# PRISM – An Overview

Dave Parker
Marta Kwiatkowska
Gethin Norman

THE UNIVERSITY OF BIRMINGHAM

EPSRC QinetiQ

# PRISM – Overview

- Probabilistic Symbolic Model Checker
  - formal analysis of probabilistic systems
- Developed:
  - at the University of Birmingham
  - for the past 4½ years
  - by: M. Kwiatkowska, G. Norman, D. Parker
  - also: J. Meyer-Kayser, S. Gilmore, A. Hinton, R. Downing

# PRISM – Supported Models

- Discrete-time Markov chains (DTMCs)
  - discrete state transition probabilities
- Continuous-time Markov chains (CTMCs)
  - time modelled as exponential distributions
- Markov decision processes (MDPs)
  - nondeterminism + probabilities
  - e.g. distributed probabilistic systems

# PRISM – Specification Formalisms

- Extensions of temporal logic CTL
- PCTL (for properties of DTMCs, MDPs)
  - "Prob. of leader eventually being elected is 1"
  - "Prob. of error occuring within k steps is < 0.01"
- CSL (for properties of CTMCs)
  - "Prob. of queue being full within 1 hour is < 0.2"
  - "Long-run prob. of server being down is < 0.05"

# PRISM – Basic Functionality

- Parse model description (PRISM language)
- Construct probabilistic model
- Compute reachable/deadlock states

- Parse properties (temporal logic)
- Model check each property
  - Return Yes/No + actual probability

# Recent Extensions

- Process algebra operators added
  - More flexible parallel composition
  - Translation from stochastic process algebra PEPA
- Support for rewards/costs added
  - State-based rewards/costs in probabilistic models
  - Verification of properties such as
    - “Expected number of steps before termination is...”
    - “Expected time to elect a leader is...”

# The PRISM Language

- State-based model description language
  - based on Reactive Modules [Alur, Henzinger]
- Basic ingredients:
  - modules with local, integer-valued variables
  - defined by guarded commands
  - combine through parallel composition
- Also:
  - synchronisation over action labels (CSP-style)
  - global variables

# Example

- Randomised self-stabilisation [Israeli-Jalfon]

- Ring of identical processes, each has a token

- Stable state - only one process has a token

- Compute:
  - minimum probability of stabilising
  - maximum expected time to stabilise

```
// Israeli-Jalfon algorithm
// dxp/gxn 10/06/02
//ring of size 3

nondeterministic

global q1 : [0..1] init 1;
global q2 : [0..1] init 1;
global q3 : [0..1] init 1;

module start

	start : [0..1] init 0;

	// end of initilization phase
	[] start=0 -> start'=1;

endmodule

module process1

	// initialization (non-deterministic choice as to value of bits)
	[] start=0 -> q1'=0;
	[] start=0 -> q1'=1;

	// the protocol
	[] start=1 & (q1=1) -> 0.5 : q1'=0 & q3'=1 + 0.5 : q1'=0 & q2'=1;

endmodule

// add further processes through renaming
module process2=process1[q1=q2, q2=q3, q3=q1] endmodule
module process3=process1[q1=q3, q2=q1, q3=q2] endmodule
```

# PEPA Extension

- Module parallel composition alternatives:
  - $P_1 \; || \; P_2$
    - synchronise over alphabets
  - $P_1 \; |[a,b,c]| \; P_2$
    - synchronise over a named set of actions
  - $P_1 \; ||| \; P_2$
    - fully asynchronous
- Action hiding: $P \setminus \{a,b,c\}$

# Case Studies

- Randomised distributed algorithms/protocols
  - Leader election algorithms
  - Consensus protocol
  - Byzantine agreement protocol
  - IEEE 1394 FireWire Root Contention
  - Crowds (anonymity) protocol
  - IEEE 802.11 Wireless LAN
- CTMC models
  - queues, networks, manufacturing systems

# Implementation

- Java – User interfaces, high-level code
- C++ – underlying data structures/engines

- CUDD – BDD/MTBDD library
- JavaCC – language parsers

- Source code – GPL, web site download
- Platforms – Solaris, Linux, ...

# Future Plans

- Simulator
- GUI development
- Parallel/distributed implementations
- Abstraction, compositionality, ...