Probabilistic model checking with PRISM

Marta Kwiatkowska
Department of Computer Science, University of Oxford

IMT, Lucca, May 2016
Lecture plan

- Course slides and lab session
  - http://www.prismmodelchecker.org/courses/imt16/

- 3 sessions: lectures 9–11
  - 1 – Discrete time Markov chains (DTMCs)
  - 2 – Markov decision processes (MDPs)
  - 3 – LTL model checking for DTMCs/MDPs

- For extended versions of this material
  - and an accompanying list of references
  - see: http://www.prismmodelchecker.org/lectures/
## Probabilistic models

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

---

The table above categorizes probabilistic models based on whether they are fully probabilistic or nondeterministic, and whether they are discrete-time or continuous-time. Discrete-time models include Markov chains, while continuous-time models include continuous-time Markov chains and Markov decision processes.
Part 3

LTL Model Checking
Overview (Part 3)

- Linear temporal logic (LTL)
- Strongly connected components
- \(\omega\)-automata (Büchi, Rabin)
- LTL model checking for DTMCs
- LTL model checking for MDPs
- New developments and beyond PRISM
Limitations of PCTL

- PCTL, although useful in practice, has limited expressivity
  - essentially: probability of reaching states in X, passing only through states in Y (and within k time-steps)

- One useful approach: extend models with costs/rewards
  - see slides for the last two lectures

- Another direction: Use more expressive logics. e.g.:
  - LTL [Pnu77] – (non-probabilistic) linear-time temporal logic
  - PCTL* [ASB+95,BdA95] – which subsumes both PCTL and LTL
  - both allow path operators to be combined
  - (in PCTL, $P \neg P [...]$ always contains a single temporal operator)
LTL – Linear temporal logic

• LTL syntax (path formulae only)
  \[ \psi ::= \text{true} \mid a \mid \psi \land \psi \mid \neg \psi \mid X \psi \mid \psi U \psi \]
  – where \( a \in AP \) is an atomic proposition
  – usual equivalences hold: \( F \phi \equiv \text{true} U \phi \), \( G \phi \equiv \neg (F \neg \phi) \)

• LTL semantics (for a path \( \omega \))
  \[ \omega \models \text{true} \quad \text{always} \]
  \[ \omega \models a \iff a \in L(\omega(0)) \]
  \[ \omega \models \psi_1 \land \psi_2 \iff \omega \models \psi_1 \text{ and } \omega \models \psi_2 \]
  \[ \omega \models \neg \psi \iff \omega \not\models \psi \]
  \[ \omega \models X \psi \iff \omega[1\ldots] \models \psi \]
  \[ \omega \models \psi_1 U \psi_2 \iff \exists k \geq 0 \text{ s.t. } \omega[k\ldots] \models \psi_2 \land \forall i < k \omega[i\ldots] \models \psi_1 \]

where \( \omega(i) \) is \( i \)th state of \( \omega \), and \( \omega[i\ldots] \) is suffix starting at \( \omega(i) \)
LTL examples

• \((F \text{ tmp}_{\text{fail}}_1) \land (F \text{ tmp}_{\text{fail}}_2)\)
  – “both servers suffer temporary failures at some point”

• GF ready
  – “the server always eventually returns to a ready-state”

• FG error
  – “an irrecoverable error occurs”

• \(G (\text{req} \rightarrow X \text{ack})\)
  – “requests are always immediately acknowledged”
LTL for DTMCs

• Same idea as PCTL: probabilities of sets of path formulae
  – for a state $s$ of a DTMC and an LTL formula $\psi$:
  – $\text{Prob}(s, \psi) = \Pr_s \{ \omega \in \text{Path}(s) \mid \omega \models \psi \}$
  – all such path sets are measurable [Var85]

• A (probabilistic) LTL specification often comprises an LTL (path) formula and a probability bound
  – e.g. $P_{\geq 1} [ GF \text{ ready } ]$ – “with probability 1, the server always eventually returns to a ready-state”
  – e.g. $P_{\leq 0.01} [ FG \text{ error } ]$ – “with probability at most 0.01, an irrecoverable error occurs”

• PCTL* subsumes both LTL and PCTL
  – e.g. $P_{>0.5} [ GF \text{ crit}_1 ] \land P_{>0.5} [ GF \text{ crit}_2 ]$
Long-run behaviour of DTMCs

\[ k=0: \]

\[ k=1: \]

\[ k=2: \]

\[ k=3: \]
Strongly connected components

- Long-run properties of DTMCs rely on an analysis of their underlying graph structure (i.e. ignoring probabilities)

- **Strongly connected** set of states $T$
  - for any pair of states $s$ and $s'$ in $T$, there is a path from $s$ to $s'$, passing only through states in $T$

- **Strongly connected component (SCC)**
  - a maximally strongly connected set of states (i.e. no superset of it is also strongly connected)

- **Bottom strongly connected component (BSCC)**
  - an SCC $T$ from which no state outside $T$ is reachable from $T$
Example – (B)SCCs

SCC

BSCC

BSCC
Fundamental property of (finite) DTMCs...

With probability 1, some BSCC will be reached and all of its states visited infinitely often.

Formally:
\[ \Pr_{s_0}(s_0s_1s_2 \ldots | \exists i \geq 0, \exists \text{BSCC } T \text{ such that } \forall j \geq i \ s_j \in T \text{ and } \forall s \in T \ s_k = s \text{ for infinitely many } k ) = 1 \]
LTL model checking for DTMCs

- LTL model checking for DTMCs relies on:
  - computing the probability \( \text{Prob}(s, \psi) \) for LTL formula \( \psi \)
  - reduces to probability of reaching a set of “accepting” BSCCs
  - 2 simple cases: \( \text{GF } a \) and \( \text{FG } a \)

- \( \text{Prob}(s, \text{GF } a) = \text{Prob}(s, F T_{GFa}) \)
  - where \( T_{GFa} \) = union of all BSCCs containing some state satisfying \( a \)

- \( \text{Prob}(s, \text{FG } a) = \text{Prob}(s, F T_{FGa}) \)
  - where \( T_{FGa} \) = union of all BSCCs containing only \( a \)-states

- To extend this idea to arbitrary LTL formula, we use \( \omega \)-automata...

Example:

\[
\text{Prob}(s_0, \text{GF } a) = \text{Prob}(s_0, F T_{GFa}) = \text{Prob}(s_0, F \{s_3, s_2, s_5\}) = 2/3 + 1/6 = 5/6
\]
Overview (Part 3)

- Linear temporal logic (LTL)
- Strongly connected components
- $\omega$–automata (Büchi, Rabin)
- LTL model checking for DTMCs
- LTL model checking for MDPs
- New developments and beyond PRISM
Reminder – Finite automata

• A regular language over alphabet $\Sigma$
  – is a set of finite words $L \subseteq \Sigma^*$ such that either:
    • $L = L(E)$ for some regular expression $E$
    • $L = L(A)$ for some nondeterministic finite automaton (NFA) $A$
    • $L = L(A)$ for some deterministic finite automaton (DFA) $A$

• Example:

Regexp: $(\alpha+\beta)^*\beta(\alpha+\beta)$

NFA $A$:

• NFAs and DFAs have the same expressive power
  – we can always determinise an NFA to an equivalent DFA
  – (with a possibly exponential blow-up in size)
Büchi automata

• \( \omega \)-automata represent sets of infinite words \( L \subseteq \Sigma^\omega \)
  – e.g. Büchi automata, Rabin automata, Streett, Muller, ...

• A nondeterministic Büchi automaton (NBA) is...
  – a tuple \( A = (Q, \Sigma, \delta, Q_0, F) \) where:
    – \( Q \) is a finite set of states
    – \( \Sigma \) is an alphabet
    – \( \delta : Q \times \Sigma \rightarrow 2^Q \) is a transition function
    – \( Q_0 \subseteq Q \) is a set of initial states
    – \( F \subseteq Q \) is a set of “accept” states

• NBA acceptance condition
  – language \( L(A) \) for \( A \) contains \( w \in \Sigma^\omega \) if there is a corresponding run in \( A \) that passes through states in \( F \) infinitely often

Example:
words \( w \in \{\alpha, \beta\}^\omega \)
with infinitely many \( \alpha \)
ω-regular properties

• Consider a model, i.e. an LTS/DTMC/MDP/…
  – for example: DTMC $D = (S, s_{\text{init}}, P, \text{Lab})$
  – where labelling $\text{Lab}$ uses atomic propositions from set $\text{AP}$

• We can capture properties of these using ω-automata
  – let $\omega \in \text{Path}(s)$ be some infinite path in $D$
  – $\text{trace}(\omega) \in (2^{\text{AP}})^{\omega}$ denotes the projection of state labels of $\omega$
  – i.e. $\text{trace}(s_0s_1s_2s_3...) = \text{Lab}(s_0)\text{Lab}(s_1)\text{Lab}(s_2)\text{Lab}(s_3)…$
  – can specify a set of paths of $D$ with an ω-automaton over $2^{\text{AP}}$

• Let $\text{Prob}^D(s, A)$ denote the probability…
  – from state $s$ in a discrete-time Markov chain $D$
  – of satisfying the property specified by automaton $A$
  – i.e. $\text{Prob}^D(s, A) = \Pr^D_s\{ \omega \in \text{Path}(s) \mid \text{trace}(\omega) \in L(A) \}$
- **Nondeterministic Büchi automaton**
  - for LTL formula $FG\ a$, i.e. “eventually always $a$”
  - for a DTMC with atomic propositions $AP = \{a,b\}$

- We abbreviate this to just:
Büchi automata + LTL

• **Nondeterministic Büchi automata (NBAs)**
  – define the set of \( \omega \)-regular languages

• **\( \omega \)-regular languages are more expressive than LTL**
  – can convert any LTL formula \( \psi \) over atomic propositions \( \text{AP} \)
  – into an equivalent NBA \( A_\psi \) over \( 2^{\text{AP}} \)
  – i.e. \( \omega \models \psi \Leftrightarrow \text{trace}(\omega) \in L(A_\psi) \) for any path \( \omega \)
  – for LTL-to-NBA translation, see e.g. [VW94], [DGV99], [BK08]
  – worst-case: exponential blow-up from \(|\psi|\) to \(|A_\psi|\)

• **But deterministic Büchi automata (DBAs) are less expressive**
  – e.g. there is no DBA for the LTL formula \( \text{FG} \ a \)
  – for probabilistic model checking, need deterministic automata
  – so we use deterministic Rabin automata (DRAs)
Deterministic Rabin automata

• A deterministic Rabin automaton is a tuple $(Q, \Sigma, \delta, q_0, \text{Acc})$:
  – $Q$ is a finite set of states, $q_0 \in Q$ is an initial state
  – $\Sigma$ is an alphabet, $\delta : Q \times \Sigma \rightarrow Q$ is a transition function
  – $\text{Acc} = \{(L_i, K_i)\}_{i=1..k} \subseteq 2^Q \times 2^Q$ is an acceptance condition

• A run of a word on a DRA is accepting iff:
  – for some pair $(L_i, K_i)$, the states in $L_i$ are visited finitely often and (some of) the states in $K_i$ are visited infinitely often

  – or in LTL: $\bigvee_{1 \leq i \leq k} (FG \neg L_i \land GF K_i )$

• Example: DRA for $FG a$
  – acceptance condition is $\text{Acc} = \{ (\{q_0\}, \{q_1\}) \}$
LTL model checking for DTMCs

- LTL model checking for DTMC D and LTL formula $\psi$

  1. Construct DRA $A_\psi$ for $\psi$

  2. Construct product $D \otimes A$ of DTMC D and DRA $A_\psi$

  3. Compute $\text{Prob}^D(s, \psi)$ from DTMC $D \otimes A$

- Running example:
  - compute probability of satisfying LTL formula $\psi = G\neg b \land GF a$ on:

```
  s_0 -> s_1: 0.1
  s_0 -> s_3: 0.2
  s_1 -> s_2: 0.5
  s_1 -> s_4: 0.3
  s_2 -> s_5: 0.1
  s_3 -> s_0: 0.6
  s_3 -> s_4: 0.3
  s_4 -> s_5: 0.9
  s_4: 1
  s_5: 1
```

  - $\{a\}$
  - $\{b\}$
Example – DRA

- DRA $A_{\psi}$ for $\psi = G\neg b \land GF a$
  - acceptance condition is $Acc = \{ (\emptyset, \{q_1\}) \}$
  - (i.e. this is actually a deterministic Büchi automaton)

If $G\neg b$ violated (because we see a $b$), end up stuck here

Need to visit here infinitely often to satisfy $GF a$
Product DTMC for a DRA

- We construct the product DTMC
  - for DTMC $D$ and DRA $A$, denoted $D \otimes A$
  - $D \otimes A$ can be seen as an unfolding of $D$ with states $(s, q)$, where $q$ records state of automaton $A$ for path fragment so far
  - since $A$ is deterministic, $D \otimes A$ is also a DTMC
  - each path in $D$ has a corresponding (unique) path in $D \otimes A$
  - the probabilities of paths in $D$ are preserved in $D \otimes A$

- Formally, for $D = (S, s_{init}, P, L)$ and $A = (Q, \Sigma, \delta, q_0, \{(L_i, K_i)\}_{i=1..k})$
  - $D \otimes A$ is the DTMC $(S \times Q, (s_{init}, q_{init}), P', L')$ where:
    - $q_{init} = \delta(q_0, L(s_{init}))$
    - $P'((s_1, q_1), (s_2, q_2)) = \begin{cases} P(s_1, s_2) & \text{if } q_2 = \delta(q_1, L(s_2)) \\ 0 & \text{otherwise} \end{cases}$
    - $l_i \in L'(s, q)$ if $q \in L_i$ and $k_i \in L'(s, q)$ if $q \in K_i$
Example – Product DTMC

DTMC $D$

![Diagram of DTMC D](image)

Product DTMC $D \otimes A_\psi$

- $s_0$ is initial state of DTMC $D$
- $s_0$ satisfies neither $a$ nor $b$ so we stay in $q_0$ in DRA $A_\psi$

DRA $A_\psi$ for $\psi = G\neg b \land GF a$

![Diagram of DRA A_\psi](image)

$\text{Acc} = \{ \{\}, \{q_1\} \}$
Example – Product DTMC

DTMC $D$

- States: $s_0, s_1, s_2, s_3, s_4, s_5$
- Transitions:
  - $s_0$ to $s_1$: $0.6$, $s_0$ to $s_3$: $0.3$
  - $s_1$ to $s_0$: $0.1$, $s_1$ to $s_2$: $0.2$
  - $s_2$ to $s_1$: $0.3$, $s_2$ to $s_4$: $0.1$
  - $s_3$ to $s_1$: $1$, $s_3$ to $s_2$: $0.3$
  - $s_4$ to $s_5$: $1$

DRA $A_\psi$ for $\psi = G\neg b \land GF a$

- States: $q_0, q_1, q_2$
- Transitions:
  - $q_0$ to $q_0$: $a \land \neg b$
  - $q_0$ to $q_2$: $\neg a \land \neg b$
  - $q_2$ to $q_1$: $b$
  - $q_2$ to $true$

Product DTMC $D \otimes A_\psi$

- States: $s_0 q_0, s_1 q_2, s_3 q_1$
- Transitions:
  - $s_0 q_0$ to $s_1 q_2$: $0.1$
  - $s_3 q_1$ to $s_0 q_0$: $0.6$
  - $s_3 q_1$ to $s_1 q_2$: $0.3$

- $s_1$ satisfies $b$ so we move to $q_2$ in $A_\psi$
- $s_3$ satisfies $a$ but not $b$ so we move to $q_1$ in $A_\psi$

Acc $= \{ \{\}, \{q_1\} \}$
Example – Product DTMC

DTMC \( D \) for \( \psi = G \neg b \land GF a \):

Product DTMC \( D \otimes A_\psi \):

- 2 copies of \( s_3/s_4 \), one after seeing a b and one no b’s.
- Label states satisfying acceptance pair \((L_1,K_1)\).
For DTMC D and DRA A

\[
\text{Prob}^D(s, A) = \text{Prob}^{D \otimes A}((s, q_s), \bigvee_{1 \leq i \leq k} (FG \neg l_i \land GF k_i))
\]

- where \(q_s = \delta(q_0, L(s))\)

Hence:

\[
\text{Prob}^D(s, A) = \text{Prob}^{D \otimes A}((s, q_s), F T_{\text{Acc}})
\]

- where \(T_{\text{Acc}}\) is the union of all accepting BSCCs in \(D \otimes A\)
- an accepting BSCC \(T\) of \(D \otimes A\) is such that, for some \(1 \leq i \leq k\), no states in \(T\) satisfy \(l_i\) and some state in \(T\) satisfies \(k_i\)

- Reduces to computing BSCCs and reachability probabilities
Example: LTL for DTMCs

- Compute $\text{Prob}(s_0, G \neg b \land GF a)$ for DTMC $D$:

DTMC $D$

DRA $A_\psi$ for $\psi = G \neg b \land GF a$

Acc = $\{(\{} , \{q_1\}\} , \}$
Example: LTL for DTMCs

DTMC $D$

DRA $A_\psi$ for $\psi = G\neg b \land GF a$

Product DTMC $D \otimes A_\psi$
Example: LTL for DTMCs

DTMC $D$  

DRA $A_\psi$ for $\psi = G \neg b \land GF a$

Product DTMC $D \otimes A_\psi$

$\text{Prob}^D(s_0, \psi) = \text{Prob}^{D\otimes A_\psi}(s_0 q_0, F T_1) = \frac{3}{4}$
Complexity of LTL model checking

- Complexity of model checking LTL formula $\psi$ on DTMC $D$
  - is doubly exponential in $|\psi|$ and polynomial in $|D|$ (for the algorithm presented in these lectures)

- Double exponential blow-up comes from use of DRAs
  - size of NBA can be exponential in $|\psi|$;
  - and DRA can be exponentially bigger than NBA;
  - in practice, this does not occur and $\psi$ is small anyway

- Polynomial–time operations required on product model
  - BSCC computation – linear in (product) model size
  - probabilistic reachability – cubic in (product) model size

- In total: $O(\text{poly}(|D|,|A_\psi|))$

- Complexity can be reduced to single exponential in $|\psi|$
  - see e.g. [CY88,CY95]
PCTL* model checking

- **PCTL* syntax:**
  - $\phi ::= \text{true} \mid a \mid \phi \land \phi \mid \neg \phi \mid P_{\neg p} [ \psi ]$
  - $\psi ::= \phi \mid \psi \land \psi \mid \neg \psi \mid X \psi \mid \psi U \psi$

- **Example:**
  - $P_{>p} [ GF (send \rightarrow P_{>0} [ F \text{ack} ] ) ]$

- **PCTL* model checking algorithm**
  - bottom-up traversal of parse tree for formula (like PCTL)
  - to model check $P_{\neg p} [ \psi ]$:
    - replace maximal state subformulae with atomic propositions
    - (state subformulae already model checked recursively)
    - modified formula $\psi$ is now an LTL formula
    - which can be model checked as for LTL
Overview (Part 3)

- Linear temporal logic (LTL)
- Strongly connected components
- $\omega$-automata (Büchi, Rabin)
- LTL model checking for DTMCs
- LTL model checking for MDPs
- New developments and beyond PRISM
End components in MDPs

• End components of MDPs are the analogue of BSCCs in DTMCs

• An end component is a strongly connected sub-MDP

• A sub-MDP comprises a subset of states and a subset of the actions/distributions available in those states, which is closed under probabilistic branching

Note:

- action labels omitted
- probabilities omitted where = 1
End components of MDPs are the analogue of BSCCs in DTMCs.

For every end component, there is an adversary which, with probability 1, forces the MDP to remain in the end component, and visit all its states infinitely often.

Under every adversary $\sigma$, with probability 1 some end component will be reached and all of its states visited infinitely often (union of ECs reached with prob 1).
Long-run properties of MDPs

- **Maximum probabilities**
  - $p_{\text{max}}(s, GF \ a) = p_{\text{max}}(s, F T_{GFa})$
    - where $T_{GFa}$ is the union of sets $T$ for all end components $(T, \text{Steps'})$ with $T \cap \text{Sat}(a) \neq \emptyset$

  - $p_{\text{max}}(s, FG \ a) = p_{\text{max}}(s, F T_{FGa})$
    - where $T_{FGa}$ is the union of sets $T$ for all end components $(T, \text{Steps'})$ with $T \subseteq \text{Sat}(a)$

- **Minimum probabilities**
  - need to compute from maximum probabilities...
  - $p_{\text{min}}(s, GF \ a) = 1 - p_{\text{max}}(s, FG \neg a)$
  - $p_{\text{min}}(s, FG \ a) = 1 - p_{\text{max}}(s, GF \neg a)$
Example

- **Model check:** \( P_{<0.8} \ [\ GF \ b ] \) for \( s_0 \)

- **Compute** \( p_{\text{max}}(GF \ b) \)
  - \( p_{\text{max}}(GF \ b) = p_{\text{max}}(s, F T_{\text{GFb}}) \)
  - \( T_{\text{GFb}} \) is the union of sets \( T \) for all end components with \( T \cap \text{Sat}(b) \neq \emptyset \)
  - \( \text{Sat}(b) = \{ s_4, s_6 \} \)
  - \( T_{\text{GFb}} = T_1 \cup T_2 \cup T_3 = \{ s_1, s_3, s_4, s_6 \} \)
  - \( p_{\text{max}}(s, F T_{\text{GFb}}) = 0.75 \)
  - \( p_{\text{max}}(GF \ b) = 0.75 \)

- **Result:** \( s_0 \models P_{<0.8} \ [\ GF \ b ] \)
Automata-based properties for MDPs

• For an MDP $M$ and automaton $A$ over alphabet $2^{AP}$
  – consider probability of “satisfying” language $L(A) \subseteq (2^{AP})^\omega$
  – $\text{Prob}^M,_{adv}(s, P) = \text{Pr}_s^M,_{adv}\{ \omega \in \text{Path}^M,_{adv}(s) \mid \text{trace}(\omega) \in L(A) \}$
  – $p_{\max}^M(s, A) = \sup_{adv \in \text{Adv}} \text{Prob}^M,_{adv}(s, A)$
  – $p_{\min}^M(s, A) = \inf_{adv \in \text{Adv}} \text{Prob}^M,_{adv}(s, A)$

• Might need minimum or maximum probabilities
  – e.g. $s \models P_{\geq 0.99}[\psi_{\text{good}}] \iff p_{\min}^M(s, \psi_{\text{good}}) \geq 0.99$
  – e.g. $s \models P_{\leq 0.05}[\psi_{\text{bad}}] \iff p_{\max}^M(s, \psi_{\text{bad}}) \leq 0.05$

• But, $\psi$–regular properties are closed under negation
  – as are the automata that represent them
  – so can always consider maximum probabilities…
  – $p_{\max}^M(s, \psi_{\text{bad}})$ or $1 - p_{\max}^M(s, \neg \psi_{\text{good}})$
LTL model checking for MDPs

- Model check LTL specification $P_{\sim p}[\psi]$ against MDP $M$

- 1. Convert problem to one needing maximum probabilities
   - e.g. convert $P_{>p}[\psi]$ to $P_{<1-p}[\neg\psi]$

- 2. Generate a DRA for $\psi$ (or $\neg\psi$)
   - build nondeterministic Büchi automaton (NBA) for $\psi$ [VW94]
   - convert the NBA to a DRA [Saf88]

- 3. Construct product MDP $M \otimes A$
- 4. Identify accepting end components (ECs) of $M \otimes A$
- 5. Compute max. probability of reaching accepting ECs
   - from all states of the $D \otimes A$
- 6. Compare probability for $(s, q_s)$ against $p$ for each $s$
Product MDP for a DRA

• For an MDP \( M = (S, s_{\text{init}}, \text{Steps}, L) \)
  
• and a (total) DRA \( A = (Q, \Sigma, \delta, q_0, \text{Acc}) \)
  
  – where \( \text{Acc} = \{ (L_i, K_i) | 1 \leq i \leq k \} \)

• The product MDP \( M \otimes A \) is:
  
  – the MDP \( (S \times Q, (s_{\text{init}}, q_{\text{init}}), \text{Steps}', L') \) where:
    \( s_{\text{init}} = \delta(q_0, L(s_{\text{init}})) \)
    \( \text{Steps}'(s,q) = \{ \mu^q | \mu \in \text{Step}(s) \} \)
    \[ \mu^q(s', q') = \begin{cases} 
      \mu(s') & \text{if } q' = \delta(q, L(s)) \\
      0 & \text{otherwise}
    \end{cases} \]

  \( l_i \in L'(s,q) \) if \( q \in L_i \) and \( k_i \in L'(s,q) \) if \( q \in K_i \)

(i.e. state sets of acceptance condition used as labels)
Product MDP for a DRA

- For MDP $M$ and DRA $A$

\[
p_{\text{max}}^M(s, A) = p_{\text{max}}^{M \otimes A}((s, q_s), \bigvee_{1 \leq i \leq k} (\text{FG } \neg l_i \land \text{GF } k_i))
\]

- where $q_s = \delta(q_0, L(s))$

- Hence:

\[
p_{\text{max}}^M(s, A) = p_{\text{max}}^{M \otimes A}((s, q_s), \text{F } T_{\text{Acc}})
\]

- where $T_{\text{Acc}}$ is the union of all sets $T$ for accepting end components $(T, \text{Steps'})$ in $D \otimes A$

- an accepting end components is such that, for some $1 \leq i \leq k$:
  - $q \models \neg l_i$ for all $(s, q) \in T$ and $q \models k_i$ for some $(s, q) \in T$
  - i.e. $T \cap (S \times L_i) = \emptyset$ and $T \cap (S \times K_i) \neq \emptyset$
Example: LTL for MDPs

- Model check $P_{<0.8} \left[ G \neg b \land GF a \right]$ for MDP $M$:
  - need to compute $p_{\text{max}}(s_0, G \neg b \land GF a)$

MDP $M$

DRA $A_\psi$ for $\psi = G\neg b \land GF a$

Acc $= \{ (\emptyset, \{q_1\}) \}$
Example: LTL for MDPs

MDP $M$

DRA $A_\psi$ for $\psi = G\neg b \land GF a$

Product MDP $M \otimes A_\psi$

$p_{\text{max}}^M(s_0, \psi) = p_{\text{max}}^{M \otimes A_\psi} (s_0q_0, F T_1) = 0.7$
LTL model checking for MDPs

- **Complexity of model checking LTL formula** $\psi$ **on MDP** $M$
  - is doubly exponential in $|\psi|$ and polynomial in $|M|$.
  - Unlike DTMCs, this cannot be improved upon.

- **PCTL* model checking**
  - LTL model checking can be adapted to PCTL*, as for DTMCs.

- **Maximal end components**
  - Can optimise LTL model checking using maximal end components (there may be exponentially many ECs).

- **Optimal adversaries for LTL formulae**
  - E.g., memoryless adversary always exists for $p_{\text{max}}(s, GF a)$, but not for $p_{\text{max}}(s, FG a)$. 
Summary (LTL model checking)

• Linear temporal logic (LTL)
  – combines path operators; PCTL* subsumes LTL and PCTL
• ω–automata: represent ω–regular languages/properties
  – can translate any LTL formula into a Büchi automaton
  – for deterministic ω–automata, we use Rabin automata
• Long–run properties of DTMCs
  – need bottom strongly connected components (BSCCs)
• LTL model checking for DTMCs
  – construct product of DTMC and Rabin automaton
  – identify accepting BSCCs, compute reachability probability
• LTL model checking for MDPs
  – MDP–DRA product, reachability of accepting end components
PRISM: Recent & new developments

• **New features:**
  1. parametric model checking
  2. parameter synthesis
  3. strategy synthesis
  4. stochastic multi-player games
  5. real-time: probabilistic timed automata (PTAs)

• **Further new additions:**
  – enhanced statistical model checking
    (approximations + confidence intervals, acceptance sampling)
  – efficient CTMC model checking (fast adaptive uniformisation)
  – benchmark suite & testing functionality
  – [www.prismmodelchecker.org](http://www.prismmodelchecker.org)

  – **Beyond PRISM...**
Parametric model checking and synthesis

System

Parametric model
e.g. Markov chain

0.5+x
0.4-x
0.1

Probabilistic model checker
PRISM PARAM

Result

Quantitative results

P < 0.01 [ F ≤ t fail]

Concrete model

System requirements

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

48
1. Parametric model checking 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*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]
2. Parameter synthesis

- Find optimal parameter value given a parametric model and PCTL/CSL property
  - *parametric* probabilities and rates

- Techniques
  - discretisation and integer parameters
  - constraint solving, including parametric symbolic constraints
  - iterative refinement to improve accuracy
  - sampling to improve efficiency
  - *but* scalability is still the biggest challenge

- Implementation
  - using tool combination involving Z3, python, PRISM
  - see also Prophecy from Katoen’s group
3. Controller (strategy) synthesis

- **Can synthesise permissive controllers** [TACAS14]
  - a *permissive* controller allows more than one action per state
  - adds flexibility in case an action become temporarily unavailable, improving robustness
  - e.g. StockPrice Viewer (Android)
  - expressed in terms of multi-strategies

- **Can synthesise controllers using machine learning** [ATVA14]
  - partial exploration of the state space, with guarantees of accuracy
  - combines real-time dynamic programming with value iteration
  - focus on updating “most important parts” = most often visited by good strategies
  - speeds up value iteration

- **Implemented in PRISM for both MDPs and SMGs**
4. Stochastic multi-player games

- **Extension of PRISM**
  - modelling of stochastic multi-player games
  - probabilistic model checking of rPATL and extensions
  - strategy synthesis and analysis
    - optimal strategy generation
    - strategy simulation and export
    - model checking of applied strategies
  - graphical user interface (editors, simulator, graph plotting, ...)
- **PRISM-games 2.0**:
  - long-run average and ratio properties
  - multi-objective strategy synthesis
  - Pareto curve generation and visualisation
  - compositional strategy synthesis techniques
- **Available from** [http://www.prismmodelchecker.org/games/](http://www.prismmodelchecker.org/games/)
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*
5. 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)
  [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

```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

**New for PTAs:**
- clocks
- invariants
- guards/resets

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

• PRISM has two different PTA model checking techniques…

• “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
Beyond PRISM: Cardiac pacemaker

- **Develop model-based framework**
  - *timed automata* model for pacemaker software [Jiang et al]
  - hybrid heart models in *Simulink*, adopt synthetic ECG model (non-linear ODE) [Clifford et al]

- **Properties**
  - (basic safety) maintain 60–100 beats per minute
  - (advanced) detailed analysis *energy usage*, plotted against timing parameters of the pacemaker
  - parameter synthesis: find values for timing delays that optimise energy usage
Optimal timing delays problem

- Optimal timing delay synthesis for timed automata [EMSOFT2014][HSB 2015]

- The parameter synthesis problem solved is
  - given a parametric network of timed I/O automata, set of controllable and uncontrollable parameters, CMTL property $\phi$ and length of path $n$
  - find the optimal controllable parameter values, for any uncontrollable parameter values, with respect to an objective function $O$, such that the property $\phi$ is satisfied on paths of length $n$, if such values exist

- Consider family of objective functions
  - maximise volume, minimise energy

- Discretise parameters, assume bounded integer parameter space and path length
  - decidable but high complexity (high time constants)
Previously, no nondeterminism and no probability in the model considered

Consider parametric probabilistic timed automata (PPTA),
  – e.g. guards of the form $x \leq b$,

Can we synthesise optimal timing parameters to optimise the reachability probability?

Semi-algorithm [RP 2014]
  – exploration of parametric symbolic states, i.e. location, time zone and parameter valuations
  – forward exploration only gives upper bounds on maximum probability (resp. lower for minimum)
  – but stochastic game abstraction yields the precise solution...

Implementation in progress
Quantitative verification – Trends

• Being ‘younger’, generally lags behind conventional verification
  – much smaller model capacity
  – compositional reasoning in infancy
  – automation of model extraction/adaptation very limited

• Tool usage on the increase, in academic/industrial contexts
  – real-time verification/synthesis in embedded systems
  – probabilistic verification in security, reliability, performance

• Shift towards greater automation
  – specification mining, model extraction, synthesis, verification, ...

• But many challenges remain!
Acknowledgements

• My group and collaborators in this work
• Project funding
  – ERC, EPSRC, Microsoft Research
  – Oxford Martin School, Institute for the Future of Computing

• See also
  – VERIWARE www.veriware.org
  – PRISM www.prismmodelchecker.org
You are welcome to visit Oxford!

PhD scholarships, postdocs in verification and synthesis, and more
Thank you for your attention

More info here:
www.prismmodelchecker.org