Probabilistic Model Checking

Marta Kwiatkowska
Gethin Norman
Dave Parker

University of Oxford

Part 9 – PRISM

Overview

- Tool support for probabilistic model checking
  - motivation, existing tools
- The PRISM model checker
  - functionality, features
  - resources
  - modelling language
  - property specification
- PRISM tool demo

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
- MDP tools
  - 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), VESTA
- CSL model checking for CTMCs:
  - APNN-Toolbox, SMART
- Multiple formalism/tool solutions:
  - CADP, Möbius

Overview

- Tool support for probabilistic model checking
  - motivation, existing tools
- The PRISM model checker
  - functionality, features
  - resources
  - modelling language
  - property specification
- PRISM tool demo

The PRISM tool

- PRISM: Probabilistic symbolic model checker
  - developed at the Birmingham/Oxford University, since 1999
  - free, open source (GPL)
  - versions for Linux, Unix, Mac OS X, Windows, 64-bit OSs
- Modelling of:
  - DTMCs, MDPs, CTMCs + costs/rewards
- Verification of:
  - PCTL, CSL + extensions + costs/rewards
- Features:
  - high-level modelling language, wide range of model analysis methods, graphical user interface, efficient implementation

Getting PRISM + Other Resources

- PRISM website: www.prismmodelchecker.org
  - tool download: binaries, source code (GPL)
  - on-line example repository (40+ case studies)
  - on-line documentation:
    - PRISM manual
    - PRISM tutorial
  - support: help forum, bug tracking, feature requests
    - hosted on Sourceforge
  - related publications, talks, tutorials, links

PRISM – Model building

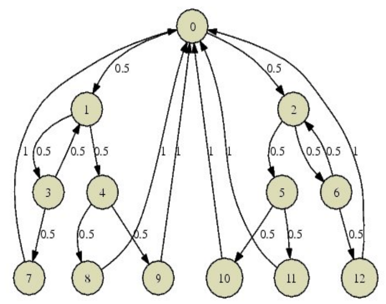
- First step of verification = construct full probabilistic model (not always necessary in non-probabilistic model checking)

High-level model

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

dtmc // Algorithm is fully synchronous

module process1 // first of N=5 symmetric processes

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

(PRISM language)

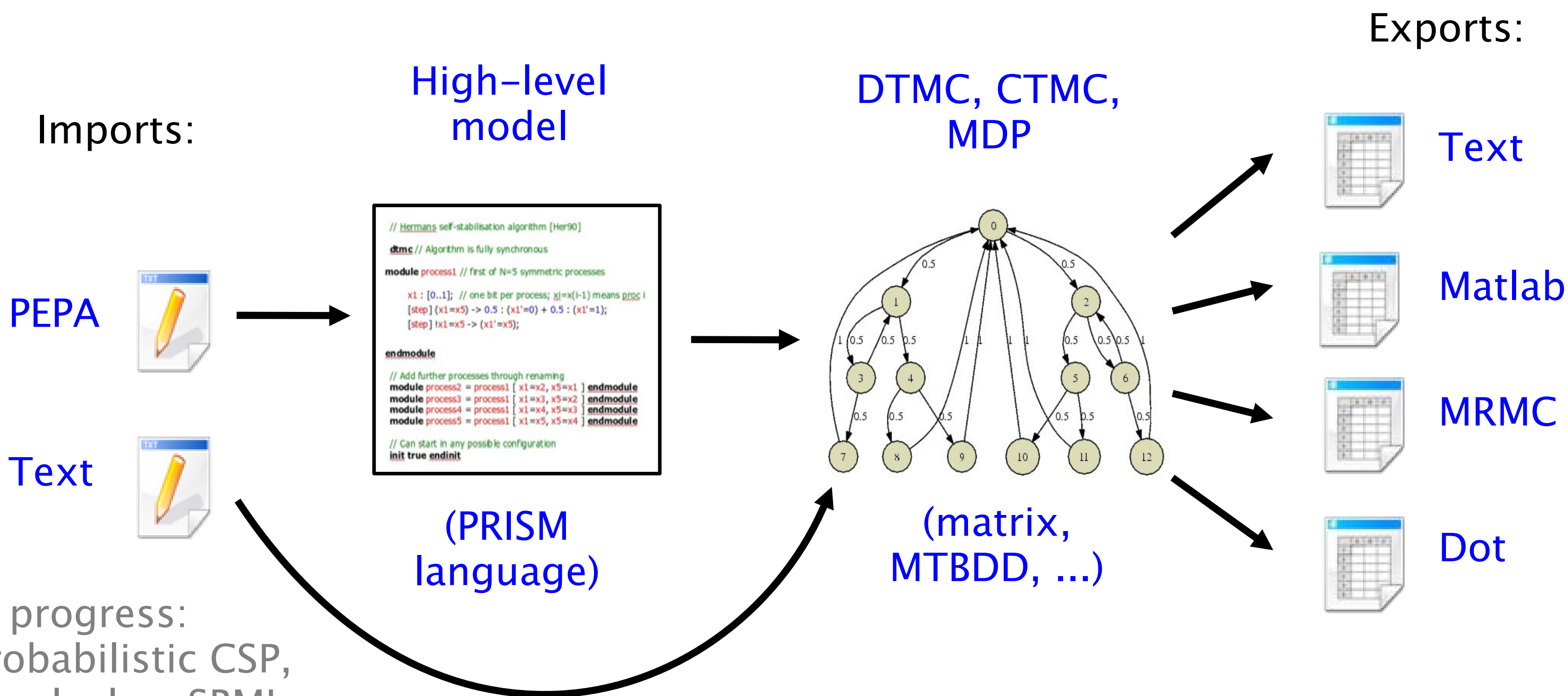
DTMC, CTMC, MDP



(matrix, MTBDD, ...)

PRISM – Imports and exports

- Support for connections to other formats/tools:



In progress: probabilistic CSP, pi calculus, SBML, Probmela, ...

PRISM modelling language

- Simple, state-based language for DTMCs/MDPs/CTMCs
  - based on Reactive Modules [AH99]
- 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)

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

action	guard	probability	update	probability	update

PRISM language example

```
// Herman's self-stabilisation algorithm [Her90]

dtmc // Algorithm is fully synchronous

module process1  // First of N=5 symmetric processes

        x1 : [0..1];  // One bit per process; xi=x(i-1) means proc 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
```

PRISM language example 2 (fragment)

```
// Embedded control system
ctmc

const int MIN_SENSORS = 2;
const double lambda_p = 1/(365*24*60*60); // MTTF = 1 year
...

module sensors
      s : [0..3] init 3; // Number of sensors working
      [] s>1 -> s*lambda_s : (s'=s-1); // Failure of a single sensor
endmodule

module proci  // (takes data from sensors and passes onto main processor)
      i : [0..2] init 2; // 2=ok, 1=transient fault, 0=failed
      [] i>0 & s>=MIN_SENSORS -> lambda_p : (i'=0); // Failure of processor
      [] i=2 & s>=MIN_SENSORS -> delta_f : (i'=1); // Transient fault
      [reboot] i=1 & s>=MIN_SENSORS -> delta_r : (i'=2); // Transient reboot
endmodule
```

Costs and rewards

- Real-valued quantities assigned to model states/transitions
  - many possible uses, e.g. time, power consumption, current queue size, number of messages lost, ...
- No distinction between costs (“bad”) and rewards (“good”)
  - PRISM terminology is rewards
- The meaning of these rewards varies depending on:
  - the type of property used to analyse the model: instantaneous or cumulative

Rewards in the PRISM language

```
rewards "total_queue_size"
  true : queue1+queue2;
endrewards
```

(instantaneous, state rewards)

```
rewards "time"
  true : 1;
endrewards
```

(cumulative, state rewards)

```
rewards "power"
  sleep=true : 0.25;
  sleep=false : 1.2 * up;
endrewards
```

(cumulative, state rewards)
(up = number of operational components)

```
rewards "dropped"
  [receive] q=q_max : 1;
endrewards
```

(cumulative, transition rewards)
(q = queue size, q_max = max queue size)

PRISM property specifications

- Based on (probabilistic extensions of) temporal logic
  - incorporates PCTL for DTMCs/MDPs, CSL for CTMCs
  - also includes: quantitative extensions, costs/rewards
- Simple PCTL/CSL example:
  - P<0.001 [true U shutdown] – "the system eventually shuts down with probability at most 0.001"
- Usually focus on quantitative properties:
  - P=? [true U shutdown] – "what is the probability that the system eventually shuts down?"
  - nested probabilistic operators must be probability-bounded

Basic types of property specifications

- (Unbounded) reachability:
  - P=? [true U shutdown] – “probability of eventual shutdown”
- Transient/time-bounded properties:
  - P=? [true U[t,t] (deliv_rate < min)] – “probability that the packet delivery rate has dropped below minimum at time t”
  - P=? [!repair U≤200 done] – “probability of the process completing within 200 hours and without requiring repairs”
- Steady-state properties:
  - S=? [num_sensors ≥ min] – “long-run probability that an adequate number of sensors are operational”

Cost- and reward-based properties

- Two different interpretations of model rewards
  - instantaneous and cumulative properties
  - reason about expected values of rewards

- Instantaneous reward properties
  - state rewards only
  - state-based measures: “queue size”, “number of operational channels”, “concentration of reactant X”, ...

- R=? [I=t]
  - e.g. “expected size of the message queue at time t?”
- R=? [S]
  - e.g. “long-run expected size of the queue?”

Cost- and reward-based properties

- Cumulative reward properties
  - both state and transition rewards
  - CTMC state rewards interpreted as reward rates
  - e.g. “time”, “power consumption”, “number of messages lost”

- R=? [F end]
  - e.g. “expected time taken for the protocol to terminate?”
- R=? [C≤2]
  - e.g. “expected power consumption during the first 2 hours that the system is in operation?”
  - e.g. “expected number of messages lost during...”

Best/worst-case scenarios

- Combining "quantitative" and "exhaustive" aspects

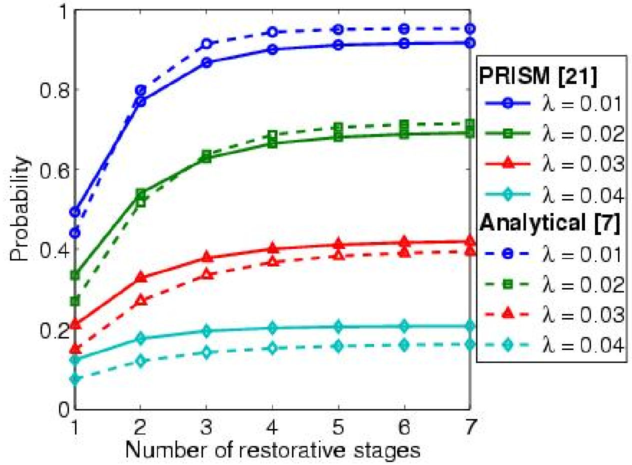
- Computing values for a range of states
  - R=? [F end {"init"}{max}] – "maximum expected run-time over all possible initial configurations"
  - P=? [true U$\leq$t elected {tokens$\leq$k}{min}] – "minimum probability of the leader election algorithm completing within t steps from any state where there are at most k tokens"

- All possible resolutions of nondeterminism (MDPs)
  - Pmin=? [!end2 U end1] – "minimum probability of process 1 finishing before process 2, for any scheduling of processes?"
  - Rmax=? [F message_delivered] – "maximum expected number of bits revealed under any eavesdropping strategy?"

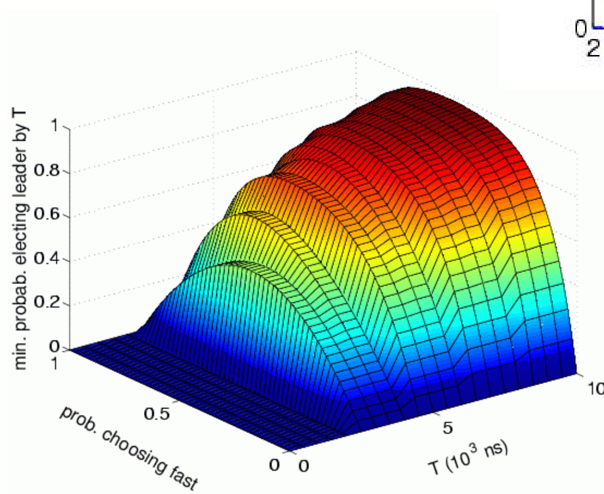
Identifying trends and anomalies

- Counterexamples (error traces)
  - widely used in non-probabilistic model checking
  - situation much less clear in probabilistic model checking
  - counterexample for P<p [true U error] ? and for P=? [...] ?
  - work in progress...

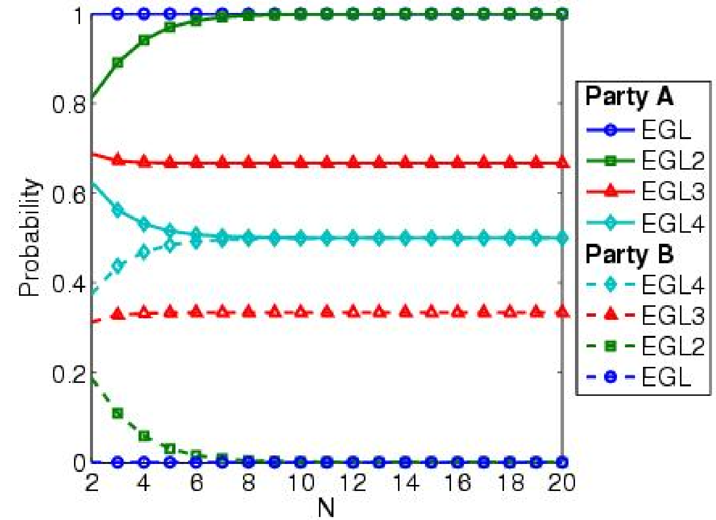
- Experiments: ranges of model/property parameters
  - e.g. P=? [true U≤T error] for N=1..5, T=1..100
    where N is some model parameter and T a time bound
  - identify patterns, trends, anomalies in quantitative results



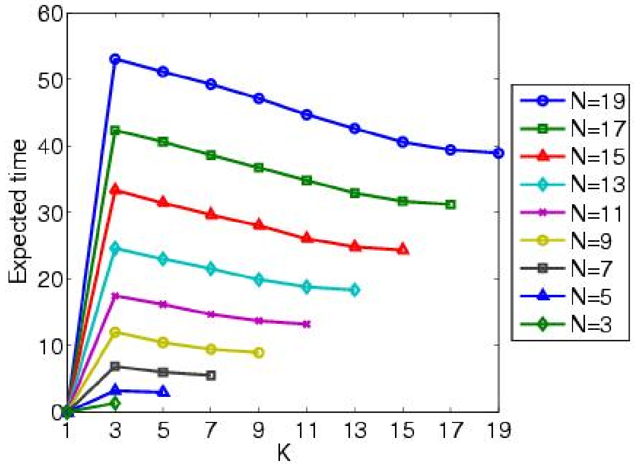
Probability that 10% of gate outputs are erroneous for varying gate failure rates and numbers of stages



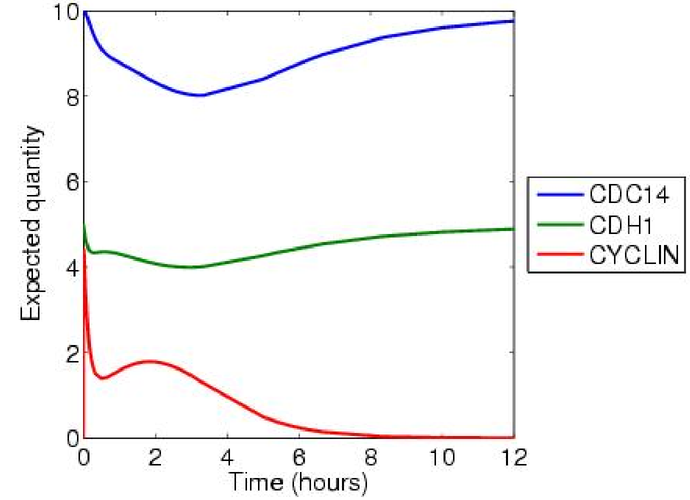
Optimum probability of leader election by time T for various coin biases



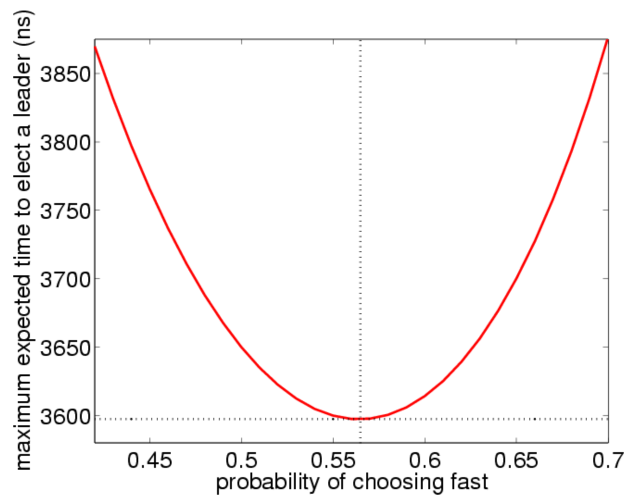
Probability that parties gain unfair advantage for varying numbers of secret packets sent



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



Expected reactant concentrations over the first 12 hours



Maximum expected time for leader election for various coin biases

PRISM functionality

- Graphical user interface
  - model/property editor
  - discrete-event simulator - model traces for debugging, etc.
  - verification of PCTL, CSL + costs/rewards, etc.
  - approximate verification using simulation + sampling
  - easy automation of verification experiments
  - graphical visualisation of results

- Command-line version
  - same underlying verification engines
  - useful for scripting, batch jobs

Overview

- Tool support for probabilistic model checking
  - motivation, existing tools
- The PRISM model checker
  - functionality, features
  - resources
  - modelling language
  - property specification
- PRISM tool demo