PRISM

Overview, Recent Updates and Future Directions

Dave Parker

University of Oxford

ERC Workshop on Software Quality, Venice, Sep 2011

UNIVERSITY OF OXFORD

PRISM – An overview

- PRISM is a probabilistic model checker
  - automatic verification of systems with stochastic behaviour
  - e.g. due to unreliability, uncertainty, randomisation, …
- Construction/analysis of probabilistic models…
  - discrete- and continuous-time Markov chains, Markov decision processes, probabilistic timed automata
- Verification of properties in probabilistic temporal logics…
  - PCTL, CSL, LTL, PCTL*, quantitative extensions, costs/rewards
- Various model checking engines and techniques
  - symbolic, explicit-state, simulation-based data structures, symmetry reduction, quantitative abstraction refinement, …
- PRISM is free and open source
  - www.prismmodelchecker.org

Overview

- Probabilistic models
  - model types, modelling language, case studies/benchmarks
- Property specification
  - temporal logics + extensions
- Underlying techniques and implementation
  - symbolic/explicit-state, PTA model checking, statistical m/c
- Future additions
  - probabilistic counterexamples, multi-objective model checking, compositional model checking, stochastic games

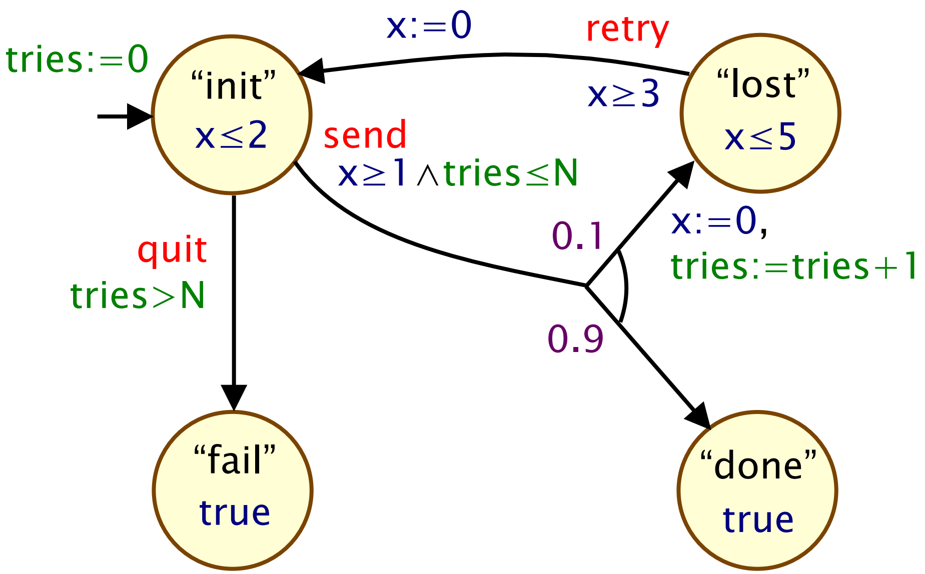
PRISM – Probabilistic models

- Discrete-time Markov chains (DTMCs)
  - discrete states + probability
  - for: randomisation, unreliable communication media, …
- Continuous-time Markov chains (CTMCs)
  - discrete states + exponentially distributed delays
  - for: component failures, job arrivals, molecular reactions, …
- Markov decision processes (MDPs)
  - in fact: probabilistic automata [Segala]
  - probability + nondeterminism (e.g. for concurrency, control)
  - for: randomised distributed algorithms, security protocols, …
- Probabilistic timed automata (PTAs) [new in PRISM 4.0]
  - probability, nondeterminism + real-time
  - for wireless comm. protocols, embedded control systems, …

Probabilistic timed automata (PTAs)

- Probability + nondeterminism + real-time
  - timed automata + discrete probabilistic choice, or…
  - probabilistic automata + real-valued clocks

- PTA example: message transmission over faulty channel



States
- locations + data variables

Transitions
- guards and action labels

Real-valued clocks
- state invariants, guards, resets

Probability
- discrete probabilistic choice

The PRISM modelling language

- Simple textual modelling language for probabilistic systems
  - inspired by “Reactive Modules” formalism [Alur/Henzinger]

```
pta

const int N;

module transmitter

    s : [0..3] init 0;
    tries : [0..N+1] init 0;

    x : clock;

    invariant (s=0 ⇒ x≤2) & (s=1 ⇒ x≤5) endinvariant

    [send] s=0 & tries≤N & x≥1
           → 0.9 : (s’=3)
           + 0.1 : (s’=1) & (tries’=tries+1) & (x’=0);
    [retry] s=1 & x≥3 → (s’ =0) & (x’ =0);
    [quit]  s=0 & tries>N → (s’ =2);

endmodule

rewards “energy” (s=0) : 2.5; endrewards
```

The PRISM modelling language

- Simple textual modelling language for probabilistic systems
  - inspired by “Reactive Modules” formalism [Alur/Henzinger]

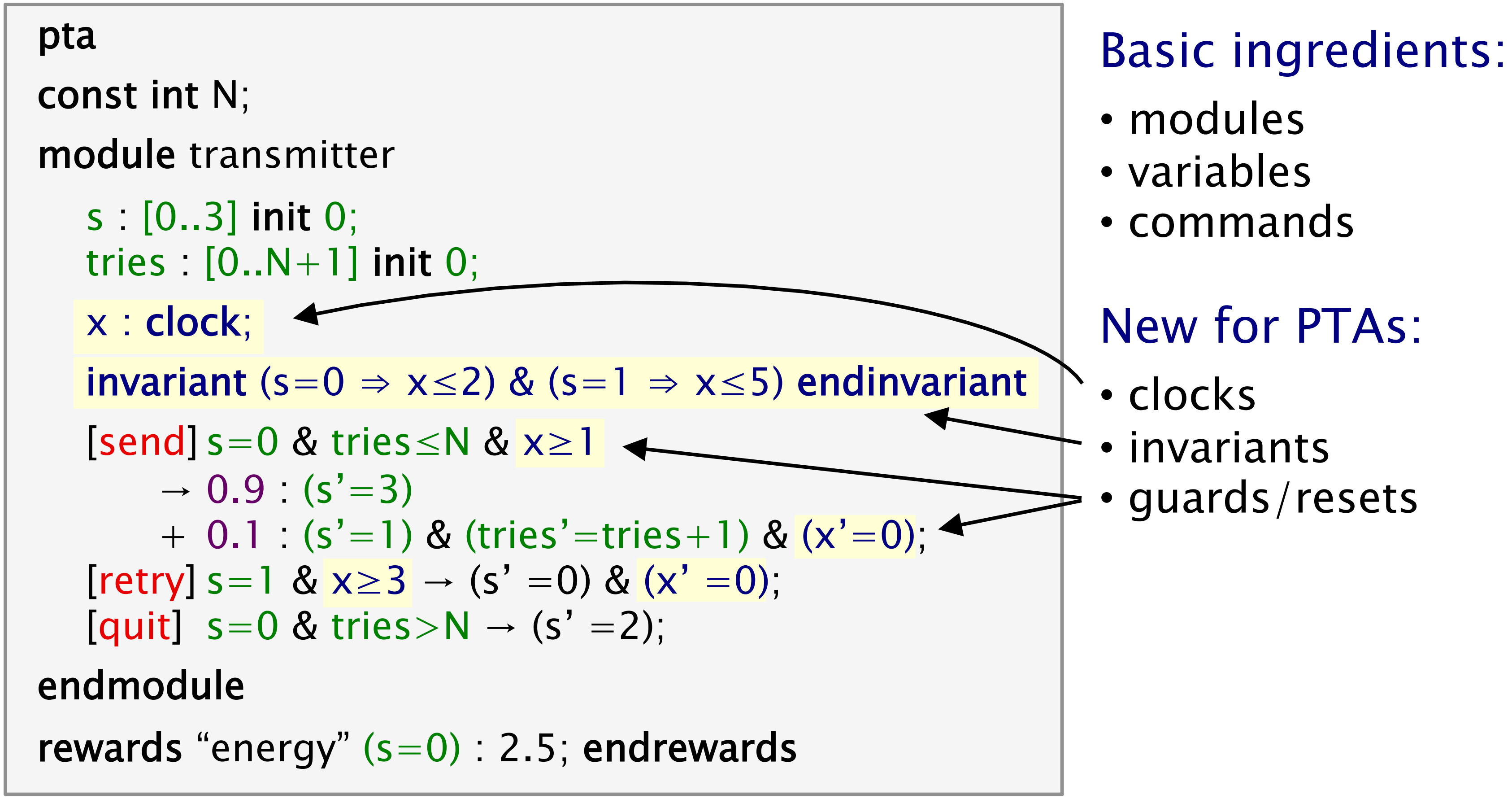
```
pta
const int N;
module transmitter
    s : [0..3] init 0;
    tries : [0..N+1] init 0;
    x : clock;
    invariant (s=0 ⇒ x≤2) & (s=1 ⇒ x≤5) endinvariant
    [send] s=0 & tries≤N & x≥1
          → 0.9 : (s’=3)
          + 0.1 : (s’=1) & (tries’=tries+1) & (x’=0);
    [retry] s=1 & x≥3 → (s’ =0) & (x’ =0);
    [quit]  s=0 & tries>N → (s’ =2);
endmodule
rewards “energy” (s=0) : 2.5; endrewards
```

Basic ingredients:

- modules
- variables
- commands

The PRISM modelling language

- Simple textual modelling language for probabilistic systems
  - inspired by “Reactive Modules” formalism [Alur/Henzinger]



```
pta
const int N;
module transmitter
    s : [0..3] init 0;
    tries : [0..N+1] init 0;
    x : clock;
    invariant (s=0 ⇒ x≤2) & (s=1 ⇒ x≤5) endinvariant
    [send] s=0 & tries≤N & x≥1
          → 0.9 : (s’=3)
          + 0.1  : (s’=1) & (tries’=tries+1) & (x’=0);
    [retry] s=1 & x≥3 → (s’ =0) & (x’ =0);
    [quit]  s=0 & tries>N → (s’ =2);
endmodule
rewards “energy” (s=0) : 2.5; endrewards
```

Basic ingredients:

- modules
- variables
- commands

New for PTAs:

- clocks
- invariants
- guards/resets

The PRISM modelling language

- Simple textual modelling language for probabilistic systems
  - inspired by “Reactive Modules” formalism [Alur/Henzinger]

```
pta
const int N;
module transmitter
    s : [0..3] init 0;
    tries : [0..N+1] init 0;
    x : clock;
    invariant (s=0 ⇒ x≤2) & (s=1 ⇒ x≤5) endinvariant
    [send] s=0 & tries≤N & x≥1
           → 0.9 : (s’=3)
           + 0.1 : (s’=1) & (tries’=tries+1) & (x’=0);
    [retry] s=1 & x≥3 → (s’ =0) & (x’ =0);
    [quit]  s=0 & tries>N → (s’ =2);
endmodule
rewards “energy” (s=0) : 2.5; endrewards
```

Basic ingredients:

- modules
- variables
- commands

New for PTAs:

- clocks
- invariants
- guards/resets

Also:

- rewards
  (i.e. costs, prices)
- parallel composition

PRISM – Case studies

- Randomised distributed algorithms
  - consensus, leader election, self-stabilisation, …
- Randomised communication protocols
  - Bluetooth, FireWire, Zeroconf, 802.11, Zigbee, gossiping, …
- Security protocols/systems
  - contract signing, anonymity, pin cracking, quantum crypto, …
- Biological systems
  - cell signalling pathways, DNA computation, …
- Planning & controller synthesis
  - robotics, dynamic power management, …
- Performance & reliability
  - nanotechnology, cloud computing, manufacturing systems, …

- See: www.prismmodelchecker.org/casestudies

The PRISM benchmark suite

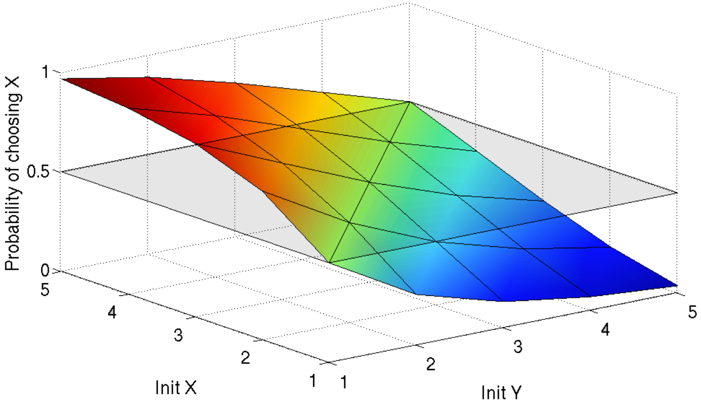
- PRISM models are widely used for testing/benchmarking
  - but there are many case studies in several locations
  - can be hard to find the right type of examples for testing
- The PRISM benchmark suite
  - collection of probabilistic model checking benchmarks
  - designed to make it easy to test/evaluate/compare tools
  - currently, approx. 20 models, of various types and sizes
  - wide range of model checking properties, grouped by type
  - PRISM can also export built models in various formats
- See: www.prismmodelchecker.org/benchmarks

PRISM – Property specification

- Temporal logic-based property specification language
  - subsumes PCTL, CSL, probabilistic LTL, PCTL*, (CTL), …

- Simple examples:
  - $P_{\leq 0.01}$ [F “crash”] – “the probability of a crash is at most 0.01”
  - $S_{>0.999}$ [“up”] – “long-run probability of availability is >0.999”

- Usually focus on quantitative (numerical) properties:
  - $P_{=?}$ [F “crash”]
    “what is the probability of a crash occurring?”
  - typically, use “experiments”, i.e. analyse plots/trends in quantitative properties as system parameters vary



PRISM – Property specification

- Properties can combine numerical + exhaustive aspects
  - $P_{max=?}$ [$F^{\leq 10}$ "fail"] – "worst-case probability of a failure occurring within 10 seconds, for any possible scheduling of system components"
  - $P_{=?}$ [$G^{\leq 0.02}$!"deploy" {"crash"}{max}] – "the maximum probability of an airbag failing to deploy within 0.02s, from any possible crash scenario"

- Reward-based properties (rewards = costs = prices)
  - $R_{\{\text{"time"}\}=?}$ [F "end"] – "expected algorithm execution time"
  - $R_{\{\text{"energy"}\}max=?}$ [$C^{\leq 7200}$] – "worst-case expected energy consumption during the first 2 hours"

- Properties can be combined with e.g. arithmetic operators
  - e.g. $P_{=?}$ [F $fail_1$] / $P_{=?}$ [F $fail_{any}$] – "conditional failure prob."

PRISM – Underlying techniques

- Basic ingredients for probabilistic model checking
  - construction of probabilistic model (from high-level descr.)
  - graph-based algorithms (reachability, SCC decomposition, …)
  - iterative numerical computation (lin. equ.s, value iteration, …)

- Recent additions/extensions (in PRISM 4.0):

- 1. Explicit-state probabilistic model checking

- 2. Probabilistic timed automata (PTA) model checking

- 3. Approximate/statistical model checking

Explicit-state (vs. symbolic) techniques

- To date, PRISM's implementation has been mostly symbolic
  - i.e. (multi-terminal) binary decision diagrams – (MT)BDDs
  - can be very compact/efficient for large, structured models
  - 3 model checking engines, but all partially symbolic

- New explicit-state engine in PRISM
  - no BDDs; uses: vectors, bit-sets, sparse matrices
  - more efficient for small, unstructured models
  - more efficient if model needs to manipulated on-the-fly
  - particularly well suited to prototyping new techniques (designed to be used as a standalone library)
  - also being developed into a fully fledged PRISM engine
  - some additional functionality: e.g. extra techniques for MDPs (policy iteration, …), extra models (CTMDPs, stoch. games)

PTA model checking in PRISM

- Properties for PTAs similar to those for other models:
  - min/max probability of reaching X (within time T)
  - min/max expected cost/reward to reach X
- But infinite state space necessitates different techniques
  - PRISM has two different approaches to PTA model checking…
- “Digital clocks” – conversion to finite-state MDP
  - preserves min/max probability + expected cost/reward/price
  - (for PTAs with closed, diagonal-free constraints)
  - efficient, in combination with PRISM’s symbolic engines
- Quantitative abstraction refinement
  - zone-based abstractions of PTAs using stochastic games
  - provide lower/upper bounds on quantitative properties
  - automatic iterative abstraction refinement

Approximate/statistical model checking

- Discrete event (Monte Carlo) simulation + sampling
  - much better scalability/applicability, at expense of precision
  - full probabilistic models only (no nondeterminism)
- PRISM 4.0 has a completely re-written simulator engine
  - two approximate model checking approaches…
- Estimation: approximate result for $P_{=?} [\phi]$, plus a
  - confidence interval (for a given confidence level)
  - probabilistic guarantee for result precision [Hérault et al.]
- Acceptance sampling: yes/no answer for $P_{\sim p} [\phi]$
  - correct with high probability [Younes/Simmons]
  - stop sampling as soon as the result can be given
  - PRISM implements SPRT (sequential probability ratio test)

Future additions to PRISM

- Recent/current work being integrated into PRISM:
- 1. Probabilistic counterexamples
- 2. Multi-objective model checking
- 3. Compositional probabilistic verification
- 4. Game-based probabilistic models
- 5. Incremental probabilistic model checking
  - (see Mateusz's talk)

Probabilistic counterexamples

- In conventional (non-probabilistic) model checking
  - counterexamples are typically single traces to an error
  - and are essential to the usefulness of model checkers

- Probabilistic counterexamples
  - e.g. for property “probability of an error occurring is $\leq$ p”
  - *sets* of error traces with combined probability $>$ p

- PRISM extended to generate probabilistic counterexamples
  - aim to build “small” counterexample (few traces) which includes “most likely” events (largest probabilities)
  - reduces to solving “k-shortest paths” problem [Han/Katoen]
  - currently use REA algorithm [Jiménez/Marzal]
  - various optimisations possible: regexps, subgraphs, SCCs,

Multi-objective model checking

- Model checking for MDPs quantifies over all adversaries
  - adversary = strategy = policy = resolution of nondeterminism
  - verification: “worst case probability of error is always < 0.01”
  - controller synthesis: “how to minimise expected run-time?”
  - PRISM 4.0 generates optimal (best/worst-case) adversaries
- Multi-objective probabilistic model checking
  - investigate trade-offs between conflicting objectives
  - e.g. “maximum probability of message transmission, assuming expected battery life-time is > 10 hrs”
- PRISM extension
  - extension of property specification language [TACAS’11]
  - support for probabilistic omega-regular and reward properties
  - reduces to solution of linear programming problem

Compositional probabilistic verification

- Assume-guarantee (A-G) framework for MDPs [TACAS'10]
  - assumptions/guarantees are probabilistic safety properties
  - e.g. "warn signal sent before shutdown signal with prob. 0.99"
  - can be generalised to more expressive properties [TACAS'11]
- Example A-G proof rule:

$$\frac{\begin{array}{c} M_1 \models \langle A\rangle_{\geq p_A} \\ \langle A\rangle_{\geq p_A} \; M_2 \; \langle G\rangle_{\geq p_G} \end{array}}{M_1 \;||\; M_2 \models \langle G\rangle_{\geq p_G}} \qquad \text{(A}\textsc{sym}\text{)}$$

- A-G model checking reduces to multi-objective queries
  - "every adversary that satisfies A must also satisfy G"
- In progress: integration into PRISM
  - extend input language with automata-based properties
  - allow specification of which proof rule(s) to apply

Game-based probabilistic models

- Game-theoretic approach to model checking
  - models competitive and/or collaborative behaviour
  - e.g. for verification of security protocols, …
- Extending PRISM with stochastic multi-player games
  - native support in PRISM modelling language
  - modules and/or synchronous action labels assigned to players
- Probabilistic model checking for:
  - PATL: probabilistic version of Alternating Time Temporal Logic
  - “can players 1 and 2 collaborate such that the probability of … is at least p, whatever players 3 and 4 do?”
  - also: cost/reward-based properties
  - reduction to analysis of stochastic two-player games

More information…

- More info and resources online
  - www.prismmodelchecker.org
- Documentation + related papers
- Tutorials, teaching material, support
- Case studies repository + benchmark suite
- Questions welcome…