Model checking the probabilistic π-calculus

Gethin Norman[1], Catuscia Palamidessi[2], David Parker[1] and Peng Wu[3]

[1] Oxford University Computing Laboratory, Wolfson Building, Parks Road, Oxford, OX1 3QD, England
[2] INRIA Futurs and LIX, École Polytechnique, Rue de Saclay, 91128 Palaiseau Cedex, France
[3] CNRS and LIX, École Polytechnique, Rue de Saclay, 91128 Palaiseau Cedex, France
gethin.norman@comlab.ox.ac.uk, catuscia@lix.polytechnique.fr,
david.parker@comlab.ox.ac.uk, wu@lix.polytechnique.fr

Abstract

We present an implementation of model checking for the probabilistic π-calculus, a process algebra which supports modelling of concurrency, mobility and discrete probabilistic behaviour. Formal verification techniques for this calculus have clear applications in several domains, including mobile ad-hoc network protocols and random security protocols. Despite this, no implementation of automated verification exists. Building upon the (non-probabilistic) π-calculus model checker MMC, we first show an automated procedure for constructing the Markov decision process representing a probabilistic π-calculus process. This can then be verified using existing probabilistic model checkers such as PRISM. Secondly, we demonstrate how for a large class of systems a more efficient, compositional approach can be applied, which uses our extension of MMC on each parallel component of the system and then translates the results into a high-level model description for the PRISM tool. The feasibility of our techniques is demonstrated through three case studies from the π-calculus literature.

1 Introduction

The π-calculus [15] is a process algebra for modelling concurrency and mobility. It is well suited to modelling, for example, communication protocols for dynamic network topologies and security protocols. For both classes of systems, probability is often also a key ingredient. Mobile ad-hoc network protocols, for example, can exhibit stochastic behaviour both in terms of communication failures and random back-off procedures. Randomised security protocols are used, for example, to tackle anonymity or contract-signing [7]. The probabilistic π-calculus, which extends the original process algebra with discrete probabilistic choice, has been proposed as a formalism to model and reason about such systems. The benefits for automatic formal verification and tool support in this context are clear: reasoning correctly about the behaviour of such models, particularly interactions between probabilistic and nondeterministic behaviour, is known to be non-trivial. Furthermore, the state spaces of probabilistic 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 the simple probabilistic π-calculus: an extension of the π-calculus which adds a discrete probabilistic choice operator in addition to the existing nondeterministic choice operator. This probabilistic choice is *blind*, in the sense that each probability is associated with a silent τ action. This proves to be sufficiently expressive for modelling the classes of system we are interested in, whilst simplifying the semantics, and thus verification, of the formalism.

Our approach is to adapt and reuse existing tools for verification of mobile systems and of probabilistic systems. We first developed an extension of the tool MMC [26], a logic programming based model checker for the π-calculus. This extension, MMC$_{sp}$, can derive the semantic model for an arbitrary process in the (finite-control) probabilistic π-calculus. The semantic model, which is given by a Markov decision process (MDP), can then be analysed using standard tools, such as the probabilistic model checker PRISM [11]. For efficiency reasons, however, we take a compositional approach, applying MMC$_{sp}$ to each parallel component of a system, processing the results to produce a high-level description in the modelling language of PRISM and then performing probabilistic verification. This avoids a potential blow-up in the size of the intermediate MDP 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 three π-calculus case studies.

Related work. Various tools exist for automatic verification of the (non-probabilistic) π-calculus. The Mobility Workbench (MWB'99) [23] provides a bisimulation checker and a π-μ-calculus model checker. MMC (Mobility Model Checker) [26], 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 [2] 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 [4], which verifies π-calculus processes augmented with type signatures based on an extraction of sound models using types and CCS processes, and [24, 22] which translate a subset of the π-calculus to the language Promela for model checking in the SPIN tool.

A number of existing papers have proposed probabilistic extensions of the π-calculus. The first [10] extended the asynchronous version of the calculus, which removes the output prefix construct, meaning processes must terminate immediately after sending output. In [5], a variant which is essentially the same as that used in this paper was presented and probabilistic testing equivalences were defined to reason about randomised security protocols. In [1], the probabilistic π-calculus was used to formalise definitions of anonymity. To our knowledge, this paper constitutes the first attempt to implement *automated verification* in this area.

Also related are *stochastic* variants of the π-calculus [20] whose semantics are continuous-time Markov chains. A number of associated discrete-event simulators for this formalism are available, (e.g. SPIM, BioSpi) but no model checking tools. Both the stochastic π-calculus and probabilistic model checking techniques have been applied successfully to case studies in the field of systems biology. It is hoped that the techniques proposed in this paper will also prove to be valuable in this domain.

Structure. The remainder of this paper is structured as follows. Section 2 introduces and explains the simple probabilistic π-calculus and its semantics. Sections 3 and 4 describe our extension of MMC for evaluating these semantics and how the result of this can be processed into input for the PRISM tool. Section 5 presents experimental results and Section 6 concludes the paper.

2 The simple probabilistic π-calculus

The π-calculus is a process algebra for modelling concurrency and mobility. Based on the process algebra CCS, a key distinguishing feature of the calculus is that it uses a single datatype, *names*, for both channels and variables, with the consequence that it is possible to communicate channel names between processes. We use a probabilistic extension of the π-calculus called the *simple probabilistic π-calculus* or π_{sp}.

Syntax. We let $\mathcal{N}$ denote a countable set of names, ranged over by x, x_i, y, etc. Using P, P_i to range over terms and α to denote an action, the syntax of the simple probabilistic π-calculus is:

$$\begin{array}{lll} \alpha & ::= & \tau \;\big|\; x(y) \;\big|\; \bar{x}y \\ P & ::= & 0 \;\big|\; \alpha.P \;\big|\; \sum_{i\in I} P_i \;\big|\; \sum_{i\in I} p_i\tau.P_i \;\big| \\ & & P \mid P \mid \nu x P \;\big|\; [x=y]P \;\big|\; A(y_1,\ldots,y_n) \end{array}$$

where I is an index set, $p_i \in (0,1]$ with $\sum_{i\in I} p_i = 1$ and $A(x_1,\ldots,x_n)$ is a process definition.

Intuitively, the operators of the calculus are described as follows. The inactive process, denoted 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 $\sum_{i\in I} p_i\tau.P_i$. 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. Notice that branches of the probabilistic choice operator are always prefixed with τ actions, indicating that $\sum_{i\in I} p_i\tau.P_i$ randomly selects an index $i \in I$ with probability p_i, performs a τ action and then evolves as process P_i. This restricted form of probabilistic choice is in practice sufficiently expressive but simplifies semantics and analysis.

Parallel composition $P_1 \mid P_2$ means that processes P_1 and P_2 can either proceed asynchronously or interact though matching input/output actions. The restriction $\nu x P$, localises 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 to process P only if the names x and y are identical. Finally, $A(y_1,\ldots,y_n)$ is a recursively defined process with a definition clause of the form $A(x_1,\ldots,x_n) \triangleq P$.

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.

A *substitution* σ is a mapping from $\mathcal{N}$ to $\mathcal{N}$. The simplest substitutions are of the form $\{y/x\}$ which map x to y and all other names to themselves. We use the notation $P\sigma$ to denote the term obtained from P by substituting names according to σ. A substitution σ satisfies the match $[x = y]$, denoted $\sigma \models [x = y]$ if $\sigma(x) = \sigma(y)$. Satisfaction extends to conjunctions of matches in the obvious way.

In order to facilitate model checking of probabilistic π-calculus processes, we make two simple assumptions.

Firstly, we restrict our attention to the *finite-control* version of the calculus, 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. Secondly, we require that the systems to which we apply model checking are *input-closed* π-calculus processes, meaning that they require no inputs from the environment.

Symbolic semantics. The operational semantics for probabilistic extensions of the π-calculus are typically expressed in terms of MDPs or, equivalently, probabilistic automata [21], which allow both probabilistic and nondeterministic behaviour. In this paper, we use a *symbolic* presentation of the operational semantics [25]. 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.

Consider the simple process $a(x).\bar{x}b$ 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 immediately establish the first step of this process, i.e. that it inputs x on a. Subsequent behaviour, however, is dependent on the actual input to x, and can only be determined once the process is composed with another which sends output on a. A symbolic approach allows the semantics of a process to include variables (e.g. x) which can be used in actions (e.g. $\bar{x}b$). This allows a compositional approach to be adopted: given a parallel composition of several processes, the semantics of each can be computed in full separately, and then composed afterwards.

The symbolic semantics of the π_{sp} calculus are expressed in terms of *probabilistic symbolic transition graphs* (PSTGs). These are a simple probabilistic extension of the *symbolic transition graphs* of [9], previously used for the (non-probabilistic) π-calculus [12, 3, 13, 14] and for CCS [9]. Alternatively, they can be seen as a symbolic extension of Markov decision processes.

Probabilistic symbolic transition graphs. Let $\mathcal{N}$ be a countable set of names and P be a π_{sp} process. The probabilistic symbolic transition graph (PSTG) for P is a tuple $(S, s_{init}, \mathcal{T})$ 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;
- $\mathcal{T} \subseteq S \times Cond \times Act \times Dist(S)$ is the set of *probabilistic symbolic transitions* and is given by the rules in Figure 1.

In the above,

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

We use the notation $Q \xrightarrow{M,\alpha} \{p_i : Q_i\}_i$ for the probabilistic symbolic transition $(Q, M, \alpha, \mu) \in \mathcal{T}$ where $\mu(R) = \sum_{Q_i=R} p_i$ for any π_{sp} term R. We abbreviate $Q \xrightarrow{M,\alpha} \{1 : Q'\}$ to $Q \xrightarrow{M,\alpha} Q'$.

A symbolic state Q encodes a set of π_{sp} terms. More specifically, it encodes the set of terms derivable from Q by substitutions applied to its input-bound names. For example the symbolic state $Q = a(x).\bar{x}b$ represents the terms $Q\{z/x\}$ for any name z. Of the four action types in Act the first three types 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 variable y.

A 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 α and then with probability p_i evolve as process $Q_i\sigma$. Formally, we have the following Lemma which relates the symbolic (PSTG) and concrete (MDP) semantics of π_{sp}. This corresponds to Lemma 2.4 in [13] for the (non-probabilistic) π-calculus and can be proved in similar fashion.

Lemma 1. Let P be a π_{sp} term.

(a) If $P \xrightarrow{M,\alpha} \{p_i : P_i\}_i$, then for any substitution σ such that $\sigma \models M$ with $bn(\alpha) \cap (fn(P) \cup n(\sigma)) = \emptyset$, $P\sigma \xrightarrow{\alpha\sigma} \{p_i : P_i\sigma\}_i$.

(b) If $P\sigma \xrightarrow{\alpha} \{p_i : P_i\}_i$, then $P \xrightarrow{M,\beta} \{p_i : P'_i\}_i$ where $\sigma \models M$, $\alpha = \beta\sigma$ and $P_i = P'_i\sigma$.

3 Generating PSTGs using MMC

In this section we describe the automatic generation of the probabilistic symbolic transition graph (PSTG) for an arbitrary process expressed in the simple probabilistic π-calculus. This is achieved with an extension of the (non-probabilistic) π-calculus model checker MMC [26], which from this point on we refer to as MMC_{sp}.

MMC_{sp} is based on only a subset of MMC's functionality: essentially the capability to construct the full set of reachable states of a process. The restrictions placed on the syntax of the calculus are the same that we impose, as

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

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

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

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

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

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

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

described 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 handle names [26]. It is implemented in the logic programming system XSB, which is a dialect of Prolog.

With π-calculus names represented by XSB variables, the symbolic semantics of the calculus can be directly encoded into XSB rules. This has several benefits: firstly it gives a clear and intuitive implementation; secondly, and more importantly, this encoding is provably correct [26].

Our implementation, MMC$_{sp}$, is a direct extension of this approach. We have a straightforward encoding of the syntax of π_{sp} into the language of XSB, with π_{sp} names and process identifiers represented by XSB variables and constants, respectively. We then adapt MMC's predicate `trans` to represent the symbolic semantics of π_{sp}. Letting function f_ρ denote the one-to-one mapping of π_{sp} conditions, actions and processes from XSB syntax to π_{sp} syntax, then a tuple `trans(P, PSteps, M)` in XSB, where `PSteps` is a list of compound structures $\mathtt{psteps}(\mathtt{p_i},\ \mathtt{act},\ \mathtt{P_i})$, represents the symbolic probabilistic transition:

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

The full definition of this encoding (the syntax of π_{sp} and the function f_ρ) can be found in the technical report version of this paper [16]. To relate this to the original version of MMC, observe that a tuple `trans(P, [psteps(1, act, Q)], M)` is equivalent to the definition `trans(P, act, M, Q)` in [26]. 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_{sp}, respectively. The proof details are similar to Theorem 2 and 3 in [26].

Finally, we add an extra XSB predicate `stg(P)`, which uses query-evaluation on `trans` to derive the PSTG of process `P` and output it in a simple textual format. This is effectively a depth-first traversal of the PSTG and enumeration of all states and probabilistic symbolic transitions found. This is also included in [16]. In Figure 2, we show the application of MMC$_{sp}$ to the simple π_{sp} process $Toss$:

$$Toss(x) \triangleq x(y).(p\tau.\bar{y}head.0 + (1-p)\tau.\bar{y}tail.0)$$

which receives a name y on channel x and then sends out, on channel y, either $head$ or $tail$, with probability p or $1-p$, respectively. In the output of the tool, lines starting `#`i show the π_{sp} term for the ith state, lines starting `*`j and `'`k enumerate transitions and the individual edges of transitions, respectively. All bound names are given unique names (e.g. `_h417`) and displayed on lines beginning >. All free names used are listed at the end of the PSTG.

```
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
```

Figure 2. Sample output from MMC$_{sp}$

4 Translating PSTGs into PRISM

The scheme described in the previous section can be used to translate an arbitrary process described in the simple probabilistic π-calculus into its probabilistic symbolic transition graph (PSTG). Since for an input-closed π_{sp} term its PSTG and concrete semantics (MDP) coincide, one can directly map the PSTG into PRISM to enable model checking of the π_{sp} term. For efficiency, however, we adopt where possible a compositional approach.

More specifically, in the case where systems are of the form $P = \nu x_1 \dots \nu x_k(P_1|\dots|P_n)$ and each P_i contains no instances of the ν operator, the basic idea is compute the PSTG for each subprocess P_i, as described in the previous section, map each PSTG to a PRISM module, and then use PRISM to construct the semantics of P through the parallel composition of these modules. At the level of PSTGs, our restricted form ensures that there are no bound output transitions from each P_i.

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 modelled as a parallel composition of multiple components and, since we assume that P is input-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 $P_1|\nu x P_2$ and $\nu x(P_1|P_2)$ 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 recursion. 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 cases we can resort to the basic approach: use MMC$_{sp}$ to construct the PSTG for the full system and import this directly into PRISM.

There are two principal challenges regarding the translation of a set of PSTGs 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 multi-way (CSP-style) synchronisation without value passing. In brief, (1) is handled by enumerating the set of all free names (which is known since the system is input-closed), assigning each an (identically named) integer constant to represent it, and (2) is handled by introducing a synchronous 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 process with bound input variable x, an identically named local (integer) variable used to represent a name.

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.

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 V. The behaviour of each module is defined by a set of guarded commands of the form:

$$[act]\ guard \rightarrow p_1 : u_1 + \dots + p_m : u_m;$$

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:

$$(x'_1 = u_{i,1})\ \&\ \dots\ \&\ (x'_k = u_{i,k})$$

where $u_{i,j}$ is a function over V. Intuitively, in global state s of the PRISM model, a command in a module is available if $s \models guard$. 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 $u_{i,j}(s)$. In practice (see for example Figure 3), we omit probabilities equal to one and update-components 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 (i.e. modules synchronise over all their common actions). The full semantics of the PRISM language can be found at [19].

Example. Consider the following parallel composition of two processes:

- $Q \triangleq \nu a\,(Q_1 \mid Q_2)$
- $Q_1 \triangleq \nu c\;\nu d\,\left(\frac{1}{2}\tau.\bar{a}c.c(v).0 + \frac{1}{2}\tau.\bar{a}d.d(w).0\right)$
- $Q_2 \triangleq \nu b\,\left(a(x).\bar{b}x.0 \;\middle|\; b(y).\bar{y}e.0\right)$

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 re-outputs 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:

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

The corresponding PSTGs are:

- $P_1:$ $$\begin{tikzcd} P_{1,1} \arrow[r, "\tau"] & \bullet \arrow[r, "\frac{1}{2}"] \arrow[dr, "\frac{1}{2}"'] & P_{1,2} \arrow[r, "\bar{a}c"] & P_{1,4} \arrow[r, "c(v)"] & P_{1,6} \\ & & P_{1,3} \arrow[r, "\bar{a}d"] & P_{1,5} \arrow[ur, "d(w)"] & \end{tikzcd}$$
- $P_2:\; P_{2,1} \xrightarrow{a(x)} P_{2,2} \xrightarrow{\bar{b}x} P_{2,3}$
- $P_3:\; P_{3,1} \xrightarrow{b(y)} P_{3,2} \xrightarrow{\bar{y}e} P_{3,3}$

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 total set of free names is $\{a, b, c, d, e\}$. The resulting PRISM model is shown in Figure 3. This example will be referred to in the full explanation of the translation given below.

```
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
```

Figure 3. PRISM code for the example

Formal translation. We assume that the set of all names in the system is $\mathcal{N}$, which is partitioned into disjoint subsets: $\mathcal{N}_f$, the set of all free names, and $\mathcal{N}_{b,1}, \ldots, \mathcal{N}_{b,n}$, 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_{sp} terms and the resulting PRISM language description. Thus, each of the n subprocesses (or PSTGs) P_i becomes a PRISM module P_i and the (finite) set of π_{sp} terms $S_i = \{Q_{i,1}, \ldots, Q_{i,k_i}\}$ that constitute the states of the PSTG become a set of integer indices $Q_{i,1}, Q_{i,2}, \ldots$ uniquely representing each one.

Module P_i has $|\mathcal{N}_{b,i}| + 1$ local variables: its local state (i.e. state of corresponding PSTG) is represented by variable s_i, with range $1, \ldots, |S_i|$, and each bound name $x_{i,j} \in \mathcal{N}_{b,i}$ has a corresponding variable $x_{i,j}$ with range $0, \ldots, |\mathcal{N}_f|$. The model also includes $\mathcal{N}_f$ integer constants, one for each free name in the system, which are assigned (in some arbitrary order) distinct, consecutive non-zero values. If the value of variable $x_{i,j}$ 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 $x_{i,j}{=}0$, no input to the bound name has occurred yet.

In this way, the conditions which label transitions of PSTGs can be translated directly into PRISM. For example, let condition M be $(x{=}a)\wedge(y{=}b)$ where x, y are bound names and a, b free names. The translation into PRISM is identical: $(x{=}a)\wedge(y{=}b)$, where x, y are integer variables and a, b integer constants.

For each symbolic probabilistic transition $Q_i \xrightarrow{M,\alpha} \{p_1 : R_{i,1}, \ldots, p_m : R_{i,m}\}$ in the PSTG for P_i, we will include a set of corresponding PRISM commands in the module P_i. We consider each type of transition separately, beginning with the case where $\alpha = \tau$.

Case 1 (internal action). For a transition:

$$Q_i \xrightarrow{M,\tau} \{p_1 : R_{i,1}, \ldots, p_m : R_{i,m}\}$$

we add the command:

$$[]\ (s_i{=}Q_i)\ \&\ M \rightarrow p_1{:}(s_i'{=}Q_{i,1}) + \cdots + p_m{:}(s_i'{=}Q_{i,m});$$

See Figure 3 line 7 for an example. This type of transition is in fact the only one which can actually include multiple probabilistic choices. The remaining types of transitions (input and output) are always of the form $Q_i \xrightarrow{M,\alpha} R_i$ (this fact can be derived easily from the semantics in Figure 1).

Case 2 (output on free name). For a transition:

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

we add, for each $j \in \{1, ..., n\}\backslash\{i\}$, the command:

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

The channel x, sender P_i, receiver P_j and sent name y are all encoded in the action label. See Figure 3 lines 8 and 18 for examples of sending free and bound names y, respectively.

Case 3 (output on bound name). For a transition:

$$Q_i \xrightarrow{M,\bar{x}y} R_i \text{ where } x \in \mathcal{N}_{b,i}$$

we add, for each $a \in \mathcal{N}_f$ and $j \in \{1, ..., n\}\backslash\{i\}$:

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

This is similar to Case 2 except that we include a command for each possible value a of x. See for example lines 24 and 25 of Figure 3.

Case 4 (input on free name). For a transition:

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

we add, for each $y \in \mathcal{N}\backslash\mathcal{N}_{b,i}$ and $j \in \{1, ..., n\}\backslash\{i\}$:

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

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 Figure 3 which match the output commands from lines 8 and 9.

Case 5 (input on bound name). For a transition:

$$Q_i \xrightarrow{M,x(z)} R_i \text{ where } x \in \mathcal{N}_{b,i}$$

we add, for $a \in \mathcal{N}_f$, $y \in \mathcal{N}\backslash\mathcal{N}_{b,i}$ and $j \in \{1, ..., n\}\backslash\{i\}$:

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

Case 5 combines elements of Cases 3 and 4: 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 4 and 5, since they correspond to input actions which will never occur. More precisely, for each module P_j we identify action labels $x\_P_i\_P_j\_y$ which appear on a command in module 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.

Correctness of the translation. By assumption the π_{sp} term being translated is of the form $P = \nu x_1 \ldots \nu x_k(P_1| \ldots |P_n)$. The first step in the proof is to show that any term in the derivation tree of P is of the form $\nu x_1 \ldots \nu x_k(Q_1\sigma_1 \,|\, \cdots \,|\, Q_n\sigma_n)$ where, for any $1{\le}j{\le}n$, Q_j is a state of the PSTG for the process P_j and σ_j is a substitution from the bound names of P_j to the free names of P. The proof is by induction of the transition rules (concrete) and using Lemma 1.

We now show that the translation is correct by constructing a mapping between such π_{sp} terms and the states of the PRISM model and demonstrating that, for any π_{sp} 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 transition. For any π_{sp} term $\nu x_1 \ldots \nu x_k(Q_1\sigma_1 \,|\, \cdots \,|\, Q_n\sigma_n)$ the state in the PRISM model is constructed as follows: for any $1 \le j \le n$, the values of the variables of module P_j are given by $s_j = Q_j, x_{j,1} = i_{j,1}, \ldots, x_{j,k_j} = i_{j,k_j}$ where if $\sigma(x_{j,l}) = z \in \mathcal{N}_f$, then $i_{j,l}$ is the integer constant corresponding to the free variable z and otherwise (i.e. $\sigma(x_{j,l}) = x_{j,l}$) $i_{j,l}$ equals 0.

Next consider any π_{sp} term P' in the derivation tree and transition:

$$Q = \nu x_1 \ldots \nu x_k(Q_1\sigma_1 \,|\, \cdots \,|\, Q_n\sigma_n) \xrightarrow{\tau} \{p_m : R_m\}_m.$$

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

Silent actions. $Q_j\sigma_j \xrightarrow{\tau} \{p_m : R^j_m\sigma_j\}_m$, $1 \le j \le n$, and $R_m = \nu x_1 \ldots \nu x_k(Q_1\sigma_1 \,|\, \ldots \,|\, R^j_m\sigma_j \,|\, \ldots \,|\, Q_n\sigma_n)$. From Lemma 1(b), we have $Q_j \xrightarrow{M_j,\tau} \{p_m : R^j_m\}$ and $\sigma_j \models M_j$, and 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_{sp} 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. $Q_j\sigma_j \xrightarrow{x(z)} R_j\sigma_j$, $Q_l\sigma_l \xrightarrow{\bar{x}y} R_l\sigma_l$, $1 \le j \ne l \le n$, $a = \tau$ and $\{p_m : R_m\}_m = \{1 : R\}$ where $R = \nu x_1 \ldots \nu x_k(Q_1\sigma_1 \,|\, \ldots \,|\, R_j\sigma_j\{y/z\} \,|\, \ldots \,|\, R_l\sigma_l \,|\, \ldots \,|\, Q_n\sigma_n)$. Now, from Lemma 1(b) we have $Q_j \xrightarrow{M_j,x(z)} R_j$ and $Q_l \xrightarrow{M_l,\bar{x}y} R_l$, $\sigma_j \models M_j$ and $\sigma_l \models M_l$. Considering the case when x is a free name (the case when x is bound follows similarly), since $y \in \mathcal{N} \backslash \mathcal{N}_{b,j}$ in the modules P_j and P_l we have the commands

$$[x\_P_l\_P_j\_y]\ (s_j{=}Q_j)\ \&\ M_j \rightarrow (s_j'{=}R_j)\ \&\ (z'{=}y);$$
$$[x\_P_l\_P_j\_y]\ (s_l{=}Q_l)\ \&\ M_k \rightarrow (s_l'{=}R_l);$$

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

To complete the proof it remains to show that any transition of the PRISM model is matched by a transition in the corresponding π_{sp} term. The result follows in a similar manner to the above using Lemma 1(a) instead of Lemma 1(b).

Optimisations. The translation from PSTGs to PRISM code described in this section can be optimised to reduce the size of the generated code and the resulting model. The basic idea is to compute an over-approximation of the possible values that each PSTG'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 over-approximation 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 Figure 3 has been reduced in this way.

Properties. Simple probabilistic reachability properties, such as the maximum probability of failure or the minimum probability of message delivery, can be encoded simply through the availability or absence of actions, as such properties are preserved in the translation to PRISM. For example, in the case of system failure, one would modify the original π-calculus description by adding to any π-calculus process term representing system failure the possibility to output on a new distinct channel/action to allow one to identify the PRISM states representing system failure as those states where this new action is available. Once these states have been identified, one can construct a PCTL formula which when verified will return either the maximum or minimum probability of reaching this set of states, that is calculate the maximum or minimum of system failure. 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 [8]. Model checking for specification formalisms more specifically tailored to the mobile aspects of the π-calculus, such as spatial logic, will be an area of future work.

5 Implementation and results

Our implementation of model checking for the simple probabilistic π-calculus is fully automated and comprises three parts: (1) MMC_{sp}, an extension of MMC (as described in Section 3), which constructs the probabilistic symbolic transition graphs (PSTGs) for one or more π_{sp} processes, (2) the translator from PSTGs to PRISM code (as described in Section 4), implemented in Java, and (3) the probabilistic model checker PRISM [11] which builds the MDP from part (2) and performs verification of PCTL properties. We based our implementation on MMC 1.0 and PRISM 3.1.

Firstly, we consider the dining cryptographers protocol (**DCP**) [6], Chaum's randomised 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 $N-1$ non-paying parties knowing the identity of the paying one. This was previously modelled in the probabilistic π-calculus in [1]. 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.

Secondly, we study the partial secret exchange (**PSE**) algorithm of [7] for anonymous contract signing between two parties. A probabilistic π-calculus model of PSE was given in [5]. The protocol was independently analysed in PRISM [17], 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 lower probability this occurring. We used a π_{sp} model of both the original and modified versions to demonstrate the same flaw.

Thirdly, we constructed a π_{sp} model of mobile communication network (**MCN**), based on the (non-probabilistic) π-calculus model in [18]. The system comprises N base stations with fixed communication links to a mobile switching centre and a mobile station which can be connected to each of the base stations via radio links. The mobile station roams between the base 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 analysed using MMC in [26]. In both this and the original paper, though, the occurrence of a failure during the handover protocol was modelled as a nondeterministic choice. We are able to model this correctly, as a random event. We check the probability of a handover operation completing

Model	N	States	Transitions	MTBDD nodes	Construction time (sec.)			Model checking time (sec.)
					PSTGs	PRISM	MDP	
DCP	5	160,543	592,397	58,641	10.9	0.81	0.77	2.49
	6	1,475,401	6,520,558	100,290	13.1	0.91	1.43	7.82
	7	13,221,889	68,121,834	154,500	15.2	1.17	2.62	21.3
	8	116,192,457	683,937,352	221,170	18.1	1.21	4.72	55.2
	9	1,005,495,499	6,657,256,911	463,425	19.1	1.37	19.3	732.9
PSE	3	9,321	32,052	37,008	4.86	0.75	1.60	1.89
	4	89,025	419,172	103,779	6.60	0.91	3.95	4.47
	5	837,361	5,028,700	173,644	8.12	1.20	8.47	11.5
PSE_3	3	9,328	32,059	37,251	5.29	0.75	2.38	2.16
	4	89,040	419,187	104,267	6.69	0.96	4.19	13.8
	5	837,392	5,028,731	175,212	7.82	1.13	7.58	52.4
MCN	2	609	950	58,430	4.33	2.49	4.8	1.17
	3	3,611	5,811	216,477	5.89	3.11	22.4	5.24

Table 1. Performance of probabilistic model checking π_{sp} on three case studies

successfully, within a given number of communications.

Table 1 shows the performance of our implementation on the three case studies. Experiments were run on a 2 GHz PC with 512 MB RAM running Linux. For the DCP model, we varied the number of parties N; for the PSE model, we considered two variants (the original protocol EGL and the modified version EGL3 from [17]) and varied the size of contract N. For the MCN model, we varied the number of base stations N. The table shows the size of the resulting MDPs (number of states/transitions) and corresponding storage in PRISM (MTBDD nodes, where 1 node uses 20 bytes). We also give the time required for each stage of the process, i.e. constructing the PSTGs (using MMC_{sp}), the PRISM code (using the translator) and the MDP (using PRISM). Finally, we give the time to check a single (quantitative) PCTL property for each using PRISM (with the MTBDD engine).

The results are very encouraging. We see that our techniques are scalable to the construction and analysis of π_{sp} models with extremely large state spaces. Furthermore the times required for all stages of the process are relatively small. The MCN case study, although smallest in terms of state space, is perhaps the best example of the applicability of this implementation since it fully exploits the 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 the probabilistic π-calculus. The variant of the calculus (with blind probabilistic choice) to which our techniques are applicable has proved to be expressive enough for the appropriate application domains (probabilistic algorithms for security and dynamic communication protocols with failures and/or randomisation) and yet amenable to analysis with extensions and adaptions of existing verification tools. Furthermore we have shown, through its application to several large examples, the efficiency of the approach.

We would like to extend this work in several directions. For convenience of modelling, we plan to add support for polyadic communication over channels. We also hope to add support for more flexible property specifications using watchdog processes and to extend our approach to the stochastic π-calculus. 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 optimisations to reduce the resulting symbolic (MTBDD) storage in PRISM and bisimulation minimisation techniques.

Acknowledgements

Authors Norman and Parker were in the School of Computer Science at the University of Birmingham when this work was first carried out. They are supported in part by EPSRC grants GR/S11107 and GR/S46727 and Microsoft Research Cambridge contract MRL 2005-44. Authors Palamidessi and Wu are supported in part by the INRIA/ARC project ProNoBis.

References

[1] M. Bhargava and C. Palamidessi. Probabilistic anonymity. In *Proc. CONCUR'05*, volume 3653 of *LNCS*, pages 171–185. Springer, 2005.

[2] B. Blanchet. ProVerif: Automatic cryptographic protocol verifier user manual. 2005.

[3] M. Boreale and R. D. Nicola. A symbolic semantics for the π-calculus. *Information and Computation*, 126(1):34–52, April 10 1996.

[4] S. Chaki, S. Rajamani, and J. Rehof. Types as models: Model checking message-passing programs. In *Proc. POPL'02*, pages 45–57. ACM, 2002.

[5] K. Chatzikokolakis and C. Palamidessi. A framework to analyze probabilistic protocols and its application to the partial secrets exchange. In *Proc. International Symposium on Trustworthy Global Computing (TGC'05)*, 2005.

[6] D. Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1:65–75, 1988.

[7] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28(6):637–647, 1985.

[8] M. Goldsmith, N. Moffat, B. Roscoe, T. Whitworth, and I. Zakiuddin. Watchdog transformations for property-oriented model-checking. In *Proc. FME'03*, volume 2805 of *LNCS*. Springer, 2003.

[9] M. Hennessy and H. Lin. Symbolic bisimulations. *Theoretical Computer Science*, 138:353–389, 1995.

[10] O. Herescu and C. Palamidessi. Probabilistic asynchronous π-calculus. In *Proc. FOSSACS'00*, volume 1784 of *LNCS*, pages 146–160. Springer, 2000.

[11] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In *Proc. TACAS'06*, volume 3920 of *LNCS*, pages 441–444. Springer, 2006.

[12] H. Lin. Symbolic bisimulation and proof systems for the π-calculus. Technical report, School of Cognitive and Computer Science, University of Sussex, 1994.

[13] H. Lin. Computing bisimulations for finite-control π-calculus. *Journal of Computer Science and Technology*, 15(1):1 – 9, 2000.

[14] H. Lin. Complete inference systems for weak bisimulation equivalences in the pi-calculus. *Information and Computation*, 180(1):1–29, 2003.

[15] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes. *Information and Computation*, 100:1–40, 1992.

[16] 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, University of Birmingham, 2007.

[17] G. Norman and V. Shmatikov. Analysis of probabilistic contract signing. *Journal of Computer Security*, 14(6):561–589, 2006.

[18] F. Orava and J. Parrow. An algebraic verification of a mobile network. *Formal Aspects of Computing*, 4:497–543, 1992.

[19] Online PRISM documentation www.prismmodelchecker.org/doc/.

[20] C. Priami. Stochastic π-calculus. *The Computer Journal*, 38(7):578–589, 1995.

[21] R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2):250–273, 1995.

[22] H. Song and K. Compton. Verifying π-calculus processes by Promela translation. Technical Report CSE-TR-472-03, University of Michigan, 2003.

[23] B. Victor and F. Moller. The Mobility Workbench — a tool for the π-calculus. In *Proc. CAV'94*, volume 818 of *LNCS*, pages 428–440. Springer, 1994.

[24] P. Wu. Interpreting π-calculus with Spin/Promela. *Computer Science*, 8:7–9, 2003. Supplement.

[25] P. Wu, C. Palamidessi, and H. Lin. Symbolic bisimulations for probabilistic systems. In *Proc. QEST'07*, 2007.

[26] P. Yang, C. Ramakrishnam, and S. Smolka. A logic encoding of the π-calculus: model checking mobile processes using tabled resolution. *International Journal on Software Tools Technology Transfer*, 4:1 – 29, 2004.