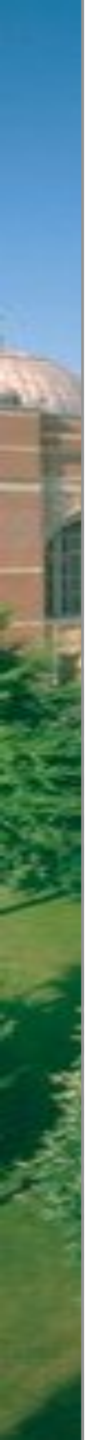


Automated Verification of Probabilistic Real-time Systems

Dave Parker

University of Birmingham

MOVEP'14 Summer School, Nantes, July 2014

Overview

- Probabilistic model checking
  - example: FireWire protocol
- Probabilistic timed automata (PTAs)
  - clocks, zones, syntax, semantics
  - property specification
- Verification techniques for PTAs
  - region graphs + digital clocks + zone-based methods
  - abstraction-refinement
- Tool support: PRISM
- Verification vs. controller synthesis
  - example: task-graph scheduling

- See: www.prismmodelchecker.org/lectures/movep14/
  - slides, tutorial papers, reference list, …

Probabilistic model checking

Probabilistic model checking

System

High-level model/design

module A

a : [0..N] **init** N;

ab : [0..N] **init** 0;

[r_1] a>0 → k_1*a : (a' =a−1)&(ab' =ab+1);

[r_2] ab>0 → k_2*ab : (a'=a+1)&(ab'=ab−1);

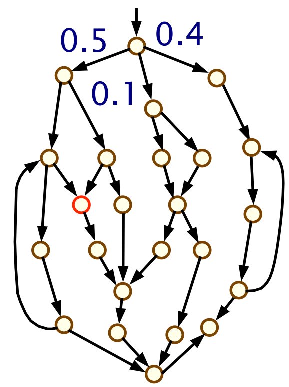
[r_3] a>0 → k_3*a : (a'=a−1);

System require-ments

$P_{<0.1}$ [F fail]

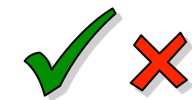
Specification (temporal logic)

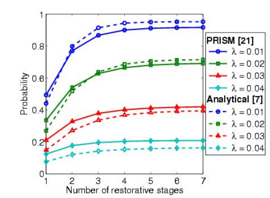
Probabilistic model checker



Low-level model (states, transitions)

Verification results





Numerical results

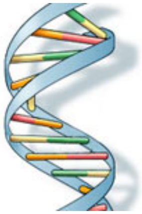
Reminder: Why probability?

- 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

- Systems with probability, nondeterminism and real-time
  - e.g. wireless communication protocols
  - e.g. randomised security protocols

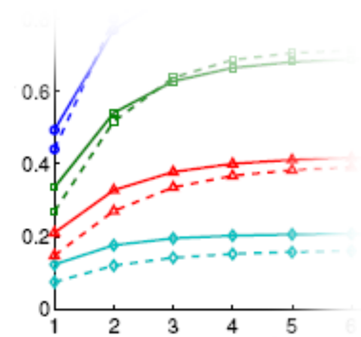
- Randomised back-off schemes
  - Ethernet, WiFi (802.11), Zigbee (802.15.4)
- Random choice of waiting time
  - Bluetooth device discovery phase
  - Root contention in IEEE 1394 FireWire
- Random choice over a set of possible addresses
  - IPv4 dynamic configuration (link-local addressing)
- Random choice of a destination
  - Crowds anonymity, gossip-based routing

Probabilistic models

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

Verifying probabilistic systems

- Quantitative notions of correctness
  - "the probability of an airbag failing to deploy within 0.02 seconds of being triggered is at most 0.001"
  - in temporal logic: $P_{\leq 0.001}$ [$G^{\leq 0.02}$!"deploy"]
- Not just correctness
  - reliability, dependability, performance, resource usage (e.g. battery life), security, privacy, trust, anonymity, …
- Usually focus on numerical properties:
  - e.g.: $P_{=?}$ [$G^{\leq 0.02}$!"deploy"]
  - or $P_{=?}$ [$G^{\leq T}$!"deploy"] for varying T
- Combine numerical + exhaustive aspects
  - i.e. worst-case (or best-case) probabilities
  - e.g.: $P_{max=?}$ [$G^{\leq 0.02}$!"deploy"]



Overview

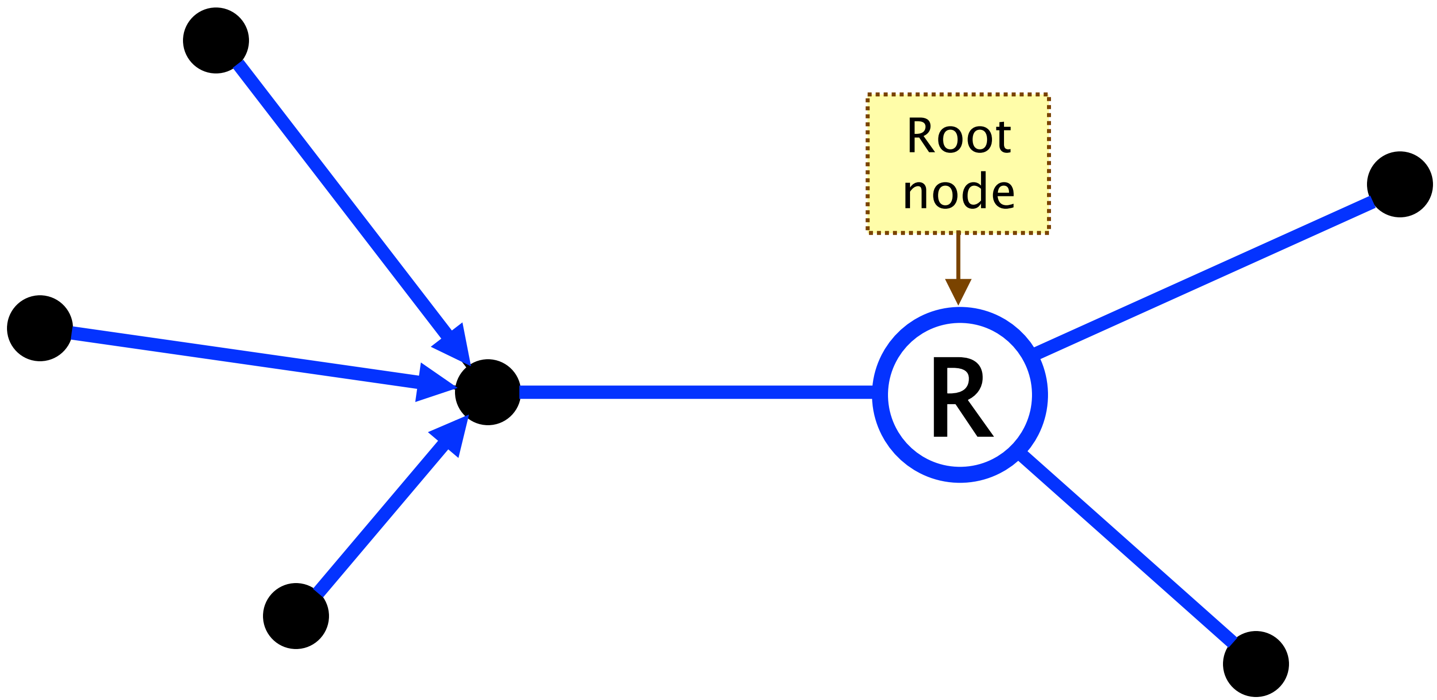
- Probabilistic model checking
  - example: FireWire protocol
- Probabilistic timed automata (PTAs)
  - clocks, zones, syntax, semantics
  - property specification
- Verification techniques for PTAs
  - region graphs + digital clocks + zone-based methods
  - abstraction-refinement
- Tool support: PRISM
- Verification vs. controller synthesis
  - example: task-graph scheduling

- See: www.prismmodelchecker.org/lectures/movep14/
  - slides, tutorial papers, reference list, …

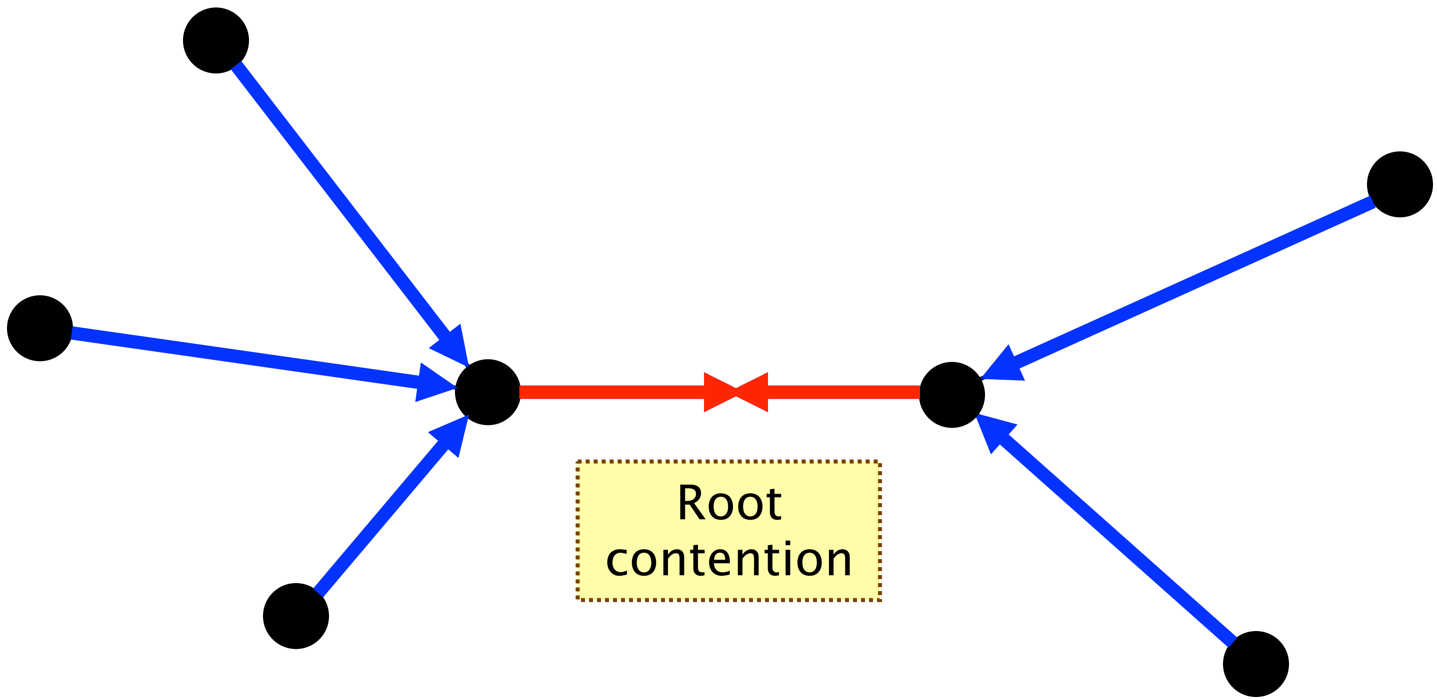
Case study: FireWire protocol

- 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

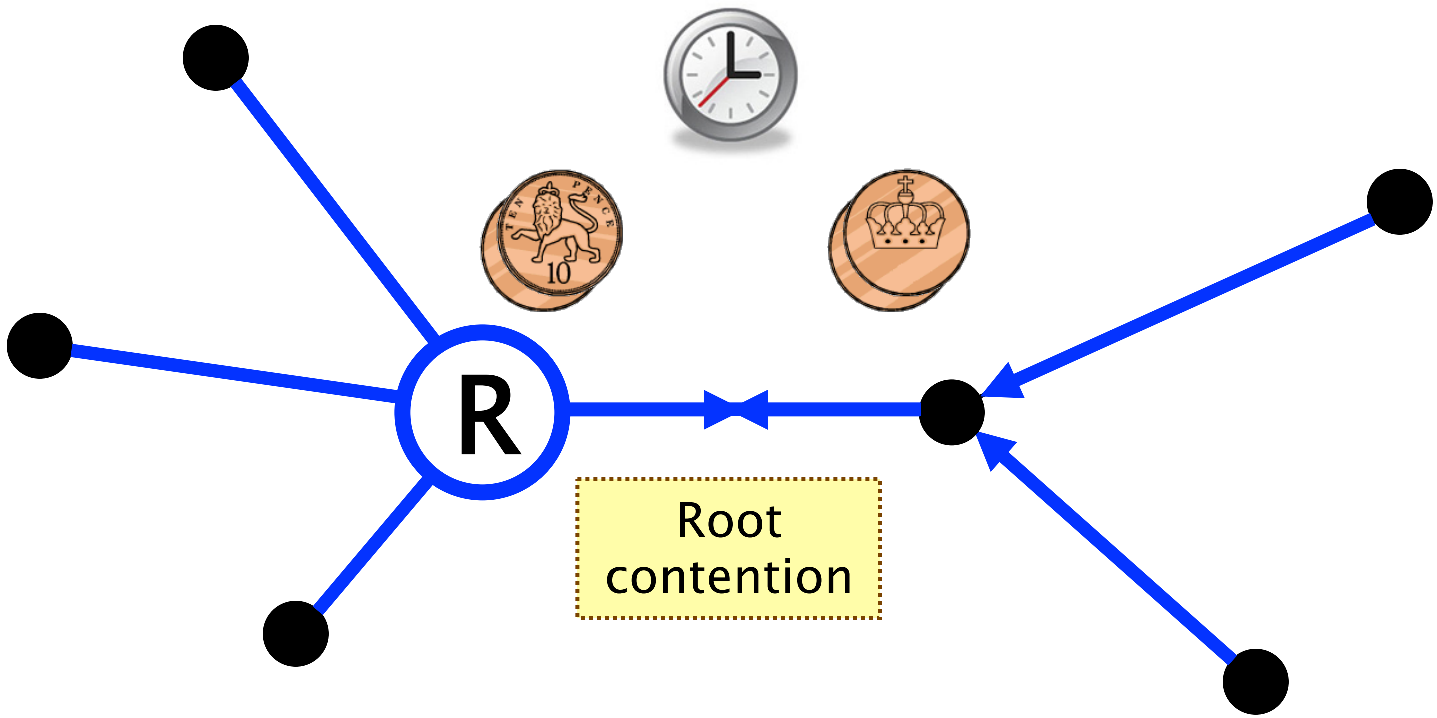
FireWire leader election



FireWire root contention

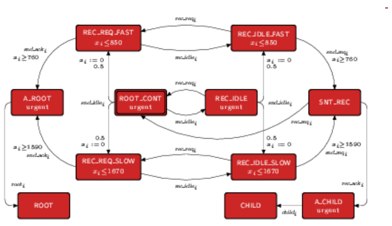


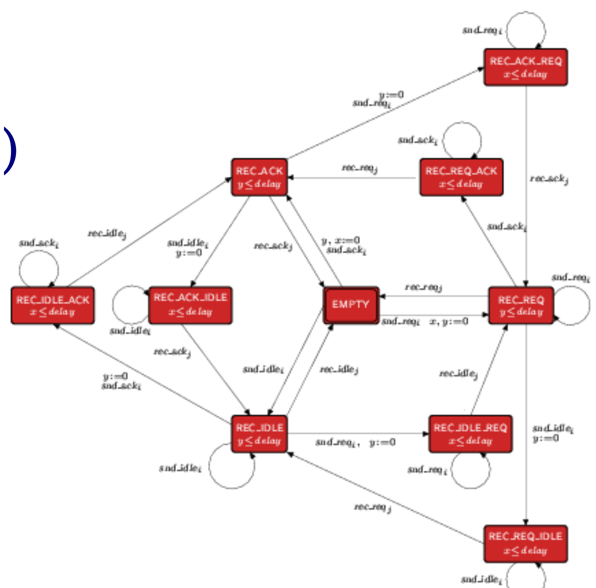
FireWire root contention



FireWire analysis

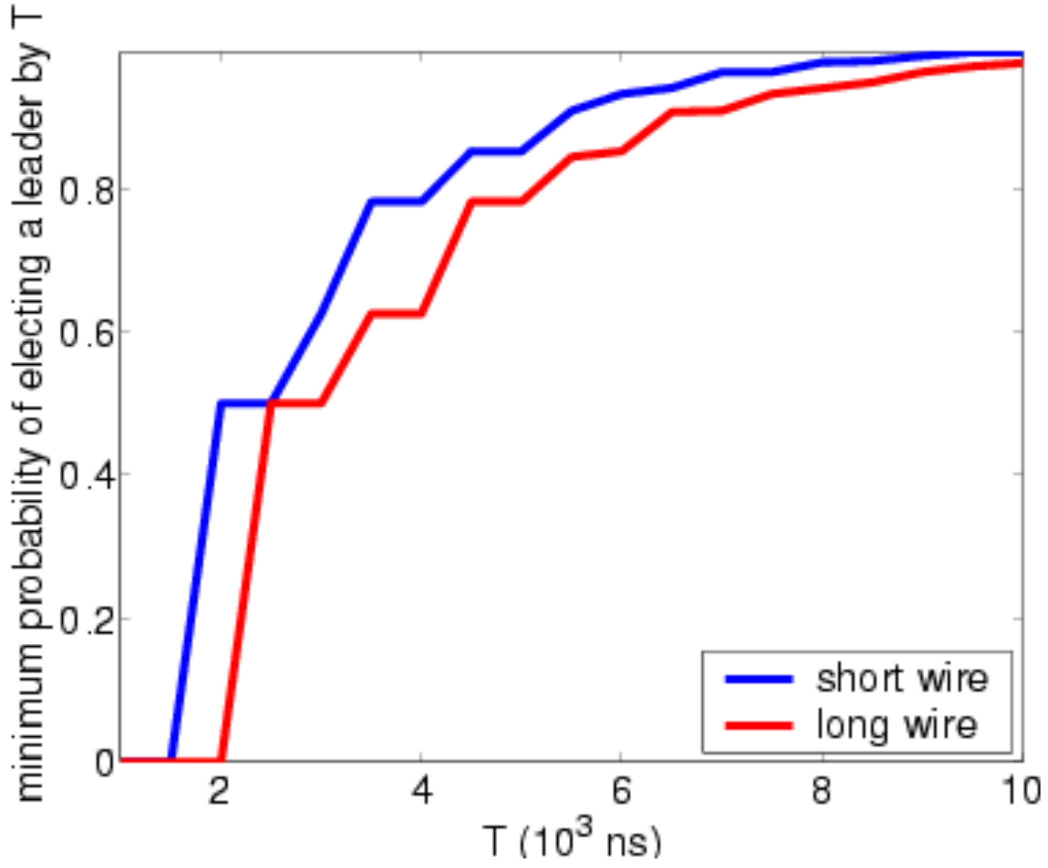
- Detailed probabilistic model:
  - probabilistic timed automaton (PTA), including:
    - concurrency: messages between nodes and wires
    - timing delays taken from official standard
    - underspecification of delays (upper/lower bounds)
  - maximum model size: 170 million states





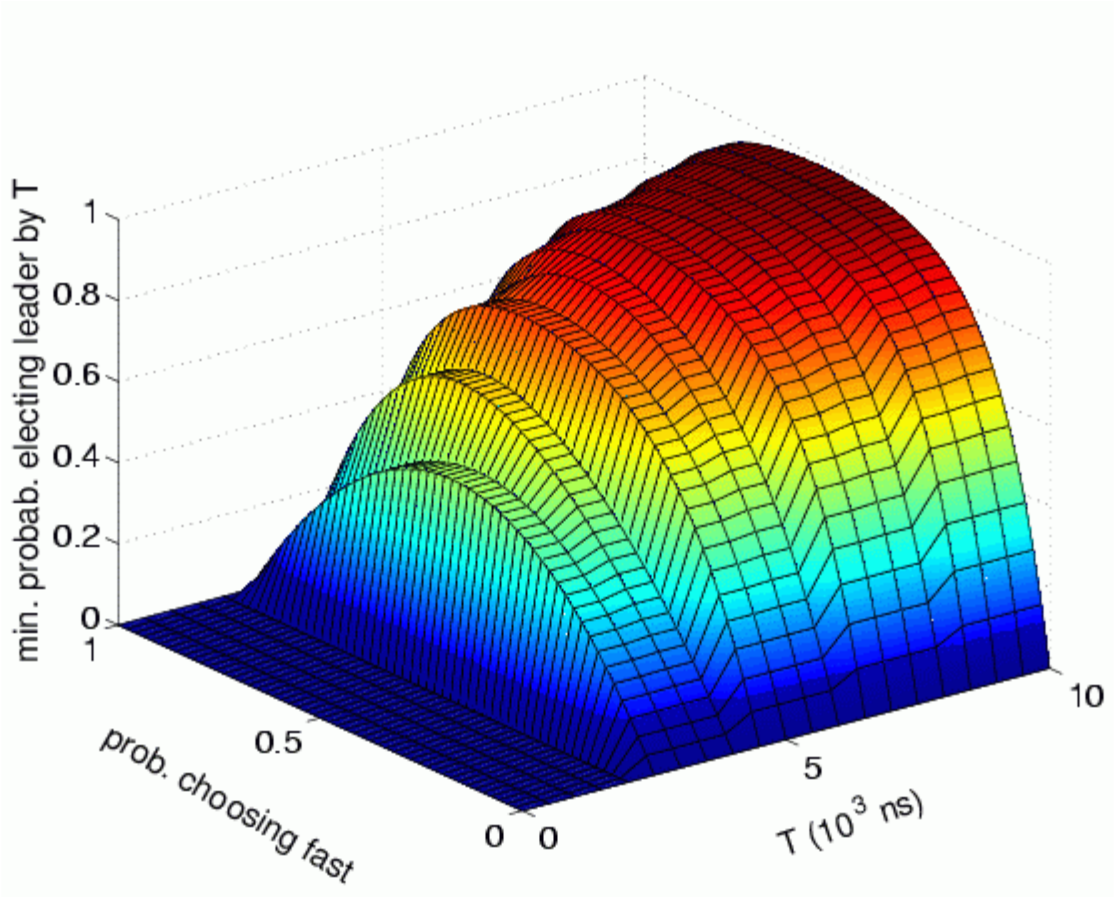
- Probabilistic model checking (with PRISM)
  - verified that root contention always resolved with probability 1
    - $P_{\geq 1}$ [F (end ∧ elected)]
  - investigated worst-case expected time taken for protocol to complete
    - $R_{max=?}$ [F (end ∧ elected)]
  - investigated the effect of using biased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

FireWire: Analysis results



"minimum probability of electing leader by time T"

FireWire: Analysis results

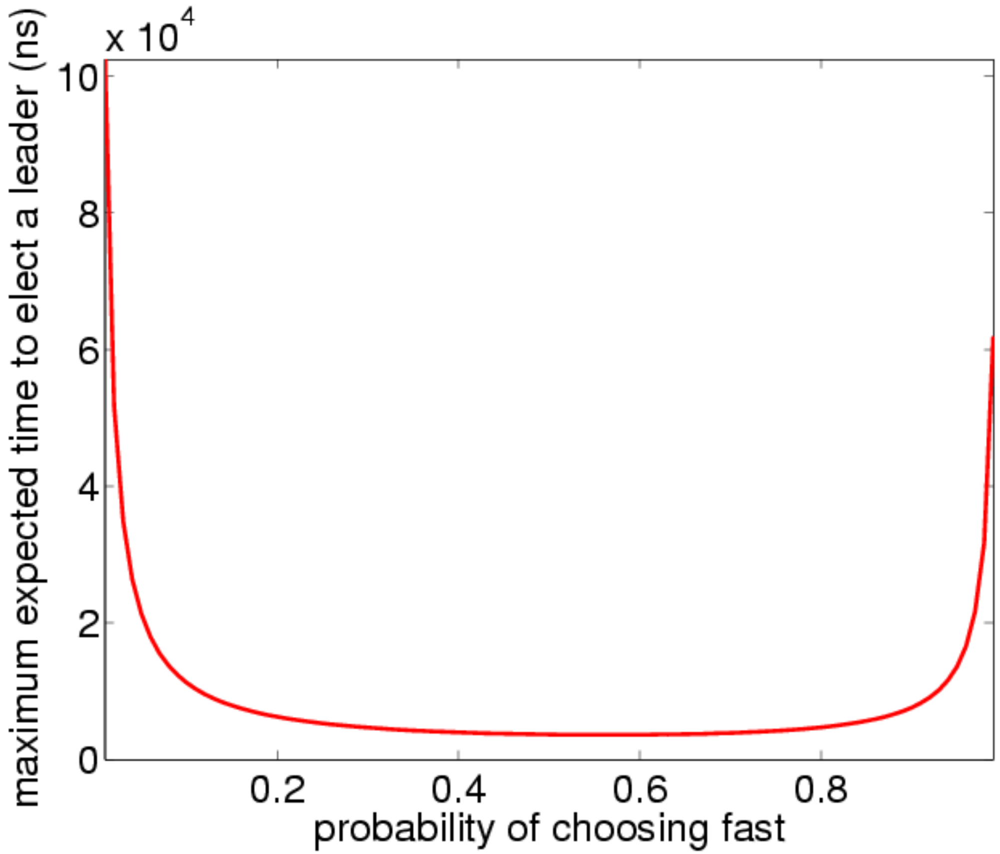


"minimum probability of electing leader by time T"

(short wire length)

Using a biased coin

FireWire: Analysis results

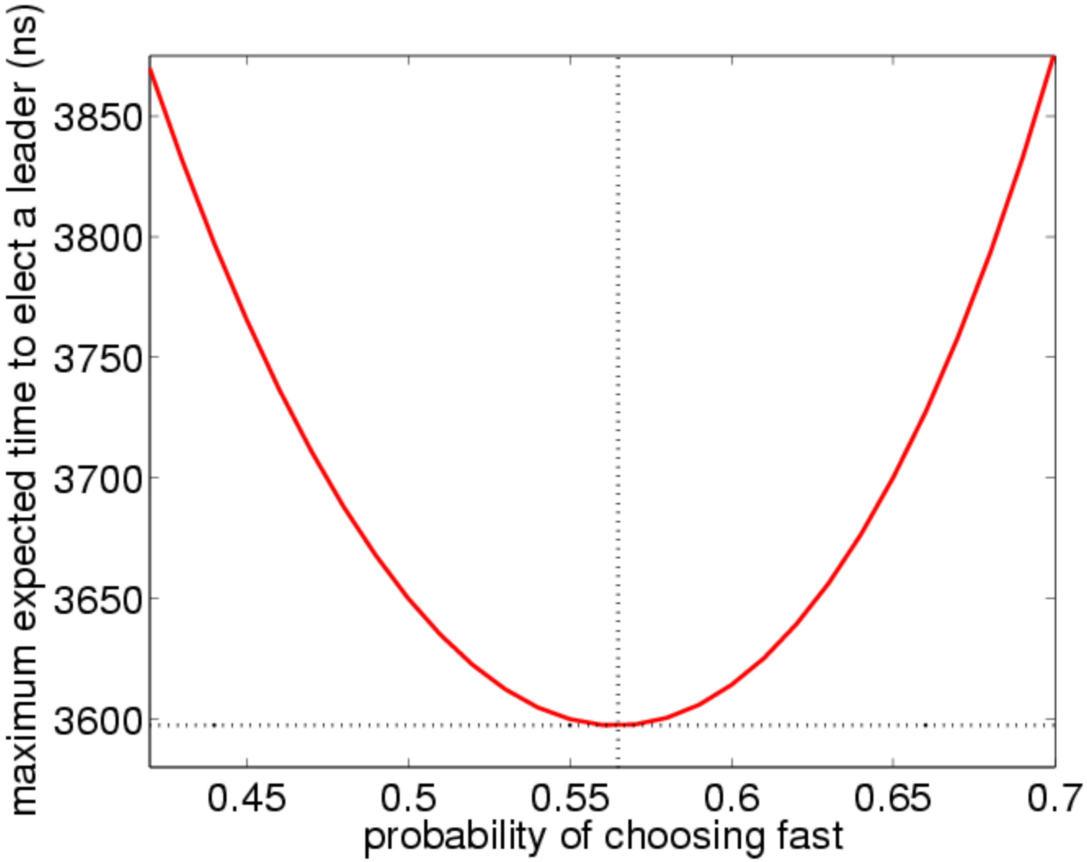


“maximum expected time to elect a leader”

(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!

Overview

- Probabilistic model checking
  - example: FireWire protocol
- Probabilistic timed automata (PTAs)
  - clocks, zones, syntax, semantics
  - property specification
- Verification techniques for PTAs
  - region graphs + digital clocks + zone-based methods
  - abstraction-refinement
- Tool support: PRISM
- Verification vs. controller synthesis
  - example: task-graph scheduling

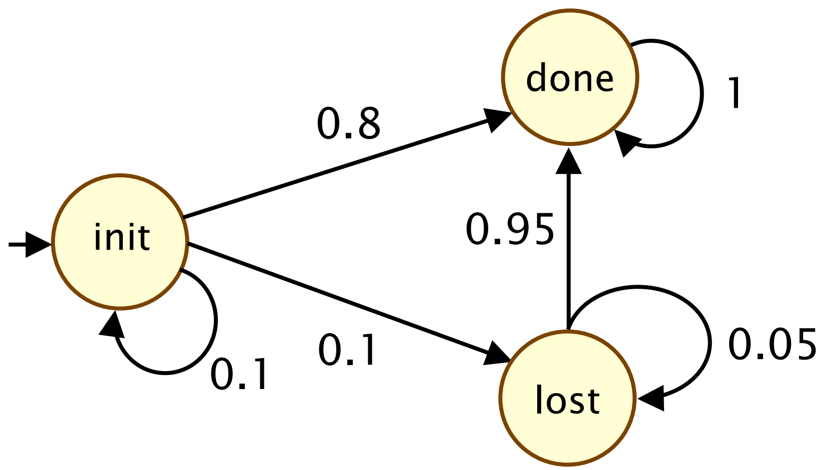
- See: www.prismmodelchecker.org/lectures/movep14/
  - slides, tutorial papers, reference list, …

Probabilistic models

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

Recap: DTMCs

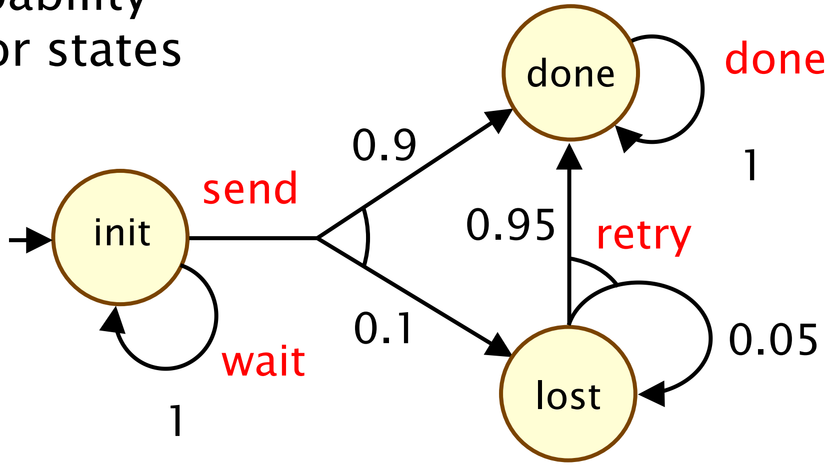
- Discrete-time Markov chains (DTMCs)
  - state-transition systems augmented with probabilities



- Model checking, e.g. with PCTL
  - based on probability measure over paths
  - e.g. $P_{<0.15}$ [F lost] – maximum probability of loss is < 0.15

Recap: MDPs

- Markov decision processes (MDPs) (or probabilistic automata)
  - mix probability and nondeterminism
  - states: nondeterministic choice over actions
  - each action leads to a probability distributions over successor states

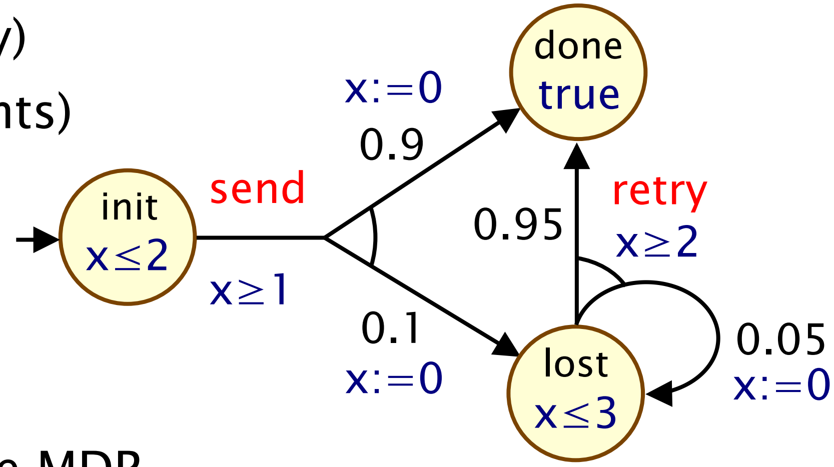


- Adversaries (schedulers, policies, …)
  - resolve nondeterministic choices based on history so far
  - properties quantify over all possible adversaries
  - e.g. $P_{<0.15}$ [F lost] – maximum probability of loss is < 0.15

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: choice of actions + elapse of time

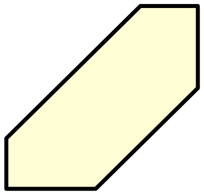
Time, clocks and clock valuations

- Dense (continuous) time domain: non-negative reals $\mathbb{R}_{\geq 0}$
  - from this point on, we will abbreviate $\mathbb{R}_{\geq 0}$ to $\mathbb{R}$
- Finite set of clocks $x \in X$
  - variables taking values from time domain $\mathbb{R}$
  - increase at the same rate as real time
- A clock valuation is a tuple $v \in \mathbb{R}^X$. Some notation:
  - $v(x)$: value of clock x in v
  - $v+t$: time increment of t for v
  - $v[Y:=0]$: clock reset of clocks $Y \subseteq X$ in v

Zones (clock constraints)

- Zones (clock constraints) over clocks X, denoted Zones(X):

$$\zeta ::= x \leq d \mid c \leq x \mid x+c \leq y+d \mid \neg\zeta \mid \zeta \vee \zeta$$



  - where $x, y \in X$ and $c, d \in \mathbb{N}$
  - e.g.: $x \leq 2$, $x \leq y$, $(x\geq2)\wedge(x<3)\wedge(x\leq y)$

- Can derive:
  - logical connectives, e.g. $\zeta_1 \wedge \zeta_2 \equiv \neg(\neg\zeta_1 \vee \neg\zeta_2)$
  - strict inequalities, through negation, e.g. $x>5 \equiv \neg(x\leq5)$…

- Used for both:
  - syntax of PTAs/properties
  - algorithms/implementations for model checking

Zones and clock valuations

- A clock valuation v satisfies a zone ζ, written $v \triangleright \zeta$ if
  - ζ resolves to true after substituting each clock x with v(x)

- The semantics of a zone $\zeta \in Zones(X)$ is the set of clock valuations which satisfy it (i.e. a subset of $\mathbb{R}^X$)
  - NB: multiple zones may have the same semantics
  - e.g. $(x\leq2)\wedge(y\leq1)\wedge(x\leq y+2)$ and $(x\leq2)\wedge(y\leq1)\wedge(x\leq y+3)$
  - but we assume canonical ("tight") zones
  - allows us to use syntax for zones interchangeably with semantic, set-theoretic operations

- Some useful classes of zones:
  - closed: no strict inequalities (e.g. $x>5$)
  - diagonal-free: no comparisons between clocks (e.g. $x\leq y$)
  - convex: define a convex set, efficient to manipulate

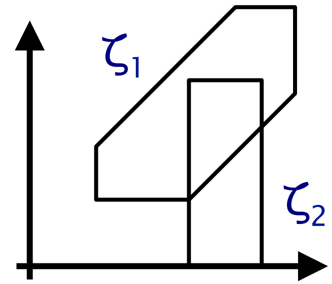
c-equivalence and c-closure

- Clock valuations v and v' are c-equivalent if for any x,y∈X
  - either v(x) = v'(x), or v(x) > c and v'(x) > c
  - either v(x)−v(y) = v'(x)−v'(y) or v(x)−v(y) > c and v'(x)−v'(y) > c

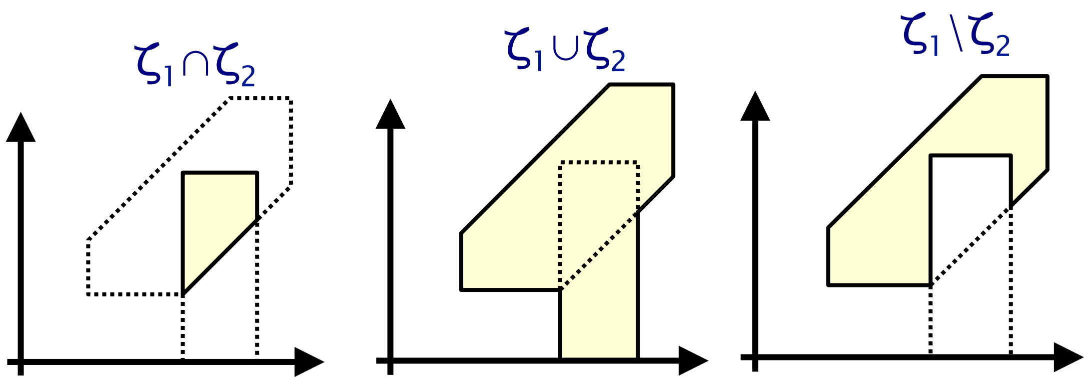
- The c-closure of the zone ζ, denoted close(ζ,c), equals
  - the greatest zone ζ'⊇ ζ such that, for any v' ∈ ζ', there exists v ∈ ζ and v and v' are c-equivalent
  - c-closure ignores all constraints which are greater than c
  - for a given c, there are only a finite number of c-closed zones

Operations on zones

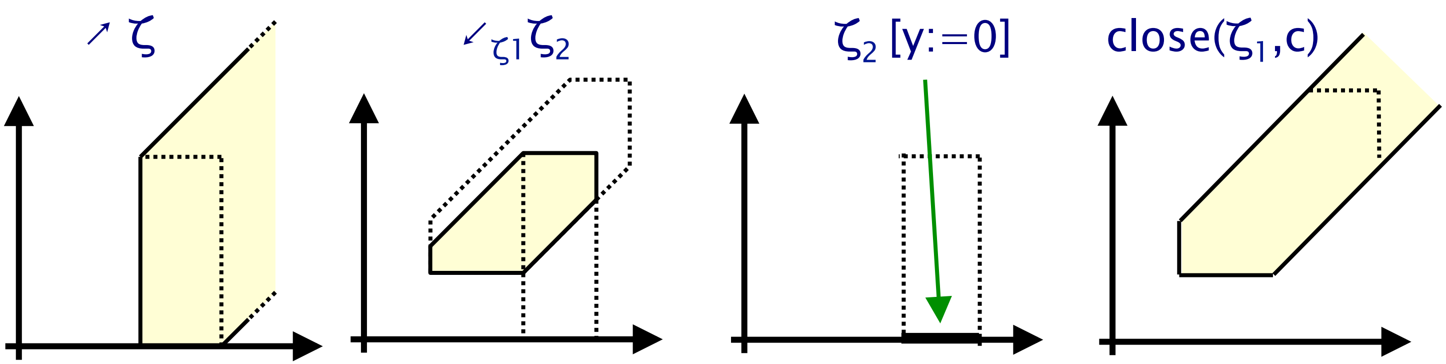
- Operations on zones:



- Set-theoretic operations



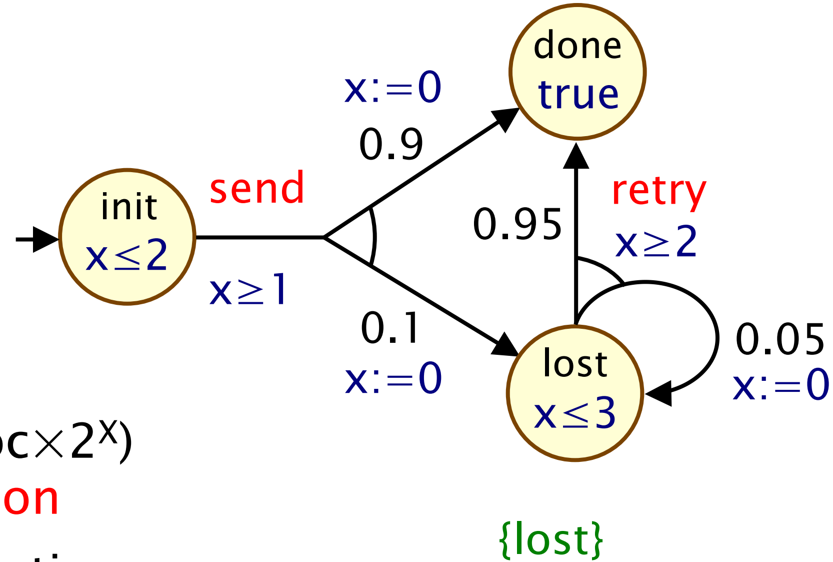
- Time operations



Probabilistic timed automata – Syntax

- A probabilistic timed automata (PTA) is:
  - a tuple (Loc, l_{init}, Act, X, inv, prob, L)

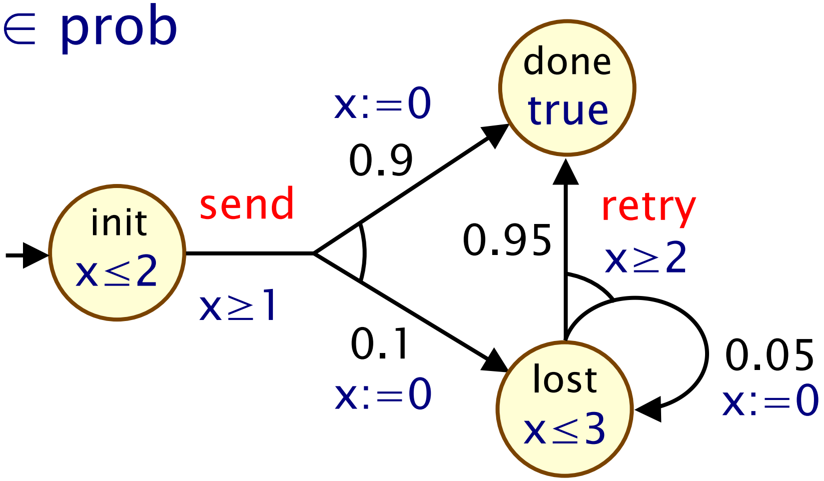
- where:
  - Loc is a finite set of locations
  - $l_{init} \in$ Loc is the initial location
  - Act is a finite set of actions
  - X is a finite set of clocks
  - inv : Loc → Zones(X) is the invariant condition
  - prob ⊆ Loc×Zones(X)×Dist(Loc×2^X) is the probabilistic edge relation
  - L : Loc → 2^{AP} is a labelling function



Probabilistic edge relation

- Probabilistic edge relation
  - prob ⊆ Loc×Zones(X)×Act×Dist(Loc×2^X)

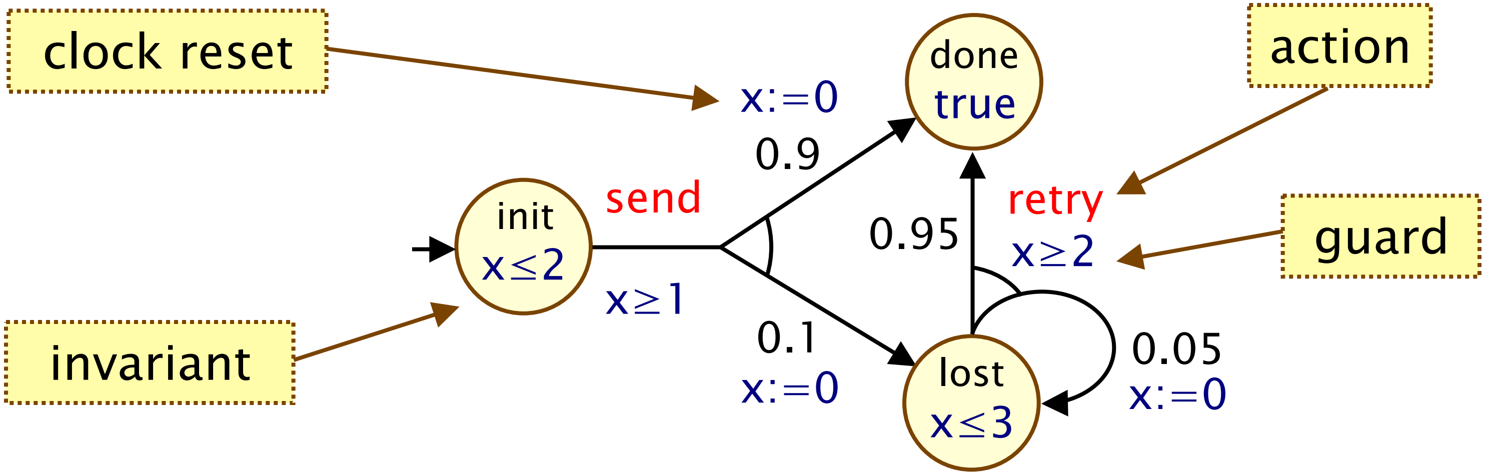
- Probabilistic edge (l,g,a,p) ∈ prob
  - l is the source location
  - g is the guard
  - a is the action
  - p target distribution



- Edge (l,g,a,p,l',Y)
  - from probabilistic edge (l,g,a,p) where p(l',Y)>0
  - l' is the target location
  - Y is the set of clocks to be reset (to zero)

PTA – Example

- Models a simple probabilistic communication protocol
  - starts in location init; after between 1 and 2 time units, the protocol attempts to send the data:
    - with probability 0.9 data is sent correctly, move to location done
    - with probability 0.1 data is lost, move to location lost
  - in location lost, after 2 to 3 time units, attempts to resend
    - correctly sent with probability 0.95 and lost with probability 0.05



PTAs – Behaviour

- A state of a PTA is a pair $(l,v) \in Loc \times \mathbb{R}^X$ such that $v \triangleright inv(l)$

- Start in the initial location with all clocks set to zero
  - i.e. initial state is $(l_{init},\underline{0})$

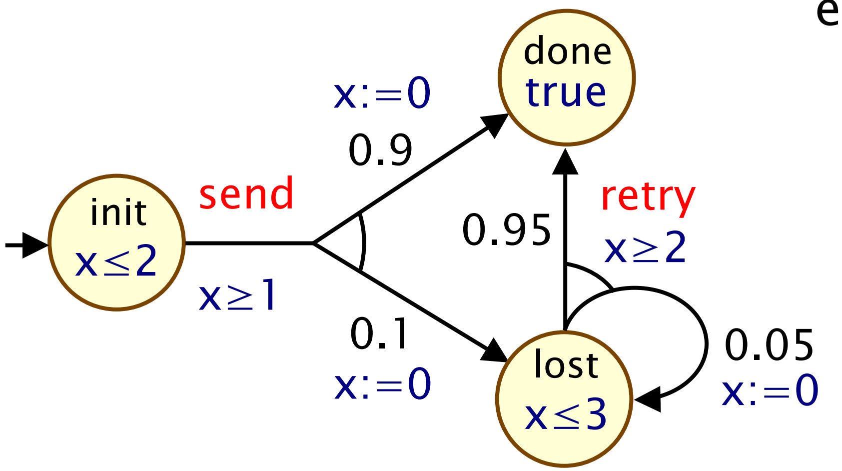
- For any state (l,v), there is nondeterministic choice between making a discrete transition and letting time pass

  - discrete transition (l,g,a,p) enabled if $v \triangleright g$ and probability of moving to location l’ and resetting the clocks Y equals $p(l',Y)$

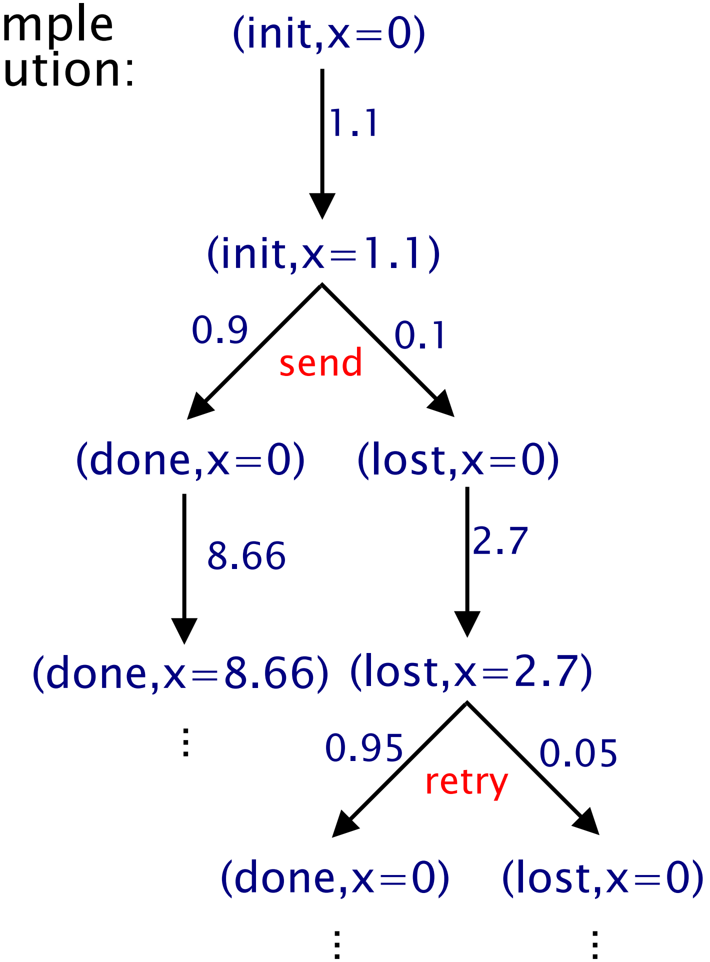
  - time transition available only if invariant $inv(l)$ is continuously satisfied while time elapses

PTA – Example execution

PTA:



Example execution:



PTAs – Formal semantics

- Formally, the semantics of a PTA P is an infinite-state MDP $M_P = (S_P, s_{init}, \alpha_P, \delta_P, L_P)$ with:

- States: $S_P = \{ (l,v) \in Loc \times \mathbb{R}^X$ such that $v \triangleright inv(l) \}$
- Initial state: $s_{init} = (l_{init}, \underline{0})$

actions of MDP M_P are the actions of PTA P or real time delays

- Actions: $\alpha_P = Act \cup \mathbb{R}$
- $\delta_P \subseteq S_P \times \alpha_P \times Dist(S_P)$ such that $(s, a, \mu) \in \delta_P$ iff:
  - (time transition) $a \in \mathbb{R}$, $\mu(l,v+t)=1$ and $v+t' \triangleright inv(l)$ for all $t' \leq t$
  - (discrete transition) $a \in Act$ and there exists $(l,g,a,p) \in prob$

    such that $v \triangleright g$ and, for any $(l',v') \in S_P$: $\mu(l',v') = \sum_{Y \subseteq X \wedge v[Y:=0]=v'} p(l',Y)$

- Labelling: $L_P(l,v) = L(l)$

multiple resets may give same clock valuation

Overview

- Probabilistic model checking
  - example: FireWire protocol
- Probabilistic timed automata (PTAs)
  - clocks, zones, syntax, semantics
  - property specification
- Verification techniques for PTAs
  - region graphs + digital clocks + zone-based methods
  - abstraction-refinement
- Tool support: PRISM
- Verification vs. controller synthesis
  - example: task-graph scheduling

- See: www.prismmodelchecker.org/lectures/movep14/
  - slides, tutorial papers, reference list, …

Properties of PTAs – PTCTL

- PTCTL: Probabilistic timed computation tree logic [KNSS02]
  - derived from PCTL [BdA95] and TCTL [AD94]

- Syntax:
  - $\phi ::= \text{true} \mid a \mid \zeta \mid z.\ \phi \mid \phi \wedge \phi \mid \neg\phi \mid P_{\sim p}\,[\,\phi\ U\ \phi\,]$

"zone over X∪Z"

"freeze quantifier" (formula clock z)

ϕ U ϕ is true with probability ~p (for all adversaries)

- where:
  - where Z is a set of formula clocks, $\zeta \in \text{Zones}(X \cup Z)$, $z \in Z$,
  - a is an atomic proposition, $p \in [0,1]$ and $\sim\ \in \{<,>,\leq,\geq\}$

- Usual equivalences
  - e.g. $F\ \phi \equiv \text{true}\ U\ \phi$ and $G\ \phi \equiv \neg F(\neg\phi)$

PTCTL – Examples

- $z \,.\, P_{>0.99}$ [F delivered $\wedge$ $(z<5)$]
  - "with probability greater than 0.99, the system delivers the packet within 5 time units"
- $z \,.\, P_{>0.95}$ [$(x\leq 3)$ U $(z=8)$]
  - "with probability at least 0.95, the system clock x does not exceed 3 before 8 time units elapse"
- $z \,.\, P_{\leq 0.1}$ [G (failure $\vee$ $(z\leq 60)$)]
  - "the system fails after the first 60 time units have elapsed with probability at most 0.01"

Properties of PTAs (PRISM)

- PRISM property specification for PTAs [NPS13]
  - PCTL + zones + time bounds + expected rewards
- Syntax:
  - $\phi ::= \text{true} \mid a \mid \zeta \mid \phi \wedge \phi \mid \neg\phi \mid P_{\sim p} [\, \psi \,] \mid R^{r}_{\sim q} [\, \rho \,]$
  - $\psi ::= \phi \; U^{\leq k} \; \phi \mid \phi \; U \; \phi$
  - $\rho ::= I^{=k} \mid C^{\leq k} \mid F \; \phi$

expected reward ρ
(for reward structure r)
satisfies $\sim q$
(for all adversaries)

- Expected reward (costs/prices)
  - at time k ($I^{=k}$)
  - cumulated up to time k ($C^{\leq k}$)
  - cumulated until a ϕ-state is reached ($F \; \phi$)
- Reward structures
  - location rewards (rate accumulated) + transition rewards
- Also: numerical variants: $P_{max=?}$, $R^{r}_{min=?}$, etc.

Examples

- Examples
  - $P_{\geq 0.8}$ [$F^{\leq k}$ ack_n] – "the probability that the sender has received *n* acknowledgements within time *k* is at least 0.8"
  - trigger → $P_{<0.0001}$ [$G^{\leq 20}$ ¬deploy] – "the probability of the airbag failing to deploy within 20 milliseconds of being triggered is strictly less than 0.0001"
  - $P_{max=?}$ [¬sent U fail] – "what is the maximum probability of a failure occurring before message transmission is complete?"
  - $R^{time}_{max=?}$ [F end] – "what is the maximum expected time for the protocol to terminate?"
  - $R^{pwr}_{<q}$[$C^{\leq 60}$] – "the expected energy consumption during the first 60 seconds is $< q$"

- Property reductions [NPS13]
  - verification reduces to probabilistic reachability (P [F ϕ]) and expected reachability (R[F ϕ]), e.g. by adding extra clocks

Time divergence

- We restrict our attention to time divergent behaviour
  - a common restriction imposed in real-time systems
  - unrealisable behaviour (i.e. corresponding to time not advancing beyond a time bound) is disregarded
  - also called non-zeno behaviour

- For a path $\omega = s_0(a_0,\mu_0)s_1(a_1,\mu_1)s_2(a_2,\mu_2)\ldots$ in the MDP M_P
  - $D_\omega(n)$ denotes the duration up to state s_n
  - i.e. $D_\omega(n) = \Sigma \{| \, a_i \mid 0 \leq i < n \wedge a_i \in \mathbb{R} \, |\}$

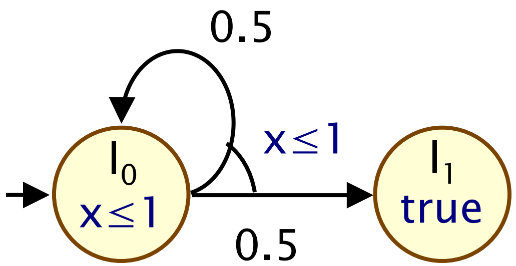
- A path ω is time divergent if, for any $t \in \mathbb{R}_{\geq 0}$:
  - there exists $j \in \mathbb{N}$ such that $D_\omega(j) > t$

- Example of non-divergent path:
  - $s_0(1,\mu_0)s_0(0.5,\mu_0)s_0(0.25,\mu_0)s_0(0.125,\mu_0)s_0\ldots$

Time divergence

- An adversary of M_P is divergent if, for each state $s \in S_P$:
  - the probability of divergent paths under A is 1
  - i.e $Pr^A_s\{ \omega \in Path^A(s) \mid \omega \text{ is divergent } \} = 1$

- Motivation for probabilistic definition of divergence:



  - in this PTA, any adversary has one non-divergent path:
    - takes the loop in l_0 infinitely often, without 1 time unit passing
  - but the probability of such behaviour is 0
  - a stronger notion of divergence would mean no divergent adversaries exist for this PTA

Overview

- Probabilistic model checking
  - example: FireWire protocol
- Probabilistic timed automata (PTAs)
  - clocks, zones, syntax, semantics
  - property specification
- Verification techniques for PTAs
  - region graphs + digital clocks + zone-based methods
  - abstraction-refinement
- Tool support: PRISM
- Verification vs. controller synthesis
  - example: task-graph scheduling

- See: www.prismmodelchecker.org/lectures/movep14/
  - slides, tutorial papers, reference list, ...

PTA model checking – Summary

- 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)
  - forwards reachability [KNSS02]
  - backwards reachability [KNSW07]
  - game-based abstraction refinement [KNP09c]

The region graph

- Region graph construction for PTAs [KNSS02]
  - adapts region graph construction for timed automata [ACD93]
  - partitions PTA states into a finite set of regions
  - based on notion of clock equivalence
  - construction is also dependent on PTCTL formula

- For a PTA P and PTCTL formula ϕ
  - construct a time-abstract, finite-state MDP R(ϕ)
  - translate PTCTL formula ϕ to PCTL formula ϕ’
  - ϕ is preserved by region equivalence
  - i.e. ϕ holds in a state of M_P if and only if ϕ’ holds in the corresponding state of R(ϕ)
  - model check R(ϕ) using standard methods for MDPs

The region graph – Clock equivalence

- Regions are sets of clock equivalent clock valuations

- Some notation:
  - let c be largest constant appearing in PTA or PTCTL formula
  - let $\lfloor t \rfloor$ denotes the integral part of t
  - t and t' agree on their integral parts if and only if

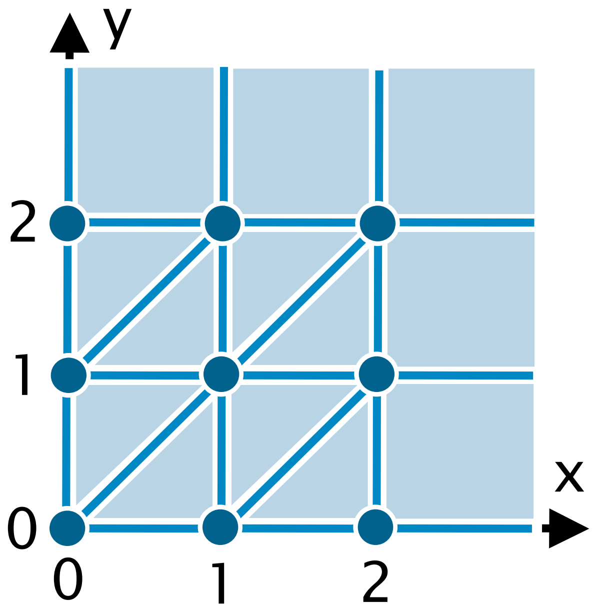
    (1) $\lfloor t \rfloor = \lfloor t' \rfloor$

    (2) t and t' are both integers or neither is an integer

- Clock valuations v and v' are clock equivalent ($v \cong v'$) if:
  - for all clocks $x \in X$, either:
    - v(x) and v'(x) agree on their integral parts
    - $v(x)>c$ and $v'(x)>c$
  - for all clock pairs $x,y \in X$, either:
    - $v(x) - v(x')$ and $v'(x) - v'(x')$ agree on their integral parts
    - $v(x) - v(x') > c$ and $v'(x) - v'(x') > c$

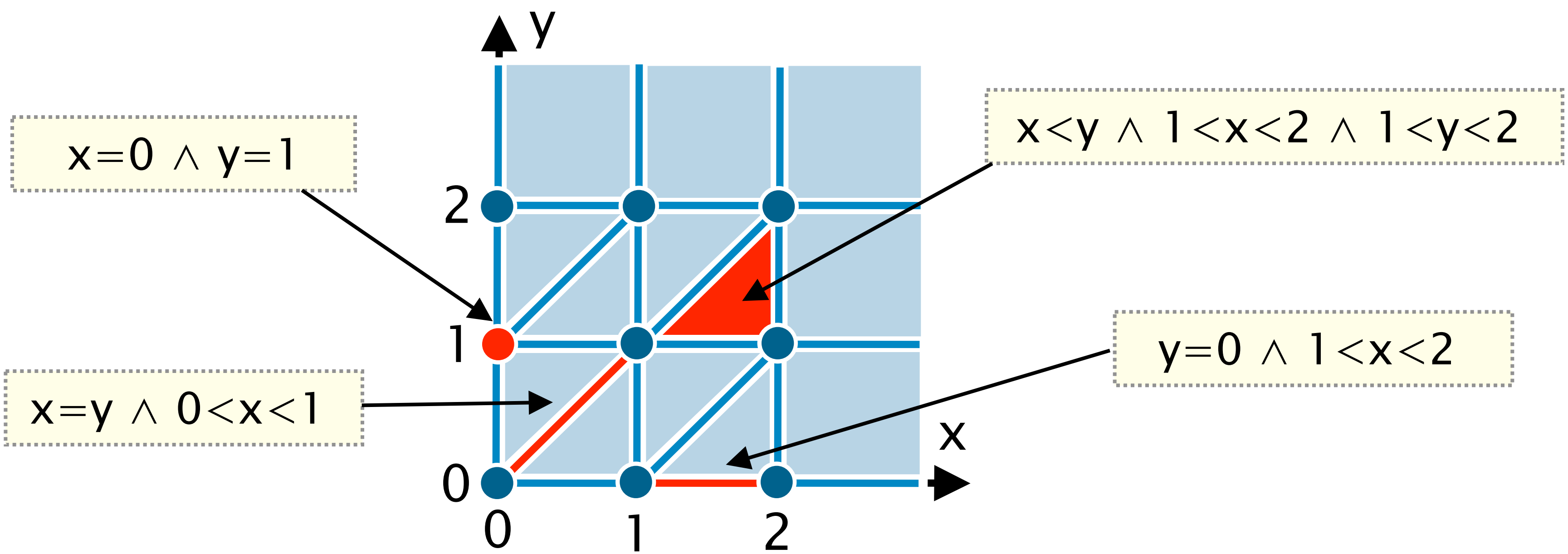
Region graph – Clock equivalence

- Example regions (for 2 clocks x and y)



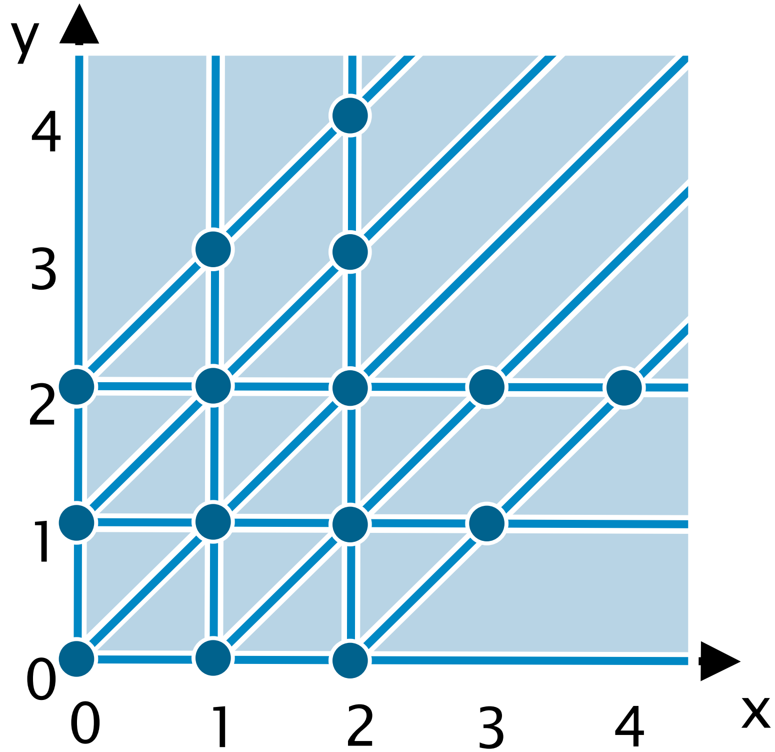
Region graph – Clock equivalence

- Example regions (for 2 clocks x and y)



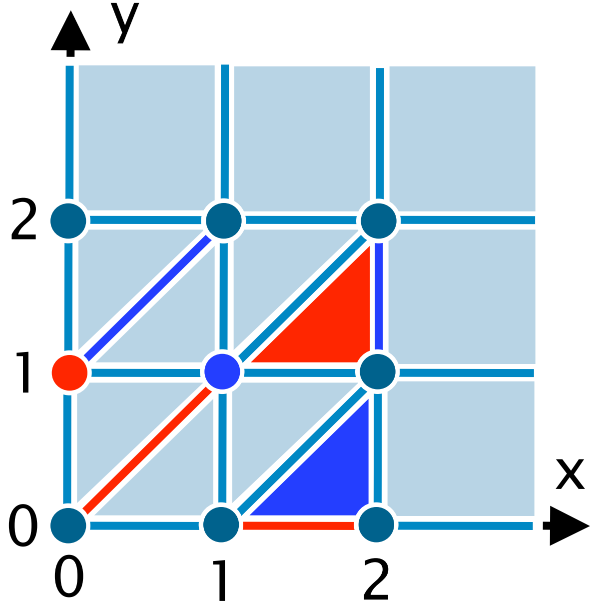
Region graph – Clock equivalence

- Fundamental result: if $v \cong v'$, then $v \triangleright \zeta \Leftrightarrow v' \triangleright \zeta$
  - it follows that $r \triangleright \zeta$ is well defined for a region r
- All regions (for 2 clocks x and y), max constant c=2:



Region graph – Clock equivalence

- r’ is the (time) successor region of r, written succ(r) = r, if
  - for each $v \in r$, there exists $t > 0$ such that:
  - $v+t \in r'$ and $v+t' \in r \cup r'$ for all $t' < t$
- Examples (region and successor):



- Region graph: MDP over states (l,r) for location l, region r

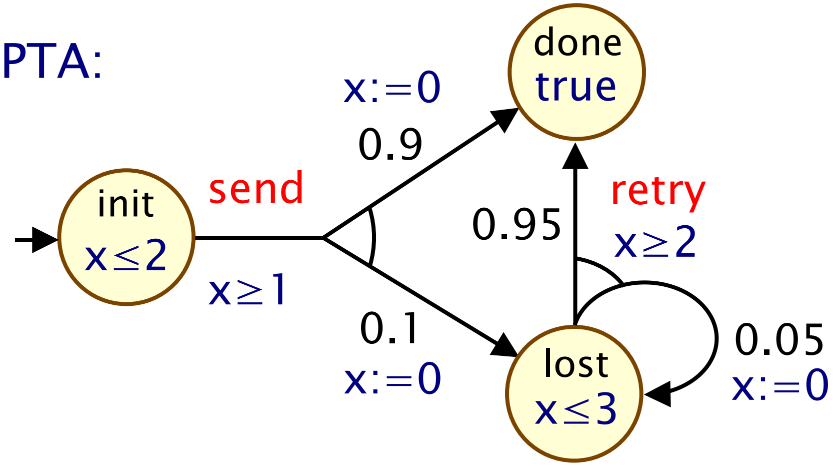
The region graph

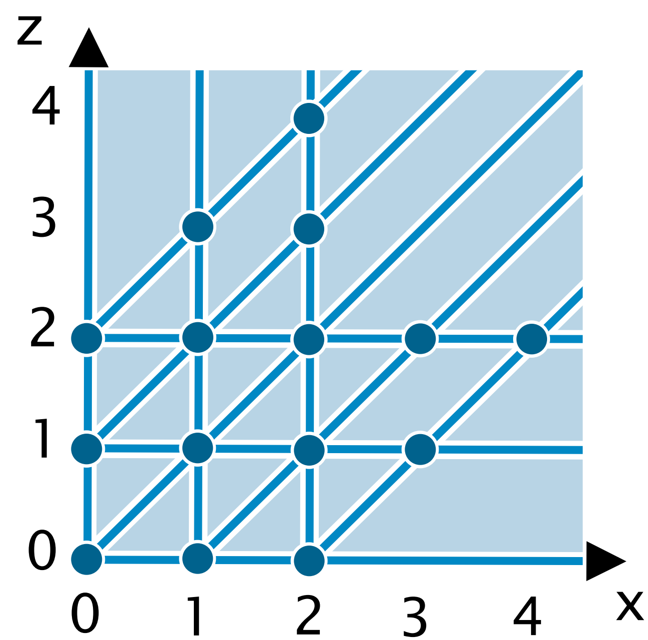
- The region graph MDP is $(S_R, s_{init}, \mathbf{Steps}_R, L_R)$ where…
  - the set of states S_R comprises pairs (l,r) such that l is a location and r is a region over $X \cup Z$
  - the initial state is $(l_{init}, \underline{0})$
  - the set of actions is $\{succ\} \cup Act$
    - succ is a unique action denoting passage of time
  - the probabilistic transition function $\mathbf{Steps}_R$ is defined as:
  - $S_R \times 2^{(\{succ\} \cup Act) \times Dist(S_R)}$
  - $(succ, \mu) \in \mathbf{Steps}_R(l,r)$ iff $\mu(l, succ(r)) = 1$
  - $(a, \mu) \in \mathbf{Steps}_R(l,r)$ if and only if $\exists\ (l,g,a,p) \in prob$ such that

    $r \triangleright g$ and, for any $(l',r') \in S_R$: $\mu(l',r') = \sum_{Y \subseteq X \wedge r[Y:=0]=r'} p(l',Y)$
  - the labelling is given by: $L_R(l,r) = L(l)$

Region graph – Example

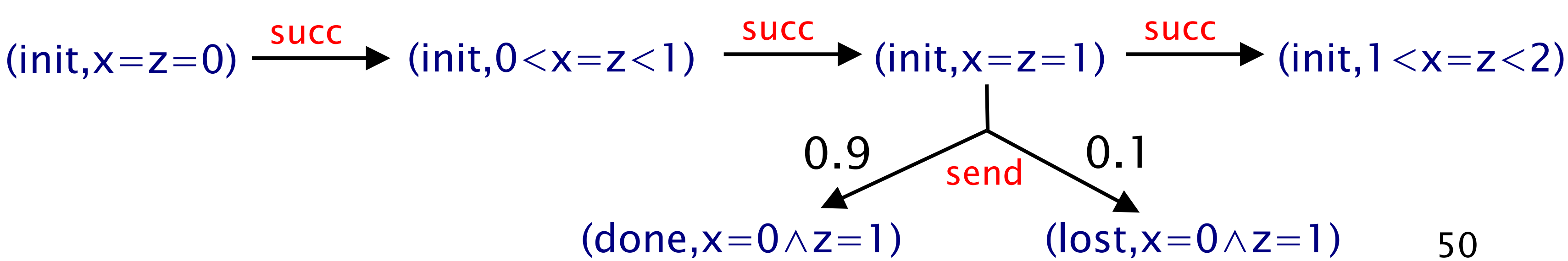
PTA:





PTCTL formula: $z.P_{\leq 0.1}$ [F (done ∧ z<2)]

Region graph (fragment):



Region graph construction

- Region graph
  - useful for establishing decidability of model checking
  - or proving complexity results for model checking algorithms
- But…
  - the number of regions is exponential in the number of clocks and the size of largest constant
  - so model checking based on this is extremely expensive
  - and so not implemented (even for timed automata)
- Improved approaches based on:
  - digital clocks
  - zones (unions of regions)

Overview

- Probabilistic model checking
  - example: FireWire protocol
- Probabilistic timed automata (PTAs)
  - clocks, zones, syntax, semantics
  - property specification
- Verification techniques for PTAs
  - region graphs + digital clocks + zone-based methods
  - abstraction-refinement
- Tool support: PRISM
- Verification vs. controller synthesis
  - example: task-graph scheduling

- See: www.prismmodelchecker.org/lectures/movep14/
  - slides, tutorial papers, reference list, …

Digital clocks

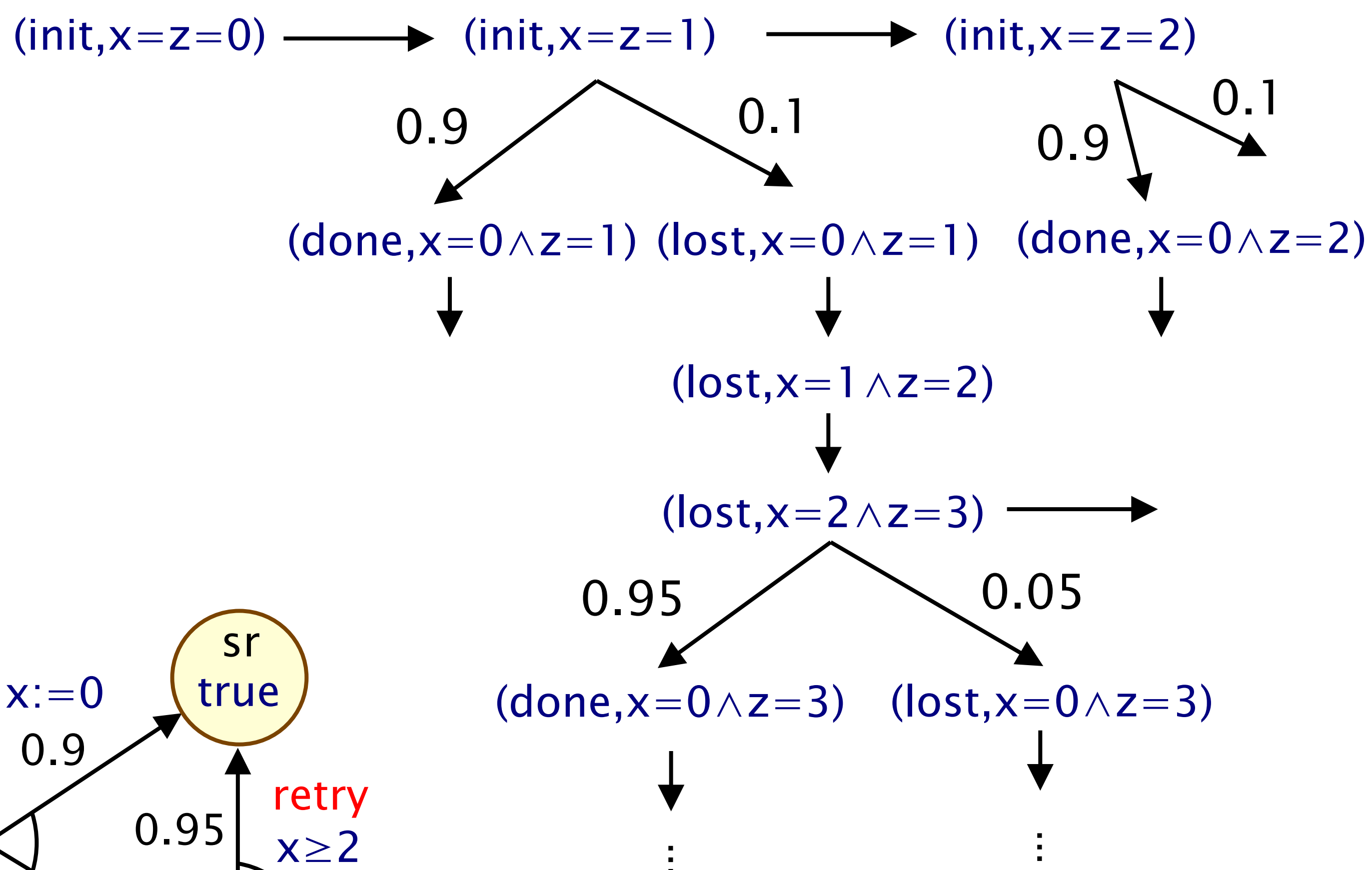
- Simple idea: Clocks can only take integer (digital) values
  - i.e. time domain is $\mathbb{N}$ as opposed to $\mathbb{R}$
  - based on notion of ϵ-digitisation [HMP92]

- Only applies to a restricted class of PTAs; zones must be:
  - closed – no strict inequalities (e.g. $x>5$)
  - diagonal-free: no comparisons between clocks (e.g. $x \leq y$)

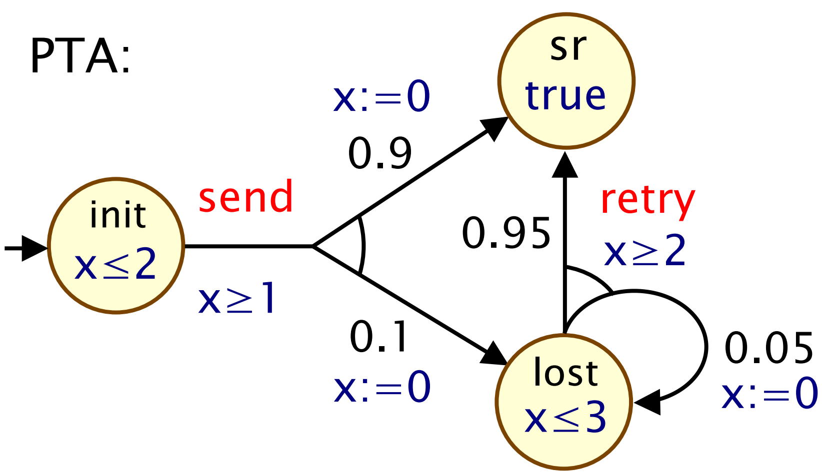
- Digital clocks semantics yields a finite-state MDP
  - state space is a subset of $Loc \times \mathbb{N}^X$, rather than $Loc \times \mathbb{R}^X$
  - clocks bounded by c_{max} (max constant in PTA and formula)
  - then use standard techniques for finite-state MDPs

Example – Digital clocks

MDP: (digital clocks)



PTA:

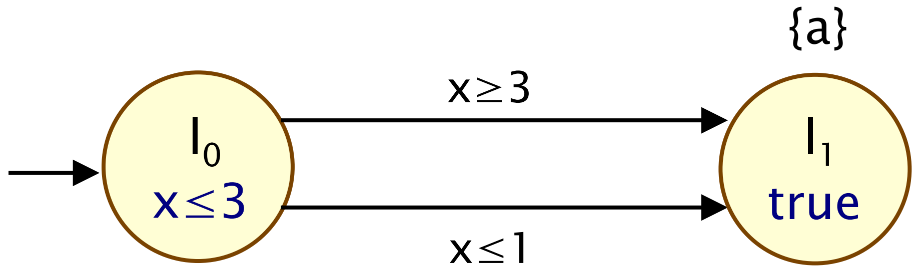


Digital clocks

- Digital clocks approach preserves:
  - minimum/maximum reachability probabilities
  - a subset of PTCTL properties
    - (no nesting, only closed zones in formulae)
  - only works for the initial state of the PTA
    - (but can be extended to any state with integer clock values)
  - also: expected rewards (priced PTAs)

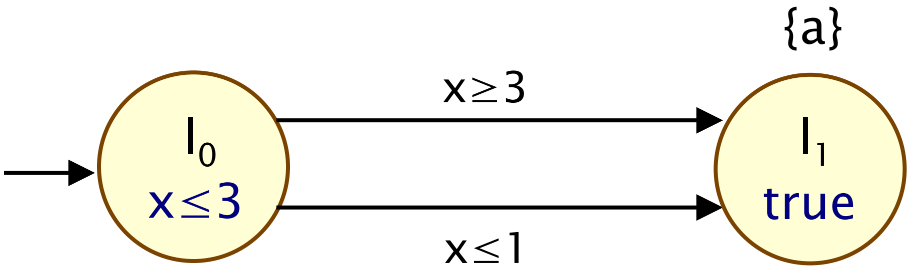
- In practice:
  - translation from PTA to MDP can often be done manually
  - (by encoding the PTA directly into the PRISM language)
  - automated translations exist: mcpta and PRISM
  - many case studies, despite "closed" restriction
  - potential problem: can lead to very large MDPs
  - alleviated partially by efficient symbolic model checking

Digital clocks do not preserve PTCTL



- Consider the PTCTL formula $\phi = z.P_{<1}$ [F (a $\wedge$ $z \leq 1$)]
  - a is an atomic proposition only true in location l_1
- Digital semantics:
  - no state satisfies ϕ since for any state we have $Prob^A(s, \mathcal{E}[z:=0], \text{true } U\ (a \wedge z \leq 1)) = 1$ for some adversary A
  - hence $P_{<1}$ [true U ϕ] is trivially true in all states

Digital clocks do not preserve PTCTL



- Consider the PTCTL formula $\phi = z.P_{<1}$ [F ($a \wedge z \leq 1$)]
  - a is an atomic proposition only true in location l_1
- Dense time semantics:
  - any state (l_0, v) where $v(x) \in (1,2)$ satisfies ϕ
    more than one time unit must pass before we can reach l_1
  - hence $P_{<1}$ [true U ϕ] is not true in the initial state

Overview

- Probabilistic model checking
  - example: FireWire protocol
- Probabilistic timed automata (PTAs)
  - clocks, zones, syntax, semantics
  - property specification
- Verification techniques for PTAs
  - region graphs + digital clocks + zone-based methods
  - abstraction-refinement
- Tool support: PRISM
- Verification vs. controller synthesis
  - example: task-graph scheduling

- See: www.prismmodelchecker.org/lectures/movep14/
  - slides, tutorial papers, reference list, …

Zone-based approaches

- An alternative is to use zones to construct an MDP
- Conventional symbolic model checking relies on computing
  - **post**(S’) the states that can be reached from a state in S’ in a single step
  - **pre**(S’) the states that can reach S’ in a single step
- Extend these operators to include time passage
  - **dpost**[e](S’) the states that can be reached from a state in S’ by traversing the edge e
  - **tpost**(S’) the states that can be reached from a state in S’ by letting time elapse
  - **pre**[e](S’) the states that can reach S’ by traversing the edge e
  - **tpre**(S’) the states that can reach S’ by letting time elapse

Zone-based approaches

- Symbolic states (l, ζ) where
  - l $\in$ Loc (location)
  - ζ is a zone over PTA clocks and formula clocks
  - generally fewer zones than regions

- **tpost**(l,ζ) = (l, $\nearrow\zeta \wedge$inv(l))
  - $\nearrow\zeta$ can be reached from ζ by letting time pass
  - $\nearrow\zeta\wedge$inv(l) must satisfy the invariant of the location l

- **tpre**(l,ζ) = (l, $\swarrow\zeta\wedge$inv(l))
  - $\swarrow\zeta$ can reach ζ by letting time pass
  - $\swarrow\zeta\wedge$ inv(l) must satisfy the invariant of the location l

Zone-based approaches

- For an edge e= (l,g,a,p,l’,Y) where
  - l is the source
  - g is the guard
  - a is the action
  - l’ is the target
  - Y is the clock reset

- **dpost**[e](l,ζ) = (l’, ($\zeta \wedge$g)[Y:=0])
  - $\zeta \wedge$g satisfy the guard of the edge
  - ($\zeta \wedge$g)[Y:=0] reset the clocks Y

- **dpre**[e](l’,ζ’) = (l, [Y:=0]ζ’ $\wedge$ (g $\wedge$ inv(l)))
  - [Y:=0]ζ’ the clocks Y were reset
  - [Y:=0]ζ’ $\wedge$ (g $\wedge$ inv(l)) satisfied guard and invariant of l

Forwards reachability

- Based on the operation **post**[e](l,ζ) = **tpost**(**dpost**[e](l,ζ))
  - (l',v') $\in$ **post**[e](l,ζ) if there exists (l,v) $\in$ (l,ζ) such that after traversing edge e and letting time pass one can reach (l',v')
- Forwards algorithm (part 1)
  - start with initial state S_F={**tpost**((l_{init},$\underline{0}$))} then iterate

    for each symbolic state (l,ζ) $\in$ S_F and edge e

    add **post**[e](l,ζ) to S_F
  - until set of symbolic states S_F does not change
- To ensure termination need to take c-closure of each zone encountered (c is the largest constant in the PTA)

Forwards reachability

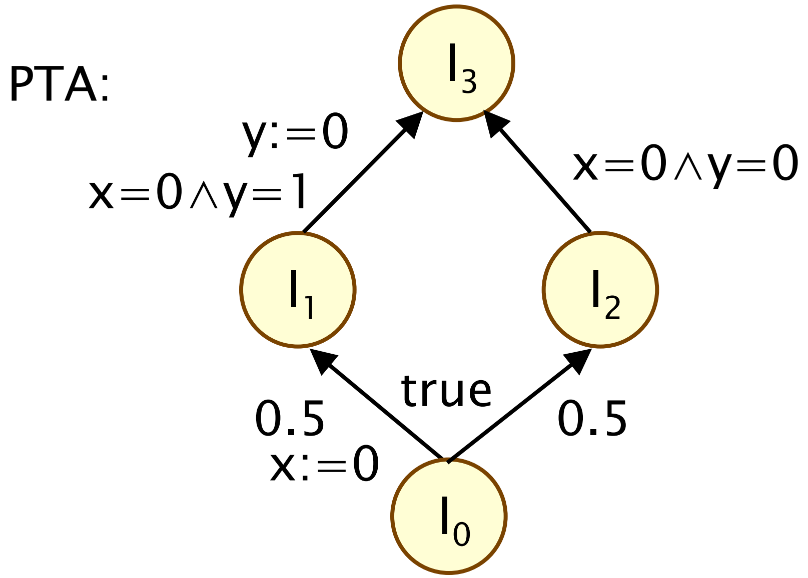
- Forwards algorithm (part 2)
  - construct finite state MDP $(S_F,(l_{init},\underline{0}),\mathbf{Steps}_F,L_F)$
  - states S_F (returned from first part of the algorithm)
  - $L_F(l,\zeta)=L(l)$ for all $(l,\zeta) \in S_F$
  - $\mu \in \mathbf{Steps}_F(l,\zeta)$ if and only if

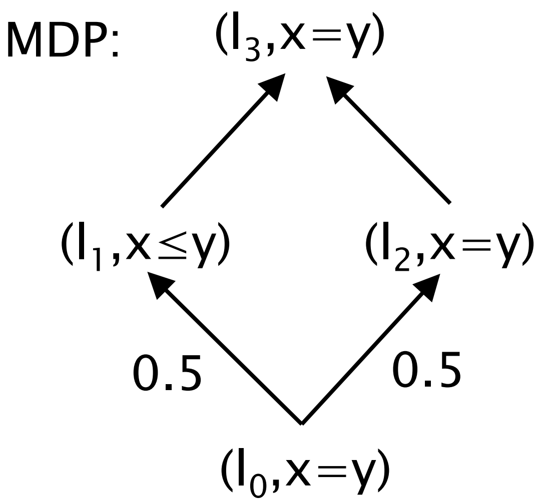
    there exists a probabilistic edge (l,g,a,p) of PTA such that for any $(l', \zeta') \in Z$:

$$\mu(l',\zeta') = \sum \{|\, p(l',X) \mid (l,g,\sigma,p,l',X) \in edges(p) \wedge post[e](l,\zeta) = (l',\zeta') \,|\}$$

summation over all the edges of (l,g,a,p) such that applying **post** to (l,ζ) leads to the symbolic state (l',ζ')

Forwards reachability – Example



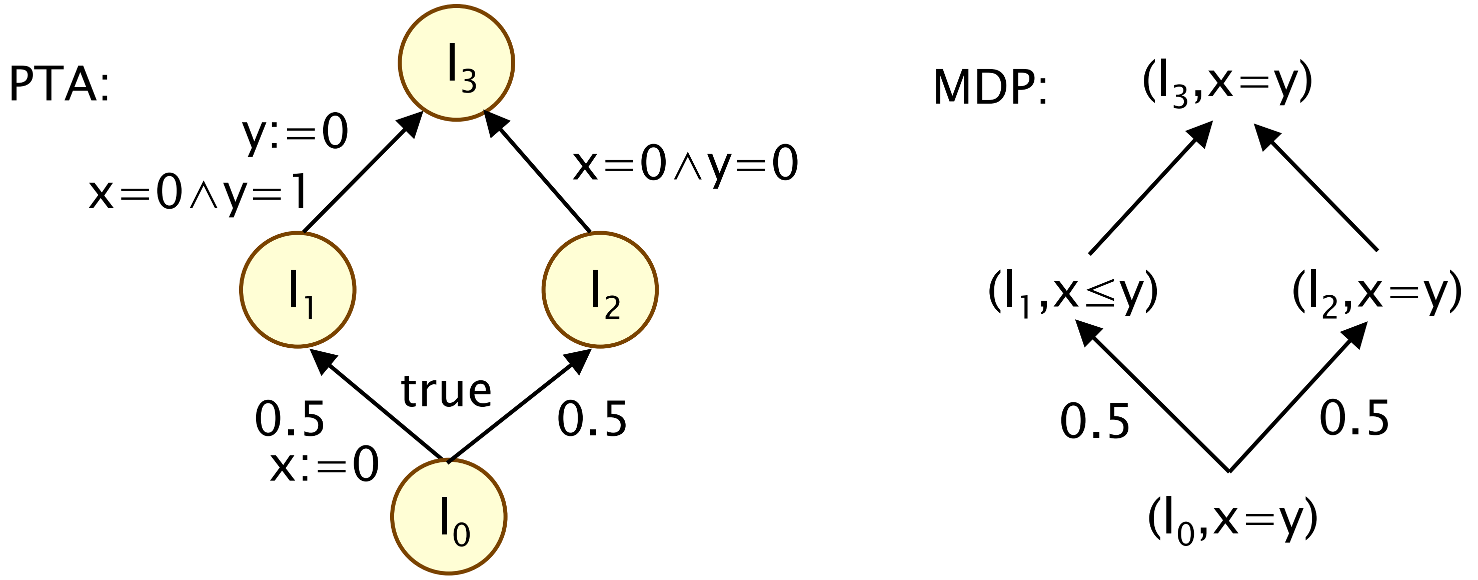


Forwards reachability – Limitations

- Problem reduced to analysis of finite-state MDP, but…
- Only obtain upper bounds on maximum probabilities
  - caused by when edges are combined
- Suppose **post**$[e_1](l,\zeta)=(l_1,\zeta_1)$ and **post**$[e_2](l,\zeta)=(l_2, \zeta_2)$
  - where e_1 and e_2 from the same probabilistic edge
- By definition of **post**
  - there exists $(l,v_i) \in (l,\zeta)$ such that a state in (l_i, ζ_i) can be reached by traversing the edge e_i and letting time pass
- Problem
  - we combine these transitions but are (l,v_1) and (l,v_2) the same?
  - may not exist states in (l,ζ) for which both edges are enabled

Forwards reachability – Example

- Maximum probability of reaching l_3 is 0.5 in the PTA
  - for the left branch need to take the first transition when $x=1$
  - for the right branch need to take the first transition when $x=0$
- However, in the forwards reachability graph probability is 1
  - can reach l_3 via either branch from $(l_0, x=y)$



Backwards reachability

- An alternative zone-based method: backwards reachability
  - state-space exploration in opposite direction, from target to initial states; uses **pre** rather than **post** operator
- Basic ideas: (see [KNSW07] for details)
  - construct a finite-state MDP comprising symbolic states
  - need to keep track of branching structure and take conjunctions of symbolic states if necessary
  - MDP yields maximum reachability probabilities for PTA
  - for min. probs, do graph-based analysis and convert to max.
- Advantages:
  - gives (exact) minimum/maximum reachability probabilities
  - extends to full PTCTL model checking
- Disadvantage:
  - operations to implement are expensive, limits applicability
  - (requires manipulation of non-convex zones)

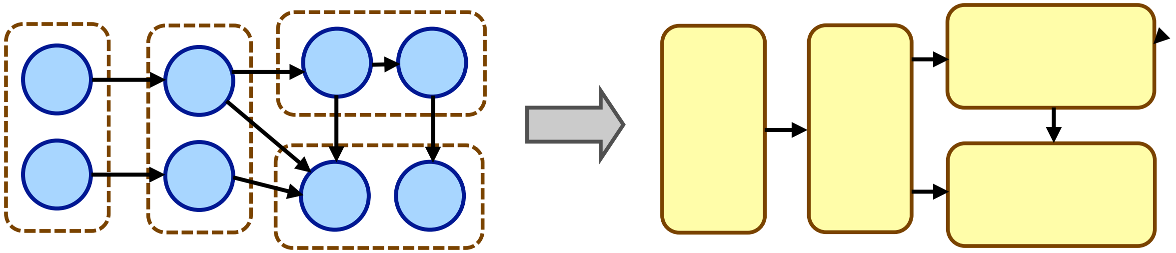
Overview

- Probabilistic model checking
  - example: FireWire protocol
- Probabilistic timed automata (PTAs)
  - clocks, zones, syntax, semantics
  - property specification
- Verification techniques for PTAs
  - region graphs + digital clocks + zone-based methods
  - abstraction-refinement
- Tool support: PRISM
- Verification vs. controller synthesis
  - example: task-graph scheduling

- See: www.prismmodelchecker.org/lectures/movep14/
  - slides, tutorial papers, reference list, …

Abstraction

- Very successful in (non-probabilistic) formal methods
  - essential for verification of large/infinite-state systems
  - hide details irrelevant to the property of interest
  - yields smaller/finite model which is easier/feasible to verify
  - loss of precision: verification can return "don't know"
- Construct abstract model of a concrete system
  - e.g. based on a partition of the concrete state space
  - an abstract state represents a set of concrete states

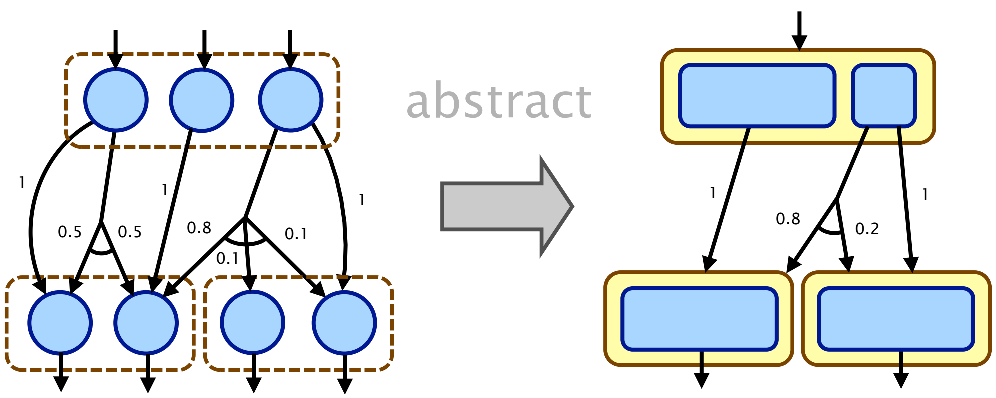


- Automatic generation of abstractions using refinement
  - start with a simple coarse abstraction; iteratively refine

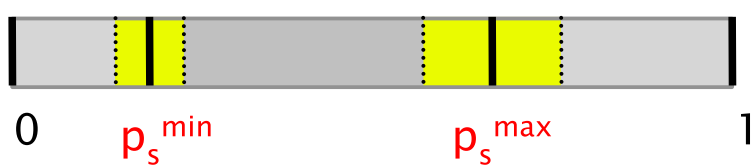
Abstraction of MDPs

- Abstraction increases degree of nondeterminism [DDJL01]
  - i.e. minimum probabilities are lower and maximums higher

- We build abstractions of MDPs as stochastic games [KNP06b]

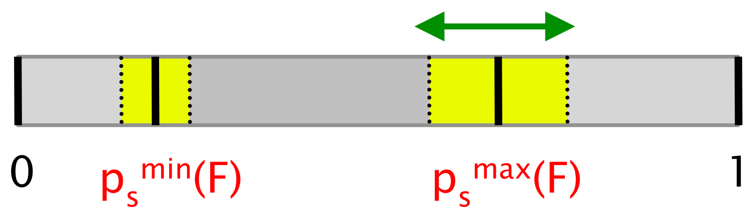


  - yields lower/upper bounds for min/max probabilities



Abstraction refinement

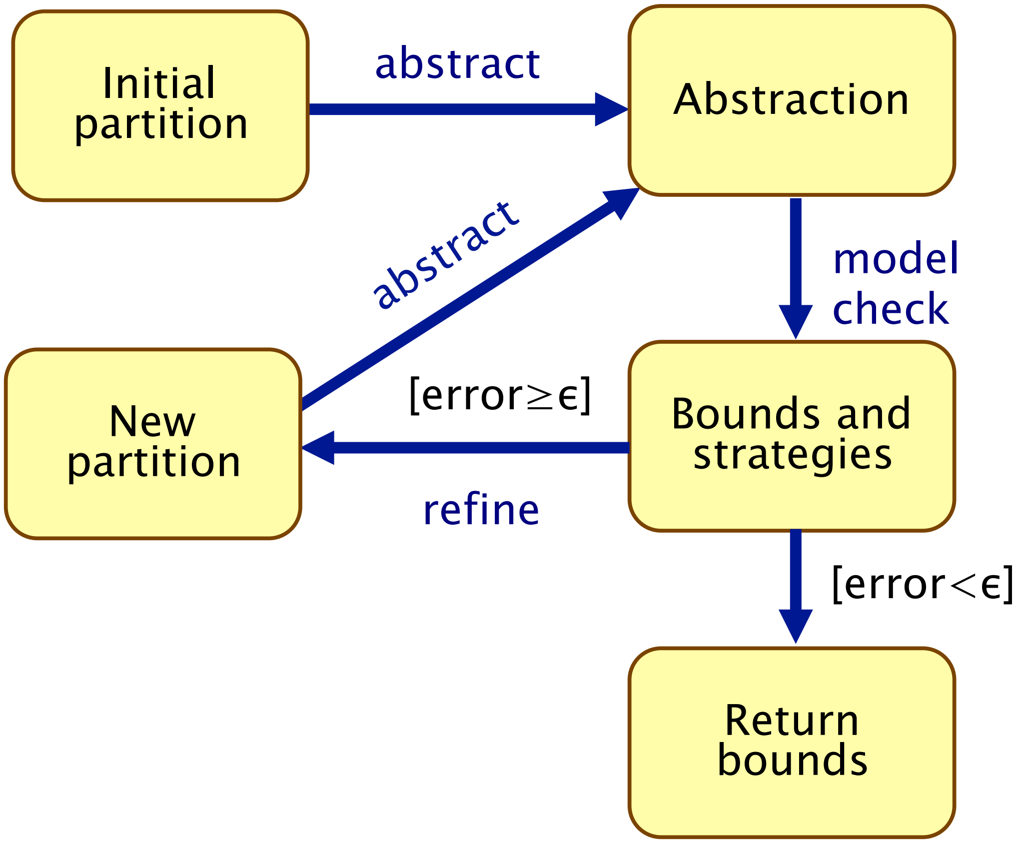
- Consider (max) difference between lower/upper bounds
  - gives a quantitative measure of the abstraction’s precision



- If the difference (“error”) is too great, refine the abstraction
  - a finer partition yields a more precise abstraction
  - lower/upper bounds can tell us where to refine (which states)
  - (memoryless) strategies can tell us how to refine

Abstraction-refinement loop

- Quantitative abstraction-refinement loop for MDPs

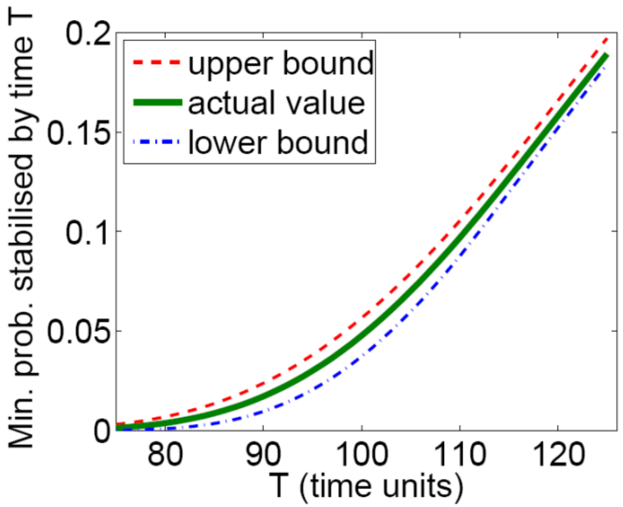


- Refinements yield strictly finer partition
- Guaranteed to converge for finite models
- Guaranteed to converge for infinite models with finite bisimulation

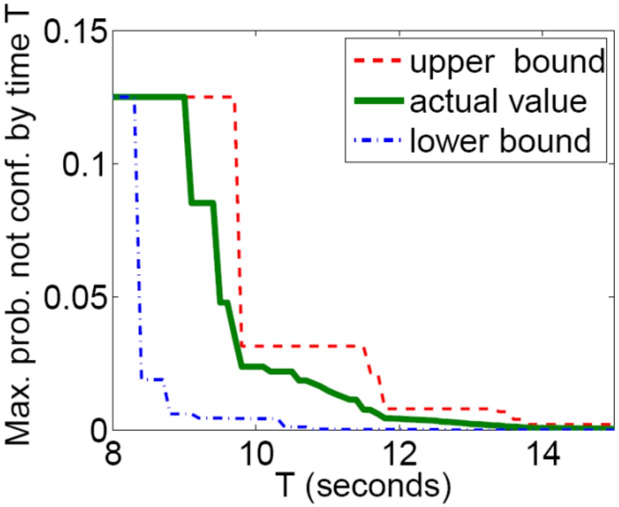
Abstraction refinement: Applications

- Examples (MDPs):

IJ90 self stabilisation alg.
(1,048,575 states abstracted to 627)



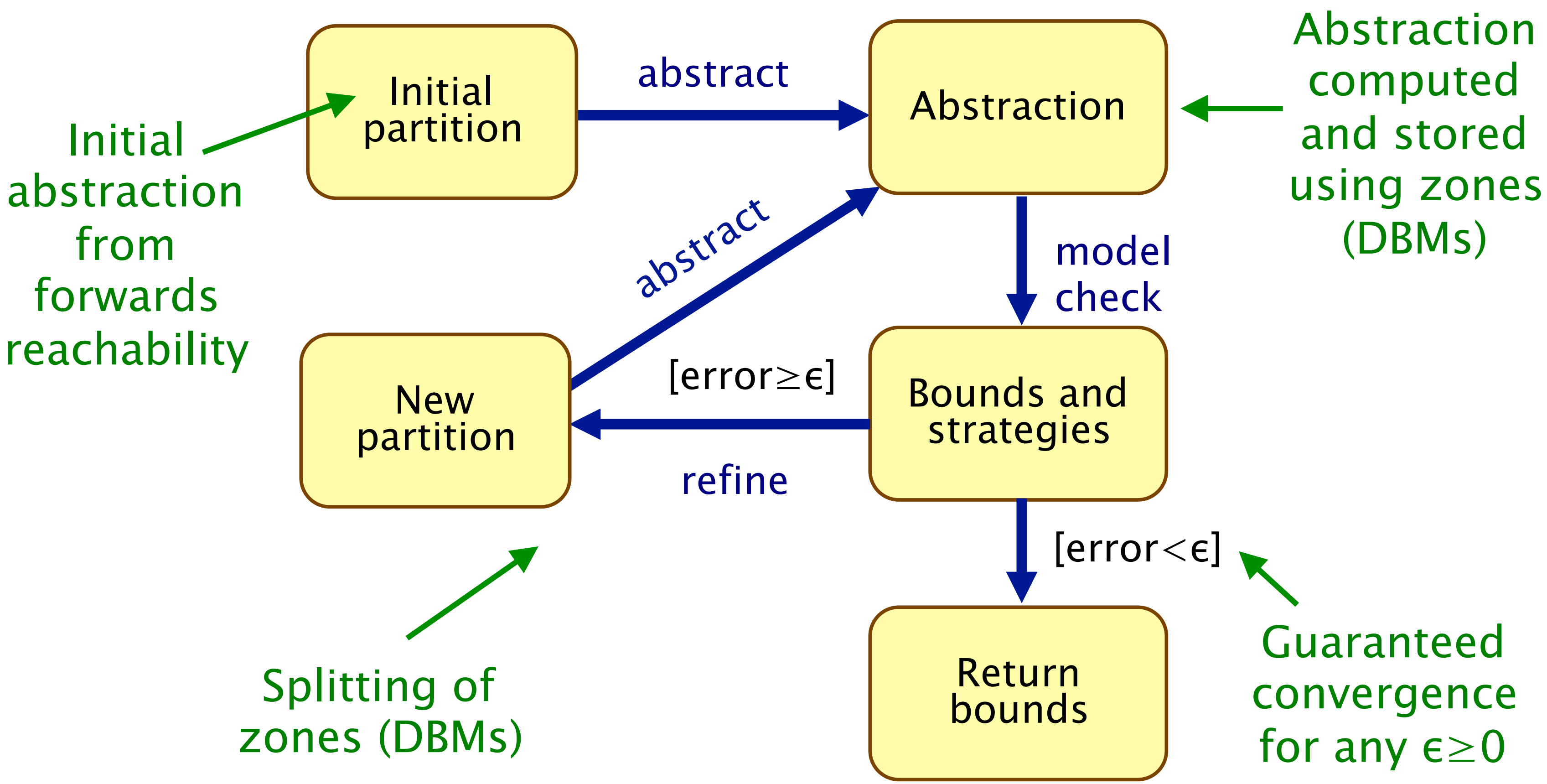
Zeroconf protocol
(838,905 states abstracted to 881)



- Applications
  - probabilistic software (C + probabilities) [qprover] [KKNP10]
  - concurrent probabilistic programs [PASS] [HHWZ10b]
  - probabilistic timed automata (exact) [PRISM] [KNP09c]

Abstraction refinement for PTAs

- Model checking for PTAs using abstraction refinement



Abstraction refinement for PTAs

- Computes reachability probabilities in PTAs
  - minimum or maximum, exact values ("error" $\epsilon=0$)
  - also time-bounded reachability, with extra clock

- In practice, performs very well
  - implemented in PRISM (using DBMs)
  - faster than digital clocks or backwards on large example set
  - (sometimes by several orders of magnitude)
  - handles larger PTAs than the digital clocks approach

Overview

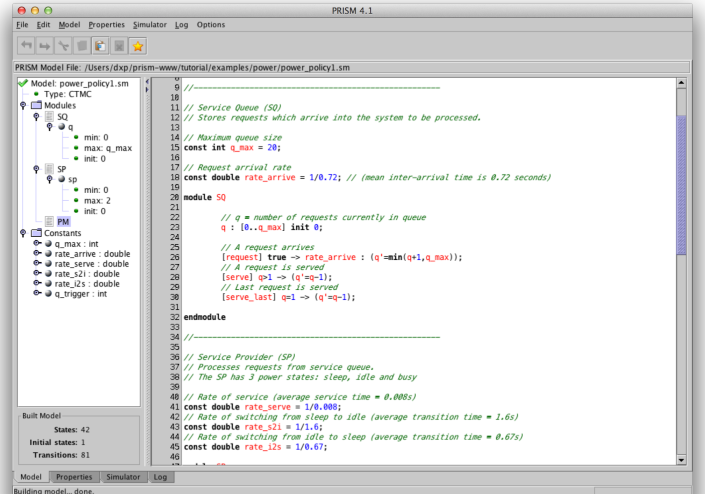
- Probabilistic model checking
  - example: FireWire protocol
- Probabilistic timed automata (PTAs)
  - clocks, zones, syntax, semantics
  - property specification
- Verification techniques for PTAs
  - region graphs + digital clocks + zone-based methods
  - abstraction-refinement
- Tool support: PRISM
- Verification vs. controller synthesis
  - example: task-graph scheduling

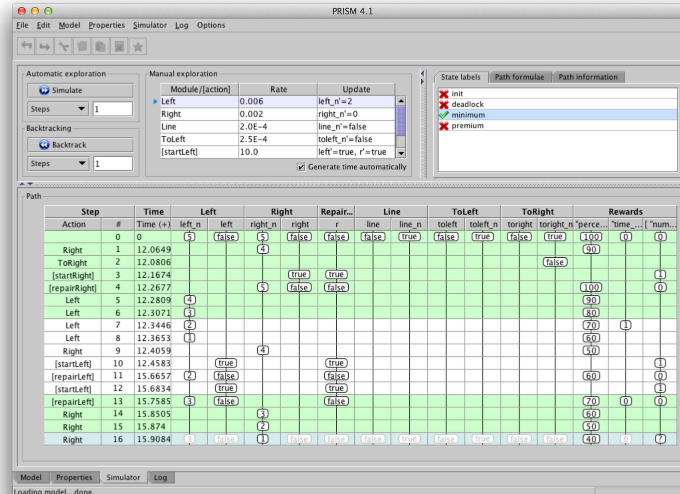
- See: www.prismmodelchecker.org/lectures/movep14/
  - slides, tutorial papers, reference list, ...

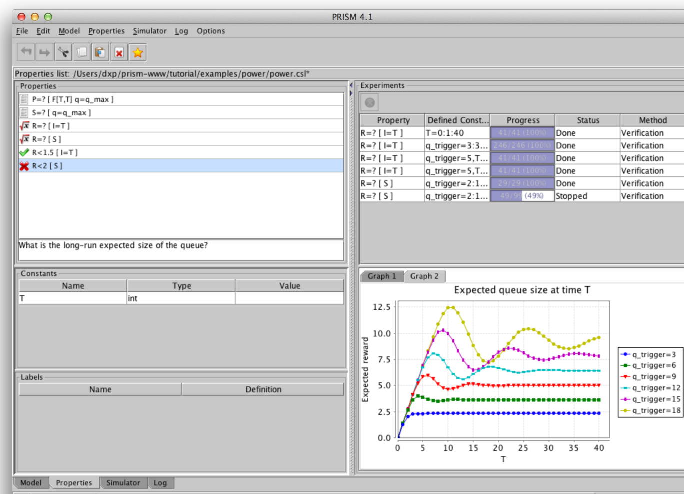
The PRISM tool

- PRISM: Probabilistic symbolic model checker
  - developed at Birmingham/Oxford University, since 1999
  - free, open source (GPL), runs on all major OSs
- Support for:
  - models: DTMCs, CTMCs, MDPs, PAs, PTAs
  - (see also PRISM-games: stochastic multi-player games)
  - properties: PCTL, CSL, LTL, PCTL*, costs/rewards, numerical extensions, multi-objective, …
- 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/

The PRISM tool

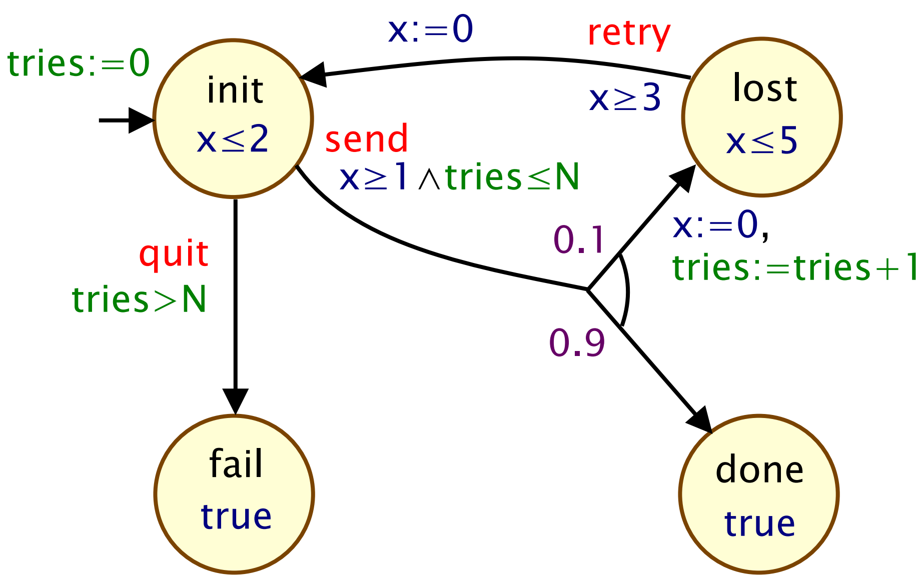






Modelling PTAs in PRISM

- 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:

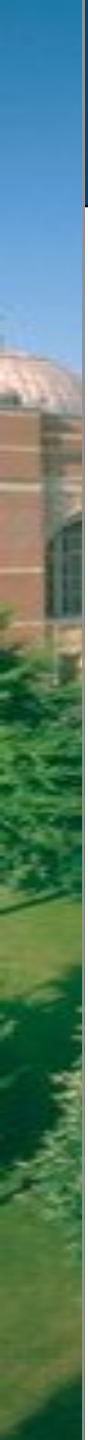
- rewards (i.e. costs, prices)
- parallel composition

PRISM – Case studies

PTA

- 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, non-repudiation, …
- Planning & controller synthesis
  - robotics, dynamic power management, task-graph scheduling
- Performance & reliability
  - nanotechnology, cloud computing, manufacturing systems, …
- Biological systems
  - cell signalling pathways, DNA computation, pacemakers, …

- See: www.prismmodelchecker.org/casestudies



Overview

- Probabilistic model checking
  - example: FireWire protocol
- Probabilistic timed automata (PTAs)
  - clocks, zones, syntax, semantics
  - property specification
- Verification techniques for PTAs
  - region graphs + digital clocks + zone-based methods
  - abstraction-refinement
- Tool support: PRISM
- Verification vs. controller synthesis
  - example: task-graph scheduling

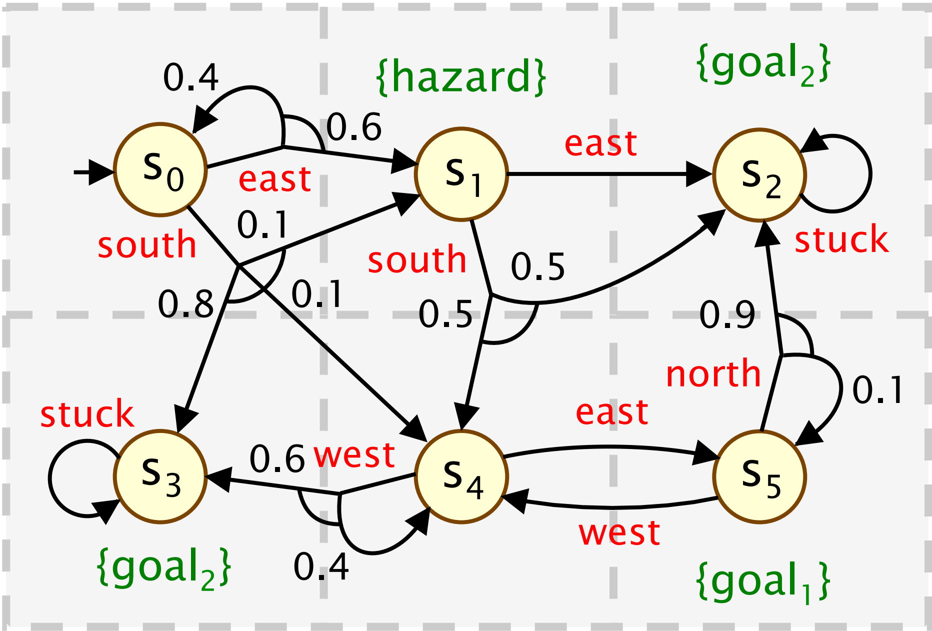
- See: www.prismmodelchecker.org/lectures/movep14/
  - slides, tutorial papers, reference list, …

Verification vs. Controller synthesis

- Verification vs. synthesis
  - verification = check that a (model of) system satisfies a specification of correctness
  - synthesis = build a "correct-by-construction" system directly from a specification of correctness
- Controller synthesis (for MDPs)
  - generate a controller/scheduler (an adversary) that chooses actions such that a correctness specification is satisfied
  - dual problem to verification on MDPs
- For example: $P_{<0.01}$[F err]
  - verification: “the probability of an error is always $<$ 0.01”
  - controller synthesis: “does there exist a controller (adversary) for which the probability of an error occurring is $<$ 0.01?”
  - or, optimise: “what is the minimum probability of an error?”

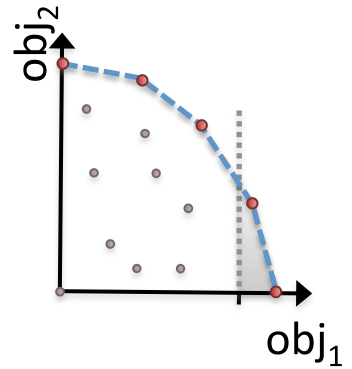
Controller synthesis

- Controller synthesis (for MDPs)
  - nondeterminism: actions available to controller
  - probability: uncertainty about environment's behaviour
- For example: robot controller



Controller synthesis: Extensions

- Multi-objective probabilistic model checking
  - investigate trade-offs between conflicting objectives
  - e.g. "is there a strategy such that the probability of message transmission is > 0.95 and expected battery life > 10 hrs?"
  - e.g. "maximum probability of message transmission, assuming expected battery life-time is > 10 hrs?"
  - e.g. "Pareto curve for maximising probability of transmission and expected battery life-time"

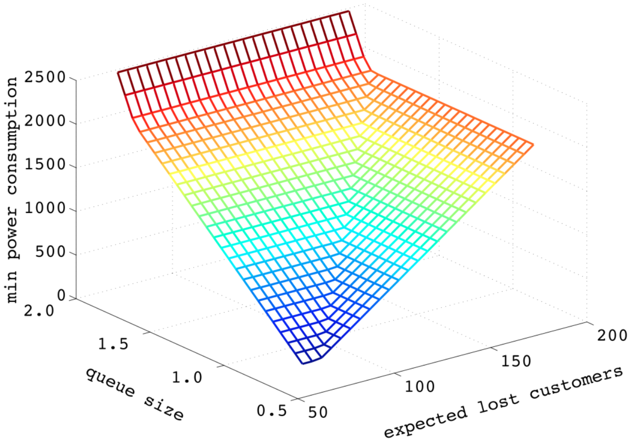


- Controller synthesis with stochastic games
  - player 1 = controller (as for MDPs)
  - player 2 = environment ("uncontrollable" actions)
- Multi-strategies
  - strategies (adversaries) which can choose between multiple actions at each time step

Controller synthesis – Applications

- Examples of PRISM-based controller synthesis

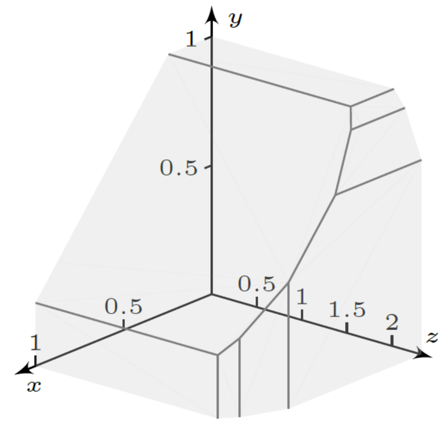
Synthesis of dynamic power management controllers [FKN+11]



Minimise energy consumption, subject to constraints on:
(i) expected job queue size;
(ii) expected number of lost jobs

Motion planning for a service robot using LTL [LPH14b]

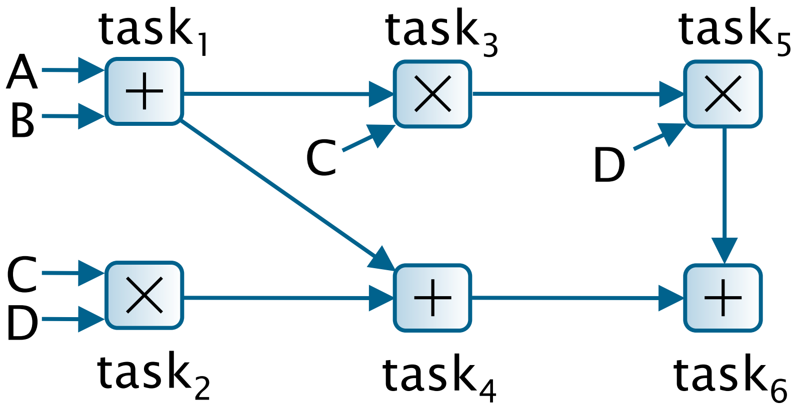
Synthesis of team formation strategies [CKPS11, FKP12]



Pareto curve:
x="probability of completing task 1";
y="probability of completing task 2";
z="expected size of successful team"

Example: Task-graph scheduling

- Use probabilistic model checking of PTAs to solve scheduling problems, e.g. for a task-graph
  - task-graph = tasks to complete + dependencies/ordering
  - for ex.: real-time scheduling, embedded systems controllers
- Simple example: [adapted from BFLM11]
  - evaluate expression: D×(C×(A+B))+((A+B)+(C×D))
  - with subterms evaluated on one of two processors, P_1 or P_2

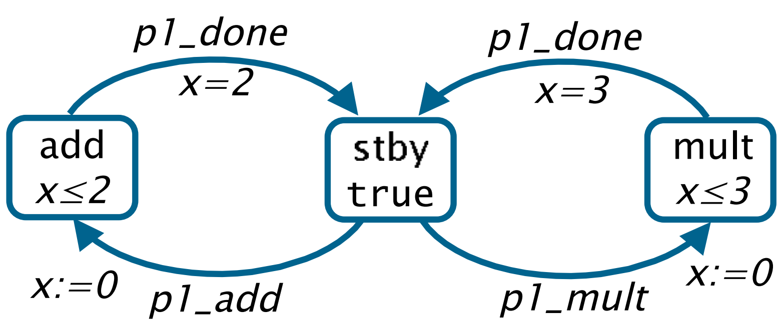


	P_1	P_2
+	2 picoseconds	5 picoseconds
×	3 picoseconds	7 picoseconds
idle	10 Watts	20 Watts
active	90 Watts	30 Watts

Example: Task-graph scheduling

- Task-graph scheduling
  - aim to find optimal (time, energy usage, etc.) schedulers
  - successful application of (non-probabilistic) timed automata
  - PTAs allow us to reason about uncertain delays + failures
  - optimal scheduler derived from optimal adversary

- PTA model
  - parallel composition of 3 PTAs: one scheduler, two processors
  - for example, processor P_1, with local clock x:



Locations also labelled with costs/rewards for time/energy usage

Example: Task-graph scheduling

- Property specification:
  - $R^{time}_{min=?}$ [F complete] – minimise (expected) time
  - $R^{energy}_{min=?}$ [F complete] – minimise (expected) energy usage
- Model check with PRISM (digital clocks)
  - and extract optimal adversary/scheduler
- Time optimal (12 picoseconds)

<table>
<tr><td>time</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td></tr>
<tr><td>P_1</td><td colspan="2">task1</td><td colspan="3">task3</td><td colspan="3">task5</td><td colspan="2">task4</td><td colspan="2">task6</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>P_2</td><td colspan="7">task2</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>

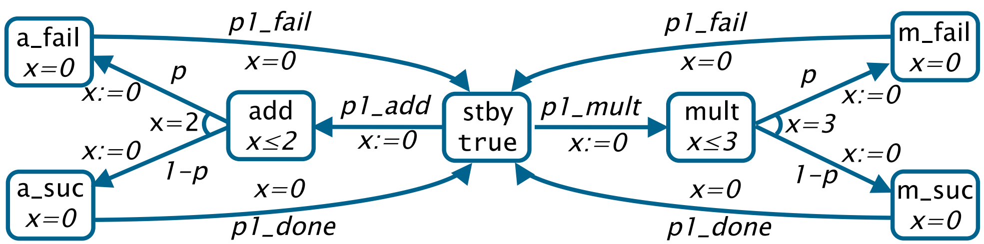
- Energy optimal (1.32 nanojoules)

<table>
<tr><td>time</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td></tr>
<tr><td>P_1</td><td colspan="2">task1</td><td colspan="3">task3</td><td colspan="2">task4</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>P_2</td><td colspan="7">task2</td><td colspan="7">task5</td><td colspan="5">task6</td><td></td></tr>
</table>

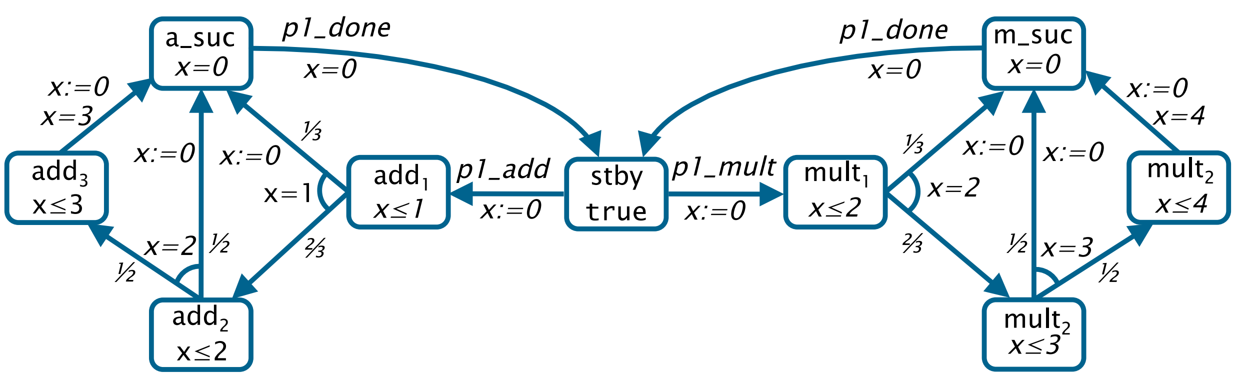
- No probabilities yet…

Adding probabilities

- Faulty processors
  - add third processor P_3: faster, but may fail to execute task



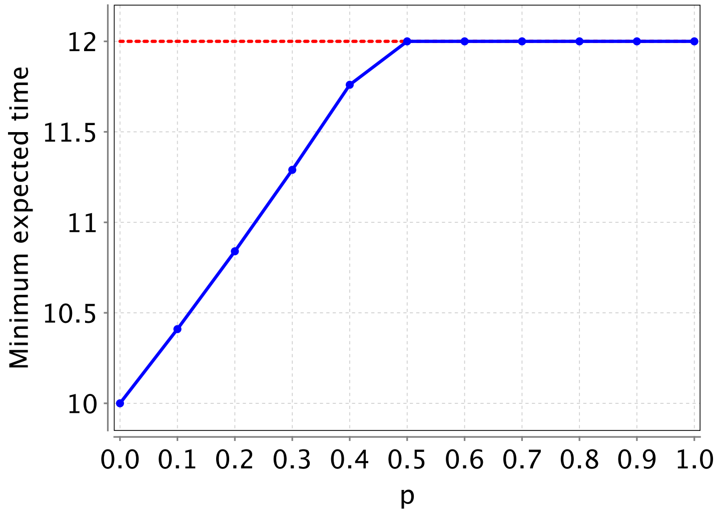
- Probabilistic task execution times
  - simple example: (deterministic) delay of 3 in processor P_1 replaced by distribution: ⅓:2, ⅓:3, ⅓:4



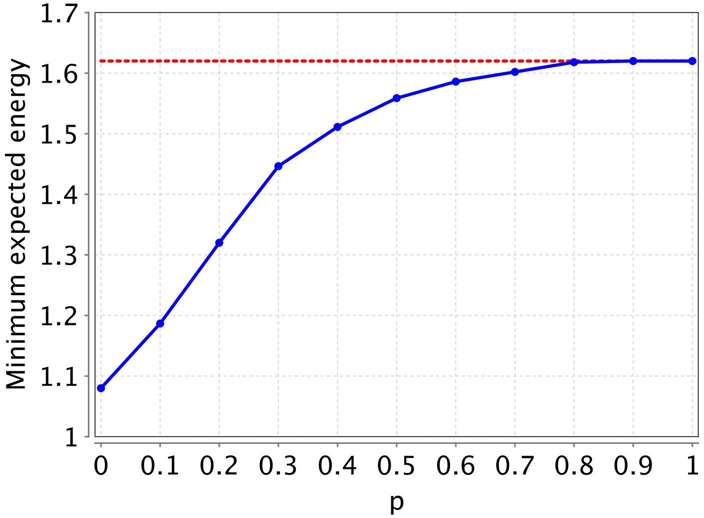
Results (with faulty processor)

- Compute optimal (time/energy) schedulers
  - (using same properties as before)
- Results (for varying failure rates p of processor P_3):
  - dotted red line shows original results (no failures)
  - conclusion: better performance for low values of failure probability p; no benefit for higher values

Expected time

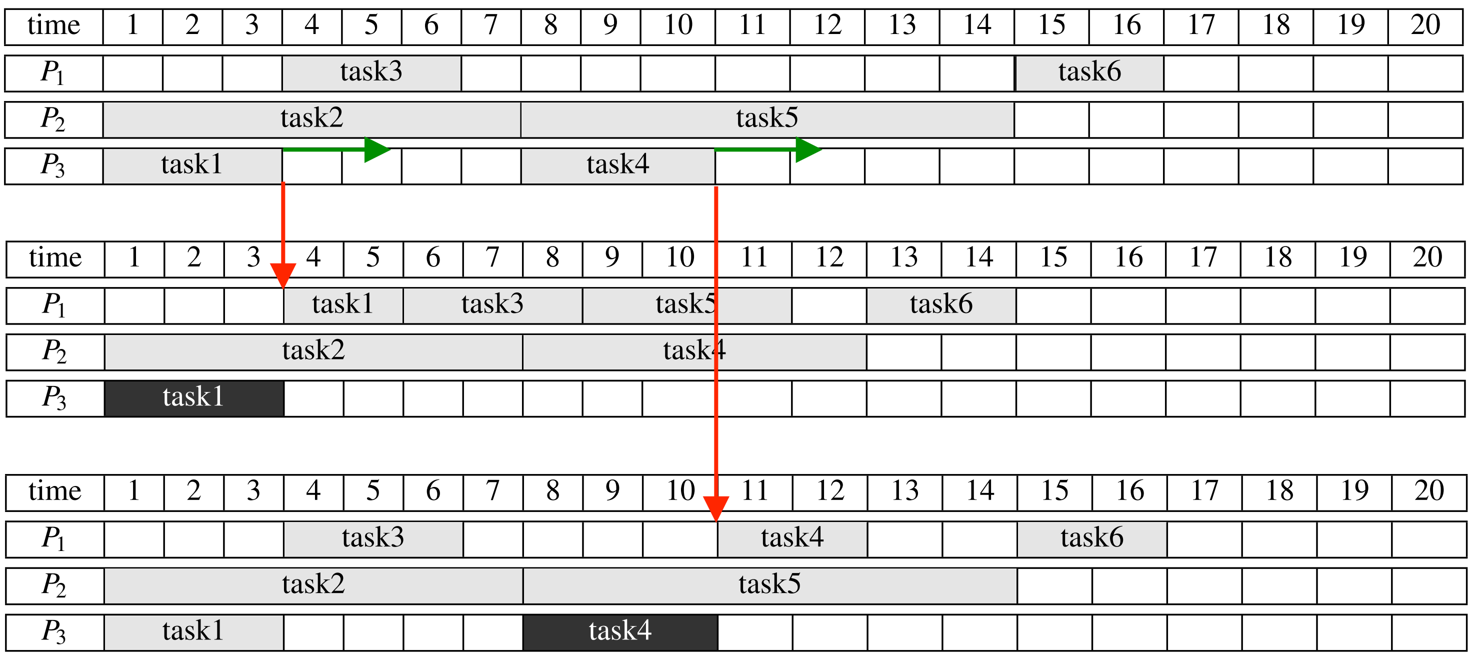


Expected energy usage



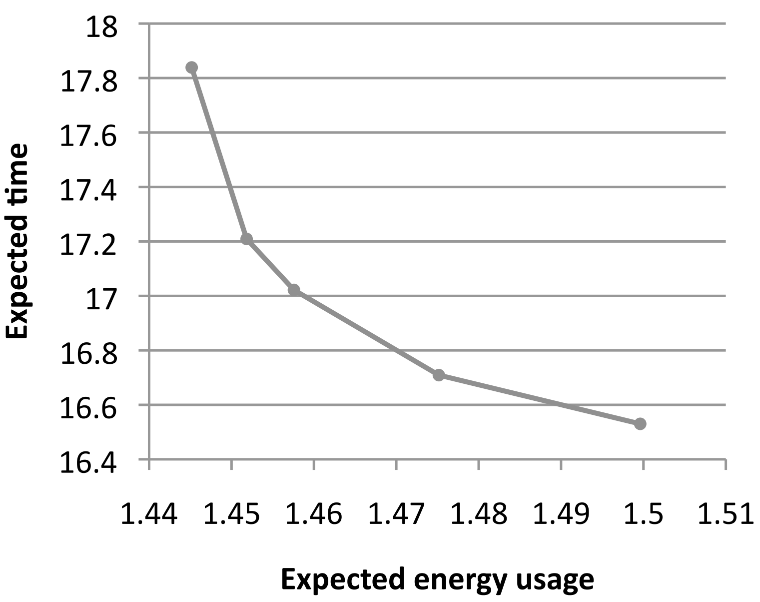
Schedulers (with faulty processor)

- Example (for p=0.5)
  - optimal scheduler to minimise energy consumption
- Optimal scheduler again obtained from adversary
  - now, behaviour depends on outcome of task execution



Multi-objective properties

- Multi-objective controller synthesis
  - (on MDP generated via digital clocks approach)
  - explore trade-off between time/energy usage
- Properties
  - e.g. minimise expected time, subject to bound on energy
  - or: Pareto curve for two objectives: time/energy
  - NB: both may generate randomised schedulers

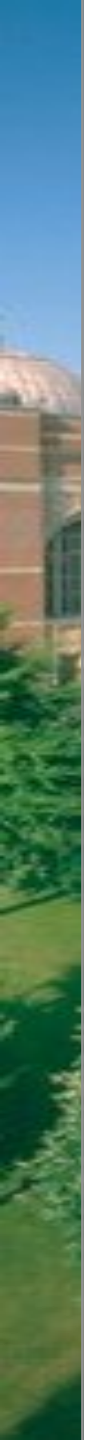


Overview

- Probabilistic model checking
  - probabilistic real-time systems

- Probabilistic timed automata (PTAs)
  - probability + nondeterminism + (dense) time
  - property specification; PTCTL, PCTL, …

- Model checking techniques for PTAs
  - region graphs + digital clocks
  - zone-based methods + abstraction-refinement
  - tool support: PRISM
  - verification vs. controller synthesis



Thanks for your attention

More info here:

www.prismmodelchecker.org/lectures/movep14/