Quantitative verification techniques for probabilistic software

Marta Kwiatkowska

Oxford University Computing Laboratory

Summer School on Model Checking, Beijing, October 2010

Course overview

- 3 sessions (Mon/Tue/Thur): 6 $\times$ 50 minute lectures
  - 1: Markov decision processes (MDPs)
  - 2: Probabilistic LTL model checking
  - 3: Compositional probabilistic verification
  - 4: Abstraction, refinement and probabilistic software
  - 5: Probabilistic timed automata (PTAs)
  - 6: Software with time and probabilities
- For additional background material
  - and an accompanying list of references
  - see: http://www.prismmodelchecker.org/lectures/

Part 6

Software with
time and probabilities

Overview (Part 6)

- Model checking for PTAs
  - recap, summary
  - zone-based approaches:
  - (i) forwards reachability
  - (ii) backwards reachability
  - (iii) game-based abstraction refinement

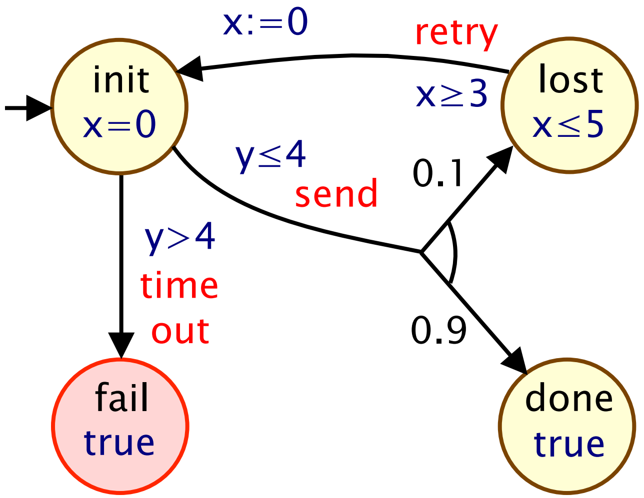
- Verifying software with time and probabilities
  - probabilistic timed programs (PTPs)
  - verifying PTPS with abstraction + refinement

- Looking ahead: Quantitative verification of SystemC

Recap: Probabilistic timed automata

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

- Like timed automata
  - all clocks increase at same rate
  - clocks can be reset (to zero)

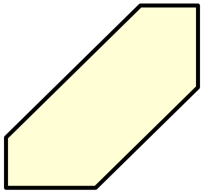


- PTA model checking
  - the semantics of a PTA is an infinite-state MDP
  - probabilistic (timed) reachability is defined as for MDPs
  - but computation is more complex...

Recap: Zones

- 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 $c, d \in \mathbb{N}$
  - zone defines a set of clock valuations, i.e. a subset of $\mathbb{R}^X$
  - used for both syntax of PTAs/properties and algorithms

- Can be efficiently represented/manipulated
  - using difference bound matrices (DBMs)

- Operations:
  - intersection, union, difference, resets, projections
  - (some preserve convexity, some do not)

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]

Zone-based approaches

- An alternative is to use zones to construct an MDP
  - similar to classical timed automata techniques

- Conventional symbolic model checking relies on computing
  - **post**(S'): states reached from a state in S' in a single step
  - **pre**(S'): states that can reach S' in a single step

- Extend these operators to include time passage
  - **dpost**[e](S'): states that can be reached from a state in S' by traversing the edge e
  - **tpost**(S'): states that can be reached from a state in S' by letting time elapse
  - **pre**[e](S'): states that can reach S' by traversing the edge e
  - **tpre**(S'): 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**$(l,\zeta) = (l, \nearrow\zeta \wedge inv(l))$
  - $\nearrow\zeta$ can be reached from ζ by letting time pass
  - $\nearrow\zeta \wedge inv(l)$ must satisfy the invariant of the location l

- **tpre**$(l,\zeta) = (l, \swarrow\zeta \wedge inv(l))$
  - $\swarrow\zeta$ 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

- First step: forwards exploration of PTA
  - using **dpost**[e](l,ζ) and **tpost**(l,ζ)
  - to ensure termination, need to take c-closure of each zone encountered (c is the largest constant in the PTA)
  - resulting state space is a set of zones S_F

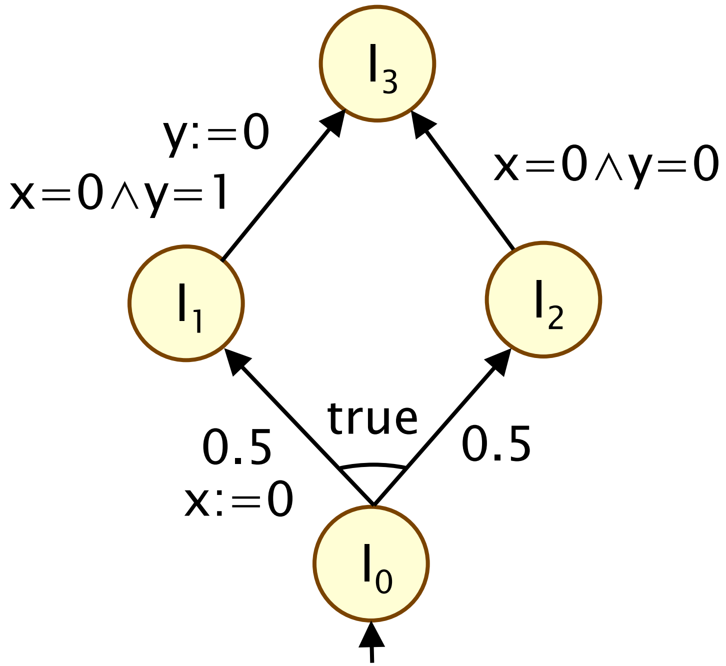
- Second step: construct finite state MDP
  - $(S_F, (l_{init},\underline{0}), Act, \delta_F, L_F)$
  - $L_F(l,\zeta) = L(l)$ for all $(l,\zeta) \in S_F$
  - $((l,\zeta), a, \mu) \in \delta_F$ iff there exists a probabilistic edge (l,g,a,p) of PTA such that for any $(l', \zeta') \in Z$:

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

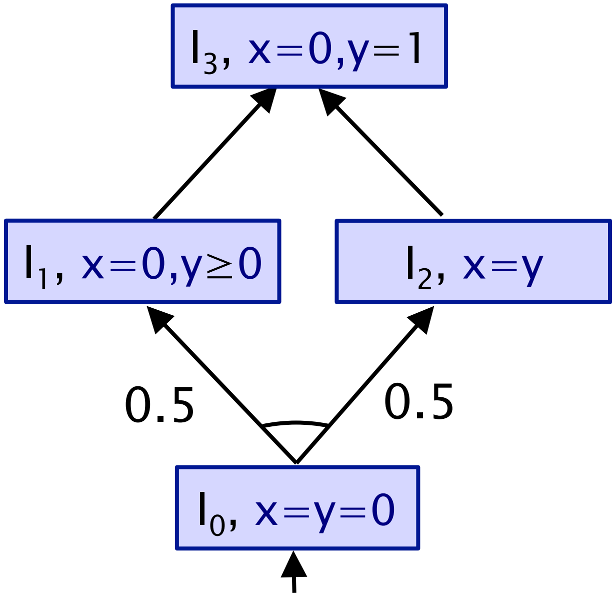
summation over all the edges of (l,g,a,p) such that applying **post** to (l,ζ) leads to the symbolic state (l',ζ')

Forwards reachability – Example

PTA:



MDP:



Forwards reachability – Limitations

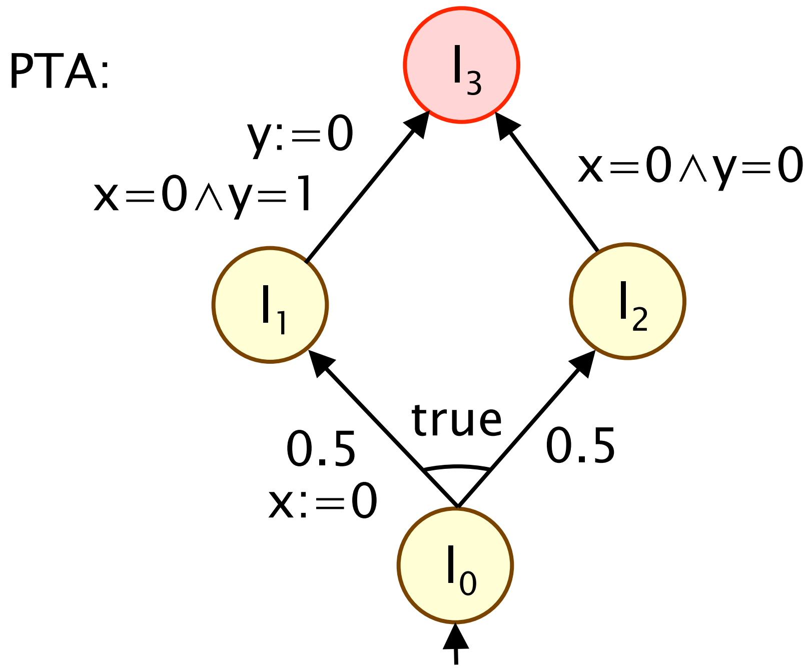
- Problem reduced to analysis of finite-state MDP, but…

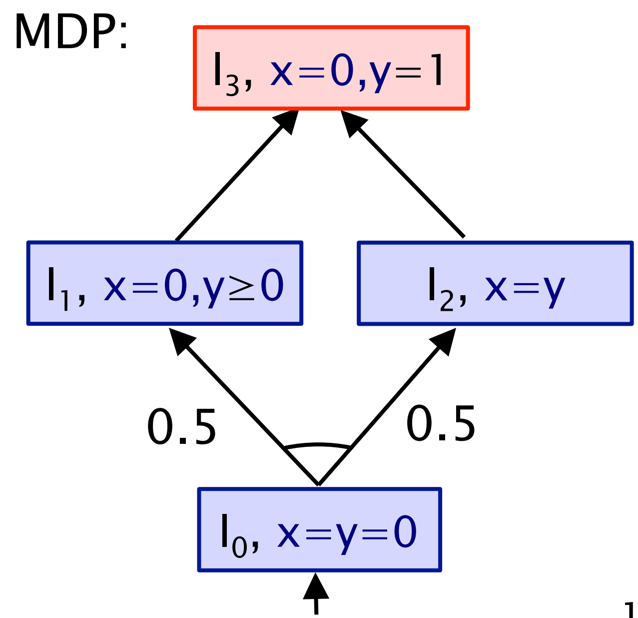
- Only obtain upper bounds on maximum probabilities
  - caused by when edges are combined

- 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, maximum probability in the MDP 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 (Part 6)

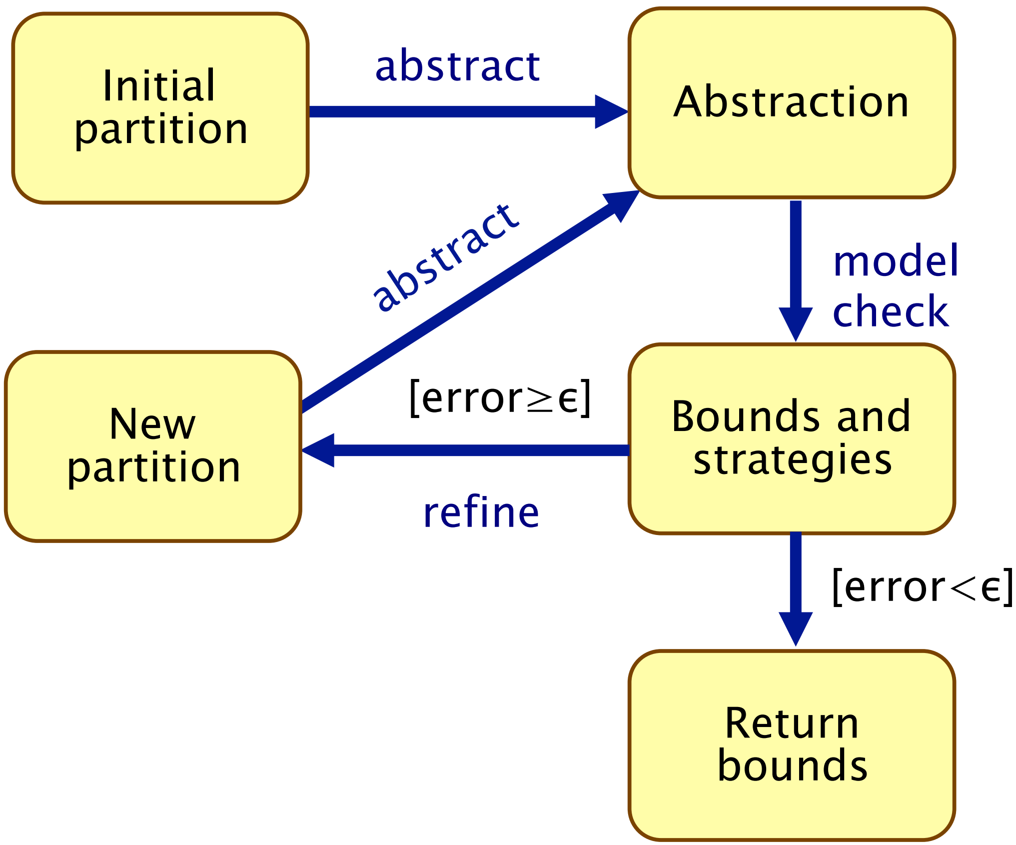
- Model checking for PTAs
  - recap, summary
  - zone-based approaches:
  - (i) forwards reachability
  - (ii) backwards reachability
  - (iii) game-based abstraction refinement

- Verifying software with time and probabilities
  - probabilistic timed programs (PTPs)
  - verifying PTPS with abstraction + refinement

- Looking ahead: Quantitative verification of SystemC

Recap: Abstraction-refinement loop

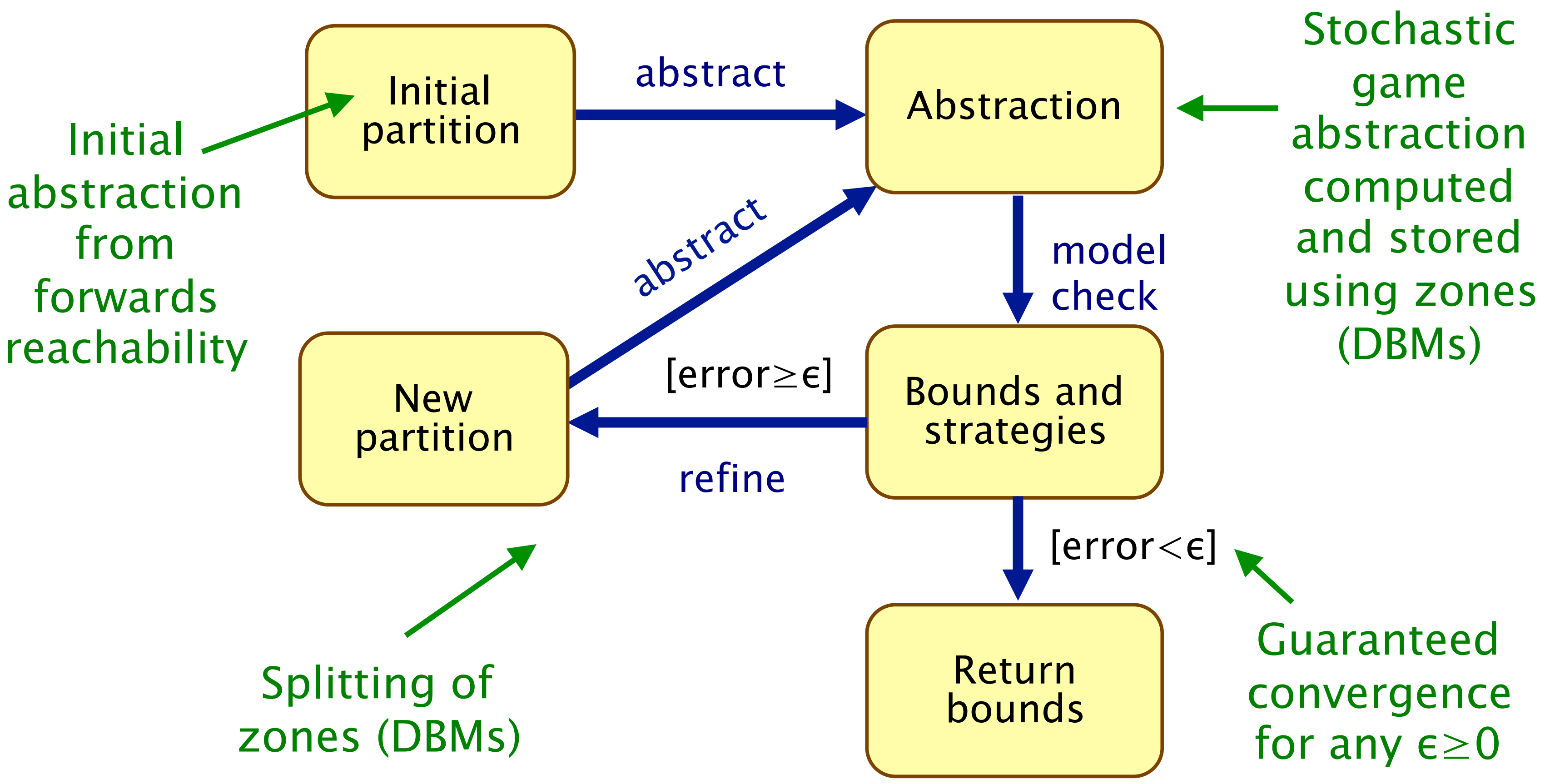
- Quantitative abstraction-refinement loop for MDPs
  - based on abstractions of MDPs as stochastic games



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

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
- Integrated in PRISM (development release)
  - PRISM modelling language extended with clocks
  - implemented using DBMs
- In practice, performs very well
  - faster than digital clocks or backwards on large example set
  - (sometimes by several orders of magnitude)
  - handles larger PTAs than the digital clocks approach
- And: use of abstraction allows exension to other models…

Overview (Part 6)

- Model checking for PTAs
  - recap, summary
  - zone-based approaches:
  - (i) forwards reachability
  - (ii) backwards reachability
  - (iii) game-based abstraction refinement

- Verifying software with time and probabilities
  - probabilistic timed programs (PTPs)
  - verifying PTPS with abstraction + refinement

- Looking ahead: Quantitative verification of SystemC

Probabilistic timed programs

- Probabilistic timed programs (PTPs)
  - probability, nondeterminism and real-time and data
  - probabilistic timed automata + discrete-valued variables

- Time – assume a finite set X of real-valued clocks
  - Zones(X) is the set of zones ζ over X
  - i.e. $\zeta ::= x \leq d \mid c \leq x \mid x+c \leq y+d \mid \neg\zeta \mid \zeta \vee \zeta$
  - where $x, y \in X$ and $c, d \in \mathbb{N}$

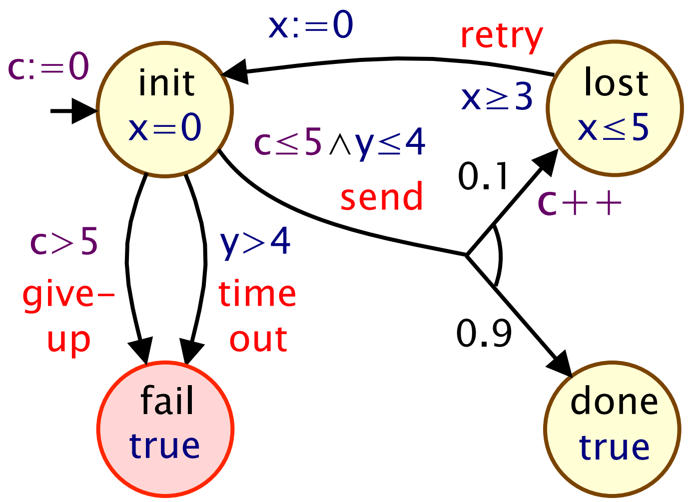
- Data – assume a finite set D of data variables
  - Val(D) is the set of all valuations of D
  - Pred(D) is the set of predicates over D
  - Up(D) is the set of all update functions over D
  - i.e. set of all functions $up : Val(D) \rightarrow Val(D)$

Probabilistic timed programs

- A PTP is a tuple (L, l_{init}, D, u_{init}, X, Act, inv, enab, prob)
  - L = locations, D = data variables, X = clocks, Act = actions
  - $l_{init} \in$ L is initial location and $u_{init} \in$ Val(D) is initial valuation
  - inv : L → Zones(X) is the invariant condition
    - clocks X must satisfy inv(l) whilst in location l
  - enab : L×Act → Pred(D) × Zones(X) is the enabling condition
    - guard for action a in location l split into $enab_D$(l,a) and $enab_X$(l,a)
    - can only take action a in l if $enab_D(l,a) \wedge enab_X(l,a)$
  - prob : L×Act → Dist(Up(D) × 2^X × L)
    is the probabilistic transition function
    - if take action a in l, then with probability prob(l,a)(up,Y,l'):
    - update D according to up, reset clocks in Y⊆X, move to location l'

Example – PTP

- Simple communication protocol
  - aims to send a message over an unreliable channel
  - tries to send up to 5 times
  - or until time-out of 4 secs
  - delay between tries: 3–5 secs



- In the PTP:
  - L = {init, lost, done, fail}
  - D = {c} (c counts number of tries)
  - X = {x, y} (x for delay, y for timeout)
  - Act = {send, retry, giveup, timeout}

- Property of interest: maximum probability of reaching "fail"
  - actual max. probability is 0.1 (time-out after after 1 send)

Abstraction of PTPs

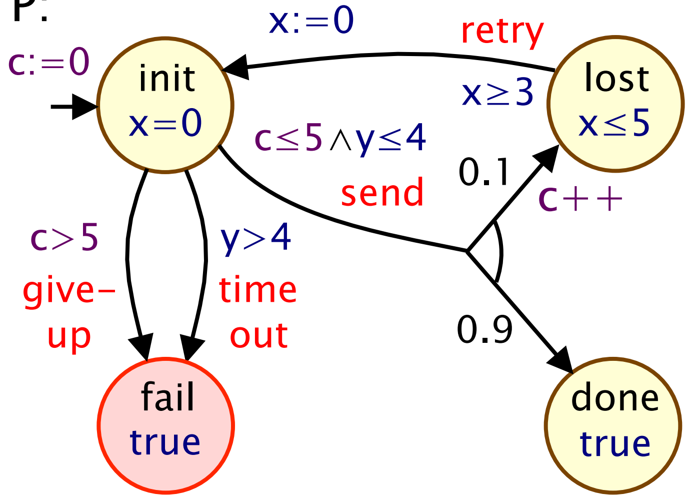
- Formal semantics of a PTP is an infinite-state MDP
  - over state space $L \times Val(D) \times \mathbb{R}^X$
  - data domain $Val(D)$ may be large/infinite; so need abstraction
  - time domain $\mathbb{R}$ is dense; so need abstraction
- In general, can use an abstract domain $((A, \sqcup, \sqcap, \sqsubseteq), \alpha, \gamma)$
  - lattice of abstract states, abstraction/concretisation functions
  - here, we use predicate abstraction for data and zones for time
  - i.e. abstract states are $(l, b, \zeta) \in L \times \{F,T\}^n \times Zones(X)$
  - assuming a set of data predicates $\Phi = \{\Phi_1, \ldots, \Phi_n\}$
  - (see [KNP10b] for details of other cases)
- We use (finite-state) stochastic games to abstract PTPs
  - i.e. state space is $L \times \{F,T\}^n \times Zones(X)$

Abstraction/refinement of PTPs

- 1. Build reachability graph for PTP
  - all reachable abstract states and possible transitions between
  - constructed through (classical) forwards reachability search
  - as in, for example, UPPAAL, but not on-the-fly
  - zone operations (DBMs) and SAT/SMT for symbolic post
- 2. Build stochastic game abstraction for PTP
  - i.e. of underlying infinite-state MDP semantics
  - constructed from reachability graph
  - further zone operations and/or SAT/SMT solving needed
  - yields lower/upper bound on reachability probabilities
- 3. Refine the abstraction (iteratively)
  - split zones, or generate new predicates

Example 1 – Abstraction

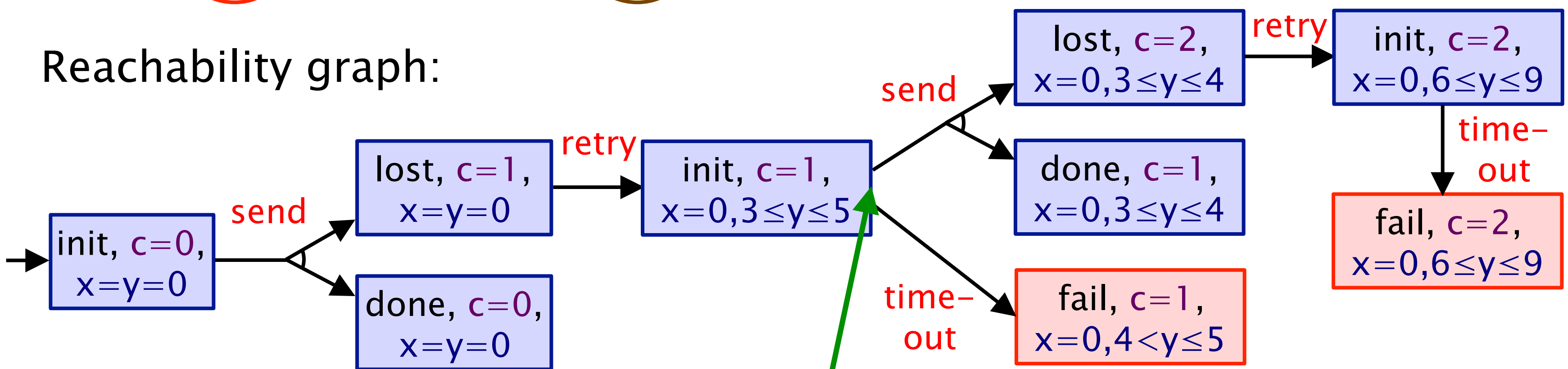
PTP:



In this example:

- just abstract time, not data
- i.e. abstract states are of the form:
- (l,d,ζ) ∈ L×Val(D)×Zones(X)

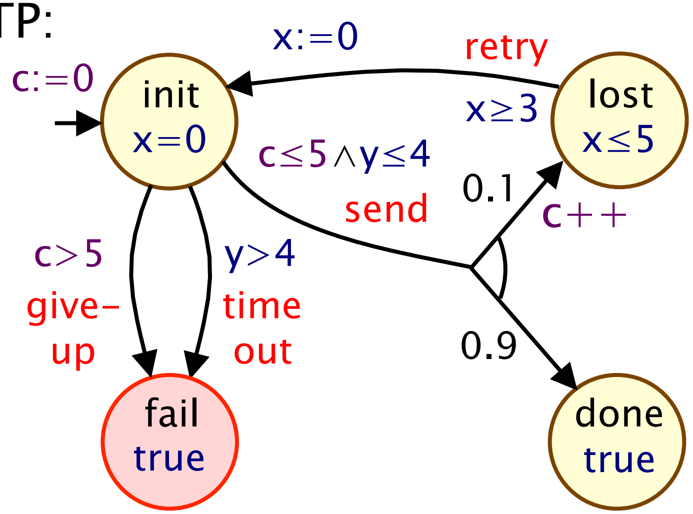
Reachability graph:



Actions send and time–out
are both enabled since
abstract state satisfies 3≤y≤5

Example 1 – Abstraction

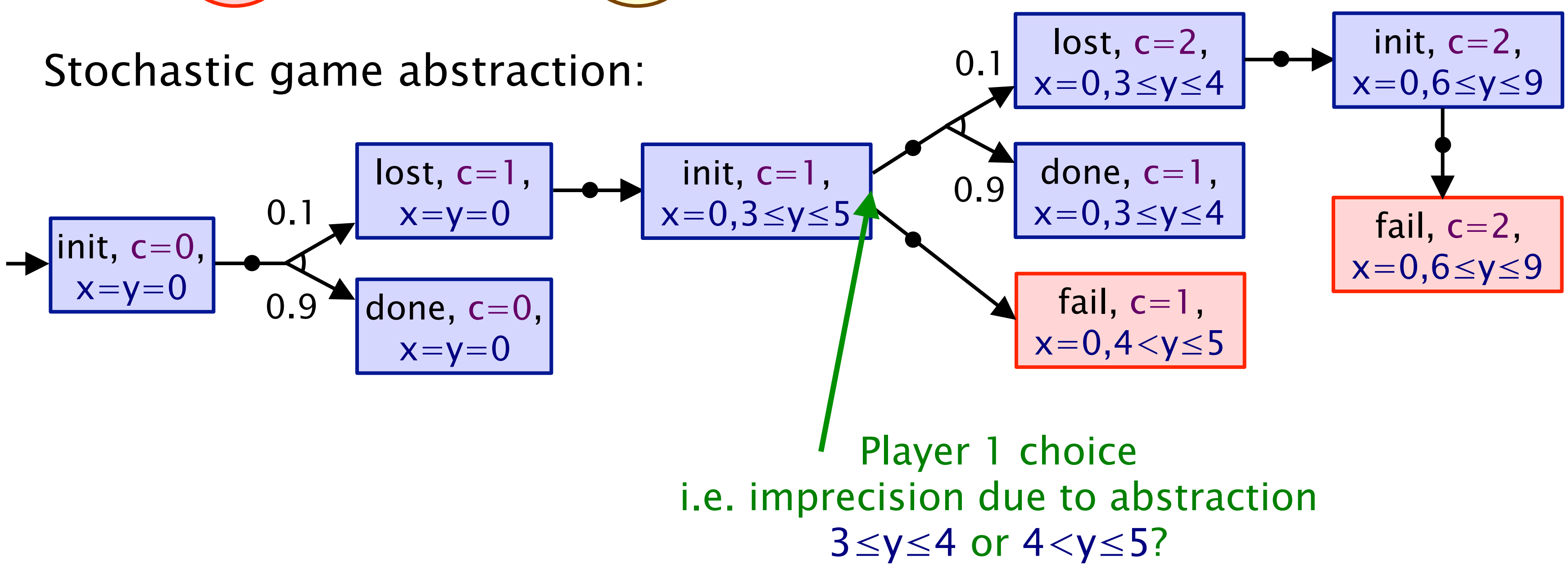
PTP:



Results:

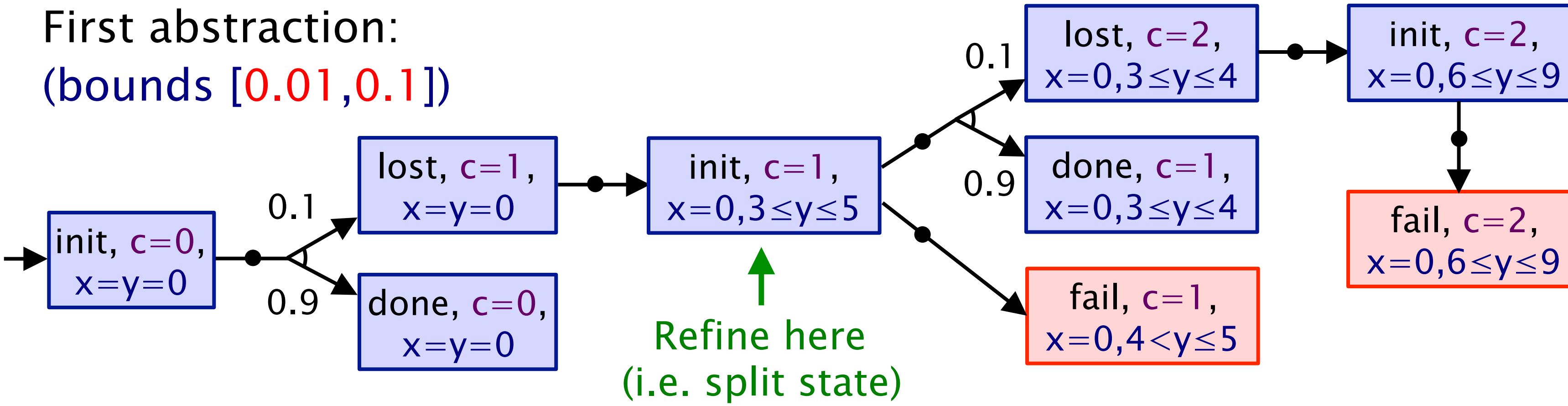
- max probability to reach fail?
- lower/upper bounds: [0.01,0.1]
- (in abstraction, can try to send either once or twice)

Stochastic game abstraction:

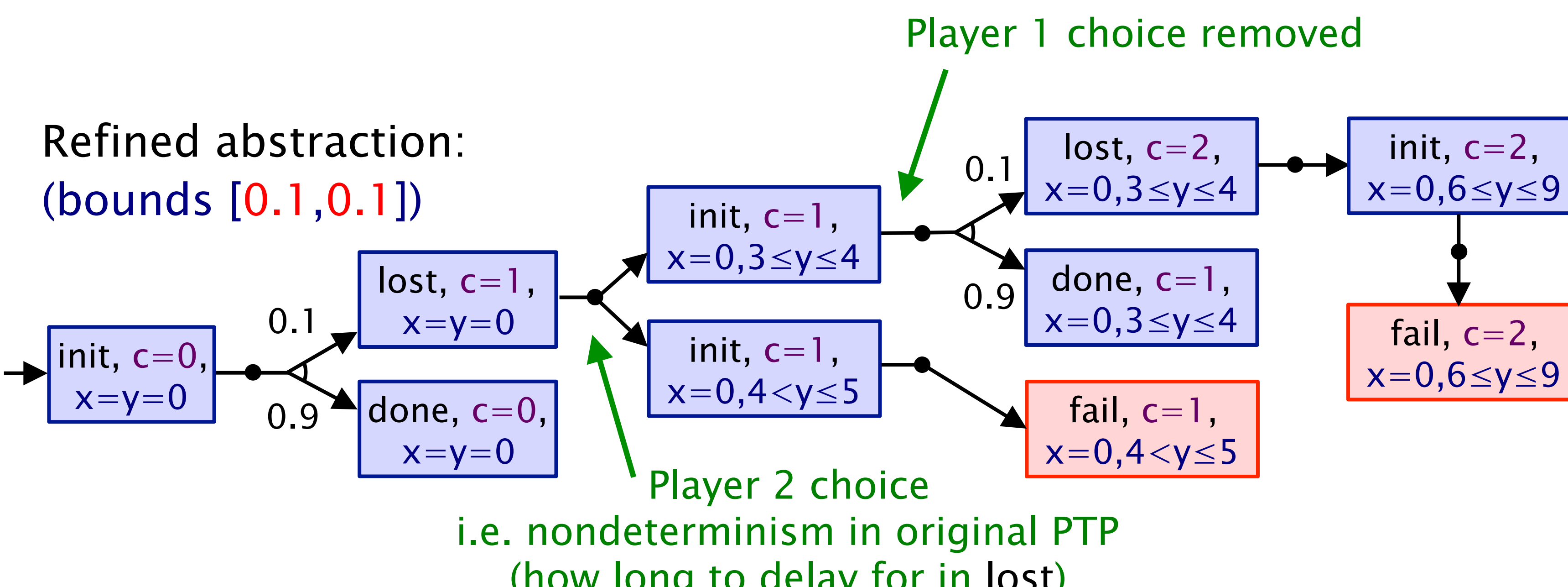


Example 1 – Refinement

First abstraction:
(bounds [0.01,0.1])

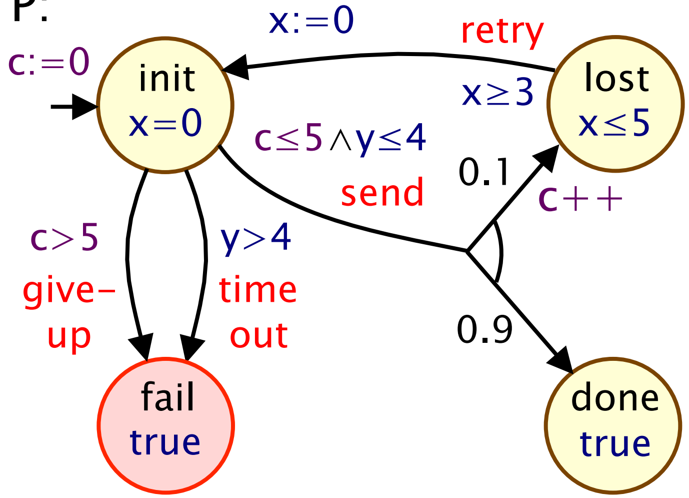


Refined abstraction:
(bounds [0.1,0.1])



Example 2 – Time and data

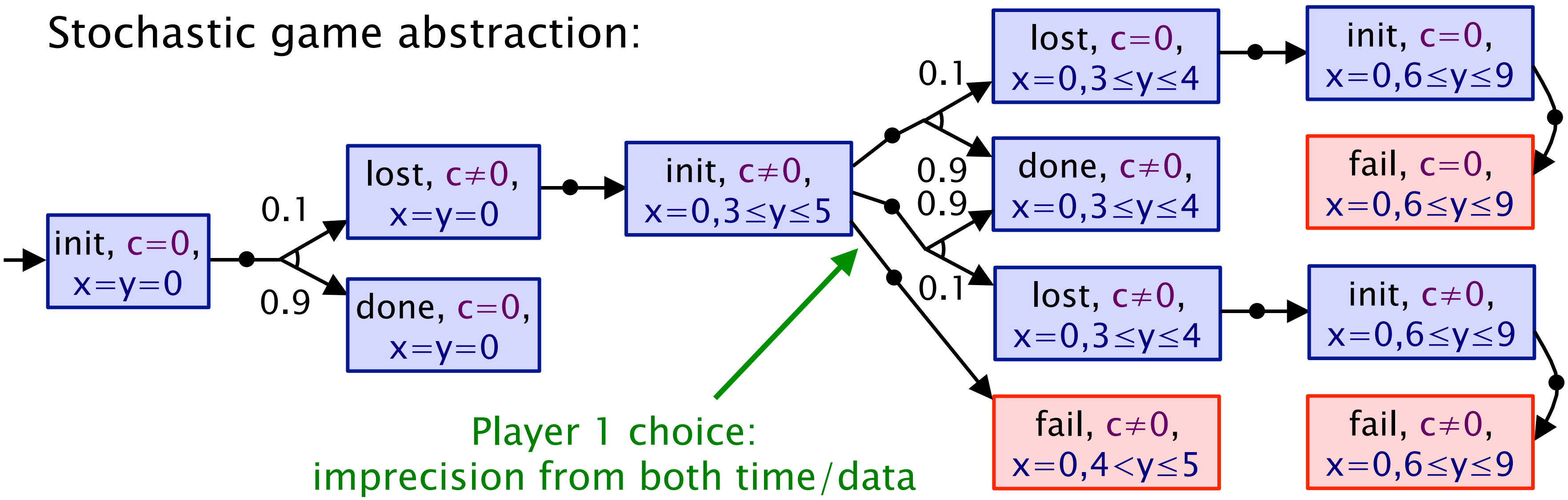
PTP:



In this example:

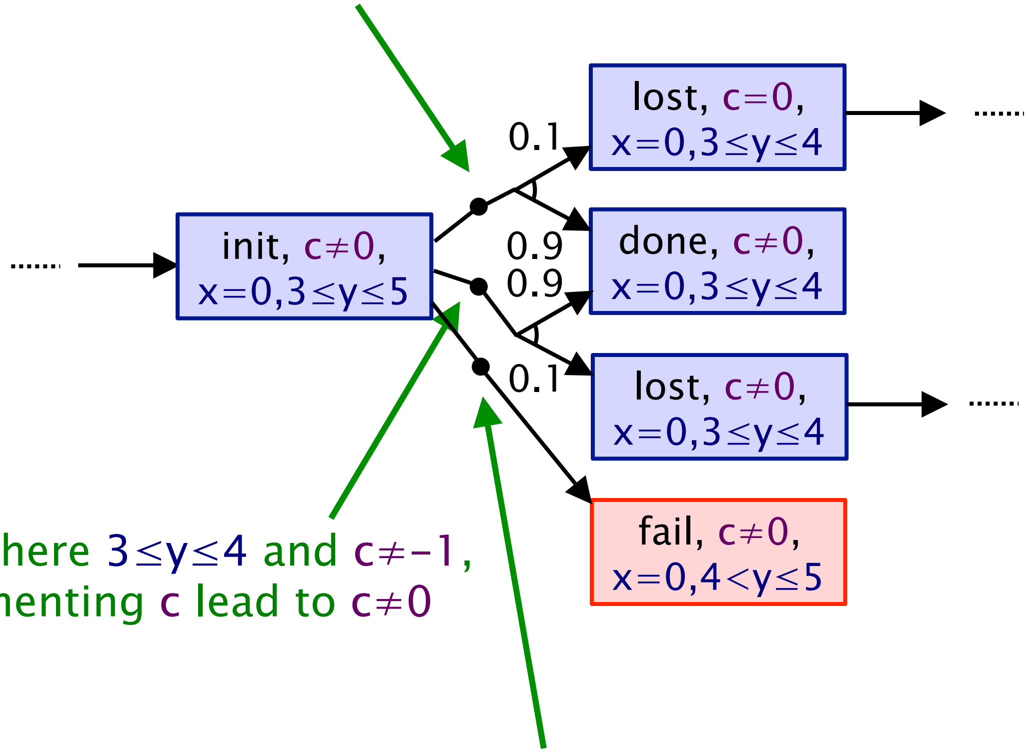
- abstract time *and* data
- i.e. abstract states are of the form:
- $(l,b,\zeta) \in L \times \{F,T\}^n \times Zones(X)$
- single data predicate: {c=0}

Stochastic game abstraction:



Example 2 – Time and data

States where $3 \leq y \leq 4$ and $c=-1$,
incrementing c lead to $c=0$



States where $3 \leq y \leq 4$ and $c \neq -1$,
incrementing c lead to $c \neq 0$

States where $4 < y \leq 5$,
only possibility is time-out

Results:

- imprecise, as in earlier example
- bounds on max. prob. of failure are [0.01,0.1]

Symbolic operations

- Need symbolic manipulation of abstract states

- For example, the post operator
  - to construct reachability graph
  - over abstract states $A = L \times \{F,T\}^n \times Zones(X)$
  - split into two parts, timed and discrete:
  - $tpost[l] : A \rightarrow 2^A$ – elapse of time in location l
  - $dpost[e] : A \rightarrow 2^A$ – discrete transition on edge $e = (l,\alpha,up,Y,l')$

- Also need (not discussed here) operations to:
  - construct player 1/2 choices in stochastic game
  - split abstract states during refinement

Symbolic operations: Post

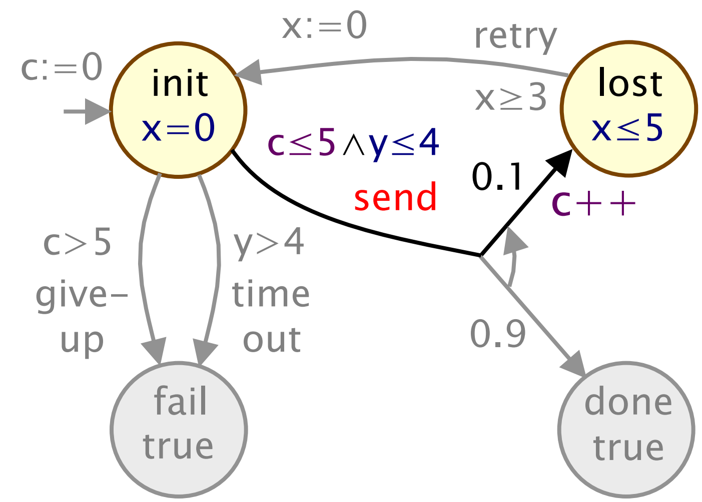
- Time (clocks X)
  - use zone operations, implemented with DBMs
  - for zone $\zeta \in \text{Zones}(X)$:
  - $\text{tpost}_X[l](\zeta) = \text{inv}(l) \wedge \nearrow\zeta$
  - $\text{dpost}_X[e](\zeta) = (\zeta \wedge \text{enab}(l,\alpha))[Y:=0] \wedge \text{inv}(l')$
- Data (variables D)
  - formulate as SAT/SMT problem, use solver to enumerate
  - for predicate valuation $b \in \{F,T\}^n$:
  - $\text{dpost}_D[e](b)$ contains all instances of $b' \in \{F,T\}^n$ such that
  - $\exists u,u' \in \text{Val}(D)$ satisfying: $\text{up}(u)=u' \wedge \Phi(u)=b \wedge \Phi(u')=b'$
- Combined time/data
  - for an abstract state $(l,b,\zeta) \in L \times \{F,T\}^n \times \text{Zones}(X)$:
  - $\text{tpost}[l](l,b,\zeta) = \{ (l,b,\text{tpost}_X[l](\zeta)) \}$
  - $\text{dpost}[e](l,b,\zeta) = \{ (l',b',\text{dpost}_X[e](\zeta)) \mid b' \in \text{dpost}_D[e](b) \}$

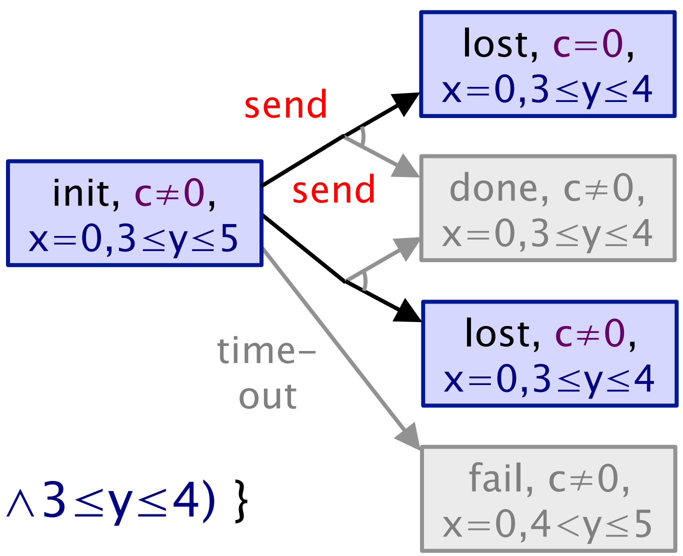
Example: Post operator

- Abstract state a = (l,b,ζ)
  - where l=init, b=(f), $\zeta = x{=}0 \wedge 3 \leq y \leq 5$
  - and edge e = (init,send,c++,{},lost)
- Time
  - $tpost_X[init](\zeta) = x{=}0 \wedge 3 \leq y \leq 5$
  - $dpost_X[e](\zeta) = x{=}0 \wedge 3 \leq y \leq 4$
- Data
  - $dpost_D[e](b) = \{(f),(t)\}$
- Combined (tpost, then dpost)
  - tpost[init](a) = { a' }

    where a' = (init,(f),$x{=}0 \wedge 3 \leq y \leq 5$)
  - dpost[e](a') =

    { (lost,(f),$x{=}0 \wedge 3 \leq y \leq 4$), (lost,(t),$x{=}0 \wedge 3 \leq y \leq 4$) }





Overview (Part 6)

- Model checking for PTAs
  - recap, summary
  - zone-based approaches:
  - (i) forwards reachability
  - (ii) backwards reachability
  - (iii) game-based abstraction refinement

- Verifying software with time and probabilities
  - probabilistic timed programs (PTPs)
  - verifying PTPS with abstraction + refinement

- Looking ahead: Quantitative verification of SystemC

A concrete challenge: SystemC

- SystemC: A system-level modelling language
  - increasingly prominent in the development of embedded systems, e.g. for System-on-Chip (SoC) designs
  - close enough to hardware level to support synthesis to RTL
  - but models complex designs at a higher level of abstraction
  - very efficient simulation at design phase
- Basic ingredients
  - C++-based, with low-level data-types for hardware
  - an object-oriented approach to design
  - and convenient high-level abstractions of concurrent communicating processes
- Analysis of SystemC designs
  - mostly simulation currently; growing interest in verification
  - identified as an important but challenging direction [Vardi'07]

Quantitative verification of SystemC

- Challenges involved in quantitative verification of SystemC:
- Software
  - basic process behaviour is defined in terms of C++ code, using a rich array of data types
- Concurrency
  - designs comprise multiple concurrent processes, communicating through message-passing primitives
- Timing
  - processes can be subjected to precisely timed delays, through interaction with the SystemC scheduler
- Probability
  - SystemC components may link to unpredictable devices
  - due to communication failures (e.g. wireless/radio), or randomisation (e.g. ZigBee/Bluetooth)

Quantitative verification of SystemC

- Outline approach to quantitative SystemC verification…
- SystemC designs comprise multiple modules/threads
  - communicating through ports/channels
  - translate to parallel composition of PTPs
  - C++ control-flow graph maps to PTP locations/transitions
  - various SystemC model extractors exist to do this
- Concurrency/timing between SystemC threads
  - controlled by precisely defined (co-operative/non-preemptive) scheduler, incorporating thread-specified delays
  - existing translation from SystemC to UPPAAL [Herber et al.'08]
- Probabilistic behaviour - randomisation or failures
  - randomisation: map rand() calls to PTP probabilistic choice
  - failures: replace e.g. network calls with probabilistic stubs
  - similar approach applied to probabilistic ANSI-C [VMCAI'09]

Summary (Part 6)

- Probabilistic timed automata (PTAs)
  - combine probability, nondeterminism, real-time
- PTA model checking
  - region graph: decidability results, exponential complexity
  - digital clocks: simple and effective, some scalability issues
  - forwards reachability: only upper bounds on max. prob.s
  - backwards reachability: exact results but often expensive
  - abstraction refinement using stochastic games: performs well
  - tool support: PRISM, mcpta, UPPAAL-Pro
- Probabilistic timed programs
  - probability + nondeterminism + real-time + data
  - amenable to verification with abstraction/refinement

Course summary

- Quantitative verification
  - probability (e.g. randomisation, failures)
  - nondeterminism (e.g. concurrency, underspecification)
  - real-time behaviour and constraints (e.g. delays, time-outs)
- Probabilistic models:
  - discrete-time Markov chains, Markov decision processes, probabilistic timed automata, probabilistic timed programs
- Probabilistic model checking:
  - temporal logics, e.g. PCTL, PTCTL
  - efficient techniques, tools exist
- Compositional probabilistic verification
  - MDP-based assume-guarantee framework
- Quantitative abstraction refinement
  - fully automatic construction/analysis of abstractions
  - essential for large, complex systems such as software

謝謝

Thanks for your attention

More info here:
www.prismmodelchecker.org