Probabilistic model checking with PRISM

Marta Kwiatkowska
Department of Computer Science, University of Oxford

4th SSFT, Menlo College, May 2014
What is probabilistic model checking?

- **Probabilistic model checking...**
  - is a **formal verification** technique for modelling and analysing systems that exhibit **probabilistic** behaviour

- **Formal verification...**
  - is the application of rigorous, mathematics-based techniques to establish the correctness of computerised systems
Why formal verification?

- Errors in computerised systems can be costly...

  - Pentium chip (1994)
    Bug found in FPU. Intel (eventually) offers to replace faulty chips. Estimated loss: $475m
  - Infusion pumps (2010)
    Patients die because of incorrect dosage. Cause: software malfunction. 79 recalls.
  - Toyota Prius (2010)
    Software “glitch” found in anti-lock braking system. 185,000 cars recalled.

- Why verify?
  - “Testing can only show the presence of errors, not their absence.” [Edsger Dijkstra]
Model checking

System

Finite-state model

Model checker e.g. SMV, Spin

Temporal logic specification

¬EF fail

Result

Counter-example

System requirements
Probabilistic model checking

System

Probabilistic model
e.g. Markov chain

Result

System requirements

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

Probabilistic model checker
e.g. PRISM

Quantitative results

Counter-example

P < 0.1 [ F fail ]
Why probability?

• Some systems are inherently probabilistic…

• **Randomisation**, e.g. in distributed coordination algorithms
  – as a symmetry breaker, in gossip routing to reduce flooding

• **Examples: real-world protocols featuring randomisation:**
  – Randomised back-off schemes
    • CSMA protocol, 802.11 Wireless LAN
  – Random choice of waiting time
    • IEEE1394 Firewire (root contention), Bluetooth (device discovery)
  – Random choice over a set of possible addresses
    • IPv4 Zeroconf dynamic configuration (link-local addressing)
  – Randomised algorithms for anonymity, contract signing, …
Why probability?

• Some systems are inherently probabilistic…

• **Randomisation**, e.g. in distributed coordination algorithms
  – as a symmetry breaker, in gossip routing to reduce flooding

• **To model uncertainty and performance**
  – to quantify rate of failures, express Quality of Service

• **Examples:**
  – computer networks, embedded systems
  – power management policies
  – nano-scale circuitry: reliability through defect–tolerance
Why probability?

• Some systems are inherently probabilistic…

• Randomisation, e.g. in distributed coordination algorithms
  – as a symmetry breaker, in gossip routing to reduce flooding

• To model uncertainty and performance
  – to quantify rate of failures, express Quality of Service

• To model biological processes
  – reactions occurring between large numbers of molecules are naturally modelled in a stochastic fashion
Verifying probabilistic systems

• We are not just interested in correctness

• We want to be able to quantify:
  – security, privacy, trust, anonymity, fairness
  – safety, reliability, performance, dependability
  – resource usage, e.g. battery life
  – and much more...

• Quantitative, as well as qualitative requirements:
  – how reliable is my car’s Bluetooth network?
  – how efficient is my phone’s power management policy?
  – is my bank’s web–service secure?
  – what is the expected long–run percentage of protein X?
## 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></td>
<td></td>
</tr>
<tr>
<td>Markov chains (DTMCs)</td>
<td></td>
<td>Markov decision processes (MDPs)</td>
</tr>
<tr>
<td><strong>Continuous time</strong></td>
<td></td>
<td>Simple stochastic games (SMGs)</td>
</tr>
<tr>
<td>Markov chains (CTMCs)</td>
<td></td>
<td>Probabilistic timed automata (PTAs)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Interactive Markov chains (IMCs)</td>
</tr>
</tbody>
</table>
### 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><strong>Discrete-time Markov chains (DTMCs)</strong></td>
<td><strong>Markov decision processes (MDPs)</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Simple stochastic games (SMGs)</strong></td>
</tr>
<tr>
<td><strong>Continuous time</strong></td>
<td><strong>Continuous-time Markov chains (CTMCs)</strong></td>
<td><strong>Probabilistic timed automata (PTAs)</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Interactive Markov chains (IMCs)</strong></td>
</tr>
</tbody>
</table>
Course material

• 4th SSFT slides and lab session
  − http://www.prismmodelchecker.org/courses/ssft14/

• Reading
  − [DTMCs/MDPs/LTL] Principles of Model Checking by Baier and Katoen, MIT Press 2008

• See also
  − 20 lecture course taught at Oxford
    − http://www.prismmodelchecker.org/lectures/pmc/

• PRISM website www.prismmodelchecker.org
Part 1

Discrete-time Markov chains
Overview (Part 1)

• Introduction
• Model checking for discrete-time Markov chains (DTMCs)
  – DTMCs: definition, paths & probability spaces
  – PCTL model checking
  – Costs and rewards
• PRISM: overview
  – Modelling language
  – Properties
  – GUI, etc
  – Case studies: Bluetooth, DNA programming
• Summary
Discrete-time Markov chains

- **Discrete-time Markov chains (DTMCs)**
  - state-transition systems augmented with probabilities

- **States**
  - *discrete set of states* representing possible configurations of the system being modelled

- **Transitions**
  - transitions between states occur in *discrete time-steps*

- **Probabilities**
  - probability of making transitions between states is given by *discrete probability distributions*

![Diagram of a discrete-time Markov chain with states S0, S1, S2, and S3, showing transition probabilities and labels try, fail, succ.](image-url)
Discrete–time Markov chains

- Formally, a DTMC D is a tuple \((S, s_{init}, P, L)\) where:
  - \(S\) is a finite set of states ("state space")
  - \(s_{init} \in S\) is the initial state
  - \(P : S \times S \rightarrow [0,1]\) is the transition probability matrix
    where \(\sum_{s' \in S} P(s, s') = 1\) for all \(s \in S\)
  - \(L : S \rightarrow 2^{AP}\) is function labelling states with atomic propositions
- Note: no deadlock states
  - i.e. every state has at least one outgoing transition
  - can add self loops to represent final/terminating states
Paths and probabilities

- A (finite or infinite) path through a DTMC
  - is a sequence of states $s_0s_1s_2s_3...$ such that $P(s_i,s_{i+1}) > 0 \ \forall i$
  - represents an execution (i.e. one possible behaviour) of the system which the DTMC is modelling

- To reason (quantitatively) about this system
  - need to define a probability space over paths

- Intuitively:
  - sample space: $\text{Path}(s) = \text{set of all infinite paths from a state } s$
  - events: sets of infinite paths from $s$
  - basic events: cylinder sets (or “cones”)
  - cylinder set $C(\omega)$, for a finite path $\omega$
    - set of infinite paths with the common finite prefix $\omega$
  - for example: $C(ss_1s_2)$
Probability space over paths

- **Sample space** $\Omega = \text{Path}(s)$
  set of infinite paths with initial state $s$

- **Event set** $\Sigma_{\text{Path}(s)}$
  - the **cylinder set** $C(\omega) = \{ \omega' \in \text{Path}(s) \mid \omega \text{ is prefix of } \omega' \}$
  - $\Sigma_{\text{Path}(s)}$ is the **least $\sigma$–algebra** on $\text{Path}(s)$ containing $C(\omega)$ for all finite paths $\omega$ starting in $s$

- **Probability measure** $\Pr_s$
  - define probability $P_s(\omega)$ for finite path $\omega = s s_1 \ldots s_n$ as:
    - $P_s(\omega) = 1$ if $\omega$ has length one (i.e. $\omega = s$)
    - $P_s(\omega) = P(s, s_1) \cdot \ldots \cdot P(s_{n-1}, s_n)$ otherwise
    - define $\Pr_s(C(\omega)) = P_s(\omega)$ for all finite paths $\omega$
  - $\Pr_s$ extends **uniquely** to a probability measure $\Pr_s : \Sigma_{\text{Path}(s)} \to [0, 1]$

- See [KSK76] for further details
Probability space – Example

• Paths where sending fails the first time
  - $\omega = s_0s_1s_2$
  - $C(\omega) =$ all paths starting $s_0s_1s_2$...
  - $P_{s_0}(\omega) = P(s_0, s_1) \cdot P(s_1, s_2)$
    $$= 1 \cdot 0.01 = 0.01$$
  - $Pr_{s_0}(C(\omega)) = P_{s_0}(\omega) = 0.01$

• Paths which are eventually successful and with no failures
  - $C(s_0s_1s_3) \cup C(s_0s_1s_1s_3) \cup C(s_0s_1s_1s_1s_3) \cup ...$
  - $Pr_{s_0} (C(s_0s_1s_3) \cup C(s_0s_1s_1s_3) \cup C(s_0s_1s_1s_1s_3) \cup ...)$
    $$= P_{s_0}(s_0s_1s_3) + P_{s_0}(s_0s_1s_1s_3) + P_{s_0}(s_0s_1s_1s_1s_3) + ...$$
    $$= 1 \cdot 0.98 + 1 \cdot 0.01 \cdot 0.98 + 1 \cdot 0.01 \cdot 0.01 \cdot 0.98 + ...$$
    $$= 0.9898989898...$$
    $$= 98/99$$
PCTL

• Temporal logic for describing properties of DTMCs
  – PCTL = Probabilistic Computation Tree Logic [HJ94]
  – essentially the same as the logic pCTL of [ASB+95]

• Extension of (non–probabilistic) temporal logic CTL
  – key addition is probabilistic operator $P$
  – quantitative extension of CTL’s A and E operators

• Example
  – send \(\rightarrow P_{\geq 0.95} \text{ true } U^{\leq 10} \text{ deliver} \]
  – “if a message is sent, then the probability of it being delivered within 10 steps is at least 0.95”
PCTL syntax

- **PCTL syntax:**
  
  - $\phi ::= \text{true} | a | \phi \land \phi | \neg \phi | P_{\sim p} [ \psi ]$  
    (state formulas)
  
  - $\psi ::= X \phi | \phi U^{\leq k} \phi | \phi U \phi$  
    (path formulas)

- define $F \phi \equiv \text{true} U \phi$ (eventually), $G \phi \equiv \neg(F \neg \phi)$ (globally)
- where $a$ is an atomic proposition, used to identify states of interest, $p \in [0,1]$ is a probability, $\sim \in \{<,>,\leq,\geq\}$, $k \in \mathbb{N}$

- **A PCTL formula is always a state formula**
  
  - path formulas only occur inside the $P$ operator
PCTL semantics for DTMCs

- **PCTL formulas interpreted over states of a DTMC**
  - $s \models \phi$ denotes $\phi$ is “true in state $s$” or “satisfied in state $s$”

- **Semantics of (non-probabilistic) state formulas:**
  - for a state $s$ of the DTMC $(S, s_{\text{init}}, P, L)$:
    - $s \models a \iff a \in L(s)$
    - $s \models \phi_1 \land \phi_2 \iff s \models \phi_1$ and $s \models \phi_2$
    - $s \models \neg \phi \iff s \models \phi$ is false

- **Examples**
  - $s_3 \models \text{succ}$
  - $s_1 \models \text{try} \land \neg \text{fail}$
PCTL semantics for DTMCs

- **Semantics of path formulas:**
  - for a path \( \omega = s_0s_1s_2... \) in the DTMC:
    - \( \omega \models X \phi \iff s_1 \models \phi \)
    - \( \omega \models \phi_1 U^{\leq k} \phi_2 \iff \exists i \leq k \text{ such that } s_i \models \phi_2 \text{ and } \forall j < i, s_j \models \phi_1 \)
    - \( \omega \models \phi_1 U \phi_2 \iff \exists k \geq 0 \text{ such that } \omega \models \phi_1 U^{\leq k} \phi_2 \)

- **Some examples of satisfying paths:**
  - \( X \text{ succ} \)  \( \{\text{try}\} \{\text{succ}\} \{\text{succ}\} \{\text{succ}\} \)
    \[\begin{array}{cccc}
    s_1 & s_3 & s_3 & s_3 & \ldots \\
    \end{array}\]
  - \( \neg \text{fail} U \text{ succ} \)
    \( \{\text{try}\} \{\text{try}\} \{\text{succ}\} \{\text{succ}\} \)
    \[\begin{array}{cccc}
    s_0 & s_1 & s_3 & s_3 & \ldots \\
    \end{array}\]

- Diagram of DTMC with states and transitions.
PCTL semantics for DTMCs

- **Semantics of the probabilistic operator $P$**
  - Informal definition: $s \models P_{\sim p} [\psi]$ means that “the probability, from state $s$, that $\psi$ is true for an outgoing path satisfies $\sim p$”
  - Example: $s \models P_{<0.25} [X \text{ fail }] \iff \text{“the probability of atomic proposition fail being true in the next state of outgoing paths from } s \text{ is less than 0.25”}$
  - Formally: $s \models P_{\sim p} [\psi] \iff \text{Prob}(s, \psi) \sim p$
  - Where: $\text{Prob}(s, \psi) = \Pr_s \{ \omega \in \text{Path}(s) \mid \omega \models \psi \}$
  - (Sets of paths satisfying $\psi$ are always measurable [Var85])

\[
\begin{array}{c}
\text{s} \\
\text{Prob}(s, \psi) \sim p ?
\end{array}
\]
Quantitative properties

- Consider a PCTL formula $P_{\sim p} [ \psi ]$
  - if the probability is unknown, how to choose the bound $p$?
- When the outermost operator of a PTCL formula is $P$
  - we allow the form $P=? [ \psi ]$
  - “what is the probability that path formula $\psi$ is true?”
- Model checking is no harder: compute the values anyway
- Useful to spot patterns, trends

- Example
  - $P=? [ F \text{ err/total}>0.1 ]$
  - “what is the probability that 10% of the NAND gate outputs are erroneous?”
PCTL model checking for DTMCs

- Algorithm for PCTL model checking [CY88,HJ94,CY95]
  - inputs: DTMC $D=(S,s_{\text{init}},P,L)$, PCTL formula $\phi$
  - output: $\text{Sat}(\phi) = \{ s \in S \mid s \models \phi \}$ = set of states satisfying $\phi$

- What does it mean for a DTMC $D$ to satisfy a formula $\phi$?
  - sometimes, want to check that $s \models \phi \ \forall s \in S$, i.e. $\text{Sat}(\phi) = S$
  - sometimes, just want to know if $s_{\text{init}} \models \phi$, i.e. if $s_{\text{init}} \in \text{Sat}(\phi)$

- Sometimes, focus on quantitative results
  - e.g. compute result of $P=?[F \text{ error}]$
  - e.g. compute result of $P=?[F_{\leq k} \text{ error}]$ for $0 \leq k \leq 100$
PCTL model checking for DTMCs

• Basic algorithm proceeds by induction on parse tree of $\phi$
  - example: $\phi = (\neg\text{fail} \land \text{try}) \rightarrow P_{>0.95} [\neg\text{fail} \cup \text{succ}]$

• For the non-probabilistic operators:
  - Sat(true) = $S$
  - Sat(a) = $\{ s \in S \mid a \in L(s) \}$
  - Sat($\neg\phi$) = $S \setminus$ Sat($\phi$)
  - Sat($\phi_1 \land \phi_2$) = Sat($\phi_1$) $\cap$ Sat($\phi_2$)

• For the $P_{\sim_p}[\psi]$ operator
  - need to compute the probabilities $\text{Prob}(s, \psi)$
    for all states $s \in S$
  - focus here on “until” case: $\psi = \phi_1 \cup \phi_2$
• Computation of probabilities $\text{Prob}(s, \phi_1 U \phi_2)$ for all $s \in S$
  • First, identify all states where the probability is 1 or 0
    – $S_{\text{yes}} = \text{Sat}(\mathbb{P}_{\geq 1} [ \phi_1 U \phi_2 ])$
    – $S_{\text{no}} = \text{Sat}(\mathbb{P}_{\leq 0} [ \phi_1 U \phi_2 ])$
  • Then solve linear equation system for remaining states

• We refer to the first phase as “precomputation”
  – two algorithms: $\text{Prob0}$ (for $S_{\text{no}}$) and $\text{Prob1}$ (for $S_{\text{yes}}$)
  – algorithms work on underlying graph (probabilities irrelevant)

• Important for several reasons
  – reduces the set of states for which probabilities must be computed numerically (which is more expensive)
  – gives exact results for the states in $S_{\text{yes}}$ and $S_{\text{no}}$ (no round-off)
  – for $\mathbb{P}_{\sim p} [\cdot]$ where $p$ is 0 or 1, no further computation required
PCTL until – Linear equations

• Probabilities $\text{Prob}(s, \phi_1 \cup \phi_2)$ can now be obtained as the unique solution of the following set of linear equations:

\[
\text{Prob}(s, \phi_1 \cup \phi_2) = \begin{cases} 
1 & \text{if } s \in S^{\text{yes}} \\
0 & \text{if } s \in S^{\text{no}} \\
\sum_{s' \in S} P(s, s'). \text{Prob}(s', \phi_1 \cup \phi_2) & \text{otherwise}
\end{cases}
\]

– can be reduced to a system in $|S^2|$ unknowns instead of $|S|$ where $S^2 = S \setminus (S^{\text{yes}} \cup S^{\text{no}})$

• This can be solved with (a variety of) standard techniques
  – direct methods, e.g. Gaussian elimination
  – iterative methods, e.g. Jacobi, Gauss–Seidel, ...
    (preferred in practice due to scalability)
PCTL until – Example

- Example: $P_{>0.8} [\neg a U b ]$
PCTL until – Example

- Example: $P_{>0.8} [\neg a \mathbin{U} b ]$

\[ S_{\text{no}} = \]
\[ \text{Sat}(P_{\leq 0} [\neg a \mathbin{U} b ]) \]

\[ S_{\text{yes}} = \]
\[ \text{Sat}(P_{\geq 1} [\neg a \mathbin{U} b ]) \]
PCTL until – Example

• Example: $P_{>0.8} [\neg a \cup b ]$

• Let $x_s = \text{Prob}(s, \neg a \cup b)$

• Solve:

$x_4 = x_5 = 1$
$x_1 = x_3 = 0$
$x_0 = 0.1x_1 + 0.9x_2 = 0.8$
$x_2 = 0.1x_2 + 0.1x_3 + 0.3x_5 + 0.5x_4 = \frac{8}{9}$
$
\text{Prob}(\neg a \cup b) = x = [0.8, 0, \frac{8}{9}, 0, 1, 1]$

$S^{\text{no}} = \text{Sat}(P_{\leq 0} [\neg a \cup b ])$

$S^{\text{yes}} = \text{Sat}(P_{\geq 1} [\neg a \cup b ])$

$\text{Sat}(P_{>0.8} [\neg a \cup b ]) = \{ s_2, s_4, s_5 \}$
PCTL model checking – Summary

- **Computation of set** $\text{Sat}(\Phi)$ **for DTMC D and PCTL formula** $\Phi$
  - recursive descent of parse tree
  - combination of graph algorithms, numerical computation

- **Probabilistic operator $P$:**
  - $\Phi_1 \leq_k \Phi_2$: $k$ matrix–vector multiplications, $O(k|S|^2)$
  - $\Phi_1 \leq_k \Phi_2$: linear equation system, at most $|S|$ variables, $O(|S|^3)$

- **Complexity:**
  - linear in $|\Phi|$ and polynomial in $|S|$
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)

• More expressive logics can be used, for example:
  – 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~p […] always contains a single temporal operator)
  – supported by PRISM
  – (not covered in this lecture)

• Another direction: extend DTMCs with costs and rewards…
Costs and rewards

• We augment DTMCs 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 will consistently use the terminology “rewards” regardless
Reward–based properties

- Properties of DTMCs augmented with rewards
  - allow a wide range of quantitative measures of the system
  - basic notion: expected value of rewards
  - formal property specifications will be in an extension of PCTL

- More precisely, we use two distinct classes of property...

- **Instantaneous properties**
  - the expected value of the reward at some time point

- **Cumulative properties**
  - the expected cumulated reward over some period
For a DTMC \((S, s_{\text{init}}, P, L)\), a reward structure is a pair \((\rho, \iota)\)

- \(\rho : S \rightarrow \mathbb{R}_{\geq 0}\) is the state reward function (vector)
- \(\iota : S \times S \rightarrow \mathbb{R}_{\geq 0}\) is the transition reward function (matrix)

Example (for use with instantaneous properties)
- “size of message queue”: \(\rho\) maps each state to the number of jobs in the queue in that state, \(\iota\) is not used

Examples (for use with cumulative properties)
- “time-steps”: \(\rho\) returns 1 for all states and \(\iota\) is zero (equivalently, \(\rho\) is zero and \(\iota\) returns 1 for all transitions)
- “number of messages lost”: \(\rho\) is zero and \(\iota\) maps transitions corresponding to a message loss to 1
- “power consumption”: \(\rho\) is defined as the per-time-step energy consumption in each state and \(\iota\) as the energy cost of each transition

40
• Extend PCTL to incorporate reward–based properties
  – add an $R$ operator, which is similar to the existing $P$ operator

\[
\phi ::= \ldots \mid P_{\neg p}[\psi] \mid R_{\sim r}[I=k] \mid R_{\sim r}[C\leq k] \mid R_{\sim r}[F\phi]
\]

– where $r \in \mathbb{R}_{\geq 0}$, $\sim \in \{<,>,\leq,\geq\}$, $k \in \mathbb{N}$

• $R_{\sim r}[\cdot]$ means “the expected value of $\cdot$ satisfies $\sim r$”
Reward formula semantics

- Formal semantics of the three reward operators
  - based on random variables over (infinite) paths

- Recall:
  - \( s \models P_{\neg p} [ \psi ] \iff Pr_s \{ \omega \in \text{Path}(s) \mid \omega \models \psi \} \sim p \)

- For a state \( s \) in the DTMC (see [KNP07a] for full definition):
  - \( s \models R_{\sim r} [ I=^k ] \iff \text{Exp}(s, X_{I=^k}) \sim r \)
  - \( s \models R_{\sim r} [ C_{\leq k} ] \iff \text{Exp}(s, X_{C_{\leq k}}) \sim r \)
  - \( s \models R_{\sim r} [ F \Phi ] \iff \text{Exp}(s, X_{F\Phi}) \sim r \)

where: \( \text{Exp}(s, X) \) denotes the expectation of the random variable \( X : \text{Path}(s) \to \mathbb{R}_{\geq 0} \) with respect to the probability measure \( Pr_s \)
Model checking reward properties

- **Instantaneous**: $R_{=r}[I=k]$
- **Cumulative**: $R_{r}[C\leq k]$
  - variant of the method for computing bounded until probabilities
  - solution of recursive equations

- **Reachability**: $R_{r}[F\phi]$
  - similar to computing until probabilities
  - precomputation phase (identify infinite reward states)
  - then reduces to solving a system of linear equation

- **For more details, see e.g. [KNP07a]**
  - complexity not increased wrt classical PCTL
• **PRISM: Probabilistic symbolic model checker**
  - developed at Birmingham/Oxford University, since 1999
  - free, open source software (GPL), runs on all major OSs

• **Construction/analysis of probabilistic models...**
  - discrete-time Markov chains, continuous-time Markov chains,
    Markov decision processes, probabilistic timed automata,
    stochastic multi-player games, ...

• **Simple but flexible high-level modelling language**
  - based on guarded commands; see later...

• **Many import/export options, tool connections**
  - in: (Bio)PEPA, stochastic π-calculus, DSD, SBML, Petri nets, ...
  - out: Matlab, MRMC, INFAMY, PARAM, ...
• Model checking for various temporal logics…
  – PCTL, CSL, LTL, PCTL*, rPATL, CTL, …
  – quantitative extensions, costs/rewards, …

• Various efficient model checking engines and techniques
  – symbolic methods (binary decision diagrams and extensions)
  – explicit-state methods (sparse matrices, etc.)
  – statistical model checking (simulation-based approximations)
  – and more: symmetry reduction, quantitative abstraction refinement, fast adaptive uniformisation, …

• Graphical user interface
  – editors, simulator, experiments, graph plotting

• See: [http://www.prismmodelchecker.org/](http://www.prismmodelchecker.org/)
  – downloads, tutorials, case studies, papers, …
**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

```
[send] (s=2) -> p_{loss} : (s'=3) & (lost'=lost+1) + (1-p_{loss}) : (s'=4);
```

- action
- guard
- probability
- update
- probability
- update
Simple example

dtmc

module M1
    x : [0..3] init 0;
    [a] x=0 -> (x’ =1);
    [b] x=1 -> 0.5 : (x’ =2) + 0.5 : (x’ =3);
endmodule

module M2
    y : [0..3] init 0;
    [a] y=0 -> (y’ =1);
    [b] y=1 -> 0.4 : (y’ =2) + 0.6 : (y’ =3);
endmodule
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 “time”**
  - `true : 1;`
  - `endrewards`

  (cumulative, 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 “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:**
  - \( P_{\leq 0.01} \left[ F \text{ “crash” } \right] \) – “the probability of a crash is at most 0.01”
  - \( S_{>0.999} \left[ \text{“up”} \right] \) – “long–run probability of availability is >0.999”

- **Usually focus on quantitative (numerical) properties:**
  - \( P = ? \left[ F \text{ “crash” } \right] \)
    “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
  - $P_{\max=?}[F_{\leq 10} \text{ “fail”}]$ – “worst-case probability of a failure occurring within 10 seconds, for any possible scheduling of system components”
  - $P=?[G_{\leq 0.02} \text{ “deploy”} \{\text{“crash”}\{\text{max}\}]}$ – “the maximum probability of an airbag failing to deploy within 0.02s, from any possible crash scenario”

- Reward-based properties (rewards = costs = prices)
  - $R_{\{\text{“time”}\}=?}[F \text{ “end”}]$ – “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.”
PRISM GUI: Editing a model
PRISM GUI: The Simulator
PRISM GUI: Model checking and graphs
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
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

freq = [CLK_{15-12}+k+ (CLK_{4-2}0-CLK_{15-12}) \mod 16] \mod 32
DNA programming

• “Computing with soup” (The Economist 2012)
  – DNA strands are mixed together in a test tube
  – single strands are inputs and outputs
  – computation proceeds autonomously

• Can we transfer verification to this new application domain?
  – probability essential!
Case study: DNA programming

- 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 $<y \ t^>$

- 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:
(1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
(0,1,1,0,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
(0,0,1,0,1,1,1,1,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
(0,0,1,0,1,1,1,1,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
(0,0,1,0,1,1,0,1,0,0,1,1,1,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0)
(0,0,1,0,1,1,0,1,0,0,1,0,1,0,0,0,0,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0)
Summary

• **Discrete–time Markov chains (DTMCs)**
  – state transition systems + discrete probabilistic choice
  – probability space over paths through a DTMC

• **Property specifications**
  – probabilistic extensions of temporal logic, e.g. PCTL, LTL
  – also: expected value of costs/rewards

• **Model checking algorithms**
  – combination of graph–based algorithms, numerical computation, automata constructions
  – also applicable to continuous–time Markov chains via discretisation

• **Next: Markov decision processes (MDPs)**