programs by Kozen [7] and the application of classical abstract interpretation [30] to probabilistic languages are perhaps the closest alternatives to our LOS and PAI approach. The main differences are however that (i) the LOS – in contrast to Kozen's semantics – is not only able to capture the input/output behaviour but rather, because it uses the generator of a Markov Chain, defines the details and intermediate steps of a computational process, and (ii) PAI provides closest estimates to probabilities rather than worst case bounds for them. This makes PAI a more useful alternative in all those cases where the expected outcomes of a static analysis is not a yes-or-no answer but some estimates on which to base a speculative optimisation (compiler design, tradeoffs and cost analysis, etc). Moreover, as we have shown in Section 4.2, PAI can be used to develop (semi)-automatic tools for average-case complexity analysis.

References

1. Di Pierro, A., Wiklicky, H.: Concurrent Constraint Programming: Towards Probabilistic Abstract Interpretation. In: PPDP'00, ACM (2000) 127–138
2. 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
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. Cousot, P., Cousot, R.: Systematic Design of Program Analysis Frameworks. In: Proceedings of POPL'79. (1979) 269–282
5. Kubrusly, C.S.: The Elements of Operator Theory. second edn. Birkhäuser (2011)
6. Groetsch, C.W.: Stable Approximate Evaluation of Unbounded Operators. Volume 1894 of Lecture Notes in Mathematics. Springer (2007)
7. Kozen, D.: Semantics of probabilistic programs. J. Comput. Syst. Sci. **22**(3) (1981) 328–350
8. Nielson, F., Nielson, H.R., Hankin, C.: Principles of Program Analysis. Springer (1999)
9. Roman, S.: Advanced Linear Algebra. second edn. Springer (2005)
10. Kadison, R., Ringrose, J.: Fundamentals of the Theory of Operator Algebras: Elementary Theory. AMS (1997) reprint from Academic Press edition 1983.
11. Di Pierro, A., Wiklicky, H.: Semantics of probabilistic programs: A weak limit approach. In: Proc. APLAS'13. Volume 8301 of LNCS., Springer (2013) 241–256
12. Di Pierro, A., Wiklicky, H.: Measuring the precision of abstract interpretations. In: Proc. of LOPSTR'00. Volume 2042 of LNCS., Springer (2001) 147–164
13. Di Pierro, A., Hankin, C., Wiklicky, H.: Measuring the confinement of probabilistic systems. Theoretical Computer Science **340**(1) (2005) 3–56
14. Ben-Israel, A., Greville, T.N.E.: Gereralized Inverses – Theory and Applications. second edn. CMS Books in Mathematics. Springer, New York (2003)
15. Böttcher, A., Silbermann, B.: Introduction to Large Truncated Toeplitz Matrices. Springer (1999)
16. Deutsch, F.: Best Approximation in Inner-Product Spaces. Springer (2001)
17. Pinkus, A.M.: On L^1-Approximation. Cambridge University Press (1989)
18. Di Pierro, A., Wiklicky, H.: Probabilistic data flow analysis: a linear equational approach. In: Proc. GandALF'13. Volume 119 of EPTCS. (2013) 150–165
19. Fabian, M., Habala, P., Hájek, P., Montesinos, V., Zizler, V.: Banach Space Theory – The Basis for Linear and Nonlinear Analysis. Springer (2011)
20. Atkinson, K., Han, W.: Theoretical Numerical Analysis – A Functional Analysis Framework. third edn. Springer (2009)
21. Nailin, D.: Finite-dimensional approximation settings for infinite-dimensional Moore-Penrose inverses. SIAM J. of Numerical Analysis **46**(3) (2008) 1454–1482
22. Kulkarni, S., Ramesh, G.: Projection methods for computing Moore-Penrose inverses of unbounded operators. Indian J. Pure Appl.Math. **41**(5) (2010) 647–662
23. Groetsch, C.: Spectral methods for linear inverse problems with unbounded operators. Journal of Approximation Theory **70** (1992) 16–28
24. Groetsch, C.: Dykstra's algorithm and a representation of the Moore-Penrose inverse. Journal of Approximation Theory **117** (2002) 179–184
25. Groetsch, C.: An iterative stabilization method for the evaluation of unbounded operators. Proceedings of the AMS **134** (2005) 1173–1181
26. Plateau, B., Atif, K.: Stochastic automata network of modeling parallel systems. IEEE Trans. Softw. Eng. **17**(10) (1991) 1093–1108
27. Fourneau, J.M., Plateau, B., Stewart, W.: Product form for stochastic automata networks. In: Proceedings of ValueTools '07, ICST (2007) 32:1–32:10
28. Gugercin, S., Antoulas, A.: Model reduction of large-scale systems by least squares. Linear Algebra and its Applications **415** (2006) 290–321
29. Buchholz, P., Kriege, J.: Aggregation of markovian models - an alternating least squares approach. In: QEST. (2012) 43–52
30. Cousot, P., Monerau, M.: Probabilistic abstract interpretation. In Seidel, H., ed.: ESOP12. Volume 7211 of LNCS., Springer (2012) 166–190