

Probabilistic Model Checking of Randomised Distributed Protocols using PRISM

Dave Parker



University of Birmingham

VPSM PhD School, Copenhagen, October 2006

Part II

Tool Support: PRISM

Overview

- Tool support for probabilistic model checking
- The PRISM tool
 - functionality, features, resources
 - modelling language
 - property specification
 - tool demo
 - efficient symbolic implementations
- Related work/research topics

Motivation

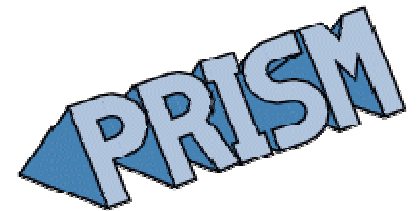
- Complexity of PCTL model checking
 - generally polynomial in model size (number of states)
- State space explosion problem
 - models for realistic case studies are typically huge
- Clearly tool support is required
- Benefits:
 - fully automated process
 - high-level languages/formalisms for building models
 - visualisation of quantitative results

Probabilistic model checkers

- [PRISM](#) (this talk) – DTMCs, MDPs, CTMCs + rewards
- [ETMCC/MRMC](#) – DTMCs, CTMCs + reward extensions
- [LiQuor](#) – LTL verification for MDPs (Probmela language)
- [RAPTURE](#) - prototype for abstraction/refinement of MDPs
- Simulation-based probabilistic model checking:
 - [APMC](#), [Ymer](#) (both based on PRISM language)
- CSL model checking for CTMCs: [APNN-Toolbox](#), [SMART](#)
- Multiple formalism/tool solutions: [CADP](#), [Möbius](#)

The PRISM tool

- **PRISM: Probabilistic symbolic model checker**
 - developed at the University of Birmingham, since approx. 1999
 - free, open source
 - versions for Linux, Unix, Mac OS X, Windows
- **Construction of models:**
 - DTMCs, MDPs , CTMCs + costs/rewards
- **Verification of:**
 - PCTL, CSL + extensions + costs/rewards
- www.cs.bham.ac.uk/~dxp/prism



PRISM - Functionality

- **Constructs three types of probabilistic models:**
 - DTMCs, MDPs, CTMCs
 - also: PTAs with digital clocks by manual translation
 - augmented with costs/rewards
- **The PRISM language** – high-level model description language
- **PRISM simulator** - generate model traces for debugging, etc.
- **Variety of import/export functionality:**
 - model output: text files, Dot graphs, Matlab, ETMCC/MRMC
 - model import: text files
 - other input formalisms via language translation: PEPA, CSP
 - direct connections to other tools: APMC, ProVer/Ymer

PRISM - Functionality

- Supports verification of:
 - PCTL (for DTMCs, MDPs), CSL (for CTMCs)
 - plus “quantitative” extensions
 - cost/reward-based properties
- Powerful, flexible implementation
 - efficient symbolic (BDD-based) implementations
 - multiple computation engines
 - wide range of model analysis methods
 - sampling-based computation (discrete-event simulation)

PRISM - Functionality

- Graphical user interface
 - model/property editor
 - easy automation of verification experiments
 - graphical visualisation of results
 - debugging tool: simulation engine
- Command-line version
 - same underlying verification engines
 - useful for scripting, batch jobs

Getting PRISM + Other Resources

- PRISM website: www.cs.bham.ac.uk/~dxp/prism
 - [tool download](#): binaries, source code
 - [online example repository](#) (40+ case studies)
 - [online documentation](#)
 - [support: help forum, bug tracking, feature requests](#)
 - hosted on Sourceforge
 - [related publications, links](#)

PRISM modelling language

- **Simple, state-based** language for DTMCs/MDPs/CTMCs
 - based on Reactive Modules [Alur/Henzinger]
- **Modules** (system components, composed in parallel)
- **Variables** (finite-valued, local or global)
- **Guarded commands** (labelled with probabilities/rates)
- **Synchronisation** (CSP-style) + **process-algebraic operators** (parallel composition, action hiding/renaming)

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



PRISM language example

```
// hermans self-stabilisation algorithm [Her90]
```

```
dtmc // algorithm is synchronous
```

```
module process1 // first of N=5 symmetric processes
```

```
    x1 : [0..1]; // one bit per process; xi=x(i-1) means process i has a token
```

```
    [step] (x1=x5) -> 0.5 : (x1'=0) + 0.5 : (x1'=1);
```

```
    [step] !x1=x5 -> (x1'=x5);
```

```
endmodule
```

```
// add further processes through renaming
```

```
module process2 = process1 [ x1=x2, x5=x1 ] endmodule
```

```
module process3 = process1 [ x1=x3, x5=x2 ] endmodule
```

```
module process4 = process1 [ x1=x4, x5=x3 ] endmodule
```

```
module process5 = process1 [ x1=x5, x5=x4 ] endmodule
```

```
// can start in any possible configuration
```

```
init true endinit
```

```
// cost - 1 in each state (expected number of steps)
```

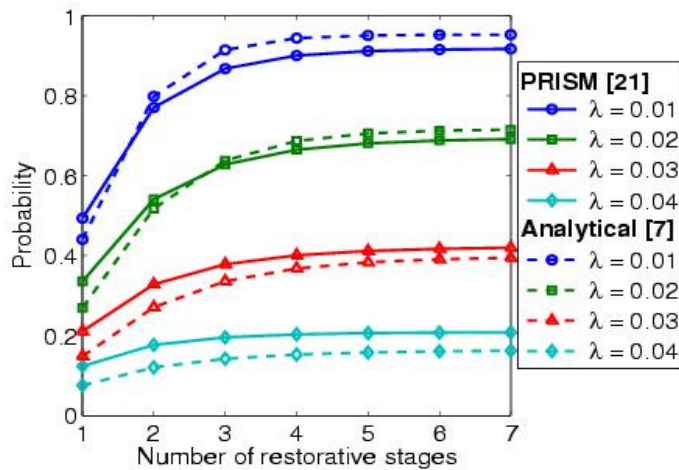
```
rewards true : 1; endrewards
```

PRISM – Property specifications

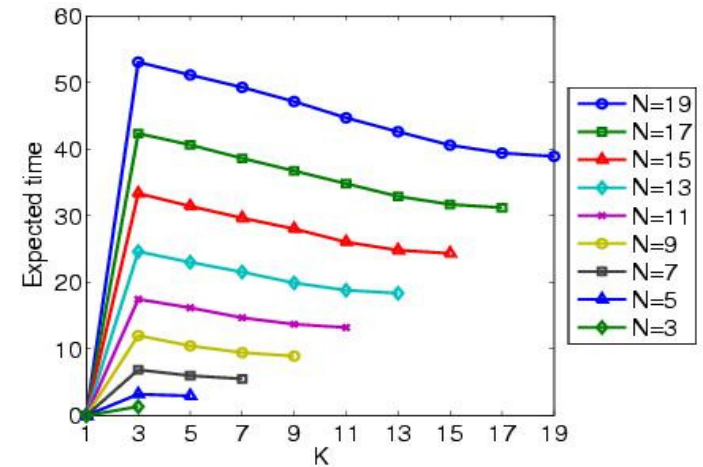
- Based on (probabilistic extensions of) temporal logic
 - incorporates PCTL for DTMCs/MDPs, CSL for CTMCs
- Examples:
 - $P < 0.001 [F \text{ shutdown }]$ - “shutdown eventually occurs with probability at most 0.001”
 - $P < 0.2 [F[t,t] (\text{deliv_rate} < \text{min})]$ “the probability that the current packet delivery rate has dropped below minimum at time t is less than 0.2”
 - $P \geq 0.95 [!\text{repair } U \leq 200 \text{ done }]$ - “with probability 0.95 or greater, the process will successfully complete within 200 hours and without requiring any repairs”
- No counterexamples (error traces) in prob. model checking

PRISM – Property specifications

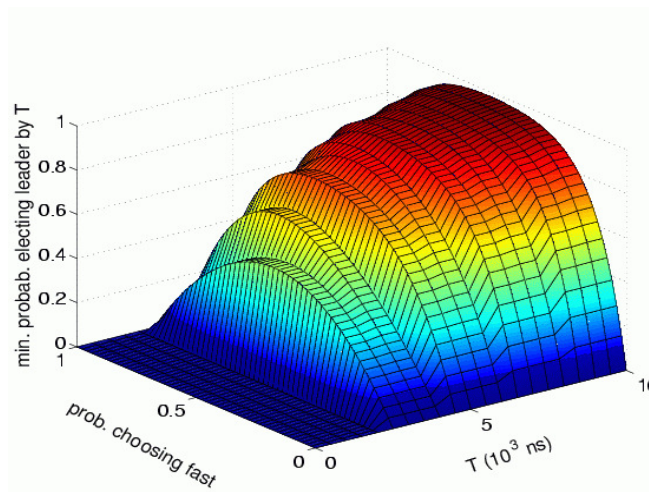
- Focus on quantitative properties, compute actual values
 - $P=? [F \leq T \text{ "shutdown" }]$ - “what is the probability of shutdown occurring within T hours?”
- Best/worst-case scenarios
 - $P=? [F \text{ "error" } \{ \text{"init"} \} \{ \text{max} \}]$ - “what is the worst-case error probability over all possible initial configurations?”
 - $P_{\min}=? [!\text{end2 } U \text{ end1 }]$ - “what is the minimum probability of process 1 finishing before process 2, over all possible schedulings of the processes?”
- Experiments – ranges of model/property parameters
 - $P=? [F \leq T \text{ error }]$ for $N=1..5, T=1..100$
 - identify patterns, trends, anomalies in results



Probability that 10% of gate outputs are erroneous for varying gate failure rates and numbers of stages



Worst-case expected number of steps to stabilise for initial configurations with K tokens amongst N processes



Optimum probability of leader election by time T for various coin biases

Cost- and reward-based properties

- **Costs** and **rewards**
 - real-valued quantities assigned to states/transitions
- **Instantaneous** – state-based measures
 - current queue size, number of operational channels, ...
 - “what is the **expected size** of the message queue at time t ?”
 - “what is the **long-run expected size** of the queue?”
- **Cumulative** – state or transition (impulse) costs/rewards
 - time, power consumption, messages lost, ...
 - “what is the **expected power consumption** during the first 2 hours of operation?”
 - “what is the **worst-case expected time** taken for the protocol to terminate?”

PRISM Demo

PRISM Screenshots

The screenshot displays the PRISM 3.0.beta1 software interface. The main window shows the source code for a model file named 'cluster.sm'. The code is written in a high-level modeling language and includes comments in green and code in black. The code defines a stochastic model with constants for the number of workstations (N, left_mx, right_mx), a formula for a minimum condition, and rates for transitions. It also defines two modules, 'Left' and 'Right', which are interconnected.

```
// workstation cluster [HHK00]
// dxp/gxn 11/01/00

stochastic

const int N; // number of workstations in each cluster
const int left_mx = N; // number of work stations in left cluster
const int right_mx = N; // number of work stations in right cluster

// minimum QoS requires 3/4*N connected workstations operational
const int k=floor(0.75*N);
formula minimum = (left_n>=k & Toleft_n) |
                  (right_n>=k & Toright_n) |
                  ((left_n+right_n)>=k & Toleft_n & Toright_n);

// rates
const double line_rate = 0.0002;
const double Toleft_rate = 0.00025;
const double Toright_rate = 0.00025;

// left cluster
module Left

    left_n : [0..left_mx] init left_mx; // number of workstations operational
    left : bool; // being repaired?

    [startLeft] !left & (left_n<left_mx) -> 1 : (left'=true);
    [repairLeft] left & (left_n<left_mx) -> 1 : (left'=false) & (left_n'=left_n+1);
    [] (left_n>0) -> 0.002*left_n : (left_n'=left_n-1);

endmodule

// right cluster
module Right = Left[left_n=right_n,
                  left=right,
                  left_mx=right_mx,
                  startLeft=startRight,
                  repairLeft=repairRight ]
```

The left sidebar shows the model's structure, including modules (Left, Right, Repairman, Line, ToLeft, ToRight) and constants (N, left_mx, right_mx, k, line_rate, Toleft_rate, Toright_rate, OPERATIONAL, MINIMUM, REPAIR). The bottom status bar indicates the model has been built successfully, with 4180 states and 19552 transitions.

PRISM Screenshots

PRISM 3.0.beta1

File Edit Model Properties Options

Properties list: /data/private/luser/prism-examples/cluster/cluster.csl

Properties

```

S=? [ "premium" ]
S=? [ !"minimum" ]
P>=1 [ true U "premium" ]
P=? [ true U<=T !"minimum" ]
P=? [ true U[T,T] !"minimum" {"!minimum"}{max} ]
P=? [ true U<=T "premium" {"!minimum"}{min} ]
P=? [ "minimum" U<=T "premium" {"!minimum"}{min} ]
P=? [ !"minimum" U>=T "minimum" {"!minimum"}{max} ]
R=? [ I=T {"!minimum"}{min} ]
R=? [ C<=T ]
R=? [ C<=T ]

```

e that QOS drops below minimum quality within T time units (from the initial state)

Constants

Name	Type	Value
T	double	

Labels

Name	Definition
minimum	(left_n>=k&Toleft_n)(right_n>=k&Tori...
premium	(left_n>=left_mx&Toleft_n)(right_n>=r...

Experiments

Property	Defined Const...	Progress	Status	Method
P=? [true U[T...	T=0.0:1.0E-...	660/660 (100%)	Done	Verification
P=? [true U[T...	N=3,T=0.0:1...	101/101 (100%)	Done	Simulation
P=? [true U[T...	N=3,T=0.0:1...	44/101 (43%)	Stopped	Verification
P=? [true U<...	N=3,T=0.0:1...	21/21 (100%)	Done	Verification
P=? [true U<...	N=3:1.5,T=0...	63/63 (100%)	Done	Verification

Graph1 Graph2 Graph3 Graph4 Graph5

New Graph

Probability

T

Legend: N=3, N=4, N=5

Model Properties Simulator Log

Running experiment... done.

PRISM Screenshots

PRISM 3.0.beta1

File Edit Model Properties Options

✂️ 📄 🗑️

Exploration

Auto Update

No. Steps:

Do Update

State time: Auto

Action	Rate	Update
Left	0.016	left_n'=7
Right	0.02	right_n'=9
Line	2.0E-4	line_n'=false
ToRight	2.5E-4	Toright_n'=fal
[startLeft]	10.0	left'=true, r' =
[startToLeft]	10.0	r'=true, Toleft

Path Modification

Backtrack Remove

To Step: Before:

Formulae

Path formulae:

- ? true U<=T !"minimum"
- ? true U[T,T] !"minimum"
- ✓ true U<=T "premium"
- ✓ true U<=T "premium"

State labels:

- ✗ init
- ✗ deadlock
- ✓ minimum
- ✓ premium

Simulation Path

New Path Reset Path Export Path

Model Type: Stochastic (CTMC)
State Rewards: 4373.594264201196
Defined Constants: N=10,T=100.0

Path Length: 18
Transition Rewards: 0.0

Total Time: 44.030215385327985
Total Reward: 4373.594264201196

Step	left_n	left	right_n	right	r	line	line_n	Toleft
0	10	false	10	false	false	false	true	false
1			9					
2				true	true			
3			10	false	false			
4	9							
5		true			true			
6	10	false			false			
7			9					
8				true	true			
9			10	false	false			
10	9							
11								
12			9					
13	8							
14	7							
15		true			true			
16	8	false			false			
17				true	true			
18	9	false	10	false	false	false	true	false

Model Properties Simulator Log

Loading properties... done.

Efficiency - Symbolic techniques

- State space explosion
 - models of real-life systems typically huge
- Symbolic probabilistic model checking
 - data structures based on binary decision diagrams (BDDs)
 - compact storage: exploit model structure and regularity
 - efficient implementation of graph traversal fixed point algorithms
- PRISM: multiple computation engines
 - MTBDDs (BDD extension): storage/analysis of very large models (given structure/regularity), numerical computation can blow up
 - sparse matrices: fastest solution for smaller models ($<10^6$ states), prohibitive memory consumption for larger models
 - hybrid: combine MTBDD storage with explicit storage, ten-fold increase in analysable model size ($\sim 10^7$ states)

Efficiency – Other strategies

- **Approximate model checking** (see also APMC [LHP06])
 - sampling using Monte Carlo discrete-event simulation
 - performed at modelling language level – better scalability
 - potentially huge number of samples for accurate answers
 - also: statistical hypothesis testing, see e.g. [YS02]
- **Parallelisation of model checking**
 - distribution of storage/computation across multi-processor machines [KPZM04], networked clusters [ZPK05], grids
 - potentially promising for symbolic approaches – reduced I/O
 - simulation-based computations much easier to distribute

Ongoing research areas

- **Abstraction and refinement**, see e.g. [DJJL01,KNP06a]
 - construct smaller, abstract model by removing information/variables not relevant to property being checked, iteratively refine abstraction if analysis fails
- **Symmetry reduction** [DM06, KNP06b]
 - exploit replication of identical components
- **Partial order reduction**, see e.g. [BGC04], [DN04]
 - exploit commutativity of concurrently executed transitions
- **Compositionality**, see e.g. [dAHJ01,Che06]
 - analyse full model based on analysis of sub-components

References

- **[BGC04]** C. Baier, M. Grosser, and F. Ciesinski. Partial order reduction for probabilistic systems. In *Proc. QEST'04*, pages 230–239. IEEE Computer Society Press, 2004.
- **[Che06]** L. Cheung. Reconciling Nondeterministic and Probabilistic Choices. Ph.D. thesis, Radboud University of Nijmegen. 2006.
- **[DJJL01]** P. D'Argenio, B. Jeannet, H. Jensen, and K. Larsen. Reachability analysis of probabilistic systems by successive refinements. In *Proc. PAPM/PROBMIV'01*, volume 2165 of *LNCS*, pages 39–56, Springer, 2001.
- **[DN04]** P. D'Argenio and P. Niebert. Partial order reduction on concurrent probabilistic programs. In *Proc. QEST'04*, pages 240–249. IEEE Computer Society Press, 2004.

References

- **[dAHJ01]** L. de Alfaro, T. Henzinger and R. Jhala. Compositional Methods for Probabilistic Systems. In *CONCUR 01: Concurrency Theory, 12th International Conference*, LNCS, Springer-Verlag, 2001.
- **[DM06]** A. Donaldson and A. Miller. Symmetry Reduction for Probabilistic Model Checking using Generic Representatives. In *Proc. 4th International Symposium on Automated Technology for Verification and Analysis (ATVA'06)*, Springer. October 2006.
- **[KNP06a]** M. Kwiatkowska, G. Norman and D. Parker. Game-based Abstraction for Markov Decision Processes. In *Proc. 3rd International Conference on Quantitative Evaluation of Systems (QEST'06)*, pages 157-166, IEEE CS Press. 2006.
- **[KNP06b]** M. Kwiatkowska, G. Norman and D. Parker. Symmetry Reduction for Probabilistic Model Checking. In *Proc. 18th International Conference on Computer Aided Verification (CAV'06)*, volume 4144 of LNCS, pages 234-248, Springer, 2006.

References

- **[KPZM04]** M. Kwiatkowska, D. Parker, Y. Zhang and R. Mehmood. Dual-Processor Parallelisation of Symbolic Probabilistic Model Checking. In *Proc. MASCOTS'04*, pages 123-130, IEEE CS Press. 2004.
- **[LHP06]** R. Lassaigne, T. Héroult and S. Peyronnet. APMC 3.0: Approximate verification of Discrete and Continuous Time Markov Chains. In *Proc. 3rd International Conference on Quantitative Evaluation of Systems (QEST'06)*, 2006.
- **[YS02]** H. Younes and R. Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In *Proceedings of the 14th International Conference on Computer Aided Verification*, volume 2404 of *LNCS*, pages 223-235, Copenhagen, Denmark, July 2002.
- **[ZPK05]** Y. Zhang, D. Parker and M. Kwiatkowska. A Wavefront Parallelisation of CTMC Solution using MTBDDs. In *Proc. International Conference on Dependable Systems and Networks (DSN'05)*, pages 732-742, IEEE Computer Society Press. 2005.