Probabilistic model checking with PRISM: an overview

Marta Kwiatkowska
Department of Computer Science, University of Oxford

EQINOCS, Paris, January 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 → Result

System requirements → Temporal logic specification → ¬EF fail

Counter-example
Probabilistic model checking

System

Probabilistic model
e.g. Markov chain

Probabilistic model checker
e.g. PRISM

Result

Quantitative results

System requirements

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

$P_{<0.1} [ F \text{ fail} ]$
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>Discrete time</th>
<th>Continuous time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully probabilistic</td>
<td>Markov decision processes (MDPs)</td>
</tr>
<tr>
<td>DTMCs</td>
<td>Simple stochastic games (SMGs)</td>
</tr>
<tr>
<td>Continuous–time Markov chains (CTMCs)</td>
<td>Probabilistic timed automata (PTAs)</td>
</tr>
<tr>
<td>Nondeterministic</td>
<td>Interactive Markov chains (IMCs)</td>
</tr>
<tr>
<td>Discrete–time Markov chains (DTMCs)</td>
<td></td>
</tr>
<tr>
<td>Markov decision processes (MDPs)</td>
<td></td>
</tr>
<tr>
<td>Simple stochastic games (SMGs)</td>
<td></td>
</tr>
<tr>
<td>Probabilistic timed automata (PTAs)</td>
<td></td>
</tr>
<tr>
<td>Interactive Markov chains (IMCs)</td>
<td></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></td>
<td></td>
</tr>
<tr>
<td><strong>Discrete-time Markov chains (DTMCs)</strong></td>
<td></td>
<td>Markov decision processes (MDPs)</td>
</tr>
<tr>
<td></td>
<td>Simple stochastic games (SMGs)</td>
<td></td>
</tr>
<tr>
<td><strong>Continuous time</strong></td>
<td>Continuous-time Markov chains (CTMCs)</td>
<td>Probabilistic timed automata (PTAs)</td>
</tr>
<tr>
<td></td>
<td>Interactive Markov chains (IMCs)</td>
<td></td>
</tr>
</tbody>
</table>
Overview

• Introduction
• Model checking for discrete-time Markov chains (DTMCs)
  – DTMCs: definition, paths & probability spaces
  – PCTL model checking
  – Costs and rewards
  – Case studies: Bluetooth, (CTMC) DNA computing
• PRISM: overview
  – Functionality, GUI, etc
• PRISM: recent developments
  – e.g. multi-objective, parametric, etc
• 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
Discrete-time Markov chains

- Formally, a DTMC D is a tuple \((S, s_{\text{init}}, P, L)\) where:
  - \(S\) is a finite set of states ("state space")
  - \(s_{\text{init}} \in S\) is the initial state
  - \(P : S \times S \to [0, 1]\) is the transition probability matrix
    where \(\sum_{s' \in S} P(s, s') = 1\) for all \(s \in S\)
  - \(L : S \to 2^{\text{AP}}\) 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 $\omega = ss_1...s_n$ as:
    - $P_s(\omega) = 1$ if $\omega$ has length one (i.e. $\omega = s$)
    - $P_s(\omega) = P(s,s_1) \cdot ... \cdot P(s_{n-1},s_n)$ otherwise
  - define $\Pr_s(C(\omega)) = P_s(\omega)$ for all finite paths $\omega$
  - $\Pr_s$ extends uniquely to a probability measure $\Pr_s: \Sigma_{\text{Path}(s)} \rightarrow [0,1]$

- 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 \( s_0s_1s_2 \ldots \)
  – \( P_{s_0}(\omega) = P(s_0,s_1) \cdot P(s_1,s_2) \)
    \[ = 1 \cdot 0.01 = 0.01 \]
  – \( Pr_{s_0}(C(\omega)) = P_{s_0}(\omega) = 0.01 \)

• Paths which are eventually successful and with no failures
  – \( C(s_0s_1s_3) \cup C(s_0s_1s_1s_3) \cup C(s_0s_1s_1s_1s_3) \cup \ldots \)
  – \( Pr_{s_0}( C(s_0s_1s_3) \cup C(s_0s_1s_1s_3) \cup C(s_0s_1s_1s_1s_3) \cup \ldots ) \)
    \[ = P_{s_0}(s_0s_1s_3) + P_{s_0}(s_0s_1s_1s_3) + P_{s_0}(s_0s_1s_1s_1s_3) + \ldots \]
    \[ = 1 \cdot 0.98 + 1 \cdot 0.01 \cdot 0.98 + 1 \cdot 0.01 \cdot 0.01 \cdot 0.98 + \ldots \]
    \[ = 0.9898989898\ldots \]
    \[ = 98/99 \]
• **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 → \( P_{\geq 0.95} [ \text{true} \ U^{\leq 10} \text{deliver} ] \)
  – “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

  $\psi$ is true with probability $\sim p$
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 $(S, s_{init}, P, L)$:
    - $s \models a \iff a \in L(s)$
    - $s \models \phi_1 \land \phi_2 \iff s \models \phi_1$ 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 $\omega = s_0 s_1 s_2 \ldots$ in the DTMC:
  – $\omega \models X \phi \iff s_1 \models \phi$
  – $\omega \models \phi_1 U^{\leq k} \phi_2 \iff \exists i \leq k$ such that $s_i \models \phi_2$ and $\forall j < i$, $s_j \models \phi_1$
  – $\omega \models \phi_1 U \phi_2 \iff \exists k \geq 0$ such that $\omega \models \phi_1 U^{\leq k} \phi_2$

• Some examples of satisfying paths:
  – $X$ succ
    
    $$\{\text{try}\} \{\text{succ}\} \{\text{succ}\} \{\text{succ}\}$$
    $$s_0 \xrightarrow{0.01} s_1 \xrightarrow{0.98} s_1 \xrightarrow{1} s_3 \xrightarrow{1} \ldots$$
    
  – $\neg$fail U succ
    
    $$\{\text{try}\} \{\text{try}\} \{\text{succ}\} \{\text{succ}\}$$
    $$s_0 \xrightarrow{1} s_1 \xrightarrow{1} s_1 \xrightarrow{0.98} s_1 \xrightarrow{1} s_3 \xrightarrow{1} \ldots$$
• Semantics of the probabilistic operator $P$
  – informal definition: $s \models P_{\neg p} \left[ \psi \right]$ means that “the probability, from state $s$, that $\psi$ is true for an outgoing path satisfies $\neg p$”
  – example: $s \models P_{<0.25} \left[ X \text{ fail} \right] \iff “the probability of atomic proposition fail being true in the next state of outgoing paths from $s$ is less than 0.25”$
  – formally: $s \models P_{\neg p} \left[ \psi \right] \iff \text{Prob}(s, \psi) \sim p$
  – 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])

![Diagram of state transitions and probabilities](image)
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
  - $P_{=?} [ \text{F err/total} > 0.1 ]$
  - “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: $Sat(\phi) = \{ s \in S \mid s \models \phi \} = \text{set of states satisfying } \phi$

• **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. $Sat(\phi) = S$
  - sometimes, just want to know if $s_{\text{init}} \models \phi$, i.e. if $s_{\text{init}} \in Sat(\phi)$

• **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:
  - $\text{Sat}($true$) = S$
  - $\text{Sat}(a) = \{ s \in S \mid a \in L(s) \}$
  - $\text{Sat}(\neg \phi) = S \setminus \text{Sat}(\phi)$
  - $\text{Sat}(\phi_1 \land \phi_2) = \text{Sat}(\phi_1) \cap \text{Sat}(\phi_2)$

- 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: $\psi = \phi_1 \cup \phi_2$
PCTL until for DTMCs

- 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
  - $S^{\text{yes}} = \text{Sat}(P_{\geq 1}[\phi_1 U \phi_2])$
  - $S^{\text{no}} = \text{Sat}(P_{\leq 0}[\phi_1 U \phi_2])$
- Then solve linear equation system for remaining states

- We refer to the first phase as “precomputation”
  - two algorithms: Prob0 (for $S^{\text{no}}$) and 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 $P_{\sim p}[\cdot]$ where $p$ is 0 or 1, no further computation required
PCTL until – Linear equations

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

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

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

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

\[
\begin{align*}
S_{\text{no}} &= \text{Sat}(P_{\leq 0} [\neg a U b ]) \\
S_{\text{yes}} &= \text{Sat}(P_{\geq 1} [\neg a U b ])
\end{align*}
\]
PCTL until – Example

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

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

• Solve:

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

$\text{Sat}(P_{\leq 0} [\neg a \cup b ])$

$S_{no} = \{ s_2, s_4, s_5 \}$

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

$S_{yes} = \{ \}$
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 $P$:
  - $X \Phi$: one matrix–vector multiplication, $O(|S|^2)$
  - $\Phi_1 U \leq_k \Phi_2$: $k$ matrix–vector multiplications, $O(k|S|^2)$
  - $\Phi_1 U \Phi_2$: 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, $P_{\neg p} [...]$ 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 \((\rho, \iota)\)
- \(\rho : S \to \mathbb{R}_{\geq 0}\) is the state reward function (vector)
- \(\iota : S \times S \to \mathbb{R}_{\geq 0}\) 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
PCTL and rewards

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

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

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

- **Reachability**: $R_{\sim r} \ [ F \phi ]$
  - 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
PCTL model checking summary…

• Introduced probabilistic model checking for DTMCs
  – discrete time and probability only
  – PCTL model checking via linear equation solving
  – LTL also supported, via automata-theoretic methods

• Continuous-time Markov chains (CTMCs)
  – discrete states, continuous time
  – temporal logic CSL
  – model checking via uniformisation, a discretisation of the CTMC

• Markov decision processes (MDPs)
  – add nondeterminism to DTMCs
  – PCTL, LTL and PCTL* supported
  – model checking via linear programming
• **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/
  – downloads, tutorials, case studies, papers, …
PRISM GUI: Editing a model

```plaintext
// Service Queue (SQ)
// Stores requests which arrives into the system to be processed.
const int q_max = 20;

// Request arrival rate
const double rate_arrive = 1/0.72; // (mean inter-arrival time is 0.72 seconds)

module SQ

// q = number of requests currently in queue
q : [0..q_max] init 0;

// A request arrives
[request] true -> rate_arrive : (q'=min(q+1,q_max));

// A request is served
[serv] q+1 -> (q'=q-1);

// Last request is served
[serve_last] q=1 -> (q=q-1);

endmodule

// Service Provider (SP)
// Processes requests from service queue.
// The SP has 3 power states: sleep, idle and busy
// Rate of service (average service time = 0.008s)
const double rate_serv = 1/0.008;

// Rate of switching from sleep to idle (average transition time = 1.6s)
const double rate_s2i = 1/1.6;

// Rate of switching from idle to sleep (average transition time = 0.67s)
const double rate_i2s = 1/0.67;
```

Built Model
- States: 42
- Initial states: 1
- Transitions: 81

Model Properties Simulator Log

Building model... done.

Building model... done.
PRISM GUI: The Simulator

![PRISM GUI](image)

<table>
<thead>
<tr>
<th>Step</th>
<th>Time (s)</th>
<th>Left</th>
<th>Right</th>
<th>Repair</th>
<th>Line</th>
<th>ToLeft</th>
<th>ToRight</th>
<th>Rewards</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>100</td>
</tr>
<tr>
<td>1</td>
<td>12.0649</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>90</td>
</tr>
<tr>
<td>2</td>
<td>12.0806</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>12.1674</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>90</td>
</tr>
<tr>
<td>4</td>
<td>12.2677</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>12.2809</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>90</td>
</tr>
<tr>
<td>6</td>
<td>12.3071</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>100</td>
</tr>
<tr>
<td>7</td>
<td>12.3446</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>90</td>
</tr>
<tr>
<td>8</td>
<td>12.3653</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>100</td>
</tr>
<tr>
<td>9</td>
<td>12.4059</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>90</td>
</tr>
<tr>
<td>10</td>
<td>12.4585</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>100</td>
</tr>
<tr>
<td>11</td>
<td>15.6657</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>90</td>
</tr>
<tr>
<td>12</td>
<td>15.6834</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>100</td>
</tr>
<tr>
<td>13</td>
<td>15.7585</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>90</td>
</tr>
<tr>
<td>14</td>
<td>15.8505</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>100</td>
</tr>
<tr>
<td>15</td>
<td>15.874</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>90</td>
</tr>
<tr>
<td>16</td>
<td>15.9084</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>100</td>
</tr>
</tbody>
</table>

[Image 27x61 to 568x98]

[Image 121x146 to 551x716]
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
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...)

\[ t \quad x \]
\[ y \quad t \]
\[ t \quad x \quad t \quad a \quad t \quad a \]
\[ x \quad t \quad y \quad t \quad a \quad t \]
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:
(1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
(0,1,1,0,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
(0,0,1,0,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
(0,0,1,0,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
(0,0,1,0,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
(0,0,1,0,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)

reactive gates
PRISM: Recent & new developments

• Major new features:
  1. multi-objective model checking
  2. parametric model checking
  3. real-time: probabilistic timed automata (PTAs)
  4. games: stochastic multi-player games (SMGs)

• Further new additions:
  – strategy (adversary) synthesis (see ATVA’13 invited lecture)
  – CTL model checking & counterexample generation
  – enhanced statistical model checking
    (approximations + confidence intervals, acceptance sampling)
  – efficient CTMC model checking
    (fast adaptive uniformisation) [Mateescu et al., CMSB'13]
  – benchmark suite & testing functionality [QEST'12]
    www.prismmodelchecker.org/benchmarks/
1. 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)
  - \( P_{<0.01}[ \text{F err} ] : \) “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?”
1. 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?”
  - $\text{multi}(P_{\text{max=?}}[F \text{ transmit }], R_{\text{time} > 10}[C])$

- **Pareto queries**
  - e.g. "Pareto curve for maximising probability of transmission and expected battery life-time”
  - $\text{multi}(P_{\text{max=?}}[F \text{ transmit }], R_{\text{time max=?}}[C])$
Case study: Dynamic power management

- **Synthesis of dynamic power management schemes**
  - for an IBM TravelStar VP disk drive
  - 5 different power modes: active, idle, idlep, 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

Conclusion

- Introduction to probabilistic model checking
- Overview of PRISM
- More models and logics
  - continuous-time Markov chains
  - Markov decision processes
  - probabilistic timed automata
  - stochastic multi-player games
- Related/future work
  - quantitative runtime verification [TSE’11, CACM’12]
  - statistical model checking [TACAS’04, STTT’06]
  - multi-objective stochastic games [MFCS’13, QEST’13]
  - verification of cardiac pacemakers [RTSS’12, HSCC’13]
  - probabilistic hybrid automata [CPSWeek’13 tutorial]
References

• Tutorial papers

• PRISM tool paper
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](http://www.veriware.org)
  – **PRISM** [www.prismmodelchecker.org](http://www.prismmodelchecker.org)