Overview

- Research activities at Birmingham
- Probabilistic π-calculus model checking
  - (ongoing joint work with Catuscia, Peng)
- Game-based abstraction for MDPs
  - (to be presented at QEST'06)

Research activities at Birmingham

Birmingham – People

- Research focus: probabilistic verification
  - in particular, probabilistic model checking

- Group leader: Marta Kwiatkowska
- Post-docs: Gethin Norman, Dave Parker, Maria Vigliotti
- PhDs: Fuzhi Wang, Oksana Tymchyshyn, Matthias Fruth
- Current visitors: Husain Aljazzar

Some ongoing projects

- Automated Verification of Probabilistic Protocols with PRISM
  - EPSRC, 2003-2006, with: Segala (Verona)
- Probabilistic Model Checking of Mobile Ad-Hoc Network Protocols
  - EPSRC, 2003-2006, with: Marshall (BTexact), UCL
- UbiVal: Fundamental Approaches to Validation of Ubiquitous Computing Applications and Infrastructures
  - EPSRC, 2006-2010, with: UCL, Imperial College
- Predictive modelling of signalling pathways via probabilistic model checking with PRISM
  - MSR Cambridge, 2006-2007, with: Biosciences (Birmingham), Andrew Finney (Physiomics PLC)

The PRISM tool

- PRISM probabilistic model checker
  - Markov decision processes (MDPs)
    - also discrete/continuous time Markov chains (D/CTMCs)
  - model checking of PCTL (and CSL) + extensions
  - efficient symbolic (MTBDD) implementation
- Recent/ongoing functionality improvements
  - discrete-event simulation engine
    - approximate results (sampling) and debugging tool
  - cost/reward-based property analysis
  - improved tool links: e.g. CADP (bisimulation tools)
  - counterexample generation

Research areas

- Efficiency improvements
  - symbolic (BDD, MTBDD) implementations
  - parallelisation, grid computing
- Model checking algorithms
  - symmetry reduction
  - abstraction techniques for MDPs
  - partial order reduction (with Baier et al.)
  - compositionality
- Additional models, formalisms, ..
  - real-time probabilistic model checking (PTAs)
  - probabilistic calculi for mobility (π-calculus, ambients)

Research areas...

- Applications of probabilistic model checking
  - ubiquitous computing systems: network protocols, embedded systems, mobile ad-hoc network protocols, ...
    - Bluetooth, Zeroconf, 802.11 WLANs, Zigbee
  - security protocols
    - probabilistic contract signing (with Shmatikov), anonymity
  - systems biology: Computational modelling and analysis
    - continuous-time Markov chains (CTMCs)
    - signalling pathways: cyclin, FGF, ecoli (σ_{32})

Symmetry reduction in PRISM [CAV'06]

- Full (component) symmetry in MDPs (and D/CTMCs)
  - system of interchangeable but non-trivial components
    - e.g. randomised distributed algorithms
  - induced quotient model up to factorially smaller
  - strong probabilistic bisimulation => preserves PCTL
- Symbolic (MTBDD-based) algorithm
  - construct full model first (actually smaller: more regularity)
  - construct quotient model via bubblesort
- Implementation: prototype extension of PRISM
  - promising results on a range of cases studies (randomised protocols: CSMA/CD, consensus, Byzantine agreement)

Probabilistic π-calculus model checking

Probabilistic π-calculus model checking

- π-calculus
  - modelling concurrency and mobility
  - applications: e.g. cryptographic protocols, mobile communication protocols
- Probabilistic π-calculus
  - adds discrete probabilistic choice
  - applications: randomised algorithms, failures, ...
  - e.g. probabilistic security protocols, mobile ad-hoc network protocols
- Currently, no tool support

(Simple) probabilistic π-calculus

- Syntax: P :: =
  - $0 \quad | \quad \alpha.P \quad | \quad P + P \quad | \quad \Sigma_i\, p_i\, \tau.P_i \quad |$

    (null) (prefix) (nondet. choice) (internal probabilistic choice)

    $P \mid P \quad | \quad \nu x\ P \quad | \quad [x=y]\ P \quad | \quad A(y_1,\ldots,y_n)$

    (parallel) (restriction) (match) (identifier)
  - α ::= in(x,y) | out(x,y) | τ
- Semantics: probabilistic automata (Segala/Lynch)
- Restrictions
  - finite control (no recursion within parallel composition)
  - input closed (no inputs from environment)

Example: DCP

- Dining cryptographers protocol (DCP)
  - **Master** = out(m_0,pay).out(m_1,not_pay).out(m_2,not_pay).0
    \+ out(m_0,not_pay).out(m_1,pay).out(m_2,not_pay).0 + ...
  - **Crypt0** = in(m_0,x).out(s_0,-),out(s_1,-).in(c_{00},y).in(c_{01},z).
    if x=pay then out(pay,-).
    if y=z out(o_0,agree).0 else out(o_0,disagree).0
    else
    if y=z out(o_0,disagree).0 else out(o_0,agree).0
  - **Coin0** = in(s_0,-).in(s_1,-) 0.5 : tau.out(c_{00},head).out(c_{01},head).0
    \+ 0.5 : tau.out(c_{00},tail).out(c_{01},tail).0
  - **DCP** = ν m_0,m_1,m_2 (**Master** | ν c_{00},c_{01},...,s_{00},s_{01},...
    (**Crypt0** | **Crypt1** | **Crypt2** | **Coin0** | **Coin1** | **Coin2**))

Combine existing tools

- MMC: Mobility Model Checker (Stony Brook)
  - finite-control π-calculus, model checking for μ-calculus
  - logic programming: built on XSB Prolog
- PRISM: Probabilistic Symbolic Model Checker
  - Markov decision processes (also discrete/cont. Markov chains)
  - simple state-based modelling language:
    - modules, finite-valued variables, guarded commands, synchronisation, ...

MMC to PRISM

- Modifications/extensions of MMC
  - generation of symbolic transition graph
  - add probabilistic version of choice operator to MMC
- Possible routes for MMC to PRISM
  - direct construction of underlying data structures (MTBDDs)
  - generation/import of full MDP (matrix)
  - language-level translation (monolithic – one module)
  - language-level translation (compositional)
    - avoids product state-space blow-up
    - preserve regularity to decrease BDD size

Compositional translation

- Translate MMC π-calc. processes to PRISM modules
  - require description in form $P_1 \mid P_2 \mid \ldots \mid P_n$
  - P_i can contain local nondeterminism (choice, parallel)
  - translate each P_i in MMC
  - symbolic transition graph for each process
- DCP example
  - $\nu\, m_0, m_1, m_2$ (**Master** | $\nu\, c_{00}, c_{01}, \ldots, s_{00}, s_{01}, \ldots$ (**Crypt0** | **Crypt1** | **Crypt2** | **Coin0** | **Coin1** | **Coin2**))
  - $\nu\, m_0, m_1, m_2, c_{00}, c_{01}, \ldots, s_{00}, s_{01}, \ldots$ (**Master** |**Crypt0** | **Crypt1** | **Crypt2** | **Coin0** | **Coin1** | **Coin2**)

Symbolic transition graph: coin0

Free names: s00, s20, c00, c20, head, tail

Bound names: _h481, _h487

States:

#1: proc(coin(s00,s20,c00,c20,head,tail))

#2: pref(in(s20,_h487),prob_choice([pref(tau(0.5),proc(face (c00,c20,head))),pref(tau(0.5),proc(face(c00,c20,tail)))]))

...

Transitions:

*1: 1 -- 1:in(s00,_h481) --> 2

*2: 2 -- 1:in(s20,_h487) --> 3

*3: 3 -- 0.5:tau --> 4, 0.5:tau --> 5

...

Modelling channel communication

- One possibility
  - introduce PRISM variables for buffers
  - break communication into steps: read/write/ack
  - blow-up due to additional interleavings
- Map channels in π-calc. to synchronisation in PRISM
  - π-calc: binary synchronisation (CCS), name passing
  - PRISM: multi-way synchr. (CSP), no value/name passing
  - translation scheme: encode all info in action name

Modelling channel communication...

PRISM code:

```
const int a;

module P

    P_state : [1..P_n];

    [x_P_Q_a] P_state=1 -> (P_state'=2);

endmodule

module Q

    Q_state : [1..Q_n];

    Q_y : [1..y_n];

    [x_P_Q_a] Q_state=1 -> (Q_state'=2) & (Q_y'=a);

endmodule
```

P = out(x,a).P'
Q = in(x,y).Q'

(where a is a free name)

Modelling channel communication...

PRISM code:

```
const int a;
const int b;
module P
    P_state : [1..P_n];
    [x_P_Q_a] P_state=1 -> (P_state'=2);
    [x_P_Q_b] P_state=1 -> (P_state'=3);
endmodule
module Q
    Q_state : [1..Q_n];
    Q_y : [1..y_n];
    [x_P_Q_a] Q_state=1 -> (Q_state'=2) & (Q_y'=a);
    [x_P_Q_b] Q_state=1 -> (Q_state'=2) & (Q_y'=b);
endmodule
```

P = out(x,a).P' + out(x,b).P''
Q = in(x,y).Q'

(where a,b are free names)

Modelling channel communication...

PRISM code:

```
module P
    P_state : [1..P_n];
    P_z : [1..z_n];
    [x_P_Q_z] P_state=1 -> (P_state'=2);
endmodule
module Q
    Q_state : [1..Q_n];
    Q_y : [1..y_n];
    [x_P_Q_z] Q_state=1 -> (Q_state'=2) & (Q_y'=P_z);
endmodule
```

$P = \nu z\ out(x,z).P'$
$Q = in(x,y).Q'$

(where z is a bound name)

Implementation

- Fully automatic translation/construction of model
  - MMC (+extensions) & Java code & PRISM
  - currently static configurations only
    - all channels (and their contents) are constants (free names)
- Algorithm:
  - identify all possible senders/receivers on each channel
  - identify all names sent along each channel
  - identify which names can be assigned to each bound name
- Fully automatic translation of DCP example
  - compute min/max probability of each observable in PRISM

Current/future work

- Extend/improve translation process
  - polyadic π-calculus, e.g. out(x,(a,b))
  - scope extrusion: sending private channel names
  - translate properties too
    - action vs. state based properties
- Another simple example: Partial Secret Exchange
- More complex case studies (with mobility)
- Stochastic π-calculus, CTMCs, biological case studies

Game-based abstraction of Markov decision processes

Model checking for MDPs

- Probabilistic model checking for MDPs
  - temporal logic PCTL: probabilistic reachability
  - probability only defined for a single adversary/scheduler
  - minimum/maximum probabilities (best/worst case)
  - also: expected cost/reward to reach...
- Typically focus on quantitative properties
  - e.g. “what is the minimum probability of reaching...”?
- Tool support for automatic verification, e.g. PRISM
  - iterative methods (dynamic programming)
  - efficient symbolic (MTBDD) implementations, but...
  - state space explosion still a major issue

Abstraction

- Very successful in (non-probabilistic) model checking
- Construct abstract model M' from concrete model M
  - details not relevant to property of interest removed
  - merge states according to a given partition of state space
  - e.g. from set of predicates
- Conservative abstraction
  - satisfaction of property in M' implies satisfaction in M
  - converse does not hold, but...
  - information from model checking process
    (e.g. counterexample) can be used to refine the abstraction

Abstraction of MDPs

- Abstraction increases degree of nondeterminism
  - min probability may be smaller, max may be larger
- Key idea: separate two forms of nondeterminism
  - (a) from abstraction and (b) from original MDP
- Generate separate lower/upper bounds for min/max
  - especially useful if min/max probs not close
  - worst-case: pmin=0, pmax=1
- If lower/upper bounds not close enough,
  - refine abstraction and repeat

Simple stochastic games (SSGs)

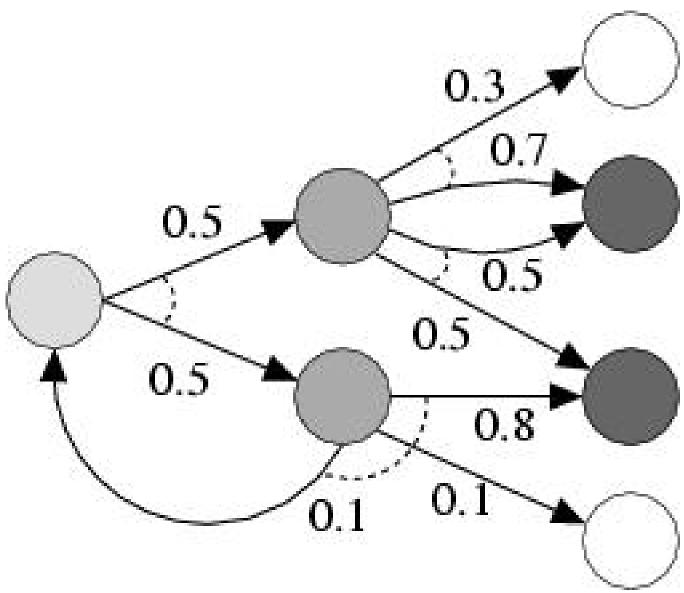
- Simple stochastic two-player games [Condon'92]
- Game $G = ((V,E),v_{init},(V_1,V_2,V_P),\delta)$
  - (V,E) is a finite directed graph
  - v_{init} is the initial vertex
  - (V_1,V_2,V_P) is a partition of V into 'player 1', 'player 2' and 'probabilistic' vertices
  - $\delta : V_P \rightarrow Dist(V)$ is a probabilistic transition function
- Execution of G: successor in each vertex chosen...
  - by player 1/2 for V_1/V_2 vertices, at random (δ) for V_P vertices

Abstract MDP = SSG

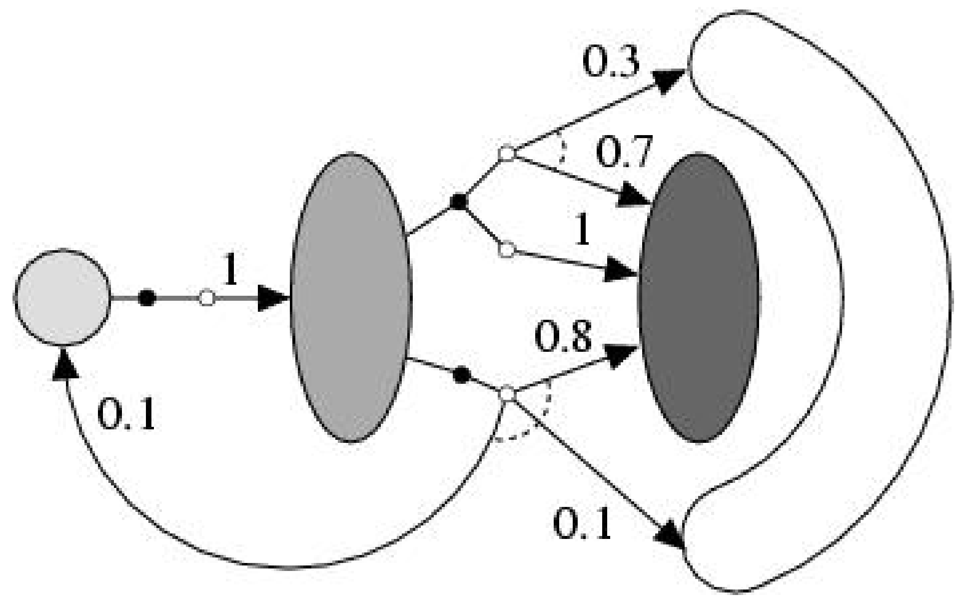
- Player 1 controls nondeterminism from abstraction
- Player 2 controls nondeterminism from original MDP
- Strict alternation between V_1, V_2, V_P vertices
- Based on a partition P of MDP state space S
  - V_1 states are elements of P (subsets of S)
  - V_2 states are sets of probability distributions
  - V_P states are single probability distributions from MDP

Simple example

Original MDP



Abstract MDP
(simple stochastic game)



Analysis

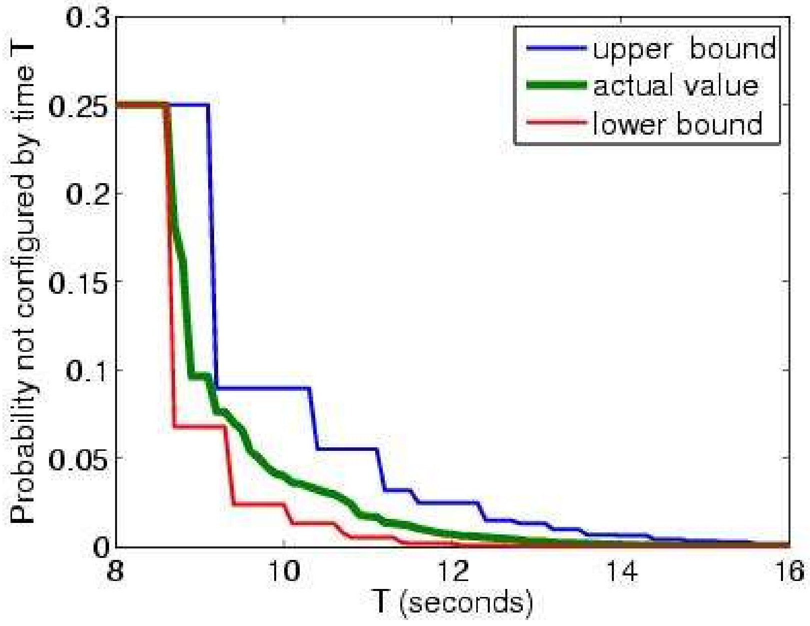
- Analysis of SSGs: reachability of vertex goal set F
  - $p_{a1,a2}(F)$: probability reach F under player strategies a1,a2
  - optimal probabilities for player 1 and player 2:
  - $\sup_{a1} \inf_{a2} p_{a1,a2}(F)$ and $\sup_{a2} \inf_{a1} p_{a1,a2}(F)$
  - computable via iterative method, similar to MDPs
- Compute bounds for pmin(F) and pmax(F) in MDP
  - $\inf_{a1,a2} p_{a1,a2}(F) \leq \text{pmin}(F) \leq \sup_{a1} \inf_{a2} p_{a1,a2}(F)$
  - $\sup_{a2} \inf_{a1} p_{a1,a2}(F) \leq \text{pmax}(F) \leq \sup_{a1,a2} p_{a1,a2}(F)$

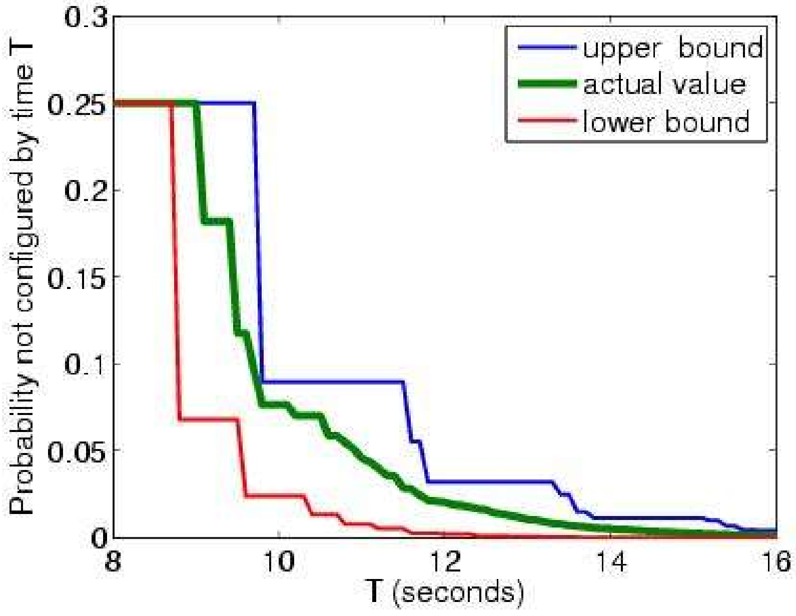
Case study: Zeroconf protocol

- Decentralised self configuration of local IP addresses
  - new node joining network of N existing nodes, M addresses
  - probabilistic: based on random selection of IP address
  - nondeterministic: concurrency from scheduling, unknown message propagation delays (different range for each node)
- Abstraction
  - abstract M address to 2 values: fresh/in-use
  - channels: just store type of message, not sender
    - lose information about message timings

Results

- Substantial reduction in model size, e.g. (for N=8,M=32)
  - MDP: 432,185 states, 1,244,480 transitions
  - Abstract MDP (SSG): 881 states, 1,850 transitions
- Min/max probability not configured by time T:





Future work

- Perform abstraction at PRISM language level
  - bypass construction of full MDP
  - infinite-state MDPs?
- Efficient symbolic implementation of SSG algorithms
  - very similar to existing PRISM algorithms for MDPs
- Automatic/semi-automatic generation of partitions