Verification of Probabilistic Real-time Systems

Dave Parker

University of Birmingham

ETR'13, Toulouse, August 2013
What is probabilistic model checking?

• **Formal verification**…
  – is the application of *rigorous*, mathematics–based techniques to establish the *correctness* of computerised systems

• **Probabilistic model checking**…
  – is an automated formal verification technique for modelling and analysis of systems with *probabilistic* behaviour
Model checking

- Finite-state model
- Temporal logic specification
- System requirements
- Model checker (e.g., SMV, Spin)
- Result
  - ✔️
  - ❌
- Counter-example

¬EF fail
Probabilistic model checking

System

Probabilistic model
e.g. Markov chain

0.5
0.4
0.1

System requirements

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

Probabilistic model checker
e.g. PRISM

Result

Quantitative results

Counter-example

\[ P_{<0.1} \left[ F \text{ fail} \right] \]
Why probability?

• Many real-world systems are inherently probabilistic...

• **Unreliable or unpredictable behaviour**
  – failures of physical components
  – message loss in wireless communication

• **Use of randomisation (e.g. to break symmetry)**
  – random back-off in communication protocols
  – in gossip routing to reduce flooding
  – in security protocols, e.g. for anonymity

• And many others...
  – biological processes, e.g. DNA computation
  – quantum computing algorithms
Probabilistic real-time systems

• Many systems combine *probability* and *real-time*
  – e.g. wireless communication protocols
  – e.g. randomised security protocols

• Randomised back-off schemes
  – Ethernet, WiFi (802.11), Zigbee (802.15.4)

• Random choice of waiting time
  – Bluetooth device discovery phase
  – Root contention in IEEE 1394 FireWire

• Random choice over a set of possible addresses
  – IPv4 dynamic configuration (link-local addressing)

• Random choice of a destination
  – Crowds anonymity, gossip-based routing
Verifying probabilistic systems

- **We are not just interested in correctness**
  - “the probability of an airbag failing to deploy within 0.02 seconds of being triggered is at most 0.001”

- **We want to be able to reason about:**
  - reliability, dependability
  - performance, resource usage, e.g. battery life
  - security, privacy, trust, anonymity, fairness
  - and much more…

- **We want to reason in a quantitative manner:**
  - how reliable is my car’s Bluetooth network?
  - how efficient is my phone’s power management policy?
  - how secure is my bank’s web-service?
# Probabilistic models

<table>
<thead>
<tr>
<th>Discrete time</th>
<th>Fully probabilistic</th>
<th>Nondeterministic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Discrete-time Markov chains (DTMCs)</td>
<td>Markov decision processes (MDPs) (probabilistic automata)</td>
</tr>
<tr>
<td>Continuous time</td>
<td>Continuous-time Markov chains (CTMCs)</td>
<td>Probabilistic timed automata (PTAs)</td>
</tr>
<tr>
<td></td>
<td>CTMDPs/IMCs/…</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Fully probabilistic</td>
<td>Nondeterministic</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------------------------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td><strong>Discrete time</strong></td>
<td>Discrete-time Markov chains (DTMCs)</td>
<td>Markov decision processes (MDPs)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(probabilistic automata)</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></td>
<td></td>
</tr>
</tbody>
</table>
Contents

• Case study: the FireWire protocol

• Discrete–time Markov chains + the logic PCTL

• Adding nondeterminism: Markov decision processes

• Adding real time: probabilistic timed automata

• Probabilistic model checking in practice: PRISM

• More here: http://www.prismmodelchecker.org/lectures/
Contents

• Case study: the FireWire protocol

• Discrete-time Markov chains + the logic PCTL

• Adding nondeterminism: Markov decision processes

• Adding real time: probabilistic timed automata

• Probabilistic model checking in practice: PRISM
Case study: FireWire protocol

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

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

Root node
FireWire root contention
FireWire root contention
FireWire analysis

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

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

“minimum probability of electing leader by time T”
FireWire: Analysis results

“minimum probability of electing leader by time $T$”

(Short wire length)

Using a biased coin
FireWire: Analysis results

“maximum expected time to elect a leader”

(.short wire length)

Using a biased coin
FireWire: Analysis results

“maximum expected time to elect a leader”

(short wire length)

Using a biased coin is beneficial!
Contents

• Case study: the FireWire protocol

• Discrete–time Markov chains + the logic PCTL

• Adding nondeterminism: Markov decision processes

• Adding real time: probabilistic timed automata

• Probabilistic model checking in practice: PRISM
Discrete–time Markov chains (DTMCs)

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

- States
  - discrete set of states representing all 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
  – \(P : S \times S \rightarrow [0,1]\) is the transition probability matrix
  – \(L : S \rightarrow 2^\text{AP}\) is function labelling states with atomic propositions

• A (finite or infinite) path through a DTMC
  – is a sequence of states \(s_0 s_1 s_2 s_3 \ldots\) 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 formally about the DTMC
  – we define a probability measure over paths, \(Pr_s\)
  – via a sigma algebra over the set of all infinite paths
PCTL

- **PCTL**: temporal logic for describing properties of DTMCs
  - PCTL = Probabilistic Computation Tree Logic [HJ94, BdA95]

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

- **Example**
  - $\text{send} \rightarrow P_{\geq 0.95} [ F_{\leq 10} \text{ deliver } ]$
  - “if a message is sent, then the probability of it being delivered within 10 steps is at least 0.95”
PCTL syntax

- **Syntax of PCTL formula** $\phi$:

  - $\phi ::= \text{true} \mid a \mid \phi \land \phi \mid \neg \phi \mid \mathit{P}_{\sim p} [ \psi ]$

  (state formulae)

  - $\psi ::= \mathit{X} \phi \mid \phi \mathit{U}^{\leq k} \phi \mid \phi \mathit{U} \phi$

    (path formulae)

  - where $a$ is an atomic proposition, used to identify states of interest, $p \in [0,1]$ is a probability, $\sim \in \{<,>,\leq,\geq\}$ and $k \in \mathbb{N}$

- **Can derive other useful operators**

  - logical: $\text{false}$, $\phi_1 \lor \phi_2$, $\phi_1 \rightarrow \phi_2$

  - $\mathit{F} \phi \equiv \text{true} \mathit{U} \phi$ ("eventually") and $\mathit{G} \phi \equiv \neg (\mathit{F} \neg \phi)$ ("always")

  - bounded variants, e.g. $\mathit{F}^{\leq k} \phi \equiv \text{true} \mathit{U}^{\leq k} \phi$
PCTL semantics (for DTMCs)

• PCTL formulae interpreted over states of a DTMC
  – \( s \vDash \phi \) denotes \( \phi \) is “true in state \( s \)” or “satisfied in state \( s \)”

• Semantics of logical operators: standard meanings

• Semantics of the probabilistic operator \( P \)
  – informally, \( s \vDash P_{\sim p} [\psi] \) means: “the probability, from state \( s \), that \( \psi \) is true for outgoing paths satisfies the bound \( \sim p \)”
  – formally:
    \[
    s \vDash P_{\sim p} [\psi] \iff \text{Prob}(s, \psi) \sim p
    \]
  – where:
    \[
    \text{Prob}(s, \psi) = \text{Pr}_s \{ \omega \in \text{Path}(s) \mid \omega \vDash \psi \}
    \]
Quantitative (numerical) properties

- Consider a PCTL formula $P_{\sim p} [ \psi ]$
  - if the probability is unknown, how to choose the bound $p$?
- We also allow the numerical form $P_{=}? [ \psi ]$
  - when the outermost operator of a PTCL formula is $P$
  - “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?”
Some real PCTL examples

- **NAND multiplexing system**
  - $P_{=?} [ F \text{ err/total} \geq 0.1 ]$
  - “what is the probability that 10% of the NAND gate outputs are erroneous?”

- **Bluetooth wireless communication protocol**
  - $P_{=?} [ F^{\leq t} \text{ reply_count} = k ]$
  - “what is the probability that the sender has received $k$ acknowledgements within $t$ clock-ticks?”

- **Security: EGL contract signing protocol**
  - $P_{=?} [ F (\text{pairs_a=0} \& \text{pairs_b}>0) ]$
  - “what is the probability that the party B gains an unfair advantage during the execution of the protocol?”
PCTL model checking for DTMCs

- **Algorithm for PCTL model checking** [CY88,HJ94,CY95]
  - inputs: DTMC D=(S, s_{init}, P, L), PCTL formula φ
  - output: Sat(φ) = \{ s ∈ S | s ⊨ φ \} = set of states satisfying φ
  - or: compute result of e.g. P_{≥}\ 0.95 [ F≤k error ]

- **Basic algorithm proceeds by induction on parse tree of φ**
  - e.g. φ = (¬fail ∧ try) → P_{>0.95} [ ¬fail U succ ]
  - logical operators: straightforward

- **For the P_{≤p} [ ψ ] operator**
  - need to compute probabilities Prob(s, ψ) for all states s ∈ S
  - combination of graph algorithms and numerical computation

- **Linear in |Φ| and polynomial in |S|**
PCTL model checking: Until

• Example: computation of probabilities for "until" formula
  – i.e. Prob(s, \( \phi_1 U \phi_2 \)) for all \( s \in S \)

• First, execute graph–based analysis to identify all states where the probability is exactly 1 or 0:
  – \( S^{yes} = Sat(P_{\geq 1} [ \phi_1 U \phi_2 ]) \)
  – \( S^{no} = Sat(P_{\leq 0} [ \phi_1 U \phi_2 ]) \)

• Then, solve linear equation system for remaining states:

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

– solved with standard methods, e.g. Gaussian elimination (iterative numerical methods preferred in practice)
Example: $P_{>0.8} [\neg a \lor b]$
Example: $P_{>0.8} \left[ \neg a \cup b \right]$

$S_{no} = \text{Sat}(P_{\leq 0} \left[ \neg a \cup b \right])$

$S_{yes} = \text{Sat}(P_{\geq 1} \left[ \neg a \cup b \right])$
PCTL until – Example

- Example: \( P_{>0.8} [\neg a \cup b] \)
- Let \( x_s = \text{Prob}(s, \neg a \cup b) \)
- Solve:

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

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

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

\( S_{\text{no}} = \{ s_2, s_4, s_5 \} \)
Limitations of PCTL

- **PCTL**, although useful in practice, has limited expressivity
  - essentially: probability of reaching states in \( T \), passing only through states in \( T' \) (and within \( k \) time-steps)

- More expressive logics can be used, for example:
  - **LTL** [Pnu77] – linear-time temporal logic
  - **PCTL*** [ASB+95, BdA95] – which subsumes both PCTL and LTL
  - both allow temporal operators to be combined

- **LTL properties**:
  - \( P_{\leq 0.01} [ (F \text{ tmp\_fail}_1) \land (F \text{ tmp\_fail}_2) ] \) – “both servers eventually fail with probability at most 0.01”
  - \( P_{\geq 1} [ G F \text{ ready} ] \) – “with probability 1, the server always eventually returns to a ready-state”
  - \( P_{=?} [ F G \text{ error} ] \) – “probability of an irrecoverable error?”
Costs and rewards

• Another direction: extend DTMCs with costs and rewards…
  – to measure: elapsed time, power consumption, number of messages successfully delivered, net profit, …
  – add expected reward operator $R$ to PCTL logic

• Cost/reward–based properties:
  – $R_{\text{energy}} \leq 400 \ [ C \leq 60 ]$ – “the expected energy consumption over 60 seconds is at most 40 J”
  – $R_{\text{time}} = ? \ [ F \text{ end} ]$ – “the expected time for protocol execution”
Contents

• Case study: the FireWire protocol

• Discrete-time Markov chains + the logic PCTL

• Adding nondeterminism: Markov decision processes

• Adding real time: probabilistic timed automata

• Probabilistic model checking in practice: PRISM
Nondeterminism

- Some aspects of a system may not be probabilistic and should not be modelled probabilistically; for example:
  - Concurrency – scheduling of parallel components
    - e.g. randomised distributed algorithms – multiple probabilistic processes operating asynchronously
  - Unknown environments or controllers
    - e.g. probabilistic security protocols – unknown adversary
    - e.g. controller synthesis & planning
  - Underspecification and abstraction
    - e.g. a probabilistic communication protocol designed for message propagation delays of between \( d_{\text{min}} \) and \( d_{\text{max}} \)
• **Markov decision processes (MDPs)**
  – extension of DTMCs which allow **nondeterministic choice**

• **Like DTMCs:**
  – discrete set of states representing possible configurations of the system being modelled
  – transitions between states occur in discrete time-steps

• **Probabilities and nondeterminism**
  – in each state, a nondeterministic choice between several actions
  – each of which gives a probability distributions over successor states
  – formally: \( \delta : S \times \text{Act} \rightarrow \text{Dist}(S) \)
  – instead of \( P : S \times S \rightarrow [0,1] \)
Adversaries

• **How to reason about probabilities for MDPs?**
  – need to separate nondeterminism and probability

• **An adversary resolves nondeterministic choice in an MDP**
  – based on the history of execution so far
  – also known as “schedulers”, “strategies” or “policies”
  – formally: an adversary $\sigma$ of an MDP is a function mapping every finite path $s_0a_0s_1a_1...s_n$ to an action available in $s_n$

• **Adversary $\sigma$ induces a probability measure $Pr_{s,\sigma}$ over paths**
  – via construction of an (infinite-state) DTMC
• **Consider the simple MDP below**
  – \( s_1 \) is the only state for which an adversary makes a choice

• **Adversary \( \sigma_1 \)**
  – picks action \( c \) the first time
  – \( \sigma_1(s_0s_1)=c \)

• **Adversary \( \sigma_2 \)**
  – picks action \( b \) the first time, then \( c \)
  – \( \sigma_2(s_0s_1)=b, \sigma_2(s_0s_1s_1)=c, \sigma_2(s_0s_1s_0s_1)=c \)
Adversaries – Examples

- Fragment of DTMC for adversary $\sigma_1$
  - $\sigma_1$ picks action $c$ the first time
Adversaries – Examples

- Fragment of DTMC for adversary $\sigma_2$
  - $\sigma_2$ picks action b, then c

```
\begin{align*}
  s_0 &\xrightarrow{0.7} s_0s_1s_0 &\xrightarrow{1} s_0s_1s_0s_1 \\
  s_0 &\xrightarrow{0.3} s_0s_1s_1 &\xrightarrow{0.5} s_0s_1s_1s_2 \\
  s_0s_1s_0 &\xrightarrow{1} s_0s_1s_0s_1 &\xrightarrow{0.5} s_0s_1s_0s_1s_2 \\
  s_0s_1s_1 &\xrightarrow{0.5} s_0s_1s_1s_2 &\xrightarrow{1} s_0s_1s_1s_2s_2 \\
  s_0s_1s_1s_2 &\xrightarrow{1} s_0s_1s_1s_2s_3 &\xrightarrow{0.5} s_0s_1s_1s_2s_3s_3 \\
  \vdots & & \vdots \\
  s_0 &\xrightarrow{1} s_2 &\xrightarrow{1} s_3 \\
  s_2 &\xrightarrow{c} s_1 &\xrightarrow{b} s_0 \\
}\end{align*}
```
Model checking for MDPs

- Verification for MDPs quantifies over all adversaries
  - e.g. PCTL: $P_{\geq 0.95} [ F \text{ deliver } ]$ – "the probability of the message being delivered is at least 0.95 for any possible adversary"
  - formally: $s \models P_{\sim p}[\psi] \iff \Pr_s^{\sigma}(\psi) \sim p$ for all adversaries $\sigma$

- For model checking, we need min./max. probabilities:
  - $\Pr_s^{\max}(\psi) = \sup_\sigma \Pr_s^{\sigma}(\psi)$ and $\Pr_s^{\min}(\psi) = \inf_\sigma \Pr_s^{\sigma}(\psi)$

- Quantitative (numerical) queries
  - $P_{\min=?}[\psi]$ and $P_{\max=?}[\psi]$
  - analyses best-case or worst-case behaviour of the system

![Graph showing probability over time](image)
PCTL model checking for MDPs

• Basic algorithm same as PCTL model checking for DTMCs
  – recursive procedure, graph-based + numerical solution
  – now: computation of min/max probabilities
  – still linear in size of property, polynomial in size of model

• For example, for "until" formulae
  – either: solve linear programming (LP) problem
  – or: iterative numerical methods (dynamic programming)
  – or: policy iteration
Contents

• Case study: the FireWire protocol
• Discrete-time Markov chains + the logic PCTL
• Adding nondeterminism: Markov decision processes
• Adding real time: probabilistic timed automata
• Probabilistic model checking in practice: PRISM
• Systems with probability, nondeterminism and real-time
  – e.g. communication protocols, randomised security protocols

• Randomised back-off schemes
  – Ethernet, WiFi (802.11), Zigbee (802.15.4)

• Random choice of waiting time
  – Bluetooth device discovery phase
  – Root contention in IEEE 1394 FireWire

• Random choice over a set of possible addresses
  – IPv4 dynamic configuration (link-local addressing)

• Random choice of a destination
  – Crowds anonymity, gossip-based routing
Probabilistic timed automata (PTAs)

- Probabilistic timed automata (PTAs)
  - Markov decision processes (MDPs) + real-valued clocks
  - or: timed automata + discrete probabilistic choice
  - model probabilistic, nondeterministic and timed behaviour

- PTAs comprise:
  - clocks (increase simultaneously)
  - locations (labelled with invariants)
  - transitions (action + guard + probabilities + resets)

- Semantics
  - PTA represents an infinite-state MDP
  - states are location/clock valuation pairs \((l,v) \in \text{Loc} \times \mathbb{R}^X\)
  - nondeterminism: elapse of time + choice of actions
• Example:

- **Invariant:**
  - $x \leq 2$
  - $x \geq 1$

- **Guard:**
  - $0.9$
  - $0.1$

- **Action:**
  - $x := 0$
  - $x := 0$

- **Location:**
  - 0.95

- **Clock Reset:**
  - $x \leq 3$
  - $x := 0$

- **Done:**
  - $x \geq 2$
  - $x := 0$

- **Init:**
  - $x := 0$
  - $x := 0$

- **Send:**
  - $x := 1$

- **Lost:**
  - 0.05
PTA – Example execution

PTA:

Example execution:

(init, x=0) ——> 1.1

(init, x=1.1) ——> 0.9 ——> send

0.1 ——> x:=0

0.95 ——> retry

x ≥ 2

x:=0

0.1 ——> x:=0

0.9 ——> send

x ≥ 2

x:=0

0.05 ——> done

true

x:=0

8.66 ——> done

x=8.66

true

0.95 ——> retry

0.05

2.7 ——> done

x=2.7

true

0.95 ——> retry

0.05

...
Properties of PTAs

• **Temporal logic**
  – again, can use PCTL to represent properties
  – e.g. $P_{\geq 0.99} [ F_{\leq 5} \text{deliv} ]$ – “with probability 0.99 or greater, a data packet will always be delivered within 5 seconds”
  – we verify behaviour over all possible adversaries (actually all time–divergent adversaries)

• **Timed extensions**
  – can extend to the logic PTCTL (adds zones + formula clocks)

• **In practice:**
  – (min/max) probabilistic reachability often suffices
PTA model checking

• Several different approaches developed
  – basic idea: reduce to the analysis of a finite-state model
  – in most cases, this is a Markov decision process (MDP)

• Region graph construction [KNSS02]
  – shows decidability, but gives exponential complexity

• Digital clocks approach [KNPS06]
  – (slightly) restricted classes of PTAs
  – works well in practice, still some scalability limitations

• Zone-based approaches:
  – (preferred approach for non-probabilistic timed automata)
  – backwards reachability [KNSW07]
  – game-based abstraction refinement [KNP09c]
• Case study: the FireWire protocol

• Discrete-time Markov chains + the logic PCTL

• Adding nondeterminism: Markov decision processes

• Adding real time: probabilistic timed automata

• Probabilistic model checking in practice: PRISM
The PRISM tool

- **PRISM: Probabilistic symbolic model checker**
  - developed at Birmingham/Oxford University, since 1999
  - free, open source (GPL), runs on all major OSs

- **Support for:**
  - discrete-/continuous-time Markov chains (D/CTMCs)
  - Markov decision processes (MDPs)
  - probabilistic timed automata (PTAs)
  - PCTL, CSL, LTL, PCTL*, costs/rewards, ...

- **Features:**
  - simple but flexible high-level modelling language
  - user interface: editors, simulator, experiments, graph plotting
  - multiple efficient model checking engines (e.g. symbolic)
  - (mostly symbolic – BDDs; up to $10^{10}$ states, $10^7$–$10^8$ on avg.)

- **See:** [http://www.prismmodelchecker.org/](http://www.prismmodelchecker.org/)
Modelling PTAs in PRISM

- **PTA example**: message transmission over faulty channel

```
- init: x \leq 2
- lost: x \leq 5

States
- locations + data variables

Transitions
- guards and action labels

Real-valued clocks
- state invariants, guards, resets

Probability
- discrete probabilistic choice

tries := 0
send x \geq 1 \land \text{tries} \leq N
x := 0
retry x \geq 3
x := 0, \text{tries} := \text{tries} + 1
0.1
0.9
```

```
quit tries > N
```

```
true
```

```
true
```

```
0.1
```

```
0.9
```

Modelling PTAs in PRISM

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

```plaintext
pta
cost 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

```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

**For PTAs:**
- clocks
- invariants
- guards/resets

---

For PTAs:
- clocks
- invariants
- guards/resets
Modelling PTAs in PRISM

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

\[\text{pta}\]
\[\text{const int } N;\]
\[\text{module } \text{transmitter}\]
\[s : [0..3] \text{ init } 0;\]
\[\text{tries} : [0..N+1] \text{ init } 0;\]
\[x : \text{clock};\]
\[\text{invariant } (s=0 \Rightarrow x \leq 2) \land (s=1 \Rightarrow x \leq 5) \text{ endinvariant}\]
\[[\text{send}] s=0 \land \text{tries} \leq N \land x \geq 1\]
\[\rightarrow 0.9 : (s'=3)\]
\[+ 0.1 : (s'=1) \land (\text{tries}'=\text{tries}+1) \land (x'=0);\]
\[[\text{retry}] s=1 \land x \geq 3 \rightarrow (s'=0) \land (x'=0);\]
\[[\text{quit}] s=0 \land \text{tries} > N \rightarrow (s'=2);\]

\text{endmodule}

\[\text{rewards } \text{“energy” } (s=0) : 2.5; \text{ endrewards}\]
PRISM – Case studies

- Randomised communication protocols
  - Bluetooth, FireWire, Zeroconf, 802.11, Zigbee, gossiping, ...
- Randomised distributed algorithms
  - consensus, leader election, self-stabilisation, ...
- Security protocols/systems
  - pin cracking, anonymity, quantum crypto, contract signing, ...
- Planning & controller synthesis
  - robotics, dynamic power management, ...
- Performance & reliability
  - nanotechnology, cloud computing, manufacturing systems, ...
- Biological systems
  - cell signalling pathways, DNA computation, ...

See: www.prismmodelchecker.org/casestudies
Summary

• **Probabilistic model checking**
  – automated verification of systems with probabilistic behaviour
  – (randomisation, failures, message losses, ...)

• **Probabilistic models**
  – discrete-time Markov chains (fully probabilistic)
  – Markov decision processes (plus nondeterminism)
  – probabilistic timed automata (plus real-time)

• **Property specification**
  – probabilistic temporal logics, e.g. PCTL
  – wide range of quantitative properties

• **Tool support: PRISM** ([http://www.prismmodelchecker.org/](http://www.prismmodelchecker.org/))
  – demonstrations available
Questions?

More info here:
www.prismmodelchecker.org