Model checking the probabilistic π-calculus

Gethin Norman
Oxford University
Computing Laboratory

Catuscia Palamidessi
INRIA Futurs Saclay
LIX

Dave Parker
Oxford University
Computing Laboratory

Peng Wu
CNRS
LIX

QEST'07
Overview

• Probabilistic model checking
  – Markov decision processes, PCTL, PRISM

• The probabilistic $\pi$-calculus
  – syntax, symbolic semantics, example

• $\pi$-calculus tool support: MMC

• Adding $\pi$-calculus support to PRISM
  – extending MMC with probabilities
  – a compositional approach: translation to PRISM

• Experimental Results

• Conclusions
Probabilistic model checking

- Automatic formal verification technique for analysis of systems exhibiting probabilistic behaviour

High-level description, e.g. in PRISM modelling language

Property
- e.g. PCTL formula

Probabilistic model
- e.g. MDP (Markov decision process)

Probabilistic model checker
- e.g. PRISM

Results
Markov decision processes (MDPs)

- Model supporting probabilistic and nondeterministic choice
  - discrete state space and discrete time-steps
  - nondeterministic choice between
    (action-labelled) probability distributions over successor states
- Well suited to modelling of:
  - randomised distributed algorithms, probabilistic communication/security protocols, ...
- Verification using e.g. the logic PCTL
  - $P_{\min=?} \left[ F^{\leq t} \text{reply\_count} = k \{ \text{“init”} \}\{\text{min}\} \right]$
  “what is the minimum probability, from any initial configuration and under any scheduling, that the sender has received $k$ acknowledgements within $t$ time units?”

QEST'07
PRISM modelling language

- Simple, state-based language for MDPs (and D/CTMCs)
  - based on Reactive Modules [Alur/Henzinger]
- Modules (system components, composed in parallel)
- Variables (finite-valued – integer ranges or booleans)
- Guarded commands (labelled with probabilities/rates)
- Composition of modules: synchronisation (CSP-style) +
  process-algebraic operators (e.g. action hiding/renaming)

\[
\text{[send]} \ (s=2) \rightarrow \ p_{\text{loss}} : (s'=3) \& (\text{lost}'=\text{lost}+1) + (1-p_{\text{loss}}) : (s'=4) ;
\]

QEST'07
The π-calculus

• The π-calculus [Milner et al.]
  – process algebra for concurrency and mobility
  – single datatype, names, for both channels and variables
  – allows dynamic creation of new channel names and communication of channel names between processes
  – ...and therefore dynamic communication topologies
  – applications: e.g. cryptographic protocols, mobile communication protocols, ...

• Probabilistic π-calculus [Herescu/Palamidessi, ...]
  – adds discrete probabilistic choice for modelling of random choice (e.g. coin toss) or unpredictability (e.g. failures)
  – applications: e.g. randomised security protocols, mobile ad-hoc network protocols, ...
Simple probabilistic π-calculus: $\pi_{sp}$

[Chatzikokolakis/Palamidessi]

- **Processes:** $P ::=$
  - $0 \mid \alpha.P \mid P + P \mid \Sigma_i p_i \tau.P_i$
    - (null) (prefix) (nondet. choice) (internal probabilistic choice)
  - $P \mid P \mid Vx.P \mid [x=y]P \mid A(y_1,\ldots,y_n)$
    - (parallel) (restriction) (match) (identifier)

- **Actions:** $\alpha ::=$
  - $in(x,y) \mid out(x,y) \mid \tau$
    - (input on x to y) (output of y on x) (internal)

- **Example:** $Q ::= V\alpha (Q_1 \mid Q_2)$
  - $Q_1 ::= Vc Vd ( \frac{1}{2} \tau.out(a,c).in(c,v).0 + \frac{1}{2} \tau.out(a,d).in(d,w).0 )$
  - $Q_2 ::= Vb ( in(a,x).out(b,x).0 \mid in(b,y).out(y,e).0 )$

QEST'07
Simple probabilistic π-calculus: \( \pi_{sp} \)

- **“Simple”** refers to restriction to “blind” probabilistic choice
  - “sufficient” modelling power, but simpler semantics/analysis
- **Restrictions for model checking**
  - finite control (no recursion within parallel composition)
  - input closed (no inputs from environment)
- **Semantics are in terms of Markov decision processes**
  - or, equivalently, (simple) probabilistic automata [Segala/Lynch]
- **We use a symbolic semantics approach**
  - often better suited to proof systems, tool support
  - extension of non-probabilistic case [Lin'00,Lin'03]
  - probabilistic symbolic transition graphs (PSTGs)
Symbolic semantics

• **A PSTG is a tuple** \((S, s_{\text{init}}, T)\) **where:**
  
  − \(S\) is a set of symbolic states (\(\pi\)-calculus processes)
  − \(s_{\text{init}} \in S\) is the initial state
  − \(T \subseteq S \times \text{Cond} \times \text{Act} \times \text{Dist}(S)\) are transitions

• **And:**
  
  − \(\text{Cond}\) is the set of conditions
    · finite conjunctions of matches (name comparisons)
  − \(\text{Act}\) is the set of actions:
    · \(\tau, \text{in}(x,y), \text{out}(x,y), \text{b\_out}(x,y)\)
      for names \(x, y\)

For a transition:

\[
( Q, M, \alpha, \{ p_i : Q_i \} ) \in T
\]

written:

\[
M,\alpha
\]

\[
Q \rightarrow \{ p_i : Q_i \}
\]

“If \(M\) is true, \(Q\) can perform action \(\alpha\) and then with probability \(p_i\) evolve as \(Q_i\)”
Symbolic semantics

- A PSTG is a tuple \((S, s_{\text{init}}, T)\) where:
  - \(S\) is a set of symbolic states (\(\pi\)-calculus processes)
  - \(s_{\text{init}} \in S\) is the initial state
  - \(T \subseteq S \times \text{Cond} \times \text{Act} \times \text{Dist}(S)\) are transitions

- And:
  - \(\text{Cond}\) is the set of conditions
    - finite conjunctions of matches (name comparisons)
  - \(\text{Act}\) is the set of actions:
    - \(\tau, \text{in}(x,y), \text{out}(x,y), \text{b_out}(x,y)\)
      for names \(x, y\)

Example:

\[
\frac{1}{2} \tau.\text{out}(a,c).\text{in}(c,v).0 \\
+ \frac{1}{2} \tau.\text{out}(a,d).\text{in}(d,w).0
\]

\[
\quad \frac{1}{2} \quad \frac{1}{2}
\]

\[
\text{out}(a,c).\text{in}(c,v).0 \quad \text{out}(a,d).\text{in}(d,w).0
\]

\[
\text{in}(c,v).0 \quad \text{in}(d,w).0
\]

\[
\text{in}(c,v) \quad 0 \quad \text{in}(d,w)
\]

\[
\text{out}(a,c) \quad 1 \quad \text{out}(a,d)
\]

(empty) conditions omitted
MMC: Mobility Model Checker

• Model checker for (finite control subset of) π-calculus
  – against alternation-free π-μ-calculus

• Efficient implementation based on logic programming (XSB)
  – names in π-calculus are represented as LP variables
    • semantics of names matches variable handling in LP resolution
  – direct LP encoding of π-calculus symbolic semantics
    • efficient (XSB tabled resolution) and provably correct

• Other features of MMC:
  – identifies (some) state equivalences (structural congruence)
  – symmetry reduction: associativity/commutativity of parallel
  – additional support for spi-calculus
Translation – Part 1

• \( \text{MMC}_{sp} \): extension of MMC to support \( \pi_{sp} \)
  – add probabilistic version of choice operator
    • direct encoding of semantics, as for other operators
    • modify “trans” rule of MMC to include (textual) probabilities
  – add explicit generation/export of PSTG
  – also identifies free/bound names

• For input–closed process, direct input into PRISM
  – PSTG for input–closed process is an MDP
  – either: encode as a single module in PRISM language
  – or: direct input of transition matrix into PRISM

• Provides translation for any \( \pi_{sp} \) process
Translation – Part 2

• **Problems:**
  – for large models, enumerating state space in this way inefficient
  – product state-space blow-up (at language level)
  – lack of structure/regularity in model (and hence large MTBDDs)

• **Solution: a compositional approach to translation**
  – 1. assume process of form: $P := \nu x_1 \ldots \nu x_k (P_1 | \ldots | P_n)$
    - where each $P_i$ contains no instances of $\nu$ operator
    - can use structural congruence to get process in this form
  – 2. generate PSTG for each subprocess $P_i$ (using MMC$_{sp}$)
  – 3. translate set of $n$ PSTGs into $n$ PRISM modules
  – 4. final PRISM model is composition of $n$ modules
Translation to PRISM

• **Construction of PRISM module for subprocess** $P_i$:
  - one local variable for state (program counter)
  - one local variable per name bounded by input
  - transitions of the PSTG for $P_i$ translated to PRISM commands

• **Map names datatype into PRISM's (basic) type system**
  - integer variables, integer constant for each free name

• **Model channel communication in PRISM**
  - $\pi$-calculus: binary synchronisation (CCS), name passing
  - PRISM: multi-way synchronisation (CSP), no value passing
  - our translation: encode all information in action names
Example

- $Q := \nu a (Q_1 \mid Q_2)$
  - $Q_1 := \nu c \nu d \left( \frac{1}{2} \tau.\text{out}(a,c).\text{in}(c,v).0 + \frac{1}{2} \tau.\text{out}(a,d).\text{in}(d,w).0 \right)$
  - $Q_2 := \nu b \left( \text{in}(a,x).\text{out}(b,x).0 \mid \text{in}(b,y).\text{out}(y,e).0 \right)$

- Rewrite process $Q$ as structurally congruent process $P$

- $P := \nu a \nu b \nu c \nu d (P_1 \mid P_2 \mid P_3)$
  - $P_1 := \frac{1}{2} \tau.\text{out}(a,c).\text{in}(c,v).0 + \frac{1}{2} \tau.\text{out}(a,d).\text{in}(d,w).0$
  - $P_2 := \text{in}(a,x).\text{out}(b,x).0$
  - $P_3 := \text{in}(b,y).\text{out}(y,e).0$
Example – PRISM model structure

\[ P := va \, vb \, vc \, vd \ (P_1 \mid P_2 \mid P_3) \]
\[ P_1 := \frac{1}{2} \tau.\text{out}(a,c).\text{in}(c,v).0 \]
\[ + \frac{1}{2} \tau.\text{out}(a,d).\text{in}(d,w).0 \]
\[ P_2 := \text{in}(a,x).\text{out}(b,x).0 \]
\[ P_3 := \text{in}(b,y).\text{out}(y,e).0 \]

Free names in \( P_1, P_2, P_3 \):
\ a, b, c, d, e \n
Input-bound names:
\ v, w \ (P_1), \ x \ (P_2), \ y \ (P_3) \n
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;
...
endmodule
module P2
  s2 : [1..3] init 1
x : [0..5] init 0;
...
endmodule
module P3
  s3 : [1..2] init 1
y : [0..5] init 0;
...
endmodule

QEST'07
Example – A PRISM module

\[ P_1 := \frac{1}{2} \tau.\text{out}(a,c).\text{in}(c,v).0 + \frac{1}{2} \tau.\text{out}(a,d).\text{in}(d,w).0 \]

Each PSTG transition is mapped to one or more PRISM commands

```prism
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```

QEST'07
Example – A PRISM module

Each PSTG transition is mapped to one or more PRISM commands

\[ P_1 := \frac{1}{2} \tau.\text{out}(a,c).\text{in}(c,v).0 + \frac{1}{2} \tau.\text{out}(a,d).\text{in}(d,w).0 \]

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
Example – A PRISM module

\[ P_1 := \begin{align*} 
\frac{1}{2} \tau_{\text{out}(a,c)}.\text{in}(c,v).0 + \\
\frac{1}{2} \tau_{\text{out}(a,d)}.\text{in}(d,w).0 
\end{align*} \]

Each PSTG transition is mapped to one or more PRISM commands

```prism
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
```
Example – Module communication

\[ P_1 := \frac{1}{2} \tau.\text{out}(a,c).\text{in}(c,v).0 + \frac{1}{2} \tau.\text{out}(a,d).\text{in}(d,w).0 \]

\[ P_2 := \text{in}(a,x).\text{out}(b,x).0 \]

module \text{P1}  
\begin{align*} 
s1 : [1..6] & \text{ init } 1; 
v : [0..5] & \text{ init } 0; 
w : [0..5] & \text{ init } 0; 
[] (s1 = 1) & \rightarrow 0.5 : (s1' = 2) + 0.5 : (s1' = 3); 
[a\_P1\_P2\_c] (s1 = 2) & \rightarrow (s1' = 4); 
[a\_P1\_P2\_d] (s1 = 3) & \rightarrow (s1' = 5); 
[c\_P3\_P1\_e] (s1 = 4) & \rightarrow (s1' = 6) \& (v' = e); 
[d\_P3\_P1\_e] (s1 = 5) & \rightarrow (s1' = 6) \& (w' = e); 
\end{align*} 
endmodule

module \text{P2}  
\begin{align*} 
s2 : [1..3] & \text{ init } 1 
x : [0..5] & \text{ init } 0; 
[a\_P1\_P2\_c] (s2 = 1) & \rightarrow (s2' = 2) \& (x' = c); 
[a\_P1\_P2\_d] (s2 = 1) & \rightarrow (s2' = 2) \& (x' = d); 
[b\_P2\_P3\_x] (s2 = 2) & \rightarrow (s2' = 3); 
\end{align*} 
endmodule
Translation optimisation

- Basic form of translation makes no assumption about which processes can send which names to each other
- For example:
  - action `out(x,y)` in process $P_i$ for bound $x$ and free $y$
  - results in $a_{ Pi \_ Pj \_ y}$-labelled command for each $j=1,...,n$ ($j\neq i$) and each free name $a$
- In practice, we optimise our translation
  - by computing (an over-approximation of) which processes can send which names to each other
  - with a (finite) iterative analysis of possible values of each input-bound name (and hence each outgoing channel/name)
Property translation

• **Currently, we restrict analysis of** $\pi_{sp}$ **processes to:**
  – (min/max) probabilistic reachability of availability of actions
  – e.g. “minimum probability of getting to state where one of the $n$ subprocesses has reached an error state”
  – easily identified during construction of PSTGs
  – check reachability using PRISM's $P=? \left[ F \ldots \right]$ operator

• **Possible extensions**
  – add test/watchdog processes to system for checking more complex properties
  – expected cost/reward properties
Results

• Implementation: MMC\textsubscript{sp} + Java translator + PRISM

• 3 case studies from literature:
  − dining cryptographers protocol, partial secrets exchange algorithm, mobile communication network (MCN)

• Largest MDP = 10\textsuperscript{9} states = 40 seconds total construction
  − full results in paper

• Analysis of results
  − translation is fast and scalable
  − MCN case study, although small, provides best test of approach
  − efficiency of symbolic (MTBDD) representation from auto-generated PRISM code needs improvement in some cases
Conclusions

- **First automated verification of probabilistic π-calculus**
  - combination of existing tools: MMC and PRISM
  - encouraging experimental results

- **Future work**
  - MTBDD efficiency improvements
  - polyadic variants of π-calculus, e.g. out(x,(a,b))
  - automatic translation of (PCTL) properties
  - further properties, e.g. spatial logics, watchdog processes
  - more complex (and bigger) case studies
  - stochastic π-calculus, biological case studies
# Full results

<table>
<thead>
<tr>
<th>Model</th>
<th>$N$</th>
<th>States</th>
<th>Transitions</th>
<th>MTBDD nodes</th>
<th>Construction time (sec.)</th>
<th>Model checking time (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>PSTGs</td>
<td>PRISM</td>
</tr>
<tr>
<td>DCP</td>
<td>5</td>
<td>160,543</td>
<td>592,397</td>
<td>58,641</td>
<td>10.9</td>
<td>0.81</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>1,475,401</td>
<td>6,520,558</td>
<td>100,290</td>
<td>13.1</td>
<td>0.91</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>13,221,889</td>
<td>68,121,834</td>
<td>154,500</td>
<td>15.2</td>
<td>1.17</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>116,192,457</td>
<td>683,937,352</td>
<td>221,170</td>
<td>18.1</td>
<td>1.21</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>1,005,495,499</td>
<td>6,657,256,911</td>
<td>463,425</td>
<td>19.1</td>
<td>1.37</td>
</tr>
<tr>
<td>PSE</td>
<td>3</td>
<td>9,321</td>
<td>32,052</td>
<td>37,008</td>
<td>4.86</td>
<td>0.75</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>89,025</td>
<td>419,172</td>
<td>103,779</td>
<td>6.60</td>
<td>0.91</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>837,361</td>
<td>5,028,700</td>
<td>173,644</td>
<td>8.12</td>
<td>1.20</td>
</tr>
<tr>
<td>PSE$_3$</td>
<td>3</td>
<td>9,328</td>
<td>32,059</td>
<td>37,251</td>
<td>5.29</td>
<td>0.75</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>89,040</td>
<td>419,187</td>
<td>104,267</td>
<td>6.69</td>
<td>0.96</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>837,392</td>
<td>5,028,731</td>
<td>175,212</td>
<td>7.82</td>
<td>1.13</td>
</tr>
<tr>
<td>MCN</td>
<td>2</td>
<td>609</td>
<td>950</td>
<td>58,430</td>
<td>4.33</td>
<td>2.49</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>3,611</td>
<td>5,811</td>
<td>216,477</td>
<td>5.89</td>
<td>3.11</td>
</tr>
</tbody>
</table>
Structural congruences

• For example
  – $P_1 | \nu x P_2 \equiv \nu x (P_1 | P_2)$
  – if $x$ does not occur in $P_1$