Probabilistic model checking with PRISM: overview and recent developments

Marta Kwiatkowska

Department of Computer Science, University of Oxford

QMC 2016, Singapore, September 2016
Quantitative/probabilistic verification

- Employ (quantitative) formal models
  - can be derived or extracted from code
  - can also be used at runtime

- Specify goals/objectives/properties in temporal logic:
  - reliability, energy efficiency, performance, resource usage, ...
  - (reliability) “alert signal will be delivered with high probability in 10ms”
  - (energy) “maximum expected energy consumption in 1 hr is at most 10mA”
  - (performance) “the packet will be delivered with high probability in 10ms”

- Focus on automated, tool–supported methodologies
  - model–based design
  - model checking against temporal logic specifications
  - in conjunction with quantitative analysis
Quantitative (probabilistic) verification

Automatic verification (aka model checking) of **quantitative** properties of probabilistic system models

- **System**
  - Probabilistic model
    - e.g. Markov chain
  - System requirements
  - Probabilistic temporal logic specification
    - e.g. PCTL, CSL, LTL

- **Probabilistic model checker**
  - e.g. PRISM

- **Result**
  - ✓
  - ✗

- **Quantitative results**

- **Counter-example**
Historical perspective

- **First algorithms proposed in 1980s**
  - algorithms [Vardi, Courcoubetis, Yannakakis, …]
  - [Hansson, Jonsson, de Alfaro] & first implementations

- **2000: general purpose tools released**
  - PRISM: efficient extensions of symbolic model checking
    [Kwiatkowska, Norman, Parker, …]
  - ETMCC: model checking for continuous-time Markov chains [Baier, Hermanns, Haerkort, Katoen, …]

- **Now mature area, of industrial relevance**
  - successfully used by non-experts for many application domains, but full automation and good tool support essential
  - distributed algorithms, communication protocols, security protocols, biological systems, quantum cryptography, planning, …
  - genuine flaws found and corrected in real-world systems
  - [www.prismmodelchecker.org](http://www.prismmodelchecker.org)
PRISM

- **PRISM: Probabilistic symbolic model checker**
  - developed at Birmingham/Oxford University, since 1999
  - free, open source software (GPL), runs on all major OSs
- **Support for:**
  - models: DTMCs, CTMCs, MDPs, PTAs, SMGs, …
  - properties: PCTL/PCTL*, CSL, LTL, rPATL, costs/rewards, …
- **Features:**
  - simple but flexible high-level modelling language
  - user interface: editors, simulator, experiments, graph plotting
  - multiple efficient model checking engines (e.g. symbolic)
- **Many import/export options, tool connections**
  - MRMC, PEPA, Cosmos, DSD, ProFeat, Matlab, …
- **See:** [http://www.prismmodelchecker.org/](http://www.prismmodelchecker.org/)
## Probabilistic models

<table>
<thead>
<tr>
<th>Discrete time</th>
<th>Fully probabilistic</th>
<th>Nondeterministic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discrete-time Markov chains (DTMCs)</td>
<td></td>
<td>Markov decision processes (MDPs)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Simple stochastic games (SMGs)</td>
</tr>
<tr>
<td>Continuous time</td>
<td>Continuous-time Markov chains (CTMCs)</td>
<td>Probabilistic timed automata (PTAs)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Interactive Markov chains (IMCs)</td>
</tr>
</tbody>
</table>
Probabilistic models: discrete time

- **Discrete–time Markov chains (DTMCs)**
  - discrete states + **discrete probability**
  - for: randomisation, unreliable communication media, …

- **Markov decision processes (MDPs)**
  - discrete states + discrete probability + **nondeterminism** (e.g. for concurrency, control)
  - for: randomised distributed algorithms, security protocols, …

- **Stochastic multi–player games (SMGs)**
  - discrete states + discrete probability + **player nondeterminism** (e.g. for collaboration and competition)
  - for: robotics, user–centric networks, smartgrid protocols, …

NB all supported by PRISM
Probabilistic models: continuous time

• **Continuous–time Markov chains (CTMCs)**
  – discrete states + **exponentially distributed delays**
  – for: component failures, job arrivals, molecular reactions, ...

• **Interactive Markov chains (CTMCs)**
  – discrete states + exponentially distributed delays + **nondeterminism**
  – for: job arrivals in a distributed environment

• **Probabilistic timed automata (PTAs)**
  – discrete probability, nondeterminism + **real–time**
  – for wireless comm. protocols, embedded control systems, ...

NB  Shown in grey not supported by PRISM
Probabilistic models: continuous space

- Labelled Markov processes (LMPs)
  - continuous states + stochastically distributed jumps
  - for: aircraft controllers, search and rescue, …

- Probabilistic hybrid automata (PHAs)
  - discrete probability, nondeterminism + continuous flows (ODEs)
  - for embedded control systems, automotive controllers, …
  - partial support exists, e.g. tool chain PHAVER+PRISM

- Stochastic hybrid systems (SHSs)
  - continuous probability, nondeterminism + continuous flows
  - for embedded control systems, automotive controllers, …

NB Shown in grey not supported by PRISM
Markov decision processes (MDPs)

- Model nondeterministic as well as probabilistic behaviour
  - e.g. for concurrency, under-specification, abstraction...
  - extension of discrete-time Markov chains
  - nondeterministic choice between probability distributions

- Formally, an MDP is a tuple
  - \((S, s_{\text{init}}, \text{Act}, \delta, L)\)

- where:
  - \(S\) is a set of states
  - \(s_{\text{init}} \in S\) is the initial state
  - \(\delta : S \times \text{Act} \rightarrow \text{Dist}(S)\) is a (partial) transition probability function
  - \(L : S \rightarrow 2^{\text{AP}}\) is a labelling function
  - \(\text{Act}\) is a set of actions, \(\text{AP}\) is a set of atomic propositions
  - \(\text{Dist}(S)\) is the set of discrete probability distributions over \(S\)
Paths and strategies

- **A (finite or infinite) path through an MDP**
  - is a sequence \((s_0...s_n)\) of (connected) states
  - represents an execution of the system
  - resolves both the probabilistic and nondeterministic choices

- **A strategy \(\sigma\) (aka. “adversary” or “policy”) of an MDP**
  - is a resolution of nondeterminism only
  - is (formally) a mapping from finite paths to distributions
  - induces a fully probabilistic model
  - i.e. an (infinite-state) Markov chain over finite paths
  - on which we can define a probability space over infinite paths
Probabilistic model checking for MDPs

• To reason formally about MDPs, we use adversaries
  – an adversary $\sigma$ makes a (possibly randomised) choice
  – induces probability measure $Pr^\sigma$ over (infinite) paths

• Let $\psi$ be a temporal path property
  – specify probabilistic property $P_{\geq p}[\psi]$
  – $Pr^\sigma(\psi)$ gives probability of $\psi$ under adversary $\sigma$
  – best-/worst-case analysis: quantify over all adversaries
  – e.g. $M \models P_{\geq p}[\psi] \iff Pr^\sigma(\psi) \geq p$ for all $\sigma$
  – or just compute e.g. $Pr_{min}(\psi) = \inf_{\sigma} Pr^\sigma_s(\psi)$
  – proceeds by value iteration or linear programming

• Extend to CTL and LTL (product with property automaton)
  – logic PCTL/PCTL*, Probabilistic LTL
From verification to synthesis

• Shift towards quantitative model synthesis from specification
  – begin with simpler problems: strategy synthesis, template-based synthesis, etc
  – advantage: correct-by-construction

• Here consider the problem of strategy (controller) synthesis
  – i.e. “can we construct a strategy to guarantee that a given quantitative property is satisfied?”

• also parameter synthesis:
  – i.e. “find optimal value for parameter to satisfy quantitative objective”

• Many application domains
  – robotics (controller synthesis from temporal objectives
  – dynamic power management (optimal policy synthesis)
Quantitative/probabilistic verification

Automatic verification and **strategy synthesis** from quantitative properties for probabilistic models

- System
- Probabilistic model
  - e.g. Markov chain
- Probabilistic temporal logic specification
  - e.g. PCTL, CSL, LTL
- System requirements
- Probabilistic model checker
  - e.g. PRISM
- Result
- Quantitative results
- Strategy
- \( P_{<0.01} [ F_{\leq t} \text{ crash} ] \)
Quantitative parameter synthesis

Parametric model
e.g. Markov chain

Probabilistic model checker
PRISM PARAM

Result

Quantitative results

Concrete model

System requirements

Probabilistic temporal logic specification
e.g. PCTL, CSL, LTL

System

0.5 + x, 0.4 - x

0.1

P < 0.01 [ F ≤ t fail ]

0.6, 0.3

0.1
Verification and strategy synthesis

- **The verification problem is:**
  - Given an MDP $M$ and a property $\phi$, does $M$ satisfy $\phi$ under any possible strategy $\sigma$?

- **The synthesis problem is dual:**
  - Given an MDP $M$ and a property $\phi$, find, if it exists, a strategy $\sigma$ such that $M$ satisfies $\phi$ under $\sigma$

- Verification and strategy synthesis is achieved using the same techniques, namely computing optimal values for probability objectives:
  - $Pr_s^{\min}(\psi) = \inf_\sigma Pr_s^\sigma(\psi)$
  - $Pr_s^{\max}(\psi) = \sup_\sigma Pr_s^\sigma(\psi)$
  - and similarly for expectations
Strategy synthesis

- Compute optimal probabilities $\Pr_{s}^{\text{max}}(F b)$ for all $s \in S$

- To compute the optimal strategy $\sigma^*$, choose the locally optimal action in each state
  - must guarantee progress towards target states
  - in general depends on the method used to compute the optimal probabilities

- For reachability
  - memoryless strategies suffice

- For step-bounded reachability
  - need finite-memory strategies
  - typically requires backward computation for a fixed number of steps
Strategy synthesis for LTL objectives

- Reduce to the problem of reachability on the product of MDP M and an omega-automaton representing ψ
  - for example, deterministic Rabin automaton (DRA)

- Need only consider computation of maximum probabilities $Pr_s^{\text{max}}(\psi)$
  - since $Pr_s^{\text{min}}(\psi) = 1 - Pr_s^{\text{max}}(\neg \psi)$

- To compute the optimal strategy $\sigma^*$
  - find memoryless deterministic strategy on the product
  - convert to finite-memory strategy with one mode for each state of the DRA for ψ
The remaining models...

- **Probabilistic model checking for discrete–time Markov chains**
  - PCTL model checking reduces to linear equation solving
  - LTL/PCTL* via automata-theoretic methods
- **Continuous–time Markov Chains (CTMCs)**
  - no nondeterminism: rate of transition, real–valued time
  - model checking via uniformisation, a discretisation of the CTMC
- **Labelled Markov processes**
  - no nondeterminism, continuous flows
  - discretisation to labelled Markov chains
- **(Subclasses of) probabilistic timed/hybrid automata**
  - nondeterminism, continuous flows
  - under various restrictions, abstract continuous variables and/or discretise into Markov decision processes
Overview

• Introduction

• PRISM: current status
  – modelling language
  – properties
  – functionality
  – case studies
  – extensions/connections

• PRISM: major recent developments
  1. probabilistic timed automata
  2. parametric models
  3. multi-objective model checking
  4. stochastic games: PRISM-games 2.0

• Summary
• **PRISM: Probabilistic symbolic model checker**
  – developed at Birmingham/Oxford University, since 1999
  – free, open source software (GPL), runs on all major OSs
• **Support for:**
  – models: DTMCs, CTMCs, MDPs, PTAs, SMGs, …
  – properties: PCTL/PCTL*, CSL, LTL, rPATL, costs/rewards, …
• **Features:**
  – simple but flexible high-level modelling language
  – user interface: editors, simulator, experiments, graph plotting
  – multiple efficient model checking engines (e.g. symbolic)
• **Many import/export options, tool connections**
  – MRMC, PEPA, Cosmos, DSD, ProFeat, Matlab, …
• **See:** [http://www.prismmodelchecker.org/](http://www.prismmodelchecker.org/)
PRISM modelling language

• Simple, textual, state-based modelling language
  – used for all probabilistic models supported by PRISM
  – based on Reactive Modules [AH99]

• Language basics
  – system built as parallel composition of interacting modules
  – state of each module given by finite-ranging variables
  – behaviour of each module specified by guarded commands
    + annotated with probabilities/rates and (optional) action label
  – transitions are associated with state-dependent probabilities
  – interactions between modules through synchronisation

\[
\text{[send]} \ (s=2) \rightarrow p_{\text{loss}} : (s'=3) \& (\text{lost}'=\text{lost}+1) \ + \ (1-p_{\text{loss}}) : (s'=4);
\]
dtmc
module die
// local state s : [0..7] init 0;
// value of the dice d : [0..6] init 0;
[] s=0 -> 0.5 : (s'=1) + 0.5 : (s'=2);
...
[] s=3 ->
  0.5 : (s'=1) + 0.5 : (s'=7) & (d'=1);
[] s=4 ->
  0.5 : (s'=7) & (d'=2) + 0.5 : (s'=7) & (d'=3);
...
[] s=7 -> (s'=7);
endmodule
rewards "coin_flips"
[] s<7 : 1;
endrewards
int s, d;

s = 0; d = 0;
while (s < 7) {
    bool coin = Bernoulli(0.5);
    if (s == 0)
        if (coin) s = 1 else s = 2;
...
    else if (s == 3)
        if (coin) s = 1 else {s = 7; d = 1;}
    else if (s == 4)
        if (coin) {s = 7; d = 2} else {s = 7; d = 3;}
...
}
return (d)

Given as a probabilistic program
Costs and rewards

- We augment models with rewards (or, conversely, costs)
  - real-valued quantities assigned to states and/or transitions
  - these can have a wide range of possible interpretations

- Some examples:
  - elapsed time, power consumption, size of message queue, number of messages successfully delivered, net profit, ...

- Costs? or rewards?
  - mathematically, no distinction between rewards and costs
  - when interpreted, we assume that it is desirable to minimise costs and to maximise rewards
  - we consistently use the terminology “rewards” regardless

- Properties (see later)
  - reason about expected cumulative/instantaneous reward
Rewards in the PRISM language

Rewards "total_queue_size"
  true : queue1 + queue2;
endrewards

(instantaneous, state rewards)

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

(cumulative, transition rewards)
(q = queue size, q_max = max. queue size, receive = action label)

Rewards "time"
  true : 1;
endrewards

(cumulative, state rewards)

Rewards "power"
  sleep=true : 0.25;
  sleep=false : 1.2 * up;
  [wake] true : 3.2;
endrewards

(cumulative, state/trans. rewards)
(up = num. operational components, wake = action label)
PRISM – Property specification

- Temporal logic–based property specification language
  - subsumes PCTL, CSL, probabilistic LTL, PCTL*, ...

- 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?”
  - then analyse trends in quantitative properties as system parameters vary
PRISM – Property specification

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

• Reward–based properties (rewards = costs = prices)
  – $R_{\text{"time"}=?} [ F \text{ "end" } ]$ – “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. $P_{=?} [ F \text{ fail}_1 ] / P_{=?} [ F \text{ fail}_\text{any} ]$ – “conditional failure prob.”
Graphical user interface
PRISM GUI: The Simulator
PRISM GUI: Model checking and graphs
PRISM – Underlying techniques

• Construction and analysis of finite probabilistic models
  – specified in high-level modelling formalisms

• Basic ingredients
  – graph-based algorithms, e.g. reachability, qualitative verif.
  – numerical solution techniques, e.g. probability computation
  – usually rely on iterative methods: uniformisation-based for transient properties, Gauss–Seidel/etc. for linear equations
  – Monte Carlo simulation
  – simulation-based approximate model checking

• Symbolic implementation
  – data structures based on binary decision diagrams
  – fast construction + compact storage of huge models possible
  – exploit structure, regularity in high-level model
  – usually: up to $10^7$–$10^8$ states; sometimes: up to $10^{10}$ states
• **Approximate (statistical) probabilistic model checking**
  - discrete event (Monte Carlo) simulation + sampling

• **Two distinct approaches (both implemented in PRISM)**
  - **Estimation** [Hérault et al.]
    - approximate result for quantitative query such as $P_{\equiv_p} [ \phi ]$
    - plus a probabilistic guarantee regarding result precision
    - $\text{Prob}( |p_{\text{actual}} - p_{\text{estimated}}| \leq \epsilon ) \geq 1 - \delta$
    - can also generate corresponding confidence intervals
  - **Hypothesis testing/acceptance sampling** [Younes/Simmons]
    - applied to boolean-valued queries such as $P_{\sim_p} [ \phi ]$
    - basic idea: stop sampling as soon as the result can be shown to be either true or false with high probability
    - sensitive to distance between bound $p$ and actual answer
PRISM technicalities

• **PRISM: multiple computation engines**
  – MTBDDs (BDD extension): storage/analysis of very large models, numerical computation can blow up
  – sparse matrices: fastest solution for smaller models
  – hybrid: combine MTBDD storage with explicit storage
  – explicit: state enumeration

• **Underlying computation involves**
  – Reachability, qualitative model checking, BDD-based
  – Linear equation system solution – Jacobi, Gauss–Seidel, ...
  – Uniformisation/fast adaptive uniformisation (CTMCs)
  – Dynamic programming (MDPs)
  – Symmetry reduction (explicit and symbolic) [CAV’06]
  – Quantitative abstraction refinement [QEST’06]
  – Simulation and sampling-based (DTMCs, CTMCs) [CAV’11]
  – Constraint solving (strategy/parameter synthesis)
  – Efficient generation of automata for LTL [CAV’15]
Extensions/Connections

- Symbolic extensions (to be integrated into PRISM) (Baier et al [TACAS’16])
  - variable reordering
  - conditional probabilities
  - quantiles (trade offs)
- ProFeat (Baier et al [FASE’16])
  - family-based modelling and analysis a’la software product lines and feature-oriented systems
  - translation into PRISM’s input language
  - supports arrays, parametrization, etc
- Connection to the Cosmos tool for Petri nets and LHA specifications (Kwi&Barbot [CMSB’15])
  - fast statistical model checking
  - code generation and parallelisation

• Thanks to several groups and Google Summer of Code!!!
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](http://www.prismmodelchecker.org/casestudies)
Case study: Bluetooth

- **Device discovery between pair of Bluetooth devices**
  - performance essential for this phase

- **Complex discovery process**
  - two asynchronous 28-bit clocks
  - pseudo-random hopping between 32 frequencies
  - random waiting scheme to avoid collisions
  - 17,179,869,184 initial configurations
    (too many to sample effectively)

- **Probabilistic model checking**
  - e.g. “worst-case expected discovery time is at most 5.17s”
  - e.g. “probability discovery time exceeds 6s is always < 0.001”
  - shows weaknesses in simplistic analysis

\[
\text{freq} = \left[\text{CLK}_{15-12} + k + (\text{CLK}_{4-2} - \text{CLK}_{15-12}) \mod 16\right] \mod 32
\]
• DNA: easily accessible, cheap to synthesise information processing material
• DNA Strand Displacement language, induces CTMC models
  – for designing DNA circuits [Cardelli, Phillips, et al.]
  – accompanying software tool for analysis/simulation
  – now extended to include auto-generation of PRISM models
• Transducer: converts input \(<t^x>\) into output \(<y t^>\)

\[
\begin{array}{c}
\text{t} \\
\text{x}
\end{array} \hspace{1cm} \begin{array}{c}
\text{a} \\
\text{t}
\end{array} \hspace{1cm} \begin{array}{c}
\text{y} \\
\text{t}
\end{array}
\]

• Formalising correctness: does it finish successfully?...
  – A [ G "deadlock" => "all_done" ]
  – E [ F "all_done" ]

(CTL, but probabilistic also...)
Transducer flaw

- PRISM identifies a 5-step trace to the “bad” deadlock state
  - problem caused by “crosstalk” (interference) between DSD species from the two copies of the gates
  - previously found manually [Cardelli’10]
  - detection now fully automated

- Bug is easily fixed (and verified)

Counterexample:

\[(1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)\]
\[(0,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)\]
\[(0,0,1,1,1,1,1,1,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)\]
\[(0,0,1,1,1,1,1,1,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)\]
\[(0,0,1,1,1,1,1,1,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)\]
\[(0,0,1,1,1,1,1,1,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)\]
1. 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
```
Modelling PTAs in PRISM

- **PRISM modelling language**
  - textual language, based on guarded commands

```plaintext
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
```
Modelling PTAs in PRISM

- **PRISM modelling language**
  - textual language, based on guarded commands

```plaintext
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
Modelling PTAs in PRISM

• PRISM modelling language
  – textual language, based on guarded commands

```
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
Modelling PTAs in PRISM

- **PRISM modelling language**
  - textual language, based on guarded commands

```
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)
Model checking PTAs in PRISM

- **Properties for PTAs:**
  - min/max probability of reaching X (within time T)
  - min/max expected cost/reward to reach X
    (for “linearly-priced” PTAs, i.e. reward gain linear with time)

- **“Digital clocks” – conversion to finite-state MDP**
  - preserves min/max probability + expected cost/reward/price
  - (for PTAs with closed, diagonal-free constraints)
  - can be 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

- **Backwards reachability**
  - more efficient zone-based abstraction in some cases
Case study: FireWire root contention

- **FireWire (IEEE 1394)**
  - high-performance serial bus for networking multimedia devices; originally by Apple
  - "hot-pluggable" – add/remove devices at any time
  - no requirement for a single PC (but need acyclic topology)

- **Root contention protocol**
  - leader election algorithm, when nodes join/leave
  - symmetric, distributed protocol
  - uses **randomisation** (electronic coin tossing) and **timing** delays
  - nodes send messages: "be my parent"
  - root contention: when nodes contend leadership
  - random choice: "fast"/"slow" delay before retry
Case study: FireWire root contention

• **Detailed probabilistic model:**
  - probabilistic timed automaton (PTA), including:
    • concurrency: messages between nodes and wires
    • timing delays taken from official standard
    • underspecification of delays (upper/lower bounds)
  - maximum model size: 170 million states

• **Probabilistic model checking (with PRISM)**
  - verified that root contention always resolved with probability 1
    • $P_{\geq 1} [ F (\text{end } \wedge \text{elected}) ]$
  - investigated worst-case expected time taken for protocol to complete
    • $R_{\text{max}=?} [ F (\text{end } \wedge \text{elected}) ]$
  - investigated the effect of using biased coin
Case study: FireWire root contention

“minimum probability of electing leader by time $T$” (using a biased coin)

“maximum expected time to elect a leader” (using a biased coin)
2. Parametric models in PRISM

- **Parametric Markov chain models in PRISM**
  - probabilistic parameters expressed as unevaluated constants
  - e.g. const double \( x \);
  - transition probabilities are expressions over parameters, e.g. \( 0.4 + x \)

- **Properties are given in PCTL, with parameter constants**
  - new construct constfilter (min, \( x1\times x2 \), \( \phi \))
  - filters over parameter values, rather than states

- **Implemented in ‘explicit’ engine**
  - returns mapping from parameter regions (e.g. \([0.2,0.3],[-2,0]\)) to rational functions over the parameters
  - filter properties used to find parameter values that optimise the function
  - reimplementation of PARAM 2.0 [Hahn et al]
Case study: parametric models

Checking if minimal exp. number of attacks $\geq 20$

Property $\text{constfilter}(\min, \ldots, R_{\text{"attacks"}} \geq 20 \ [ \ F \ "end"])$

Model (network virus) has 809 states, $\epsilon = 0.05$

Optimal value found in 2mins, showing optimal parameter values
Quantitative parameter synthesis

System

Parametric model
e.g. Markov chain

Probabilistic temporal
logic specification
e.g. PCTL, CSL, LTL

Result

Quantitative results

Concrete model

P<0.01 [ F≤t fail]
• So far parameters are probabilities in DTMC/MDP
  – see also tool Prophecy (Katoen et al, [CAV’15])
• Can we implement parameter synthesis for probabilistic real–time systems?

• The parameter synthesis problem we consider
  – given a parametric model and property \( \phi \)
  – find the optimal (controllable) parameter values, with respect to an objective function \( O \), such that the property \( \phi \) is satisfied for all uncontrollable parameters, if such values exist

• Parameters: timing delays, rates
• Objectives: optimise probability, reward/volume

• Beyond PRISM…
Three frameworks

• **Timed automata**: find optimal timing delays [EMSOFT’14, HSB’15]
  – timed I/O automata, Counting MTL specifications
  – quantitative objectives
  – solution: constraint solving, discretisation + sampling or evolutionary computation

• **Probabilistic timed automata**: find delays to optimise probability [RP2014]
  – solution: parametric symbolic abstraction-refinement based on stochastic games

• **Continuous-time Markov chains**: find optimal rates [CMSB’15]
  – time-bounded CSL
  – solution: constraint solving, uniformisation + sampling
  – GPU parallelisation
  – tool PRISM-PSY [TACAS’16]
3. Multi-objective model checking

- **Markov decision processes (MDPs)**
  - generalise DTMCs by adding **nondeterminism**
  - for: control, concurrency, abstraction, ...

- **Strategies (or "adversaries", "policies")**
  - resolve nondeterminism, i.e. choose an action in each state based on current history
  - a strategy induces an (infinite-state) DTMC

- **Verification (probabilistic model checking) of MDPs**
  - quantify over all possible strategies... (i.e. best/worst-case)
  - $P_{<0.01}[F\ err]: \ "the\ probability\ of\ an\ error\ is\ always\ <0.01"$

- **Strategy synthesis (dual problem)**
  - "does there exist a strategy for which the probability of an error occurring is < 0.01?"
  - “how to minimise expected run-time?”
Multi-objective model checking

- **Multi-objective probabilistic model checking**
  - investigate trade-offs between conflicting objectives
  - in PRISM, objectives are probabilistic LTL or expected rewards
- **Achievability queries**
  - e.g. “is there a strategy such that the probability of message transmission is > 0.95 and expected battery life > 10 hrs?”
  - $\text{multi}(P_{>0.95}[F \text{ transmit }], R_{\text{time} >10}[C])$
- **Numerical queries**
  - e.g. “maximum probability of message transmission, assuming expected battery life-time is > 10 hrs?”
  - $\text{multi}(P_{\max=?}[F \text{ transmit }], R_{\text{time} >10}[C])$
- **Pareto queries**
  - e.g. "Pareto curve for maximising probability of transmission and expected battery life-time”
  - $\text{multi}(P_{\max=?}[F \text{ transmit }], R_{\text{time} \max=?}[C])$
Case study: Dynamic power management

- Synthesis of dynamic power management schemes
  - for an IBM TravelStar VP disk drive
  - 5 different power modes: active, idle, idlelp, stby, sleep
  - power manager controller bases decisions on current power mode, disk request queue, etc.

- Build controllers that
  - minimise energy consumption, subject to constraints on e.g.
  - probability that a request waits more than K steps
  - expected number of lost disk requests

- See: http://www.prismmodelchecker.org/files/tacas11/
4. Stochastic multi-player games

- **Introduce stochastic multi-player games (SMGs)**
  - argue that games are an appropriate modelling abstraction for competitive behaviour, in adversarial environments
  - stochasticity to model e.g. failure, sensor uncertainty

- **Property specification: rPATL**
  - single-objective properties
  - verification
  - strategy synthesis

- **Extensions**
  - multiobjective properties, Pareto sets
  - compositional strategy synthesis

- **Tool support: PRISM-games 2.0**

- **Case studies**
Why games?

• Autonomous systems
  – are reactive, continuously interact with their environment
    • including other components or human users, adversarial
  – have goals/objectives
    • often quantitative, may conflict
  – take decisions based on current state and external events

• Natural to adopt a game-theoretic view
  – need to account for the uncontrollable behaviour of components, possibly with differing/opposing goals
  – in addition to controllable events

• Many occurrences in practice
  – e.g. decision making in economics, power distribution networks, controller synthesis, motion planning, security, distributed consensus, energy management, sensor network co-ordination, …
Playing games with Google car...

http://theoatmeal.com/blog/google_self_driving_car
“This is a classic example of the negotiation that’s a normal part of driving – we’re all trying to predict each other’s movements. In this case, we clearly bear some responsibility, because if our car hadn’t moved there wouldn’t have been a collision”. 
Stochastic multi-player games

- **Stochastic multi-player game (SMGs)**
  - multiple players + nondeterminism + probability
  - generalisation of MDPs: each state controlled by unique player

- **A (turn-based) SMG is a tuple** \((\Pi, S, \langle S_i \rangle_{i \in \Pi}, A, \Delta, L)\):
  - \(\Pi\) is a set of \(n\) players
  - \(S\) is a (finite) set of states
  - \(\langle S_i \rangle_{i \in \Pi}\) is a partition of \(S\)
  - \(A\) is a set of action labels
  - \(\Delta : S \times A \to \text{Dist}(S)\) is a (partial) transition probability function
  - \(L : S \to 2^{\text{AP}}\) is a labelling with atomic propositions from \(\text{AP}\)

- **NB tool does not support concurrent games**
Rewards

- Annotate SMGs with rewards (or costs)
  - real-valued quantities assigned to states and/or transitions

- Wide range of possible uses:
  - elapsed time, power consumption, number of messages successfully delivered, net profit, ...

- We work with:
  - state rewards: $r : S \rightarrow \mathbb{R}_{\geq 0}$
  - action rewards: $r : A \rightarrow \mathbb{R}_{\geq 0}$

- Form basis for a variety of quantitative objectives
  - expected cumulative (total) reward (denoted $C$)
  - mean-payoff (limit-average) reward (denoted $S$)
  - ratio reward
  - (and many more not considered here)
Paths, strategies + probabilities

• **A path is an (infinite) sequence of connected states in SMG**
  – i.e. \( s_0a_0s_1a_1 \ldots \) such that \( a_i \in A(s_i) \) and \( \Delta(s_i,a_i)(s_{i+1}) > 0 \) for all \( i \)
  – represents a system execution (i.e. one possible behaviour)
  – to reason formally, need a probability space over paths

• **A strategy for player \( i \in \Pi \) resolves choices in \( S_i \) states**
  – based on history of execution so far
  – i.e. a function \( \sigma_i : (SA)^*S_i \rightarrow \text{Dist}(A) \)
  – \( \Sigma_i \) denotes the set of all strategies for player \( i \)
  – deterministic if \( \sigma_i \) always gives a Dirac distribution
  – memoryless if \( \sigma_i(s_0a_0\ldots s_k) \) depends only on \( s_k \)
  – also finite-memory, infinite memory, ...
  – history based or explicit memory representation

• **A strategy profile is tuple** \( \sigma = (\sigma_1, \ldots, \sigma_n) \)
  – combining strategies for all \( n \) players
• **For a strategy profile** $\sigma$:
  - the game’s behaviour is fully probabilistic
  - essentially an (infinite-state) Markov chain
  - yields a probability measure $Pr_s^\sigma$
    over set of all paths $\text{Path}_s$ from $s$

• **Allows us to reason about the probability of events**
  - under a specific strategy profile $\sigma$
  - e.g. any (\(\omega\)-)regular property over states/actions

• **Also allows us to define expectation of random variables**
  - i.e. measurable functions $X : \text{Path}_s \rightarrow \mathbb{R}_{\geq 0}$
  - $E_s^\sigma[X] = \int_{\text{Path}_s} X \ dPr_s^\sigma$
  - used to define expected costs/rewards...
Property specification: rPATL

- Temporal logic rPATL:
  - reward probabilistic alternating temporal logic
- CTL, extended with:
  - coalition operator $\langle\langle C \rangle\rangle$ of ATL (Alternating Temporal Logic)
  - probabilistic operator $P$ of PCTL, where $P_{\bowtie q}[\psi]$ means “the probability of ensuring $\psi$ satisfies $\bowtie q$”
  - reward operator $R$ of PRISM, where $R_{\bowtie q} [\rho]$ means “the expected value of $\rho$ satisfies $\bowtie q$”
- Example:
  - $\langle\langle\{1,2\}\rangle\rangle P_{<0.01} [ F^{\leq 10} \text{error} ]$
  - “players 1 and 2 have a strategy to ensure that the probability of an error occurring within 10 steps is less than 0.1, regardless of the strategies of other players”
rPATL properties

• Syntax:

\[ \phi ::= \langle\langle C \rangle\rangle P_{\bowtie q}[\psi] \mid \langle\langle C \rangle\rangle R_{\bowtie q}[\rho] \mid \langle\langle C \rangle\rangle R^{r/c}_{\bowtie q}[\rho] \]

\[ \psi ::= F \ a \]

\[ \rho ::= C \mid S \]

• where:

- \( a \in AP \) is an atomic proposition, \( C \subseteq \Pi \) is a coalition of players,
- \( \bowtie \in \{\leq, <, >, \geq\} \), \( q \in \mathbb{R}_{\geq 0} \), \( r \) and \( c \) are reward structures

• \( \langle\langle C \rangle\rangle P_{\geq 1}[F \ “end”] \)

- “players in coalition \( C \) have a collective strategy to ensure that the game reaches an “end”–state almost surely, regardless of the strategies of other players”
rPATL reward properties

• Syntax:

\[ \phi ::= \langle\langle C\rangle\rangle P_{\bowtie q}[\psi] \mid \langle\langle C\rangle\rangle R_{\bowtie q}[\rho] \mid \langle\langle C\rangle\rangle R_{r/c\bowtie q}[\rho] \]

\[ \psi ::= F \ a \quad \text{“reachability”} \]

\[ \rho ::= C \mid S \quad \text{“longrun average”} \]

\[ \langle\langle C\rangle\rangle R_{\text{fuel} < q}[C] \quad \text{“cumulative”} \]

– “players in coalition C have a strategy to ensure that the expected total fuel consumption is less than q, regardless of the strategies of other players”

\[ \langle\langle C\rangle\rangle R_{\text{fuel/time} \leq q}[S] \]

– “players in coalition C have a strategy to ensure that the expected longrun fuel consumption per time unit is at most q, regardless of the strategies of other players”
rPATL semantics

- Semantics for most operators is standard
- Just focus on P and R operators…
  - use reduction to a stochastic 2–player game

- Coalition game $G_C$ for SMG $G$ and coalition $C \subseteq \Pi$
  - 2–player SMG where $C$ and $\Pi \setminus C$ collapse to players 1 and 2

- $\langle \langle C \rangle \rangle P_{\bowtie q}[\psi]$ is true in state $s$ of $G$ iff:
  - in coalition game $G_C$:
    - $\exists \sigma_1 \in \Sigma_1$ such that $\forall \sigma_2 \in \Sigma_2 . \Pr_s^{\sigma_1,\sigma_2}(\psi) \bowtie q$

- Semantics for R operator defined similarly…
Examples

\[ \langle \langle \circ \rangle \rangle P_{\geq \frac{1}{4}}[ F \checkmark ] \]

true in initial state

\[ \langle \langle \circ \rangle \rangle P_{\geq \frac{1}{3}}[ F \checkmark ] \]

\[ \langle \langle \circ, \circ \rangle \rangle P_{\geq \frac{1}{3}}[ F \checkmark ] \]
Examples

$\langle\langle\bigcirc\rangle\rangle P_{\geq \frac{1}{4}}[F \checkmark]$  
true in initial state

$\langle\langle\bigcirc\rangle\rangle P_{\geq \frac{1}{3}}[F \checkmark]$  
false in initial state

$\langle\langle\bigcirc,\bigcirc\rangle\rangle P_{\geq \frac{1}{3}}[F \checkmark]$
Examples

\[ \langle \langle \circ \rangle \rangle P \geq {\frac{1}{4}} [ F \checkmark ] \]
true in initial state

\[ \langle \langle \circ \rangle \rangle P \geq {\frac{1}{3}} [ F \checkmark ] \]
false in initial state

\[ \langle \langle \circ, \circ \rangle \rangle P \geq {\frac{1}{3}} [ F \checkmark ] \]
true in initial state
Verification and strategy synthesis

- The verification problem is:
  - Given a game $G$ and rPATL property $\phi$, does $G$ satisfy $\phi$?
- e.g. $\langle\langle C \rangle\rangle P_{\bowtie q}[\psi]$ is true in state $s$ of $G$ iff:
  - in coalition game $G_C$:
  - $\exists \sigma_1 \in \Sigma_1$ such that $\forall \sigma_2 \in \Sigma_2$. $Pr_s^{\sigma_1,\sigma_2}(\psi) \bowtie q$

- The synthesis problem is:
  - Given a game $G$ and a coalition property $\phi$, find, if it exists, a coalition strategy $\sigma$ that is a witness to $G$ satisfying $\phi$

- Reduce to computing optimal values and winning strategies in 2–player games
  - e.g. $\langle\langle C \rangle\rangle P_{\ge q}[\psi] \iff \sup_{\sigma_1 \in \Sigma_1} \inf_{\sigma_2 \in \Sigma_2} Pr_s^{\sigma_1,\sigma_2}(\psi) \ge q$
  - complexity $NP \cap coNP$ (this fragment), cf $P$ for MDPs
Verification and strategy synthesis

• The verification problem is:
  – Given a game $G$ and rPATL property $\phi$, does $G$ satisfy $\phi$?

• The synthesis problem is:
  – Given a game $G$ and a coalition property $\phi$, find, if it exists, a coalition strategy $\sigma$ that is a witness to $G$ satisfying $\phi$.

• Reduce to computing optimal values and winning strategies in 2–player games
  – typically employ value iteration to specified convergence
  – both players have optimal strategies
  – memoryless deterministic strategies suffice
  – (epsilon–optimal) strategies can be typically extracted from optimal values in linear time
Multi-objective properties

- May need to explore trade-offs
  - e.g. between performance and resource usage: maximise probability of success \textit{and} minimise energy usage
- Consider \textit{conjunctions} of objectives (for stopping games), also known as \textit{multidimensional}
  - expected total rewards, mean-payoffs or ratios
  - almost sure mean-payoffs/ratios
- Example
  - “the expected longrun average fuel consumption \textit{and} profit are simultaneously at least $v_1$ and $v_2$, respectively”
    \[
    \langle\langle C \rangle\rangle ( R_{fuel} \geq v_1 [S] \& R_{profit} \geq v_2 [S] ) \]
- NB Boolean combinations may be needed for implication
  \[
  \langle\langle C \rangle\rangle ( R_{fuel/time} \geq v_1 [S] \Rightarrow R_{profit} \geq v_2 [S] )
  \]
• Consider the simpler scenario of MDPs (1½ player games)
• Pareto optimum for conjunction of two objectives
  – probability of reaching D is greater than 0.2 and
  – probability of reaching E is greater than 0.6
• Randomised strategies may be needed…
Multi-objective properties

• For MDPs, optimal strategies exist but randomised strategies may be needed
• For stochastic games:
  – optimal strategies may not exist
  – infinite memory may be required
• Therefore
  – work with restricted games (e.g. stopping)
  – use stochastic memory update representation [Brazdil et al, 2014]
    • exponentially more succinct than deterministic update
    • equivalent power if infinite memory allowed
• Decision procedure
  – complexity is NP ∩ coNP
  – compute epsilon-approximations of Pareto sets and epsilon-optimal strategies, fixed point reached in finitely many steps
Multidimensional Pareto set

Pareto set approximation for a mixed multi-objective property
Computation of Pareto sets

- Multi-objective strategy synthesis
  - value iteration over polytopic sets
  - maintains a vector of such sets for each state, one for each dimension
- Pareto sets
  - optimal achievable trade-offs between objectives
- Visualisation of high-dimensional Pareto sets
  - projection
  - slicing
Compositional strategy synthesis

• Componentised games
  – improve scalability of analysis, avoid product state space
  – devise a composition operator for SMGs

• Assume–guarantee synthesis:
  – need a strategy for the full system satisfying a global property
  – synthesise one strategy per component, for local properties
  – use assume–guarantee rules to compose local strategies

• Example: local strategies for $G_1 \models \phi^A$ and $G_2 \models \phi^A \Rightarrow \phi^B$
  compose to a global strategy for $G_1 \parallel G_2 \models \phi^B$

• For any player 1 strategy, each game $G_i$ becomes MDP $M_i$
  – can leverage matching compositional assume–guarantee rules
    for MDPs, e.g. [Etessami et al 2007][Kwiatkowska et al, 2013]
Compositional strategy synthesis

- **Extension of rPATL: Boolean combinations of objectives**
  - expected total rewards (for stopping games)
  - expected mean-payoffs or ratios (controllable multi-chain)
  - conjunctions of almost sure mean-payoffs/ratios (all games)

- **Example**
  - “Player 1 can guarantee that, whenever the expected ratio of longrun average values for \( r_1 \) and \( c \) is at most \( v_1 \), then the ratio for \( r_2 \) and \( c \) is at least \( v_2 \)
  - \( \langle \langle 1 \rangle \rangle ( R\{r_1/\ c\} \leq v_1 [ S ] \Rightarrow R\{r_2/\ c\} \geq v_2 [ S ] ) \)

- **Employ strategy synthesis on component games:**
  - multi-objective properties to use in local and global properties
  - admit also longrun properties (e.g. ratios of rewards)
  - need to consider **fairness** requirements
Compositional strategy synthesis

- Based on assume-guarantee contracts over component interfaces
- Synthesise local strategies for components, then compose into a global strategy using assume-guarantee rules
- Under-approximation of Pareto sets

\[
\langle 1 \rangle (R\{r_1/\text{"c"}\} \leq v_1[S] \\
\rightarrow R\{r_2/\text{"c"}\} \geq v_2[S]) \\
\wedge R\{r_3/\text{"c"}\} \leq v_3[S])
\]

\[
\langle 1 \rangle (R\{r_1/\text{"c"}\} \leq v_1[S] \\
\wedge R\{r_2/\text{"c"}\} \geq v_2[S]) \\
\wedge R\{r_3/\text{"c"}\} \leq v_3[S])
\]

\[
\langle 1 \rangle (R\{r_1/\text{"c"}\} \leq v_1[S] \\
\wedge R\{r_2/\text{"c"}\} \geq v_2[S]) \\
\wedge R\{r_3/\text{"c"}\} \leq v_3[S])
\]
Tool support: PRISM-games 2.0

- **Model checker for stochastic games**
  - integrated into PRISM model checker
  - using new explicit-state model checking engine

- **SMGs added to PRISM modelling language**
  - guarded command language, based on reactive modules
  - finite data types, parallel composition, proc. algebra op.s, ...

- **rPATL added to PRISM property specification language**
  - implemented value iteration based model checking

- **Supports strategy synthesis**
  - single and multiple objectives, Pareto curve
  - total expected reward, longrun average, ratio rewards
  - compositional strategy synthesis

- **Available now:**
Case studies

• Evaluated on several case studies:
  – team formation protocol [CLIMA’11]
  – futures market investor model [McIver & Morgan]
  – collective decision making for sensor networks [TACAS’12]
  – energy management in microgrids [TACAS’12]
  – reputation protocol for user-centric networks [SR’13]
  – DNS bandwidth amplification attack [Deshpande et al]
  – self-adaptive software architectures [Camara, Garlan et al]
  – attack–defence scenarios in RFID goods man. [Aslanyan et al]

• Case studies using PRISM-games 2.0 functionality:
  – autonomous urban driving (multi-objective) [QEST’13]
  – UAV path planning with operator (multi-objective) [ICCPS’15]
  – aircraft electric power control (compositional) [TACAS’15]
  – temperature control (compositional) [Wiltsche PhD]

Case study: Energy management

- Energy management protocol for Microgrid
  - Microgrid: local energy management
  - randomised demand management protocol [Hildmann/Saffre'11]
  - probability: randomisation, demand model, ...

- Existing analysis
  - simulation-based
  - assumes all clients are unselfish

- Our analysis
  - stochastic multi-player game
  - clients can cheat (and cooperate)
  - exposes protocol weakness
  - propose/verify simple fix

Automatic Verification of Competitive Stochastic Systems, Chen et al., In *Proc TACAS 2012*
Case study: Autonomous urban driving

• Inspired by DARPA challenge
  – represent map data as a stochastic game, with environment active, able to select hazards
  – express goals as conjunctions of probabilistic and reward properties
  – e.g. “maximise probability of avoiding hazards and minimise time to reach destination”

• Solution (PRISM-games 2.0)
  – synthesise a probabilistic strategy to achieve the multi-objective goal
  – enable the exploration of trade-offs between subgoals
  – applied to synthesise driving strategies for English villages

Synthesis for Multi-Objective Stochastic Games: An Application to Autonomous Urban Driving, Chen et al., In Proc QEST 2013
Case study: UAV path planning

- **Human operator**
  - sensor tasks
  - high-level commands for piloting

- **UAV autonomy**
  - low-level piloting function

- **Quantitative mission objectives**
  - road network surveillance with the *minimal* time, fuel, or restricted operating zone visits

- **Analysis of trade-offs**
  - consider operator fatigue and workload
  - *multi-objective*, MDP and SMG models

Case study: Aircraft power distribution

- Consider Honeywell high-voltage AC (HVAC) subsystem
  - power routed from generators to buses through switches
  - represent as a stochastic game, modelling competition for buses, with stochasticity used to model failures
  - specify control objectives in LTL using longrun average
  - e.g. “maximise uptime of the buses and minimise failure rate”

- Solution (PRISM-games 2.0)
  - compositional strategy synthesis
  - enable the exploration of trade-offs between uptime of buses and failure rate

Compositional Controller Synthesis for Stochastic Games, Basset et al., In Proc CONCUR 2014
Resources & Information

• **Website**: www.prismmodelchecker.org
  – downloads, manual, tutorials, case studies, papers, ...

• **PRISM benchmark suite**
  – www.prismmodelchecker.org/benchmarks/

• **GitHub**: github.com/prismmodelchecker/
  – source code, bug tracking, feature requests, test suite

• **Coming soon**:  
  – probabilistic counterexample generation  
  – partially observable MDPs and PTAs [FORMATS’15]  
  – enhanced strategy synthesis and analysis  
  – ....
Conclusion

- Introduction to probabilistic model checking
- Overview of PRISM
- New developments
  1. real-time: probabilistic timed automata (PTAs)
  2. parametric model checking
  3. multi–objective model checking
  4. games: stochastic multi–player games (SMGs)
- Related/future work
  - quantitative runtime verification [TSE’11, CACM’12]
  - statistical model checking [TACAS’04, STTT’06]
  - multi–objective stochastic games [MFCS’13, CONCUR’14, TACAS’15]
  - verification of cardiac pacemakers [HSCC’13, HSB’15, HSCC’16]
  - probabilistic hybrid automata [CPSWeek’13 tutorial]
References

• **Tutorial papers**

• **PRISM tool paper**
Acknowledgements

• My group and collaborators in this work
• Project funding
  – ERC Advanced Grant
  – EPSRC Mobile Autonomy Programme Grant
  – Oxford Martin School, Institute for the Future of Computing

• See also
  – VERIWARE www.veriware.org
  – PRISM www.prismmodelchecker.org