On Dynamical Probabilities, or: How to Learn to Shoot Straight

Herbert Wiklicky(✉)

Department of Computing, Imperial College London, London, UK
herbert@doc.ic.ac.uk

Abstract. In order to support, for example, a quantitative analysis of various algorithms, protocols etc. probabilistic features have been introduced into a number of programming languages and calculi. It is by now quite standard to define the formal semantics of (various) probabilistic languages, for example, in terms of Discrete Time Markov Chains (DTMCs). In most cases however the probabilities involved are represented by constants, i.e. one deals with static probabilities. In this paper we investigate a semantical framework which allows for changing, i.e. dynamic probabilities which is still based on time-homogenous DTMCs, i.e. the transition matrix representing the semantics of a program does not change over time.

1 Introduction

Over the last 20 years or so probabilistic programming languages, model checking, programming, semantics etc. have become more and more popular. It appears now to be rather straight forward to add probabilities to any language, formalism, calculus, etc. one might be interested in. Most "probabilistic" programming languages, etc. however use **constant** probabilities [10,11] etc., as we also did in our own work [4,7], especially when anything beyond a simple operational semantics is considered.

One of the motivations for introducing probabilities, as a form of quantified non-determinism, into a programming language is to allow for the formulation and analysis of so-called "randomised algorithms" [13], i.e. algorithms where chance is exploited in order to obtain a certain result, may it be probabilistic primality tests, Monte Carlo integration, etc.

However, there is a large class of randomised algorithms in the area of stochastic programming which have dynamic probabilities at their core, such a stimulated annealing, the Metropolis algorithm, Boltzmann machines, etc. [1,17]. All of these try to find global optimal solutions and in order to avoid getting trapped into a local minima (as might, for example, be the effect of a steepest gradient method) there are random perturbations. The effect of these perturbations is decreasing over time, i.e. during optimisation the chances of a perturbation changes slowly to become zero. Without going into the details of such "cooling"

A. Lluch Lafuente and J. Proença (Eds.): COORDINATION 2016, LNCS 9686, pp. 262–277, 2016.
DOI: 10.1007/978-3-319-39519-7_16

schemes or schedules we are in this paper interested in how to formalise dynamically changing probabilities in an appropriate semantical model.

Probabilistic features, e.g. choices, introduce also a subtle, nevertheless extremely important form of coordination. Probabilities have to be normalised, not as a formal requirement but quasi because of the fundamental laws of nature: Something must happen, so the probabilities of all possibilities at any moment must add up to one. Thus, whatever model we employ in order to describe probabilistic choices, assignments, etc. the different options or possibilities are "communicating" in some form via their probabilities: if one option becomes more likely, another one must give up its chances to be executed/realised.

2 A Probabilistic Language

In the following we will denote by $\mathbf{Var} = \{x_1, \ldots, x_v\}$ the set of all variables of a program P and by $\mathbf{Value}(x)$ the range of possible values of a variable x.

Technical restriction: In this paper we assume that $\mathbf{Value}(x)$ is finite for all $x \in \mathbf{Var}$. We will allow below for variables as probabilities which thus also will have to come from a finite set of (possible) values. From a computational point of view probabilities should in any case perhaps be modeled as rational numbers in $[0, 1]$. Using real numbers can, as always, create a number of fundamental problems related to computability etc., e.g. [19].

To simplify the presentation we will go even a step further and only consider positive *integers* in $\mathbb{Z}^+$ as "*weights*"[1]: Given several options with "weights" w_i these correspond to probabilities $p_i = w_i / \sum_j w_j$. As we have to (re)normalise probabilities in any case (even for static probabilities as constants, unless we can trust the programmer that all probabilities in a choice or a probability distribution always add up to one) this does not imply any restriction. It only means that in effect we consider *proportions* or *ratios* rather than rational values.

Conceptual restriction: We do not allow for any kind of pure "non-determinism" as part of the actual execution of the program. The reasons for this are: (i) From a conceptual point of view it seems to be a contradiction to the notions like that of a Turing machine as an unambiguous procedure ("Entscheidungsproblem") to allow for (e.g. angelic) "non-determinism"; (ii) we also do not believe that any physical implementation of a purely "non-deterministic" choice exists (e.g. one could use quantum devices to realise probabilistic but never "non-deterministic" choices); and (iii) there are several mathematical (pseudo-)problems which disappear when one eliminates " non-determinism" during the execution of a program (e.g. related to boundedness, etc. [7]).

However, our semantical model still accommodates "non-determinism" in several aspects such as "non-determinism" as "under-specification" and "openness". Concretely, the semantical model provides for our language (pure) "non-determinism" in two ways: (i) We leave it open which initial configuration will

[1] Weights however have to be distinguished from *priorities* in other contexts.

be used, as we have no further interaction with the environment this can also be seen as allowing for an "open" system; and (ii) we also allow for parameters as probabilities, i.e. our semantics allows for "under-specification" in the sense that the concrete probabilities are only determined in a concrete implementation.

2.1 Syntax

The syntax of statements in our language **pWhile** is given in Table 1. We also provide a labelled version of this syntax (cf. [14]) in order to be able to refer to certain program points in a program analysis context, see also Table 1. We will denote by **Label** the set of all labels of a program. For details on expressions $f(x_1, \ldots, x_n)$ (also sometimes denoted simply by e) etc. we refer to e.g. [5,14].

Table 1. The syntax of **pWhile**

$$
\begin{array}{l|l}
S ::= \texttt{skip} & S ::= [\texttt{skip}]^\ell \\
\quad | \quad x := f(x_1, \ldots, x_n) & \quad | \quad [x := f(x_1, \ldots, x_n)]^\ell \\
\quad | \quad S_1 ; S_2 & \quad | \quad S_1 ; S_2 \\
\quad | \quad \texttt{choose } p_1 : S_1 \texttt{ or } p_2 : S_2 \texttt{ ro} & \quad | \quad [\texttt{choose}]^\ell \; p_1 : S_1 \texttt{ or } p_2 : S_2 \texttt{ ro} \\
\quad | \quad \texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2 \texttt{ fi} & \quad | \quad \texttt{if } [b]^\ell \texttt{ then } S_1 \texttt{ else } S_2 \texttt{ fi} \\
\quad | \quad \texttt{while } b \texttt{ do } S \texttt{ od} & \quad | \quad \texttt{while } [b]^\ell \texttt{ do } S \texttt{ od}
\end{array}
$$

For this language we have the usual intuitive semantics: We have an "empty" `skip` statement, assignment to variables, sequential composition as well as `if` statements and `while` loops. The only probabilistic construct is the `choose` statement which executes S_1 or S_2 according to the probabilities p_1 and p_2 (which we assume to be normalised, i.e. $p_1 + p_2 = 1$, or which will be (re)normalised as part of the execution of the program, mor below). The `choose` statement can also be extended from its binary version to an n-ary one. We will not consider in this core language random assignments – as in some of our other papers or, e.g., [11] – but just note that obviously one can implement a random assignment (involving finite values) using the `choose` construct.

2.2 Operational Semantics

The SOS semantics for **pWhile** is given in Table 2. We use the (additional) statement `stop` to indicate successful termination and (re)normalise probabilities in **R7**, otherwise these are the usual SOS rules for procedural languages. The operational (SOS) semantics of **pWhile** is defined in terms of a probabilistic transition system on configurations. A *configuration* is a pair $\langle S, s \rangle \in \mathbf{Conf}$ with S a statement in **pWhile** and $s \in \mathbf{State}$ a (classical) *state*, i.e. a function $\mathbf{Var} \to \mathbf{Value}$. The SOS semantics is essentially also the same for the labelled version of the language, in this case we can however simplify the presentation

Table 2. The rules of the SOS semantics of **pWhile** (static)

$$\mathbf{R1}\ \langle \texttt{stop}, s\rangle \longrightarrow_1 \langle \texttt{stop}, s\rangle \qquad \mathbf{R4}_1\ \frac{\langle S_1, s\rangle \longrightarrow_p \langle S_1', s'\rangle}{\langle S_1; S_2, s\rangle \longrightarrow_p \langle S_1'; S_2, s'\rangle}$$

$$\mathbf{R2}\ \langle \texttt{skip}, s\rangle \longrightarrow_1 \langle \texttt{stop}, s\rangle$$

$$\mathbf{R3}\ \langle v\ \texttt{:=}\ e, s\rangle \longrightarrow_1 \langle \texttt{stop}, s[v \mapsto \mathcal{E}(e)s]\rangle \qquad \mathbf{R4}_2\ \frac{\langle S_1, s\rangle \longrightarrow_p \langle \texttt{stop}, s'\rangle}{\langle S_1; S_2, s\rangle \longrightarrow_p \langle S_2, s'\rangle}$$

$$\mathbf{R5}_1\ \langle \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2\ \texttt{fi}, s\rangle \longrightarrow_1 \langle S_1, s\rangle \quad \text{if } \mathcal{E}(b)s = \texttt{true}$$

$$\mathbf{R5}_2\ \langle \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2\ \texttt{fi}, s\rangle \longrightarrow_1 \langle S_2, s\rangle \quad \text{if } \mathcal{E}(b)s = \texttt{false}$$

$$\mathbf{R6}_1\ \langle \texttt{while}\ b\ \texttt{do}\ S\ \texttt{od}, s\rangle \longrightarrow_1 \langle S;\ \texttt{while}\ b\ \texttt{do}\ S\ \texttt{od}, s\rangle \quad \text{if } \mathcal{E}(b)s = \texttt{true}$$

$$\mathbf{R6}_2\ \langle \texttt{while}\ b\ \texttt{do}\ S\ \texttt{od}, s\rangle \longrightarrow_1 \langle \texttt{stop}, s\rangle \quad \text{if } \mathcal{E}(b)s = \texttt{false}$$

$$\mathbf{R7}_1\ \langle \texttt{choose}\ p_1 : S_1\ \texttt{or}\ p_2 : S_2\ \texttt{ro}, s\rangle \longrightarrow_{\tilde{p}_1} \langle S_1, s\rangle \quad \text{with } \tilde{p}_1 = p_{1[p_1,p_2]}$$

$$\mathbf{R7}_2\ \langle \texttt{choose}\ p_1 : S_1\ \texttt{or}\ p_2 : S_2\ \texttt{ro}, s\rangle \longrightarrow_{\tilde{p}_2} \langle S_2, s\rangle \quad \text{with } \tilde{p}_2 = p_{2[p_1,p_2]}$$

by identifying each statement S with the label of the initial block of S, i.e. a configuration $\langle S, s\rangle$ is identified with the pair $\langle s, init(S)\rangle \in \mathbf{State} \times \mathbf{Label}$ (for a formal definition of *init* see e.g. [5]). Most transitions are in fact deterministic (i.e. the associated probability is 1) just for choices, i.e. rules **R7** do we use the normalised probabilities $\tilde{p}_i$ (more on the actual normalisation procedure below).

The probabilistic transition system defined in Table 2 describes a Discrete Time Markov Chain (DTMC) (cf. e.g. [15,18]) as we obviously have a memoryless process: the transitions in Rules **R1** to **R7** depend only on the current configuration and not on the sequence of the configurations that preceded it. One can also easy to show that the probabilities of out-going transitions from each state sum up to one. It is well-known that the matrix of transition probabilities of a DTMC on a countable state space is a stochastic matrix, i.e. a square (possibly infinite) matrix $\mathbf{P} = (p_{ij})$ whose elements are real numbers in the closed interval $[0, 1]$, for which $\sum_j p_{ij} = 1$ for all i [18,20]. We can therefore represent the SOS semantics for a **pWhile** program P by the stochastic matrix on the vector space over the set **Conf** of all configurations of a program P defined by the rules in Table 2.

2.3 States and Observables

For our language we also allow for the specification of the range of possible values of variables, i.e. $\mathbf{Value}(x)$, via *declarations*. Without going into the details of the formal syntax, we distinguish between parameters, indicated by `para`, and proper variables for which we specify their **Value** as a subset of the integers.

This allows us (also because $\mathbf{Value}(x)$ are assumed to be finite) to describe the space of probabilistic *states* σ (of a program) as (probability) distributions over classical states, i.e. $\sigma \in \mathcal{D}(\mathbf{State})$. We can also see σ simply as a vector in the so-called free vector space $\mathcal{V}(\mathbf{State})$ over **State** (distributions correspond to positive vectors with 1-norm 1) cf. [5,7].

For a single variable x we have (the isomorphism) $\mathbf{State} = \mathbf{Value}(x)$ and when we consider several variables we can identify a classical state s with an

element in the Cartesian product $\mathbf{Value}(x_i) \times \ldots \times \mathbf{Value}(x_v)$. When we consider probabilistic states of a single variable x then we have $\sigma \in \mathcal{D}(\mathbf{State}) \subseteq \mathcal{V}(\mathbf{Value}(x))$. But for more than one variable we have $\sigma \in \bigotimes_{i=1}^{v} \mathcal{V}(\mathbf{Value}(x_i))$, i.e. the so-called *tensor product*, rather than the Cartesian product of $\mathcal{V}(\mathbf{Value})$. This unfortunately leads to a form of combinatorial explosion but is needed accommodate all possible *joint probability distributions* as we have (the isomorphism) $\mathcal{V}(X_1 \times \ldots \times X_v) = \mathcal{V}(X_1) \otimes \ldots \otimes \mathcal{V}(X_v)$.

Concretely, the tensor product – more precisely, the Kronecker product, i.e. the coordinate based version of the abstract concept of a tensor product – of two vectors $(x_1, \ldots, x_n)$ and $(y_1, \ldots, y_m)$ is $(x_1y_1, \ldots, x_1y_m, \ldots, x_ny_1, \ldots, x_ny_m)$ i.e. an nm dimensional vector. For an $n \times m$ matrix $\mathbf{A} = (\mathbf{A}_{ij})$ and an $n' \times m'$ matrix $\mathbf{B} = (\mathbf{B}_{kl})$ we construct similarly an $nn' \times mm'$ matrix $\mathbf{A} \otimes \mathbf{B} = (\mathbf{A}_{ij}\mathbf{B})$, i.e. each entry $\mathbf{A}_{ij}$ in $\mathbf{A}$ is multiplied with a copy of the matrix or block $\mathbf{B}$, for further details we refer e.g. to [16, Chapter 14].

In the following we also will use the notion of an *observable* which describes properties a program or system might have (for further details see [7]). Formally, an observable is a linear functional on the probabilistic state space, i.e. an element of its dual space. For finite dimensional spaces, as we have them here, we can identify state and observable space. States and observables are related to each other by the notion of expected value, $\mathbf{E}(x, \sigma)$, which gives the probability that we will observe a certain property x when the state of the system is described by σ. In our finite setting (and by Riesz's representation theorem) we can utilise an inner product $\langle ., . \rangle$ in order to to obtain $\mathbf{E}(x, \sigma) = \langle x, \sigma \rangle$.

3 Static Probabilities

If the probabilities in the `choose` statement are required to be constants (or parameters) then we can us a simple (re)normalisation procedure (at compile time) in order to obtain the effective probabilities that a certain alternative is executed, i.e. we (re) normalise probabilities in the SOS in Table 2 via:

$$\tilde{p} = p_{[p_1 \ldots p_n]} = \frac{p}{p_1 + \ldots + p_n}.$$

Not least because we will allow later also variable values p_i we have to address the issue whether $p_{[p_1 \ldots p_n]}$ is always well-defined. We will exclude negative weights (if the nevertheless appear we could consider the absolute values). However, one problem remains, namely whether or not we allow for $p_i = 0$. One argument – which we will adopt – would be to allow this to indicate "blocked" alternatives, especially when we consider (below) dynamical probabilities. This implies another issue we need to consider, namely the case where all $p_i = 0$. In this case, normalisation would imply a division by zero. To overcome this we set $\tilde{p} = p_{[p_1 \ldots p_n]} = 0$ if we have for all $p_i = 0$.

3.1 Linear Operator Semantics (LOS)

The Linear Operator Semantics (LOS) in [4,7] constructs the generator of the DTMC which represents the dymanics of a program (executions) in a syntax

directed fashion. Like Kozen's semantics [11] we can represent the LOS as an operator on the vector space of probabilistic states, i.e. in the finite case as a matrix.

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 a linear operator on $\mathcal{V}(\mathbf{Conf})$

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

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. $\{\!\{.\}\!\}_{LOS} : \mathbf{Stmt} \to \mathcal{P}(\Gamma \cup \Lambda)$, cf Table 3. 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 operators as $\{\langle \mathbf{F}_i, \ell_i \rangle\} \rhd \ell' = \{\mathbf{F}_i \otimes \mathbf{E}(\ell_i, \ell')\}$ (for global operators we have $\mathbf{G} \rhd \ell' = \mathbf{G}$). We denote by $\mathbf{E}(i,j)$ matrix units: $(\mathbf{E}(i,j))_{ij} = 1$ and 0 otherwise.

Table 3. The LOS semantics of **pWhile** (static)

$$\begin{aligned}
\{\!\{[\texttt{skip}]^\ell\}\!\}_{LOS} &= \{\langle \mathbf{I}, \ell \rangle\} \\
\{\!\{[x \texttt{ := } e]^\ell\}\!\}_{LOS} &= \{\langle \mathbf{U}(x \leftarrow e), \ell \rangle\} \\
\{\!\{S_1;\ S_2\}\!\}_{LOS} &= ([\![S_1]\!] \rhd \mathit{init}(S2)) \cup [\![S_2]\!] \\
\{\!\{[\texttt{choose}]^\ell\ p_1 : S_1 \texttt{ or } p_2 : S_2 \texttt{ ro}\}\!\}_{LOS} &= \{p_{1[p_1,p_2]} \cdot \mathbf{I} \otimes \mathbf{E}(\ell, \mathit{init}(S_1))\} \cup \{\!\{S_1\}\!\}_{LOS} \cup \\
&\quad \{p_{2[p_1,p_2]} \cdot \mathbf{I} \otimes \mathbf{E}(\ell, \mathit{init}(S_2))\} \cup \{\!\{S_2\}\!\}_{LOS} \\
\{\!\{\texttt{if } [b]^\ell \texttt{ then } S_1 \texttt{ else } S_2 \texttt{ fi}\}\!\}_{LOS} &= \{\langle \mathbf{P}(b), \ell \rangle\} \rhd \mathit{init}(S_1)\} \cup \{\!\{S_1\}\!\}_{LOS} \cup \\
&\quad \{\langle \mathbf{P}(b)^\perp, \ell \rangle\} \rhd \mathit{init}(S_2)\} \cup \{\!\{S_2\}\!\}_{LOS} \\
\{\!\{\texttt{while } [b]^\ell \texttt{ do } S \texttt{ od}\}\!\}_{LOS} &= \{\langle \mathbf{P}(b), \ell \rangle\} \rhd \mathit{init}(S)\} \cup \{\!\{S\}\!\}_{LOS} \rhd \ell \\
&\quad \cup \{\langle \mathbf{P}(b)^\perp, \ell \rangle\}
\end{aligned}$$

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 $\mathcal{V}(\mathbf{Value})^{\otimes n}$:

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

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:

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

As we model the semantics of a program as DTMCs 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.

As said, the operator $[\![P]\!]_{LOS}$ is the generator of a DTMC which implements the dynamic behaviour or executions of the program P. In particular, we can take any (initial) configuration c_0, represented by a (point) distribution in $\mathcal{V}(\mathbf{Conf})$ and compute the distribution over all configurations we will have after n steps as $c_n = c_0 \cdot [\![P]\!]^n_{LOS}$ (using post-multiplication as our convention).

3.2 A Small Example

The LOS semantics specifies the semantics of a program as the generator of a DTMC. We use a simple experimental tool – `pwc` – which "compiles" a **pWhile** program into an `octave` [8] script which defines the different matrices/operators. To illustrate this let us look at a simple example involving a probabilistic choice.

Example 1. The concrete program P we consider, for which we also provide the labelling (which is in fact produced by the `pwc` tool) is given by:

```
var
  p :para; x :{0,1};
begin
  [choose]^1 1: [x:=0]^2 or 1: [x:=1]^3 or p: [skip]^4 ro;
  [stop]^5
end
```

Here we deal with one parameter p, the value of this can be set to any (integer) value before the program is actually executed, and one variable x which can take two values in $\{0,1\}$. The state space is thus given just by $\mathcal{V}(\{0,1\}) = \mathbb{R}^2$ (as the parameter p does not change we do not record its value as part of the state). The program is made up from 5 blocks: $[\texttt{choose}]^1, [x := 0]^2, [x := 1]^3, [\texttt{skip}]^4, [\texttt{skip}]^5$. We thus have as the (probabilistic) space of configurations on which the LOS operator acts $\mathcal{V}(\{0,1\} \times \mathcal{V}(\{\ell_1, \ell_2, \ell_3, \ell_4, \ell_5\}) = \mathbb{R}^2 \otimes \mathbb{R}^5 = \mathbb{R}^{10}$, i.e. $[\![P]\!]_{LOS}$ is a 10×10 matrix which represents the generator of a DTMC on a space of 10 elements. Each dimension corresponds to a possible configuration, i.e. a tuple $\langle s_i, \ell_j \rangle$ with s a (classical) state $s : \{x\} \to \{0,1\}$ and a statement or block identified by its label $\ell \in \{\ell_1, \ell_2, \ell_3, \ell_4, \ell_5\}$. Concretely we have the following base vectors e_i in $\mathbb{R}^{10}$ for the state spaces of the DTMC: $e_1 = \langle x \mapsto 0, \ell_1 \rangle, e_2 = \langle x \mapsto 0, \ell_2 \rangle, \ldots, e_5 = \langle x \mapsto 0, \ell_5 \rangle, e_6 = \langle x \mapsto 1, \ell_1 \rangle, \ldots, e_{10} = \langle x \mapsto 1, \ell_5 \rangle$.

For each of the 5 blocks we have a local transfer operator $\mathbf{F}_1, \ldots \mathbf{F}_5$ which are (stochastic) 2×2 matrices, i.e. linear operators on our state space $\mathbb{R}^2$. For blocks 4 and 5 these $\mathbf{F}_i$ are trivial, i.e. the identity 2×2 matrix, for label ℓ_2 and ℓ_3 the transfer operators are slightly more interesting:

$$\mathbf{F}_1 = \mathbf{F}_4 = \mathbf{F}_5 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad \mathbf{F}_2 = \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix}, \quad \mathbf{F}_3 = \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix}.$$

This allows us to specify the local LOS operators for each basic block:

$$\{\!\{[x := 0]^2\}\!\}_{LOS} = \{\langle \mathbf{F}_1, 2 \rangle\}, \{\!\{[x := 1]^3\}\!\}_{LOS} = \{\langle \mathbf{F}_1, 3 \rangle\},$$
$$\{\!\{[\texttt{skip}]^4\}\!\}_{LOS} = \{\langle \mathbf{F}_4, 4 \rangle\}, \{\!\{[\texttt{skip}]^5\}\!\}_{LOS} = \{\langle \mathbf{F}_5, 5 \rangle\}.$$

We could also consider explicitely $\{\!\{[\texttt{choose}]^1\}\!\}_{LOS} = \{\langle \mathbf{F}_1, 1 \rangle\}$, however this will be covered when we consider the global operators.

The control flow of P is made up from 7 control-flow step triples $\langle i, p, j \rangle$, where i is the initial label, p the transition probability and j the final label:

$$1 - \langle 1,1,2 \rangle, 2 - \langle 1,1,3 \rangle, 3 - \langle 1,p,4 \rangle,$$
$$4 - \langle 2,1,5 \rangle, 5 - \langle 3,1,5 \rangle, 6 - \langle 4,1,5 \rangle, 7 - \langle 5,1,5 \rangle.$$

For each of these control-flow steps we construct a global operator, typically the tensor product of the local transfer operator $\mathbf{F}_i$ at the initial label i and a control-flow step given by the matrix unit $\mathbf{E}(i,j)$, eventually weighted by a probability. Here we have to consider the (global) operators: $\mathbf{T}_1 = \mathbf{F}_1 \otimes \mathbf{E}(1,2), \mathbf{T}_2 = \mathbf{F}_1 \otimes \mathbf{E}(1,3), \mathbf{T}_3 = \mathbf{F}_1 \otimes \mathbf{E}(1,4), \mathbf{T}_4 = \mathbf{F}_2 \otimes \mathbf{E}(2,5), \mathbf{T}_5 = \mathbf{F}_3 \otimes \mathbf{E}(3,5), \mathbf{T}_5 = \mathbf{F}_4 \otimes \mathbf{E}(4,5), \mathbf{T}_7 = \mathbf{F}_5 \otimes \mathbf{E}(5,5)$. The first three operators allow us to define the LOS of the choices statement. For this we have to specify a particular value for the parameter p. For example, for $p = 0$ we get after renormalisation:

$$\{\!\{[\texttt{choose}]^1 \ldots \texttt{ro}\}\!\}_{LOS} = \{\frac{1}{2}\mathbf{T}_1, \frac{1}{2}\mathbf{T}_2\} \cup \{\!\{[x := 0]^2\}\!\}_{LOS} \cup \{\!\{[x := 1]^3\}\!\}_{LOS}.$$

If we instead take $p = 1$ we get after renormalisation:

$$= \{\frac{1}{3}\mathbf{T}_1, \frac{1}{3}\mathbf{T}_2, \frac{1}{3}\mathbf{T}_3, \} \cup \{\!\{[x := 0]^2\}\!\}_{LOS} \cup \{\!\{[x := 1]^3\}\!\}_{LOS} \cup \{\!\{[\texttt{skip}]^4\}\!\}_{LOS}.$$

The LOS $\{\!\{[\texttt{choose}]^1 \ldots \texttt{ro}\}\!\}_{LOS}$ contains global as well as local operators: The global ones represent control-flow steps where the destination is already known, while the local ones (here for the labels ℓ_2, ℓ_3 and ℓ_4 are still unresolved. However, when we consider the whole program then the operation $\triangleright$ resolves the destinations of local operators and turns them into global ones, e.g.

$$\begin{array}{l} \{\!\{[x := 0]^2\}\!\}_{LOS} \triangleright \ell_5 = \{\mathbf{T}_4\} = \{\mathbf{F}_2 \otimes \mathbf{E}(2,5)\} \\ \{\!\{[x := 1]^3\}\!\}_{LOS} \triangleright \ell_5 = \{\mathbf{T}_5\} = \{\mathbf{F}_3 \otimes \mathbf{E}(3,5)\} \\ \{\!\{[\texttt{skip}]^4\}\!\}_{LOS} \triangleright \ell_5 \quad = \{\mathbf{T}_6\} = \{\mathbf{F}_4 \otimes \mathbf{E}(4,5)\} \end{array}$$

Resolving the self-loop for label 5 using $\mathbf{T}_7$ we get the semantics for $p = 0$ as:

$$\{\!\{P\}\!\}_{LOS} = \{\frac{1}{2}\mathbf{T}_1, \frac{1}{2}\mathbf{T}_2, \mathbf{T}_4, \mathbf{T}_5, \mathbf{T}_6, \mathbf{T}_7\}$$

and for $p = 1$ we have (similarly also for other values of p):

$$\{\!\{P\}\!\}_{LOS} = \{\frac{1}{3}\mathbf{T}_1, \frac{1}{3}\mathbf{T}_2, \frac{1}{3}\mathbf{T}_3, \mathbf{T}_4, \mathbf{T}_5, \mathbf{T}_6, \mathbf{T}_7\}$$

The DTMC generator in both case is $[\![P]\!]_{LOS} = \sum \{\mathbf{T} \mid \mathbf{T} \in \{\!\{P\}\!\}_{LOS}\}$.

4 Dynamical Probabilities

The main purpose of this work is to allow for "dynamical" probabilities in programs. That is we would like to allow for variables in choice constructs which allow a change of their values in the course of a computation. Given that our LOS semantics constructs a single operator $[\![P]\!]_{LOS}$ for every program P which does not change during the execution, i.e. represents a (time) homogenous DTMC, this seems to be a hopeless task. On the other hand, the state of the system does obviously contain all the information which could influence how the execution of a program should continue, so if it encodes the values of variables in choices, then this information should somehow be exploitable.

For the SOS semantics it is still relatively easy to extend it towards variable probabilities: We have to replace the normalisation condition in rules **R7** in Table 2 by reference to the current state s, i.e. $\tilde{p}_i = s(p_i)/s(p_1)+s(p_2)$ rather than constant values of p_i. The way to introduce dynamical or variable probabilities into the LOS semantics of the choice construct is to test or check whether we are in a certain state where variables have certain concrete values, if this is the case then the corresponding normalisation is applied.

4.1 Linear Operator Semantics (LOS)

In order to extend the LOS semantics as to allow for variable probabilities we have to consider the way we construct the LOS operator for the choice statement with static, i.e. constant, probabilities: $\{\!\{[\texttt{choose}]^\ell\ p_1 : S_1\ \texttt{or}\ p_2 : S_2\ \texttt{ro}\}\!\}_{LOS} =$

$\{\tilde{p}_1 \cdot \mathbf{I} \otimes \mathbf{E}(\ell, init(S_1))\} \cup \{\!|S_1|\!\}_{LOS} \cup \{\tilde{p}_2 \cdot \mathbf{I} \otimes \mathbf{E}(\ell, init(S_2))\} \cup \{\!|S_2|\!\}_{LOS}$, or more general for n alternatives in a choice statement:

$$\{\!|[\texttt{choose}]^\ell\ p_1 : S_1\ \texttt{or}\ \ldots \texttt{or}\ p_n : S_n\ \texttt{ro}|\!\}_{LOS} = \\ = \bigcup_{i=1}^{n} \{\tilde{p}_i \cdot \mathbf{I} \otimes \mathbf{E}(\ell, init(S_i))\} \cup \{\!|S_i|\!\}_{LOS}.$$

In these rules all p_i are known, either because they are constants or because they are constant parameters. We thus can compute the normalised probabilities $\tilde{p}_i$ or, when we need to explicitely record the context in which we normalise, $\tilde{p}_i = p_{i[p_1 \ldots p_n]}$ in exactly the same way as in the operational semantics.

When it comes to dynamical probabilities then we need to consider all possible contexts, i.e. all possible values $p_1, \ldots, p_n$ could take, in which we might need to normalise a probability. Formally we define a context for probabilities $p_1, \ldots, p_n$ where each p_i can be a constant value (incl. a parameter) or a variable (name) as a set of sequences $i_1, \ldots, i_n$ of integers:

$$\mathcal{C}[p_1, p_2, \ldots, p_n] = \begin{cases} \emptyset & \text{if n = 0} \\ \{[p_1]\} & \text{if } n = 1 \text{ and } p_i \text{ constant} \\ \{[c] \mid c \in \mathbf{Value}(p_1)\} & \text{if } n = 1 \text{ and } p_i \text{ a variable} \\ \bigcup_{[i] \in \mathcal{C}[p_1]} \{[i] \cdot \mathcal{C}[p_2, \ldots, p_n]\} & \text{otherwise, i.e. } n > 1. \end{cases}$$

where "$\cdot$" denotes the concatenation of integer sequences $[i_1, \ldots, i_m]$ defined and extended to sets of sequences in the obvious way.

Example 2. Assume we have a variable x with $\mathbf{Value}(x) = \{0, 1\}$ and a parameter $p = 0$ or $p = 1$ then contexts are given by:

$$\mathcal{C}[x, 1, p] = \{[0, 1, 0], [1, 1, 0]\} \quad \text{and} \quad \mathcal{C}[x, 1, p] = \{[0, 1, 1], [1, 1, 1]\}$$

With this we can now define an extended version of the LOS which also allows for variables as choice probabilities:

$$\{\!|[\texttt{choose}]^\ell\ p_1 : S_1\ \texttt{or}\ \ldots \texttt{or}\ p_n : S_n\ \texttt{ro}|\!\}_{LOS} = \\ = \bigcup_{i=1}^{n} \left\{ \sum_{c_j \in p_i} \sum_{[d_1, \ldots, d_n] \in \mathcal{C}[p_1 \ldots p_n]} c_{j\,[d_1 \ldots d_n]} \cdot \mathbf{P}^{p_i[p_1 \ldots p_n]}_{c_j[d_1 \ldots d_n]} \otimes \mathbf{E}(\ell, init(S_i)) \right\} \cup \{\!|S_i|\!\}_{LOS}.$$

To explain this construction: The LOS of the choices is given – as in the static case – as the union of all (global) operators which implement the control-flow step from label ℓ to one of the alternatives $i = 1 \ldots n$ together with the LOS semantics of each of these alternatives defined by $\{\!|S_i|\!\}_{LOS}$. However, in the case of static probabilities we have to weight the operator $\mathbf{E}(\ell, init(S_i))$ not just with a normalised probability but instead we test if the values of the probabilities (which can be variables, after all) are described by a particular context and then apply the corresponding normalised weight $c_{j\,[d_1 \ldots d_n]}$. This test operator $\mathbf{P}^{p_i[p_1 \ldots p_n]}_{c_j[d_1 \ldots d_n]}$ is very similar to the test we apply in order to identify a particular

state, i.e. $\mathbf{P}(\sigma)$, except that in a context the same variable can appear several times:

$$\mathbf{P}^{p_i[p_1 \ldots p_n]}_{c_j[d_1 \ldots d_n]} = \mathbf{P}(p_i = c_j) \cdot \left(\prod_{k=1,\ldots,n} \mathbf{P}(p_k = d_k) \right).$$

The first sum is over all possible values of the guard probability p_i, where we use the short-hand notation $c_j \in p_i$ for $c_j \in \mathbf{Value}(p_i)$ which for constants and parameters reduces to a single term $c_j = p_i$. The second sum is over all possible values of all probabilities in all possible contexts. It might be interesting to note that if a variable appears twice it has to have the same value (as $\mathrm{diag}(e_i) \otimes \mathrm{diag}(e_j) = \mathrm{diag}(e_i)$ if and only if $i = j$ and the zero matrix otherwise). For constant values we can also omit the tests (as $e_i\mathbf{T} = e_i\mathrm{diag}(e_i)\mathbf{T}$ for all $\mathbf{T}$).

It is simple to show that the LOS semantics for choice with variable probabilities is equivalent to the SOS semantics, for the other construct things are unchanged [7].

4.2 A Small Example

In order to illustrate the LOS for dynamical variables let us again first consider a very simple example, similar to Example 1.

Example 3. The program Q we consider is given by:

```
var
  p :para; x :{0,1};
begin
  [choose]^1 x: [x:=0]^2 or 1:[x:=1]^3 or p:[skip]^4 ro;
  [stop]^5
end
```

As we have the same declarations, we have exactly the same state spaces as in Example 1. Furthermore, we also have the same 5 blocks as in the previous example and therefore the DTMC state space of configurations is again $\mathbb{R}^{10}$. We also have the same transfer operators $\mathbf{F}_i$ (and local LOS operators for the basic blocks). However, though the control flow has again 7 control-flow steps and it is nearly identical, except for the step from ℓ_1 to ℓ_2 which here is guarded by a variable probability x:

$$1 - \langle 1, x, 2\rangle, 2 - \langle 1, 1, 3\rangle, 3 - \langle 1, p, 4\rangle,$$
$$4 - \langle 2, 1, 5\rangle, 5 - \langle 3, 1, 5\rangle, 6 - \langle 4, 1, 5\rangle, 7 - \langle 5, 1, 5\rangle.$$

We can still use the same operators $\mathbf{T}_i$ from Example 1 but the complete LOS semantics now looks slightly different. For $p = 0$ or $p = 1$ we need to work with the contexts given in Example 2. For $p = 0$ we have $\mathcal{C}[x, 1, p] = \{[0, 1, 0], [1, 1, 0]\}$ and thus get

$$\begin{aligned} \{\!\{Q\}\!\}_{LOS} = \{ &(\mathbf{P}(x = 0) \; + \; \tfrac{1}{2}\mathbf{P}(x = 1)) \otimes \mathbf{E}(1, 3), \\ &(\qquad\qquad\; \tfrac{1}{2}\mathbf{P}(x = 1)) \otimes \mathbf{E}(1, 4), \mathbf{T}_4, \mathbf{T}_5, \mathbf{T}_6, \mathbf{T}_7\}. \end{aligned}$$

and for the parameter value $p = 1$ we have $\mathcal{C}[x, 1, p] = \{[0, 1, 1], [1, 1, 1]\}$ and:

$$\begin{aligned}\{\!\{Q\}\!\}_{LOS} = \{\,(& \tfrac{1}{3}\mathbf{P}(x = 1)) \otimes \mathbf{E}(1, 2),\\ (\tfrac{1}{2}\mathbf{P}(x = 0) \;+\; & \tfrac{1}{3}\mathbf{P}(x = 1)) \otimes \mathbf{E}(1, 3),\\ (\tfrac{1}{2}\mathbf{P}(x = 0) \;+\; & \tfrac{1}{3}\mathbf{P}(x = 1)) \otimes \mathbf{E}(1, 4), \mathbf{T}_4, \mathbf{T}_5, \mathbf{T}_6, \mathbf{T}_7\}.\end{aligned}$$

Note that test operators like $\mathbf{P}(x = 1)$ should actually be expressed as, for example: $\mathbf{P}(x = 1)\mathbf{P}(x = 1)\mathbf{P}(1 = 1)\mathbf{P}(p = 1)$. However, as said before, in the case of constants (and parameters) these tests are redundant and as projections are always idempotents we also have: $\mathbf{P}(x = 1) = \mathbf{P}(x = 1)\mathbf{P}(x = 1)$.

5 Example: Duel at High Noon

We illustrate the generation of the LOS semantics – i.e. the DTMC generator of a probabilistic program – by considering an example given in [9,10], see also [12, p. 211], which concerns the kind of "duel" between two "cowboys" A and B. We first reproduce essentially the results of [9,10] regarding the chances that A (or B) will win/survive the "duel" with static probabilities. We then also consider the case where one the two duellists (here A) improves his hitting chances during the contest. This situation obviously requires dynamical/changing probabilities.

5.1 Static Probabilities

The idea is that two "cowboys", A (Adam) and B (Boris), have a duel. At each turn one of them is allowed to shoot at the other, if he misses the other one can try, if he also misses it is the first ones turn again until one is "successful". That is, at the beginning one of the two – either Adam or Boris – is allowed to shoot at the other one. Which of the two starts is left open, i.e. decided non-deterministically. We assume that there is a probability a for A hitting B and a probability b that B manages to shoot A. More precisely, we have $a = \frac{ak}{ak+am}$ for a "killing" and a "missing" weight ak and bk, respectively (and similar for b). In the original version it is non-deterministically decided whether A or B starts, but in order to get simple numerical results we will flip a fair coin to determine who has the first attempt. The concrete **pWhile** program is given on the left hand side in Table 4.

The variable c determines whether the duel should be continued, if $c = 1$ the duel continues, otherwise it is over. This is essentially to simulate a `until` statement using the `while` construct. The variable t determines which of the two duellists is allowed to try to shoot, for $t = 0$ it is A's turn, otherwise it is B's turn. As long as the duel is continued (i.e. $c = 1$) it is either A which gets a try (if $t = 0$) or B (for $t = 1$). If it is A's turn he will hit B with probability a – in this case the duel is over and c is set to 0; and with probability $1 - a$ it might be a miss – in this case the next round it will be B's turn. Similarly, for $t = 1$ the duellist B gets his chance.

At the end of the duel the value of t determines who has lost/won – i.e. whose turn it was when the loop terminated, i.e. c was set to zero. In order to extract

Table 4. pWhile programs for the Duel at High Noon

```
var
ak: para; # A kills
am: para; # A misses
bk: para; # B kills
bm: para; # B misses
t: {0,1}; # turn     0=A,  1=B
c: {0,1}; # continue 0=no, 1=yes

begin
# who's first turn
choose 1:{t:=0} or 1:{t:=1} ro;
# continue until ...
c := 1;
while c == 1 do
if (t==0) then
  choose ak: c:=0] or am: t:=1 ro
else
  choose bk: c:=0 or bm: t:=0 ro
fi;
od;
stop; # terminal loop
end
```

```
var
ak: para; # A kills  (initially)
am: para; # A misses (initially)
bk: para; # B kills
bm: para; # B misses
t: {0,1}; # turn     0=A,  1=B
c: {0,1}; # continue 0=no, 1=yes
akl: {0..10}; # A kills  (learned)
aml: {0..10}; # A misses (learned)

begin
# initialise skills of A
akl := ak; aml := am;
# who's first
choose 1:{t:=0} or 1:{t:=1} ro;
# continue until ...
c := 1;
while c == 1 do
  if (t==0) then
    choose akl: c:=0 or aml: t:=1 ro
  else
    choose bk: c:=0 or bm: t:=0 ro
  fi;
  akl:=@inc(akl); aml:=@dec(aml);
od;
stop; # terminal loop
end
```

information about the probability distribution describing a particular variable – in our case t – at a given label ℓ, i.e. program point ℓ, we can use an abstraction operator $\mathbf{A}_\ell$. This operator/matrix leaves the first variable (i.e. t) unchanged and "forgets" about all other variables in a particular label ℓ:

$$\mathbf{A}_\ell = \mathbf{I} \otimes \mathbf{A}_f \otimes \ldots \otimes \mathbf{A}_f \otimes (e_\ell)^t$$

with $\mathbf{I}$ the identity matrix for the first variable (for t it is a 2×2 matrix), $\mathbf{A}_f$ a so-called "forgetfull abstraction" for the remaining variables and e_ℓ^t the transposed (column) base vector in $\mathcal{V}(\mathbf{Label})$ which selects or projects the state at label ℓ. The operators $\mathbf{A}_f$ are given by column vectors (or $n \times 1$ matrices) which only contain 1s, i.e. $\mathbf{A}_f = (1, 1, 1, \ldots, 1)^t$ with $n = \dim(\mathcal{V}(\mathbf{Value}(x)) = |\mathbf{Value}(x)|$. This is an instance of a more general framework of Probabilistic Abstract Interpretation (PAI), cf. e.g. [5–7].

With this abstraction $\mathbf{A}_\ell$ we can extract the probabilities that t is 0 or 1, i.e. who has won the duel, if we take $\ell = \ell^*$, i.e. the final label ℓ^8 of the program once the program has "terminated". For this we have to consider the (long-run) input/output behaviour for an initial configuration $c_0 = s_0 \otimes e_0$, i.e. an initial state s_0 which determines the initial values of all variables at the initial label

ℓ_0. We then have to apply the LOS operator $[\![P]\!]_{LOS}$ until we reach a limit $\lim_{n\to\infty}(s_0 \otimes e_0)[\![P]\!]^n_{LOS}$. This essentially gives Kozen's input/output semantics [11] of the program, cf. [7].

To obtain numerical results we can stop this iteration for a finite value of n, in our case $n = 100$ is sufficient. Finally, we have to extract the state of t using $\mathbf{A}_{\ell^*}$ and the observable $w = (1, 0)$ which gives the probability that $t = 0$, i.e. that the winner is A. In other words, the aim of the analysis is to determine:

$$\omega = \lim_{n\to\infty} \langle w, (s_0 \otimes e_0) \cdot [\![P]\!]^n_{LOS} \cdot \mathbf{A}_{\ell^*} \rangle \,.$$

or a numerical approximation (for $n = 100$). In our case t and c are both initialised, so ω is idependent of the initial state s_0. If we consider the "non-deterministic" version, i.e. dropping '`choose 1: t:=0 or 1: t:=1 ro`', the value of ω would depend on s_0.

We use our tool `pwc` to construct $[\![P]\!]_{LOS}$. The program has 13 labels or elementary blocks (with $\ell^* = 13$). The dimension of the DTMC is then $2 \times 2 \times 13 = 52$ as t and c take two possible values. With this we can compute ω for different values of the parameters ak, am, etc. The top left diagram in Fig. 1 depicts the chances of A surviving the duel depending on $a = ak/(ak + am)$ and $b = bk/(bk + bm)$.

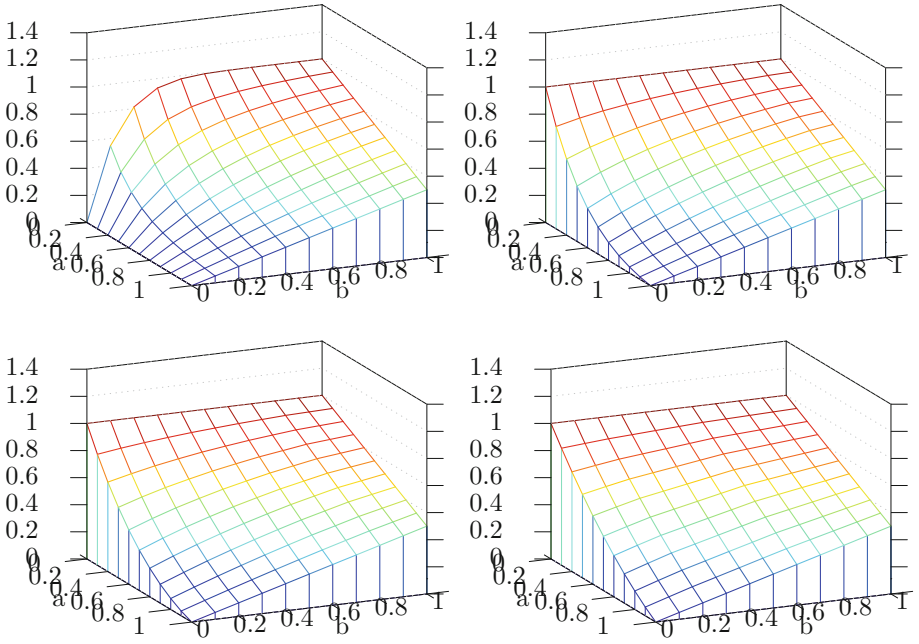

Fig. 1. Survival probabilities ω for A with learning rates 0, 1, 2 and 4

5.2 Dynamic Probabilities

If we assume that probabilities (of hitting) are not constant, but that for example one of the duellists is getting better during the shoot-out we have to consider a different model as in the following **pWhile** program as on the right in Table 4.

Here we use the same parameters ak, etc. as in the static case. However for A these are only the initial values. During the duel A will improve his shooting skills (while B's abilities do not change). The (learned) chances of A hitting is given by akl and the chances of missing aml. These are changed using "external" functions `@inc` and `@dec` which depend on a learning rate r defined directly in `octave` as `min(max(x+r,0),10)` and `min(max(x-r,0),10)`, respectively.

For different values of the parameters ak, am, etc. we can again construct the LOS operator $[\![P]\!]_{LOS}$. In this case we have 17 lables/blocks and two additional variables akl and akm which each can have 11 possible values, thus we have to consider a DTMC on $2 \times 2 \times 11 \times 11 \times 17 = 8228$ states.

The survival chances for A can be computed in the same way as in the static case, using the corresponding abstraction $\mathbf{A}_{17}$, the same w and based on a numeric approximation based on $n = 100$ iterations of $[\![P]\!]_{LOS}$. For different learning rates r we depict the survival rate for A in Fig. 1. For $r = 0$ we get exactly the same as in the static case – after all, A is stuck with his initial shooting abilities and does not improve at all.

6 Conclusions

We presented a model for probabilistic programs which essentially encodes the semantics of a program in terms of time homogenous DTMCs, i.e. the operator representing the semantics is given by a time invariant, "eternal" stochastic operator/matrix. Nevertheless, within this static model it is possible to also realise changing probabilities.

The language we based this on is a simple procedural language. Nevertheless, it is obvious that this model also applies to (proper) coordination languages like pKLAIM [2,3]. This concerns in particular concurrency aspects: The rules of the duel in the cowboy example essentially implement an explicit round robin scheduler and the extension to more general schedulers seems not to be difficult. Surviving the duel itself can also be seen as an ultimate coordination problem in which the role of probability normalisation is essential: Ones survival depends not only on ones own (shooting) abilities but also on the one of the opponent. A hit rate of 50 % for A means almost sure survival for A if B is a bad shooter with a 2 % hit rate, but if B a perfect duelist with 100 % hit rate then this will give the same A no chance of survival if B begins the duel.

It seems also feasible to extend this "probability testing" approach to continuous time models, continuous probabilities and hybrid systems, although this will require more careful considerations of the underlying measure theoretic structure (Borel structure, σ-algebras, measures instead of distributions, integrals in place of sums, etc.).

References

1. Aarts, E., Korst, J.: Simulated Annealing and Boltzmann Machines. Wiley, Chichester (1989)
2. Di Pierro, A., Hankin, C., Wiklicky, H.: Continuous-time probabilistic KLAIM. In: SecCo 2004, ENTCS. Elsevier (2004)
3. Di Pierro, A., Hankin, C., Wiklicky, H.: Probabilistic KLAIM. In: De Nicola, R., Ferrari, G.-L., Meredith, G. (eds.) COORDINATION 2004. LNCS, vol. 2949, pp. 119–134. Springer, Heidelberg (2004)
4. Di Pierro, A., Hankin, C., Wiklicky, H.: A systematic approach to probabilistic pointer analysis. In: Shao, Z. (ed.) APLAS 2007. LNCS, vol. 4807, pp. 335–350. Springer, Heidelberg (2007)
5. Di Pierro, A., Hankin, C., Wiklicky, H.: Probabilistic semantics and program analysis. In: Aldini, A., Bernardo, M., Pierro, A., Wiklicky, H. (eds.) SFM 2010. LNCS, vol. 6154, pp. 1–42. Springer, Heidelberg (2010)
6. Di Pierro, A., Sotin, P., Wiklicky, H.: Relational analysis and precision via probabilistic abstract interpretation. In: Proceedings of QAPL 2008. ENTCS, vol. 220, no. 3, pp. 23–42. Elsevier (2008)
7. Di Pierro, A., Wiklicky, H.: Semantics of probabilistic programs: a weak limit approach. In: Shan, C. (ed.) APLAS 2013. LNCS, vol. 8301, pp. 241–256. Springer, Heidelberg (2013)
8. Eaton, J.W., Bateman, D., Hauberg, S.: GNU Octave - a high-level interactive language for numerical computations, 3rd edn. version 3.8.0, February 2011
9. Gretz, F., Katoen, J.P., McIver, A.: Operational versus weakest precondition semantics for the probabilistic guarded command language. In: Proceedings of QEST 2012, pp. 168–177. IEEE Computer Society (2012)
10. Gretz, F., Katoen, J.P., McIver, A.: Operational versus weakest pre-expectation semantics for the probabilistic guarded command language. Perform. Eval. **73**, 110–132 (2014)
11. Kozen, D.: Semantics of probabilistic programs. J. Comput. Syst. Sci. **22**(3), 328–350 (1981)
12. McIver, A., Morgan, C.: Abstraction, Refinement and Proof for Probabilistic Systems. Mongraphs in Computer Science. Springer, New York (2005)
13. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press, Cambridge, England (1995)
14. Nielson, F., Nielson, H.R., Hankin, C.: Principles of Program Analysis. Springer, Heidelberg (1999)
15. Norris, J.: Markov Chains. Cambidge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, Cambridge (1997)
16. Roman, S.: Advanced Linear Algebra, 2nd edn. Springer, New York (2005)
17. Schneider, J.J., Kirkpatrick, S.: Stochastic Optimization. Springer, Heidelberg (2006)
18. Seneta, E.: Non-negative Matrices and Markov Chains. Springer, New York (1981)
19. Stannett, M.: X-machines and the halting problem: building a super-turing machine. Formal Aspects Comput. **2**, 331–341 (1990)
20. Woess, W.: Denumerable Markov Chains. EMS (2009)