Probabilistic Abstract Intepretation: From Trace Semantics to DTMC's via Linear Regression

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. In order to perform *probabilistic program analysis* we need to consider probabilistic languages or languages with a probabilistic semantics, as well as a corresponding framework for the analysis which is able to accomodate probabilistic properties and properties of probabilistic computations. To this purpose we investigate in this paper the relationship between three different types of probabilisitic semantics for a core imperative language, namely Kozen's Fixpoint Semantics (KFS), our Linear Operator Semantics (LOS) as well as probabilistic versions of Maximal Trace Semantics (MTS). We also discuss the relationship between Probabilistic Abstract Interpretation (PAI) and statistical or linear regression analysis. While classical Abstract Interpretation, based on Galois connection, allows only for worst-case analyses, the use of the Moore-Penrose pseudo inverse in PAI opens the possibility of exploiting statistical and noisy observations in order to analyse and identify various system properties.

1 Introduction

In this contribution we will address a topic that we believe is dear to the hearts of Hanne and Flemming, namely Abstract Interpretation based techniques in program analysis (e.g. [1–4]). We will concentrate on the treatment of the probabilistic setting, where either the program or its semantics or both contain an element of chance that can be used to refine the possible nondeterminism associated to their models. As program analysis is essentially based on the semantics of programs, we will first describe three different probabilistic semantics that could be used as a basis for probabilistic analysis by clarifying the differences and relationship between them, and discussing their potential for the construction of precise program analyses. As a result of this comparison it will be clear that the Probabilistic Abstract Interpretation (PAI) framework originally introduced in [5, 6] is not an instance of a probabilistic application of classical Abstract Interpretation (AI) as recently proposed in [7] in order to analyse probabilistic programs.

The use of linear operators on vector spaces – more concretely on Hilbert spaces – for the definition of a probabilistic semantics is an important feature of the Probabilistic Abstract Interpretation framework for several reasons: (i) it provides us with a well-defined notion of generalised inverse that enjoys properties similar to the concretisation/abstraction functions in the Galois connection framework; (ii) it allows us to exploit a well-defined metric (the Euclidean distance) in order to achieve quantitative results for our static analyses; (iii) it is an appropriate setting where statistical models can be used to enhance the power of static analysis techniques with information gathered via observations.

While we have variously addressed the first two points in our previous work, the potentiality of probabilistic abstract interpretation for performing a kind of *statistical* program analysis was never completely explored before. As another result we will show in this paper that, contrary to the typical computer scientist approach that constructs observations from models, it is sometimes useful to define a model starting from observations, as typically done in statistics. To this end, the particular notion of generalised inverse defining Probabilistic Abstract Interpretation – namely the Moore-Penrose pseudo-inverse [8–10] – makes it very natural to use statistical techniques such as linear regression [11, 10] for constructing abstractions that are as close as possible to the actual system with respect to the observed behaviour of the system.

2 Probabilistic Semantics

There exist a number of proposals for probabilistic languages. These can be based on procedural languages, e.g. [12–14], functional ones, e.g. [15–17], but also declarative ones, like [5, 18]. Besides this there is also a substantial work in probabilistic process algebras [19, 20]. It would be impossible to discuss or even to mention all these approaches here in detail, so we will only concentrate on a small (core) procedural language as it can be found already, for example, in [12] to which we will refer as **pWhile**.

Similarly, a number of approaches have been proposed to defining a semantics for probabilistic programs, not least in order to allow for some form of static program analysis. Usually, it is straightforward to define an operational semantics for a probabilistic extension of a deterministic language; this can be achieved for example by replacing the original (unlabelled) transition relation of an SOS semantics with a weighted version, where the weights represent the probabilities associated with random choices or assignments. Some arguably more useful kinds of semantics are, for example, Kozen's Fixed-Point Semantics (KFS) [12], the Linear Operator Semantics (LOS) introduced by the authors in [21], and the probabilistic Maximal Trace Semantics (MTS) of [7]. We will concentrate in the following on these three models but again stress that many other approaches exist, which are based e.g. on domain theory [22–24], weakest preconditions [25, 26], and the 'monadic' approach in [16, 27].

2.1 A Probabilistic Language

The syntax of the language we consider is a straightforward extension of an imperative language with a probabilistic assignment "x `?=` ρ" where ρ represents a probability distribution on the values of x. We denote distributions as a set of pairs $\{\langle v_i, p_i\rangle\}_i$ which indicate that a constant value v_i has probabilities p_i. As usual we require that $p_i > 0$ (tuples with probability $p_i = 0$ are omitted) and $\sum_i p_i = 1$; these probabilities are all constant, i.e. we do not consider here dynamical changes of distributions. For so-called sub-probability distributions we require only $\sum_i p_i \leq 1$.

The syntax of statements is given below. We also provide a labelled version of this syntax (cf. [4]) in order to be able to refer to certain program points in a program analysis context. For details on expressions $f(x_1, \ldots, x_n)$ (also sometimes denoted simply by e) etc. we refer to e.g. [4, 14].

```
S ::= skip                              S ::= [skip]^ℓ
    | x := f(x_1, ..., x_n)                 | [x := f(x_1, ..., x_n)]^ℓ
    | x ?= ρ                                | [x ?= ρ]^ℓ
    | S_1; S_2                              | S_1; S_2
    | if b then S_1 else S_2 fi             | if [b]^ℓ then S_1 else S_2 fi
    | while b do S od                       | while [b]^ℓ do S od
```

It would also be possible to allow for a probabilistic choice construct of the form "choose p_1 : S_1 or p_2 : S_2 ro", but in order to keep things simple we omit it in our treatment; this statement can be implemented, for example, as c ?= ρ; if $c == 0$ then S_1 else S_2 fi with $\rho = \{\langle 0, p_1\rangle, \langle 1, p_2\rangle\}$. Further details on the (intuitive and operational) semantics of this language can be found for example in [21, 14, 28].

Though we only deal with constant probabilities in the following we will implicitly always normalise probabilities in a distribution (we cannot assume that a programmer provides the correct probabilities), and we will only allow for rational values (non-rational real values for p_i raise issues of computability we will avoid). This means that we can also require that the p_i are integers indicating the probability ratio between different alternatives.

Example 1. We will consider the following **pWhile** program as a running example throughout the paper (its labelled version can be found below in Example 5):

```
while true do
   if (x == 1)
      then x ?= {⟨0, p⟩, ⟨1, 1 − p⟩}
      else x ?= {⟨0, 1 − q⟩, ⟨1, q⟩}
   fi
od
```

This program may be thought of implementing a scheduler in some protocol where $x \mapsto 0$ and $x \mapsto 1$ determines which of two processes has, for example, control over a communication channel.

For a language extension which allows for a probabilistic choice construct we can implement this basic 'scheduler' directly as:

```
while true do
   if ($x == 1$)
      then choose $p$ :  $x := 0$ or $1 - p$ :  $x := 1$ ro
      else choose $1 - q$ :  $x := 0$ or $q$ :  $x := 1$ ro
   fi
od
```

Clearly the execution of this program never terminates: a random switching between the state $x \mapsto 0$ and $x \mapsto 1$ is performed indefinitely according to the probabilities p and q.

In the following we will assume that the state space (and thus the set of configurations) is finite. This makes the treatment substantially simpler as we can avoid topological and measure theoretic details (for which we refer to [28]) and work with just linear algebraic notions instead of functional analytical or operator algebraic ones, cf. [29, 30] etc. This finiteness condition is fulfilled by the example above. It should be noted that the finiteness of the state space however still allows for infinite executions.

2.2 Kozen's Fixed-Point Semantics (KFS)

A well-known denotational semantics for probabilistic programs was introduced by Kozen in the 1980s [12] based on bounded Banach space operators. This is a fixed-point I/O semantics that describes how an input probability distribution (or in general a measure) is transformed into an output sub-probability distribution/measure. It only records contributions of terminating processes. The probabilities of non-terminating, i.e. infinite, computations "gets lost" so the final outcome is no longer normalised or a full probability distribution/measure. As a consequence the semantics of all non-terminating processes is the same (cf. also [28]).

Kozen's language in [12] has no probabilistic choices but only random assignments. It is thus only the random assignment which introduces an element of chance into the execution of a program. However, as discussed before, the two constructs can simulate each other.

From a conceptual point of view, probabilistic choices (or random assignments) are in fact not even part of the actual executions of a program in this setting. Instead all possible choices are made beforehand [12, Section 3.2.2,p336]. Before the execution of a program commences, all later probabilistic choices are already resolved by picking an $\omega \in \Omega$ where $(\Omega, \mathcal{E}, \mu)$ is an appropriate measure space (with $\mathcal{E}$ the σ-algebra of measurable events and μ a probability measure). The semantics of a program is then parametric in this event or scenario ω which determines the probability that the otherwise deterministic execution of a program may effectively happen.

Example 2. Consider the following simple program:

$$x \texttt{ ?= } \{\langle 0, \frac{1}{3}\rangle, \langle 1, \frac{2}{3}\rangle\};\ x \texttt{ ?= } \{\langle 0, \frac{1}{2}\rangle, \langle 1, \frac{1}{2}\rangle\}.$$

The minimal event space we need for defining a semantics for this program is $\Omega = \{0,1\} \times \{0,1\}$ and, because this is a finite set, we can take the whole powerset $\mathcal{E} = \mathcal{P}(\Omega)$ as the σ-algebra of measurable sets. The probability measure of the elements in Ω is then: $\mu(\{(0,0)\}) = \frac{1}{6}$, $\mu(\{(0,1)\}) = \frac{1}{6}$, $\mu(\{(1,0)\}) = \frac{1}{3}$, and $\mu(\{(1,1)\}) = \frac{1}{3}$.

After a scenario ω has been picked, the program behaves exactly as one of the following deterministic programs:

for $\omega = (0,0)$ we execute "x := 0; x := 0" with probability $\frac{1}{6}$,
for $\omega = (0,1)$ we execute "x := 0; x := 1" with probability $\frac{1}{6}$,
for $\omega = (1,0)$ we execute "x := 1; x := 0" with probability $\frac{1}{3}$,
for $\omega = (1,1)$ we execute "x := 1; x := 1" with probability $\frac{1}{3}$.

In the Kozen semantics we can identify a state with a distribution on $\mathbf{Value}^n$, where n is the number of variables and **Value** is the set of possible values of a variable which we assume here – as said before – to be finite. Thus, a probabilistic state (as a distribution $\sigma \in \mathcal{D}(\mathbf{Value}^n)$) can be seen as a normalised element (in the sense of the 1-norm) in the vector space $\mathcal{V}(\mathbf{Value}^n)$. The space $\mathcal{V}(X)$, which allows for the representation of distributions as well as sub-distributions on X, is defined as the set of linear combinations of elements in X, i.e.

$$\mathcal{V}(X) = \left\{ \sum_i \lambda_i x_i \mid x_i \in X \wedge \lambda_i \in \mathbb{R} \right\}.$$

This space is obviously isomorphic to $\mathbb{R}^{|X|}$ with $|X|$ the cardinality of X. Vector addition and scalar product are defined pointwise. We can identify $x_i \in X$ with the base vectors of $\mathcal{V}(X)$ and any element in $\mathcal{V}(X)$ with its coordinates, i.e. the tuple $(\lambda_i)_i$. This space is equipped with an inner product $\langle (\lambda_i)_i | (\nu_i)_i \rangle = \sum_i \lambda_i \nu_i$ and various norms, e.g. $\|(\lambda_i)_i\|_1 = \sum_i |\lambda_i|$ and $\|(\lambda_i)_i\|_2 = \sqrt{\sum_i |\lambda_i|^2} = \sqrt{\langle (\lambda_i)_i | (\lambda_i)_i \rangle}$. The choice of one norm or another is nevertheless irrelevant in the finite dimensional case where all norms are equivalent. In fact, the topology on finite dimensional vector spaces is uniquely determined by the algebraic structure, cf. e.g. [31, 1.22].

The Kozen semantics of a program P is then given by the linear operator $[\![P]\!]_{KFS} \in \mathcal{L}(\mathcal{V}(\mathbf{Value}^n))$ where $\mathcal{L}(X)$ is the set of linear maps $\mathbf{T}$ on X, i.e. $\mathbf{T}(x + y) = \mathbf{T}(x) + \mathbf{T}(y)$ and $\mathbf{T}(\lambda x) = \lambda \mathbf{T}(x)$:

$$[\![P]\!]_{KFS} : \mathcal{V}(\mathbf{Value}^n) \to \mathcal{V}(\mathbf{Value}^n),$$

which is the solution to the following set of equations:

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

The operator $\mathbf{I}$ represents the identity matrix on $\mathcal{V}(\mathbf{Value}^n)$ with $(\mathbf{I})_{vv} = 1$ and 0 otherwise for $v = (v_1,\ldots,v_n) \in \mathbf{Value}^n$. The matrix representation of the test or projection operators $\mathbf{P}$ is given by a diagonal matrix with $(\mathbf{P}(b))_{vv} = 1$ if $b(v)$ holds for $v \in \mathbf{Value}^n$ and 0 otherwise. Note that $\mathbf{P}(\texttt{true}) = \mathbf{I}$ and that $\mathbf{P}(\neg b) = \mathbf{I} - \mathbf{P}(b)$. The assignment or update operator $\mathbf{U}$ is given by a matrix with entries $(\mathbf{U}(x_i \leftarrow f(x_1,\ldots,x_n)))_{v,F(v)} = 1$ for all $v \in \mathbf{Value}^n$ and 0 otherwise, where $F : \mathbf{Value}^n \to \mathbf{Value}^n$ is defined as

$$F(v_1,\ldots,v_{i-1},v_i,v_{i+1},\ldots,v_n) = (v_1,\ldots,v_{i-1},f(v_1,\ldots,v_n),v_{i+1},\ldots,v_n).$$

This definition is equivalent to that given in [12, p339]).

The existence of a solution to these equations is guaranteed in general (i.e. also for infinite state spaces) by the Brouwer-Schauders fixed-point theorem (see e.g. [32, 30]). The least fixed-point can be constructed iteratively via a "super-operator" $\tau : \mathcal{L}(\mathcal{V}(\mathbf{Value}^n)) \to \mathcal{L}(\mathcal{V}(\mathbf{Value}^n))$ which encodes the above equations and by exploiting the lifted point-wise order on distributions/measures.

It should be noted that for infinite sets $\mathbf{Value}$ we need to be more careful with respect to the topology of the state space, e.g. by considering a Banach space like $\ell_1(\mathbf{Value}^n)$ or the Hilbert space $\ell_2(\mathbf{Value}^n)$, cf e.g. [29, 30]. In this case we also need to restrict the space of operators and consider only bounded linear maps in $\mathcal{B}(\ell_1(\mathbf{Value}^n))$ or $\mathcal{B}(\ell_2(\mathbf{Value}^n))$, respectively.

Example 3. Consider again the program P in Example 1. As no executions of this program will ever terminate, there is no proper (sub-)probability distribution describing the final state. Thus Kozen's semantics, which describes the I/O behaviour, is trivial:

$$[\![P]\!]_{KFS} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} = \mathbf{O}$$

i.e. the zero operator $[\![P]\!]_{KFS} : \mathcal{V}(\{x \mapsto 0, x \mapsto 1\}) \to \mathcal{V}(\{x \mapsto 0, x \mapsto 1\})$.

This is also justified by the fixed-point construction described in [12, p341]. The semantics of the statement S given by

$$\texttt{if } (x == 1) \texttt{ then } x \texttt{ ?= } \{\langle 0,p\rangle,\langle 1,1-p\rangle\} \texttt{ else } x \texttt{ ?= } \{\langle 0,1-q\rangle,\langle 1,q\rangle\} \texttt{ fi}$$

forming the body of the loop is easily computed as:

$$[\![S]\!]_{KFS} = \begin{pmatrix} p & 1-p \\ 1-q & q \end{pmatrix},$$

but whatever the semantics $[\![S]\!]_{KFS}$ of the body of loop is, the Kozen semantics of the whole program P is the (appropriate) supremum of a sequence of matrices $\tau^n(\mathbf{O})$ with $n = 1, 2, 3, \ldots$ (starting with the zero matrix $\mathbf{O}$):

$$\tau^n(\mathbf{O}) = \sum_{k=0}^{n-1} (\mathbf{P}(\texttt{true})[\![S]\!]_{KFS})^k \mathbf{P}(\texttt{false}) = \sum_{k=0}^{n-1} (\mathbf{I}[\![S]\!]_{KFS})^k \mathbf{O} = \mathbf{O}.$$

That is, for all $n = 1, 2, 3, \ldots$ we have $\tau^n(\mathbf{O}) = \mathbf{O}$ and thus $[\![P]\!]_{KFS} = \mathbf{O}$.

To clarify the notation used in this example: We represent (sub-)distributions (i.e. vectors) as row vectors; the application of an operator or linear map $\mathbf{T}(x)$ is thus expressed by *post-multiplication* $x \cdot \mathbf{T}$ rather than *pre-multiplication* as it can be found elsewhere (e.g. [12]).

Example 3 describes the situation of a program that does never terminate on all inputs. More interestingly, Kozen's semantics also allows us to model programs that terminate with probability $0 < p < 1$, as shown in the following example.

Example 4. Consider the programs Q, Q' and Q'' which incorporate the program P in Example 3:

```
if (x == 1)            x := 1;                x ?= {⟨0, 1/2⟩, ⟨1, 1/2⟩};
   then x := 0         if (x == 1)            if (x == 1)
   else P                 then x := 0            then x := 0
fi                        else P                 else P
                       fi                     fi
```

The operator $[\![Q]\!]_{KFS}$ of the first program can be easily computed (based on $[\![P]\!]_{KFS}$ in Example 3). We have the Kozen semantics of the two branches of the `if` statements:

$$[\![x \texttt{ := } 0]\!]_{KFS} = \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix} \qquad [\![P]\!]_{KFS} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$$

as well as for the tests guarding the `if` statement:

$$[\![x = 0]\!]_{KFS} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \qquad [\![x = 1]\!]_{KFS} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} = \mathbf{I} - [\![x = 0]\!]_{KFS}$$

Thus by the fifth equation in the definition of the KFS (or section (3.3.4) in [12, p340]) we get:

$$[\![Q]\!]_{KFS} = [\![x = 1]\!]_{KFS}[\![x \texttt{ := } 0]\!]_{KFS} + [\![x = 0]\!]_{KFS}[\![P]\!]_{KFS} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

This means that if we have an initial state $\sigma = (p, 1-p)^t$ which describes the fact that the initial value of x is zero with probability p, and one with probability $1-p$, then $\sigma[\![Q]\!]_{KFS} = (p, 0)^t$ (where $.^t$ denotes the transposition). This is in general (unless $p = 1$) only a sub-probability distribution expressing the fact that this program will terminate with probability p with a zero value for x and that with probability $1-p$ we have non-termination.

If we consider instead the second program Q' then we have

$$[\![x \texttt{ := } 1]\!]_{KFS} = \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} \quad \text{and thus} \quad [\![Q']\!]_{KFS} = [\![x \texttt{ := } 1]\!]_{KFS}[\![Q']\!]_{KFS} = \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix}$$

That means that independently of the initial value of x we always get (i.e. with probability one) a termination and a zero value for x.

Finally, if we consider the program Q'' we get

$$[\![Q'']\!]_{KFS} = \left(\frac{1}{2}[\![x \texttt{ := } 0]\!]_{KFS} + \frac{1}{2}[\![x \texttt{ := } 1]\!]_{KFS}\right)[\![Q]\!]_{KFS} = \begin{pmatrix} \frac{1}{2} & 0 \\ \frac{1}{2} & 0 \end{pmatrix}$$

Here we terminate (again with the resulting x being zero) with a half probability, independently from the initial value of x.

2.3 Linear Operator Semantics (LOS)

The Linear Operator Semantics in [21, 28] constructs the generator of a Discrete Time Markov Chain (DTMC) in a syntax directed fashion. Like Kozen's semantics we can represent the LOS as an operator on the vector space of probabilistic states, i.e. in the finite case as a matrix. In the general case of infinite dimensional state spaces, the LOS is defined on Hilbert spaces instead of the Banach spaces used in the Kozen semantics. This difference is however irrelevant in the finite case.

It is possible to obtain the DTMC generator representing the LOS of a program in an *unstructured* way, simply as a translation of the SOS transition relation (e.g. [5]) into a corresponding (adjacency) matrix. The *structured*, i.e. syntax directed, approach at the base of LOS is based in particular on a tensor (or Kronecker) product construction "$\otimes$" (cf. e.g. [33, 34] or [21]).

The state space is constructed starting from the classical states, i.e. states that associate concrete values in $v_i \in \mathbf{Value}$ to variables $x_i \in \mathbf{Var} = \{x_1, \ldots, x_n\}$. The classical state space can therefore be defined as $\mathbf{State} = \mathbf{Var} \to \mathbf{Value}$ or equivalently $\mathbf{State} = \mathbf{Value}_1 \times \ldots \times \mathbf{Value}_n = \mathbf{Value}^n$.

In order to describe the probabilistic state of a computation we consider (probability) distributions over (classical) states again – as in Kozen's construction – as elements in $\mathcal{V}(\mathbf{Value}^n)$. However, we can use the tensor product operation "$\otimes$" to decompose this probabilistic state space, i.e. $\mathcal{V}(X \times Y) = \mathcal{V}(X) \otimes \mathcal{V}(Y)$ and represent probabilistic states thus as elements in $\mathcal{V}(\mathbf{Value}^n) = \mathcal{V}(\mathbf{Value}_1) \otimes \mathcal{V}(\mathbf{Value}_2) \otimes \ldots \otimes \mathcal{V}(\mathbf{Value}_n) = \mathcal{V}(\mathbf{Value})^{\otimes n}$.

The LOS is based on the labelled version of the syntax of **pWhile**. This means that we can record not only the values of all variables but also the current point in the program we are executing, the "program counter" value. That means that the state space of the corresponding DTMC are (probabilistic) configurations which also contain information about the current label, i.e. with

$\mathbf{Conf} = \mathbf{State} \times \mathbf{Label}$ we consider distributions in $\mathcal{D}(\mathbf{Conf}) \subseteq \mathcal{V}(\mathbf{Conf}) = \mathcal{V}(\mathbf{State}) \otimes \mathcal{V}(\mathbf{Label}) = \mathcal{V}(\mathbf{Value})^{\otimes n} \otimes \mathcal{V}(\mathbf{Label})$.

The LOS, $[\![P]\!]_{LOS}$, of a program P is constructed by means of a set, $\{\!\{P\}\!\}_{LOS}$ which associated to a program P is a set of linear operators which describe local changes (at individual labels). From $\{\!\{P\}\!\}_{LOS}$ we can construct the DTMC generator $[\![P]\!]_{LOS}$ then as an operator

$$[\![P]\!]_{LOS} : \mathcal{V}(\mathbf{Value}^n) \otimes \mathcal{V}(\mathbf{Label}) \rightarrow \mathcal{V}(\mathbf{Value}^n) \otimes \mathcal{V}(\mathbf{Label})$$

or simply $[\![P]\!]_{LOS} \in \mathcal{L}(\mathcal{V}(\mathbf{Conf}))$. We obtain it by combining all the individual effects which are described in $\{\!\{P\}\!\}_{LOS}$:

$$[\![P]\!]_{LOS} = \sum \{\!\{P\}\!\}_{LOS} = \sum \{\mathbf{G} \mid \mathbf{G} \in \{\!\{P\}\!\}_{LOS}\}.$$

The $\{\!\{S\}\!\}_{LOS}$ associated to a statement S is given by a set of global and local operators, i.e. $\{\!\{.\}\!\}_{LOS} : \mathbf{Stmt} \rightarrow \mathcal{P}(\Gamma \cup \Lambda)$. Global operators are linear operators on $\mathcal{V}(\mathbf{Conf})$ i.e. $\Gamma = \mathcal{L}(\mathcal{V}(\mathbf{Value}^n) \otimes \mathcal{V}(\mathbf{Label})) = \mathcal{L}(\mathcal{V}(\mathbf{Conf}))$, and local operators are pairs of operators on $\mathcal{V}(\mathbf{State})$ and labels $\ell \in \mathbf{Label}$, i.e. $\Lambda = \mathcal{L}(\mathcal{V}(\mathbf{Value}^n)) \times \mathbf{Label}$.

Global operators are providing information about how the computational state changes at a label as well as the control flow, i.e. what is the label of the next statement to be executed. Local operators are representing statements for which the "continuation" is not yet known. In order to transform local operators into global ones (once the "continuation" is known) we define a "continuation" operation $\langle \mathbf{F}, \ell \rangle \rhd \ell' = \mathbf{F} \otimes \mathbf{E}(\ell, \ell')$ which we extend in the obvious way to sets of (local) operators as $\{\langle \mathbf{F}_i, \ell_i \rangle\} \rhd \ell' = \{\mathbf{F}_i \otimes \mathbf{E}(\ell_i, \ell')\}$ (for global operators, clearly, we have $\mathbf{G} \rhd \ell' = \mathbf{G}$). We denote by $\mathbf{E}(i,j)$ matrix units, i.e. $(\mathbf{E}(i,j))_{ij} = 1$ and 0 otherwise.

The set $\{\!\{S\}\!\}_{LOS}$ of operators for a statement S is defined inductively (based on the syntax of S) as follows:

$$\begin{aligned}
\{\!\{[\texttt{skip}]^\ell\}\!\}_{LOS} &= \{\langle \mathbf{I}, \ell \rangle\} \\
\{\!\{[x \texttt{ := } e]^\ell\}\!\}_{LOS} &= \{\langle \mathbf{U}(x \leftarrow e), \ell \rangle\} \\
\{\!\{[x \texttt{ ?= } \rho]^\ell\}\!\}_{LOS} &= \{\langle \sum_{\langle r,p \rangle \in \rho} p \cdot \mathbf{U}(x \leftarrow r), \ell \rangle\} \\
\{\!\{S_1;\ S_2\}\!\}_{LOS} &= ([\![S_1]\!] \rhd \mathit{init}(S2)) \cup [\![S_2]\!] \\
\{\!\{\texttt{if } [b]^\ell \texttt{ then } S_1 \texttt{ else } S_2 \texttt{ fi}\}\!\}_{LOS} &= \{\langle \mathbf{P}(b), \ell \rangle\} \rhd \mathit{init}(S_1)\} \cup \{\!\{S_1\}\!\}_{LOS} \cup \\
&\quad\ \{\langle \mathbf{P}(b)^\perp, \ell \rangle\} \rhd \mathit{init}(S_2)\} \cup \{\!\{S_2\}\!\}_{LOS} \\
\{\!\{\texttt{while } [b]^\ell \texttt{ do } S \texttt{ od}\}\!\}_{LOS} &= \{\langle \mathbf{P}(b), \ell \rangle\} \rhd \mathit{init}(S)\} \cup \{\!\{S\}\!\}_{LOS} \cup \{\langle \mathbf{P}(b)^\perp, \ell \rangle\}
\end{aligned}$$

We use elementary update and test operators $\mathbf{U}$ and $\mathbf{P}$ (and its complement $\mathbf{P}^\perp = \mathbf{I} - \mathbf{P}$) as in Kozen's semantics. However, the tensor product structure allows us to define these operators in a different (but equivalent) way.

For a *single* variable the assignment to a constant value $v \in \mathbf{Value}$ is represented by the operator on $\mathcal{V}(\mathbf{Value})$ given by $\mathbf{U}(v) = 1$ if $v = i$ and 0 otherwise.

Testing if a *single* variable satisfies a boolean test b is achieved by a (diagonal) projection operator on $\mathcal{V}(\mathbf{Value})$ with $(\mathbf{P}(b))_{ii} = 1$ if $b(i)$ holds and 0 otherwise.

We extend these to the multivariable case, i.e. for $|\mathbf{Var}| = n > 1$. For testing if we are in a classical state $s \in \mathbf{Value}^n$ or if an expression e evaluates to a constant v (assuming an appropriate evaluation function $\mathcal{E} : \mathbf{Expr} \to \mathbf{State} \to \mathbf{Value}$) we have operators on $\mathcal{V}(\mathbf{Value})^{\otimes n}$:

$$\mathbf{P}(s) = \bigotimes_{i=1}^{n} \mathbf{P}(\mathtt{x}_i = s(\mathtt{x}_i)) \qquad\qquad \mathbf{P}(e = v) = \sum_{\mathcal{E}(e)s=v} \mathbf{P}(s).$$

We also have operators on $\mathcal{V}(\mathbf{Value})^{\otimes n}$ for updating a variable $\mathtt{x}_k$ in the context of other variables to a constant v or to the value of an expression e:

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

As we model the semantics of a program as DTMC's we are also adding a final loop ℓ^* (for ℓ^* a fresh label not appearing already in P) when we consider a complete program (DTMC never terminate and thus we have to simulate termination by an infinite repetition of the final state), i.e. we actually have to use $(\{\!\{P\}\!\}_{LOS} \rhd \ell^*) \cup \{\mathbf{I} \otimes \mathbf{E}(\ell^*, \ell^*)\}$ when we construct $[\![P]\!]_{LOS}$. In this way we also resolve all open or dangling control flow steps, i.e. we deal ultimately with a set containing only global operators.

Example 5. Consider the labelled version of the program in Example 1

```
while [true]$^1$ do
   if [$(x == 1)$]$^2$
      then [$x$ ?= $\{\langle 0, p\rangle, \langle 1, 1-p\rangle\}$]$^3$
      else [$x$ ?= $\{\langle 0, 1-q\rangle, \langle 1, q\rangle\}$]$^4$
   fi
od
```

In order to define the LOS of this program we construct the state space as $\mathcal{V}(\{x \mapsto 0, x \mapsto 1\}) = \mathbb{R}^2$ (since we have only one variable we do not need the tensor product for this). The space of configurations is $\mathcal{V}(\{x \mapsto 0, x \mapsto 1\}) \otimes \mathcal{V}(\{1, 2, 3, 4, 5\})$, where label 5 is the label of the additional final loop. We will omit the final label (which in this program we actually never reach) in order to keep things a bit smaller. Nothing important would change if we considered label 5 except that we had to include also the operator $\mathbf{I} \otimes \mathbf{E}(5, 5)$ as an element in $\{\!\{P\}\!\}_{LOS}$ and we had to consider 5×5 matrices rather than just 4×4 matrixes in order to represent the control flow steps $\mathbf{E}(i, j)$. The LOS set of operators $\{\!\{P\}\!\}_{LOS}$ of our program P is given by the following set of (global) operators:

$$\begin{aligned}\{\!\{P\}\!\}_{LOS} = \{&\mathbf{P}(\mathtt{true}) \otimes \mathbf{E}(1, 2), \mathbf{P}(x = 1) \otimes \mathbf{E}(2, 3), \mathbf{P}(x = 1)^{\perp} \otimes \mathbf{E}(2, 4),\\ &(p \cdot \mathbf{U}(x \leftarrow 0) + (1 - p) \cdot \mathbf{U}(x \leftarrow 1) \otimes \mathbf{E}(3, 1),\\ &((1 - q) \cdot \mathbf{U}(x \leftarrow 0) + q \cdot \mathbf{U}(x \leftarrow 1) \otimes \mathbf{E}(4, 1)\}\end{aligned}$$

The concrete matrices representing the operators on $\mathcal{V}(\{x \mapsto 0, x \mapsto 1\}) \otimes \mathcal{V}(\{1,2,3,4\})$ are of the following form:

$$\{\!\!\{P\}\!\!\}_{LOS} = \left\{ \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \mathbf{E}(1,2), \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \otimes \mathbf{E}(2,3), \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \otimes \mathbf{E}(2,4), \right.$$
$$\left. \left(\begin{pmatrix} p & 0 \\ p & 0 \end{pmatrix} + \begin{pmatrix} 0 & (1-p) \\ 0 & (1-p) \end{pmatrix} \right) \otimes \mathbf{E}(3,1), \left(\begin{pmatrix} (1-q) & 0 \\ (1-q) & 0 \end{pmatrix} + \begin{pmatrix} 0 & q \\ 0 & q \end{pmatrix} \right) \otimes \mathbf{E}(4,1) \right\}$$

or

$$\{\!\!\{P\}\!\!\}_{LOS} = \left\{ \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \right.$$
$$\left. \begin{pmatrix} p & (1-p) \\ p & (1-p) \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} (1-q) & q \\ (1-q) & q \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \right\}$$

The sum of these five 8×8 matrices gives the operator $[\![P]\!]_{LOS}$ which is the generator of the corresponding DTMC. If we consider also the final label $\ell^* = 5$ then we get a 10×10 matrix. For $p = q = \frac{1}{2}$ this has, for example, the form:

$$[\![P]\!]_{LOS} = \left(\begin{array}{cccccccccc} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array} \right) \quad \begin{array}{l} \ldots x \mapsto 0, \ell = 1 \\ \ldots x \mapsto 0, \ell = 2 \\ \ldots x \mapsto 0, \ell = 3 \\ \ldots x \mapsto 0, \ell = 4 \\ \ldots x \mapsto 0, \ell = 5 \\ \ldots x \mapsto 1, \ell = 1 \\ \ldots x \mapsto 1, \ell = 2 \\ \ldots x \mapsto 1, \ell = 3 \\ \ldots x \mapsto 1, \ell = 4 \\ \ldots x \mapsto 1, \ell = 5 \end{array}$$

where we indicate the configuration (i.e. value of x and current label ℓ) each row and column corresponds to. It is perhaps worth noting that this – as one would expect for a DTMC – is indeed a stochastic matrix, i.e. all row sums are one.

There is a close relationship between the KFS and the LOS as for basic blocks B – i.e. (random) assignments, tests and skips – the LOS operator is the same as the KFS operator except for an additional control flow step. That means that $\{\!\!\{\ldots [B]^i \ldots\}\!\!\}_{LOS} = \{\ldots, \langle [\![B]\!]_{KFS}, i\rangle, \ldots\}$ or $\{\!\!\{\ldots [B]^i \ldots\}\!\!\}_{LOS} = \{\ldots, [\![B]\!]_{KFS} \otimes \mathbf{E}(i,j), \ldots\}$ for some label j.

Example 6. For the programs in Example 4, labelling them as follows:

```
if [(x == 1)]^1           [x := 1]^0;              [x ?= {⟨0, 1/2⟩, ⟨1, 1/2⟩}]^0;
   then [x := 0]^2        if [(x == 1)]^1          if [x == 1]^1
   then P                    then [x := 0]^2          then [x := 0]^2
fi                           then P                   then P
                          fi                       fi
```

(the labels of P are as in the previous example shifted by an offset of 2) we can describe the LOS using the KFS operators as follows:

$$\begin{aligned}
\{\!\{Q\}\!\}_{LOS} &= \{[\![x = 1]\!]_{KFS} \otimes \mathbf{E}(1,2), [\![x = 0]\!]_{KFS} \otimes \mathbf{E}(1,3), \\
&\qquad \langle [\![x \texttt{ := } 0]\!]_{KFS}, 2\rangle, \langle [\![\texttt{false}]\!]_{KFS}, 3\rangle\} \cup \{\!\{P\}\!\}_{LOS} \\
\{\!\{Q'\}\!\}_{LOS} &= \{[\![x \texttt{ := } 1]\!]_{KFS} \otimes \mathbf{E}(0,1)\} \cup \{\!\{Q\}\!\}_{LOS} \\
\{\!\{Q''\}\!\}_{LOS} &= \{(\frac{1}{2}[\![x \texttt{ := } 1]\!]_{KFS} + \frac{1}{2}[\![x \texttt{ := } 1]\!]_{KFS}) \otimes \mathbf{E}(0,1),\} \cup \{\!\{Q\}\!\}_{LOS}
\end{aligned}$$

where we can re-use the LOS semantics of program P (with shifted labelling):

$$\begin{aligned}
\{\!\{P\}\!\}_{LOS} = \{&[\![\texttt{true}]\!]_{KFS} \otimes \mathbf{E}(3,4), [\![x = 1]\!]_{KFS} \otimes \mathbf{E}(4,5), [\![x = 0]\!]_{KFS} \otimes \mathbf{E}(4,6), \\
&(p[\![x \texttt{ := } 0]\!]_{KFS} + (p-1)[\![x \texttt{ := } 1]\!]_{KFS}) \otimes \mathbf{E}(5,3), \\
&((q-1)[\![x \texttt{ := } 0]\!]_{KFS} + q[\![x \texttt{ := } 1]\!]_{KFS}) \otimes \mathbf{E}(6,3)\}
\end{aligned}$$

Note that the LOS of our three small programs contain not just global but also local operators, namely $\langle [\![x \texttt{ := } 0]\!]_{KFS}, 2\rangle$ and $\langle [\![\texttt{false}]\!]_{KFS}, 3\rangle$. This is because we still have to add a terminal label $\ell^* = 7$ for the construction of the complete DTMC generators $[\![Q]\!]_{LOS}$, etc. The terminal label can be reached from both branches of the `if` statement which have the (final) labels 2 and 3. However, as $[\![\texttt{false}]\!]_{KFS}$ is $\mathbf{O}$ this operator (which would correspond to a terminating program P) does actually not contribute to the DTMC generator.

2.4 Maximal Trace Semantics (MTS)

Maximal Trace Semantics for non-probabilistic programs has been discussed in [35, 36] and shown to be the most concrete semantics in a hierarchy of various semantics for (non-)deterministic programs. In [7] the MTS is extended to the probabilistic case.

Similar to the Kozen semantics, the conceptual idea is to ban any probabilistic steps from the actual execution of the program: Before we actually execute the program all probabilistic choices (coin flips, rolling of dices) have already been performed. The actual execution of a program then is purely deterministic (or purely non-deterministic) but parameterised by the results $\omega \in \Omega$ of the “pre-run” choices or (cf. [7, p171]). Therefore, the execution traces/paths depend on the outcomes of these “pre-run” events or scenarios in Ω.

Given a set of states Σ, we use the notation of [7] and denote the set of finite traces by Σ^+, by Σ^* the set $\Sigma^+ \cup \{\varepsilon\}$, where ε is the empty trace of length 0, by Σ^∞ the infinite traces, by $\Sigma^{+\infty} = \Sigma^+ \cup \Sigma^\infty$ and by $\Sigma^{*\infty} = \Sigma^* \cup \Sigma^\infty$. For sets of traces $X, Y, \ldots$ in $\Sigma^{*\infty}$, we can define the following operations: $X^\infty = X \cap \Sigma^\infty$, $X^+ = X \cap \Sigma^+$, $X|_Y = \{s\sigma_X \in X \mid \exists\sigma : \ \sigma s \in Y^+\}$, and $X;Y = X^\infty \cup \{\sigma_X s \sigma_Y \mid \sigma_X s \in X^+ \ \wedge \ s\sigma_Y \in Y\}$.

The MTS is then defined as $[\![S]\!]_{MTS} : \textbf{Stmt} \to \Omega \to \mathcal{P}(\Sigma^{+\infty})$ where $\Sigma = \textbf{State}$ (one could also consider $\Sigma = \textbf{Conf}$, more on this later). In order to combine non-determinism with probabilities each scenario $\omega \in \Omega$ is associated to a whole set of possible traces. Thus $[\![S]\!]_{MTS}$ is defined by (cf [7, Example 4]):

$$
\begin{aligned}
[\![\texttt{skip}]\!]_{MTS}(\omega) &= \{ss \mid s \in \Sigma\} \\
[\![x \texttt{ := } e]\!]_{MTS}(\omega) &= \{ss[x \mapsto [\![e]\!](\omega)s] \mid s \in \Sigma\} \\
[\![S_1; S_2]\!]_{MTS}(\omega) &= [\![S_1]\!]_{MTS}(\omega); [\![S_2]\!]_{MTS}(\omega) \\
[\![b]\!]_{MTS}(\omega) &= \{s \mid [\![b]\!](\omega)s\} \\
[\![\texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2 \texttt{ fi}]\!]_{MTS}(\omega) &= [\![b]\!]_{MTS}(\omega); [\![S_1]\!]_{MTS}(\omega) \cup [\![\neg b]\!]_{MTS}(\omega); [\![S_2]\!]_{MTS}(\omega) \\
[\![\texttt{while } b \texttt{ do } S \texttt{ od}]\!]_{MTS}(\omega) &= \mathrm{lfp} \lambda X. [\![b]\!]_{MTS}(\omega) \cup [\![\neg b]\!]_{MTS}(\omega); [\![S]\!]_{MTS}(\omega); X
\end{aligned}
$$

According to the definition in [7, Example 4] the evaluation $[\![e]\!]$ of an expression e depends on the scenario ω, i.e. $[\![e]\!] : \Omega \to (\Sigma \to \Sigma)$, and in a similar way for Boolean expressions b. The language considered in [7] does actually not have neither random assignments nor a choice construct; the former, i.e. x `?=` ρ, is instead implemented via a kind of "system call", i.e. `x := random(`ρ`)`.

We can reformulate the MTS in the case of the **pWhile** language where no non-determinism is present: Once a scenario ω is fixed there is only one trace for every initial state or configuration which is actually executed. We are not interested in the scenarios in Ω themselves but only in their probabilities $\mu(\omega)$ for $\omega \in \Omega$, i.e. the probability that a certain trace gets executed. Thus, for a fixed initial state or configuration s the MTS of a program in **pWhile** can be seen as a distribution over traces. The probability for each trace σ is inherited from the scenario ω it depends on. We will use in the following the notation $\{\langle \sigma, \mu(\omega) \rangle\}$ to express that a trace σ is executed with probability $\mu(\sigma)$.

It is possible to define the MTS for purely probabilistic programs – like the ones in **pWhile** – directly as a map from statements to weighted traces. $[\![.]\!]_{MTS} : \textbf{Stmt} \to \mathcal{V}(\Sigma^{+\infty})$ is defined implicitly, i.e. as solution to the following equations:

$$
\begin{aligned}
[\![\texttt{skip}]\!]_{MTS} &= \{\langle ss, 1\rangle \mid s \in \Sigma\} \\
[\![x \texttt{ := } e]\!]_{MTS} &= \{\langle ss[x \mapsto [\![e]\!]s], 1\rangle \mid s \in \Sigma\} \\
[\![x \texttt{ ?= } \rho]\!]_{MTS} &= \{\langle ss[x \mapsto r], \rho(r)\rangle \mid s \in \Sigma \wedge \rho(r) \neq 0\} \\
[\![S_1; S_2]\!]_{MTS} &= [\![S_1]\!]_{MTS}; [\![S_2]\!]_{MTS} \\
[\![\texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2 \texttt{ fi}]\!]_{MTS} &= \{\langle s, 1\rangle \mid \text{for } [\![b]\!](s) = \texttt{true}\}; [\![S_1]\!]_{MTS} \\
&\cup \{\langle s, 1\rangle \mid \text{for } [\![b]\!](s) = \texttt{false}\}; [\![S_2]\!]_{MTS} \\
[\![\texttt{while } b \texttt{ do } S \texttt{ od}]\!]_{MTS} &= \{\langle s, 1\rangle \mid \text{for } [\![b]\!](s) = \texttt{true}\}; [\![S; \texttt{while } b \texttt{ do } S \texttt{ od}]\!]_{MTS} \\
&\cup \{\langle s, 1\rangle \mid \text{for } [\![b]\!](s) = \texttt{false}\};
\end{aligned}
$$

For the evaluation of deterministic functions or expressions e the footnotes in [7, p173] apply, namely that $[\![e]\!]$ is now independent of the scenario ω. For random assignments we produce a set of weighted traces, one trace for each

r with non-vanishing probability according to the distribution ρ. We extend the concatenation operation for traces to probabilistic ones in the obvious way: $\langle X, p_X\rangle; \langle Y, p_Y\rangle = \langle X; Y, p_X p_Y\rangle$ in oder to define the semantics of sequential statements. The operation ";" also extends pointwise to sets of weighted traces in $\mathcal{V}(\Sigma^{+\infty})$. The union construction $\cup$ of sets of weighted tuples corresponds to a sum if we take them as elements in the vector space $\mathcal{V}(\Sigma^{+\infty})$.

It should be noted that this formulation of the MTS for a purely probabilistic language eliminates the dependency on the scenarios $\omega \in \Omega$ but not on the initial state $s \in \Sigma$. That means that for a statement S the weighted set of traces $[\![S]\!]_{MTS} \in \mathcal{V}(\Sigma^{+\infty})$ does in general itself **not** represent a distribution (on traces) but just a (positive) vector in $\mathcal{V}(\Sigma^{+\infty})$. However, if we collect all those traces which start with the same state s then we obtain a distribution over traces, i.e. $\sum\{p \mid \langle\sigma, p\rangle \text{ with } \sigma = s \ldots\} = 1$.

It would be possible to formulate the MTS also as a map which expresses the dependency on the initial state explicitly and returns directly distributions over traces, i.e. $[\![.]\!]_{MTS} : \mathbf{Stmt} \to \Sigma \to \mathcal{D}(\Sigma^{+\infty}) \subseteq \mathcal{V}(\Sigma^{+\infty})$ in which case $[\![S]\!]_{MTS}(s)$ would simply represent a distribution over traces. However, our aim is to stay as possible to the formulation in [7], which is based on the typing $[\![.]\!]_{MTS} : \mathbf{Stmt} \to \Omega \to \mathcal{P}(\Sigma^{+\infty})$ rather than, for example, $[\![.]\!]_{MTS} : \mathbf{Stmt} \to \Omega \to \Sigma \to \mathcal{P}(\Sigma^{+\infty})$.

We omit the discussion of the necessary measure theoretic constructions in the case of infinite spaces (like prefixes and so-called cylinders in order to define an appropriate σ-algebra for infinite traces, cf [37] or [38, 39], etc.) as in the following we will only consider finite traces as computationally significant.

Example 7. In order to illustrate the basic construction of the MTS we consider the following unlabelled or labelled program with $x, y \in \{0, 1\}$:

```
if  $(y < 1)$                                  if  $[(y < 1)]^1$
  then $x$ ?= $\{\langle 0, p\rangle, \langle 1, 1-p\rangle\}$        then $[x$ ?= $\{\langle 0, p\rangle, \langle 1, 1-p\rangle\}]^2$
  else $x$ := $0$                              else $[x$ := $0]^3$
fi;                                            fi;
if  $(x < 1)$                                  if  $[(x < 1)]^4$
  then $y$ ?= $\{\langle 0, q\rangle, \langle 1, 1-q\rangle\}$        then $[y$ ?= $\{\langle 0, q\rangle, \langle 1, 1-q\rangle\}]^5$
  else $y$ := $0$                              else $[y$ := $0]^6$
fi                                             fi
```

In this example we have no loops or recursions, so we know that we will need (at most) two "coin flips". That is the space of scenarios Ω is defined via the two choices, one for x and one for y.

The state space is defined as $\Sigma = \{x \mapsto 0, x \mapsto 1\} \times \{y \mapsto 0, y \mapsto 1\}$ which we denote by $\Sigma = \{[00], [01], [10], [11]\}$, i.e. $[00]$ denotes the state with $x \mapsto 0$ and $y \mapsto 0$, etc. Following the reformulation of the MTS we have:

$$
\begin{aligned}
&[\![x \texttt{ := } 0]\!]_{MTS} = \\
&= \{\langle[00][00], 1\rangle, \langle[01][01], 1\rangle, \langle[10][00], 1\rangle, \langle[11][01], 1\rangle\} \\
&[\![y \texttt{ := } 0]\!]_{MTS} =
\end{aligned}
$$

$$= \{\langle[00][00],1\rangle,\langle[01][00],1\rangle,\langle[10][10],1\rangle,\langle[11][10],1\rangle\}$$

$$[\![x \texttt{ ?= } \{\langle 0,p\rangle,\langle 1,1-p\rangle\}]\!]_{MTS} =$$

$$= \{\langle[00][00],p\rangle,\langle[01][01],p\rangle,\langle[10][00],p\rangle,\langle[11][01],p\rangle,$$
$$\langle[00][10],1-p\rangle,\langle[01][11],1-p\rangle,\langle[10][10],1-p\rangle,\langle[11][11],1-p\rangle\}$$

$$[\![y \texttt{ ?= } \{\langle 0,q\rangle,\langle 1,1-q\rangle\}]\!]_{MTS} =$$

$$= \{\langle[00][00],q\rangle,\langle[01][00],q\rangle,\langle[10][10],q\rangle,\langle[11][10],q\rangle,$$
$$\langle[00][01],1-q\rangle,\langle[01][01],1-q\rangle,\langle[10][11],1-q\rangle,\langle[11][11],1-q\rangle\}$$

With these sets of weighted traces we can easily construct the MTS for the two `if` statements:

$$[\![\texttt{if } (y<1) \texttt{ then } x \texttt{ ?= } \{\langle 0,p\rangle,\langle 1,1-p\rangle\} \texttt{ else } x \texttt{ := } 0 \text{ fi}]\!]_{MTS} =$$
$$= \{\langle[00][00][00],p\rangle,\langle[10][10][00],p\rangle,\langle[00][00][10],1-p\rangle,$$
$$\langle[10][10][10],1-p\rangle,\langle[01][01][01],1\rangle,\langle[11][11][01],1\rangle\}$$

$$[\![\texttt{if } (x<1) \texttt{ then } y \texttt{ ?= } \{\langle 0,q\rangle,\langle 1,1-q\rangle\} \texttt{ else } y \texttt{ := } 0 \text{ fi}]\!]_{MTS} =$$
$$= \{\langle[00][00][00],q\rangle,\langle[01][01][00],q\rangle,\langle[00][00][01],1-q\rangle,$$
$$\langle[01][01][01],1-q\rangle,\langle[10][10][10],1\rangle,\langle[11][11][10],1\rangle\}$$

Note that some traces which we constructed for the branches disappear because when we apply the operator ";" the last state of the first (one step) trace (representing the test) and the first state of the continuation (in one of the two branches) do not match.

The traces for the whole program are then given by:

$$[\![P]\!]_{MTS} = \{\langle[00][00][00],p\rangle;\langle[00][00][00],q\rangle,\langle[00][00][00],p\rangle;\langle[00][00][01],1-q\rangle,$$
$$\langle[10][10][00],p\rangle;\langle[00][00][00],q\rangle,\langle[10][10][00],p\rangle;\langle[00][00][01],1-q\rangle;$$
$$\langle[00][00][10],1-p\rangle;\langle[10][10][10],1\rangle,\langle[10][10][10],1-p\rangle;\langle[10][10][10],1\rangle,$$
$$\langle[01][01][01],1\rangle;\langle[01][01][00],q\rangle,\langle[01][01][01],1\rangle;\langle[01][01][01],1-q\rangle,$$
$$\langle[11][11][01],1\rangle;\langle[01][01][00],q\rangle,\langle[11][11][01],1\rangle;\langle[01][01][01],1-q\rangle\}$$

where again the matching condition eliminates a number of possible traces. Finally we get:

$$[\![P]\!]_{MTS} = \{\langle[00][00][00][00][00],pq\rangle,\langle[00][00][00][00][01],p(1-q)\rangle,$$
$$\langle[10][10][00][00][00],pq\rangle,\langle[10][10][00][00][01],p(1-q)\rangle;$$
$$\langle[00][00][10][10][10],1-p\rangle,\langle[10][10][10][10][10],1-p\rangle,$$
$$\langle[01][01][01][01][00],q\rangle,\langle[01][01][01][01][01],1-q\rangle,$$
$$\langle[11][11][01][01][00],q\rangle,\langle[11][11][01][01][01],1-q\rangle\}$$

Here we have three possible traces starting with the initial state $s = [00]$ or $s = [10]$ but only two for $s = [01]$ and $[11]$. We also observe that the probabilities associated to the traces starting with each of the four initial states sum up to one, e.g. for $s = [00]$ we have the probabilities $(pq) + (p - pq) + (1 - p) = 1$.

In this presentation of the MTS we followed the presentation given also in, for example, [7, Fig.1]. The states only record the values of the variables but not the current label (or program counter). That means that it is possible to obtain "coincidentally" the same trace for completely different executions of the program. To keep track of the control flow through a program we can also record information about the configurations executed and not just the states. For a labelled version of the program we would then replace a trace like $[00][00][00][00][00]$ by $\langle[00], 1\rangle\langle[00], 2\rangle\langle[00], 4\rangle\langle[00], 5\rangle\langle[00], \ell^*\rangle$ with ℓ^* the (additional) final label indicating termination.

3 Probabilistic vs Classical Abstract Interpretation

Abstract Interpretation (AI) is a well known mathematical theory at the base of a number of static analysis techniques, e.g. [40, 4]. Because of the need to consider computable domains for performing the analysis of programs properties, abstraction and approximation are essential features of any static analysis technique. The theory of AI establishes when the approximation is such that an analysis can be safely performed on an abstract rather than the concrete domain of computation. More precisely, the correctness of an abstract semantics is guaranteed by ensuring that a pair of functions α and γ can be defined which form a *Galois connection* between two lattices $\mathcal{C}$ and $\mathcal{D}$ representing concrete and abstract properties. This classical theory originally introduced for (non-)deterministic programs can be extended so as to include the treatment of probabilistic programs by considering the appropriate (abstract and concrete) domains as recently shown in [7] (see also [41]).

Though the approximations allowed by the AI theory will always be safe, they might also be quite unrealistic, addressing a *worst case* scenario rather than the *average case* [42]. This latter is typically the aim of a probabilistic analysis which is therefore hardly correct in the classical sense of the AI theory. However, although such an average case analysis is not guaranteed to 'err on the safe side', we can still define it so as to reduce the error margin. In order to provide a mathematical framework for probabilistic analysis, we have previously introduced in [5, 6], a theory of linear operators on Hilbert spaces (i.e. here just finite dimensional spaces as discussed before) where the notion of approximation is characterised in terms of *least square approximation*, which we have called *Probabilistic Abstract Interpretation* (PAI) [5].

The PAI approach is based, as in the classical case, on a concrete and abstract domain $\mathcal{C}$ and $\mathcal{D}$ – except that $\mathcal{C}$ and $\mathcal{D}$ are now vector spaces (or in general, Hilbert spaces which are equipped with the notion of an inner product and consequently of orthogonality, adjoint operators, etc.) instead of lattices. We assume that the pair of abstraction and concretisation function $\mathbf{A} : \mathcal{C} \rightarrow \mathcal{D}$ and

$\mathbf{G} : \mathcal{D} \to \mathcal{C}$ are again structure preserving, i.e. in our setting they are (bounded) linear maps represented by matrices $\mathbf{A}$ and $\mathbf{G}$. Finally, we replace the notion of a Galois connection by the notion of a *Moore-Penrose pseudo-inverse* [8, 10].

Definition 1. *Let $\mathcal{C}$ and $\mathcal{D}$ be two finite dimensional vector spaces, and let $\mathbf{A} : \mathcal{C} \to \mathcal{D}$ be a linear map between them. The linear map $\mathbf{A}^\dagger = \mathbf{G} : \mathcal{D} \to \mathcal{C}$ is the* Moore-Penrose pseudo-inverse *of $\mathbf{A}$ iff*

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

where $\mathbf{P}_A$ and $\mathbf{P}_G$ denote orthogonal projections (i.e. $\mathbf{P}_A^* = \mathbf{P}_A = \mathbf{P}_A^2$ and $\mathbf{P}_G^* = \mathbf{P}_G = \mathbf{P}_G^2$ where $.^*$ denotes the adjoint *[33, Ch 10]) onto the ranges of $\mathbf{A}$ and $\mathbf{G}$.*

Alternatively, if $\mathbf{A}$ is Moore-Penrose invertible (and all finite dimensional operators or matrices are), its Moore-Penrose pseudo-inverse, $\mathbf{A}^\dagger$ satisfies the following:

$$\begin{aligned} &\textbf{(i)}\ \ \mathbf{A}\mathbf{A}^\dagger\mathbf{A} = \mathbf{A},\\ &\textbf{(ii)}\ \ \mathbf{A}^\dagger\mathbf{A}\mathbf{A}^\dagger = \mathbf{A}^\dagger,\\ &\textbf{(iii)}\ \ (\mathbf{A}\mathbf{A}^\dagger)^* = \mathbf{A}\mathbf{A}^\dagger,\\ &\textbf{(iv)}\ \ (\mathbf{A}^\dagger\mathbf{A})^* = \mathbf{A}^\dagger\mathbf{A}. \end{aligned}$$

It is instructive to compare these equations with the classical setting. For example, if (α, γ) is a Galois connection we similarly have $\alpha\circ\gamma\circ\alpha = \alpha$ and $\gamma\circ\alpha\circ\gamma = \gamma$. We also have in a similar way as in the AI setting that $\mathbf{A}$ and $\mathbf{A}^\dagger$ determine each other uniquely (i.e. $(\mathbf{A}^\dagger)^\dagger = \mathbf{A}$, e.g. [10, EX. 18,p49]).

The Moore-Penrose pseudo-inverse allows us to construct the closest (i.e. least square) approximation $\mathbf{T}^\# : \mathcal{D} \to \mathcal{D}$ of a concrete semantics $\mathbf{T} : \mathcal{C} \to \mathcal{C}$ as:

$$\mathbf{T}^\# = \mathbf{G} \cdot \mathbf{T} \cdot \mathbf{A} = \mathbf{A}^\dagger \cdot \mathbf{T} \cdot \mathbf{A} = \mathbf{A} \circ \mathbf{T} \circ \mathbf{G}.$$

In [5] we show how we can transform a Probabilistic Abstract Interpretation into a classical Abstract Interpretation, simply by forgetting the concrete (non-zero) values of probabilities and only considering the support set of a distribution as the set of "possibilities". One can also lift (in a non-unique way) a classical Abstract Interpretation to a Probabilistic Abstract Interpretation (e.g. by using uniform distributions). This method is conceptually equivalent to the probabilistic version of Abstract Interpretation presented in [7], although the result does not refer explicitly to the Maximal Trace Semantics. However, AI and PAI are not equivalent in terms of the analyses that they support. As we will show in Section 5, PAI is not only a base for probabilistic static analysis but also a suitable mathematical framework for *testing*.

4 Comparison of Probabilistic Semantics

For the language **pWhile** (with finite values or states) we have described in the above section three different semantics that we can summarise as follows.

The Kozen semantics gives the I/O behaviour of (terminating) programs, the LOS semantics gives the generator for a stepwise execution of the program (as a DTMC), and the MTS determines the possible traces and their corresponding probabilities (inherited from the scenarios).

In this section we will discuss in some detail the relationship between them with the aim of clarifying their different role in the static analysis of programs.

Kozen's Semantics and LOS. One important difference between the LOS and Kozen's semantics (Semantics 2 in [12]) is the use of labels (as a kind of program counter) to model the computational steps.

As already mentioned, in Kozen's semantics all non-terminating executions are treated equally, i.e. have a trivial or zero semantics. Another difference is that Kozen's semantics is based on a state space $\mathcal{V}(\mathbf{Value}^n)$ as opposed to the LOS state space $\mathcal{V}(\mathbf{Value})^{\otimes n}$ which allows for an independent treatment of each variable. In general, the tensor construction of the LOS allows for a kind of 'compositional' program analysis where the various syntactic components of a program can be analysed individually, which is not possible with Kozen's semantics.

In [28] we have shown that Kozen's operator $[\![P]\!]_{KFS}$ is an abstraction of a limit of of iterations of the LOS semantics $[\![P]\!]_{LOS}$. This abstraction is defined by the PAI operator which "forgets" about the computational state at all labels except ℓ:

$$\mathbf{A}_\ell = \mathbf{I} \otimes \ldots \otimes \mathbf{I} \otimes e_\ell,$$

where e_ℓ is a unit or base vector in $\mathcal{V}(\mathbf{Label})$ corresponding to label $\ell \in \mathbf{Label}$, i.e. $e_\ell = (0, 0, \ldots, 0, 1, 0 \ldots, 0)$ with only one non-zero entry for the coordinate ℓ. This can also be seen as $1 \times |\mathbf{Label}|$ matrix. This operation keeps all the information about the state, i.e. values of the variables, but only when the execution is in label ℓ. If we take for $\ell = \ell^*$, i.e. the terminal looping state in the semantics of a program, then this gives clearly the probabilities of the values of all variables for those computations which have already reached the end. So for any initial (classical) state s_0 and initial label $\ell = 0$ we can obtain the computational state in the final label ℓ^* by iteration. The following propositions holds (cf. [28]):

Proposition 1. *Given a* **pWhile** *program P and an initial (probabilistic) state s_0 in $\mathcal{V}(\mathbf{Value})^{\otimes n}$, then $(s_0 \otimes e_0)[\![P]\!]_{LOS}^t \mathbf{A}_{\ell^*}$ corresponds to the distributions over all states on which P terminates in t or fewer computational steps.*

This covers all finite computations of t steps or fewer. In order to get the I/O behaviour for all terminating computations, i.e. the Kozen semantics, we need just to consider the limit of all computations of any length:

Proposition 2. *Given a* **pWhile** *program P and an initial (probabilistic) state s_0 in $\mathcal{V}(\mathbf{Value})^{\otimes n}$, let $[\![P]\!]_{KFS}$ be Kozen's semantics of P and $[\![P]\!]_{LOS}$ the DTMC generator for P. Then*

$$(s_0 \otimes e_0)(\lim_{t\to\infty} [\![P]\!]_{LOS}^t)\mathbf{A}_{\ell^*} = s_0[\![P]\!]_{KFS}.$$

Maximal Trace Semantics and LOS. Following the programme of [36] the authors present in [7] a classical abstraction for probabilistic MTS corresponding to a “strongest postcondition semantics”. This is in effect an operator semantics which maps input distributions into some output distributions, cf. formula (2) in [7, 7.4]. A common but incorrect interpretation of the claims made in Section 7.3 of [7] is that the LOS is just an abstraction of the probabilistic MTS.

To investigate the relationship between LOS and MTS in more detail let us look at a concrete construction of probabilistic traces based on the LOS. It is somewhat not really clear if the MTS in [7] should be based on $\Sigma = \mathbf{State}$ or $\Sigma = \mathbf{Conf}$. In the first case it is straightforward to see that the LOS actually contains more information than an MTS based only on state information. We will thus consider the MTS based on $\Sigma = \mathbf{Conf}$, i.e. the reformulation of the probabilistic MTS as an element in $\mathcal{V}(\mathbf{Conf}^{+\infty})$, which associates with every possible trace a probability that this is indeed the trace which will be executed during the program run. We will then relate this set of ‘weighted’ traces to the LOS as an operator on $\mathcal{V}(\mathbf{Conf})$ where we also provide the initial distribution $s_o \otimes e_0 = \rho_0 \in \mathcal{V}(\mathbf{Conf})$.

The central issue is that the LOS allows for the construction of fronts, i.e. given an initial state we know for every time step t (in the future) all the probabilities that we will be in a certain state by considering the t iterations of the LOS (applied to the initial state). The MTS instead gives us for an initial state the probabilities of a particular execution trace (starting from that initial state). The LOS allows for the construction of a *sequence of distributions* over states (fronts) while the MTS gives a *distribution over sequences* (traces). The two notions are thus somewhat ‘orthogonal’. However for a language like **pWhile**, with a semantics that can be modelled via a DTMC, the two approaches are equivalent. This is because DTMC’s abstract from the history of a computation as only the current configuration determines the probabilities of the successor configurations. However, transition probabilities are exactly what is specified in the generator matrix of the DTMC and is all one needs to reconstruct the computational traces and their probabilities.

Instantiated for purely probabilistic languages the classical abstraction given by formula (2) in [7, 7.4] is an operator from distributions over traces to distribution transformers (for a fixed initial state/configuration s), i.e.

$$\alpha_s : \mathcal{V}(\mathbf{Conf}^{+\infty}) \to \mathcal{L}(\mathcal{V}(\mathbf{Conf}))$$

rather than $(\Omega \to \mathcal{P}(\mathbf{Conf}^{+\infty})) \to (\mathcal{V}(\mathbf{Conf}) \to \mathcal{V}(\mathbf{Conf}))$ as in [7, 7.4]. The abstraction map, if we consider the purely probabilistic case, becomes:

$$((\alpha_s(\{\langle p, X\rangle\})(\delta))(s') = \sum_{s\in\Sigma} \{\delta(s)\cdot p \mid \text{for } s\sigma s' \in X^+\}.$$

In other words, we associate to every distribution over traces $\{\langle p, X\rangle\}$ a linear operator $(\alpha_s(\{\langle p, X\rangle\}) \in \mathcal{L}(\mathcal{V}(\mathbf{Conf}))$. To see how this operator transforms a distribution $\delta \in \mathcal{V}(\mathbf{Conf})$ into another distribution $\alpha_s(\{\langle p, X\rangle\})(\delta) = \delta' \in \mathcal{V}(\mathbf{Conf})$

we describe the probability of every configuration $s' \in \mathbf{Conf}$ in the new distribution δ'. This is the sum of all products of the probabilities associated with all the traces which starting from any s reach s' in finitely many steps and the probability $\delta(s)$ that we start indeed with s. In other words the probability $\delta'(s')$ describes the probability that we terminate with s'. Therefore this abstraction gives the Kozen I/O semantics. However, it does not give the LOS which instead would require a classical abstraction of the form

$$((\bar{\alpha}_s(\{\langle p, X\rangle\})(\delta))(s') = \sum_{s \in \Sigma} \{\delta(s) \cdot p \mid \text{if } ss' \ldots \in X\}$$

i.e. an operator that collects the probabilities that in one step we reach s', rather than eventually. Note that this abstraction does not require that s' is a terminating state.

The question is if α_s or $\bar{\alpha}_s$ are actually abstractions, i.e. if indeed information is lost. If we consider the dimension of the spaces involved it seems obvious that this is the case; in fact, for a finite $\mathbf{Conf}$ with n states we have that $\dim(\mathcal{L}(\mathcal{V}(\mathbf{Conf}))) = n^2$ (as the space of $n \times n$ matrices), while for the distributions $\mathcal{V}(\mathbf{Conf}^t)$ for traces of just finite length t we have $\dim(\mathcal{V}(\mathbf{Conf})^{\otimes t}) = n^t$. So this looks like a big reduction in dimensions. However, due to the memory-less property of DTMC (that is the case we are really interested in as for programming languages the current state gives all the information we need to continue the computation) we only need to consider traces of length 2 (i.e. transition steps) when we work with the abstraction α_t. In this case we have again $\dim(\mathcal{V}(\mathbf{Conf}) \otimes \mathcal{V}(\mathbf{Conf})) = n^2$. Thus, no information is lost and the abstraction is not really an abstraction but only a recasting of the MTS. If the MTS is the most concrete semantics (in the sense of [36]) then so is the LOS. In fact, we can show the following proposition.

Proposition 3. *Given a* **pWhile** *program P, then the LOS $[\![P]\!]_{LOS}$ and the (configuration based) MTS $[\![P]\!]_{MTS}$ are equivalent, i.e. it is possible to construct either semantics from the other one.*

If we base the MTS on configurations (not just on states) then it is, on one hand, straightforward to construct the LOS operator out of the MTS by considering for all initial configurations (i.e. point-distributions) the single step traces (or single step trace-prefixes) starting from this initial configuration: The probability associated to these (short) traces is exactly the transition probability recorded in the DTMC generator, i.e. the LOS – this is indeed what the map $\bar{\alpha}_s$ above achieves. On the other hand, if we have the DTMC generator then for a trace $s_{i_1}s_{i_2}s_{i_3}\ldots$ the probability associated to this trace is the product of the transition probabilities $([\![P]\!]_{LOS})_{i_1 i_2}$, $([\![P]\!]_{LOS})_{i_2 i_3}$ etc. – i.e. $\prod_j ([\![P]\!]_{LOS})_{i_j i_{j+1}}$ – times the probabilities given by the initial distribution $\delta(s_{i_1})$.

As said before these constructions requires that the semantics of **pWhile** is modelled by a *homogenous* DTMC, i.e. that the transition probabilities from one configuration to another one do not change over time. This and the memory-less property of DTMC's seems to be a reasonable requirement for a programming language.

5 Statistical Analysis of Probabilistic Programs via PAI

Probabilistic semantics provides the basis for the static analysis of probabilistic programs. In [7] this semantics is the MTS $[\![P]\!]_{MTS}$ explained in the previous sections. The probabilistic information in such a semantics depends on the choice of an appropriate measure space which attaches probabilities to the (non)deterministic program traces.

While the AI as well as the PAI framework allow us to use traces (quasi as the 'concretest' semantics) as a basis to construct more abstract semantics, e.g. the generator of the underlying DTMC, there is an important difference: In the AI setting these traces are assumed to be the ones which are indeed the *true* ones which we obtain when a program is executed; in the PAI setting – similar to the situation in statistical analysis, learning etc. – we can attempt to utilise not just ideal traces but also experimentally *observed*, maybe *corrupted* traces (which might be *distorted* by noise) in order to (re)construct the (most plausible) underlying abstract semantics (e.g. the DTMC generator).

In this section we show an approach where the probabilistic information about the program's executions is inferred by *observing* some sample runs. This way we establish in some sense a link between static program analysis and testing. Based on this, we can use PAI to calculate best estimates of the program's properties in a way similar to the so-called *linear statistical model* or, equivalently, the *linear regression* method.

The approach we are going to present is based on the idea of identifying observations with a linear combination of a set of random variables x_i, whose weights are chosen with the method of least squares so as to minimise the distance from the observations and the actual model expressing the program's behaviour. Thus the framework of Probabilistic Abstract Interpretation is particularly appropriate as a base of this approach.

5.1 The Linear Statistical Model

In several contexts it is often useful to predict or estimate a variable β (or a vector of variables), given that we have the opportunity to observe variables $y_1, y_2, \ldots, y_n$ which somehow (statistically) depend on β. This is a very important statistical problem which is typically faced by using so-called linear regression analysis, also known as linear statistical model (cf e.g. [11], [10, Section 8.3] or [8, Section 6.4]). This widely used statistical technique applies to situations such as the one mentioned above, where a random vector y depends *linearly* on a vector of parameters β, i.e. (using post-multiplication)

$$y = \beta \mathbf{X} + \varepsilon, \tag{1}$$

where y represents some measurement results and the parameters β are unknown, the matrix $\mathbf{X}$ is the *design matrix*, and ε is a random vector representing the errors of observing y which is conventionally assumed to have expected value equal to zero (as well as some further mild statistical conditions regarding the variance and co-variance of ε). These requirements mean that there is no underlying or systematic reason for the distortions ε (but only 'random' noise).

The role of least square approximations and the Moore-Penrose pseudo-inverse in this context is of particular relevance for the well-known Gauss-Markov theorem (e.g. [10, Section 8.3, Thm. 1] for $B = \mathbf{I}$, i.e. a *naive least-square estimator*):

Theorem 1 (Gauss-Markov). *Consider the linear model $y = \beta\mathbf{X} + \varepsilon$ with $\mathbf{X}$ of full column rank and ε fulfilling the conditions in [10, Section 8.3]. Then the* Best Linear Unbiased Estimator *(BLUE) is given by*

$$\hat{\beta} = y\mathbf{X}^{\dagger}.$$

In its simplest version, the Gauss-Markov theorem thus asserts that the best estimate $\hat{\beta}$ of the unknown parameters β can be obtained from some experimentally observed y by calculating $y\mathbf{X}^{\dagger}$, i.e. via the Moore-Penrose pseudo-inverse of the design matrix $\mathbf{X}$, cf. [10, Section 8.3, eqn (35)].

5.2 Application to Security Analysis

In order to give a motivation for why the reconstruction of unknown parameters or properties of a (computational) system is relevant let us consider an example from Computer Security, namely Kocher's attack on crypto protocols [43].

Modular exponentiation is a basic operation for computing the private key in crypto systems using the Diffie-Hellman or the RSA protocols. In [43], it is shown that by carefully measuring the time required to perform such an operation, an attacker may be able to find the Diffie-Hellman exponents or factor the RSA keys and break the cryptosystems.

The crucial point is the estimation of a single bit b in the secret key k. Since modular exponentiation takes very different execution times depending on the value of a certain bit b being 0 or 1, what the attacker needs are good estimate of these execution times in order to deduce the value of each bit of the key. Thus, linear statistical models play a crucial role in the analysis of security. We show how the problem of the timing attacks can be described as a statistical analysis problem, by using as an example a simplified implementation of the RSA exponentiation algorithm. By doing this we can also sketch the relationship between PAI and linear regression.

Suppose that t_0 is the time it takes to perform multiplication in the modular exponentiation procedure if a single bit b of the cryptographic key k is $b = 0$ and t_1 if $b = 1$. We thus need to consider two possible DTMC models (or generators) one for the case $b = 0$ and one $b = 1$ which realise the corresponding exponentiation processes (incorporating maybe also some noise which we do not control which is due to the fact that the physical device we observe is also involved in other tasks/threads like network communication etc.). If we observe the (maybe distorted) running time (without knowing b) the aim is to guess correctly which of the two models is actually being executed (i.e. the value of

b). We can also set the vector β to represent the strength/weights/probabilities that in a given model $b = 0$ or $b = 1$, respectively.

More concretely, we can set the vector $\beta_0 = (1, 0)$ to represent the models of the system where the bit b of the key is $b = 0$ and $\beta_1 = (0, 1)$ to the key with $b = 1$. We can now define a linear statistical model by constructing a design matrix $\mathbf{X}$ (in the PAI sense a concretisation operator), which maps a model (element in the abstract domain) onto its timing behaviour (element in the concrete domain). As an example, we can consider the situation where we can observe ten possible execution times t_i that we enumerate and use as column indices for $\mathbf{X}$. Suppose that t_0 corresponds to the 3rd and t_1 to the 7th column in this enumeration. In this case we obtain a design matrix of the form:

$$\mathbf{X} = \begin{pmatrix} 0\,0\,1\,0\,0\,0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0\,0\,1\,0\,0\,0 \end{pmatrix},$$

and we can calculate

$$\beta_0 \mathbf{X} = (1, 0) \cdot \mathbf{X} = (0, 0, 1, 0, 0, 0, 0, 0, 0, 0)$$

which tells us that for $b = 0$ the chances of observing any other time signature than t_0 is zero, and that t_0 will definitively be observed. Similarly, we could have also used $\beta_1 = (0, 1)$.

If we begin instead by observing the time behaviour, i.e. if we test the program and obtain, for example, an (undistorted) observation vector of the form

$$y = (0, 0, 1, 0, 0, 0, 0, 0, 0, 0),$$

then, by calculating

$$y\mathbf{X}^\dagger = (1, 0)$$

we will get that b is definitely 0. If we now add a (Gaussian) error to our experiment then the observed times, corresponding to an estimate y would perhaps be something like (cf. Figures 1 and 2 in [43]):

$$\hat{y} = (0.1, 0.2, 0.7, 0.2, 0.1, 0, 0, 0, 0, 0)$$

i.e. we get a distorted observation result (e.g. because in 10 measurements we have observed once the first possible time, twice the second, etc.) The estimation (based on these observations) leads to a guess of the weights of the parameters in β that we calculate as

$$\hat{y}\mathbf{X}^\dagger = (0.7, 0).$$

This result reflects the fact that it is very likely that the value of bit b is 0 as we have observed, although with some errors, a time behaviour where the times cluster around the value t_0.

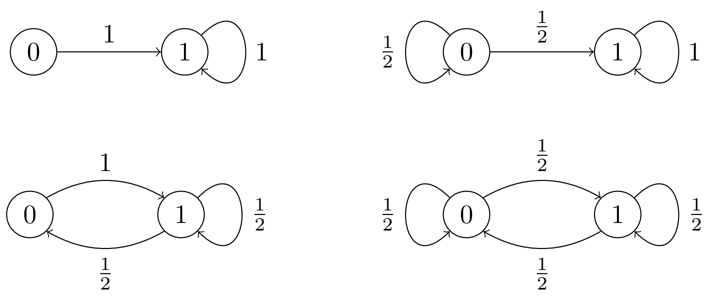
5.3 Abstraction and Linear Regression

Our aim is to show how statistics can be used in static analysis in all those cases where we have some observations at hand and we want to use them in order to improve the precision of the analysis. To this purpose we can use the theory of linear regression in order to determine a best estimate of the model underlying those observations, e.g. the DTMC generator that with highest probability produces the traces that we observe.

Note that classical abstract interpretation cannot be used in this scenario even in its probabilistic re-formulation as given e.g. in [7]; this is because the safety constraint at the base of the framework does not permit the consideration of expectation values in the analysis result, as these would not guarantee the (classical) correctness of the analysis (cf. Section 3).

In the setting of linear statistical models, the concretisation operator $\mathbf{G}$ of the PAI framework is a mapping from an abstract domain consisting of all possible Markov models for the observed program to all possible observable traces corresponding to the different runnings of the program. $\mathbf{G}$ plays the role of the design matrix of the statistical model. Thus, if y is a vector defining the probabilities of certain traces according to some observations and β represents a parameterised Markov model, then we can use the linear statistical equation (1) in its simplest instance, i.e. with $\varepsilon = 0$, $\mathbf{y} \in \mathbb{R}^n, \mathbf{X} \in \mathbb{R}^{n\times p}$ and $\beta \in \mathbb{R}^p$, and obtain the best estimate of the concrete Markov model by $\hat{\beta} = \mathbf{y}\mathbf{X}^\dagger$.

Example 8. Consider the following simple examples of DTMC's:



The generator matrices of these probabilistic transition relations are

$$\mathbf{T}_{0,1} = \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} \quad \mathbf{T}_{\frac{1}{2},1} = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ 0 & 1 \end{pmatrix} \quad \mathbf{T}_{0,\frac{1}{2}} = \begin{pmatrix} 0 & 1 \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix} \quad \mathbf{T}_{\frac{1}{2},\frac{1}{2}} = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix}$$

Clearly with $\mathbf{T}_{\frac{1}{2},\frac{1}{2}}$ we can generate all infinite 0/1 sequences. Note that since these are uncountably many, the probability structure on the maximal trace space will thus require a measure theoretical modelling.

These DTMC's are in essence the processes (for different values of p and q) which describe the core (loop body) of our Example 1 in the LOS or Kozen semantics, cf. also $[\![S]\!]_{KFS}$ in Example 3.

The processes above depend on the parameters p and q in the real interval $[0, 1]$ which we can see as the probability to remain in state 0 and state 1, respectively. They are represented by the DTMC

$$\begin{tikzcd} 0 \arrow[loop left, "p"] \arrow[r, bend left, "1-p"] & 1 \arrow[l, bend left, "1-q"] \arrow[loop right, "q"] \end{tikzcd}$$

with generator

$$\mathbf{T}_{pq} = \begin{pmatrix} p & 1-p \\ 1-q & q \end{pmatrix}$$

Clearly any property of a program whose behaviour can be described as above depends on the parameters p and q. Moreover, observing the property may be influenced by some distorted execution of $\mathbf{T}_{pq}$. By applying the statistical linear model we can find best estimates for the parameters p and q. In the next example we show that this corresponds to performing a static analysis based on PAI. The application of the linear statistical model allows us to compute best estimates for some properties of the models encoded in a vector β. Alternatively, we can use the linear statistical model to establish a weighting β of some given DTMC's generating some observed traces. These weighting are guaranteed to be the best linear unbiased estimators by the least squares property of the Moore-Penrose pseudo-inverse and can be used as a base for trade-off or speculative analysis.

In the following example we show how a concrete and abstract space for PAI can be constructed for supporting statistical analysis.

Example 9. Consider the DTMC in Example 8 with DTMC generator

$$\mathbf{T}_{pq} = \begin{pmatrix} p & 1-p \\ 1-q & q \end{pmatrix}$$

This system is completely specified when both the values of p and q and the initial state s are specified. Thus we can identify the abstract semantic domain with the set of all pairs of initial states $s \in \{0, 1\}$ and matrices $\mathbf{T}_{pq}$ of the form above.

$$\mathcal{M} = \{\langle s, \mathbf{T}_{pq}\rangle\} = \left\{\langle s, \begin{pmatrix} p & 1-p \\ 1-q & q \end{pmatrix}\rangle\right\}$$

or equivalently with the set of triples $\mathcal{M} = \{\langle s, p, q\rangle \mid s \in \{0, 1\}, p, q \in [0, 1]\}$.

Note that this parametric DTMC generator encodes the same information as the set of all parametric traces starting from any initial state (cf. Section 4). In order to apply PAI we consider the distributions over $\mathcal{M}$, i.e. the normalised, positive elements in the vector space over $\mathcal{M}$, i.e. we take $\mathcal{D} = \mathcal{V}(\mathcal{M})$.

The concrete domain consists of the set of all (infinite) sequences of 0 and 1, $\mathcal{T} = \Sigma^{+\infty} = \{0, 1\}^{+\infty}$, representing the execution traces resulting from fixing actual values of the parameters p and q and the input state. The concrete domain of PAI is therefore the space of distributions on traces $\mathcal{C} = \mathcal{V}(\mathcal{T})$.

$$\mathbf{G} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & \frac{1}{4} & \frac{1}{4} \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

s	p	q
0	0	0
1	0	0
0	$\frac{1}{2}$	0
1	$\frac{1}{2}$	0
0	1	0
1	1	0
0	0	$\frac{1}{2}$
1	0	$\frac{1}{2}$
0	$\frac{1}{2}$	$\frac{1}{2}$

s	p	q
1	$\frac{1}{2}$	$\frac{1}{2}$
0	1	$\frac{1}{2}$
1	1	$\frac{1}{2}$
0	0	1
1	0	1
0	$\frac{1}{2}$	1
1	$\frac{1}{2}$	1
0	1	1
1	1	1

$trace$
0 0 0
0 0 1
0 1 0
0 1 1
1 0 0
1 0 1
1 1 0
1 1 1

$$\mathbf{G}^\dagger = \begin{pmatrix} 0 & 0 & 0 & 0 & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 & \frac{1}{3} & 0 \\ -\frac{2}{3} & 0 & \frac{4}{3} & 0 & -\frac{1}{3} & 0 & -\frac{2}{3} & 0 & \frac{4}{3} & 0 & -\frac{1}{3} & 0 & -\frac{2}{3} & 0 & \frac{4}{3} & 0 & -\frac{1}{3} & 0 \\ \frac{11}{15} & 0 & \frac{1}{5} & 0 & 0 & 0 & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 & -\frac{1}{15} & 0 & -\frac{1}{5} & 0 & 0 & 0 \\ -\frac{1}{15} & 0 & -\frac{1}{5} & 0 & 0 & 0 & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 & \frac{11}{15} & 0 & \frac{1}{5} & 0 & 0 & 0 \\ 0 & -\frac{1}{15} & 0 & \frac{1}{3} & 0 & \frac{11}{15} & 0 & -\frac{1}{5} & 0 & 0 & 0 & \frac{1}{5} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{11}{15} & 0 & \frac{1}{3} & 0 & -\frac{1}{15} & 0 & \frac{1}{5} & 0 & 0 & 0 & -\frac{1}{5} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -\frac{2}{3} & 0 & -\frac{2}{3} & 0 & -\frac{2}{3} & 0 & \frac{4}{3} & 0 & \frac{4}{3} & 0 & \frac{4}{3} & 0 & -\frac{1}{3} & 0 & -\frac{1}{3} & 0 & -\frac{1}{3} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 & \frac{1}{3} \end{pmatrix}$$

Fig. 1. Relating models and traces (for the cut-down version)

Numerical Experiments. Even for the simple example given above, the sets involved are uncountably infinite. In order to be able to compute an analysis of the system in Example 8 we will consider here the simple case where transition probabilities can only assume values in a finite set, i.e. $p, q \in \{p_0, \ldots, p_n\}$ and where traces can only be of length t, for a given t.

If we consider $p, q \in \{0, \frac{1}{2}, 1\}$, we obtain 9 possible semantics, i.e. DTMC generators while we can choose among two possible initial states, namely either 0 or 1. We will also restrict to traces of length $t = 10$. The abstract domain is thus specified by $\mathcal{D} = \mathcal{V}(\{0,1\}) \otimes \mathcal{V}(\{0, \frac{1}{2}, 1\}) \otimes \mathcal{V}(\{0, \frac{1}{2}, 1\}) = \mathbb{R}^2 \otimes \mathbb{R}^3 \otimes \mathbb{R}^3 = \mathbb{R}^{18}$, while the concrete traces are modelled by $\mathcal{C} = \mathcal{V}(\{0,1\}^{10}) = \mathcal{V}(\{0,1\})^{\otimes 10} = (\mathbb{R}^2)^{\otimes 10} = \mathbb{R}^{1024}$. We can now construct a linear map $\mathbf{G} : \mathcal{D} \to \mathcal{C}$ that associates to each instance model and initial input the distributions over the traces that are obtained in that model. As we have 18 possible models and 1024 possible traces, the operator $\mathbf{G}$ is given by the 18×1024 matrix.

As it is impossible to present here the actual 18×1024 matrix $\mathbf{G}$ (due to its size) we just try to illustrate how it looks like by restricting us to the case of 8 possible traces of at most 3 steps. However, it should be stressed that in this case the (abstract) space of models $\mathcal{M}$ is larger than that of the (concrete) space of traces $\mathcal{T}$, which is not what we have in mind. For this cut-down case we can explicitly state the matrix $\mathbf{G}$ with rows representing the possible instance models and the columns representing the possible traces as depicted in Figure 1. The entries of this matrix specify the probabilities that a given model (row) generates a certain trace (column). For example, the entry $\mathbf{G}_{33} = \frac{1}{2}$ means that with the third model in the enumeration given above, i.e. for initial state $s = 0$, $p = \frac{1}{2}$ and $p = 0$, we get the third trace, i.e. 010, with probability $\frac{1}{2}$.

We then compute the Moore-Penrose pseudo-inverse of the $\mathbf{G}$ matrix which we can use to calculate the best estimator of the parameters p and q. Intuitively, $\mathbf{G}^\dagger$ gives us the probabilities that when a certain trace is observed this comes from a certain model. In order to illustrate how $\mathbf{G}^\dagger$ looks like we also present it for the cut-down version in Figure 1.

We undertook some numerical experiments and we report below some of results we obtained. For $p = 0 = q$ we simulated 10000 times the execution of the system (including random noise) and observed traces with a length $t = 10$. This gives a concrete domain of $\dim(\mathcal{C}) = 2^{10} = 1024$, i.e. there are about one thousand possible traces we can observe. We have considered DTMC without distortion, i.e. no error, as well as the case where a noise of strength ε was applied according to a normal distribution (mean zero and variance one, cf. `randn()` in Octave 3.8.0, p391). The simulations resulted in an estimated distribution over all possible 2^{10} traces. For the undistorted case we denote this distribution vector by y, for $\varepsilon = 0.01$ by y' and for $\varepsilon = 0.1$ by y''. The initial state was always chosen with probability $\frac{1}{2}$ as state 0 or state 1. The resulting distributions over possible models we obtained are listed below:

$$y\mathbf{G}^\dagger = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$
$$y'\mathbf{G}^\dagger = \begin{pmatrix} 0.48 & 0.48 & 0.02 & 0.01 & 0 & 0 & 0.01 & 0.02 & -0.02 & -0.01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$
$$y''\mathbf{G}^\dagger = \begin{pmatrix} 0.33 & 0.33 & 0.17 & 0.11 & 0 & 0 & 0.11 & 0.18 & -0.12 & -0.12 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

These results mean that if we observe the undisturbed DTMC in order to obtain experimentally the probabilities for all possible 2^{10} traces then we can identify the underlying model uniquely. The $\hat{\beta} = y\mathbf{G}^\dagger$ describes an "estimate" corresponding to the statement that the actual process which was running when we observed y is a mixture of the first two models which are given by $s = 0$ or $s = 1$ and $p = 0$ and $q = 0$. Indeed these are exactly the two models we used in the simulation.

For $\varepsilon = 0.01$ we also identify the unknown system (parameters) with high probability but there is some tendency to mismatch the observations with a system or model that was actually *not* used in the simulation (namely the 3rd, 4th, etc model with either p or q being equal to $\frac{1}{2}$ – interestingly, the case were $p = \frac{1}{2} = q$ gets a negative weighting).

If we increase the error term in the simulation, i.e. for the distortion $\varepsilon = 0.1$, the possibility of a wrong identification of the actual model(s) is reinforced: The weights associated to the actual systems (i.e. β_1 and β_2) decreases further, while other possible models get stronger. Clearly, if we further increase ε the estimate for β will still be the optimal one (BLUE) but ultimately it will not allow any meaningful identification of the actual system – we will get only (white) noise. We also get the same (qualitative) results for other choices of p and q.

6 Conclusions

We have presented a comparison of three different probabilistic semantics: (i) Kozen's I/O Fixed-Point Semantics, (ii) a Linear Operator Semantics previously introduced by the authors, and finally (iii) a probabilistic version of the Maximal Trace Semantics. We have argued that Kozen's semantics can be recovered from the LOS as an abstract limit of the LOS (cf. [28]) and that the abstraction α_s in [7, Section 7.4] in fact gives Kozen's semantics (by collecting the information/probability along finite, i.e. terminating, traces in the MTS) rather than the LOS. We also demonstrated that the LOS in fact contains more information than the MTS (namely information about the label or program counter) but that otherwise LOS and MTS are essentially equivalent.

The second part of this paper relates the Probabilistic Abstract Interpretation framework introduced in [5] with the most widely used statistical technique, namely Linear Regression. We have already shown in [5] that classical Abstract Interpretations can be recovered (as the support of a PAI) from a Probabilistic Abstract Interpretation. We have extended the (re)construction of the LOS from the MTS alluded to in [7] – though this involves the "abstraction" $\bar{\alpha}_s$ rather than α_s – to deal also with distorted observation of traces. This allows for a bridge between statistics (testing) and static program analysis. Intended application areas include problems in computer security like covert channels and non-interference notions reinterpreted as concepts related to system identification.

Our semantical presentation was restricted to finite state spaces. However a full treatment of the different semantical models is possible though slightly more complex as it involves a deeper study of the underlying measure-theoretic notion (e.g. the σ-algebras generated by trace pre-fixes) as well as topological notions (e.g. Hilbert vs Banach spaces and their operators, weak limits etc., cf. [28]). Another restriction is the exclusion of (pure) non-determinism in our model. The reason for this is more a conceptual than a technical one (though the absence of a normalisation condition seems to require a consideration, for example, of unbounded operators for pure non-determinism). Arguably, we can simulate non-determinism by parametric, unknown probabilities.

Finally, it might be worth pointing out the rich literature on filtering, system identification, Hidden Markov Models (e.g. [44–46]), and related topics which we did not discuss but which are clearly related. Our approach to Linear Regression could be considered to be very simple and basic. However, we think it is worth to highlighten the relationship between PAI and statistics. Given the role of least square methods – i.e. the Moore-Penrose pseudo-inverse – play in control theory etc. – for example, for the well-known and celebrated technique of Kalman filters [11] – we aim to further explore this field.

References

1. Jones, N.D., Nielson, F.: Abstract Interpretation: A Semantics-Based Tool for Program Analysis. In: Handbook of Logic in Computer Science. Clarendon Press, Oxford (1985) 527–636
2. Nielson, F.: Strictness analysis and denotational abstract interpretation. Information and Computation **76**(1) (1988) 29 – 92
3. Nielson, F., Nielson, H.R.: Infinitary control flow analysis: a collecting semantics for closure analysis. In: Proceedings of POPL'97, ACM Press (1997) 332–345
4. Nielson, F., Nielson, H.R., Hankin, C.: Principles of Program Analysis. Springer Verlag (1999)
5. Di Pierro, A., Wiklicky, H.: Concurrent Constraint Programming: Towards Probabilistic Abstract Interpretation. In: Proceedings of PPDP'00, ACM (2000) 127–138
6. Di Pierro, A., Wiklicky, H.: Measuring the precision of abstract interpretations. In: Proceedings of LOPSTR'00. Volume 2042 of LNCS., Springer Verlag (2001) 147–164
7. Cousot, P., Monerau, M.: Probabilistic abstract interpretation. In Seidel, H., ed.: Proceedings of ESOP'12. Volume 7211 of LNCS., Springer Verlag (2012) 166–190
8. Campbell, S.L., Meyer, C.D.: Generalized Inverses of Linear Transformations. Pitman – Dover, London (1979)
9. Deutsch, F.: Best Approximation in Inner-Product Spaces. Springer Verlag (2001)
10. Ben-Israel, A., Greville, T.N.E.: Gereralized Inverses – Theory and Applications. second edn. CMS Books in Mathematics. Springer Verlag (2003)
11. Albert, A.: Regression and the Moore-Penrose Pseudoinverse. Volume 94 of Mathematics in Science and Engineering. Elsevier (1972)
12. Kozen, D.: Semantics of probabilistic programs. J. Comput. Syst. Sci. **22**(3) (1981) 328–350
13. Di Pierro, A., Sotin, P., Wiklicky, H.: Relational analysis and precision via probabilistic abstract interpretation. In: Proceedings of QAPL'08. Volume 220(3) of ENTCS., Elsevier (2008) 23–42
14. 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 Verlag (2010) 1–42
15. Di Pierro, A., Hankin, C., Wiklicky, H.: Probabilistic lambda calculus and quantitative program analysis. Journal of Logic and Computation **15**(2) (2005) 159–179
16. Ramsey, N., Pfeffer, A.: Stochastic lambda calculus and monads of probability distributions. ACM SIGPLAN Notices **37**(1) (2002) 154–165
17. Pfeffer, A.: Practical Probabilistic Programming. Manning (2015)
18. Di Pierro, A., Hankin, C., Wiklicky, H.: Probabilistic linda-based coordination languages. In de Boer, F., Bonsangue, M., Graf, S., de Roever, W.P., eds.: Proceedings of FMCO 2004. Volume 3657 of LNCS., Springer Verlag 120–140
19. Priami, C.: Stochastic π-calculus. Computer Journal **38**(7) (1995) 578–589
20. Hillston, J.: A Compositional Approach to Performance Modelling. Cambridge University Press (1996)
21. Di Pierro, A., Hankin, C., Wiklicky, H.: A systematic approach to probabilistic pointer analysis. In: Proceedings of APLAS'07. Volume 4807 of LNCS., Springer Verlag (2007) 335–350
22. Jones, C., Plotkin, G.D.: A probabilistic powerdomain of evaluations. In: Proceedings of LICS'89, IEEE (1989) 186–195

23. Jones, C.: Probabilistic Non-determinism. PhD thesis, University of Edinburgh (1989)
24. Jung, A., Tix, R.: The troublesome probabilistic powerdomain. ENTCS **13** (1998) 70–91
25. Morgan, C., McIver, A., Seidel, K.: Probabilistic predicate transformers. ACM Transactions on Programming Languages and Systems **18**(3) (1996) 325–353
26. Gretz, F., Katoen, J.P., McIver, A.: Operational versus weakest pre-expectation semantics for the probabilistic guarded command language. Performance Evaluation **73** (2014) 110132
27. Park, S., Pfenning, F., Thrun, S.: A probabilistic language based upon sampling functions. In: Proceedings of POPL'05, ACM (2005) 171–182
28. Di Pierro, A., Wiklicky, H.: Semantics of probabilistic programs: A weak limit approach. In chieh Shan, C., ed.: Proceedings of APLAS13. Volume 8301 of LNCS., Springer Verlag (2013) 241–256
29. Lax, P.D.: Functional Analysis. Pure and Applied Mathematics. John Wiley & Sons (2002)
30. Kubrusly, C.S.: The Elements of Operator Theory. second edn. Birkhäuser (2011)
31. Greub, W.H.: Linear Algebra. Springer Verlag (1967)
32. Goebel, K., Kirk, W.: Topics in Metric Fixed Point Theory. Volume 28 of Cambridge Studies in Advanced Mathematics. Cambridge University Press, Cambridge (1990)
33. Roman, S.: Advanced Linear Algebra. 2nd edn. Springer Verlag (2005)
34. Kadison, R., Ringrose, J.: Fundamentals of the Theory of Operator Algebras: Elementary Theory. AMS (1997) reprint from Academic Press edition 1983.
35. Cousot, P., Cousot, R.: Systematic design of program transformation frameworks by abstract interpretation. In: Proceedings POPL'02, ACM Press (2002) 178–190
36. Cousot, P.: Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. Theoretical Computer Science **277**(1–2) (2002) 47–103
37. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press (2008)
38. Billingsley, P.: Probability and Measure. John Wiley & Sons (1979)
39. Klenke, A.: Probability Theory - A Comprehensive Course. Springer Verlag (2006)
40. Cousot, P., Cousot, R.: Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In: Proceedings of POPL'77. (1977) 238–252
41. Monniaux, D.: Abstract interpretation of probabilistic semantics. In: SAS'00. Volume 1824 of LNCS., Springer Verlag (2000) 322–339
42. Di Pierro, A., Hankin, C., Wiklicky, H.: Abstract interpretation for worst and average case analysis. In Reps, T., Sagiv, M., Bauer, J., eds.: Program Analysis and Compilation, Theory and Practice: Essays dedicated to Reinhard Wilhelm. LNCS 4444. Springer-Verlag (2007) 160–174
43. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. Lecture Notes in Computer Science **1109** (1996) 104–113
44. Crassidis, J.L., Junkins, J.L.: Optimal Estimation of Dynamic Systems. Chapman & Hall/CRC Applied Mathematics & Nonlinear Science. CRC Press (2004)
45. Verhaegen, M., Verdult, V.: Filtering and System Identification: A Least Squares Approach. Cambridge University Press (2007)
46. Rao, C.R., Toutenburg, H., Shalabh, Heumann, C.: Linear Models and Generalizations: Least Squares and Alternatives. 3rd edn. Springer Series in Statistics. Springer Verlag (2008)