Norman, G., Palamidessi, C., Parker, D. and Wu, P. (2009) *Model checking probabilistic and stochastic extensions of the pi-calculus.* IEEE Transactions on Software Engineering 35 (2-3). pp. 209-223. ISSN 0098-5589

Model Checking Probabilistic and Stochastic Extensions of the π-Calculus

Gethin Norman, Catuscia Palamidessi, David Parker, and Peng Wu

Abstract—We present an implementation of model checking for probabilistic and stochastic extensions of the π-calculus, a process algebra which supports modeling of concurrency and mobility. Formal verification techniques for such extensions have clear applications in several domains, including mobile ad hoc network protocols, probabilistic security protocols, and biological pathways. Despite this, no implementation of automated verification exists. Building upon the π-calculus model checker Mobility Model Checker (MMC), we first show an automated procedure for constructing the underlying semantic model of a probabilistic or stochastic π-calculus process. This can then be verified using existing probabilistic model checkers such as PRISM. Secondly, we demonstrate how, for processes of a specific structure, a more efficient, compositional approach is applicable, which uses our extension of MMC on each parallel component of the system and then translates the results into a high-level modular description for the PRISM tool. The feasibility of our techniques is demonstrated through a number of case studies from the π-calculus literature.

Index Terms—Verification, model checking, Markov processes, stochastic processes.

1 Introduction

The π-calculus [1] is a process algebra for modeling concurrency and mobility. It has been used to model, for example, communication protocols for dynamic network topologies, security protocols, and biological pathways. For each class of systems, probabilistic and stochastic behaviors are often also key ingredients. Mobile ad hoc network protocols, for example, can exhibit probabilistic behavior through either communication failures or random back off procedures. Similarly, randomization is frequently applied in security protocols, e.g., for anonymity [2] or contract-signing [3]. For biological systems, the times between reactions are of a stochastic nature.

Consequently, suitable variants of the π-calculus have been developed: probabilistic versions, for example [4], which extend the original calculus with discrete probabilistic choice, have been proposed as a formalism to model and reason about randomized security protocols [5], [6]; and stochastic extensions, for example [7], which augment the calculus with exponential delays, have been shown to be a suitable formalism for modeling and reasoning about complex biological pathways [8], [9].

The benefits of automatic formal verification and tool support in this context are clear: reasoning correctly about the behavior of such models, particularly interactions between probabilistic and nondeterministic behavior, is known to be nontrivial. Furthermore, the state spaces of probabilistic or stochastic models of realistic systems have a tendency to grow extremely quickly, making manual verification difficult or infeasible.

In this paper, we describe an implementation of probabilistic model checking for models described in two different extensions of the π-calculus. The first, the *simple probabilistic π-calculus*, is an extension of the π-calculus obtained by introducing a discrete probabilistic choice operator in addition to the existing nondeterministic choice operator. The second, the *stochastic π-calculus*, extends the original calculus by associating rates (parameters of exponential distributions) with both silent transitions and channels.

Our approach is to adapt and reuse existing tools for verification of mobile systems and of probabilistic and stochastic systems. We first developed an extension of the tool MMC [10], a logic-programming-based model checker for the π-calculus. This extension, MMC_{prob}, can derive the semantic model for an arbitrary process in the (finite-control) probabilistic or stochastic π-calculus. The semantic model, which is given by a Markov decision process (MDP) or continuous-time Markov chain (CTMC), can then be analyzed using standard tools, such as the probabilistic model checker PRISM [11]. To improve efficiency, when the process has a specific structure, we employ a compositional approach, applying MMC_{prob} to each parallel component of a system, processing the results to produce a high-level modular description in the modeling language of PRISM, and then performing probabilistic verification. This avoids a potential blowup in the size of the intermediate MDP or CTMC representation and allows us to exploit the efficient symbolic model construction and analysis techniques in PRISM. We present experimental results to illustrate the performance of our implementation on a number of case studies. To our knowledge, this paper constitutes the first attempt to implement *automated verification* in this area.

- *G. Norman and D. Parker are with the Oxford University Computing Laboratory, Wolfson Building, Parks Road, Oxford OX1 3QD, UK. E-mail: {gethin.norman, david.parker}@comlab.ox.ac.uk.*
- *C. Palamidessi is with INRIA Saclay and École Polytechnique, Rue de Saclay, 91128 Palaiseau, France. E-mail: catuscia@lix.polytechnique.fr.*
- *P. Wu is with the University College London, Adastral Park, Martlesham Heath, Ipswich IP5 3RE, UK. E-mail: p.wu@adastral.ucl.ac.uk.*

Manuscript received 25 Jan. 2008; revised 30 June 2008; accepted 20 Aug. 2008; published online 15 Sept. 2008.
Recommended for acceptance by J. Hillston, M. Kwiatkowska, and M. Telek.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSESI-2008-01-0027.
Digital Object Identifier 10.1109/TSE.2008.77.

Related work. Various tools exist for automatic verification of the (nonprobabilistic) π-calculus. The Mobility Workbench (MWB'99) [12] provides a bisimulation checker and a π-μ-calculus model checker. Mobility Model Checker (MMC) [10], a more recently developed tool, also supports the π-μ-calculus. The latter places particular emphasis on efficiency and is built using logic programming technology. ProVerif [13] supports verification of the applied π-calculus, a variant of the basic calculus. It is aimed primarily at analysis of cryptographic protocols and is theorem-prover based. Two alternative approaches are the PIPER system [14], which verifies π-calculus processes augmented with type signatures based on an extraction of sound models using types and CCS processes [15], [16], and which translate a subset of the π-calculus to the language Promela for model checking in the SPIN tool. Static analysis techniques have also been applied to the π-calculus, including abstract interpretation [17] and control flow analysis [18].

A number of existing papers have proposed probabilistic extensions of the π-calculus. The first [4] extended the asynchronous version of the calculus, which removes the output prefix construct, meaning processes must terminate immediately after sending output. A version was then proposed in [5], considering only silent probabilistic transitions. This variant, which is essentially the same as the one used in this paper, was introduced to specify and reason about randomized security protocols. In [6], the probabilistic π-calculus was used to formalize definitions of anonymity.

A stochastic extension of the π-calculus was first considered in [7] in which the action prefix construct was replaced with an action-rate prefix construct. A number of different variants have since been proposed differing in how rates are added to the prefix construct. In this paper, we follow [19] and parameterize silent (τ) actions with rates and associate a (fixed) rate with each channel. A number of discrete-event simulators for the stochastic π-calculus are available, e.g., BioSpi [9] and SPiM [19], but to our knowledge, no model checking tools.

Structure. The remainder of this paper is structured as follows: Section 2 introduces the syntax and semantics for probabilistic and stochastic extensions of the π-calculus. Sections 3 and 4 describe our extension of MMC for evaluating these semantics and show how the result of this extension can be processed into input for the PRISM tool. Section 5 presents experimental results and Section 6 concludes the paper. A preliminary version of this paper (with only the discrete probabilistic case) appeared as [20].

2 The π-Calculus

The π-calculus is a process algebra for modeling concurrency and mobility. Based on value-passing CCS [21], a key distinguishing feature of the calculus is that it uses a single datatype, *names*, for both channels and values, with the consequence that it is possible to communicate channel names between processes.

In this section, we present the probabilistic and stochastic extensions of the π-calculus for which we have developed automated model checking procedures. In order to facilitate model checking, we make two simple assumptions. First, we restrict our attention to *finite-control* π-calculus processes, i.e., where recursion is not permitted within parallel composition. This is necessary to ensure that the resulting models are finite-state and is, in fact, also imposed by the MMC π-calculus model checker, on which our work relies.

Second, we require that the systems to which we apply model checking are *closed*, intuitively meaning that they receive no inputs from their environment and send no outputs to it. This is due to the nature of the properties that are analyzed by probabilistic model checkers such as PRISM. We will discuss this issue further in Section 4.6.

Preliminaries. Before describing the probabilistic variants of the π-calculus, we present some preliminary notation and definitions. Throughout the paper, we will assume a countable set $\mathcal{N}$ of *names*, ranged over by x, x_i, y, etc.

A *match* is an equality test on names from $\mathcal{N}$ and a *condition* M is a finite conjunction of matches, i.e., M is of the form $[x_1 = y_1] \wedge \cdots \wedge [x_n = y_n]$. We denote by $n(M)$ the set of names that appear in M (ignoring any trivial equality tests of the form $[x = x]$).

A *substitution* σ is a partial mapping from $\mathcal{N}$ to $\mathcal{N}$. The simplest substitutions are of the form $\{y/x\}$ which maps x to y. We let $n(\sigma)$ denote the set of names that the substitution affects, i.e., $n(\sigma) = \{x \mid \exists y(\neq x) \in \mathcal{N}.\ \sigma(x) = y\} \cup \{x \mid \exists y(\neq x) \in \mathcal{N}.\sigma(y) = x\}$. A substitution σ *satisfies* the match $[x = y]$, denoted by $\sigma \models [x = y]$ if $\sigma(x) = \sigma(y)$. Satisfaction extends to conjunctions of matches in the obvious way, e.g., $\sigma \models [x_1 = y_1] \wedge [x_2 = y_2]$ if $\sigma \models [x_1 = y_1]$ and $\sigma \models [x_2 = y_2]$.

We will use five different action types for the two extensions of the π-calculus: τ (silent action), $r (\in \mathbb{R})$ (rate action), $x(y)$ (input), $\bar{x}y$ (output), and $\bar{x}(y)$ (bound output). The *bound names* for an action α, denoted by $bn(\alpha)$, are defined as follows: $bn(\tau) = bn(r) = bn(\bar{x}y) = \emptyset$ and $bn(x(y)) = bn(\bar{x}(y)) = \{y\}$. A substitution σ can also be applied to an action α, denoted by $\alpha\sigma$. The definition of this is: $\tau\sigma = \tau$, $r\sigma = r$, $(x(y))\sigma = \sigma(x)(y)$ if $y \notin n(\sigma)$, $(\bar{x}y)\sigma = \sigma(\bar{x})\,\sigma(y)$ and $(\bar{x}(y))\sigma = \sigma(\bar{x})(y)$ if $y \notin n(\sigma)$. Note that in the case of input and bound output actions (i.e., those with bound variables), the substitution is only defined when the substitution does not change the bound names.

2.1 The Simple Probabilistic π-Calculus

We use a probabilistic extension of the π-calculus called the *simple probabilistic π-calculus* or π_{prob}, which adds a discrete probabilistic choice operator to the basic calculus. This choice operator is *blind*, meaning that probabilities are associated only with silent τ actions, and not input or output actions.

Syntax. We will let P, P_i range over terms and α range over actions. Using, as above, x, y, y_i to range over names, the syntax of the simple probabilistic π-calculus is

$$\begin{aligned} \alpha &::= \tau | x(y) | \bar{x}y, \\ P &::= \mathbf{0} | \alpha.P | \sum_{i \in I} P_i \,\Big|\, \textstyle\sum\!\!\!\!\bigcirc_{i \in I} p_i \tau.P_i \Big| P|P \Big|, \\ &\quad\ \nu x P \big| [x = y]P \big| A(y_1, \ldots, y_n), \end{aligned}$$

where I is an index set, $p_i \in (0, 1]$ with $\sum_{i \in I} p_i = 1$, and A is a process identifier. In the following paragraphs, we provide an informal description of the calculus. The next section presents the formal semantics.

The inactive process, denoted by $\mathbf{0}$, can perform no actions. The action-prefixed process $\alpha.P$ can perform action α and then evolve into P, where α is one of three types: $x(y)$ inputs a name on x and stores it in y, $\bar{x}y$ outputs the name y on x, and τ is the silent action representing internal communication.

There are two types of choice: nondeterministic $\sum_{i\in I} P_i$ and probabilistic $\textstyle\sum_{i\in I} p_i\tau.P_i$. The former is standard in the π-calculus (and indeed CCS). The latter is the only new operator in this probabilistic extension of the π-calculus. As mentioned above, branches of the probabilistic choice operator are always prefixed with τ actions. The process $\textstyle\sum_{i\in I} p_i\tau.P_i$ randomly selects an index $i \in I$ with probability p_i, performs a τ action, and then evolves to process P_i. We use $p_1\tau.P_1 \oplus p_2\tau.P_2$ to denote the binary form of probabilistic choice.

The parallel composition $P_1 \mid P_2$ can either proceed asynchronously or interact through matching input/output actions. The restriction $\nu x\, P$ localizes the scope of x in process P, i.e., x can be considered a new and unique name within P. The match construction $[x = y]P$ can evolve as process P only if the match $[x = y]$ is satisfied, i.e., names x and y are identical. Finally, $A(y_1, \dots, y_n)$ is a recursive call with a corresponding process definition clause of the form $A(x_1, \dots, x_n) \stackrel{\triangle}{=} P$.

An occurrence of name y in process P is *bound* if it is in a subexpression of P of the form $x(y)$ (input-bound) or νy (ν-bound); otherwise, it is *free*. The sets of free and bound names of P are denoted by $fn(P)$ and $bn(P)$, respectively, and the set of all names is $n(P)$. Without loss of generality, we also make the assumption that bound names are all distinct from each other and from free names. This can always be achieved through alpha conversion. A process which contains no free names is said to be *closed*.

Symbolic semantics. The operational semantics for probabilistic extensions of the π-calculus are typically expressed in terms of Markov decision processes (MDPs) or, equivalently, probabilistic automata [22], which allow both probabilistic and nondeterministic behavior. Existing presentations of the semantics (for example, [5], which describes a calculus essentially identical to π_{prob}) are *concrete* in the sense that the semantic rules directly define the MDP that corresponds to a process term. In this paper, we use a *symbolic* presentation of the operational semantics [23]. This approach is, in fact, quite common for the π-calculus and is particularly beneficial in the context of automatic tool support, as is the case here, or for development of bisimulation theories [23], [24].

The main features of the symbolic semantics, which allow one to obtain compact models, are the following:

- As in the *late* semantics of the π-calculus, the input variable of input transitions is kept as a name variable (in contrast to the *early* semantics, where a different transition is generated for every possible name instance).
- Analogously to the match rule, in the communication rule, the match between the input and the output channel is represented by a constraint (*condition*).

In principle, it is possible to define an early version of the symbolic semantics, but such a version would differ from a concrete semantics only because it would contain the free variables of the initial process (and conditions on them). Therefore, such a version would lack the "raison d'être" of the symbolic semantics: efficiently representing the effects of the run-time communications.

Consider the simple process $a(x)\,.\,\bar{x}b\,.\,\mathbf{0}$ which inputs a name x on channel a and then uses x as a channel on which to output the name b. A *concrete* approach to the semantics can establish that this process can accept an input on channel a, but its subsequent behavior (which is dependent on the input x) can only be captured once it is known which other processes it will be composed with. A *symbolic* approach allows the semantics of a process to include variables (e.g., x) that can be used in actions (e.g., $\bar{x}b$). This allows us to adopt a compositional approach: given a parallel composition of several processes, the semantics of each of them can be computed separately in full, and then composed afterward.

The symbolic semantics of the π_{prob}, calculus is expressed in terms of *probabilistic symbolic transition graphs* (PSTGs). These are a simple probabilistic extension of the *symbolic transition graphs* of [23], previously used for the (nonprobabilistic) π-calculus [25], [26], [27], [28] and for CCS [23]. Alternatively, they can be seen as a symbolic extension of Markov decision processes.

Let P be a π_{prob} process. The *probabilistic symbolic transition graph* (PSTG) representing the semantics of the process P is a tuple $(S, s_{init}, \mathcal{T}_{\text{prob}})$, where

- S is the set of symbolic states, each of which is a term of the simple probabilistic π-calculus;
- $s_{init} \in S$, the initial state, is the term P;
- $\mathcal{T}_{\text{prob}} \subseteq S \times Cond \times Act \times Dist(S)$ is the *probabilistic symbolic transition relation* and is the least relation given by the rules in Fig. 1.

In the above,

- $Cond$ denotes the set of all conditions (finite conjunctions of matches) over $\mathcal{N}$;
- Act is a set of actions of four basic types: τ, $x(y)$, $\bar{x}y$, and $\bar{x}(y)$, where $x, y \in \mathcal{N}$;
- $Dist(S)$ is the set of probability distributions over S.

We use the notation $Q \xrightarrow{M,\alpha} \{|p_i : Q_i|\}_i$ for the probabilistic symbolic transition $(Q, M, \alpha, \mu) \in \mathcal{T}_{\text{prob}}$, where $\mu(R) = \sum_{Q_i=R} p_i$ for any π_{prob} term R. For simplicity, we abbreviate the transition $Q \xrightarrow{M,\alpha} \{|1 : Q'|\}$ to $Q \xrightarrow{M,\alpha} Q'$ and omit the trivial condition $true$. We use multisets to ensure that processes with duplicate components such as $Q = \frac{1}{2}\tau.\mathbf{0} \oplus \frac{1}{2}\tau.\mathbf{0}$ have transitions of the form $Q \xrightarrow{\tau} \{|\frac{1}{2} : \mathbf{0}, \frac{1}{2} : \mathbf{0}|\}$ as opposed to $Q \xrightarrow{\tau} \{\frac{1}{2} : \mathbf{0}\}$.

Of the four action types in Act, the first three are described in the previous section. The fourth, $\bar{x}(y)$, denotes output of a bound name and is used by the rules OPEN and CLOSE to extend the scope of the bound name y.

A symbolic state Q encodes a set of π_{prob} terms. More specifically, it encodes the set of terms obtained from Q by applying substitutions to its name variables. A substitution

$$\textsc{Pre}\quad \frac{}{\alpha.P \xrightarrow{\alpha} \{\!|1 : P|\!\}} \qquad \textsc{Prob}\quad \frac{}{(\sum_i p_i\tau.P_i) \xrightarrow{\tau} \{\!|p_i : P_i|\!\}_i} \qquad \textsc{Sum}\quad \frac{P_j \xrightarrow{M,\alpha} \{\!|p_{j_k} : P_{j_k}|\!\}_{j_k}}{(\sum_{i\in I} P_i) \xrightarrow{M,\alpha} \{\!|p_{j_k} : P_{j_k}|\!\}_{j_k}}\, j \in I$$

$$\textsc{Par}\quad \frac{P \xrightarrow{M,\alpha} \{\!|p_i : P_i|\!\}_i}{P\,|\,Q \xrightarrow{M,\alpha} \{\!|p_i : (P_i\,|\,Q)|\!\}_i}\, bn(\alpha)\cap fn(Q)=\emptyset \qquad \textsc{Com}\quad \frac{P \xrightarrow{M,y(z)} \{\!|1 : P'|\!\} \quad Q \xrightarrow{N,\bar{x}v} \{\!|1 : Q'|\!\}}{P\,|\,Q \xrightarrow{[x=y]\wedge M\wedge N,\tau} \{\!|1 : P'\{v/z\}\,|\,Q'|\!\}}$$

$$\textsc{Res}\quad \frac{P \xrightarrow{M,\alpha} \{\!|p_i : P_i|\!\}_i}{\nu x\,P \xrightarrow{\nu x M,\alpha} \{\!|p_i : \nu x\,P_i|\!\}_i}\, x\notin n(\alpha) \qquad \textsc{Close}\quad \frac{P \xrightarrow{M,y(z)} \{\!|1 : P'|\!\} \quad Q \xrightarrow{N,\bar{x}(v)} \{\!|1 : Q'|\!\}}{P\,|\,Q \xrightarrow{[x=y]\wedge M\wedge N,\tau} \{\!|1 : \nu v(P'\{v/z\}\,|\,Q')|\!\}}$$

$$\textsc{Open}\quad \frac{P \xrightarrow{M,\bar{y}x} \{\!|1 : P'|\!\}}{\nu x\,P \xrightarrow{\nu x M,\bar{y}(x)} \{\!|1 : P'|\!\}}\, x\neq y \qquad \textsc{Match}\quad \frac{P \xrightarrow{M,\alpha} \{\!|p_i : P_i|\!\}_i}{[x{=}y]P \xrightarrow{[x=y]\wedge M,\alpha} \{\!|p_i : P_i|\!\}_i}\, \{x,y\}\cap bn(\alpha)=\emptyset$$

$$\textsc{Ide}\quad \frac{P\{y_1,\ldots,y_n/x_1,\ldots,x_n\} \xrightarrow{M,\alpha} \{\!|p_i : P_i|\!\}_i}{A(y_1,\ldots,y_n) \xrightarrow{M,\alpha} \{\!|p_i : P_i|\!\}_i}\, A(x_1,\ldots,x_n)\triangleq P$$

$$\boxed{\begin{array}{rcll} \nu x\,(\mathit{true}) & = & \mathit{true} & \\ \nu x\,[x{=}x] & = & \mathit{true} & \\ \nu x\,[x{=}y] & = & \mathit{false} & (x{\neq}y) \\ \nu x\,[y{=}z] & = & [y{=}z] & (x{\neq}y \wedge x{\neq}z) \\ \nu x\,(M\wedge N) & = & (\nu x\,M)\wedge(\nu x\,N) & \end{array}}$$

Fig. 1. The symbolic semantics for π_{prob}, including (inset) application of operator νx to conditions.

σ is applied to a process Q, denoted by $Q\sigma$, by replacing each action α in Q with $\alpha\sigma$. Consider, for example, the process $Q = a(x).\bar{x}b.\mathbf{0}$. We have that $Q \xrightarrow{a(x)} Q'$, where $Q' = \bar{x}b.\mathbf{0}$. The symbolic state Q' represents the terms $Q'\{z/x\}$ for any name z.

A symbolic transition $Q \xrightarrow{M,\alpha} \{\!|p_i : Q_i|\!\}_i$ represents the fact, that under any substitution σ satisfying M, the process term $Q\sigma$ can perform action $\alpha\sigma$ and then with probability p_i evolve to process $Q_i\sigma$. This is formally stated in Lemma 1 below, which relates the symbolic (PSTG) semantics of π_{prob}, as given in Fig. 1, and the concrete (MDP) semantics, as presented, e.g., in [5]. This corresponds to [27, Lemma 2.5], which discusses symbolic semantics for the (nonprobabilistic) π-calculus. In the lemma, $\sigma \models M$ indicates that the substitution σ satisfies the condition M of the transition, and the constraint $bn(\alpha)\cap(fn(P)\cup n(\sigma))=\emptyset$ corresponds to the fact that bound names are not substituted in order to prevent possible conflicts between bound and free names.

Lemma 1. *Let P be a π_{prob} term:*

1. *If $P \xrightarrow{M,\alpha} \{\!|p_i : P_i|\!\}_i$, then for any substitution σ such that $\sigma\models M$ with $bn(\alpha)\cap(fn(P)\cup n(\sigma))=\emptyset$, $P\sigma \xrightarrow{\alpha\sigma} \{|p_i : P_i\sigma|\}_i$.*
2. *If $P\sigma \xrightarrow{\alpha} \{\!|p_i : P_i'|\!\}_i$ and $bn(\alpha)\cap(fn(P)\cup n(\sigma))=\emptyset$, then $P \xrightarrow{M,\beta} \{\!|p_i : P_i|\!\}_i$ where $\sigma\models M$ and $(\beta.P_i)\sigma = \alpha.P_i'$.*

Proof. Since the symbolic and concrete semantics of π_{prob} share the same types of actions as the (standard) π-calculus, the proof follows the one for [27, Lemma 2.5], which is straightforward by transition induction. □

2.2 The Stochastic π-Calculus

We now describe a stochastic extension of the π-calculus denoted by π_{stoc}, the underlying semantics of which is expressed in terms of continuous-time Markov chains (CTMCs). Each transition will thus be labeled with a *rate*, representing the parameter of an exponential distribution characterizing the delay until the associated transition is enabled. More precisely, for rate r, the probability that the transition is enabled within t time-units is given by $1-\mathrm{e}^{-r\cdot t}$. As in [19], stochastic behavior is introduced at the syntactic level by associating a rate with each channel x, denoted by $rate(x)$, and by annotating silent τ actions with the rate r at which they occur, i.e., τ_r.

Syntax. Using P, P_i to range over terms and α to range over actions, the syntax of the stochastic π-calculus is

$$\begin{aligned} \alpha &::= \tau_r \big| x(y) \big| \bar{x}y, \\ P &::= \mathbf{0}\big|\alpha.P\big|\sum_{i\in I} P_i\big|P|P\big|, \\ &\qquad \nu x P\big|[x=y]P\big|A(y_1,\ldots,y_n), \end{aligned}$$

where $r\in\mathbb{R}_{>0}$, I is an index set, and A is a process identifier.

As in the probabilistic case, the terms $\mathbf{0}$, $P_1\,|\,P_2$, $\nu x\,P$, $[x=y]P$, and $A(y_1,\ldots,y_n)$ denote inactivity, parallel composition, restriction, match, and recursive call. The prefix process $\tau_r.P$ can (internally) evolve to P with rate r. The choice $\sum_{i\in I} P_i$ represents a *race condition* between the transitions of each P_i: the first of these transitions to become enabled is the one that is taken. Race conditions also arise from parallel composition $(P_1\,|\,P_2)$ between processes. In this case, when two processes synchronize on matching input/output actions on a channel x, the rate of this transition is $rate(x)$.

Symbolic semantics. The operational semantics for the stochastic π-calculus is in terms of CTMCs. Usually (as in,

$$\textsc{Pre}_\tau\ \frac{}{\tau_r.P \xrightarrow{r} P} \qquad \textsc{Pre}_{\textsc{in}}\ \frac{}{x(y).P \xrightarrow{x(y)} P} \qquad \textsc{Pre}_{\textsc{out}}\ \frac{}{\bar{x}y.P \xrightarrow{\bar{x}y} P} \qquad \textsc{Sum}\ \frac{P_j \xrightarrow{M,\alpha} P'_j}{\left(\sum_{i\in I} P_i\right) \xrightarrow{M,\alpha} P'_j}\, j \in I$$

$$\textsc{Par}\ \frac{P \xrightarrow{M,\alpha} P'}{P \,|\, Q \xrightarrow{M,\alpha} P \,|\, Q}\, bn(\alpha) \cap fn(Q) = \emptyset \qquad \textsc{Com}\ \frac{P \xrightarrow{M,y(z)} P' \quad Q \xrightarrow{N,\bar{x}v} Q'}{P \,|\, Q \xrightarrow{[x=y]\wedge M\wedge N, rate(x)} P'\{v/z\} \,|\, Q'}$$

$$\textsc{Res}\ \frac{P \xrightarrow{M,\alpha} P'}{\nu x\, P \xrightarrow{\nu x M,\alpha} P'}\, x \notin n(\alpha) \qquad \textsc{Close}\ \frac{P \xrightarrow{M,y(z)} P' \quad Q \xrightarrow{N,\bar{x}(v)} Q'}{P \,|\, Q \xrightarrow{[x=y]\wedge M\wedge N, rate(x)} \nu v(P'\{v/z\} \,|\, Q')}$$

$$\textsc{Open}\ \frac{P \xrightarrow{M,\bar{y}x} P'}{\nu x\, P \xrightarrow{\nu x M,\bar{y}(x)} P'}\, x \neq y \qquad \textsc{Match}\ \frac{P \xrightarrow{M,\alpha} P'}{[x{=}y]P \xrightarrow{[x=y]\wedge M,\alpha} P'}\, \{x,y\} \cap bn(\alpha) = \emptyset$$

$$\textsc{Ide}\ \frac{P\{y_1,\ldots,y_n/x_1,\ldots,x_n\} \xrightarrow{M,\alpha} P'}{A(y_1,\ldots,y_n) \xrightarrow{M,\alpha} P'}\, A(x_1,\ldots,x_n) \triangleq P$$

$$\begin{array}{rcll} \nu x\,(true) & = & true & \\ \nu x\,[x{=}x] & = & true & \\ \nu x\,[x{=}y] & = & false & (x{\neq}y) \\ \nu x\,[y{=}z] & = & [y{=}z] & (x{\neq}y \wedge x{\neq}z) \\ \nu x\,(M \wedge N) & = & (\nu x\, M) \wedge (\nu x\, N) & \end{array}$$

Fig. 2. The symbolic semantics for π_{prob}, including (inset) application of operator νx to conditions.

e.g., [19], on which our syntax is based), a *concrete* semantics is presented which maps each process term directly to the CTMC it represents. However, as for the probabilistic case (see the discussion in the previous section), in order to adopt a compositional approach, we employ a *symbolic* semantics based on an extension of symbolic transition graphs [23].

Let P be a π_{stoc} process. The *stochastic symbolic transition graph* (SSTG) representing the semantics for the process P is a tuple $(S, s_{init}, \mathcal{T}_{\text{stoc}})$, where

- S is the set of symbolic states, each of which is a term of the stochastic π-calculus;
- $s_{init} \in S$, the initial state, is the term P;
- $\mathcal{T}_{\text{stoc}} \subseteq S \times Cond \times Act \times S$ is the *stochastic symbolic transition multirelation* and is the least multirelation given by the rules in Fig. 2.

In the above,

- $Cond$ denotes the set of all conditions (finite conjunctions of matches) over $\mathcal{N}$.
- Act is a set of actions of four basic types: r, $x(y)$, $\bar{x}y$, and $\bar{x}(y)$, where $r \in \mathbb{R}_{>0}$ and $x, y \in \mathcal{N}$.

The fact that we have used a multirelation is standard for stochastic process algebras [29] and ensures that multiple transitions are generated for expressions with identical components, such as $\tau_r.P + \tau_r.P$. This requirement is because the choice operator is interpreted as a race condition: the first transition to become enabled is the one that is taken. More precisely, since the minimum of two exponential distributions with rates r_1 and r_2 is an exponential distribution whose rate is the sum $r_1 + r_2$, the behavior of the process $\tau_r.P + \tau_r.P$ should be the same as that of $\tau_{2r}.P$. This is captured in the semantics by the inclusion of two separate transitions labeled r in the multirelation $\mathcal{T}_{\text{stoc}}$.

Analogously to the case for PSTGs, discussed in the previous section, a stochastic symbolic transition $Q \xrightarrow{M,\alpha} Q'$ of an SSTG represents the fact that under any substitution σ satisfying M, the process term $Q\sigma$ can perform action $\alpha\sigma$ and then evolve to process $Q'\sigma$. This is formally stated in Lemma 2 below, which relates the symbolic (SSTG) semantics of π_{stoc}, as given in Fig. 2, and the concrete (CTMC) semantics, as found in [19]. Again, this corresponds to Lemma 2.5 in [27] for the standard (nonprobabilistic) π-calculus.

Lemma 2. *Let P be a π_{stoc} term:*

1. *If $P \xrightarrow{M,\alpha} P'$, then for any substitution σ such that $\sigma \models M$ with $bn(\alpha) \cap (fn(P) \cup n(\sigma)) = \emptyset$, $P\sigma \xrightarrow{\alpha,\sigma} P'\sigma$.*
2. *If $P\sigma \xrightarrow{\alpha} Q'$ and $bn(\alpha) \cap (fn(P) \cup n(\sigma)) = \emptyset$, then $P \xrightarrow{M,\beta} Q$ where $\sigma \models M$ and $(\beta.Q)\sigma = \alpha.Q'$.*

Proof. Straightforward by transition induction. The details are almost identical in structure to [27, Lemma 2.5] except that the action τ in the π-calculus is replaced by numerical rates r in π_{stoc}, which do not influence names. $\square$

Strictly speaking, the concrete semantics used above do not correspond precisely to the usual definition of a CTMC, since transitions can be associated with either rates (for τ actions) or inputs/output actions (which have yet to be matched). Furthermore, multiple transitions can occur between the same pair of states (due to the use of a multirelation in the definition of an SSTG). In the semantics of a closed π_{stoc} process, however, only rate-labeled transitions remain and multiple transitions between states are simply summed.

3 Generating PSTGs and SSTGs Using MMC

In this section, we describe the automatic generation of the symbolic transition graph for an arbitrary process expressed in either the simple probabilistic π-calculus or stochastic π-calculus. This is achieved with an extension of the (nonprobabilistic) π-calculus model checker MMC [10], which, from this point on, we refer to as $\mathrm{MMC_{prob}}$. In the next section, we will build upon this, presenting a more efficient, compositional scheme for processes of a specific structure.

$\mathrm{MMC_{prob}}$ is based on only a subset of MMC's functionality: essentially the capability to construct the full set of reachable states of a π-calculus process. The restrictions placed on the syntax of the calculus by MMC are the same as we impose in Section 2.

MMC works by (and derives its efficiency from) exploiting the similarity between the way in which resolution-based logic programming techniques handle variables and the way in which the symbolic semantics of the π-calculus handles names [10]. It is implemented in the logic programming system XSB, which is a dialect of Prolog. π-calculus names are represented by XSB variables. MMC then uses a direct encoding of the symbolic semantics of the calculus into XSB rules, based on the definition of a predicate called `trans`. This approach has several benefits: First, it gives a clear and intuitive implementation; second, and more importantly, this encoding is provably correct [10].

Our implementation is a direct extension of this approach. We have a straightforward encoding of the syntax of both π_{prob} and π_{stoc} into the language of XSB, with names and process identifiers represented by XSB variables and constants, respectively. We then adapt MMC's predicate `trans` to represent the symbolic semantics of each calculus. We first describe the case for the simple probabilistic π-calculus and then discuss the differences in the stochastic case.

The probabilistic case. We begin with the encoding of the syntax of π_{prob} into the language of XSB. Letting $\mathtt{X}, \mathtt{Y}, \mathtt{Y_i}$ range over variables, `P` range over processes and denoting comma-delimited lists of processes as $\overrightarrow{\mathtt{P}}$, the syntax of π_{prob} in the input language of $\mathrm{MMC_{prob}}$ is given by the following BNF grammar:

$$
\begin{aligned}
\mathtt{act} &::= \mathtt{tau} | \mathtt{in}(\mathtt{X},\mathtt{Y}) | \mathtt{out}(\mathtt{X},\mathtt{Y})\\
\mathtt{P} &::= \mathtt{zero}\\
&\quad |\mathtt{pref}(\mathtt{act},\mathtt{P})\\
&\quad |\mathtt{choice}(\overrightarrow{\mathtt{P}})\\
&\quad |\mathtt{prob\_choice}(\overrightarrow{\mathtt{pref}(\mathtt{tau}(\mathtt{p}),\mathtt{P})})\\
&\quad |\mathtt{par}(\mathtt{P},\mathtt{P})\\
&\quad |\mathtt{nu}(\mathtt{X},\mathtt{P})\\
&\quad |\mathtt{match}((\mathtt{X}=\mathtt{Y}),\mathtt{P})\\
&\quad |\mathtt{proc}(\bar{A}(\mathtt{Y_1},\ldots,\mathtt{Y_n})),
\end{aligned}
$$

where $\bar{A}$ is the lower case form of process identifier A, with the definition clause of the form $\mathtt{def}(\bar{\mathtt{A}}(\mathtt{X_1},\ldots,\mathtt{X_n}),\mathtt{P})$.

Assuming that ρ is a one-to-one function mapping XSB variables to π_{prob} names, the following function f_ρ relates the $\mathrm{MMC_{prob}}$ representation of the key components of π_{prob} (conditions, actions, and processes) into their corresponding π_{prob} notation:

Conditions:

$$
\begin{aligned}
f_\rho(\mathtt{true}) &= true,\\
f_\rho(\mathtt{X}=\mathtt{Y}) &= [\rho(\mathtt{X})=\rho(\mathtt{Y})],\\
f_\rho((\mathtt{M},\mathtt{N})) &= f_\rho(\mathtt{M}) \wedge f_\rho(\mathtt{N}).
\end{aligned}
$$

Actions:

$$
\begin{aligned}
f_\rho(\mathtt{tau}) &= \tau,\\
f_\rho(\mathtt{in}(\mathtt{X},\mathtt{Y})) &= \rho(\mathtt{X})(\rho(\mathtt{Y})),\\
f_\rho(\mathtt{out}(\mathtt{X},\mathtt{Y})) &= \overline{\rho(\mathtt{X})}\rho(\mathtt{Y}),\\
f_\rho(\mathtt{out\_bound}(\mathtt{X},\mathtt{Y})) &= \overline{\rho(\mathtt{X})}(\rho(\mathtt{Y})).
\end{aligned}
$$

Processes:

$$
\begin{aligned}
f_\rho(\mathtt{zero}) &= 0,\\
f_\rho(\mathtt{pref}(\mathtt{act},\mathtt{P})) &= f_\rho(\mathtt{act}).f_\rho(\mathtt{P}),\\
f_\rho(\mathtt{choice}(\overrightarrow{\mathtt{P}})) &= \sum_{i=1}^{n} f_\rho(\mathtt{P_i}),\\
f_\rho(\mathtt{prob\_choice}(\overrightarrow{\mathtt{pref}(\mathtt{tau}(\mathtt{p}),\mathtt{P})})) &= \sum_{i=1}^{n} \mathtt{p_i}\tau.f_\rho(\mathtt{P_i}),\\
f_\rho(\mathtt{par}(\mathtt{P_1},\mathtt{P_2})) &= f_\rho(\mathtt{P_1})|f_\rho(\mathtt{P_2}),\\
f_\rho(\mathtt{nu}(\mathtt{X},\mathtt{P})) &= \nu\rho(X)f_\rho(\mathtt{P}),\\
f_\rho(\mathtt{match}((\mathtt{X}=\mathtt{Y}),\mathtt{P})) &= [\rho(X)=\rho(Y)]f_\rho(\mathtt{P}),\\
f_\rho(\mathtt{proc}(\bar{\mathtt{A}}(\mathtt{Y_1},\ldots,\mathtt{Y_n}))) &= \mathtt{A}(\rho(\mathtt{Y_1}),\ldots,\rho(\mathtt{Y_n})),
\end{aligned}
$$

where

$$
\begin{aligned}
\overrightarrow{\mathtt{P}} &\equiv [\mathtt{P_1},\ldots,\mathtt{P_n}],\\
\overrightarrow{\mathtt{pref}(\mathtt{tau}(\mathtt{p}),\mathtt{P})} &\equiv [\mathtt{pref}(\mathtt{tau}(\mathtt{p_1}),\mathtt{P_1}),\ldots,\mathtt{pref}(\mathtt{tau}(\mathtt{p_n}),\mathtt{P_n})],
\end{aligned}
$$

and A is defined with $A(\rho(\mathtt{X_1}),\ldots,\rho(\mathtt{X_n})) \stackrel{\triangle}{=} f_\rho(\mathtt{P})$.

Using the function f_ρ, we can now define the XSB predicate `trans`, which represents the direct encoding of the symbolic semantics of π_{prob} (see Fig. 1) into XSB. A tuple `trans(P, PSteps, M)`, where `PSteps` is a list of compound structures $\mathtt{psteps}(\mathtt{p_i},\mathtt{act},\mathtt{P_i})$, represents a symbolic probabilistic transition:

$$f_\rho(\mathtt{P}) \xrightarrow{f_\rho(M),f_\rho(\mathrm{act})} \{\mathtt{p_i} : f_\rho(\mathtt{P_i})\}_{\mathrm{i}}.$$

The definition of `trans` is shown in Fig. 3. The predicates `prob_branch`, `set_par_steps`, and `set_nu_steps` are defined to construct the list `PSteps` according to the operational semantics rules PROB, PAR, and RES. Other auxiliary predicates used in Fig. 1 are given in Fig. 4. Note the close correspondence between the definitions in Fig. 3 and the rules of the symbolic semantics in Fig. 1.

The soundness and completeness of the encoding can be established by induction on the length of derivations of a query answer of `trans` and a symbolic transition in π_{prob}, respectively. The proof details are similar to Theorems 2 and 3 in [10].

Finally, we add an extra XSB predicate `stg(P)`, which uses query-evaluation on `trans` to derive the PSTG of

```
% PRE:
    trans(pref(act, P), [pstep(1, act, P)], true).
% PROB:
    trans(prob_choice(ProbBranches), PSteps, true) :- prob_branch(ProbBranches, PSteps).
    prob_branch([], []).
    prob_branch([pref(tau(FirstProb), P)|Others],PSteps) :-
        prob_branch(Others, OtherPSteps), append([pstep(FirstProb, tau, P)], OtherPSteps, PSteps).
% SUM:
    trans(choice(Branches), PSteps, M) :-
        length(Branches, Size), upto(Size, I), ith(I, Branches, Branch), trans(Branch, PSteps, M).
% PAR:
    trans(par(P, Q), PSteps, M) :-
        trans(P, PPSteps, M), set_par_psteps(PPSteps, Q, PSteps, 0).
    trans(par(P, Q), PSteps, M) :-
        trans(Q, QPSteps, M), set_par_psteps(QPSteps, P, PSteps, 1).
    set_par_psteps([], _, [], _).
    set_par_psteps([pstep(Prob, A, P)|Others], Q, PSteps, Which) :-
        set_par_psteps(Others, Q, OtherPSteps, Which),
        (Which == 0 -> append([pstep(Prob, A, par(P, Q))], OtherPSteps, PSteps)).
                    ;  append([pstep(Prob, A, par(Q, P))], OtherPSteps, PSteps))).
% RES:
    trans(nu(Y, P), PSteps, M) :-
        trans(P, PPSteps, M), not_in_any(Y, PPSteps), not_in_constraint(Y, M), set_nu_psteps(PPSteps, Y, PSteps).
    set_nu_psteps([], _, []).
    set_nu_psteps([pstep(Prob, A, P1)|Others], Y, PSteps) :-
        set_nu_psteps(Others, Y, OtherPSteps), append([pstep(Prob, A, nu(Y, P1))], OtherPSteps, PSteps).
% COM:
    trans(par(P, Q), [pstep(1, tau, par(P1, Q1))], (M, N, L)) :-
        trans(P, [pstep(1, A, P1)], M), trans(Q, [pstep(1, B, Q1)], N), complement(A, B, L).
% OPEN:
    trans(nu(Y, P), [pstep(1, outbound(X, Z), P1)], M) :-
        trans(P, [pstep(1, out(X, Z), P1)], N, V), Y == Z, Y \== X, not_in_constraint(Y, M).
% CLOSE:
    trans(par(P, Q), [pstep(1, tau, nu(W, par(P1, Q1)))], (M, N, L)) :-
        trans(P, [pstep(1, A, P1)], M), trans(Q, [pstep(1, B, Q1)], N), comp_bound(A, B, W, L).
% MATCH:
    trans(match((X=Y), P), PSteps, M) :- X == Y, trans(P, PSteps, M).
    trans(match((X=Y), P), PSteps, (X=Y, M)) :- X \== Y, trans(P, PSteps, M).
% IDE:
    trans(proc(PN), PSteps, M) :- def(PN, P), trans(P, PSteps, M).
```

Fig. 3. XSB code for the trans predicate encoding the π_{prob}, symbolic semantics.

process P and output it in a simple textual format. This is done through a depth-first traversal of the graph, followed by an enumeration of all its symbolic states and transitions. The XSB code for this can be found in [30].

Example. Consider the simple π_{prob} process $Toss$:

$$Toss(x) \triangleq x(y).\big(p\tau.\bar{y}head.\mathbf{0} \oplus (1-p)\tau.\bar{y}tail.\mathbf{0}\big),$$

and then sends out, on channel y, either $head$ or $tail$, with probability p or $1-p$, respectively. Fig. 5 shows the application of $\mathrm{MMC}_{\text{prob}}$ to the process $Toss$. The first four lines illustrate the encoding of the π_{prob} syntax into XSB. Below that is the output of the tool, i.e., the application of the rule `stg`. Lines starting #i show the π_{prob} term for the ith state, lines starting $*j$ and $'k$ enumerate transitions and the individual edges of transitions, respectively. All bound

```
complement(out(X, W), in(Y, W), W, true) :- X == Y.
complement(out(X, W), in(Y, W), W, (X=Y)) :- X \== Y.
complement(in(X, W), out(Y, W), W, true) :- X == Y.
complement(in(X, W), out(Y, W), W, (X=Y)) :- X \== Y.

comp_bound(outbound(X, W), in(Y, W), W, true) :- X == Y.
comp_bound(outbound(X, W), in(Y, W), W, (X=Y)) :- X \== Y.
comp_bound(in(X, W), outbound(Y, W), W, true) :- X == Y.
comp_bound(in(X, W), outbound(Y, W), W, (X=Y)) :- X \== Y.

not_in_any(_, []).
not_in_any(Z, [pstep(_, A, _)|L]) :-
        not_in(Z, A), not_in_any(Z, L).

not_in(_, tau).
not_in(Z, in(X,Y)) :- Z \== X, Z \== Y.
not_in(Z, out(X,Y)) :- Z \== X, Z \== Y.
not_in(Z, outbound(X,Y)) :- Z \== X, Z \== Y.
not_in(Z, outbound1(X,Y)) :- Z \== X, Z \== Y.

not_in_constraint(_, true).
not_in_constraint(X, (Y=Z)) :- X \== Y, X \== Z.
not_in_constraint(X, (M, N)) :-
        not_in_constraint(X, M), not_in_constraint(X, N).

upto(N, N) :- N > 0.
upto(N, I) :- N > 0, N1 is N - 1, upto(N1, I).
```

Fig. 4. Auxiliary XSB code for the trans predicate.

```
def(toss(X),
  pref(in(X, Y),
  prob_choice([pref(tau(p), pref(out(Y, head), zero)),
    pref(tau(1-p), pref(out(Y, tail), zero))]))).

| ?- stg(toss(try)).

#1: proc(toss(try))
*1: 1 ==
#2: prob_choice([pref(tau(p),pref(out(_h417,head),
    zero)),pref(tau(1-p),pref(out(_h417,tail),zero))])
>1: _h417
'1: -- '1':in(try,_h417) --> 2
*2: 2 ==
#3: pref(out(_h417,head),zero)
'2: -- 'p':tau --> 3
#4: pref(out(_h417,tail),zero)
'3: -- '1 - p':tau --> 4
*3: 3 ==
#5: zero
'4: -- '1':out(_h417,head) --> 5
*4: 4 ==
'5: -- '1':out(_h417,tail) --> 5
[1: try] [2: head] [3: tail]

+++ Statistics of toss(try) +++
Nodes:5, Edges:5, P-Steps:4, Free Names:3, Bound Names:1
```

Fig. 5. Sample output from $\mathrm{MMC}_{\text{prob}}$.

names are given unique names (e.g., _h417) and displayed on lines beginning >. All free names used are listed at the end, plus other statistics for the PSTG.

The stochastic case. The generation of the SSTG for a π_{stoc} process proceeds in almost identical fashion. Since the calculus has no probabilistic choice operator, the list PSteps in the representation trans(P, PSteps, M) of each symbolic transition contains only a single item of the form pstep($\mathtt{r_i}$, act, $\mathtt{P_i}$), where $\mathtt{r_i}$ now represents a real-valued rate, instead of a probability.

The encoding of a rate-labeled prefix process $\tau_r.P$ is treated as a special case of the probabilistic choice operator for π_{prob} with a singleton operand. Input and output actions over a channel x are given dummy rates of 1 which will be replaced with the channel rate $rate(x)$ subsequently. Since $\mathrm{MMC}_{\text{prob}}$ simply enumerates all matching transitions when evaluating the symbolic semantics (and does not remove any duplicates), no special treatment is required to deal with the multirelation in the definition of SSTGs.

4 Translating PSTGs and SSTGs into PRISM

We use the probabilistic model checker PRISM (which supports both MDPs and CTMCs) to perform analysis of the semantic models derived from π_{prob} or π_{stoc} processes. The scheme described in the previous section can be used to translate an arbitrary process described in either the simple probabilistic π-calculus or stochastic π-calculus into the probabilistic or stochastic symbolic transition graph representing its semantics. We apply model checking to closed processes (this issue is discussed further in Section 4.6), for which the symbolic (PSTG or SSTG) semantics and concrete (MDP or CTMC) semantics coincide. The list of states and transitions produced by $\mathrm{MMC}_{\text{prob}}$, as illustrated by the example in Fig. 5, can hence easily be imported directly into PRISM for analysis.

However, for processes of a specific structure, we instead propose to adopt a compositional translation, using the high-level modeling language supported by PRISM. This results in a much more efficient translation procedure. More specifically, we consider the case where systems are of the form $P = \nu x_1 \cdots \nu x_k\, (P_1 \mid \cdots \mid P_n)$ and each P_i contains no instances of the ν operator (including inside recursive definitions). The basic idea is to generate the symbolic transition graph for each subprocess P_i (as described in the previous section), map each individual symbolic transition graph to a PRISM module (a component of a PRISM language model), and then use PRISM to construct the semantics of P through the parallel composition of these modules. Note that the compositional nature of this approach is reliant on our use of *symbolic* semantics. Without this, we would not be able to generate the full semantics of P_i in isolation.

The overall process structure we impose (a parallel composition of a set of processes, optionally enclosed inside a restriction of one or more names) is actually fairly typical: systems are generally modeled as a parallel composition of multiple components and, since we assume that P is closed, it is likely that free names used as channels between processes will be restricted in this way. Furthermore, in most cases, a process can be rearranged to a structurally congruent process which is of the correct form, by pushing ν operators to the outside. We have, for example, that $P_1 \mid \nu x\, P_2$ and $\nu x\, (P_1 \mid P_2)$ are structurally congruent under the assumption that x does not occur in P_1. The only class of processes which cannot be renamed in this way are those that include ν inside recursive definitions. In this case, the process can, in principle, generate an infinite number of new names. This can be resolved in the context of a parallel composition with other processes, and therefore, in such a case, we can resort to the basic approach: use $\mathrm{MMC}_{\text{prob}}$ to construct the symbolic transition graph for the full system and import this directly into PRISM.

There are two principal challenges regarding the translation of symbolic transition graphs into PRISM: 1) mapping the name datatype into PRISM's basic type system and 2) mapping binary (CCS-style) communication of names over channels to PRISM's multiway (CSP-style) synchronization without value passing. In brief, 1) is handled by enumerating the set of all free names, assigning each an (identically named) integer constant to represent it and 2) is handled by introducing an action label for each required combination of process sender/receiver pair, channel, and name. Communication of names between processes is handled by including in each receiver process with a bound input variable x, an identically named local (integer) variable which will be used to store the name assigned to x.

Before discussing the details of this compositional translation, we give both an overview of the PRISM syntax and semantics and a simple example which illustrates the key aspects of the translation.

4.1 PRISM Semantics

A PRISM model comprises a set of n modules, the state of each being given by a set of finite-ranging local variables. The global state of the model is determined by the union of all local variables, which we denote by V. The behavior of each module is defined by a set of guarded commands. When modeling MDPs, these commands take the form

$$[act]\ guard \rightarrow p_1 : u_1 + \cdots + p_m : u_m,$$

where act is an (optional) action label, $guard$ is a predicate over V, $p_i \in (0,1]$, and u_i are updates of the form

$$(x_1' = u_{i,1}) \& \cdots \& (x_k' = u_{i,k}),$$

where $u_{i,j}$ is a function over V. Intuitively, in global state s of the PRISM model, the command is enabled if s satisfies $guard$. If a command is executed, the module will, with probability p_i, update its local variables according to the update u_i, by setting the value of each local variable x_j to $u_{i,j}(s)$.

When modeling CTMCs, commands are of the form

$$[act]\ guard \rightarrow r : u,$$

where act is an (optional) action label, $guard$ is a predicate over V, $r \in \mathbb{R}_{>0}$, and u is an update (of the form shown above). In this case, when the guard is satisfied, there is a transition with rate r that updates the local variables according to u. When multiple commands with the same update are enabled, the corresponding transitions are combined into a single transition whose rate is the sum of the individual rates.

```
const int a = 1; const int b = 2; const int c = 3;
const int d = 4; const int e = 5;
module P1
    s1 : [1..6] init 1;
    v : [0..5] init 0;
    w : [0..5] init 0;
    [] (s1 = 1) → 0.5 : (s1' = 2) + 0.5 : (s1' = 3);
    [a_P1_P2_c] (s1 = 2) → (s1' = 4);
    [a_P1_P2_d] (s1 = 3) → (s1' = 5);
    [c_P3_P1_e] (s1 = 4) → (s1' = 6); & (v' = e)
    [d_P3_P1_e] (s1 = 5) → (s1' = 6); & (w' = e)
endmodule
module P2
    s2 : [1..3] init 1
    x : [0..5] init 0;
    [a_P1_P2_c] (s2 = 1) → (s2' = 2) & (x' = c);
    [a_P1_P2_d] (s2 = 1) → (s2' = 2) & (x' = d);
    [b_P2_P3_x] (s2 = 2) → (s2' = 3);
endmodule
module P3
    s3 : [1..2] init 1
    y : [0..5] init 0;
    [b_P2_P3_x] (s3 = 1) → (s3' = 2) & (y' = x);
    [c_P3_P1_e] (s3 = 2) & (y = c) → (s3' = 3);
    [d_P3_P1_e] (s3 = 2) & (y = d) → (s3' = 3);
endmodule
```

Fig. 6. PRISM code for the example.

In practice (see, for example, Fig. 6), we omit probabilities (or rates) equal to one and elements of updates that are of the form $(x' = x)$. The semantics of the whole PRISM model is the parallel composition of all modules using the standard CSP parallel composition [31] (i.e., modules synchronize over all their common actions). For transitions arising from synchronization between multiple processes, the associated probability or rate is obtained by multiplying those of each component transition. See [32] for the full semantics of the PRISM language.

4.2 Example Translation

Consider the following parallel composition of two processes expressed in the simple probabilistic π-calculus:

- $Q \triangleq \nu a\,(Q_1 \mid Q_2)$;
- $Q_1 \triangleq \nu c\,\nu d\big(\frac{1}{2}\tau.\bar{a}c.c(v).\mathbf{0} \oplus \frac{1}{2}\tau.\bar{a}d.d(w).\mathbf{0}\big)$;
- $Q_2 \triangleq \nu b\big(a(x).\bar{b}x.\mathbf{0} \mid b(y).\bar{y}e.\mathbf{0}\big)$.

Process Q_1 includes two names c and d, available only within the scope of Q_1, representing private channels. It makes a random choice, outputting with equal probability either the name c or d on channel a. It then attempts to receive an input on the corresponding channel (c or d, respectively) and terminates. Process Q_2 is the parallel composition of two subprocesses which communicate over a channel b. The first subprocess inputs a name on channel a (which will be one of the two private channels from Q_1) and reoutputs it on channel b. The second subprocess inputs on channel b and then outputs e on whichever channel it received.

Noting that c and d do not occur in Q_2 and that b does not occur in Q_1, we can rewrite Q as the structurally congruent process P, defined as follows:

- $P \triangleq \nu a\,\nu b\,\nu c\,\nu d\,(P_1 \mid P_2 \mid P_3)$;
- $P_1 \triangleq \frac{1}{2}\tau.\bar{a}c.c(v).\mathbf{0} \oplus \frac{1}{2}\tau.\bar{a}d.d(w).\mathbf{0}$;
- $P_2 \triangleq a(x).\bar{b}x.\mathbf{0}$;
- $P_3 \triangleq b(y).\bar{y}e.\mathbf{0}$,

and the corresponding PSTGs are given by

- $P_1 : Q_1^1 \xrightarrow{\tau} \{\!|\frac{1}{2}{:}Q_2^1, \frac{1}{2}{:}Q_3^1|\!\}$, $Q_2^1 \xrightarrow{\bar{a}c} Q_4^1 \xrightarrow{c(v)} Q_6^1$ and $Q_3^1 \xrightarrow{\bar{a}d} Q_5^1 \xrightarrow{d(w)} Q_6^1$;
- $P_2 : Q_1^2 \xrightarrow{a(x)} Q_2^2 \xrightarrow{\bar{b}x} Q_3^2$;
- $P_3 : Q_1^3 \xrightarrow{b(y)} Q_2^3 \xrightarrow{\bar{y}e} Q_2^3 \rightarrow Q_3^3$.

In the above, we omit probabilities that are 1 and conditions *true*. The PSTGs for P_1, P_2, and P_3 have the sets of bound names $\{v, w\}$, $\{x\}$, and $\{y\}$, respectively, and the combined set of free names is $\{a, b, c, d, e\}$. The resulting PRISM model is shown in Fig. 6. This example will be referred to in the full explanation of the translation given below.

4.3 Formal Translation

We assume that the set of all names in the system is $\mathcal{N}$, which is partitioned into disjoint subsets: $\mathcal{N}^{fn}$, the set of all free names appearing in processes $P_1, \ldots, P_n$, and $\mathcal{N}_1^{bn}, \ldots, \mathcal{N}_n^{bn}$, the sets of input-bound names for processes $P_1, \ldots, P_n$.

For clarity, we will retain, wherever possible, identical notation between the π-calculus terms and the resulting PRISM language description. Thus, each of the n subprocesses (or symbolic transition graphs) P_i becomes a PRISM module P_i and the (finite) set of terms $S_i = \{Q_1^i, \ldots, Q_{k_i}^i\}$ that constitute states of the symbolic transition graph of P_i becomes a set of integer indexes $Q_1^i, \ldots, Q_{k_i}^i$ uniquely representing each one.

Module P_i has $|\mathcal{N}_i^{bn}| + 1$ local variables: its local state (i.e., the state of the corresponding symbolic transition graph) is represented by variable s_i, with range $Q_1^i, \ldots, Q_{k_i}^i$, and each bound name $x_j^i \in \mathcal{N}_i^{bn}$ has a corresponding variable x_j^i, with range $0, \ldots, |\mathcal{N}^{fn}|$. The model also includes $|\mathcal{N}^{fn}|$ integer constants, one for each free name, which are assigned (in some arbitrary order) distinct, consecutive nonzero values. If the value of variable x_j^i is equal to one of the these constants, then the corresponding bound name has been assigned the appropriate free name (by an input action). If $x_j^i = 0$, no input to the bound name has occurred yet.

In this way, the conditions which label transitions of the symbolic transition graph can be translated directly into PRISM. For example, if condition M equals $[x = a] \wedge [y = b]$, where x, y are bound names and a, b free names, then the translation of M into PRISM is identical: $(x = a)$ and $(y = b)$, where x, y are integer variables and a, b integer constants.

In addition, when translating stochastic π-calculus processes, for each free name x we add to the PRISM description a constant $rate\_x$ whose value is equal to $rate(x)$, i.e., the rate associated with the channel x.

For each transition in the symbolic transition graph for P_i, we will include a set of corresponding PRISM commands in the module P_i. We consider each type of transition separately below. Note that if P_i is a simple probabilistic π-calculus term, then from the semantics (see Fig. 1), the only transitions which can include multiple probabilistic choices are internal; therefore, the remaining types of transitions (input and output) can be written in the simplified form $Q_i \xrightarrow{M,\alpha} R_i$. For the stochastic case, since PRISM multiplies the rates of synchronizing transitions and synchronization in the π-calculus is always binary, we associate rates (e.g., $rate\_x$ for channel x) with the "output" transitions and set the rates for "input" transitions to 1 (which is the default so can be omitted).

Case 1 (Probabilistic internal transition). For a transition

$$Q_i \xrightarrow{M,\tau} \{\!|p_1 : R_1^i, \dots, p_m : R_m^i|\!\},$$

we add the command

$$[]\ (s_i = Q_i)\ \&\ M \rightarrow p_1{:}(s_i' = R_1^i) + \cdots + p_m{:}(s_i' = R_m^i).$$

See Fig. 6, line 7, for an example.

Case 2 (Stochastic internal transition). For a transition

$$Q_i \xrightarrow{M,r} R_i,$$

we add the command

$$[]\ (s_i = Q_i)\ \&\ M \rightarrow r : (s_i' = R_i).$$

Case 3 (Output on free name). For a transition

$$Q_i \xrightarrow{M,\bar{x}y} R_i, \quad \text{where } x \in \mathcal{N}^{fn},$$

when translating simple probabilistic π-calculus processes, we add, for each $j \in \{1, \dots, n\} \backslash \{i\}$, the command

$$[x\_P_i\_P_j\_y]\ (s_i = Q_i)\ \&\ M \rightarrow (s_i' = R_i),$$

while for stochastic π-calculus processes we add, for each $j \in \{1, \dots, n\} \backslash \{i\}$,

$$[x\_P_i\_P_j\_y]\ (s_i = Q_i)\ \&\ M \rightarrow rate\_x : (s_i' = R_i).$$

The channel x, sender P_i, receiver P_j, and sent name y are all encoded in the action label. See Fig. 6, lines 8 and 18, for examples of sending free and bound names y, respectively.

Case 4 (Output on bound name). For a transition

$$Q_i \xrightarrow{M,\bar{x}y} R_i, \quad \text{where } x \in \mathcal{N}_i^{bn},$$

in the probabilistic case, we add, for each $a \in \mathcal{N}^{fn}$ and $j \in \{1, \dots, n\} \backslash \{i\}$,

$$[a\_P_i\_P_j\_y]\ (s_i = Q_i)\ \&\ M\ \&\ (x = a) \rightarrow (s_i' = R_i),$$

while in the stochastic case, for each $a \in \mathcal{N}^{fn}$ and $j \in \{1, \dots, n\} \backslash \{i\}$, the command

$$[a\_P_i\_P_j\_y]\ (s_i = Q_i)\ \&\ M\ \&\ (x = a) \rightarrow rate\_a : (s_i' = R_i)$$

is added. This is similar to Case 3 except that we include a command for each possible value a of x. See, for example, lines 24 and 25 of Fig. 6.

Case 5 (Input on free name). For a transition

$$Q_i \xrightarrow{M,x(z)} R_i, \text{ where } x \in \mathcal{N}^{fn},$$

in both cases, we add, for each $y \in \mathcal{N} \backslash \mathcal{N}_i^{bn}$ and $j \in \{1, \dots, n\} \backslash \{i\}$, the command

$$[x\_P_j\_P_i\_y]\ (s_i = Q_i)\ \&\ M \rightarrow (s_i' = R_i)\ \&\ (z' = y).$$

For input actions, we add a line for each possible received name y. The assignment $(z' = y)$ models the update of the bound name z to y. See, for example, lines 16 and 17 of Fig. 6, which match the output commands from lines 8 and 9. Notice that this translation also works in the case where y is a bound name in another process P_j (see, for example, line 23 of Fig. 6).

Case 6 (input on bound name). For a transition

$$Q_i \xrightarrow{M,x(z)} R_i, \text{ where } x \in \mathcal{N}_i^{bn},$$

when translating both simple probabilistic and stochastic processes, we add for each $a \in \mathcal{N}^{fn}$, $y \in \mathcal{N} \backslash \mathcal{N}_i^{bn}$, and $j \in \{1, \dots, n\} \backslash \{i\}$, the command

$$\begin{aligned}&[a\_P_j\_P_i\_y]\ (s_i = Q_i)\ \&\ M\ \&\ (x = a)\\ &\quad \rightarrow (s_i' = R_i)\ \&\ (z' = y).\end{aligned}$$

This case combines elements of Cases 4 and 5: we add a command for each possible pairing of channel a that x may represent and name y that may be received.

Finally, we need to remove some spurious commands added in Cases 5 and 6, since they correspond to input actions which will never occur. More precisely, for each module P_j, we identify labels $x\_P_i\_P_j\_y$ which appear on a command of P_j but which do not appear in any of the commands in module P_i. Commands with such action labels are removed from P_j. For example, in Fig. 6, since process P_1 only outputs c or d on channel a, there is no label of the form $a\_P_1\_P_2\_e$ in module P_1, and therefore, commands with this label have been removed from module P_2.

4.4 Correctness of the Translation

By assumption, the term being translated is finite control, is closed, and of the form $P = \nu x_1 \dots \nu x_k\,(P_1 \mid \cdots \mid P_n)$. The first step in the proof is to show that any term in the derivation tree of P is of the form $\nu x_1 \dots \nu x_k (Q_1\sigma_1 \mid \cdots \mid Q_n\sigma_n)$, where, for any $1 \leq j \leq n$, Q_j is a state of the symbolic transition graph for the process P_j and σ_j is a substitution from the bound names of P_j to the free names of $P_1, \dots, P_n$. The proof is by induction on the (concrete) transition rules using Lemma 1 or Lemma 2, depending on whether we are considering π_{prob} or π_{stoc}.

Using this result, we now show that the translation is correct by constructing a mapping between these terms and the states of the PRISM model and demonstrating that for any term in the derivation tree of P, there is a transition in the (concrete) semantics if and only if the corresponding PRISM state has a matching transition. For any term $\nu x_1 \dots \nu x_k\,(Q_1\sigma_1 \mid \cdots \mid Q_n\sigma_n)$, the state in the PRISM model is constructed as follows: for any $1 \leq j \leq n$, the values of the variables of module P_j are given by $s_j = Q_j$, $x_1^j = i_1^j, \dots, x_{k_j}^j = i_{k_j}^j$, where if $\sigma(x_l^j) = z \in \mathcal{N}^{fn}$, then i_l^j is the integer constant corresponding to the free variable z and otherwise (i.e., $\sigma(x_l^j) = x_l^j$) i_l^j equals 0.

The remainder of the proof is dependent on whether we are in the probabilistic or stochastic setting.

4.4.1 Probabilistic Case

Consider any π_{prob} term Q in the derivation tree, where $Q = \nu x_1 \dots \nu x_k\,(Q_1\sigma_1 \mid \cdots \mid Q_n\sigma_n)$ and the transition $Q \xrightarrow{\tau} \{\!|p_m : R_m|\!\}_m$.

From the transition rules and the conditions we have imposed on the structure of π_{prob} terms, there are the following two cases to consider:

Internal transition. $Q_j\sigma_j \xrightarrow{\tau} \{\!|p_m : R^{j'}_m|\!\}_m$ and $R_m = \nu x_1 \dots \nu x_k\,(Q_1\sigma_1 \mid \cdots \mid R^{j'}_m \mid \cdots \mid Q_n\sigma_n)$. From Lemma 1b, we have $Q_j \xrightarrow{M_j,\tau} \{\!|p_m : R^j_m|\!\}$, where $\sigma_j \models M_j$ and $R^j_m\sigma_j = R^{j'}_m$. Hence, by construction, in the module P_j, there is a command of the form

$$[]\ (s_j = Q_j)\ \&\ M_j \rightarrow p_1{:}(s'_j = R^j_1) + \cdots + p_m{:}(s'_j = R^j_m).$$

Finally, since $\sigma_j \models M_j$ and by definition of the mapping between π_{prob} terms and PRISM, it follows that the PRISM state corresponding to Q satisfies the guard $(s_j = Q_j)\,\&\,M_j$ and that the transition is preserved in the translation.

Communication. $Q_j\sigma_j \xrightarrow{x(z)} R'_j$, $Q_l\sigma_l \xrightarrow{\bar{x}y} R'_l$, $j \neq l$, and $\{\!|p_m : R_m|\!\}_m = \{\!|1 : R|\!\}$, where $R = \nu x_1 \dots \nu x_k\,(Q_1\sigma_1 \mid \cdots \mid R'_j\{y/z\} \mid \cdots \mid R'_l \mid \cdots \mid Q_n\sigma_n)$. From Lemma 2b, assuming without loss of generality that z is fresh

- $Q_j \xrightarrow{M_j,x_j(z_j)} R_j$, where $\sigma_j \models M_j$ and $(x_j(z_j).R_j)\sigma_j = x(z).R'_j$;
- $Q_l \xrightarrow{M-l,\bar{x}_l y_l} R_l$, where $\sigma_l \models M_l$ and $(\bar{x}_l y_l.R_l)\sigma_l = \bar{x}y.R'_l$.

Now, since z is fresh, it follows that $z = z_j$ and, because σ_l is a substitution from bound to free names of $P_1, \dots, P_n$, it follows that $y \in \mathcal{N} \backslash \mathcal{N}^{bn}_j$. In addition, since σ_j is a substitution from bound to free names, either x_j is free and equals x, and hence, in module P_j, we have the command

$$[x\_P_l\_P_j\_y]\ (s_j = Q_j)\ \&\ M_j \rightarrow (s'_j = R_j)\ \&\ (z'_j = y),$$

or x_j is bound and, since $x_j\sigma_j = x$, it follows that x is free, and therefore, the command

$$[x\_P_l\_P_j\_y]\ (s_j = Q_j)\ \&\ M_j\ \&\ (x_j = x) \rightarrow (s'_j = R_j)\ \&\ (z'_j = y)$$

appears in module P_j. Employing similar arguments, if x_l is free, then $x_l = x$ and the command

$$[x\_P_l\_P_j\_y]\ (s_l = Q_l)\ \&\ M_l \rightarrow (s'_l = R_l)$$

appears in module P_l. While, if x_l is bound, then module P_l includes the command

$$[x\_P_l\_P_j\_y]\ (s_l = Q_l)\ \&\ M_l\ \&\ (x_l = x) \rightarrow (s'_l = R_l).$$

Since $\sigma_j \models M_j$, $\sigma_l \models M_l$, $x_j\sigma_j = x$, and $x_l\sigma_l = x$, it follows that the guards $(s_j = Q_j)\ \&\ M_j$, $(s_j = Q_j)\ \&\ M_j\ \&\ (x_j = x)$, $(s_l = Q_l)\ \&\ M_l$, and $(s_l = Q_l)\ \&\ M_l\ \&\ (x_l = x)$ hold in the PRISM state encoding Q. Finally, since the encoding of $R'_j\{y/z\}$ can be obtained from the encoding of $R_j\sigma_j$ by setting the variable z to value y, it follows that the transition is preserved by the translation.

To complete the proof, it remains to show that for any transition of the PRISM model, there is a matching transition in the corresponding π_{prob} term. The result follows in a similar manner to the above using Lemma 1a instead of Lemma 1b.

4.4.2 Stochastic Case

Consider any π_{stoc} term Q in the derivation tree, where $Q = \nu x_1 \dots \nu x_k\,(Q_1\sigma_1 | \cdots | Q_n\sigma_n)$ and the transition $Q \xrightarrow{r} \nu x_1 \dots \nu x_k\,R$.

From the transition rules and the conditions we have imposed on the structure of π_{stoc} terms, there are the following two cases to consider:

Internal transition. $Q_j\sigma_j \xrightarrow{r} R'_j$ and $R = Q_1\sigma_1 | \cdots | R'_j \mid \cdots \mid Q_n\sigma_n$. From Lemma 2b, we have $Q_j \xrightarrow{M_j,r} R_j$, where $\sigma_j \models M_j$ and $R_j\sigma_j = R'_j$. Hence, by construction, in the module P_j, there is a command of the form

$$[]\ (s_j = Q_j)\ \&\ M_j \rightarrow r : (s'_j = R_j).$$

Finally, since $\sigma_j \models M_j$ and by definition of the mapping between π_{stoc} terms and PRISM, it follows that the PRISM state corresponding to Q satisfies the guard $(s_j = Q_j)\,\&\,M_j$ and that the transition is preserved in the translation.

Communication. $Q_j\sigma_j \xrightarrow{x(z)} R'_j$, $Q_l\sigma_l \xrightarrow{\bar{x}y} R'_l$, $j \neq l$, $R = Q_1\sigma_1 \mid \cdots \mid R'_j\{y/z\} \mid \cdots \mid R'_l \mid \cdots \mid Q_n\sigma_n$, and $rate(x) = r$. From Lemma 2b, assuming without loss of generality that z is fresh:

- $Q_j \xrightarrow{M_j,x_j,(z_j)} R_j$, where $\sigma_j \models M_j$ and $(x_j(z_j).R_j)\sigma_j = x(z).R'_j$;
- $Q_l \xrightarrow{M_l,\bar{x}_l y_l} R_l$, where $\sigma_l \models M_l$ and $(\bar{x}_l y_l.R_l)\sigma_l = \bar{x}y.R'_l$.

We employ the same arguments used in the probabilistic case. If x_j is free, module P_j contains the command

$$[x\_P_l\_P_j\_y]\ (s_j = Q_j)\ \&\ M_j \rightarrow (s'_j = R_j)\ \&\ (z'_j = y),$$

while if x_j is bound, it contains the command

$$[x\_P_l\_P_j\_y]\ (s_j = Q_j)\ \&\ M_j\ \&\ (x_j = x) \rightarrow (s'_j = R_j)\ \&\ (z'_j = y).$$

Similarly, if x_l is free, the command

$$[x\_P_l\_P_j\_y]\ (s_l = Q_l)\ \&\ M_l \rightarrow rate\_x : (s'_l = R_l)$$

appears in module P_l and, if x_l is bound, then the command

$$[x\_P_l\_P_j\_y]\ (s_l = Q_l)\ \&\ M_l\ \&\ (x_l = x) \rightarrow rate\_x : (s'_l = R_l)$$

appears in module P_l.

The remaining arguments are the same as in the probabilistic case, using additionally the fact that the PRISM constant $rate\_x$ has been given the value $rate(x)$.

4.5 Optimizations

The translation from symbolic transition graphs to PRISM code described in this section can be optimized to reduce the size of the generated code and the resulting model. The basic idea is to compute an overapproximation of the possible values that each symbolic transition graph's bound name can take, and thus, the channels it can send out on and the values that can be sent on those channels. With this information, we can decrease the range of the PRISM local variables corresponding to each bound name and remove unnecessary commands corresponding to combinations of channel, value, and processes that can never occur. The overapproximation is computed iteratively, starting with an empty set of possible values for each bound name, and at each step adding any name that can be received upon any channel that can be used to assign to the bound name. The iterations required is bounded by the number of processes n. For clarity of presentation, the example in Fig. 6 has, in fact, been optimized in this way.

This optimization could be improved by employing more complex techniques based on those developed in [18] which use control flow analysis to establish an overapproximation of the set of channels a name may be bound to and the set of names that may be sent along a given channel.

4.6 Properties

For probabilistic model checking of MDPs and CTMCs, properties are typically specified using the temporal logics PCTL [33], [34] and CSL [35], [36], the key components of which are timed and untimed *probabilistic reachability*. Examples of expressible properties include the maximum probability of a failure occurring ($\mathtt{P}_{\max=?}[\mathtt{F}\ failure]$), the minimum probability of a process successfully completing ($\mathtt{P}_{\min=?}[\mathtt{F}\ success]$), the probability that a message is delivered by time $t(\in \mathbb{R})$ ($\mathtt{P}_{=?}[\mathtt{F}^{\leq t}\ delivered]$), and the probability of a reaction occurring in the time interval $[t_1, t_2](\subseteq \mathbb{R})$ ($\mathtt{P}_{=?}[\mathtt{F}^{[t_1,t_2]}\ reaction]$). In practice, a wide range of useful properties can be expressed in this way.

Most probabilistic model checking tools, including PRISM, use state-based property specifications, i.e., the atomic propositions ($failure$, $delivered$, etc.) in the examples above are quantifier-free predicates identifying a set of states in the model. Also, the models that are checked are closed: there are no inputs/outputs between the model and its environment, only between components included within the model. This is our reason for only performing probabilistic model checking on closed π-calculus processes.

In terms of the translation from π-calculus description to PRISM model, we simply need to be able to identify the particular set of target states specified in the reachability property. This is done through the $\mathrm{MMC}_{\mathrm{prob}}$ translator when it constructs a PSTG or SSTG: either by identifying which symbolic states correspond to a particular process term; or those in which a particular action is available (in the latter case, such actions can be added purely for the purposes of identifying states, and then removed through restriction).

For example, consider a distributed randomized algorithm executed between n parallel components $P_1, \dots, P_n$. A typical property to be checked is that algorithm always terminates with probability 1 (for any possible scheduling of the n components). In this case, we would identify the term in the π-calculus description of each process P_i that corresponds to that process finishing its execution of the algorithm. From the output of the $\mathrm{MMC}_{\mathrm{prob}}$ translator, we can identify the corresponding local state Q_i of the process. We would then compute (in PRISM) the (minimum probability) of reaching the state $s_1 = Q_1 \wedge \cdots \wedge s_n = Q_n$.

Although not considered in the case studies used in this paper, our implementation could also be extended to allow for the computation of cost- or reward-based properties, which are also supported by PRISM. This allows expression of properties such as the "maximum expected number of messages sent before termination" or "the minimum expected power consumption within t time units." Typically the cost/reward information needed for these properties is added to the model (MDP or CTMC) by annotating either transitions labeled with particular actions (for example, the action-label which corresponds to a message being sent between two components) or states with real values. Since our translation of the probabilistic or stochastic π-calculus to PRISM preserves both information about the state and channel communications of a process, information of this kind could be incorporated into the translation in a relatively straightforward fashion.

More general temporal properties, for example, that a certain sequence of actions is performed, could be encoded through the addition of a test/watchdog process [37]. Model checking for specification formalisms more specifically tailored to the mobile aspects of the π-calculus, such as spatial logic [38], will be an area of future work.

5 Implementation and Results

Our implementation of model checking for the simple probabilistic π-calculus and stochastic π-calculus is fully automated and comprises three parts: 1) $\mathrm{MMC}_{\mathrm{prob}}$, an extension of MMC (as described in Section 3), which constructs the symbolic transition graphs for a simple probabilistic or stochastic π-calculus process; 2) the translator from the symbolic transition graph to PRISM code (as described in Section), implemented in Java; and 3) the probabilistic model checker PRISM [11] which builds the MDP/CTMC from part 2) and performs verification of PCTL/CSL properties. We based our implementation on MMC 1.0 and PRISM 3.1.1.

First, we consider the dining cryptographers protocol (DCP) [39], Chaum's randomized solution to the classic anonymity problem in which a group of N parties collectively establish whether either one of the group or an independent party has to make a payment. If the former, this is achieved without any of the $N-1$ nonpaying parties knowing the identity of the paying one. This was previously modelled in the probabilistic π-calculus in [6]. To check anonymity, we compute the probability of reaching each of the possible outcomes of the protocol (from the point of view of an individual party) and establish that they are identical.

Second, we study the partial secret exchange (PSE) algorithm of [3] for anonymous contract signing between two parties. A probabilistic π-calculus model of PSE was given in [5]. The protocol was independently analyzed in PRISM [40], where a potential flaw of the protocol was identified, in that one party always has an advantage over the other. Several modifications to the protocol were proposed and shown to have a lower probability of this occurring. We used a π_{prob} model of both the original and a modified version to demonstrate the same flaw.

Third, we constructed both a probabilistic and stochastic model of a mobile communication network (MCN), based on the (nonprobabilistic) π-calculus model in [41]. The system comprises N base stations with fixed communication links to a mobile switching center and a mobile station which can be connected to each of the base stations via radio links. The mobile station roams between the base

TABLE 1
Performance of the Probabilistic Model Checking Process

Case study	N	Model size: States	Model size: Transitions	MTBDD size (nodes)	Construction time (sec.): PSTGs/SSTGs	Construction time (sec.): PRISM code	Construction time (sec.): MDP/CTMC	Model checking in PRISM (sec.)
DCP	5	160,543	592,397	58,448	2.20	0.27	0.93	5.21
	6	1,475,401	6,520,558	100,122	2.50	0.27	1.98	15.1
	7	13,221,889	68,121,834	154,074	2.95	0.31	3.10	39.4
	8	116,192,457	683,937,352	220,043	3.31	0.31	4.23	90.8
	9	1,005,495,499	6,657,256,911	298,285	3.62	0.36	6.26	316.2
PSE	3	9,321	32,052	17,999	1.63	0.21	0.43	0.31
	4	89,025	419,172	43,120	2.12	0.27	0.95	1.23
	5	837,361	5,028,700	88,074	2.60	0.31	1.89	2.96
PSE$_{\mathrm{mod}}$	3	9,328	32,059	18,184	1.57	0.22	0.41	0.86
	4	89,040	419,187	43,388	1.99	0.26	0.89	3.45
	5	837,392	5,028,731	89,309	2.49	0.31	1.96	14.3
MCN (probabilistic)	2	609	950	58,430	1.38	0.31	2.61	0.34
	3	3,611	5,811	216,477	1.60	0.46	12.0	6.06
MCN (stochastic)	2	565	854	32,898	1.44	0.38	2.13	1.18
	3	3,295	5,079	119,197	1.59	0.44	7.05	2.76
FGF	3	13,081	43,330	8,667	1.00	0.11	0.25	2.22
	4	87,109	315,436	28,725	1.08	0.12	1.34	24.1
	5	453,593	1,763,842	108,354	1.21	0.12	8.62	156.6
	6	2,011,729	8,318,684	304,464	1.39	0.16	32.3	999.3

stations. When it changes base station, the mobile communication network acts as an intermediate party, controlling the handover protocol and exchange of communication links between stations. This case study was analyzed using MMC in [10]. In both this and the original paper, though, the occurrence of a failure during the handover protocol was modeled as a nondeterministic choice. In the probabilistic version, we are able to correctly model this as a random event. For the stochastic model, we used the adapted version of [42]. This allows both correct modeling of the failure event and also timing characteristics of the network. We check the probability of a handover operation completing successfully, within a given number of communications (for the probabilistic case) or within a fixed time deadline (for the stochastic case).

Our final case study is a CTMC model of the Fibroblast Growth Factor (FGF) signalling pathway. We consider a slightly simplified version of the model from [43], comprising interactions between a mixture of FGF ligands and receptors. In the π_{stoc} formulation, the ν operator is used to give each FGF ligand a unique channel name. The binding between a particular FGF ligand and receptor is modeled by this name being passed between the two. Unbinding occurs through a communication over this private channel. We check the probability that all FGF receptors have relocated (are no longer active) by a certain time bound.

Table 1 shows the performance of our implementation on the case studies. Experiments were run on a 2-GHz PC with 2-GB RAM running Linux. For each case study, we analyzed several models of increasing size by varying a parameter N. For the DCP model, N represents the number of parties; for PSE (we consider two variants: the original protocol EGL and the modified version EGL3 from [40]), N is the size of contract; for the MCN models, N represents the number of base stations; and for FGF, N is the number of FGF ligands (the number of receptors remains fixed). The table shows the size of the resulting MDPs/CTMCs (number of states/transitions) and corresponding storage in PRISM (MTBDD nodes, where one node uses 20 bytes). We also give the time required for each stage of the process, i.e., constructing: the PSTGs (using $\mathrm{MMC_{prob}}$); the PRISM code (using the translator); and the MDP or CTMC model (using PRISM). Finally, we give the time to check a single (quantitative) PCTL/CSL property for each using PRISM (with the fastest available engine).

The results are very encouraging. We see that our techniques are scalable to the construction and analysis of π_{prob} and π_{stoc} models with extremely large state spaces and that the times required for all stages of the process are relatively small. Furthermore, the compositional approach to the translation proved to be essential. On the FGF model ($N = 3$), for example, constructing the full model in $\mathrm{MMC_{prob}}$ took more than 100 times as long as the compositional technique. For larger parameter values, it was not feasible to directly construct the full model.

The MCN case study, although smallest in terms of state space, is a particularly good example of the applicability of this implementation since it fully exploits all mobile aspects of the calculus. The most obvious area for improvement in our results concerns MTBDD sizes. As is often the case with automatically generated code, the PRISM models resulting from our technique do not always exhibit the kind of structure and regularity that can be exploited by PRISM's symbolic implementation. We are confident that performance can be improved in this area.

6 CONCLUSIONS

In this paper, we have demonstrated the feasibility of implementing model checking for probabilistic and stochastic extensions of the π-calculus. Furthermore, we have shown, through its application to several large examples, the efficiency of the approach. The probabilistic version of the π-calculus we used (with only blind probabilistic choice) has proved to be expressive enough for the appropriate application domains (probabilistic algorithms for security and dynamic communication protocols with failures and/or randomization) and yet amenable to analysis with extensions and adaptions of existing verification tools. Similarly, the version of the stochastic π-calculus we used (with rates assigned to τ-transitions and to channels) is both a natural formalism for modeling biological systems and well suited for the model checking techniques we have proposed.

We would like to extend this work in several directions. For convenience of modeling, we plan to add support for polyadic communication over channels. We also hope to add support for more flexible property specifications using watchdog processes. Finally, we will investigate ways to further improve the efficiency of our implementation, in particular, with regards to the automatically generated PRISM code. Possibilities include optimizations to reduce the resulting symbolic (MTBDD) storage in PRISM and bisimulation minimization techniques.

ACKNOWLEDGMENTS

Authors Gethin Norman and David Parker were with the School of Computer Science at the University of Birmingham and Peng Wu was at CNRS and LIX when parts of this work were first carried out. Gethin Norman and David Parker are supported in part by EPSRC grants GR/S11107 and GR/S46727 and Microsoft Research Cambridge contract MRL 2005-44 and Catuscia Palamidessi and Peng Wu were supported in part by the INRIA/ARC project ProNoBis. The authors thank the anonymous referees for their valuable comments.

REFERENCES

[1] R. Milner, J. Parrow, and D. Walker, "A Calculus of Mobile Processes, I," *Information and Computation,* vol. 100, pp. 1-40, 1992.

[2] M. Reiter and A. Rubin, "Crowds: Anonymity for Web Transactions," *ACM Trans. Information and System Security,* vol. 1, no. 1, pp. 66-92, 1998.

[3] S. Even, O. Goldreich, and A. Lempel, "A Randomized Protocol for Signing Contracts," *Comm. ACM,* vol. 28, no. 6, pp. 637-647, 1985.

[4] O. Herescu and C. Palamidessi, "Probabilistic Asynchronous π-Calculus," *Proc. Third Int'l Conf. Foundations of Software Science and Computation Structures,* J. Tiuryn, ed., pp. 146-160, 2000.

[5] K. Chatzikokolakis and C. Palamidessi, "A Framework to Analyze Probabilistic Protocols and Its Application to the Partial Secrets Exchange," *Proc. Int'l Symp. Trustworthy Global Computing,* R.D. Nicola and D. Sangiorgi, eds., pp. 146-162. Springer, 2005.

[6] M. Bhargava, C. Palamidessi, "Probabilistic Anonymity," *Proc. 16th Int'l Conf. Concurrency Theory,* M. Abadi and L. de Alfaro, eds., pp. 171-185, 2005.

[7] C. Priami, "Stochastic π-Calculus," *Computer J.,* vol. 38, no. 7, pp. 578-589, 1995.

[8] A. Regev, W. Silverman, and E. Shapiro, "Representation and Simulation of Biochemical Processes Using the π-Calculus Process Algebra," *Proc. Pacific Symp. Biocomputing,* R. Altman, A. Dunker, L. Hunter, and T. Klein, eds., vol. 6, pp. 459-470, 2001.

[9] C. Priami, A. Regev, W. Silverman, and E. Shapiro, "Application of a Stochastic Name Passing Calculus to Representation and Simulation of Molecular Processes," *Information Processing Letters,* vol. 80, pp. 25-31, 2001.

[10] P. Yang, C. Ramakrishnam, and S. Smolka, "A Logic Encoding of the π-Calculus: Model Checking Mobile Processes Using Tabled Resolution," *Int'l J. Software Tools Technology Transfer,* vol. 4, pp. 1-29, 2004.

[11] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker, "PRISM: A Tool for Automatic Verification of Probabilistic Systems," *Proc. 12th Int'l Conf. Tools and Algorithms for the Construction and Analysis of Systems,* H. Hermanns and J. Palsberg, eds., pp. 441-444, 2006.

[12] B. Victor and F. Moller, "The Mobility Workbench—A Tool for the π-Calculus," *Proc. Sixth Int'l Conf. Computer Aided Verification,* R. Alur and D. Peled, eds., pp. 428-440, 1994.

[13] B. Blanchet, *ProVerif: Automatic Cryptographic Protocol Verifier User Manual,* www.preverif.ens.fr/proverif-manual.ps.gz, 2005.

[14] S. Chaki, S. Rajamani, and J. Rehof, "Types as Models: Model Checking Message-Passing Programs," *Proc. 29th Symp. Principles of Programming Languages,* pp. 45-57, 2002.

[15] P. Wu, "Interpreting π-Calculus with Spin/Promela," *Computer Science,* vol. 8, pp. 7-9, supplement, 2003.

[16] H. Song and K. Compton, "Verifying π-Calculus Processes by Promela Translation," Technical Report CSE-TR-472-03, Univ. of Michigan, 2003.

[17] A. Venet, "Abstract Interpretation of the π-Calculus," *Proc. Fifth LOMAPS Workshop Analysis and Verification of Multiple-Agent Languages,* M. Dam, ed., pp. 51-75, 1996.

[18] C. Bodei, P. Degano, F. Nielson, and H.R. Nielson, "Static Analysis for the π-Calculus with Applications to Security," *Information and Computation,* vol. 165, pp. 68-92, 2001.

[19] A. Phillips and L. Cardelli, "Efficient, Correct Simulation of Biological Processes in the Stochastic π-Calculus," *Proc. Fifth Int'l Workshop Computational Methods in Systems Biology,* M. Calder and S. Gilmore, eds., pp. 184-199, 2007.

[20] G. Norman, C. Palamidessi, D. Parker, and P. Wu, "Model Checking the Probabilistic π-Calculus," *Proc. Fourth Int'l Conf. Quantitative Evaluation of Systems,* pp. 169-178, 2007.

[21] R. Milner, *Communication and Concurrency.* Prentice Hall, 1989.

[22] R. Segala and N. Lynch, "Probabilistic Simulations for Probabilistic Processes," *Nordic J. Computing,* vol. 2, no. 2, pp. 250-273, 1995.

[23] M. Hennessy and H. Lin, "Symbolic Bisimulations," *Theoretical Computer Science,* vol. 138, pp. 353-389, 1995.

[24] P. Wu, C. Palamidessi, and H. Lin, "Symbolic Bisimulations for Probabilistic Systems," *Proc. Fourth Int'l Conf. Quantitative Evaluation of Systems,* pp. 179-188, 2007.

[25] H. Lin, "Symbolic Bisimulation and Proof Systems for the π-Calculus," technical report, School of Cognitive and Computer Science, Univ. of Sussex, 1994.

[26] M. Boreale and R.D. Nicola, "A Symbolic Semantics for the π-Calculus," *Information and Computation,* vol. 126, no. 1, pp. 34-52, Apr. 1996.

[27] H. Lin, "Complete Inference Systems for Weak Bisimulation Equivalences in the π-Calculus," *Information and Computation,* vol. 180, no. 1, pp. 1-29, 2003.

[28] A. Ingólfsdóttir and H. Lin, "A Symbolic Approach to Value-Passing Processes," *Handbook of Processes Algebra,* chapter 7, Elsevier, 2001.

[29] J. Hillston, *A Compositional Approach to Performance Modelling.* Cambridge Univ. Press, 1996.

[30] G. Norman, C. Palamidessi, D. Parker, and P. Wu, "Translating the Probabilistic π-Calculus to PRISM," Technical Report CSR-07-02, School of Computer Science, Univ. of Birmingham, 2007.

[31] A. Roscoe, *The Theory and Practice of Concurrency.* Prentice-Hall, 1997.

[32] Online PRISM documentation, www.prismmodelchecker.org/doc/, 2009.

[33] H. Hansson and B. Jonsson, "A Logic for Reasoning about Time and Reliability," *Formal Aspects of Computing,* vol. 6, no. 5, pp. 512-535, 1994.

[34] A. Bianco and L. de Alfaro, "Model Checking of Probabilistic and Nondeterministic Systems," *Proc. 15th Conf. Foundations of Software Technology and Theoretical Computer Science,* P. Thiagarajan, ed., pp. 499-513, 1995.

[35] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton, "Verifying Continuous Time Markov Chains," *Proc. Eighth Int'l Conf. Computer Aided Verification,* R. Alur and T. Henzinger, eds., pp. 269-276, 1996.

[36] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen, "Model-Checking Algorithms for Continuous-Time Markov Chains," *IEEE Trans. Software Eng.*, vol. 29, no. 6, pp. 524-541, June 2003.

[37] M. Goldsmith, N. Moffat, B. Roscoe, T. Whitworth, and I. Zakiuddin, "Watchdog Transformations for Property-Oriented Model-Checking," *Proc. Second Int'l Symp. Formal Methods Europe,* K. Araki, S. Gnesi, and D. Mandrioli, eds., pp. 600-616, 2003.

[38] L. Caires and L. Cardelli, "A Spatial Logic for Concurrency (Part I)," *Information and Computation,* vol. 186, no. 2, pp. 194-235, 2003.

[39] D. Chaum, "The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability," *J. Cryptology,* vol. 1, pp. 65-75, 1988.

[40] G. Norman and V. Shmatikov, "Analysis of Probabilistic Contract Signing," *J. Computer Security,* vol. 14, no. 6, pp. 561-589, 2006.

[41] F. Orava and J. Parrow, "An Algebraic Verification of a Mobile Network," *Formal Aspects of Computing,* vol. 4, pp. 497-543, 1992.

[42] C. Priami, "Stochastic Analysis of Mobile Telephony Networks," *Proc. Fifth Int'l Workshop Process Algebra and Performance Modeling,* E. Brinskma and A. Nymeyer, eds., pp. 145-171, 1997.

[43] J. Heath, M. Kwiatkowska, G. Norman, D. Parker, and O. Tymchyshyn, "Probabilistic Model Checking of Complex Biological Pathways," *Proc. Fourth Int'l Workshop Computational Methods in Systems Biology,* C. Priami, ed., pp. 32-47, 2006.

Gethin Norman received the degree in mathematics from the University of Oxford and the PhD degree in computer science from the University of Birmingham. He is a research officer in the Oxford University Computing Laboratory. He has worked as a research fellow at the University of Birmingham. He has made a broad range of contributions to the PRISM project, ranging from the underlying theory and model checking algorithms to case studies and advanced implementation techniques.

Catuscia Palamidessi received the PhD degree from the University of Pisa in 1988. She is the director of research at INRIA Futurs, where she leads the team Comète. She worked as a full professor at the University of Genova, Italy (1994-1997) and at the Pennsylvania State University (1998-2002). Her research interests include concurrency, distributed systems, and security. Her past achievements include the proof of expressiveness gaps between various concurrent calculi and the development of a probabilistic version of the asynchronous π-calculus. Her current research is in mobile calculi, probability, and the use of probabilistic concepts in concurrency and security. She has been the program committee chair of various conferences, including MFPS 2008, SOFSEM 2008, ICALP 2005, and CONCUR 2000. She is on the editorial board of *Mathematical Structures in Computer Science* (MSCS), *Theory and Practice of Logic Programming* (TPLP), and *Electronic Notes in Theoretical Computer Science* (ENTCS), and is a member of the Executive Committee of the European Association of Theoretical Computer Science (EATCS).

David Parker received the BSc and PhD degrees from the University of Birmingham. He is a research officer atn the Oxford University Computing Laboratory. Prior to that, he spent five years working as a research fellow at the University of Birmingham. His current research interests include various topics in the field of probabilistic verification, including symbolic model checking, abstraction techniques, and software model checking. He is the lead developer of the PRISM model checker. His doctoral thesis was runner-up in the 2003 BCS Distinguished Dissertation Awards.

Peng Wu received the BSc degree in computer science from the Ocean University of China in 1998 and the MSc and PhD degrees in computer science from the Institute of Software, Chinese Academy of Sciences, in 2001 and 2005, respectively. He is currently a research associate at University College London. His research interests are automatic verification of network systems via model checking.