Verification of Probabilistic Real-time Systems

Dave Parker

University of Birmingham

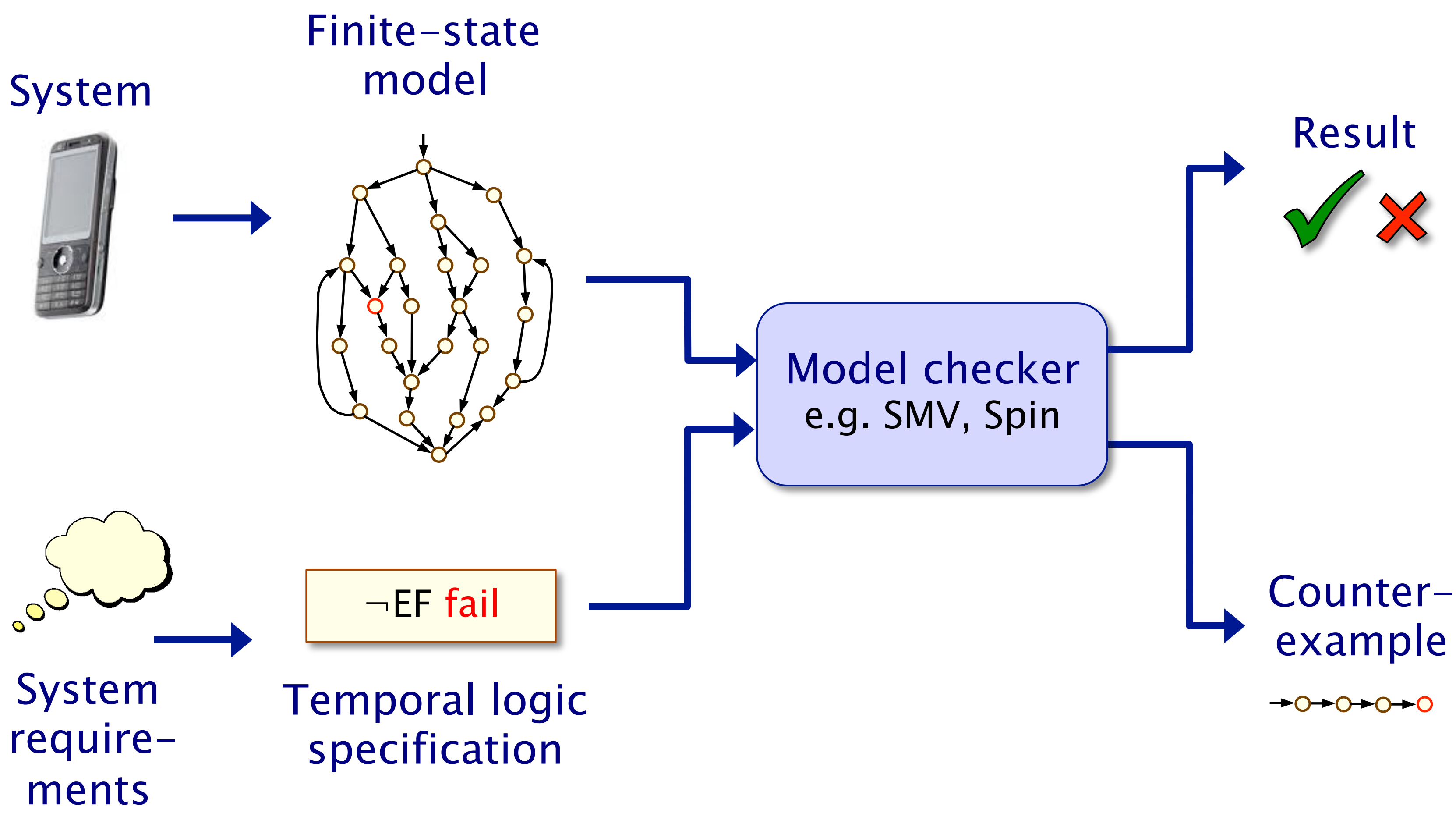
ETR'13, Toulouse, August 2013

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

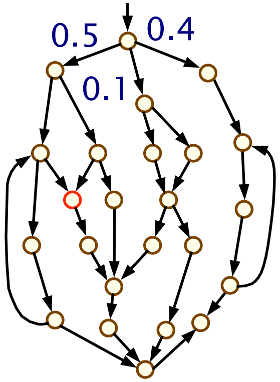
Model checking



Probabilistic model checking

System

Probabilistic model
e.g. Markov chain



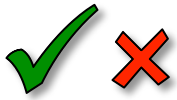
$P_{<0.1}$ [F fail]

System require–ments

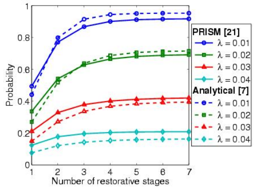
Probabilistic temporal logic specification
e.g. PCTL, CSL, LTL

Probabilistic model checker
e.g. PRISM

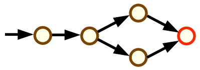
Result



Quantitative results



Counter–example



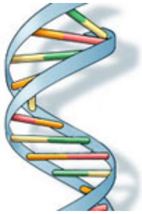
Why probability?

- 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

	Fully probabilistic	Nondeterministic
Discrete time	Discrete-time Markov chains (DTMCs)	Markov decision processes (MDPs) (probabilistic automata)
Continuous time	Continuous-time Markov chains (CTMCs)	Probabilistic timed automata (PTAs)
		CTMDPs / IMCs / …

Probabilistic models

	Fully probabilistic	Nondeterministic
Discrete time	Discrete-time Markov chains (DTMCs)	Markov decision processes (MDPs) (probabilistic automata)
Continuous time	Continuous-time Markov chains (CTMCs)	Probabilistic timed automata (PTAs) CTMDPs/IMCs/…

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/

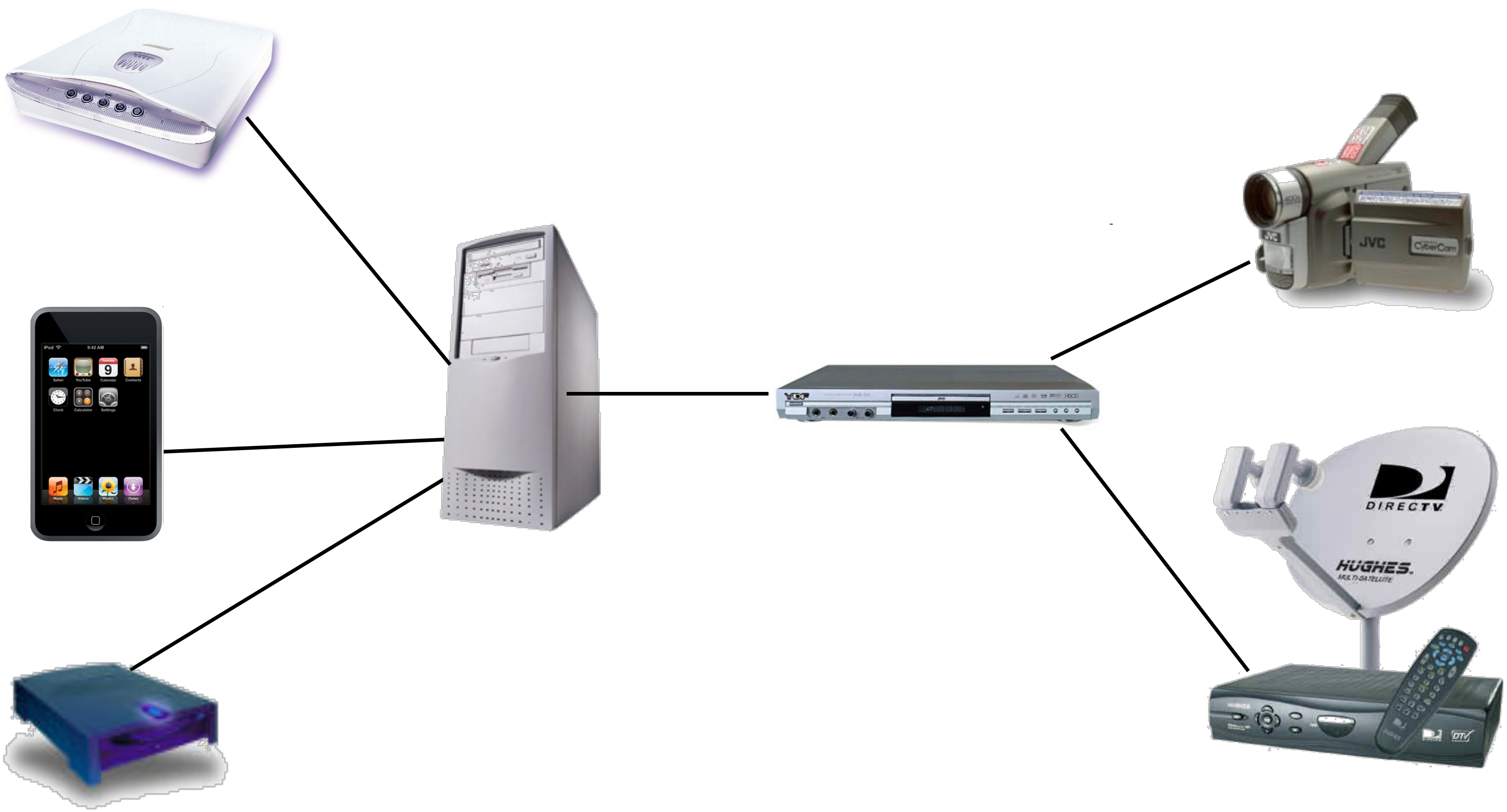
Contents

- 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

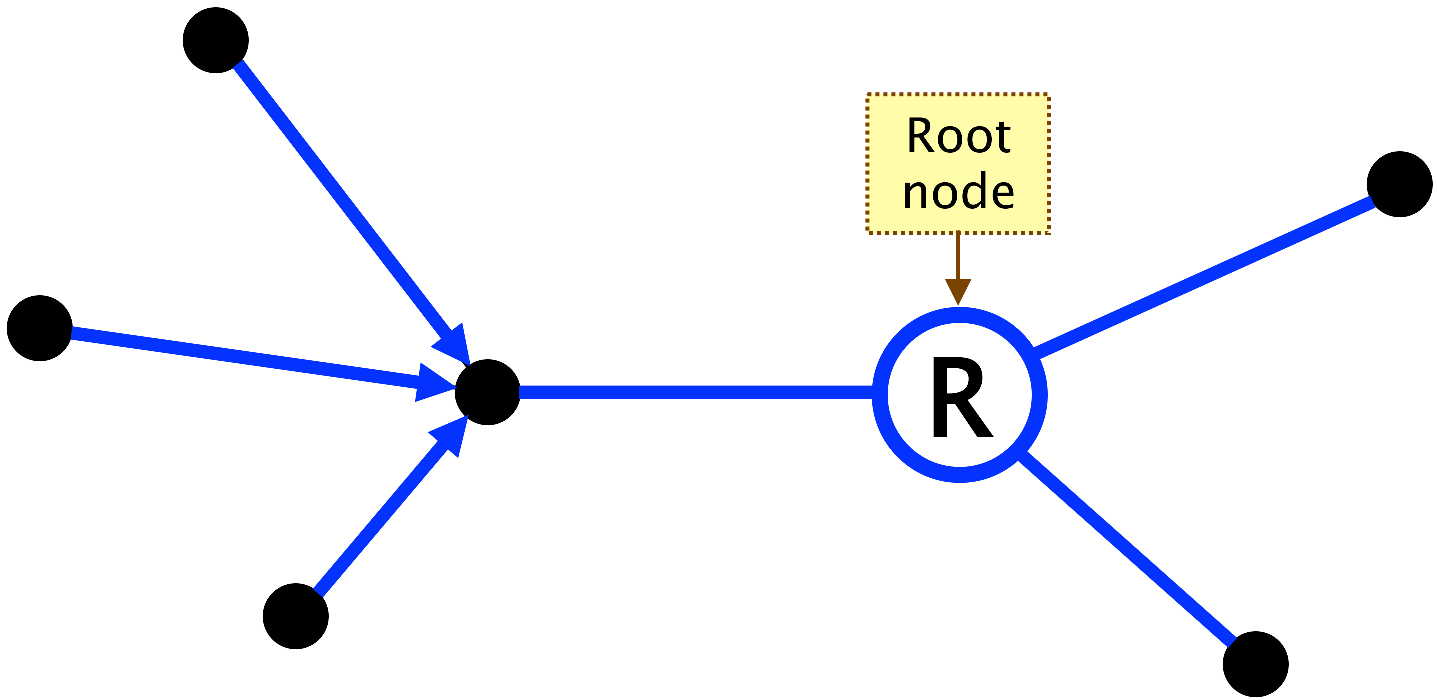
Case study: FireWire protocol

- 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

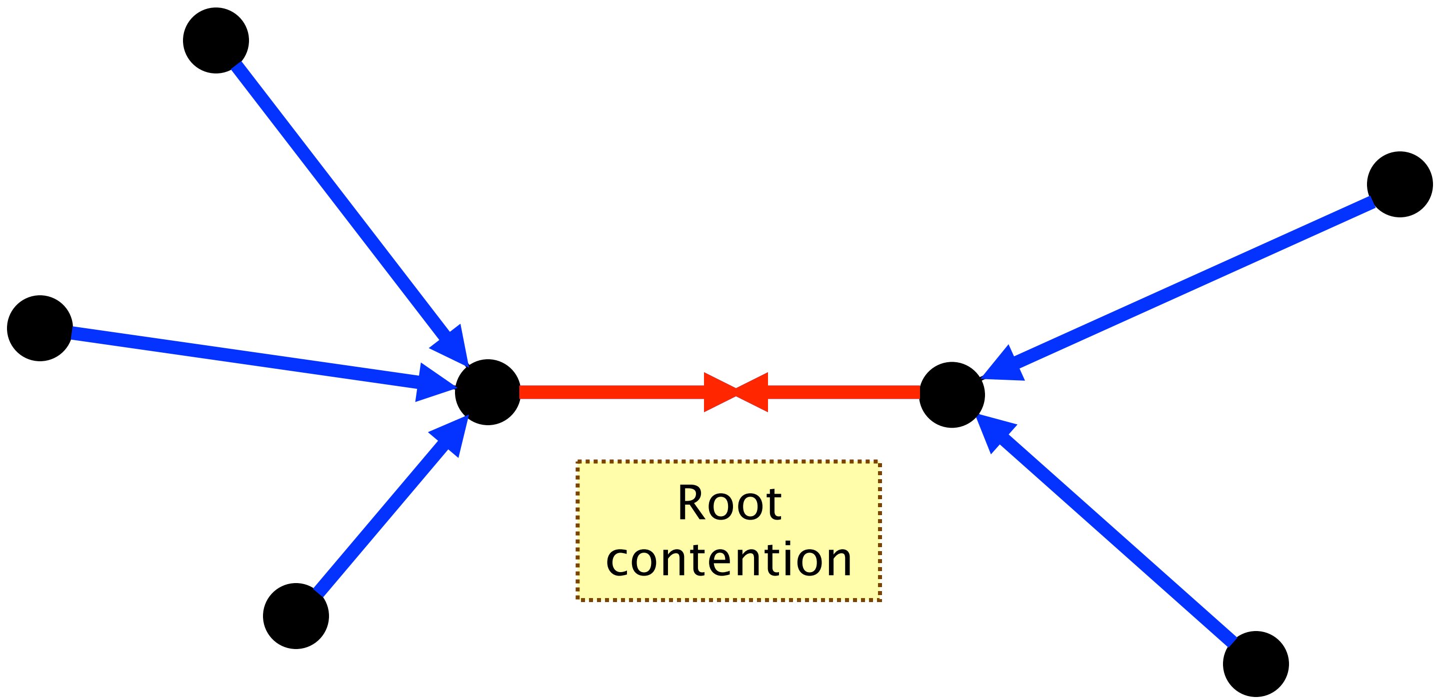
FireWire example



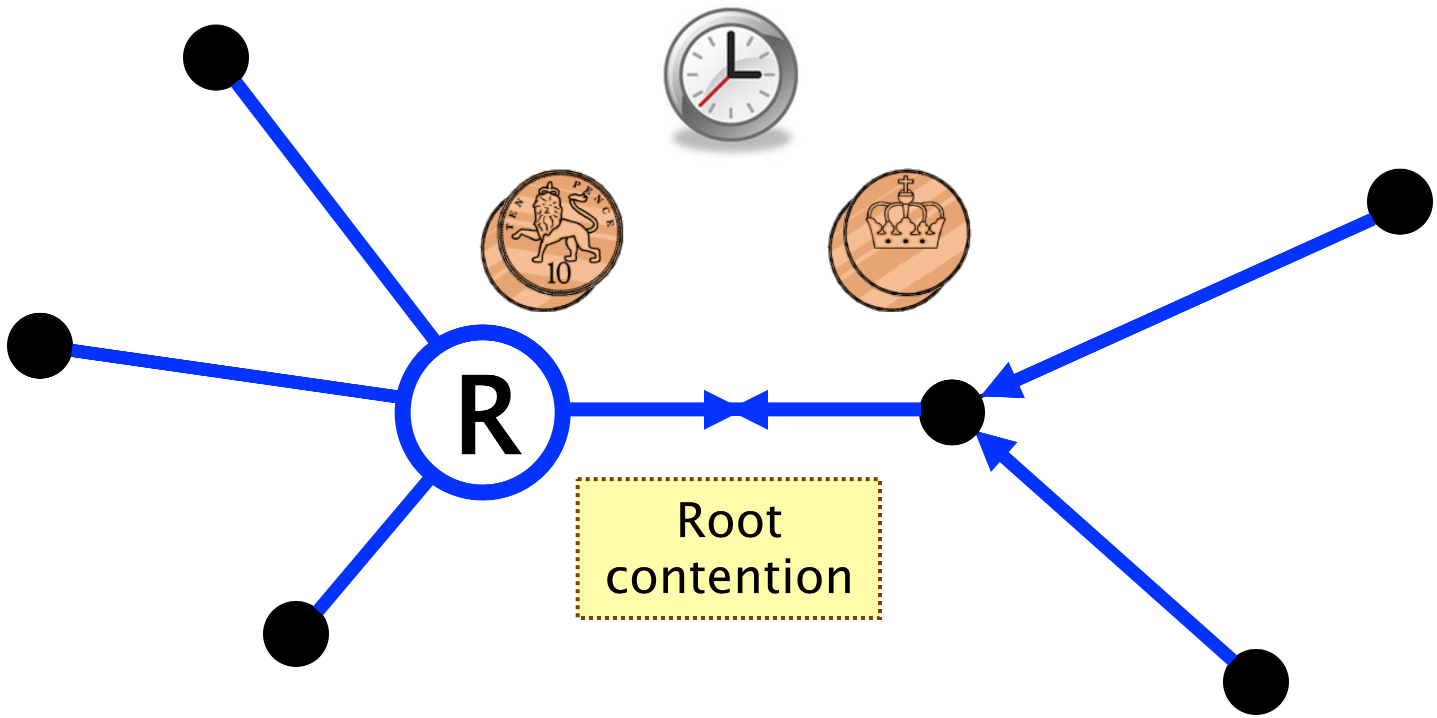
FireWire leader election



FireWire root contention

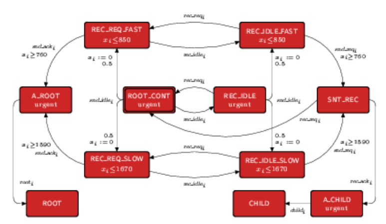


FireWire root contention

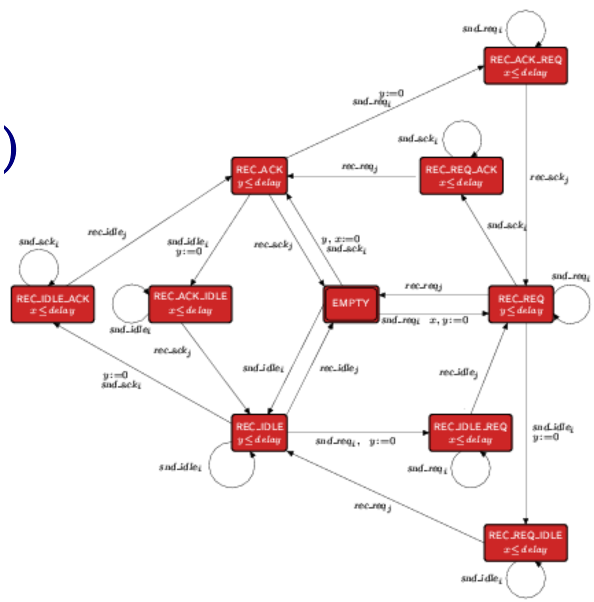


FireWire analysis

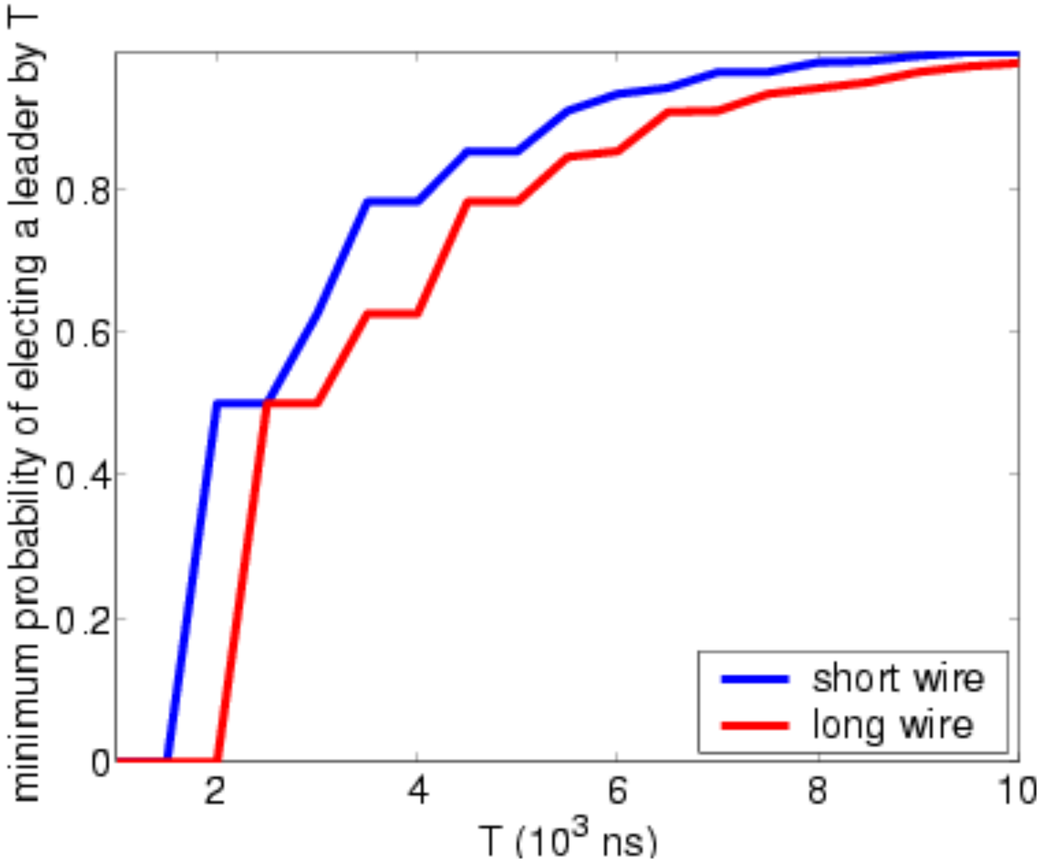
- Detailed probabilistic model:
  - probabilistic timed automaton (PTA), including:
    - concurrency: messages between nodes and wires
    - timing delays taken from official standard
    - underspecification of delays (upper/lower bounds)
  - maximum model size: 170 million states



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

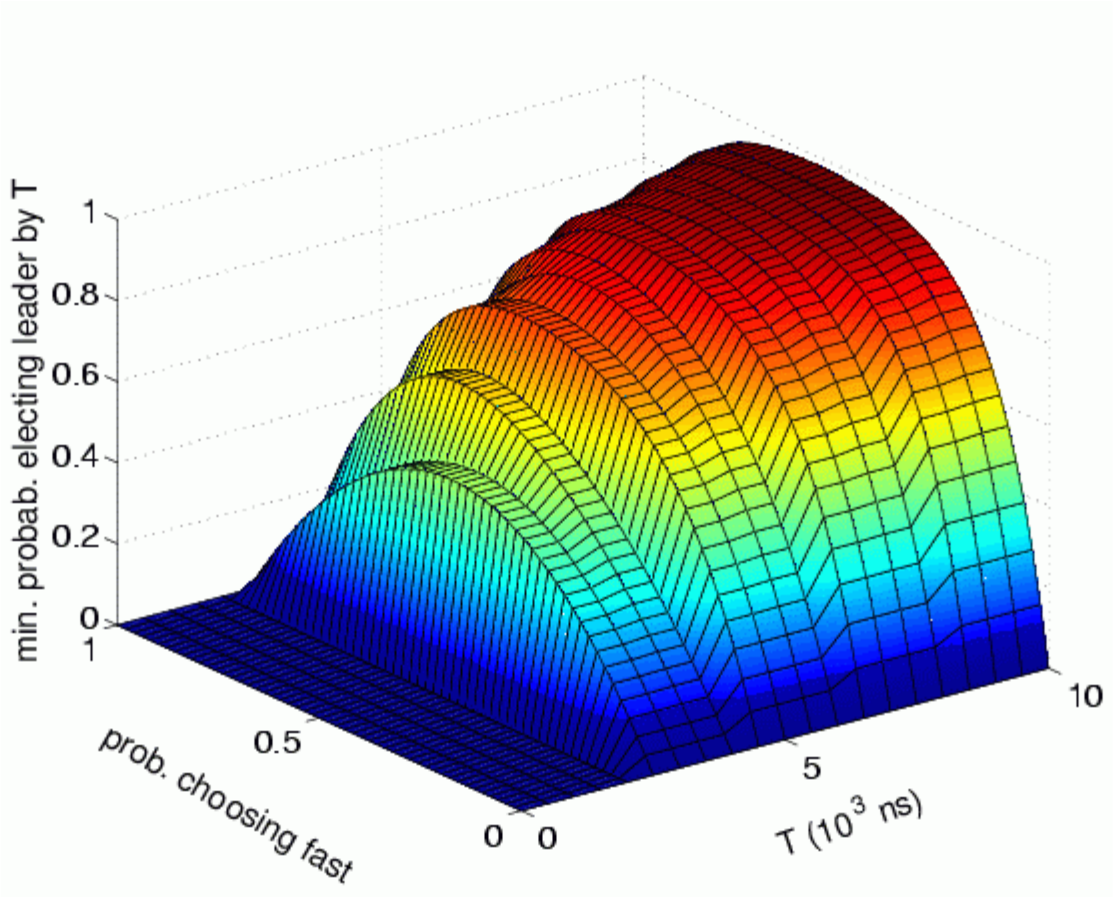


FireWire: Analysis results



"minimum probability
of electing leader
by time T"

FireWire: Analysis results

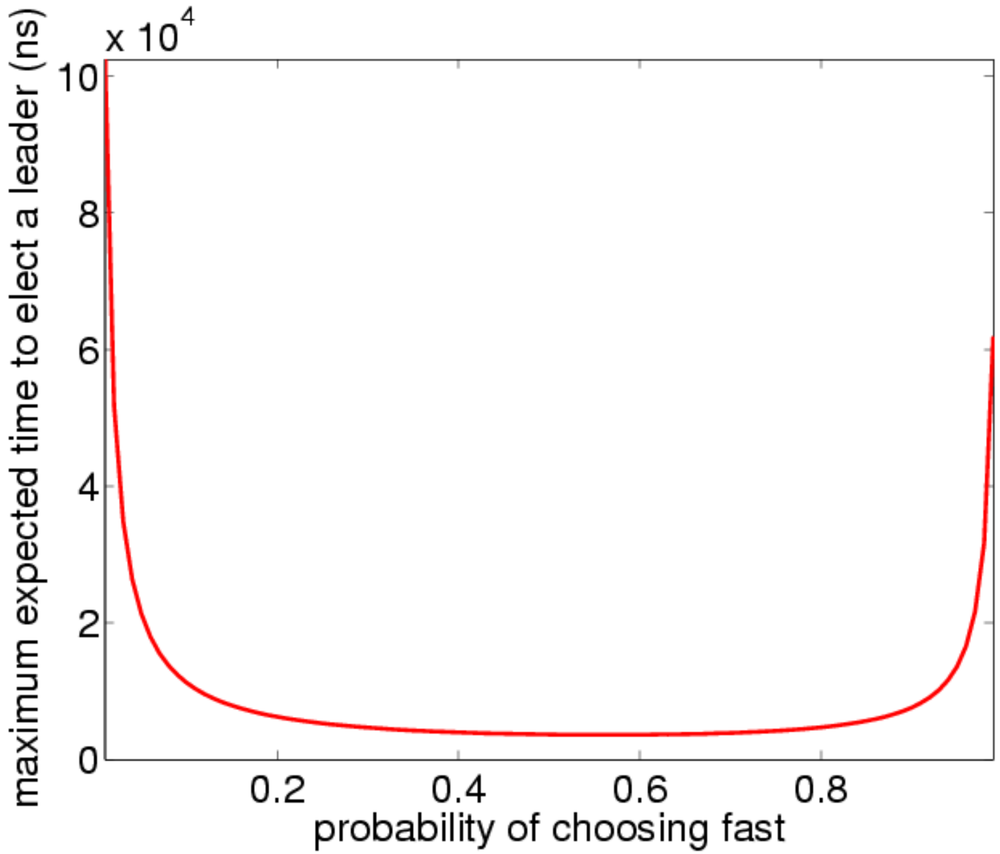


"minimum probability of electing leader by time T"

(short wire length)

Using a biased coin

FireWire: Analysis results

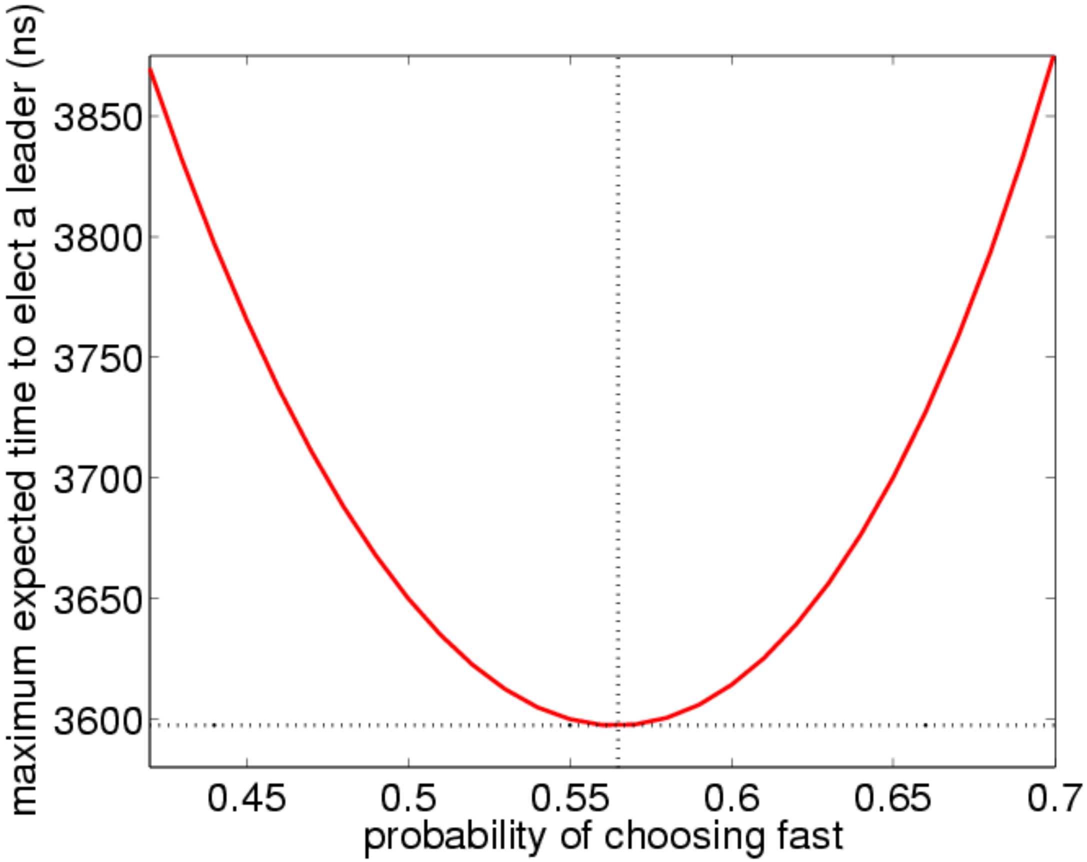


"maximum expected time to elect a leader"

(short wire length)

Using a biased coin

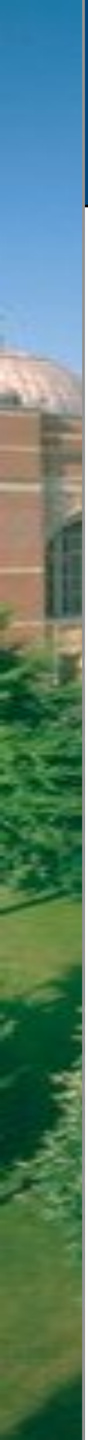
FireWire: Analysis results



"maximum expected time to elect a leader"

(short wire length)

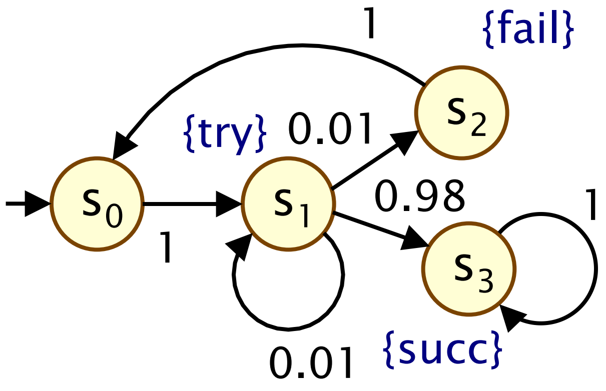
Using a biased coin is beneficial!



Contents

Discrete-time Markov chains (DTMCs)

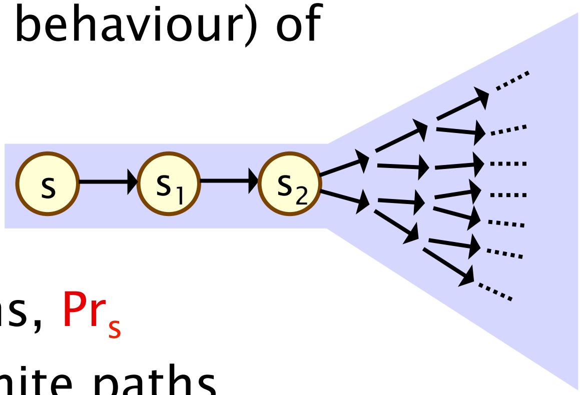
- Discrete-time Markov chains (DTMCs)
  - state-transition systems augmented with probabilities
- States
  - discrete set of states representing all possible configurations of the system being modelled
- Transitions
  - transitions between states occur in discrete time-steps
- Probabilities
  - probability of making transitions between states is given by discrete probability distributions



Discrete-time Markov chains

- Formally, a DTMC D is a tuple $(S, s_{init}, \mathbf{P}, L)$ where:
  - S is a finite set of states ("state space")
  - $s_{init} \in S$ is the initial state
  - $\mathbf{P} : S \times S \rightarrow [0,1]$ is the transition probability matrix
  - $L : S \rightarrow 2^{AP}$ 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 $\mathbf{P}(s_i, s_{i+1}) > 0 \ \forall i$
  - 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 ϕ:

ψ is true with probability ~p

  - ϕ ::= $true \mid a \mid \phi \wedge \phi \mid \neg\phi \mid P_{\sim p} [\, \psi \,]$ (state formulae)
  - ψ ::= $X\,\phi \mid \phi\, U^{\leq k}\, \phi \mid \phi\, U\, \phi$ (path formulae)

"next" "bounded until" "until"

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

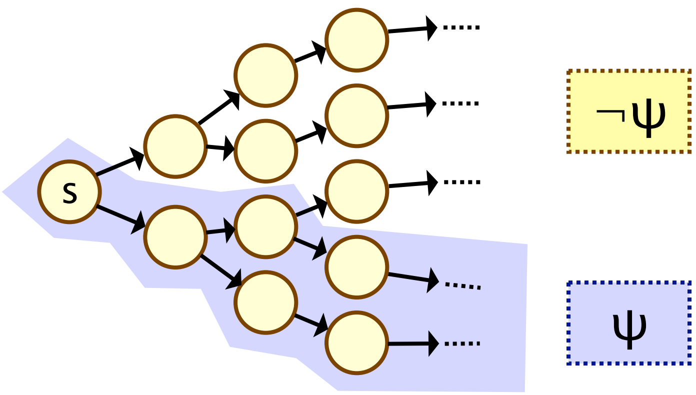
- Can derive other useful operators
  - logical: $false$, $\phi_1 \vee \phi_2$, $\phi_1 \rightarrow \phi_2$
  - $F\,\phi \equiv true\ U\,\phi$ ("eventually") and $G\,\phi \equiv \neg(F\,\neg\phi)$ ("always")
  - bounded variants, e.g. $F^{\leq k}\,\phi \equiv true\ U^{\leq k}\,\phi$

PCTL semantics (for DTMCs)

- PCTL formulae interpreted over states of a DTMC
  - $s \vDash \phi$ denotes ϕ is "true in state s" or "satisfied in state s"

- Semantics of logical operators: standard meanings

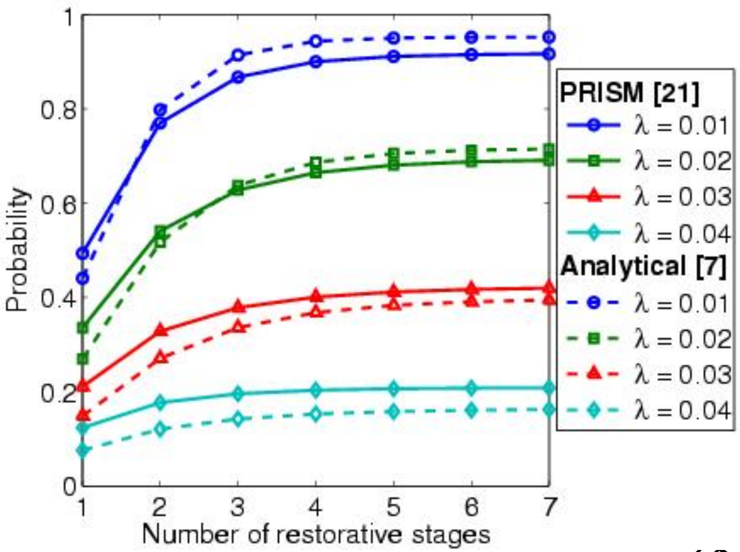
- Semantics of the probabilistic operator P
  - informally, $s \vDash P_{\sim p} [\psi]$ means: "the probability, from state s, that ψ is true for outgoing paths satisfies the bound $\sim p$"
  - formally: $s \vDash P_{\sim p} [\psi] \Leftrightarrow Prob(s, \psi) \sim p$
  - where: $Prob(s, \psi) = Pr_s \{ \omega \in Path(s) \mid \omega \vDash \psi \}$



Quantitative (numerical) properties

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

- Example
  - $P_{=?}$ [F 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 err/total>0.1]
  - "what is the probability that 10% of the NAND gate outputs are erroneous?"
- Bluetooth wireless communication protocol
  - $P_{=?}$ [$F^{\leq t}$ 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 (pairs_a=0 & pairs_b>0)]
  - "what is the probability that the party B gains an unfair advantage during the execution of the protocol?"

reliability

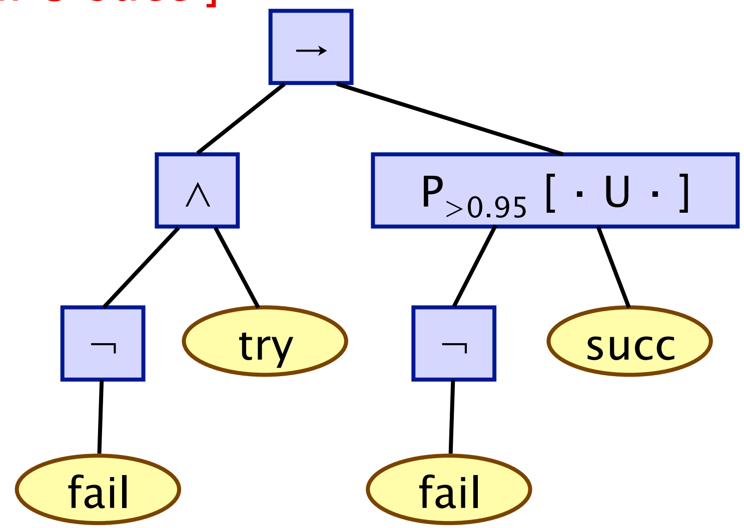
performance

fairness

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: $Sat(\phi) = \{ s \in S \mid s \vDash \phi \}$ = set of states satisfying ϕ
  - or: compute result of e.g. $P_{=?}$ [$F^{\leq k}$ error]

- Basic algorithm proceeds by induction on parse tree of ϕ
  - e.g. $\phi = (\neg fail \wedge try) \rightarrow P_{>0.95}$ [$\neg fail$ U $succ$]
  - logical operators: straightforward
- For the $P_{\sim p}$ [ψ] operator
  - need to compute probabilities $Prob(s, \psi)$ for all states $s \in S$
  - combination of graph algorithms and numerical computation
- Linear in $|\Phi|$ and polynomial in $|S|$



PCTL model checking: Until

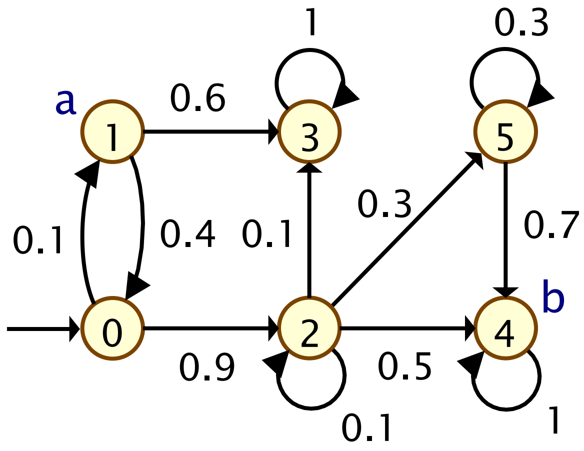
- Example: computation of probabilities for "until" formula
  - i.e. Prob(s, ϕ_1 U ϕ_2) for all s $\in$ S
- First, execute graph-based analysis to identify all states where the probability is exactly 1 or 0:
  - S^{yes} = Sat($P_{\geq 1}$ [ϕ_1 U ϕ_2])
  - S^{no} = Sat($P_{\leq 0}$ [ϕ_1 U ϕ_2])
- Then, solve linear equation system for remaining states:

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

  - solved with standard methods, e.g. Gaussian elimination (iterative numerical methods preferred in practice)

PCTL until – Example

- Example: $P_{>0.8}$ [¬a U b]

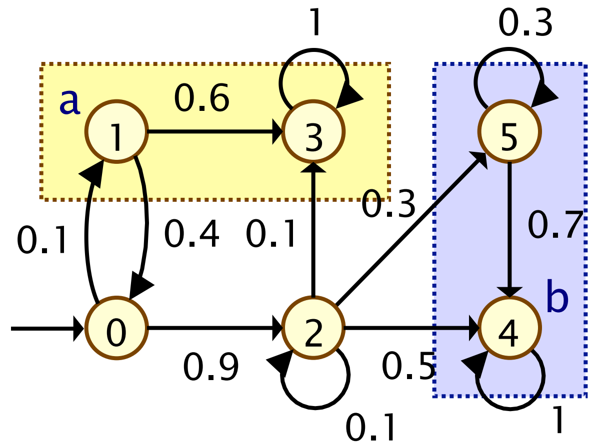


PCTL until – Example

- Example: $P_{>0.8}$ [¬a U b]

S^{no} =

Sat($P_{\leq 0}$ [¬a U b])



S^{yes} =

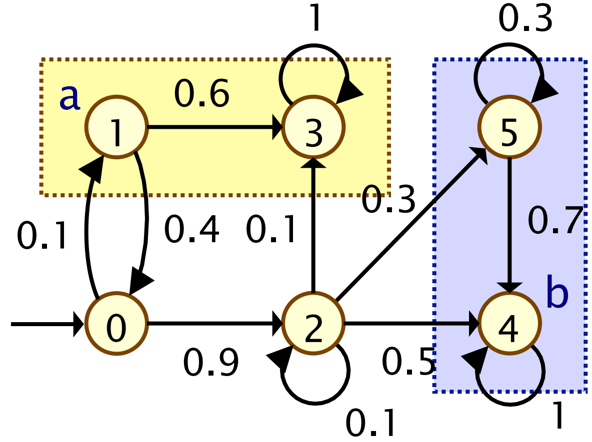
Sat($P_{\geq 1}$ [¬a U b])

PCTL until – Example

- Example: $P_{>0.8}$ [¬a U b]
- Let x_s = Prob(s, ¬a U b)
- Solve:

S^{no} =

Sat($P_{\leq 0}$ [¬a U b])



S^{yes} =

Sat($P_{\geq 1}$ [¬a U b])

$x_4 = x_5 = 1$

$x_1 = x_3 = 0$

$x_0 = 0.1x_1+0.9x_2 = 0.8$

$x_2 = 0.1x_2+0.1x_3+0.3x_5+0.5x_4 = 8/9$

Prob(¬a U b) = $\underline{x}$ = [0.8, 0, 8/9, 0, 1, 1]

Sat($P_{>0.8}$ [¬a U b]) = { 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 tmp_fail_1) $\wedge$ (F tmp_fail_2)] – "both servers eventually fail with probability at most 0.01"
  - $P_{\geq 1}$ [G F ready] – "with probability 1, the server always eventually returns to a ready-state"
  - $P_{=?}$ [F G 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:
  - $R^{energy}_{\leq 400}$ [$C^{\leq 60}$] – “the expected energy consumption over 60 seconds is at most 40 J”
  - $R^{time}_{=?}$ [F 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

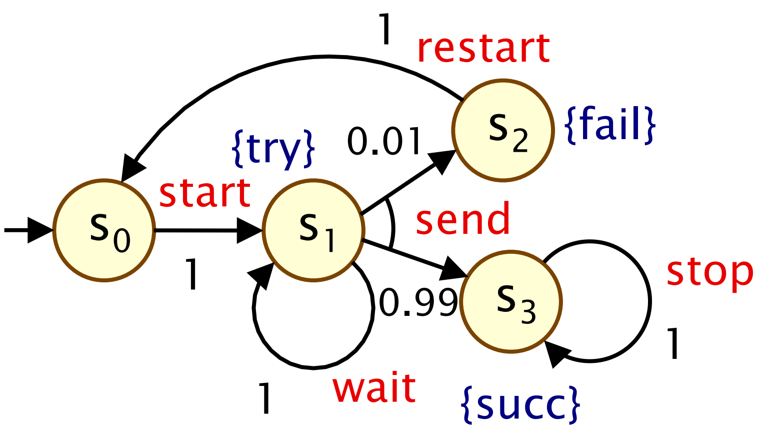
- Underspecification and abstraction
  - e.g. a probabilistic communication protocol designed for message propagation delays of between d_{min} and d_{max}

Markov decision processes (MDPs)

- 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: $\delta : S \times Act \rightarrow Dist(S)$
  - instead of $P : S \times S \rightarrow [0,1]$

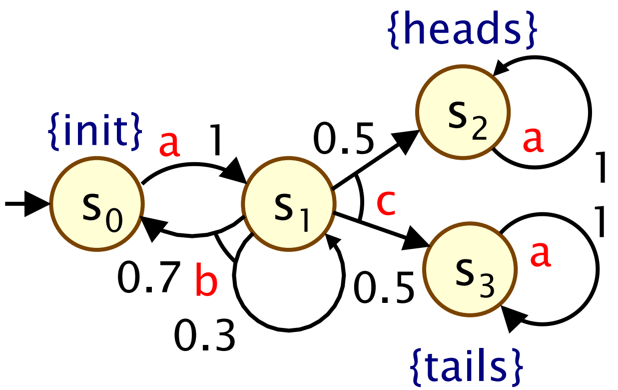


Adversaries

- 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 σ of an MDP is a function mapping every finite path $s_0 a_0 s_1 a_1 \ldots s_n$ to an action available in s_n
- Adversary σ induces a probability measure Pr_s^σ over paths
  - via construction of an (infinite-state) DTMC

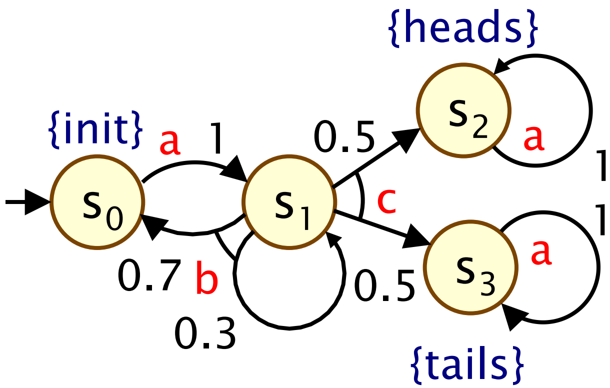
Adversaries – Examples

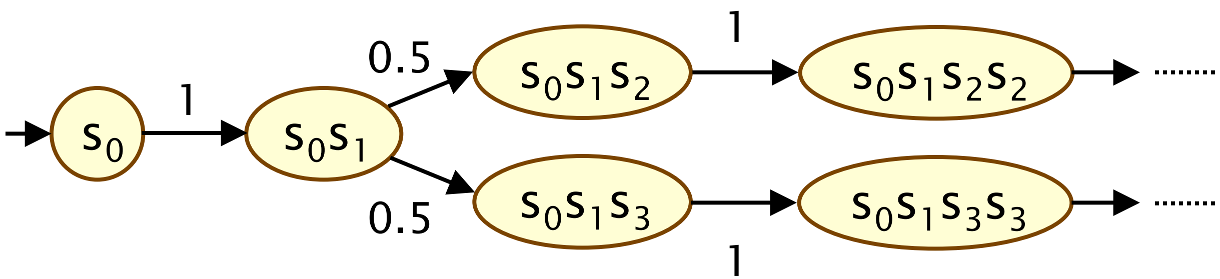
- Consider the simple MDP below
  - s_1 is the only state for which an adversary makes a choice
- Adversary σ_1
  - picks action c the first time
  - $\sigma_1(s_0 s_1)=c$
- Adversary σ_2
  - picks action b the first time, then c
  - $\sigma_2(s_0 s_1)=b$, $\sigma_2(s_0 s_1 s_1)=c$, $\sigma_2(s_0 s_1 s_0 s_1)=c$



Adversaries – Examples

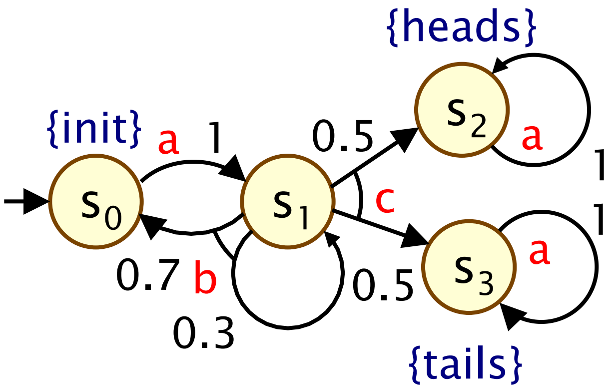
- Fragment of DTMC for adversary σ_1
  - σ_1 picks action c the first time

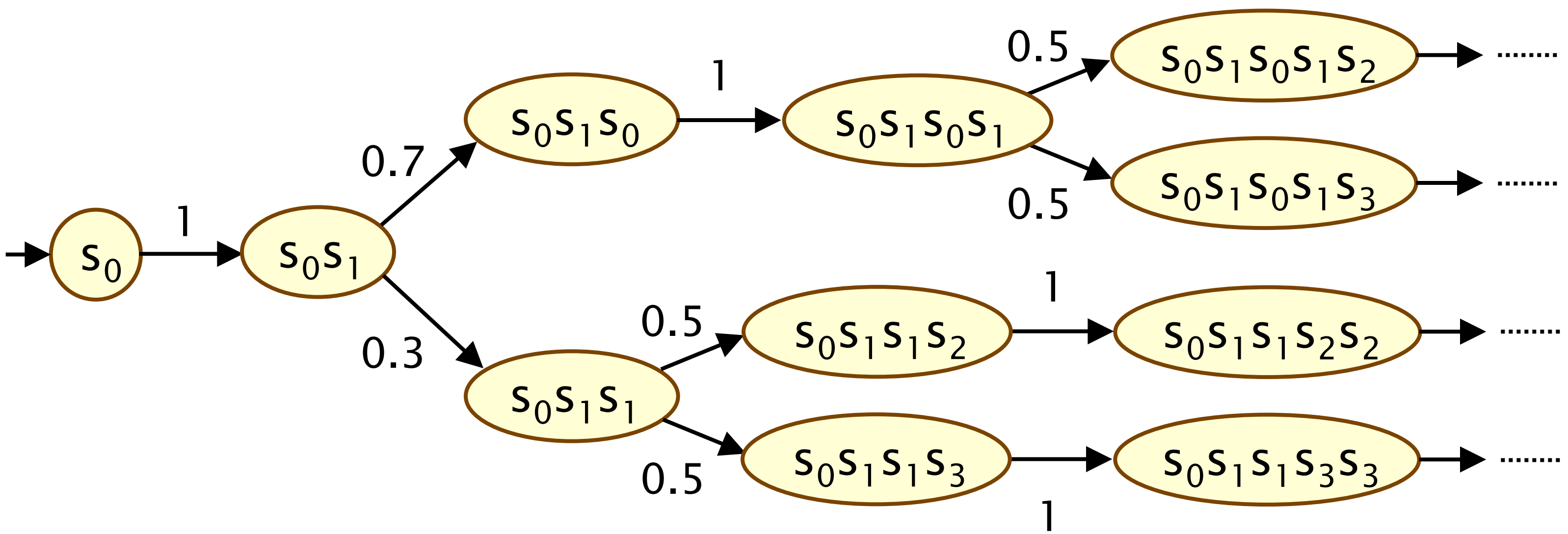




Adversaries – Examples

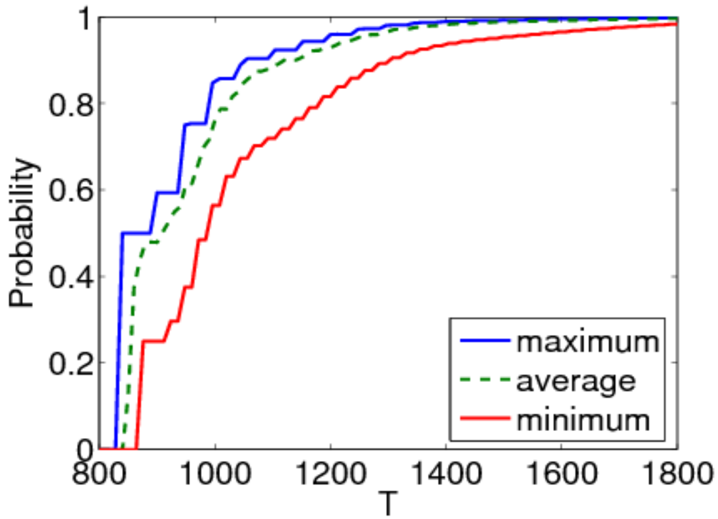
- Fragment of DTMC for adversary σ_2
  - σ_2 picks action b, then c





Model checking for MDPs

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



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

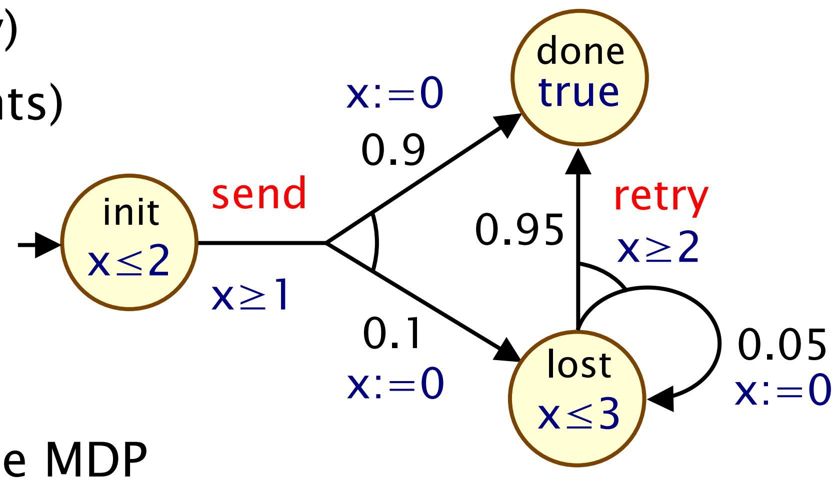
Probabilistic real-time systems

- 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

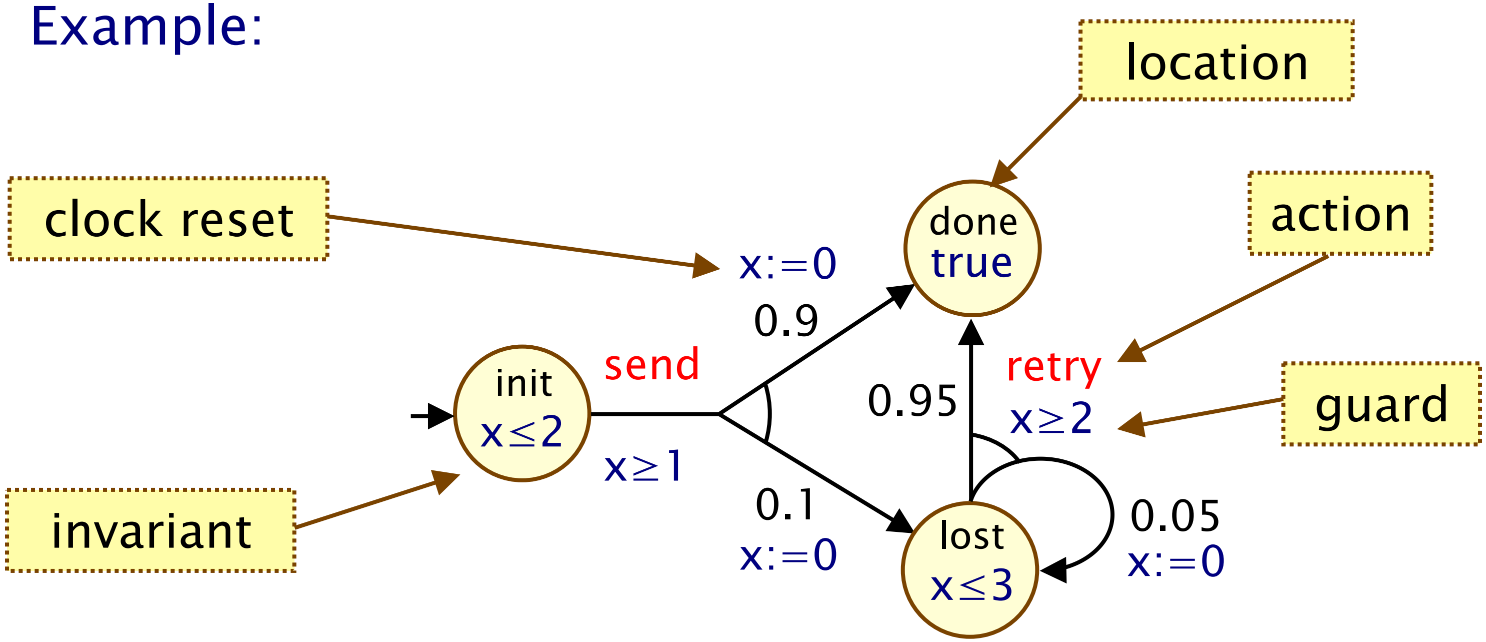
- PTAs comprise:
  - clocks (increase simultaneously)
  - locations (labelled with invariants)
  - transitions (action + guard + probabilities + resets)



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

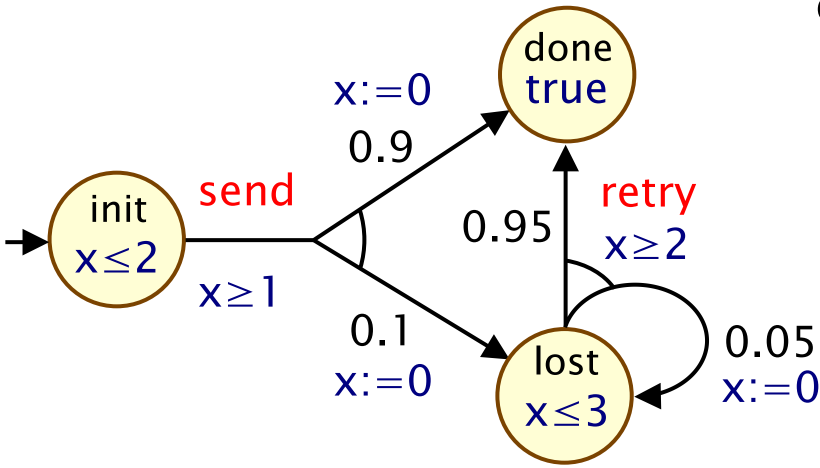
PTA – Example

- Example:

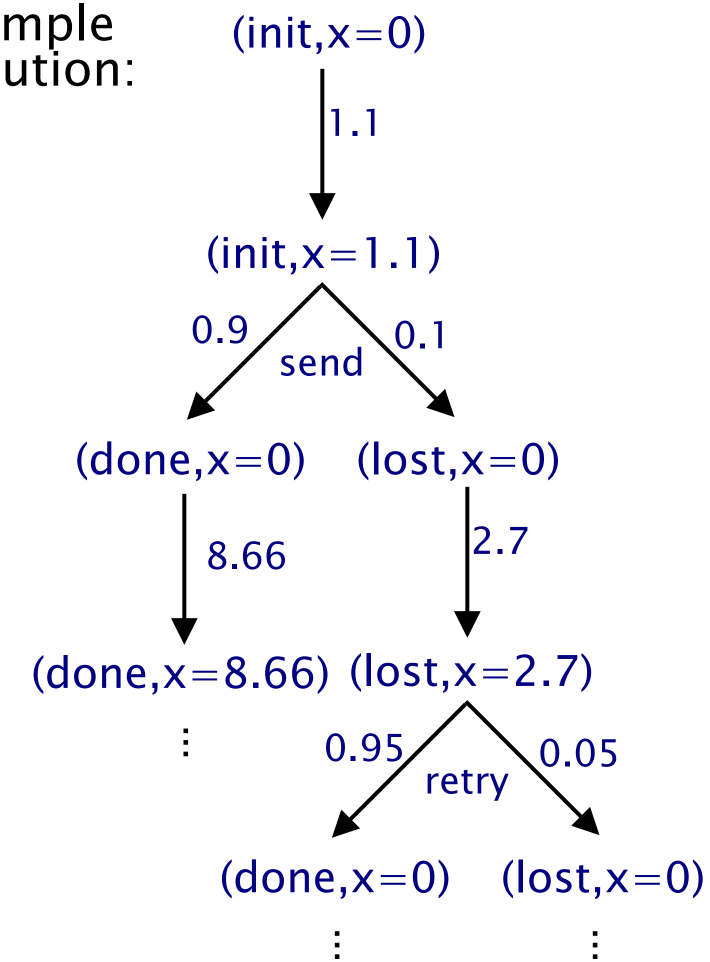


PTA – Example execution

PTA:



Example execution:



Properties of PTAs

- Temporal logic
  - again, can use PCTL to represent properties
  - e.g. $P_{\geq 0.99}$ [$F^{\leq 5}$ 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]

Contents

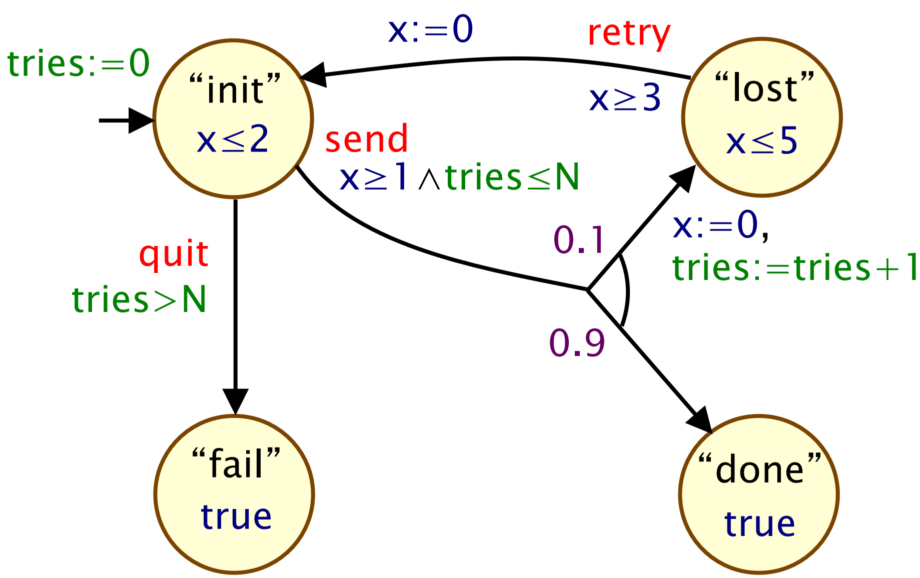
- Case study: the FireWire protocol
- Discrete-time Markov chains + the logic PCTL
- Adding nondeterminism: Markov decision processes
- Adding real time: probabilistic timed automata
- Probabilistic model checking in practice: PRISM

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/

Modelling PTAs in PRISM

- 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

```
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

```
pta
const int N;
module transmitter
    s : [0..3] init 0;
    tries : [0..N+1] init 0;
    x : clock;
    invariant (s=0 ⇒ x≤2) & (s=1 ⇒ x≤5) endinvariant
    [send] s=0 & tries≤N & x≥1
          → 0.9 : (s'=3)
          + 0.1 : (s'=1) & (tries'=tries+1) & (x'=0);
    [retry] s=1 & x≥3 → (s' =0) & (x' =0);
    [quit]  s=0 & tries>N → (s' =2);
endmodule
rewards "energy" (s=0) : 2.5; endrewards
```

Basic ingredients:

- modules
- variables
- commands

Modelling PTAs in PRISM

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

```
pta
const int N;
module transmitter
    s : [0..3] init 0;
    tries : [0..N+1] init 0;
    x : clock;
    invariant (s=0 ⇒ x≤2) & (s=1 ⇒ x≤5) endinvariant
    [send] s=0 & tries≤N & x≥1
          → 0.9 : (s'=3)
          + 0.1 : (s'=1) & (tries'=tries+1) & (x'=0);
    [retry] s=1 & x≥3 → (s' =0) & (x' =0);
    [quit]  s=0 & tries>N → (s' =2);
endmodule
rewards "energy" (s=0) : 2.5; endrewards
```

Basic ingredients:

- modules
- variables
- commands

For PTAs:

- clocks
- invariants
- guards/resets

Modelling PTAs in PRISM

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

```
pta
const int N;
module transmitter
    s : [0..3] init 0;
    tries : [0..N+1] init 0;
    x : clock;
    invariant (s=0 ⇒ x≤2) & (s=1 ⇒ x≤5) endinvariant
    [send] s=0 & tries≤N & x≥1
        → 0.9 : (s'=3)
        + 0.1 : (s'=1) & (tries'=tries+1) & (x'=0);
    [retry] s=1 & x≥3 → (s' =0) & (x' =0);
    [quit]  s=0 & tries>N → (s' =2);
endmodule
rewards "energy" (s=0) : 2.5; endrewards
```

Basic ingredients:

- modules
- variables
- commands

For PTAs:

- clocks
- invariants
- guards/resets

Also:

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

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/)
  - demonstrations available

Questions ?

More info here:
www.prismmodelchecker.org