PROBMELA: a modeling language for communicating probabilistic processes

Christel Baier, Frank Ciesinski, Marcus Größer [*]
Universität Bonn, Institut für Informatik I, Germany,
{baier|ciesinsk|groesser}@cs.uni-bonn.de

Abstract

Building automated tools to address the analysis of reactive probabilistic systems requires a simple, but expressive input language with a formal semantics based on a probabilistic operational model that can serve as starting point for verification algorithms. We introduce a higher level description language for probabilistic parallel programs with shared variables, message passing via synchronous and (perfect or lossy) fifo channels and atomic regions and provide a structured operational semantics. Applied to finite-state systems, the semantics can serve as basis for the algorithmic generation of a Markov decision process that models the stepwise behavior of the given system.

1 Introduction

Probabilistic aspects play a crucial role in many areas of computer science, e.g. randomization as an algorithmic concept or stochastic assumptions for modelling the expected behavior of unreliable components. Several variants of Markov chains and Markov decision processes have been suggested as operational models for probabilistic systems and starting points for verification algorithms [44, 10, 40, 21, 7]. While models based on Markov chains are purely probabilistic, in Markov decision processes nondeterminism and probabilism coexist. Thus, the latter are suitable to describe the behavior of systems that rely on a distributed randomized algorithm as the nondeterminism allows to describe parallelism by interleaving. Moreover, the nondeterminism can be useful for underspecification or to specify the interface with an unpredictable environment (e.g. human users).

Building automated tools for verifying probabilistic systems requires a simple, but expressive specification language for the system. The specification language must be equipped with a *formal semantics* that assigns to each program $\mathcal{Q}$ of the specification language a probabilistic model $\mathcal{M}_{\mathcal{Q}}$ such that the latter can serve as basis for the automated analysis (e.g., simulation or model checking against temporal logical specifications). Having in mind a probabilistic analogue to the model checker SPIN [24] with a guarded command language for the system specification, called PROMELA, we aim at building a probabilistic model checker with a probabilistic variant of PROMELA as input language. We called this language PROBMELA.

In our setting, the probabilistic system to be analyzed is described by a *program* $\mathcal{Q} = Q_1 \| \ldots \| Q_n$, consisting of a finite number of *processes* $Q_1, \ldots, Q_n$ that run in parallel and may communicate via shared variables, by handshaking via synchronous channels and asynchronously via fifo channels. Following the idea of probabilistic channel systems [1] the fifo channels can be declared to be unreliable and the probability of losing the message while inserting into the buffer can be specified.[1] The main concepts of the description language for the Q_i's are deterministic and randomized assignments, communication actions, conditional and repetitive statements with a nondeterministic choice between guarded commands (as in classical guarded command languages with channel-based message passing [17, 5]), atomic regions [32, 30, 36], and a probabilistic choice operator which assigns probabilities to the possible further behaviors.

In this paper, we provide an *operational semantics* for programs and processes, using SOS-rules in the Plotkin style [38], which yields the foundations for the automated construction of a Markov decision process (MDP) from its PROBMELA-specification. For sequential composition, conditional and repetitive commands, our semantic rules are obtained by adapting the corresponding rules for nonprobabilistic programs and processes [39, 4, 5]. The major difficulty is the treatment of *atomic regions* which requires a formalization of the cumulative probabilistic effect of (possibly nonterminating) processes. Although there are some simple special instances (e.g., if the body of the atomic region is loop-free or does not contain nondetermin-

[*] All authors are supported by the DFG-NWO-Project "VOSS". The last author is supported by the DFG-project “VERIAM”.

[1] Other types of unreliable fifo channels, e.g. fifo channels with possible duplication errors or fifo channels that might loose an arbitrary message of their buffer, are not considered here, but could be treated in a similar way.

istic choices), in general, we might obtain infinitely many distributions for the "final" configurations, and hence, an infinitely branching MDP for programs with atomic regions. This holds even in the case of finite-state programs which only use variables with a finite range and fifo channels with bounded buffers. We will show that finite-state programs have a representation by a finitely branching MDP that covers all relevant informations and that can be obtained in a *compositional* way using a denotational semantics.

Related work. The issue of operational and denotational semantics for probabilistic systems has been discussed extensively in the context of (data-abstract) process algebras, see e.g. [18, 43, 20, 31, 19, 33, 28, 6, 15, 27]. The language pGCL [34], probabilistic guarded command language, was introduced for formal reasoning about randomized algorithms by means of probabilistic expectations. The main ingredients of pGCL coincide with those of our basis language for processes without channel-based communication or atomic regions. Formal semantics for pGCL-like languages have been presented e.g. in [26, 35, 23, 16, 34, 14, 12] for reasoning with Hoare-like and wp-like calculi. Although the operational treatment of atomic regions will be in the spirit of the denotational semantics à la [16, 14, 12], we will use a different characterization of the input/output behavior of the body of an atomic region. The modelling language MoDeST [11] combines features of pGCL, process algebras with TCSP-like synchronization over common actions and other language-constructs, such as urgent actions, exception handling and process instantiation, to reason about stochastic timed systems. Channel-based message passing or semantic rules for atomic regions are not considered in [11].[2] In principle, MoDeST and PROBMELA could be combined to obtain a powerful description language whose operational semantics arises through combining the semantic rules in [11] with ours.

The probabilistic model checker PRISM [29] uses a variant of reactive modules [3] as modeling language for the description of a Markov decision process. The input language of the tool RAPTURE [25] is based on an explicit description of the step-wise behavior of the processes which resembles the format of the intermediate language we use in our implementation. PROBVERUS [22] uses a C-like imperative language with a Markov chain semantics. The concept of wait-commands in PROBVERUS has some similarities with our concept of atomic regions. Nondeterminism or asynchronous parallelism are not considered in [22].

Organization. Section 2 briefly explains our probabilistic model. In Section 3, we provide the syntax and MDP-semantics for the basis language where the processes communicate via shared variables only. Sections 4 and 5 are concerned with the semantic rules for channel-based message passing and the semantic treatment of atomic regions respectively. Section 6 sketches how to treat nested parallelism. Section 7 concludes the paper with a brief summary and reporting on the main features of an implementation for the model construction via the presented SOS-rules.

[2]MoDest uses a simple form of atomic statements that combines two or more assignments in a single transition, but does not allow for atomic regions that cumulate the effect of processes with nondeterminism, probabilism and iteration as we do.

2 Preliminaries

A *distribution* for a finite or countable set S denotes a function $\mu : S \to [0,1]$ such that $\sum_{s \in S} \ \mu(s) \leq 1$. We denote $\{s \in S : \mu(s) > 0\}$ by $\mathsf{Supp}(\mu)$.

We refer to $\mu(s)$ as the probability for s under μ. $\mathsf{Supp}(\mu)$ is called the support of μ. If $T \subseteq S$ then $\mu(T)$ stands short for $\sum_{t \in T} \mu(t) = 1$. μ is called substochastic if $\mu(S) < 1$. For $s \in S$, we write μ_s^1 for the distribution which assigns 1 to s and 0 to any other element in S. The distribution with empty support is denoted by 0. The set of distributions on S, $\mathsf{Distr}(S)$, is assumed to be equipped with the pointwise ordering, i.e., $\mu \leq \nu$ iff $\mu(s) \leq \nu(s)$ for all $s \in S$. Distribution μ is called a convex combination of distributions $\nu_1, \ldots \nu_m$ if there exists $q_1, \ldots, q_m \in [0,1]$ such that $q_1 + \ldots + q_m = 1$ and $\mu = q_1\nu_1 + \ldots + q_m\nu_m$.

A *Markov decision process* (MDP) is a tuple $\mathcal{M} = (S, \mathsf{Act}, \longrightarrow, S_0)$ where S is a set of states, Act a finite set of actions, $\longrightarrow \subseteq S \times \mathsf{Act} \times \mathsf{Distr}(S)$ the transition relation, $S_0 \subseteq S$ the set of initial states.[3] For $s \in S$, $a \in \mathsf{Act}$, $\mathsf{Steps}(s) = \{\langle a, \mu \rangle : s \xrightarrow{a} \mu\}$ and $\mathsf{Steps}_a(s) = \{\mu \in \mathsf{Distr}(S) : \langle a, \mu \rangle \in \mathsf{Steps}(s)\}$. State s is terminal if $\mathsf{Steps}(s) = \emptyset$. $\mathcal{M}$ is finite-state if S is finite, and finitely branching if $\mathsf{Steps}(s)$ is finite for all states s. A finite MDP means a MDP that is finite-state and finitely branching.

The intuitive behavior of $\mathcal{M}$ in state s is as follows. First, there is a nondeterministic choice between the action-distribution pairs $\langle a, \mu \rangle \in \mathsf{Steps}(s)$. If $\langle a, \mu \rangle$ is the chosen alternative then action a is performed. The effect of a is described by distribution μ which provides the probabilities for the possible successor states. If μ is substochastic then with probability $1 - \mu(S)$ the system blocks in state s.

We often write $s \xrightarrow{a} t$ rather than $s \xrightarrow{a} \mu_t^1$. If the action set of a MDP $\mathcal{M}$ is a singleton set then we skip the action label and simply write $s \longrightarrow \mu$.

A (finite) *path* in $\mathcal{M}$ is a sequence $\sigma = s_0, a_0, \mu_0, \ldots, s_{n-1}, a_{n-1}, \mu_{n-1}, s_n \in (S \times \mathsf{Act} \times \mathsf{Distr}(S))^* \times S$ such that $s_0 \in S_0$, $\langle a_i, \mu_i \rangle \in \mathsf{Steps}(s_i)$ and $s_{i+1} \in \mathsf{Supp}(\mu_i)$. $\mathsf{last}(\sigma)$ denotes the last state of σ. We refer to the value $\mathsf{Pr}(\sigma) = \prod_{1 \leq i \leq n} \mu_{i-1}(s_i)$ as the probability of σ. Paths denotes the set of all paths in $\mathcal{M}$.

[3]We slightly depart from the standard definition of a MDP, as e.g. introduced in [41], and deal with a model that essentially agrees with probabilistic automata [42].

A *scheduler* of $\mathcal{M}$ means an instance that resolves the nondeterminism. Formally, a (history-dependent, deterministic) scheduler for $\mathcal{M}$ is a function $D : \mathsf{Paths} \to (\mathsf{Act} \times \mathsf{Distr}(S)) \cup \{\perp\}$ such that $D(\sigma) \in \mathsf{Steps}(\mathsf{last}(\sigma))$ if $\mathsf{last}(\sigma)$ is non-terminal and $D(\sigma) = \perp$ if $\mathsf{last}(\sigma)$ is terminal. A D-path denotes a path that can be generated by D. By a *simple scheduler* we mean a scheduler E where $E(\sigma_1) = E(\sigma_2)$ for all finite paths with $\mathsf{last}(\sigma_1) = \mathsf{last}(\sigma_2)$. That is, simple schedulers are given by a function $E : S \to (\mathsf{Act} \times \mathsf{Distr}(S)) \cup \{\perp\}$ such that $E(s) \in \mathsf{Steps}(s)$ if s is non-terminal and $E(s) = \perp$ if s is terminal. Note that, for finite MDP $\mathcal{M}$, the set of all simple schedulers is finite, while in general the set of all schedulers is infinite.

3 The basis language of PROBMELA

Programs of the specification language PROBMELA consist of a finite number of processes $Q_1, \ldots, Q_n$ that run in parallel and communicate via shared variables or via message passing over channels. We write them as $\mathcal{Q} = Q_1 \| \ldots \| Q_n$. We start with the basis language that only allows for communication via shared variables.

The *variables* used in a program $\mathcal{Q}$ may be typed (integer, boolean, char, real, etc.) and either global or local to some process of $\mathcal{Q}$. We skip the details of variable declarations as they are irrelevant for the purposes of this paper. As local variables can be renamed in case they occur in more than one process, we may treat all variables as global variables and assume that we are given a finite set Var of variables and a domain (type) $\mathsf{Dom}(x)$ for any variable x. We write Values to denote the set of all possible values for the variables, i.e., $\mathsf{Values} = \bigcup_{x \in \mathsf{Var}} \mathsf{Dom}(x)$. In addition, we assume that the variable declaration of program $\mathcal{Q}$ contains an *initial condition* for the variables. The abstract syntax of processes (denoted by P, Q, R) is as follows:[4]

$$\begin{array}{rl} P ::= & \mathsf{skip} \;\big|\; x := \mathsf{expr} \;\big|\; x := \mathsf{random}(V) \;\big|\; P_1; P_2 \;\big| \\ & \underline{\mathbf{IF}} \;\; :: \mathsf{bexpr}_1 \Rightarrow P_1 \ldots :: \mathsf{bexpr}_n \Rightarrow P_n \;\; \underline{\mathbf{FI}} \;\;\big| \\ & \underline{\mathbf{DO}} \;\; :: \mathsf{bexpr}_1 \Rightarrow P_1 \ldots :: \mathsf{bexpr}_n \Rightarrow P_n \;\; \underline{\mathbf{OD}} \;\;\big| \\ & \underline{\mathbf{PIF}}\, [p_1] \;\Rightarrow P_1 \ldots [p_n] \;\Rightarrow P_n \; \underline{\mathbf{FIP}} \end{array}$$

Here, V is a nonempty and finite subset of $\mathsf{Dom}(x)$ and $p_1, \ldots, p_n$ are real values in the interval $[0, 1]$ such that $p_1 + \ldots + p_n \leq 1$.

In assignments $x := \mathsf{expr}$, we require type-consistency of variable x and expression expr. The precise syntax of the expressions (expr) and boolean expressions (bexpr) is not of importance here. We may assume that expr is built from constants in $\mathsf{Dom}(x)$, variables y of the same type as x and operators on $\mathsf{Dom}(x)$, while the boolean expressions are propositional formulas built by comparisons of expressions.

[4]The basis language of PROBMELA contains some more features, such as random assignments with non-uniform distributions, arrays, procedure calls or pointers which are not explained in the paper.

Statements of the form $\mathsf{bexpr} \Rightarrow P$ are called *guarded commands*, where we refer to bexpr as a *(boolean) guard*.

The intuitive meaning of the commands is as follows. skip stands for a process that terminates in one step without affecting the values of the variables. Assignment and sequential composition ($P_1; P_2$) have the obvious meaning. The effect of a random assignment $x := \mathsf{random}(V)$ is that a value $v \in V$ is probabilistically chosen and assigned to variable x (uniform distribution for V is assumed). The conditional command $\underline{\mathbf{IF}} :: \mathsf{bexpr}_1 \Rightarrow P_1 \ldots :: \mathsf{bexpr}_n \Rightarrow P_n \; \underline{\mathbf{FI}}$ stands for a non-deterministic choice between those processes P_i where the guard bexpr_i is satisfied in the current state. If none of the guards is fulfilled, an IF-FI–command blocks (and waits until another process changes the values of the shared variables such that one of the guards eventually evaluates to true). Repetitive commands $\underline{\mathbf{DO}} :: \mathsf{bexpr}_1 \Rightarrow P_1 \ldots :: \mathsf{bexpr}_n \Rightarrow P_n \; \underline{\mathbf{OD}}$ stand for the iterative execution of the non-deterministic choice between the guarded commands $\mathsf{bexpr}_i \Rightarrow P_i$. Unlike conditional commands, DO-OD-loops do not block if all guards are violated. A single-guarded loop $\underline{\mathbf{DO}} :: \mathsf{bexpr} \Rightarrow P \; \underline{\mathbf{OD}}$ has the same effect as a WHILE-loop with body P and termination condition $\neg\mathsf{bexpr}$.[5] The probabilistic choice operator PIF-FIP can be viewed as an analogue to IF-FI-statements where the former chooses its alternatives according to given probabilities instead of boolean guards. Intuitively, $\underline{\mathbf{PIF}}\; [p_1] \Rightarrow P_1 \ldots [p_n] \Rightarrow P_n \; \underline{\mathbf{FIP}}$ performs a stochastic experiment with n possible events occurring with probabilities $p_1, \ldots, p_n$, followed by the execution of the (probabilistically) chosen process P_i. We do not require the values p_i to sum up to 1. The "missing probabilities" stand for the possibility of a deadlock. E.g., $\underline{\mathbf{PIF}}\; [0.5] \Rightarrow P_1 \; [0.2] \Rightarrow P_2 \; \underline{\mathbf{FIP}}$ has a deadlock chance of 30%.

An *operational semantics* for the programs and processes that formalizes the stepwise behavior can be provided by means of MDPs. In the sequel, let $\mathcal{Q} = Q_1 \| \ldots \| Q_n$ be a program. The *states* in the MDP $\mathcal{M}_\mathcal{Q}$ for program $\mathcal{Q}$ have the form $\langle P_1, \ldots, P_n, \eta \rangle$ where η is a variable evaluation, i.e., a function $\eta : \mathsf{Var} \to \mathsf{Values}$ such that $\eta(x) \in \mathsf{Dom}(x)$ for all $x \in \mathsf{Var}$, and P_i a "subprocess" of Q_i, i.e., either a process that stands for the "remaining" process of Q_i that still has to be executed or the special symbol exit if Q_i has terminated. (Intuitively, we may think of P_i to be a program counter for process Q_i.) The *initial states* in $\mathcal{M}_\mathcal{Q}$ are the tuples $\langle Q_1, \ldots, Q_n, \eta \rangle$ where η is a variable evaluation under which all initial conditions for the variables are fulfilled. Action labels are not of importance and therefore omitted. The transition relation $\longrightarrow_\mathcal{Q}$ in $\mathcal{M}_\mathcal{Q}$ is obtained by the following *interleaving rule*:

$$\frac{\langle P_i, \eta \rangle \longrightarrow \mu}{\langle P_1, \ldots, P_i, \ldots, P_n, \eta \rangle \longrightarrow_\mathcal{Q} \nu}$$

[5]We depart here from the PROMELA-syntax and semantics which uses a special command "break" to terminate a loop.

$$\langle \mathsf{skip}, \eta\rangle \longrightarrow \langle \mathsf{exit}, \eta\rangle$$

$$\langle x := \mathsf{expr}, \eta\rangle \longrightarrow \langle \mathsf{exit}, \eta[x := \mathsf{expr}]\rangle$$

$$\langle x := \mathsf{random}(V), \eta\rangle \longrightarrow \sum_{v\in V} \tfrac{1}{|V|}\mu^1_{\langle \mathsf{exit}, \eta[x:=v]\rangle}$$

$$\langle \underline{\mathbf{PIF}}\ [p_1] \Rightarrow P_1 \ \dots\ [p_n] \Rightarrow P_n\ \underline{\mathbf{FIP}}, \eta\rangle \longrightarrow \sum_{\substack{1\le i\le n\\ P_i = P}} p_i \cdot \mu^1_{\langle P_i, \eta\rangle}$$

$$\frac{\eta \models \mathsf{bexpr}_i}{\langle \underline{\mathbf{IF}}\ \dots :: \mathsf{bexpr}_i \Rightarrow P_i\ \dots\ \underline{\mathbf{FI}} :, \eta\rangle \longrightarrow \langle P_i, \eta\rangle}$$

$$\frac{\langle P_1, \eta\rangle \longrightarrow \mu}{\langle P_1; P_2, \eta\rangle \longrightarrow \nu} \quad \text{where} \quad \begin{array}{l}\nu(\langle P_1'; P_2, \eta'\rangle) = \mu(\langle P_1', \eta'\rangle)\\ \text{and we identify } \mathsf{exit}; P \text{ with } P,\\ \nu(\cdot) = 0 \text{ otherwise.}\end{array}$$

For $Q = \underline{\mathbf{DO}}\ :: \mathsf{bexpr}_1 \Rightarrow P_1\ \dots\ :: \mathsf{bexpr}_n \Rightarrow P_n\ \underline{\mathbf{OD}}$:

$$\frac{\eta \models \mathsf{bexpr}_i}{\langle Q, \eta\rangle \longrightarrow \langle P_i; Q, \eta\rangle} \qquad \frac{\eta \not\models \mathsf{bexpr}_1 \vee \dots \vee \mathsf{bexpr}_n}{\langle Q, \eta\rangle \longrightarrow \langle \mathsf{exit}, \eta\rangle}$$

Figure 1. SOS-rules for the basis language

where $\nu(\langle P_1', \dots, P_i', \dots, P_n', \eta'\rangle) = \mu(\langle P_i', \eta'\rangle)$ if $P_k = P_k'$ for $k \in \{1, \dots, n\} \setminus \{i\}$ and 0 otherwise. The SOS-rules for $\longrightarrow$ (see Fig. 1)[6] are obtained by adapting the corresponding rules for non-probabilistic processes [39, 4, 5].

4 Channel-based message passing

We now extend the basis language by communication via synchronous and fifo channels. We skip the details of (global or local) channel declarations and assume that we are given a fixed, finite set Chan of channels used in the given program $\mathcal{Q} = Q_1 \| \dots \| Q_n$. Chan is partitioned into two disjoint subsets: the set SChan of *synchronous channels* and the set FChan of *fifo channels*. The letter d will be used to denote a (synchronous or fifo) channel, c for synchronous and f for fifo channels. Any channel d is associated with a domain $\mathsf{Dom}(d)$ which declares the type of messages that can be sent via d. In addition, the declaration of a fifo channel f specifies the *capacity* of the buffer for channel f by a value $\mathsf{cap}(f) \in I\!N \cup \{\infty\}$ and the *probability for losing a message* while inserting into the buffer by a value $p(f) \in [0, 1[$. ($p(f) = 0$ indicates a perfect channel.) The *syntax of processes* is extended by communication actions $d!\mathsf{expr}$ (sending the current value of expr along channel d) and $d?x$ (receiving a value for variable x from channel d) and guarded commands where the guard may ask for a communication action. (Type-consistency for d, x and expr is assumed.) We refer to guards of the form $\mathsf{bexpr}\wedge d?x$ or $\mathsf{bexpr}\wedge d!\mathsf{expr}$ as *generalized guards*. The intuitive meaning of a generalized guard g is that the boolean part bexpr holds for the variable evaluation and that the communication action is enabled in the current configuration. In addition, we allow for guarded commands of the form "$\underline{\mathbf{ELSE}} \Rightarrow P$" which are enabled if none of the other guarded commands (with boolean or generalized guards) is enabled. Note that the ELSE-option in loops always leads to non-terminating behaviors.

$$\begin{array}{rcl} P & ::= & \mathsf{skip} \mid x := \mathsf{expr} \mid x := \mathsf{random}(V) \mid \\ & & d?x \mid d!\mathsf{expr} \mid P_1; P_2 \mid \\ & & \underline{\mathbf{IF}}\ \ :: g_1 \Rightarrow P_1\ \dots\ :: g_n \Rightarrow P_n\ \ \underline{\mathbf{FI}} \mid \\ & & \underline{\mathbf{DO}}\ \ :: g_1 \Rightarrow P_1\ \dots\ :: g_n \Rightarrow P_n\ \ \underline{\mathbf{OD}} \mid \\ & & \underline{\mathbf{PIF}}\ \ [p_1] \Rightarrow P_1 \dots [p_n] \Rightarrow P_n\ \underline{\mathbf{FIP}} \end{array}$$

where the g_i's have the form bexpr, $\mathsf{bexpr}\wedge d?x$ or $\mathsf{bexpr}\wedge d!\mathsf{expr}$ or $g_i = \underline{\mathbf{ELSE}}$. We simply write $d?x$ resp. $d!\mathsf{expr}$ rather than $\mathsf{tt}\wedge d?x$ resp. $\mathsf{tt}\wedge d!\mathsf{expr}$.

Example 1. The IPv4 zeroconf protocol [9, 8] is a simple randomized algorithm for fully automated self-configuration of IP network interfaces. The goal is to assign a unique network address that is (probably) not already in use by another device. The rough idea is as follows. Let H be the host which attempts to configure its own IP link with a new local address. Host H starts by choosing at random an address addr from the address space and broadcasts the question whether addr is already in use by another host. If there is another host H' using the address addr then H' sends a reply "No, don't use the address addr", in which case H restarts the whole procedure with a new randomly chosen address. The difficulty is that the network might be unreliable and the question sent by H as well as the reply might get lost. For this reason, after sending the address, H waits for a fixed amount of time and then resends its request. Only if after a fixed number N of iterations, where H resends repeatedly the chosen address and no reply arrived at H, then H considers the chosen address addr not to be assigned to another host and starts using it.
In PROBMELA, the IPv4 zeroconf protocol can be modelled by a process P_{zc} (see Fig. 2) which communicates via lossy fifo channels f and e with the other hosts in the network (i.e., we assume $p(f)$, $p(e) > 0$). Process P_{zc} sends the randomly chosen address addr via channel f to the network, while channel e serves for sending the reply. The outer loop of P_{zc} stands for the iterations where P_{zc} chooses an address, while the inner loop stands for the attempts of P_{zc} to get a reply for the chosen address ("probing phase"). Integer variable i in P_{zc} serves as counter for the attempts to send the probabilistically chosen address. The boolean variable accepted indicates whether an address has been assigned to H, while the meaning of probing is that the protocol is in its probing phase. The initial condition is $\neg\mathsf{accepted}$.

[6]We write $s \longrightarrow t$ for $s \longrightarrow \mu^1_t$ and $[\mathsf{expr}]_\eta$ for the value of expr when each variable y occurring in expr is interpreted by $\eta(y)$. $\eta[x := \mathsf{expr}]$ denotes the variable evaluation that assigns value $[\mathsf{expr}]_\eta$ to variable x, while the value for any other variable y agrees with $\eta(y)$. $\eta \models \mathsf{bexpr}$ denotes that bexpr evaluates to true if interpreted over η.

```
DO ::¬accepted ⇒
      addr := random(IP_adresses);
      $c$!timer_on; $f$!addr; $i := 0$; probing := tt;
      DO :: probing ⇒
            IF  :: $(i < N) \wedge c?x$ ⇒
                     $i := i + 1$; $c$!timer_on; $f$!addr
                :: $(i < N) \wedge e?y$ ⇒
                     probing := ff; $c$!reset
                :: $(i = N)$ ⇒
                     probing := ff; $c$!reset;
                     accepted := tt
            FI
      OD
OD
```

Figure 2. Process P_{zc}

To model the time passage where P_{zc} has to wait for a reply before resending the address, we use an additional process that models a timer which communicates via a synchronous channel c with P_{zc}. The timer is activated by P_{zc} by sending the signal timer_on along channel c. If $i < N$ then P_{zc} is forced to wait in its IF-FI-statement until either the timer sends the timeout signal along channel c, in which case P_{zc} reactivates the timer and resends the address, or another host puts a reply into the buffer for channel e, in which case P_{zc} resets the timer (via sending the signal reset along channel c) and repeats the whole procedure. The behavior of the timer is described by the following process:

```
DO  :: $c?o$ ⇒ IF :: $c?r$ ⇒ skip :: $c$!timeout ⇒ skip FI
    :: ELSE ⇒ skip
OD
```

Intuitively, variable o stands for the signal activating the timer, while variable ”r” represents the reset-signal.
The abstract behavior of the network can be described by the process

```
DO  :: $f?z$ ⇒ IF :: $z \in A$ ⇒ $e$!”No” :: ELSE ⇒ skip FI
    :: tt ⇒ skip
OD
```

Variable z stands for the address sent by P_{zc} and A for the set of IP addresses that are assigned to some host.[7] ■

Operational semantics. The states of the MDP $\mathcal{M}_\mathcal{Q}$ for program $\mathcal{Q} = Q_1 \| \ldots \| Q_n$ consist of control components $P_1, \ldots, P_n$ for the processes $Q_1, \ldots, Q_n$, a variable evaluation η and a channel evaluation ξ, i.e., a function $\xi : \mathsf{FChan} \to \mathsf{Values}^*$ where $\xi(f) \in \mathsf{Dom}(f)^*$ and $|\xi(f)| \leq \mathsf{cap}(f)$. The initial states of $\mathcal{M}_\mathcal{Q}$ are the states $\langle Q_1, \ldots, Q_n, \eta, \xi_\emptyset \rangle$ where η is a variable evaluation satisfying the initial condition and where $\xi_\emptyset$ assigns to any fifo-channel the empty word. (Initially, all buffers are empty.)
Action labels are irrelevant for $\mathcal{M}_\mathcal{Q}$. Except for handshaking via synchronous channels, the interleaving rule for $\longrightarrow_\mathcal{Q}$ is still appropriate. To handle synchronous communication, we need a semantic rule for $\longrightarrow_\mathcal{Q}$ that “combines” two local transitions of certain processes Q_i and Q_j which stand for matching synchronous I/O-operations $c?x$ and c!expr. For this reason, we use the following action-set Act

$$\{\tau, \mathsf{else}\} \cup \{\mathsf{read}(c, v), \mathsf{write}(c, v) : c \in \mathsf{SChan},\ v \in \mathsf{Dom}(c)\}$$

for the transition relation $\longrightarrow$ for the processes. Action label else is used for the transitions representing the cases where the else-branch of a conditional or repetitive command is chosen. The action symbol τ stands for any action different from else and the synchronous communication actions $c?x$ and c!expr. Action label $\mathsf{write}(c, v)$ is used for any transition of a process Q_j in which Q_j offers a communication action c!expr where the current value of expr is v. Similarly, action-label $\mathsf{read}(c, v)$ denotes the request for a synchronous input action $c?x$. In the latter case, v is an arbitrary value of $\mathsf{Dom}(c)$. Thus, for the *local view* of the processes, we model the possible effect of $c?x$ by a nondeterministic choice between all assignments “$x := v$” for $v \in \mathsf{Dom}(c)$. The nondeterminism will be resolved in the *global* transition system $\mathcal{M}_\mathcal{Q}$ which uses the following *synchronization rule*:

$$\frac{\langle P_i, \eta, \xi \rangle \xrightarrow{\mathsf{read}(c,v)} \mu_r, \quad \langle P_j, \eta, \xi \rangle \xrightarrow{\mathsf{write}(c,v)} \mu_w}{\langle P_1, \ldots, P_i, \ldots, P_j, \ldots, P_n, \eta, \xi \rangle \longrightarrow_\mathcal{Q} \nu}$$

where

$$\nu(\langle \ldots, P_{i-1}, P_i', P_{i+1}, \ldots, P_{j-1}, P_j', P_{j+1}, \ldots, \eta', \xi \rangle) = \mu_r(\langle P_i', \eta', \xi \rangle) \cdot \mu_w(\langle P_j', \eta, \xi \rangle)$$

and $\nu(\cdot) = 0$ in all remaining cases. (Note that message passing via a synchronous channel does not affect the channel evaluation and that only the read-action can modify the variable evaluation.) The semantics of the else-branches is provided by the rule

$$\frac{\langle P_i, \eta, \xi \rangle \xrightarrow{\mathsf{else}} \mu, \quad \neg \mathsf{Syn}_i(P_1, \ldots, P_i, \ldots, P_n, \eta, \xi)}{\langle P_1, \ldots, P_i, \ldots, P_n, \eta, \xi \rangle \longrightarrow_\mathcal{Q} \nu}$$

where $\nu(\langle P_1, \ldots, P_{i-1}, P_i', P_{i+1}, \ldots, \ldots, \ldots, P_n, \eta', \xi' \rangle) = \mu(\langle P_i', \eta', \xi' \rangle)$ and $\nu(\cdot) = 0$ in all remaining cases. Predicate $\mathsf{Syn}_i(s)$ asserts that, in the global state $s = \langle P_1, \ldots, P_i, \ldots, P_n, \eta, \xi \rangle$, process P_i can synchronize with some other process P_j. That is, $\mathsf{Syn}_i(s)$ iff there exist an index $j \neq i$ and a pair $\langle a, \bar{a} \rangle$ of matching synchronous I/O-operations (i.e., $\{a, \bar{a}\} = \{\mathsf{write}(c, v),\ \mathsf{read}(c, v)\}$) with $\langle P_i, \eta, \xi \rangle \xrightarrow{a} \mu_1$ and $\langle P_j, \eta, \xi \rangle \xrightarrow{\bar{a}} \mu_2$ for some μ_1, μ_2. In

[7] We assume here that A is given. To verify the protocol independent of a concrete configuration of the network, we may assume an initialization process which chooses nondeterministically addresses from the whole address space to generate the address-set A.

Asynchronous communication via fifo channels:

$$\frac{|\xi(f)| > 0,\ \ v = \mathsf{first}(\xi, f)}{\langle f?x, \eta, \xi\rangle \xrightarrow{\tau} \langle \mathsf{exit}, \eta[x := v], \xi[\mathsf{remove}(f)]\rangle} \qquad \frac{|\xi(f)| < \mathsf{cap}(f),\ \ v = [\mathsf{expr}]_\eta}{\langle f!\mathsf{expr}, \eta, \xi\rangle \xrightarrow{\tau} \mu}$$

$$\text{where } \mu(\langle P, \eta', \xi'\rangle) = \begin{cases} p(f) & : \text{ if } \langle P, \eta', \xi'\rangle = \langle \mathsf{exit}, \eta, \xi\rangle \\ 1 - p(f) & : \text{ if } v = [\mathsf{expr}]_\eta \text{ and } \langle P, \eta', \xi'\rangle = \langle \mathsf{exit}, \eta, \xi[\mathsf{add}(f, v)]\rangle \\ 0 & : \text{ otherwise.} \end{cases}$$

Synchronous communication actions:

$$\frac{v \in \mathsf{Dom}(c)}{\langle c?x, \eta, \xi\rangle \xrightarrow{\mathsf{read}(c,v)} \langle \mathsf{exit}, \eta[x := v], \xi\rangle} \qquad \frac{v = [\mathsf{expr}]_\eta}{\langle c!\mathsf{expr}, \eta, \xi\rangle \xrightarrow{\mathsf{write}(c,v)} \langle \mathsf{exit}, \eta, \xi\rangle}$$

Conditional commands with I/O-request:

$$\frac{\eta \models \mathsf{bexpr},\ \ |\xi(f)| > 0,\ \ v = \mathsf{first}(\xi, f)}{\langle \underline{\mathbf{IF}} \ldots :: \mathsf{bexpr} \wedge f?x \Rightarrow P \ :: \ldots \underline{\mathbf{FI}}, \eta, \xi\rangle \xrightarrow{\tau} \langle P, \eta[x := v], \xi[\mathsf{remove}(f)]\rangle} \qquad \frac{\eta \models \mathsf{bexpr},\ \ |\xi(f)| < \mathsf{cap}(f),\ \ v = [\mathsf{expr}]_\eta}{\langle \underline{\mathbf{IF}} \ldots :: \mathsf{bexpr} \wedge f!\mathsf{expr} \Rightarrow P \ldots \underline{\mathbf{FI}}, \eta, \xi\rangle \xrightarrow{\tau} \mu}$$

where $\mu(\langle P, \eta, \xi[\mathsf{add}(f, v)]\rangle) = 1 - p(f)$, $\mu(\langle P, \eta, \xi\rangle) = p(f)$ and $\mu(\cdot) = 0$ otherwise.

$$\frac{\eta \models \mathsf{bexpr},\ \ v \in Dom(c)}{\langle \underline{\mathbf{IF}} \ldots :: \mathsf{bexpr} \wedge c?x \Rightarrow P \ :: \ldots \underline{\mathbf{FI}}, \eta, \xi\rangle \xrightarrow{\mathsf{read}(c,v)} \langle P, \eta[x := v], \xi\rangle} \qquad \frac{\eta \models \mathsf{bexpr},\ \ v = [\mathsf{expr}]_\eta}{\langle \underline{\mathbf{IF}} \ldots :: \mathsf{bexpr} \wedge c!\mathsf{expr} \Rightarrow P \ :: \ldots \underline{\mathbf{FI}}, \eta, \xi\rangle \xrightarrow{\mathsf{write}(c,v)} \langle P, \eta, \xi\rangle}$$

If Q is $\underline{\mathbf{IF}} :: g_1 \Rightarrow P_1 \ldots :: g_n \Rightarrow P_n :: \underline{\mathbf{ELSE}} \Rightarrow R_1 \ldots :: \underline{\mathbf{ELSE}} \Rightarrow R_m\ \underline{\mathbf{FI}}$
where $g_1, \ldots, g_n$ are boolean or generalized guards:

$$\frac{\langle \eta, \xi\rangle \not\models g_i,\ i = 1, \ldots, n}{\langle Q, \eta, \xi\rangle \xrightarrow{\mathsf{else}} \langle R_j, \eta, \xi\rangle} \quad j = 1, \ldots, m.$$

Figure 3. SOS-rules for communication actions

addition we use the interleaving rule of section 3 for action label τ, we just have to add channel evaluations for this label.

It remains to formalize the stepwise behavior of the processes via the transition relation $\longrightarrow$. For skip, assignments and sequential composition we may work with essentially the same axioms and rules shown in Fig. 1, the only difference being that we have to add the channel evaluation and the action label τ. The same holds for conditional commands, provided that the guard of the chosen guarded command is boolean. Fig. 3 shows the rules for communication actions.[8] For generalized guard g with an asynchronous communication action, $\langle \eta, \xi\rangle \models g$ iff the boolean part of g holds under η and ξ enables the communication request in g. If $g = \mathsf{bexpr}$ is a boolean guard then $\langle \eta, \xi\rangle \models g$ iff $\eta \models \mathsf{bexpr}$. For technical reasons, we always assume $\langle \eta, \xi\rangle \not\models g$ if $g \in \{\mathsf{bexpr} \wedge c?x, \mathsf{bexpr} \wedge c!\mathsf{expr}\}$. This does *not* mean that the synchronization is not enabled, but reflects the local view of the processes where the else-branches of IF-FI-statements are regarded as potential alternatives as long as none of the booleans guards or generalized guards with asynchronous communication are fulfilled in the current configuration $\langle \eta, \xi\rangle$.

The rules for loops are similar and omitted here.

[8] $\xi[\mathsf{add}(f, v)]$ denotes the channel evaluation that results from ξ by inserting value v at the end of the buffer for channel f, $\xi[\mathsf{remove}(f)]$ the channel evaluation that agrees with ξ, except for channel f where the front element has been removed. If $|\xi(f)| \geq 1$ then $\mathsf{first}(\xi, f)$ is the first element in the buffer for channel f.

5 Atomic regions

We now consider programs $\mathcal{Q} = Q_1 \| \ldots \| Q_n$ whose processes Q_i may communicate via shared variables and channels and may contain *atomic regions*. That is, we extend the syntax for processes by commands of the form $\mathsf{atomic}\{P\}$. The intuitive meaning of $\mathsf{atomic}\{P\}$ is that the activities during one possible terminating computation of P are cumulated into a single transition, to avoid any interaction by other processes. Atomic regions can serve as a user-driven *compactification technique* for the MDP of a program. They can be used to model mutual exclusion scenarios, without the explicit representation of semaphores as additional shared variables or other mechanisms for concurrency control. Moreover, they are helpful for abstraction purposes as they can yield a simplified model which abstracts away from computations that are internal to some process.

For instance, consider a parallel sorting algorithm modelled by three processes P_{main}, P^1_{sort} and P^2_{sort} where P_{main} splits the given sequence of values into two disjoint subsequences of (roughly) the same length which are sorted in parallel by P^1_{sort} and P^2_{sort}, e.g. using randomized quicksort, and finally merged by P_{main}. To prove total correctness, a simplified model obtained by using $\mathsf{atomic}\{P^1_{sort}\}$ and $\mathsf{atomic}\{P^2_{sort}\}$ suffices and leads to a reduced state-space, as all intermediate steps of P^i_{sort} are collapsed into a single transition, and thus, avoids the representation of all possible interleavings of P^1_{sort} and P^2_{sort}.[9]

[9] Of course, if the analysis detects an error then for debugging the intermediate steps of P^i_{sort} might be relevant. However, P^i_{sort} could be verified/falsified in isolation.

While many authors require the body of an atomic region to be loop-free, we do not make any syntactic restriction on the body P of an atomic region, except that P does not contain further atomic regions. (Nested atomic regions can be removed without changing the effect.) Non-termination inside an atomic region is interpreted as *livelock*.

There are several possibilities to handle blocking inside atomic regions on the semantic level.[10] The PROMELA-semantics suspends the execution of P and allows for transitions of other processes. Another possibility is to consider blocking inside an atomic region as a *deadlock* from which no further activities for the whole program are possible. We follow here the latter interpretation which – for non-probabilistic processes without the else-option in IF-FI- or DO-OD-commands – can be described by the rule

$$\frac{\langle P,\eta,\xi\rangle \longrightarrow^* \langle \mathsf{exit},\eta',\xi'\rangle}{\langle \mathsf{atomic}\{P\},\eta,\xi\rangle \xrightarrow{\tau} \langle \mathsf{exit},\eta',\xi'\rangle}$$

where $\longrightarrow^*$ denotes the transitive, reflexive closure of $\xrightarrow{\tau}$ (see e.g. [5]). For processes with the else-option, we deal with the transitive, reflexive closure of $\xrightarrow{\tau} \cup \xrightarrow{\mathsf{else}}$. Thus, e.g. in $\mathsf{atomic}\{$**IF** $:: c?x \Rightarrow P_1 ::$ **ELSE** $\Rightarrow P_2$ **FI**$\}$, process P_2 will be executed. Our goal is to provide a probabilistic analogue to the above rule for non-probabilistic processes.

Notation. In the sequel, P stands for a fixed process as in Sect. 4 and $\gamma = \langle \eta, \xi\rangle$ for a variable-channel-evaluation pair. $\mathcal{M} = \mathcal{M}_{\langle P,\gamma\rangle}$ denotes the MDP that describes the stepwise behavior of P started with the initial configuration γ when the transitions for synchronous communication actions are ignored. That is, $\langle P,\gamma\rangle$ is the initial state of $\mathcal{M}$, the state space S of $\mathcal{M}$ consists of all states that are reachable from $\langle P,\gamma\rangle$ under $\xrightarrow{\tau} \cup \xrightarrow{\mathsf{else}}$. The transition relation in $\mathcal{M}$ is $\xrightarrow{\tau} \cup \xrightarrow{\mathsf{else}}$ restricted to S. T denotes the set of terminal states in $\mathcal{M}$, T_{exit} the set of states of the form $\langle \mathsf{exit},\gamma'\rangle$. Any distribution μ with $\mathsf{Supp}(\mu) \subseteq T_{\mathsf{exit}}$ is called an *exit-distribution*. If $\mathsf{Supp}(\mu) \subseteq T$ then μ is called an *output-distribution* and the induced exit-distribution $\mu|_{\mathsf{exit}}$ is given by $\mu|_{\mathsf{exit}}(t) = \mu(t)$ if $t \in T_{\mathsf{exit}}$ and $\mu(u) = 0$ if $u \in S \setminus T_{\mathsf{exit}}$. If D is a scheduler for $\mathcal{M}$ then the output-distribution μ_D is defined by $\mu_D(t) = \sum\{\mathsf{Pr}(\sigma) : \sigma \text{ is a } D\text{-path with } \mathsf{last}(\sigma) = t\}$ for any state $t \in T$. ■

The obvious idea to provide a SOS-rule for atomic regions is the following probabilistic variant of the above mentioned rule for non-probabilistic processes:

$$\frac{\langle P,\gamma\rangle \longrightarrow^* \mu \text{ where } \mu \text{ is an output-distribution}}{\langle \mathsf{atomic}\{P\},\gamma\rangle \xrightarrow{\tau} \mu|_{\mathsf{exit}}}$$

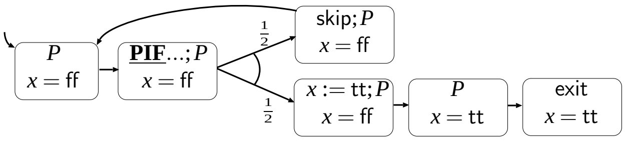
where $\longrightarrow^*$ is the least relation on $S \times \mathsf{Distr}(S)$ such that (1) $s \longrightarrow^* \mu_s^1$ for all states s and (2) If $s \xrightarrow{a} \mu$ where $a \in \{\tau, \mathsf{else}\}$ and $t \longrightarrow^* \nu_t$ for $t \in \mathsf{Supp}(\mu)$ then

$$s \longrightarrow^* \textstyle\sum_{t\in\mathsf{Supp}(\mu)} \mu(t)\cdot\nu_t.$$

The above rule is appropriate for loop-free processes (because they only have finite computations), but it is not for processes with loops. E.g., the following process P

DO $:: \neg x \Rightarrow$ **PIF** $[0.5] \Rightarrow x := \mathsf{tt}\ [0.5] \Rightarrow \mathsf{skip}$ **FIP** **OD**

(with boolean variable x and initial evaluation "$x = \mathsf{ff}$") terminates with probability 1, but there is no output-distribution μ with $\langle P, x{=}\mathsf{ff}\rangle \longrightarrow^* \mu$.



To avoid this problem, we replace the above rule with the following one which cumulates the effect of maximal (possibly infinite) computations by means of the output-distributions of schedulers:[11]

$$\frac{D \text{ is a scheduler for } \mathcal{M}_{\langle P,\gamma\rangle}}{\langle \mathsf{atomic}\{P\},\gamma\rangle \xrightarrow{\tau} \mu_D|_{\mathsf{exit}}}$$

E.g., for P as above, there is exactly one scheduler D (as P does not contain nondeterminism), the distribution μ_D is an exit distribution which assigns probability 1 to state $\langle \mathsf{exit}, x{=}\mathsf{tt}\rangle$. Thus, $\langle \mathsf{atomic}\{P\}, x{=}\mathsf{ff}\rangle \xrightarrow{\tau} \langle \mathsf{exit}, x{=}\mathsf{tt}\rangle$.

If distribution $\mu_D|_{\mathsf{exit}}$ is substochastic then the "missing probabilities" stand for non-termination inside the atomic region or for reaching a blocking state. Consider the follow-

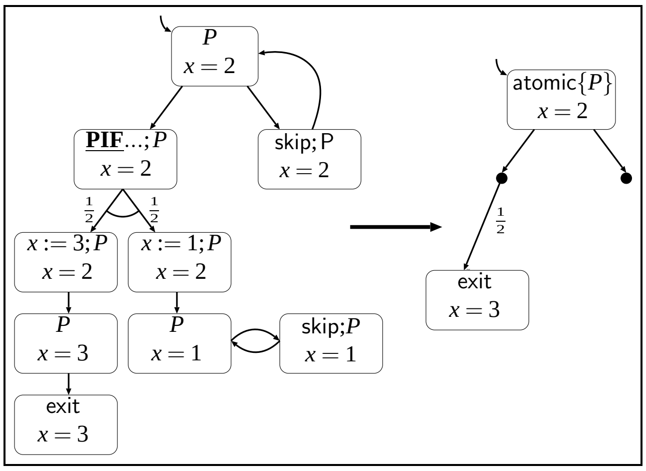


Figure 4.

[10]The execution of $\mathsf{atomic}\{P\}$ is blocked if P is forced to wait for the enabledness of a communication action or the satisfaction of a guard inside a conditional command.

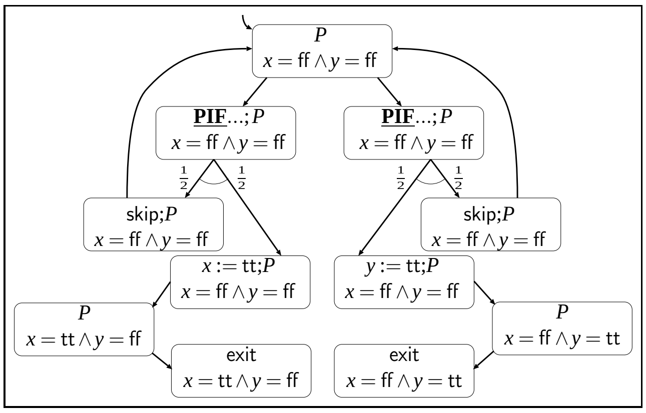
[11]The PROMELA-semantics, that suspends the execution of an atomic region if a blocked state is reached, can be obtained by replacing $\mu_D|_{\mathsf{exit}}$ with μ_D in our rule.

ing process P with integer variable x where initially $x=2$.

$$\begin{array}{ll} \underline{\textbf{DO}} & :: x = 2 \Rightarrow \underline{\textbf{PIF}}[0.5] \Rightarrow x := 1[0.5] \Rightarrow x := 3\underline{\textbf{FIP}} \\ & :: x \le 2 \Rightarrow \mathsf{skip} \\ \underline{\textbf{OD}} & \end{array}$$

See Fig. 4. Under each scheduler D which, for state $\langle P, x=2\rangle$, eventually chooses the first alternative, P terminates with probability 0.5 in state $\langle \mathsf{exit}, x=3\rangle$ and loops forever with probability 0.5. Thus, $\langle \mathsf{atomic}\{P\}, x=2\rangle \xrightarrow{\tau} \mu$ where $\mu(\langle \mathsf{exit}, x=3\rangle) = 0.5$ and $\mu(u) = 0$ for any other state u. Under the scheduler D_∞, which always chooses the second alternative, P does not terminate. We obtain $\langle \mathsf{atomic}\{P\}, x=2\rangle \xrightarrow{\tau} 0$ which provides the information that there is a computation with livelock probability 1. In a similar way, our semantics allows for reasoning about deadlock probabilities caused by reaching a state where the computation blocks.

The rule for atomic regions, applied to processes with loops, nondeterminism and probabilism, can lead to an *infinitely branching* MDP, even for finite-state processes.[12]



P
$x = \mathsf{ff} \wedge y = \mathsf{ff}$
PIF...;P
$x = \mathsf{ff} \wedge y = \mathsf{ff}$
PIF...;P
$x = \mathsf{ff} \wedge y = \mathsf{ff}$
$\frac{1}{2}$
$\frac{1}{2}$
$\frac{1}{2}$
$\frac{1}{2}$
skip;P
$x = \mathsf{ff} \wedge y = \mathsf{ff}$
skip;P
$x = \mathsf{ff} \wedge y = \mathsf{ff}$
$x := \mathsf{tt}$;P
$x = \mathsf{ff} \wedge y = \mathsf{ff}$
$y := \mathsf{tt}$;P
$x = \mathsf{ff} \wedge y = \mathsf{ff}$
P
$x = \mathsf{tt} \wedge y = \mathsf{ff}$
P
$x = \mathsf{ff} \wedge y = \mathsf{tt}$
exit
$x = \mathsf{tt} \wedge y = \mathsf{ff}$
exit
$x = \mathsf{ff} \wedge y = \mathsf{tt}$

Figure 5.

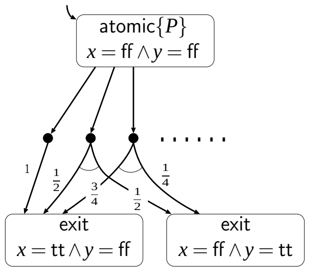
Regard the following process P with boolean variables x, y where initially $x=y=\mathsf{ff}$ (see Fig. 5).

$$\begin{array}{ll} \underline{\textbf{DO}} & :: \neg x \wedge \neg y \Rightarrow \underline{\textbf{PIF}}\ [0.5] \Rightarrow x := \mathsf{tt}\ [0.5] \Rightarrow \mathsf{skip}\ \underline{\textbf{FIP}} \\ & :: \neg x \wedge \neg y \Rightarrow \underline{\textbf{PIF}}\ [0.5] \Rightarrow y := \mathsf{tt}\ [0.5] \Rightarrow \mathsf{skip}\ \underline{\textbf{FIP}} \\ \underline{\textbf{OD}} & \end{array}$$

The two simple schedulers yield

$$\langle \mathsf{atomic}\{P\}, x=y=\mathsf{ff}\rangle \xrightarrow{\tau} \langle \mathsf{exit}, x=\mathsf{tt}, y=\mathsf{ff}\rangle,$$
$$\langle \mathsf{atomic}\{P\}, x=y=\mathsf{ff}\rangle \xrightarrow{\tau} \langle \mathsf{exit}, x=\mathsf{ff}, y=\mathsf{tt}\rangle.$$

We now consider scheduler D_n which chooses the first alternative for the first n-times state $\langle P, x=y=\mathsf{ff}\rangle$ is visited. From the $(n+1)$-st visit of state $\langle P, x=y=\mathsf{ff}\rangle$ on, D_n selects the second alternative. Using scheduler D_n, process P terminates with probability $\frac{1}{2} + \frac{1}{4} + \ldots + (\frac{1}{2})^n = 1 - (\frac{1}{2})^n$ in state $\langle \mathsf{exit}, x = \mathsf{tt}, y = \mathsf{ff}\rangle$ and with probability $(\frac{1}{2})^n$ in $\langle \mathsf{exit}, x = \mathsf{ff}, y = \mathsf{tt}\rangle$. Hence, there are infinitely many distributions μ with $\langle \mathsf{atomic}\{P\}, x=y=\mathsf{ff}\rangle \xrightarrow{\tau} \mu$.



atomic{P}
$x = \mathsf{ff} \wedge y = \mathsf{ff}$
......
1
$\frac{1}{2}$
$\frac{3}{4}$
$\frac{1}{2}$
$\frac{1}{4}$
exit
$x = \mathsf{tt} \wedge y = \mathsf{ff}$
exit
$x = \mathsf{ff} \wedge y = \mathsf{tt}$

For the automated model construction, the question for an alternative rule for atomic regions arises that yields a *finite representation* of the (possibly infinite) set

$$\begin{aligned} \mathsf{Steps}_\tau(\langle \mathsf{atomic}\{P\}, \gamma\rangle) &= \{\mu : \langle \mathsf{atomic}\{P\}, \gamma\rangle \xrightarrow{\tau} \mu\} \\ &= \left\{\mu_D|_{\mathsf{exit}} : D \text{ scheduler for } \mathcal{M}_{\langle P,\gamma\rangle}\right\} \end{aligned}$$

for finite-state processes P, but still covers all "relevant" information. We argue that for most interesting properties, the representation of a *finite basis* of $\mathsf{Steps}_\tau(\ldots)$ suffices. Here, a finite basis of $C \subseteq \mathsf{Distr}(S)$ means a finite subset B of C such that all distributions in C are *convex combinations* of the distributions in B. A formal argument that justifies the switch from $\mathsf{Steps}_\tau(\ldots)$ to a finite basis is that any MDP $\mathcal{M}$ is (strongly) *bisimilar* (in the sense of [42]) to any MDP $\mathcal{M}_{fin}$ with the same state space and where, for any state s and action a, the set of a-steps for s in $\mathcal{M}_{fin}$ is a finite basis of $\mathsf{Steps}_a^{\mathcal{M}}(s)$. Moreover, $\mathcal{M}$ and $\mathcal{M}_{fin}$ satisfy the same LTL and PCTL* formulas [7].

Theorem 2. *If P is finite-state then $\left\{\mu_E|_{\mathsf{exit}} : E \text{ is a simple scheduler for } \mathcal{M}_{\langle P,\gamma\rangle}\right\}$ is a finite basis for* $\mathsf{Steps}_\tau(\langle \mathsf{atomic}\{P\}, \gamma\rangle)$. [13]

A proof sketch is provided in the appendix. Thus, for finite-state programs, the consideration of the simple schedulers in the rule for atomic regions suffices and yields a finite MDP that is bisimilar to the MDP obtained by considering all schedulers. The same result can be established for the await-statement [37] that models *conditional atomic regions*. That is, for finite-state programs we may work with the following rule that yields a finite MDP:

$$\frac{\eta \models \mathsf{bexpr}, \quad E \text{ simpler scheduler for } \mathcal{M}_{\langle P,\gamma\rangle}}{\langle \underline{\textbf{AWAIT}}\ \mathsf{bexpr}\ \underline{\textbf{THEN}}\ P\ \underline{\textbf{END}}, \gamma\rangle \xrightarrow{\tau} \mu_E|_{\mathsf{exit}}}$$

Denotational approach. We conclude this section by providing a *compositional* characterization of our operational

[12] By a *finite-state process*, we mean a process P where the domains of its variables and channels and the capacities of all its fifo channels are finite.

[13] Note that the simple schedulers for $\mathcal{M} = \mathcal{M}_{\langle P,\gamma\rangle}$ may choose different alternatives for states $\langle R, \gamma_1\rangle$ and $\langle R, \gamma_2\rangle$ with the same control component R but distinct evaluations γ_1, γ_2; otherwise, the basis-property of the simple schedulers would not hold.

semantics for $\mathsf{atomic}\{P\}$ by means of a denotational semantics for the body P. Let Γ denote the set of variable-channel-evaluation pairs $\gamma = \langle \eta, \xi \rangle$. The denotation of P can be defined as a function $\mathcal{D}[\![P]\!] : \Gamma \to 2^{\mathsf{Distr}(\Gamma)}$ that assigns to any "input configuration" γ a set of distributions for the possible "output configurations". [14, 16] presents such a denotational semantics for a language that essentially agrees with our basis language (Sect. 3) without nondeterminism. In the approach of [14, 16], $\mathcal{D}[\![P]\!](\gamma)$ returns a unique exit-distribution μ that describes the probabilistic effect of executing P with the initial variable evaluation γ and that agrees with μ_D for the unique scheduler D for $\mathcal{M}_{\langle P,\gamma\rangle}$.

To define the denotations of processes with nondeterminism, probabilism and loops, $\mathcal{D}[\![P]\!]$ can be extended to a function $\mathcal{D}[\![P]\!] : \mathcal{P}(\mathsf{Distr}(\Gamma)) \to \mathcal{P}(\mathsf{Distr}(\Gamma))$, where $\mathcal{P}(\cdot)$ is a suitable powerdomain that allows to define the denotation of a DO-OD–process by a fixed point construction. The recent work [12] provides such a denotational semantics for a language that essentially agrees with our basis language for processes using the Egli-Milner ordering (see e.g. [2]). For our purposes, the semantics of [12] is too abstract as $\mathcal{D}[\![P]\!](\gamma)$ might be a proper superset of the convex hull of $\mathsf{Steps}_\tau(\langle \mathsf{atomic}\{P\}, \gamma\rangle)$, and thus, identifies some non-bisimilar processes.[14]

It is not clear to us whether any other topological approach for the definition of a denotational semantics (e.g. in the style of [13, 6] where metric denotational characterizations of bisimulation are provided for data-abstract probabilistic systems) allows for precise reasoning about the convex hull of the exit-distributions induced by the schedulers. However, we may follow another approach which relies on the idea that the denotation of a probabilistic process P is viewed as a function $E \mapsto \mathcal{D}_E[\![P]\!]$ that assigns to any scheduler E a function $\mathcal{D}_E[\![P]\!] : \Gamma \to \mathsf{Distr}(\Gamma)$ formalizing the input/output behavior of P under scheduler E. We shrink our attention to finite-state processes and simple schedulers and define $\mathcal{D}_E[\![P]\!]$ by structural induction. We put $\mathcal{D}_E[\![\mathsf{skip}]\!](\gamma) = \mu^1_\gamma$ and

$$\begin{aligned}\mathcal{D}_E[\![x := \mathsf{expr}]\!](\eta,\xi) &= \mu^1_{\langle \eta[x:=\mathsf{expr}],\xi\rangle},\\ \mathcal{D}_E[\![x := \mathsf{random}(V)]\!](\eta,\xi) &= \sum_{v\in V} \frac{1}{|V|}\mu^1_{\langle \eta[x:=v],\xi\rangle}.\end{aligned}$$

For successful asynchronous communication, we put

$$\mathcal{D}_E[\![f?x]\!](\eta,\xi) = \mu^1_{\langle \eta[x:=\mathsf{first}(\xi,f)],\xi[\mathsf{remove}(f)]\rangle}$$

if $|\xi(f)| > 0$. For $|\xi(f)| < \mathsf{cap}(f)$ and $v = [\![\mathsf{expr}]\!]_\eta$, we define $\mathcal{D}_E[\![f!\mathsf{expr}]\!](\eta,\xi)$ to be

$$(1-p(f))\mu^1_{\langle \mathsf{exit},\eta,\xi[\mathsf{add}(f,v)]\rangle} + p(f)\mu^1_{\langle \mathsf{exit},\eta,\xi\rangle}$$

As we interpret the blocking inside an atomic region caused by a synchronous or disabled asynchronous communication request as a livelock, we put $\mathcal{D}_E[\![c?x]\!](\gamma) = \mathcal{D}_E[\![f?x]\!](\eta,\xi) = 0$ if $|\xi(f)| = 0$, and $\mathcal{D}_E[\![c!\mathsf{expr}]\!](\gamma) = \mathcal{D}_E[\![f!\mathsf{expr}]\!](\gamma) = 0$ if $|\xi(f)| = \mathsf{cap}(f)$. For sequential composition and probabilistic choice, we put:

$$\begin{aligned}\mathcal{D}_E[\![P_1;P_2]\!](\gamma) &= \sum_{\gamma_1\in\Gamma} \mathcal{D}_E[\![P_1]\!](\gamma)(\gamma_1)\cdot \mathcal{D}_E[\![P_2]\!](\gamma_1)\\ \mathcal{D}_E[\![\underline{\mathbf{PIF}}\ [p_1] \Rightarrow P_1 \ldots [p_n] \Rightarrow P_n\ \underline{\mathbf{FIP}}]\!](\gamma) &\\ &= p_1\mathcal{D}_E[\![P_1]\!](\gamma) + \ldots + p_n\mathcal{D}_E[\![P_n]\!](\gamma)\end{aligned}$$

For resolving the nondeterministic choices in conditional or repetitive commands, we use the fixed scheduler E. Let $P = \underline{\mathbf{IF}}\ ::g_1 \Rightarrow P_1 \ldots ::g_n \Rightarrow P_n\ \underline{\mathbf{FI}}$ and $\gamma = \langle \eta,\xi\rangle \in \Gamma$. If $\langle P,\gamma\rangle$ is a terminal state w.r.t. the transition relation $\xrightarrow{\tau} \cup \xrightarrow{\mathsf{else}}$ then $\mathcal{D}_E[\![P]\!](\gamma) = 0$. If E chooses the guarded command $g_i \Rightarrow P_i$ for state $\langle P,\gamma\rangle$ then we put:[15]

- If g_i is a boolean guard or $g_i = \underline{\mathbf{ELSE}}$ then $\mathcal{D}_E[\![P]\!](\gamma) = \mathcal{D}_E[\![P_i]\!](\gamma)$.
- If $g_i = \mathsf{bexpr} \wedge f?x$ then $\mathcal{D}_E[\![P]\!](\eta,\xi)$ is
$$\mathcal{D}_E[\![P_i]\!](\eta[x := \mathsf{first}(\xi,f)], \xi[\mathsf{remove}(f)]).$$
- If $g_i = \mathsf{bexpr} \wedge f!\mathsf{expr}$, $v = [\![\mathsf{expr}]\!]_\eta$ then $\mathcal{D}_E[\![P]\!](\eta,\xi)$ is
$$(1-p(f))\,\mathcal{D}_E[\![P_i]\!](\eta,\xi[\mathsf{add}(f,v)]) + p(f)\mathcal{D}_E[\![P_i]\!](\eta,\xi).$$

For the loop $P = \underline{\mathbf{DO}}\ ::g_i \Rightarrow P_1 \ldots ::g_n \Rightarrow P_n\ \underline{\mathbf{OD}}$ we use the operator $\Omega = \Omega_{P,E} : \mathsf{Distr}(\Gamma) \to \mathsf{Distr}(\Gamma)$ which, for $\mu \in \mathsf{Distr}(\Gamma)$, $\gamma\in\Gamma$, is defined by:

$$\Omega(\mu)(\gamma) = \sum_{\gamma_1\in\Gamma} \mathcal{D}_E[\![\ldots ::g_i \Rightarrow P_i \ldots]\!](\gamma)(\gamma_1)\cdot\mu(\gamma_1)$$

where $\mathcal{D}_E[\![\ldots ::g_i \Rightarrow P_i \ldots]\!](\gamma)$ describes the probabilistic effect of one iteration of P started with γ using the scheduling strategy E. Its formal definition is obtained essentially as for conditional commands:

$$\mathcal{D}_E[\![\ldots ::g_i \Rightarrow P_i \ldots]\!](\gamma) = \mathcal{D}_{E'}[\![P']\!](\gamma)$$

where P' and e' are as follows. P' is

$$\underline{\mathbf{IF}} ::g_1 \Rightarrow P_1 \ldots ::g_n \Rightarrow P_n :: \underline{\mathbf{ELSE}} \Rightarrow \mathsf{skip}\ \underline{\mathbf{FI}}$$

if $g_1,\ldots,g_n$ are boolean or generalized guards. If $g_i = \underline{\mathbf{ELSE}}$ for at least one index i then

$$P' = \underline{\mathbf{IF}}\ ::g_1 \Rightarrow P_1 \ldots ::g_n \Rightarrow P_n\ \underline{\mathbf{FI}}.$$

[14]The reason is that there are distinct convex sets of distributions that are identified by the Egli-Milner ordering, e.g. the triangles spanned by the points $\langle 0,0\rangle$, $\langle 0.5,0\rangle$, $\langle 0.5,0.5\rangle$ and $\langle 0,0\rangle$, $\langle 0,0.5\rangle$, $\langle 0.5,0.5\rangle$ where $\langle p,q\rangle$ is identified with the distribution assigning probability p to the evaluation "$x = \mathsf{ff}$" and probability q to the evaluation "$x = \mathsf{tt}$".

[15]As E chooses an outgoing transition of state $\langle P,\gamma\rangle$ with label τ or else, g_i cannot be a generalized guard with a synchronous communication request and we have $\gamma \models g_i$ if g_i is boolean or a generalized guard with an asynchronous communication request. If $g_i = \underline{\mathbf{ELSE}}$ then $\gamma \not\models g_j$ for all indices $j \in \{1,\ldots,n\}$ where g_j is a boolean or generalized guard.

$$\frac{\langle R_i, \gamma\rangle \xrightarrow{\mathsf{start}(P_1,\ldots,P_m)} \langle R, \gamma\rangle, \ \ R \neq \mathsf{exit}}{\langle R_1, \ldots, R_{i-1}, R_i, R_{i+1}, \ldots, R_m, \gamma\rangle \xrightarrow{\tau}_{\mathcal{Q}} \langle R_1 \ldots, R_{i-1}, R, P_1, \ldots, P_m, R_{i+1}, \ldots, R_m, \gamma\rangle}$$

$$\frac{\langle R_i, \gamma\rangle \xrightarrow{\mathsf{start}(P_1,\ldots,P_m)} \langle \mathsf{exit}, \gamma\rangle}{\langle R_1, \ldots, R_{i-1}, R_i, R_{i+1}, \ldots, R_m, \gamma\rangle \xrightarrow{\tau}_{\mathcal{Q}} \langle R_1, \ldots, R_{i-1}, P_1, \ldots, P_m, R_{i+1}, \ldots, R_m, \gamma\rangle}$$

Figure 6.

If there are several possibilities then scheduler E' chooses for $\langle P', \gamma\rangle$ exactly the same guarded command as E for $\langle P, \gamma\rangle$. E' and E agree for the P_i's and their subprocesses. It is clear that $\varOmega$ is continuous as an operator on $\mathsf{Distr}(\varGamma)$ viewed as a complete partial order with the pointwise ordering for distributions. Thus, we may define $\mathcal{D}_E[\![P]\!](\gamma) = \mathsf{lfp}(\varOmega)(\mu^1_\gamma)$ where $\mathsf{lfp}(\varOmega)$ denotes the least fixed point of $\varOmega$. Using structural induction it can be shown that $\mathcal{D}_E[\![P]\!](\gamma) = \mu_E|_{\mathsf{exit}}$. Theorem 2 yields:

Theorem 3. *If P is finite-state then* $\{\mathcal{D}_E[\![P]\!](\gamma) : E$ simple scheduler for $\mathcal{M}_{\langle P,\gamma\rangle}\}$ *is a finite basis for* $\mathsf{Steps}_\tau(\langle \mathsf{atomic}\{P\}, \gamma\rangle) = \{\mu : \langle \mathsf{atomic}\{P\}, \gamma\rangle \xrightarrow{\tau} \mu\}$.

6 Nested parallelism

To keep the presentation of the paper simple we only presented the basis concepts of PROBMELA, but skipped additional features such as arrays, procedure calls, pointers or parallelism inside processes. A detailed description of these additional concepts will be explained at another occasion. Here, we will only sketch the ideas of how to handle *nested parallelism*. Parallelism inside a process is realized by a command $\mathsf{run}(P_1, \ldots, P_m)$ that starts the parallel execution of the (sub-)processes $P_1, \ldots, P_m$. To treat dynamic process creation via the run-command on the semantic level, the states in the MDP $\mathcal{M}_\mathcal{Q}$ for program $\mathcal{Q}$ have the form $\langle \varLambda, \gamma\rangle$ where $\varLambda$ is a finite sequence consisting of (control components for) the active processes together with an evaluation for their local variables and channels and γ an evaluation for the global variables and channels. For simplicity, let us assume that all variables and channels are global. Then, we may work with the axiom

$$\langle \mathsf{run}(P_1, \ldots, P_m), \gamma\rangle \xrightarrow{\mathsf{start}(P_1,\ldots,P_m)} \langle \mathsf{exit}, \gamma\rangle$$

on the process-level and with the rules shown in Fig. 6 on the program-level. The interleaving and synchronization rule have to be modified to remove successfully terminated processes from the list of active processes. Atomic regions can be handled essentially as before, the main difference being that we have to allow for handshaking of two subprocesses inside the atomic region. Finite-state MDPs are guaranteed when – in addition to the requirement that only variables and bounded fifo channels with finite domains are used – the run-command does not occur inside loops.

7 Conclusion

Our starting point was a higher level specification language for parallel programs whose processes are described by a guarded command language with nondeterminism, probabilism, repetition, message passing via synchronous and (possibly lossy) fifo channels and atomic regions. We provide an operational semantics by means of SOS-rules that yield a MDP formalizing the stepwise behavior of a given program. Although finite-state programs with atomic regions may have an infinitely branching MDP, a bisimilar finite MDP can be obtained in a compositional way using a denotational semantics.

The presented work is the first step towards the implementation of a probabilistic model checker that verifies probabilistic programs (of our language PROBMELA) against LTL-formulas. We used the SOS-rules as basis for an implementation which generates the MDP of a given finite-state program. For the internal program representation, a low level language similar to JAVAs virtual machine bytecode is used. Given a PROBMELA-program $\mathcal{Q}$, we first transform $\mathcal{Q}$ into an equivalent bytecode representation from which the reachable fragment of the state space is generated using a traversal technique that relies on a mixture between DFS (to treat nondeterministic branching) and BFS (for probabilistic branching). For a compact internal representation of the states we adapt the rubber vector technique [24] that allows shrinking and extending the vector dynamically as needed when modelling dynamic process creation and destruction as well as the use of temporary variables (not described in this paper).

The current state of our implementation yields the possibility for simulating a given PROBMELA-program where the nondeterministic (and probabilistic) choices are resolved either by user input or automatically. The implementation of a quantitative reachability analysis (which calculates minimal or maximal probabilities to reach a given set of states under all schedulers) will be finished in near future. Our goal is the implementation of an automata-based LTL-model checker for PROBMELA-programs. To obtain a probabilistic analogue to the well-known model checker SPIN, several problems have to be solved in future work, such as the development of algorithms for partial order reduction in MDPs and heuristics to derive "promising" reductions from the PROBMELA-descriptions of the processes, theoretical and experimental results about randomized memory-management (e.g. bit-state hashing) or abstraction techniques for PROBMELA-programs.

References

[1] P. Abdulla, C. Baier, P. Iyer, B. Jonsson. Reasoning about probabilistic lossy channel systems. In *Proc. CONCUR'00*, LNCS 1877:320–330, 2000.

[2] S. Abramsky, A. Jung. Domain theory. In S. Abramsky, D. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science Volume 3*, pages 1–168. Oxford University Press, 1994.

[3] R. Alur, T. Henzinger. Reactive modules. *Formal Methods in System Design: An International Journal*, 15(1):7–48, 1999.

[4] K. Apt. Correctness proofs of distributed termination algorithms. *ACM Transact.Prog.Lang.Syst.*, 8:388–405, 1986.

[5] K. Apt, E.-Olderog. *Verification of sequential and concurrent programs*. Springer Verlag, 1991.

[6] C. Baier, M. Kwiatkowska. Domain equations for probabilistic processes. *Mathematical Structures in Computer Science*, 10(6):665–717, 2000.

[7] A. Bianco, L. de Alfaro. Model checking of probabilistic and nondeterministic systems. *LNCS*, 1026:499–513, 1995.

[8] H. Bohnenkamp, H. Hermanns, M. Zhang, F. Vaandrager. Cost-Optimization of the IPv4 Zeroconf Protocol. *Proc. of the 3rd Progress Workshop on embedded systems*, 2002.

[9] S. Cheshire, B. Adoba. Dynamic configuration of IPv4 link-local addresses. Internet-draft, 2001. `http://www.ietf.org/internet-drafts/draft-ietf-zeroconf-ipv4-linklocal-04.txt`

[10] C. Courcoubetis, M. Yannakakis. The complexity of probabilistic verification. *J. of the ACM*, 42(4):857–907, 1995.

[11] P. d'Argenio, H. Hermanns, J. Katoen, R. Klaren. Modest - a modelling and description language for stochastic timed systems. *Proc. PAPM/PROBMIV*, LNCS 2165:87–104, 2001.

[12] P. d'Argenio, N. Wolovick. Verification of probabilistic and nondeterministic systems, 2004. forthcoming paper.

[13] E. de Vink, J. Rutten. Bisimulation for probabilistic transition systems: A coalgebraic approach. *Proc. ICALP'97, LNCS*, 1256:460–470, 1997.

[14] J. den Hartog. *Probabilistic Extensios of Semantical Models*. PhD thesis, Universiteit Amsterdam, 2002.

[15] J. den Hartog, E. de Vink. Mixing up nondeterminism and probability. *Proc. PROBMIV' 98, ENTCS*, Vol. 21, 1999.

[16] J. den Hartog, E. de Vink. Verifying probabilistic programs using a hoare like logic. In *Asian Computing Science Conference*, pages 113–125, 1999.

[17] E. Dijkstra. Guarded commands, non-determinacy and the formal derivation of programs. *Comm. ACM*, 18:453–457, 1975.

[18] A. Giacalone, C. Jou, S. Smolka. Algebraic reasoning for probabilistic concurrent systems. *Proc. IFIP TC2 Working Conf. on Programming Concepts and Methods*, 1990.

[19] H. Hansson. *Time and Probability in Formal Design of Distributed Systems*. Series in Real-Time Safety Critical Systems. Elsevier, 1994.

[20] H. Hansson, B. Jonsson. A calculus for communicating systems with time and probabitilies. In I. C. S. Press, editor, *Proc. of the Real-Time Systems Symposium - 1990*, pages 278–287, 1990. IEEE Computer Society Press.

[21] H. Hansson, B. Jonsson. A logic for reasoning about time and reliability. *Formal Asp. of Comp.*, 6(5):512–535, 1994.

[22] V. Hartonas-Garmhausen, S. Campos, E. Clarke. ProbVerus: Probabilistic symbolic model checking. *Proc. ARTS'99 LNCS*, 1601:96–110, 1999.

[23] J. He, K. Seidel, A. McIver. Probabilistic models for the guarded command language. *Science of Computer Programming*, 28(2-3):171–192, 1997.

[24] G. J. Holzmann. The model checker SPIN. *Software Engineering*, 23(5):279–295, 1997.

[25] B. Jeannet, P. D'Argenio, K. Larsen. RAPTURE: A tool for verifying Markov Decision Processes. In *Tools Day'02*, Technical Report. Masaryk University Brno, 2002.

[26] C. Jones. *Probabilistic Non-Determinism*. PhD thesis, University of Edinburgh, 1990.

[27] B. Jonsson, W. Yi, K. G. Larsen. Probabilistic extensions of process algebras. In *Handbook of Process Algebra*, pages 685–710. Elsevier, 2001.

[28] M. Kwiatkowska, G. Norman. Probabilistic metric semantics for a simple language with recursion. *Proc. MFCS'96*, LNCS 1113:419–430, 1996.

[29] M. Kwiatkowska, G. Norman, D. Parker. PRISM: Probabilistic symbolic model checker. In *Comp. Performance Eval. TOOLS*, pages 200–204, 2002.

[30] L. Lamport. Proving the correctness of multiprocess programs. *IEEE Trans. on Softw. Eng.*, SE-3:2:125–143, 1977.

[31] K. G. Larsen, A. Skou. Compositional verification of probabilistic processes. *Proc. CONCUR*, LNCS 630:456–471, 1992.

[32] R. Lipton. Reduction: a method of proving properties of parallel programs. *Comm. ACM*, 18:717–721, 1975.

[33] G. Lowe. Probabilistic and prioritized models of timed CSP. *Theoretical Computer Science*, 138(2):315–352, 1995.

[34] C. Morgan, A. McIver. pgcl: formal reasoning for random algorithms. *Proc. WOFACS'98, Spec. Iss.of SACJ*, 22:14–27, 1999.

[35] C. Morgan, A. McIver, K. Seidel. Probabilistic predicate transformers. *ACM Trans. on Progr. Lang. and Systems*, 18(3):325–353, 1996.

[36] S. Owicki. Verifying concurrent programs with shared data classes. *Formal Descriptions of Programming Concepts*, pages 279–299, 1978.

[37] S. Owicki, D. Gries. An axiomatic proof technique for parallel programs I. *Acta Informatica*, 6(4):319–340, 1976.

[38] G. D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Computer Science Department, Aarhus University, 1981.

[39] G. L. Plotkin. An operational semantics for csp. In *Proc. of the IFIP Working Group Conference on Programming Concepts II*, pages 199–225, 1982.

[40] A. Pnueli, L. D. Zuck. Probabilistic verification. *Information and Computation*, 103(1):1–29, Mar. 1993.

[41] M. Puterman. *Markov Decision Processes-Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 1994.

[42] R. Segala, N. Lynch. Probabilistic simulations for probabilistic processes. *Nordic J. of Comp.*, 2(2):250–273, 1995.

[43] C. Tofts. A synchronous calculus of relative frequency. *Proc. CONCUR'90, LNCS*, 458:467–480, 1990.

[44] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. *Proc. LICS'86*, pages 332–344, 1986.

A Proof sketch for Theorem 1

We first observe that it suffices to show that if P is finite-state then

$$\{\mu_E : E \text{ is a simple scheduler for } \mathcal{M}_{\langle P,\gamma\rangle}\}$$

is a finite basis for $\{\mu_D : D \text{ is a scheduler for } \mathcal{M}_{\langle P,\gamma\rangle}\}$. For this, we establish the basis property of the simple schedulers for arbitrary finite MDPs.

In the sequel, let $\mathcal{M} = (S, \mathsf{Act}, \longrightarrow, \{s_0\})$ be a finite MDP with single initial state s_0. We write T to denote the set of terminal states in $\mathcal{M}$. To simply our argumentation, we assume that for any terminal state $t \in T$ there is exactly one triple $\langle s, a, \mu\rangle$ with $s \in S$, $\langle a, \mu\rangle \in \mathsf{Steps}(s)$ and $t \in \mathsf{Supp}(\mu)$.[16] Given a scheduler D for $\mathcal{M}$, we write μ_D to denote the distribution

$$\mu_D(t) = \sum\{\mathsf{Pr}(\sigma) : \sigma \text{ is a } D\text{-path with } \mathsf{last}(\sigma) = t\}$$

for any $t \in T$ and $\mu_D(s) = 0$ for any state $s \in S \setminus T$.

Theorem 4. $\{\mu_E : E \text{ is a simple scheduler for } \mathcal{M}\}$ *is a finite basis for* $\{\mu_D : D \text{ is a scheduler for } \mathcal{M}\}$.

Proof. Only the case $s_0 \notin T$ is of interest. If $\mathsf{Supp}(\mu_D) = \emptyset$ (i.e., $\mu_D(t) = 0$ for all states $t \in T$) then we choose an arbitrary simple scheduler E such that $E(s) = D(\sigma)$ for some D-path σ with $\mathsf{last}(\sigma) = s$. We then have $\mathsf{Supp}(\mu_E) = \emptyset$, and hence, $\mu_D = \mu_E$. Thus, we may define $p_E = 1$ and $p_F = 0$ for any other simple scheduler F and obtain $\mu_D = \sum_F p_F \mu_F$.

In the sequel, we assume that $\mathsf{Supp}(\mu_D) \neq \emptyset$. We start with $\mu_0 = \mu_D$ and choose a simple scheduler E_1 for $\mathcal{M}$ such that

(1.1) $\mathsf{Supp}(\mu_{E_1}) \subseteq \mathsf{Supp}(\mu_0)$

(1.2) E_1 is maximal w.r.t. condition (1.1) in the sense that there is no simple scheduler E for $\mathcal{M}$ such that $\mathsf{Supp}(\mu_E) \subseteq \mathsf{Supp}(\mu_0)$ and $\mathsf{Supp}(\mu_{E_1})$ is a proper subset of $\mathsf{Supp}(\mu_E)$.

We then put

$$p_1 = \min\left\{\frac{\mu_D(t)}{\mu_{E_1}(t)} : t \in \mathsf{Supp}(\mu_{E_1})\right\}$$

and $\mu_1 = \mu_0 - p_1\mu_{E_1}$.

If $\mu_1 \neq 0$ then we may choose a simple scheduler E_2 for $\mathcal{M}$ such that (2.1) $\mathsf{Supp}(\mu_{E_2}) \subseteq \mathsf{Supp}(\mu_1)$ and (2.2) E_2 is maximal w.r.t. condition (2.1) in the sense that there is no simple scheduler E for $\mathcal{M}$ such that $\mathsf{Supp}(\mu_E) \subseteq \mathsf{Supp}(\mu_1)$ and $\mathsf{Supp}(\mu_{E_2})$ is a proper subset of $\mathsf{Supp}(\mu_E)$. We then define

$$p_2 = \min\{\tfrac{\mu_1(t)}{\mu_{E_2}(t)} : t \in \mathsf{Supp}(\mu_{E_2})\}$$

and $\mu_2 = \mu_1 - p_2\mu_{E_2}$.

Proceeding in this way we obtain a sequence of triples (E_j, p_j, μ_j) where $p_j > 0$ and E_j is a simple scheduler for $\mathcal{M}$ such that (j.1) $\mathsf{Supp}(\mu_{E_j}) \subseteq \mathsf{Supp}(\mu_{j-1})$ and (j.2) E_j is maximal w.r.t. condition (j.1). Distribution μ_j is given by

$$\mu_j = \mu_D - (p_1\mu_{E_1} + \ldots + p_{j-1}\mu_{E_{j-1}}).$$

Moreover, $\mathsf{Supp}(\mu_j)$ is a proper subset of $\mathsf{Supp}(\mu_{j-1})$. As S is finite there exists an index J with $\mathsf{Supp}(\mu_J) = \emptyset$, i.e., $\mu_J = 0$, and therefore,

$$\mu_D = p_1\mu_{E_1} + \ldots + p_{J-1}\mu_{E_{J-1}}.$$

It remains to show that (1) $p_j \leq 1$ for all $1 \leq j < J$ and $p_1 + \ldots + p_{J-1} = 1$ and (2) the existence of simple schedulers E_j with $\mathsf{Supp}(\mu_{E_j}) \subseteq \mathsf{Supp}(\mu_{j-1})$. We skip the proofs for these claims and just mention that for the existence of such simple schedulers E_j the maximality condition (j-1.2) for E_{j-1} is important. For instance, for the MDP $\mathcal{M} = (S, \{a,b,c\}, \longrightarrow, \{s_0\})$ with $S = \{s_0, w, t, t'\}$ and $T = \{t, t'\}$ and

- $\mathsf{Steps}(s_0) = \{\langle a, \mu\rangle\}$ where $\mu = [t \mapsto \frac{1}{2}, w \mapsto \frac{1}{2}]$,
- $\mathsf{Steps}(w) = \{\langle b, \mu^1_{s_0}\rangle, \langle c, \mu^1_{t'}\rangle\}$

and scheduler D with $D(s_0, a, \mu, w) = \langle c, \mu^1_{s_0}\rangle$ and $D(s_0, a, \mu, w, c, \mu^1_{s_0}, \ldots, a, \mu, w) = \langle b, \mu^1_{t'}\rangle$, we have

$$\begin{aligned} \mu_D &= [t \mapsto \tfrac{3}{4}, t' \mapsto \tfrac{1}{4}] \\ &= \tfrac{1}{2}\underbrace{[t \mapsto \tfrac{1}{2}, t' \mapsto \tfrac{1}{2}]}_{\mu_{E_b}} + \tfrac{1}{2}\underbrace{[t \mapsto 1]}_{\mu_{E_c}} \\ &= \tfrac{1}{2}\mu_{E_b} + \tfrac{1}{2}\mu_{E_c} \end{aligned}$$

where E_b, E_c are the two simple schedulers that choose action b resp. action c for state w. When choosing $E_1 = E_c$ in the above procedure we would obtain

$$\begin{aligned} p_1 &= \tfrac{\mu_D(t)}{\mu_{E_c}(t)} = \tfrac{\frac{3}{4}}{1} = \tfrac{3}{4}, \\ \mu_1 &= \mu_D - \tfrac{3}{4}\mu_{E_c} = [t' \mapsto \tfrac{1}{4}]. \end{aligned}$$

But then, there is no simple scheduler E with $\mathsf{Supp}(\mu_E) \subseteq \mathsf{Supp}(\mu_1) = \{t'\}$. In fact, E_c violates the maximality condition (1.2). However, choosing $E_1 = E_b$ we get

$$\begin{aligned} p_1 &= \min\{\tfrac{\mu_D(t)}{\mu_{E_b}(t)}, \tfrac{\mu_D(t')}{\mu_{E_b}(t')}\} = \tfrac{1}{2}, \\ \mu_1 &= \mu_D - \tfrac{1}{2}\mu_{E_b} = [t \mapsto \tfrac{1}{2}], \\ T_0 &= \{t\}. \end{aligned}$$

We then may define $E_2 = E_c$, $p_2 = \frac{1}{2}$ and obtain $\mu_2 = \mu_1 - \frac{1}{2}\mu_{E_c} = 0$. ■

[16] If this assumption does not hold for $\mathcal{M}$ then we may deal with copies $t_{s,a,\mu}$ for each of the terminal states t in $\mathcal{M}$. The values $\mu_D(t)$ of the output distribution μ_D are then obtained by summing up the values $\mu_D(t_{s,a,\mu})$.