A Framework for Verification of Software with Time and Probabilities

Marta Kwiatkowska

Oxford University Computing Laboratory

Joint work with:

Gethin Norman and David Parker

FORMATS'10 Invited Talk, September 2010

Probabilistic verification

- Probabilistic verification
  - formal verification of systems exhibiting stochastic behaviour
- Why probability?
  - unreliability (e.g. component failures)
  - uncertainty (e.g. message losses/delays over wireless)
  - randomisation (e.g. in protocols such as Bluetooth, ZigBee)
- Quantitative properties
  - reliability, performance, quality of service, …
  - “the probability of an airbag failing to deploy within 0.02s”
  - “the expected time for a network protocol to send a packet”
  - “the expected power usage of a sensor network over 1 hour”

Probabilistic verification

- The state of the art
  - fast/efficient techniques for a range of probabilistic models
  - (mostly Markov chains, Markov decision processes)
  - feasible for models of up to 10^7 states (10^{10} with symbolic)
  - tool support exists and is widely used
  - successfully applied to many application domains: communication protocols, security, biology, …

- The challenges
  - scalability and efficiency: larger models, verified faster
  - more realistic models: real-time behaviour, continuous dynamics, stochastic hybrid systems, …
  - ease of applicability, e.g. direct verification of mainstream modelling/programming languages (C, Simulink, SystemC, …)
  - needs: efficient and automated abstraction techniques

Probabilistic models

- Discrete-time Markov chains (DTMCs)
  - discrete states + probability
  - for: randomisation, component failures, unreliable media

- Markov decision processes (MDPs)
  - discrete states, probability and nondeterminism
  - for: concurrency, under-specification, abstraction

- Probabilistic timed automata (PTAs)
  - probability, nondeterminism and real-time

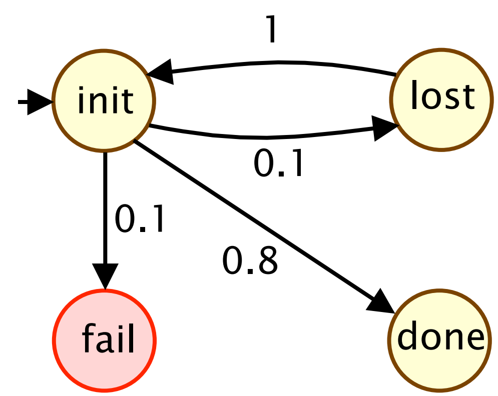
- Probabilistic timed programs (PTPs)
  - probability, nondeterminism and real-time and data
  - for: software verification of real programming languages

Overview

- Probabilistic verification
  - discrete-time Markov chains (DTMCs)
  - Markov decision processes (MDPs)
  - probabilistic timed automata (PTAs)
- Quantitative abstraction refinement
  - game-based abstraction of MDPs
  - quantitative abstraction-refinement loop
  - verification of PTAs and probabilistic software
- Verifying software with time and probabilities
  - probabilistic timed programs (PTPs)
  - verifying PTPs with abstraction + refinement
- A concrete challenge
  - quantitative verification of SystemC verification
- Conclusions

Discrete-time Markov chains

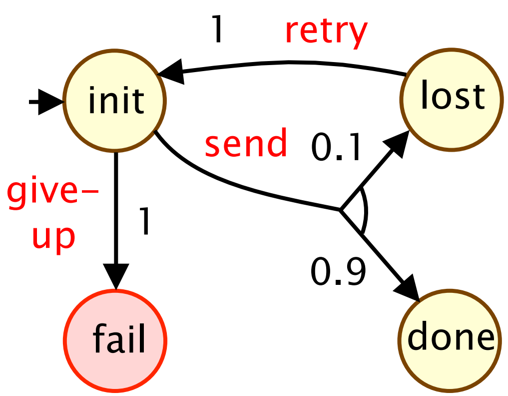
- Discrete-time Markov chains (DTMCs)
  - model fully probabilistic behaviour
  - state-transition systems augmented with probabilistic choice
- Formally, a DTMC is a tuple (S, P) where:
  - S is a set of states
  - $P : S \times S \rightarrow [0,1]$ is the transition probability matrix
- To reason formally:
  - define a probability space over infinite paths through DTMC
  - allows computation of, for example…



- Probabilistic reachability
  - key concept for model checking
  - $p_s(F)$ = probability of reaching goal states $F \subseteq S$ from state s
  - reduces to the solution of a linear equation system

Markov decision processes

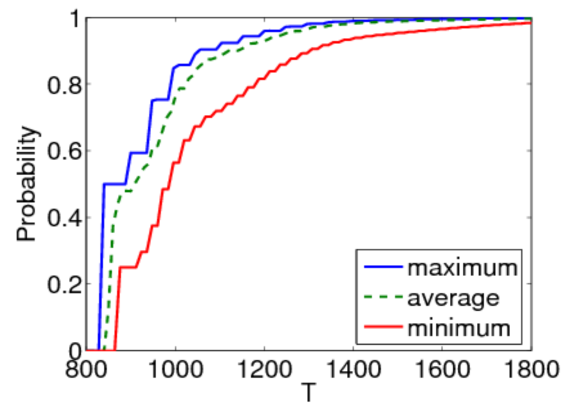
- Markov decision processes (MDPs)
  – model nondeterministic as well as probabilistic behaviour
  – nondeterministic choice between probability distributions
- Formally, an MDP is a tuple (S, Act, Steps) where:
  – S is a set of states, Act is a set of actions
  – Steps : $S \times Act \rightarrow Dist(S)$ is the transition probability function



- An adversary (aka. “scheduler”/“strategy”) of an MDP
  – is a resolution of the nondeterminism in the MDP
  – under a given adversary σ, the behaviour is fully probabilistic

Probabilistic reachability for MDPs

- Probabilistic reachability for MDPs
  - $p_s^\sigma(F)$ = probability of reaching $F \subseteq S$ starting from s under σ
  - consider the minimum/maximum values over all adversaries
  - $p_s^{min}(F) = \inf_\sigma p_s^\sigma(F)$ and $p_s^{max}(F) = \sup_\sigma p_s^\sigma(F)$



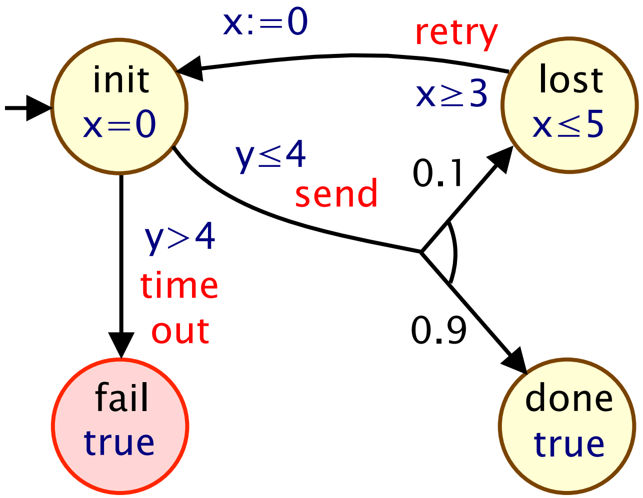
  - can be computed efficiently
  - (linear programming, value iteration)
  - optimal adversaries obtained too
  - tool support exists (e.g. PRISM, LiQuor, RAPTURE)

- Allows reasoning about best/worst-case behaviour
  - e.g. minimum probability of the protocol terminating correctly
  - e.g. maximum probability of a security breach

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...

PTA model checking – Summary

- Several PTA model checking techniques developed
  - construction/analysis of a finite-state model (usually MDP)

- Region graph construction [KNSS – TCS'02]
  - shows decidability, but gives exponential complexity
- Digital (integer) clocks approach [KNPS – FMSD'06]
  - slightly restricted classes of PTAs: closed zones only
  - works well in practice, still some scalability limitations
- Forwards reachability [KNSS – TCS'02]
  - efficient zone-based technique, approximate results only
- Backwards reachability [KNSW – I&C07]
  - exact results, expensive zone operations required
- Quantitative abstraction refinement [KNP – FORMATS'09]
  - abstraction to stochastic games, best in practice

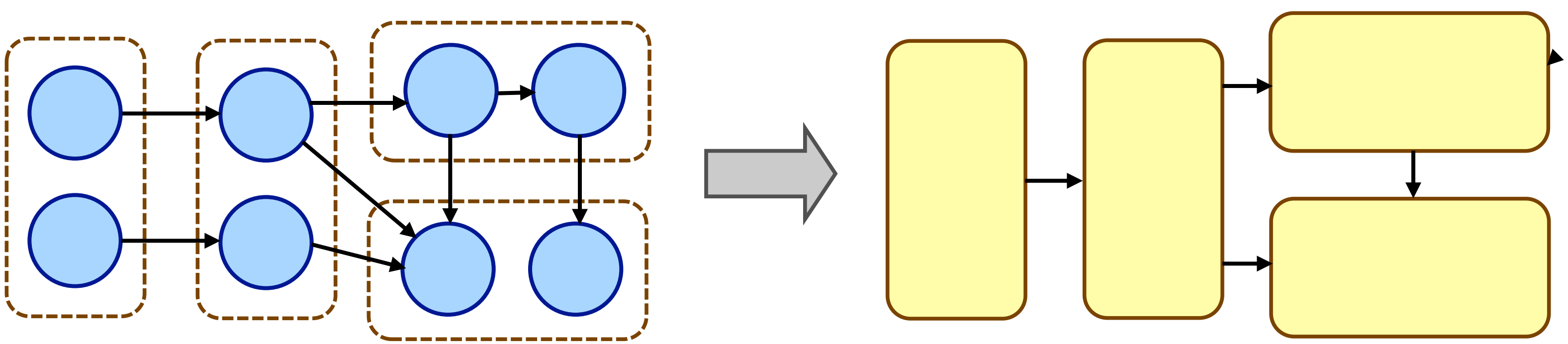
Overview

- Quantitative verification
  - discrete-time Markov chains (DTMCs)
  - Markov decision processes (MDPs)
  - probabilistic timed automata (PTAs)
- Quantitative abstraction refinement
  - game-based abstraction of MDPs
  - abstraction-refinement loop
  - verification of PTAs and probabilistic software
- Verifying software with time and probabilities
  - probabilistic timed programs (PTPs)
  - verifying PTPS with abstraction + refinement
- A concrete challenge: Quantitative SystemC verification
- Conclusions, challenges & future work

Abstraction

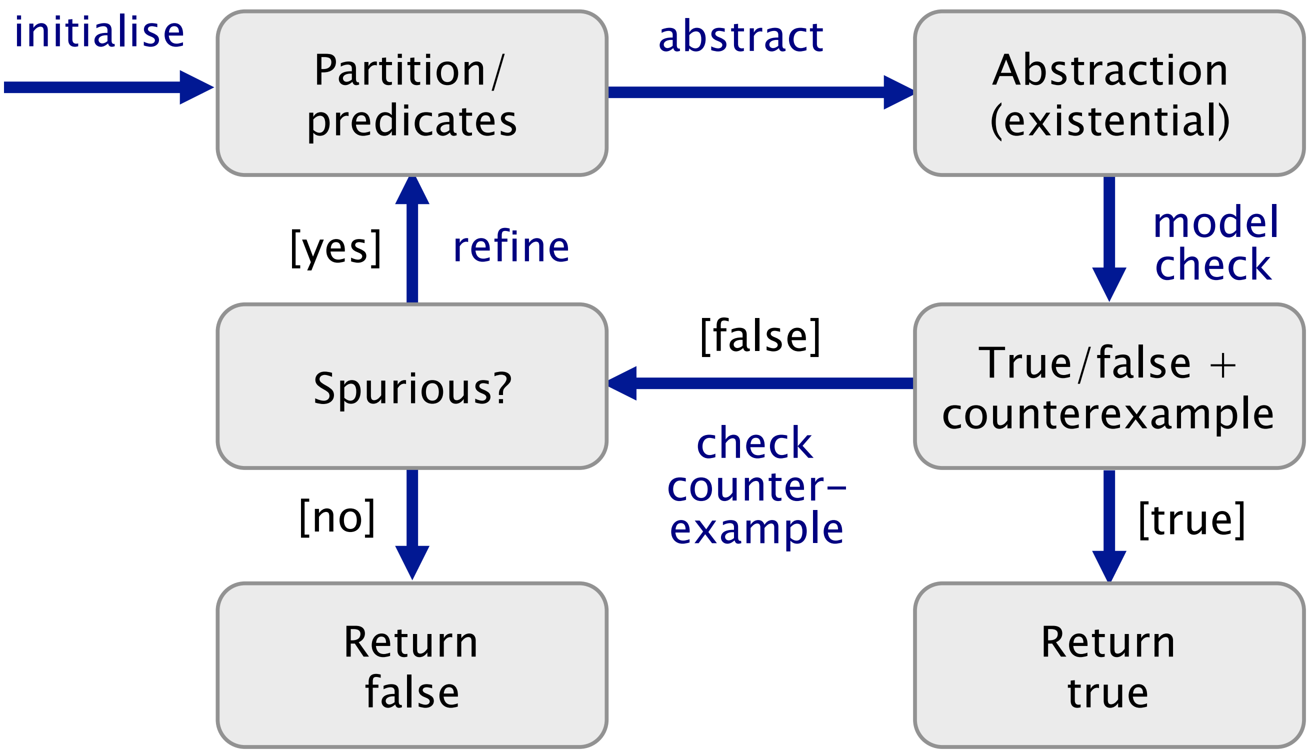
- Very successful in (non-probabilistic) formal methods
  - essential for verification of large/infinite-state systems
  - hide details irrelevant to the property of interest
  - yields smaller/finite model which is easier/feasible to verify
  - loss of precision: verification can return "don't know"

- 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



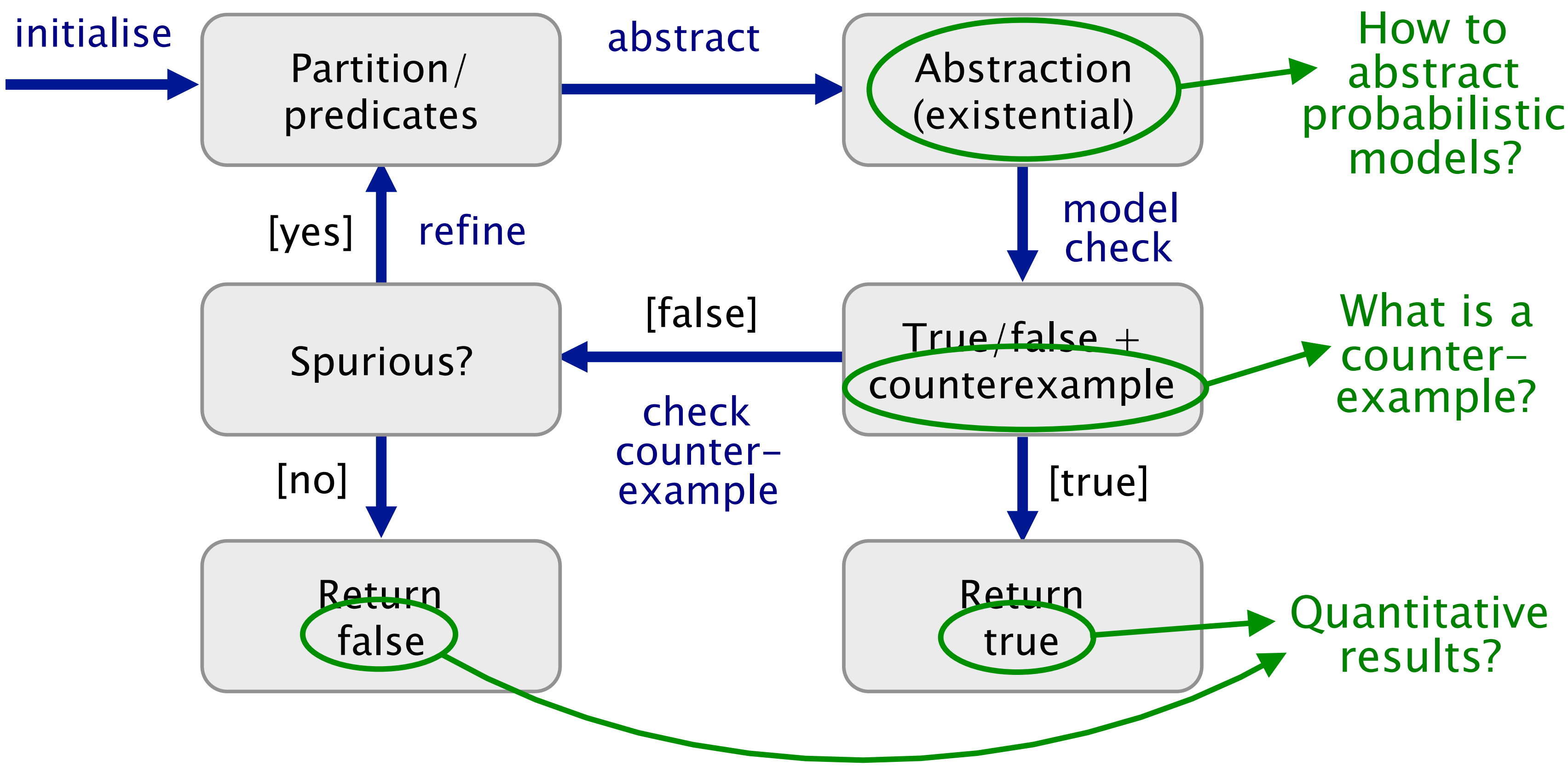
Abstraction refinement (CEGAR)

- Counterexample-guided abstraction refinement
  - (non-probabilistic) model checking of reachability properties



Abstraction refinement (CEGAR)

- Counterexample–guided abstraction refinement
  – (~~non~~–probabilistic) model checking of reachability properties



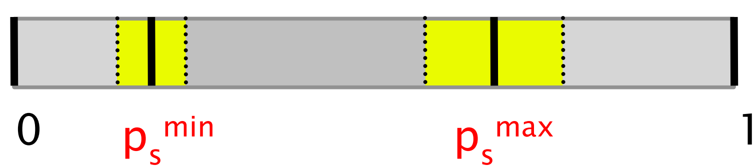
Abstraction of MDPs

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

- But what form does the abstraction of an MDP take?
- 2 possibilities:

  (i) an MDP [D'Argenio/Jeannet/Jensen/Larsen'01]
  - probabilistic simulation relates concrete/abstract models

  (ii) a stochastic two-player game [KNP – QEST'06]
  - separates nondeterminism from abstraction and from MDP
  - yields separate lower/upper bounds for min/max



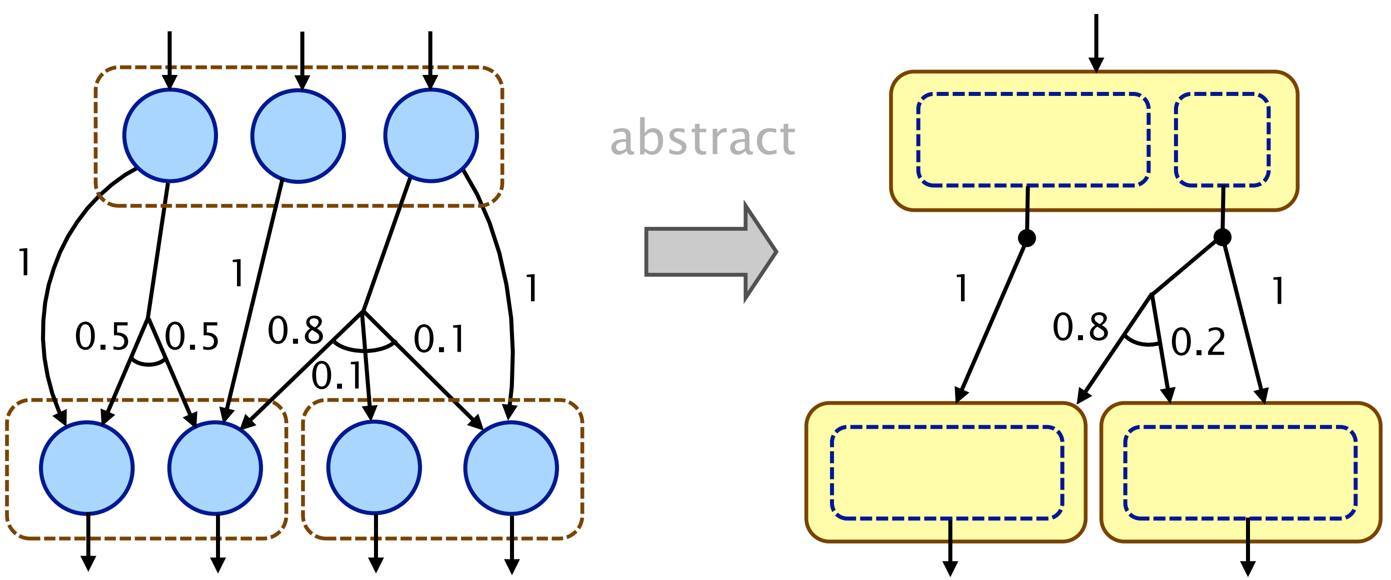
Stochastic two-player games

- Subclass of simple stochastic games [Shapley,Condon]
  - two nondeterministic players (1 and 2) and probabilistic choice
- Resolution of the nondeterminism in a game
  - corresponds to a pair of strategies for players 1 and 2: (σ_1,σ_2)
  - $p_a^{\sigma 1,\sigma 2}(F)$ probability of reaching F from a under (σ_1,σ_2)
  - can compute, e.g. : $\sup_{\sigma 1} \inf_{\sigma 2} p_a^{\sigma 1,\sigma 2}(F)$
  - informally: "the maximum probability of reaching F that player 1 can guarantee no matter what player 2 does"
- Abstraction of an MDP as a stochastic two-player game:
  - player 1 controls the nondeterminism of the abstraction
  - player 2 controls the nondeterminism of the MDP

Game abstraction (by example)

- Player 1 vertices ( ) are abstract states
- (Sets of) distributions are lifted to the abstract state space
- Player 2 vertices ( ) are states with same (sets of) choices

MDP (fragment) Stochastic game (fragment)



Properties of the abstraction

- Analysis of game yields lower/upper bounds:
  - for target $F \in A$, $s \in S$ and $a \in A$ with $s \in a$

$$\inf_{\sigma1,\sigma2} p_a^{\sigma1,\sigma2}(F) \;\leq\; p_s^{min}(F) \;\leq\; \sup_{\sigma1} \inf_{\sigma2} p_a^{\sigma1,\sigma2}(F)$$

$$\inf_{\sigma1} \sup_{\sigma2} p_a^{\sigma1,\sigma2}(F) \;\leq\; p_s^{max}(F) \;\leq\; \sup_{\sigma1,\sigma2} p_a^{\sigma1,\sigma2}(F)$$

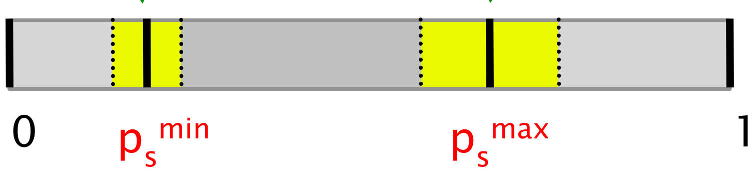
Properties of the abstraction

- Analysis of game yields lower/upper bounds:
  - for target $F \in A$, $s \in S$ and $a \in A$ with $s \in a$

$$\inf_{\sigma1,\sigma2} p_a^{\sigma1,\sigma2}(F) \;\le\; p_s^{min}(F) \;\le\; \sup_{\sigma1} \inf_{\sigma2} p_a^{\sigma1,\sigma2}(F)$$

$$\inf_{\sigma1} \sup_{\sigma2} p_a^{\sigma1,\sigma2}(F) \;\le\; p_s^{max}(F) \;\le\; \sup_{\sigma1,\sigma2} p_a^{\sigma1,\sigma2}(F)$$

min/max reachability probabilities for original MDP



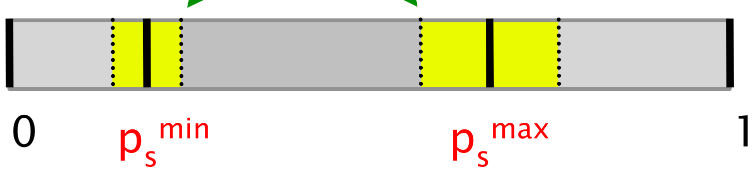
Properties of the abstraction

- Analysis of game yields lower/upper bounds:
  - for target $F \in A$, $s \in S$ and $a \in A$ with $s \in a$

$$\inf_{\sigma1,\sigma2} p_a^{\sigma1,\sigma2}(F) \leq p_s^{min}(F) \leq \sup_{\sigma1} \inf_{\sigma2} p_a^{\sigma1,\sigma2}(F)$$

$$\inf_{\sigma1} \sup_{\sigma2} p_a^{\sigma1,\sigma2}(F) \leq p_s^{max}(F) \leq \sup_{\sigma1,\sigma2} p_a^{\sigma1,\sigma2}(F)$$

optimal probabilities for player 1, player 2 in game



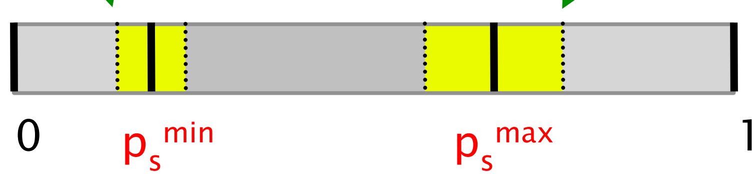
Properties of the abstraction

- Analysis of game yields lower/upper bounds:
  - for target $F \in A$, $s \in S$ and $a \in A$ with $s \in a$

$$\inf_{\sigma1,\sigma2} p_a^{\sigma1,\sigma2}(F) \le p_s^{min}(F) \le \sup_{\sigma1} \inf_{\sigma2} p_a^{\sigma1,\sigma2}(F)$$

$$\inf_{\sigma1} \sup_{\sigma2} p_a^{\sigma1,\sigma2}(F) \le p_s^{max}(F) \le \sup_{\sigma1,\sigma2} p_a^{\sigma1,\sigma2}(F)$$

min/max reachability probabilities, treating game as MDP
(i.e. assuming that players 1 and 2 cooperate)

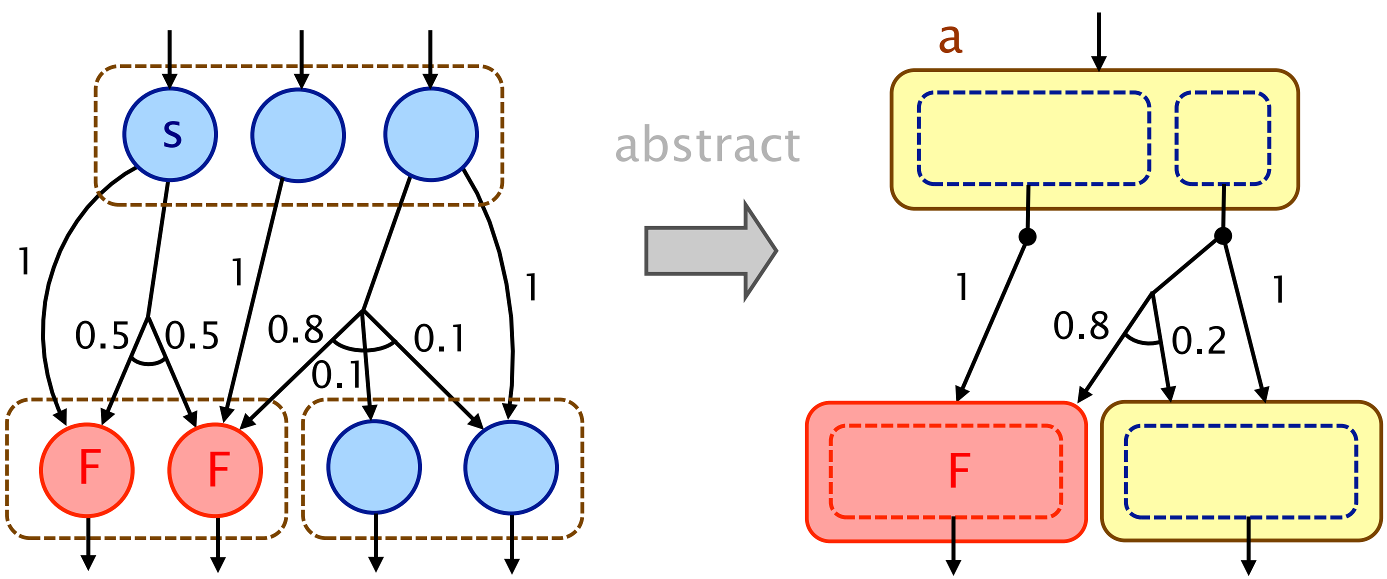


Example – Abstraction

$p_s^{max}(F) = 1 \in [0.8, 1]$

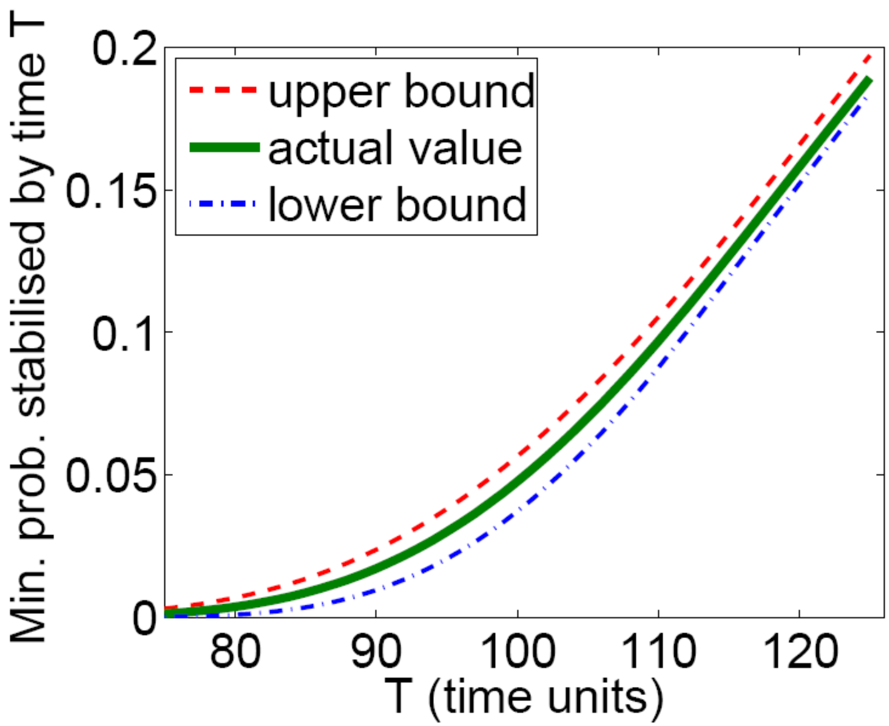
$\inf_{\sigma 1} \sup_{\sigma 2} p_a^{\sigma 1, \sigma 2}(F) = 0.8$

$\sup_{\sigma 1, \sigma 2} p_a^{\sigma 1, \sigma 2}(F) = 1$



Experimental results

- Israeli & Jalfon's Self Stabilisation
  - protocol for obtaining a stable state in a token ring
  - minimum probability of reaching a stable state by time T

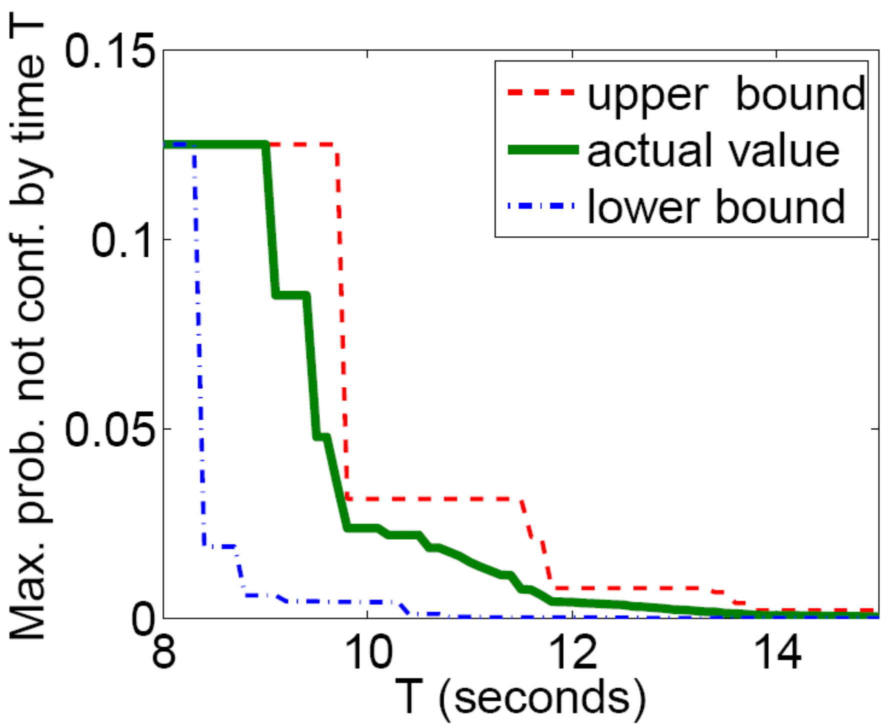


concrete states: 1,048,575

abstract states: 627

Experimental results

- IPv4 Zeroconf
  - protocol for obtaining an IP address for a new host
  - maximum probability the new host not configured by T

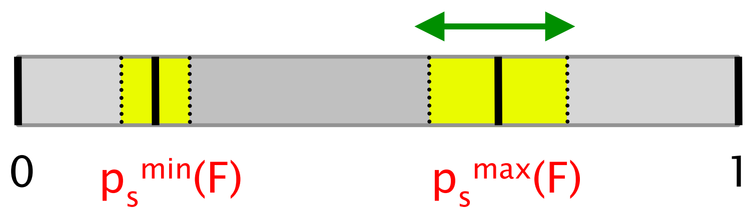


concrete states: 838,905

abstract states: 881

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

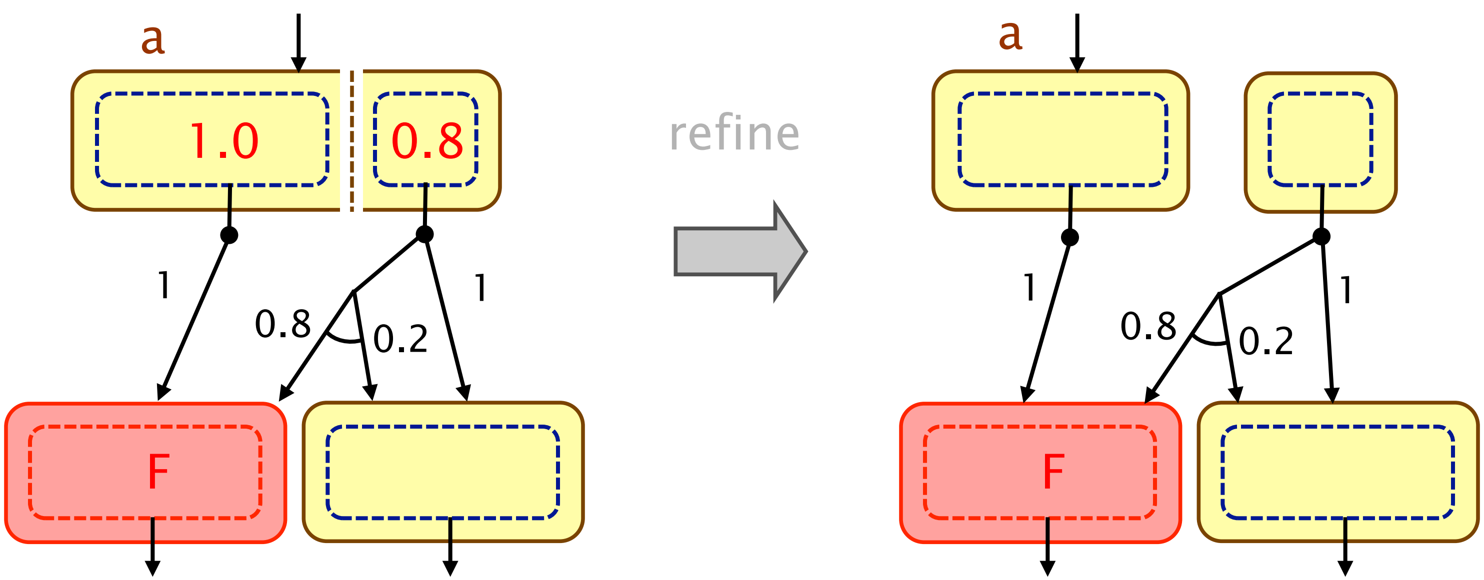
Example – Refinement

$p_s^{max}(F) = 1 \in [0.8, 1]$

"error" $= 0.2$

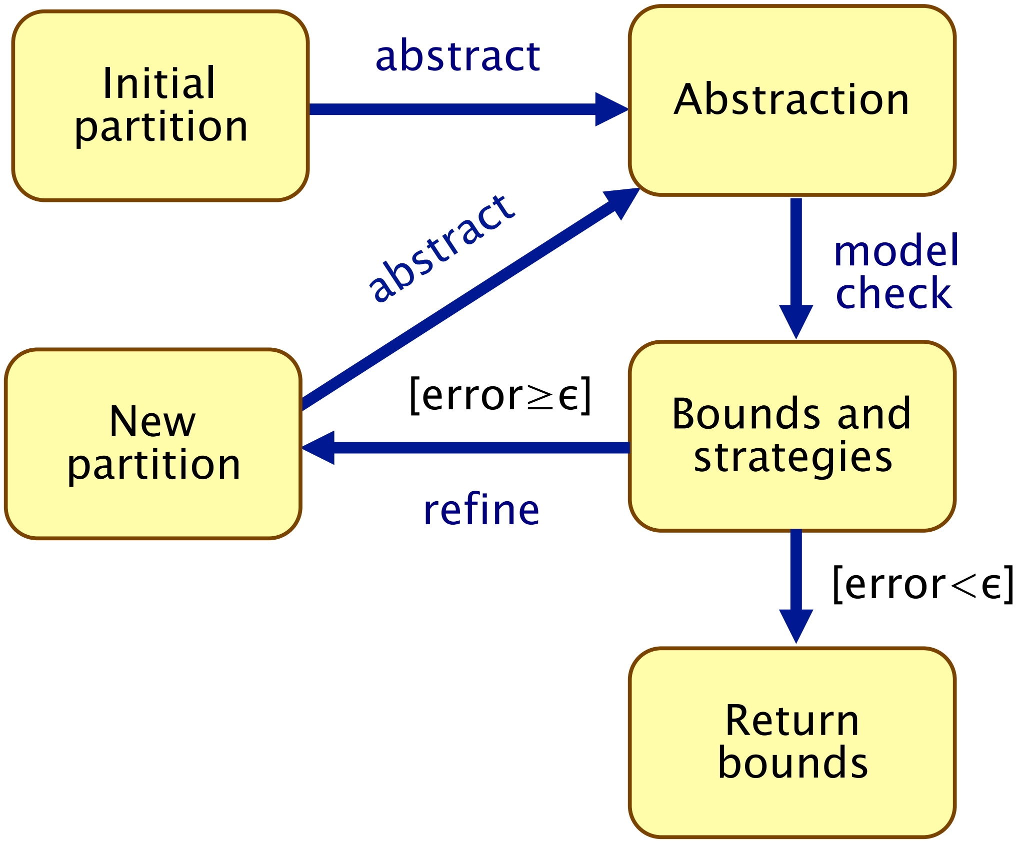
$p_s^{max}(F) = 1 \in [1, 1]$

"error" $= 0$



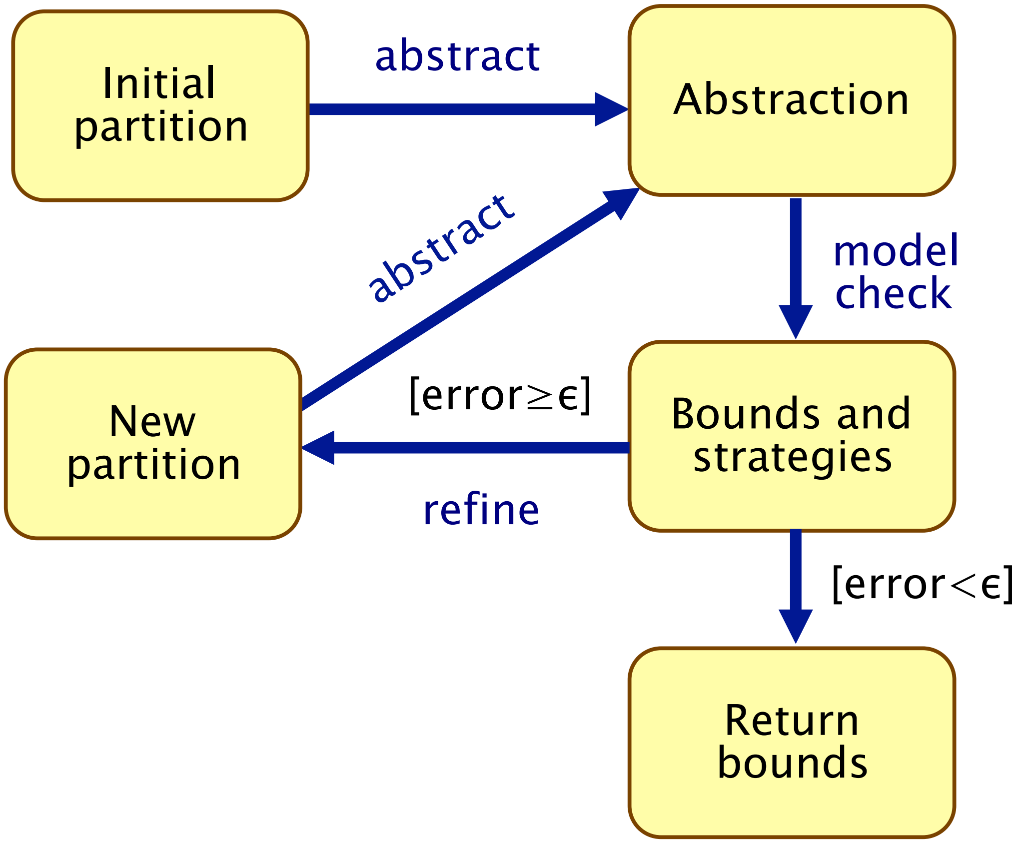
Abstraction-refinement loop

- Quantitative abstraction-refinement loop for MDPs



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 loop

- Implementations of quantitative abstraction refinement…

- Verification of probabilistic timed automata [FORMATS'09]
  - zone-based abstraction/refinement using DBMs
  - implemented in (development release of) PRISM
  - outperforms existing PTA verification techniques

- Verification of probabilistic software [VMCAI'09]
  - predicate abstraction/refinement using SAT solvers
  - implemented in tool qprover: components of PRISM, SATABS
  - analysed real network utilities (ping, tftp) - approx 1KLOC

- Verification of concurrent PRISM models [Wachter/Zhang'10]
  - implemented in tool PASS; infinite-state PRISM models

Overview

- Quantitative verification
  - discrete-time Markov chains (DTMCs)
  - Markov decision processes (MDPs)
  - probabilistic timed automata (PTAs)
- Quantitative abstraction refinement
  - game-based abstraction of MDPs
  - abstraction-refinement loop
  - verification of PTAs and probabilistic software
- Verifying software with time and probabilities
  - probabilistic timed programs (PTPs)
  - verifying PTPS with abstraction + refinement
- A concrete challenge: Quantitative SystemC verification
- Conclusions, challenges & future work

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 $c, d \in \mathbb{N}$

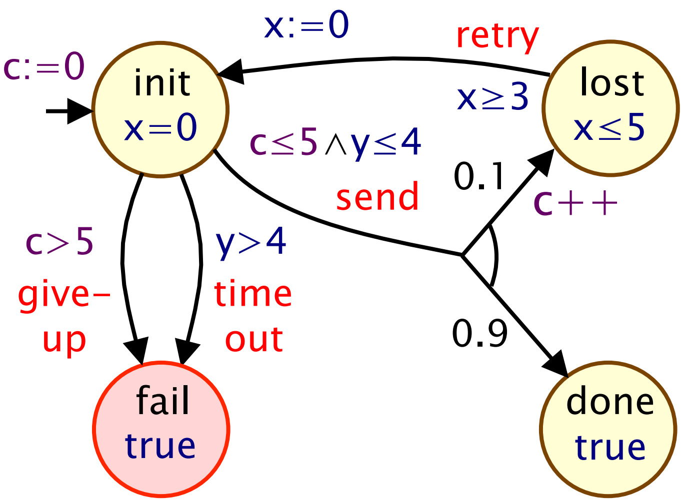
- Data – assume a finite set D of data variables
  - Val(D) is the set of all valuations of D
  - Pred(D) is the set of predicates over D
  - Up(D) is the set of all update functions over D
  - i.e. set of all functions $up : Val(D) \rightarrow Val(D)$

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
  - $inv : L \rightarrow Zones(X)$ is the invariant condition
    - clocks X must satisfy $inv(l)$ whilst in location l
  - $enab : L \times Act \rightarrow Pred(D) \times Zones(X)$ 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 $enab_D(l,a) \wedge enab_X(l,a)$
  - $prob : L \times Act \rightarrow Dist(Up(D) \times 2^X \times L)$
    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 $Y \subseteq X$, 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

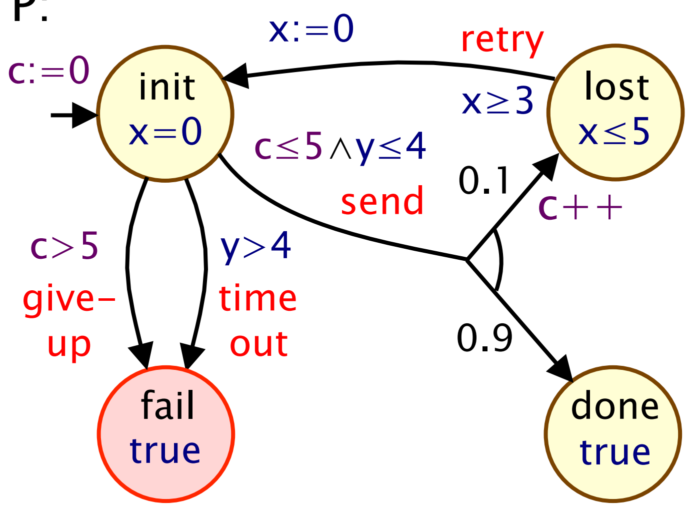
- Formal semantics of a PTP is an infinite-state MDP
  - over state space $L \times Val(D) \times \mathbb{R}^X$
  - data domain $Val(D)$ may be large/infinite; so need abstraction
  - time domain $\mathbb{R}$ is dense; so need abstraction
- In general, use an abstract domain $((A, \sqcup, \sqcap, \sqsubseteq), \alpha, \gamma)$
  - 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 $\Phi = \{\Phi_1, \ldots, \Phi_n\}$
  - (paper also covers case of predicates for data and time)
- 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

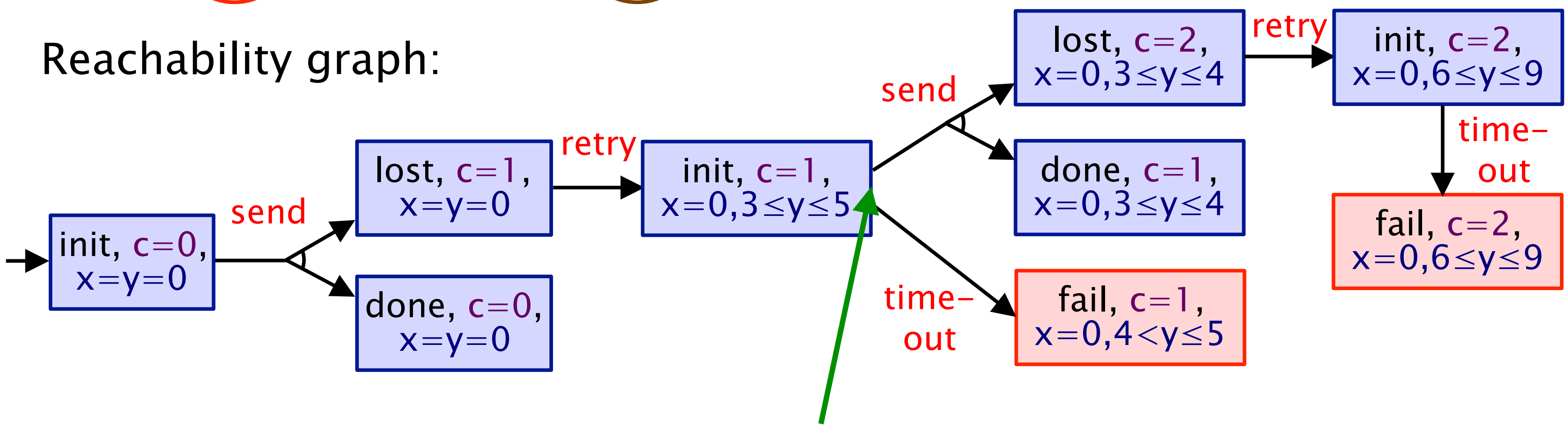
PTP:



In this example:

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

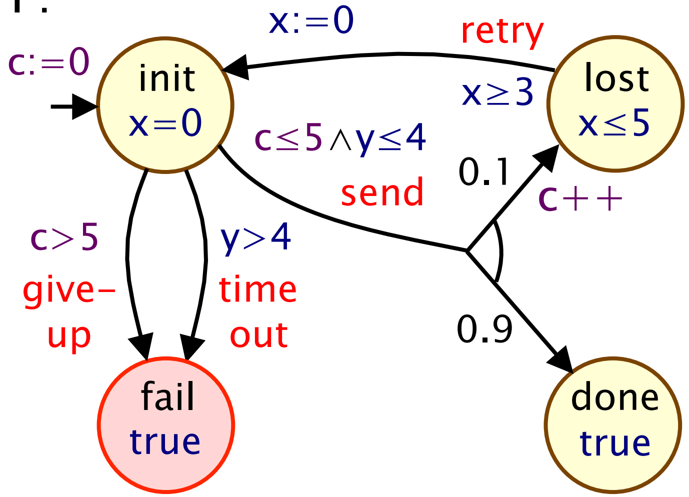
Reachability graph:



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

Example 1 – Abstraction

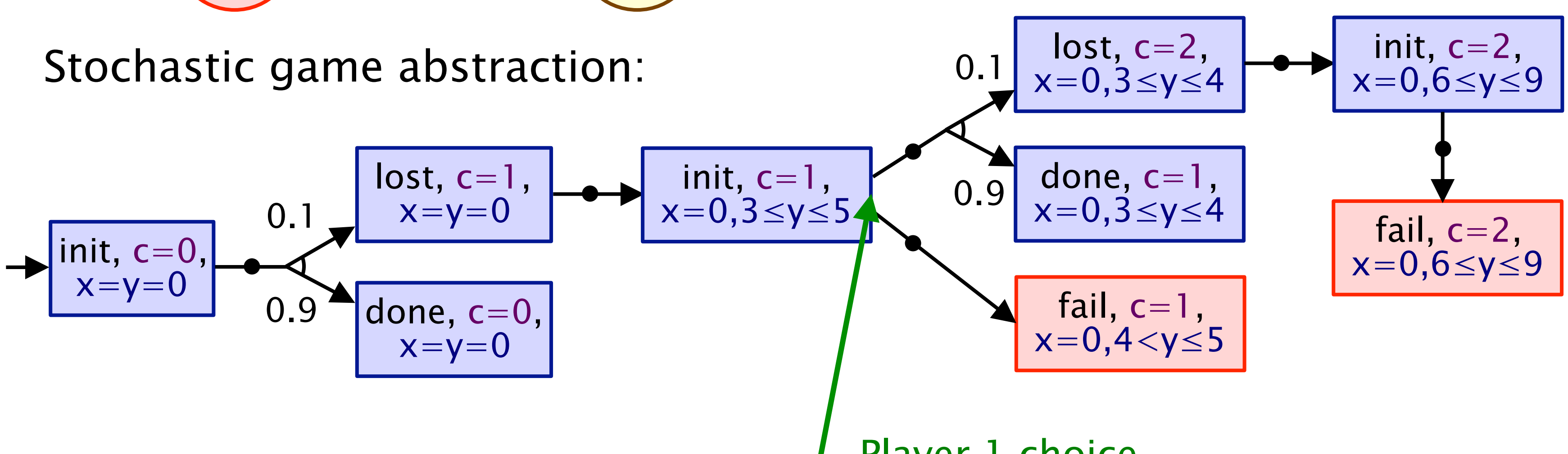
PTP:



Results:

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

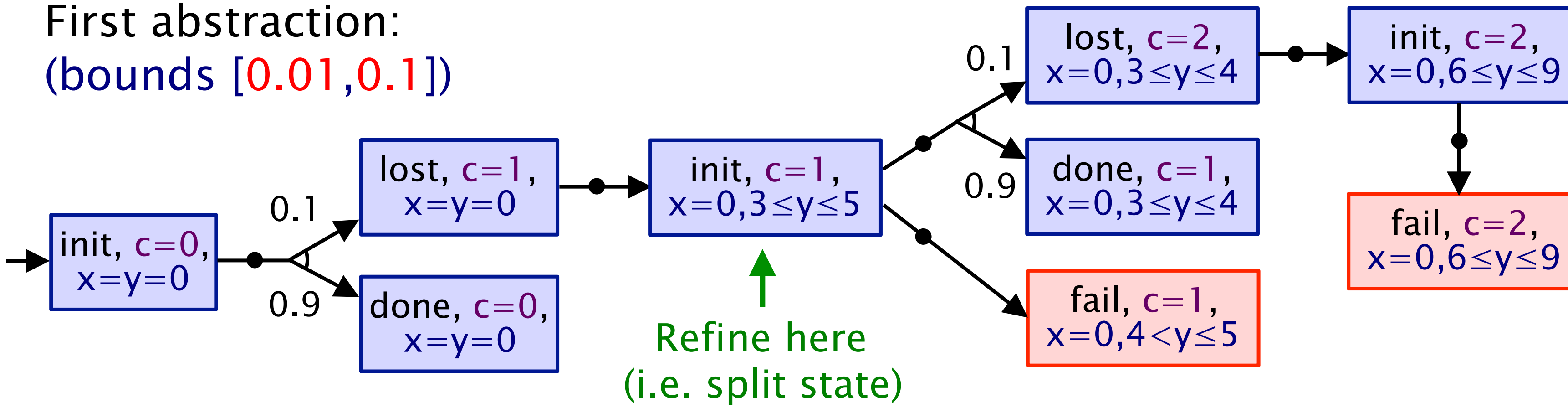
Stochastic game abstraction:



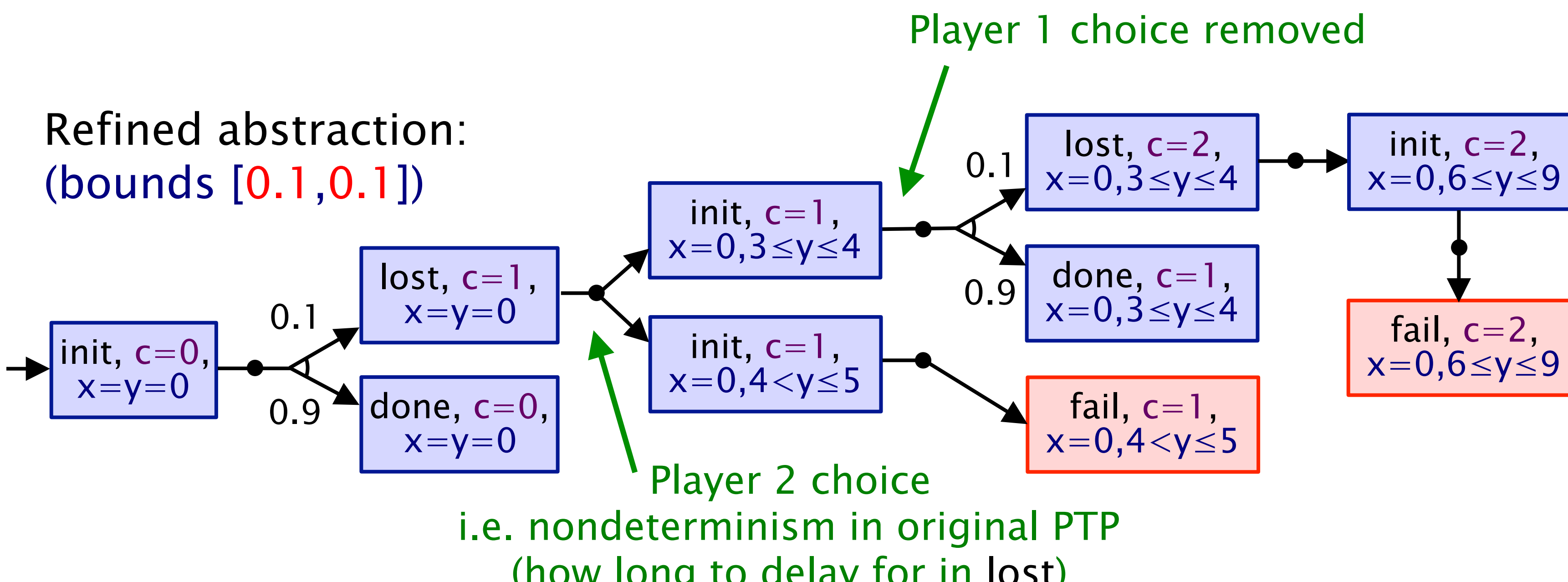
Player 1 choice
i.e. imprecision due to abstraction
3≤y≤4 or 4<y≤5?

Example 1 – Refinement

First abstraction:
(bounds [0.01,0.1])

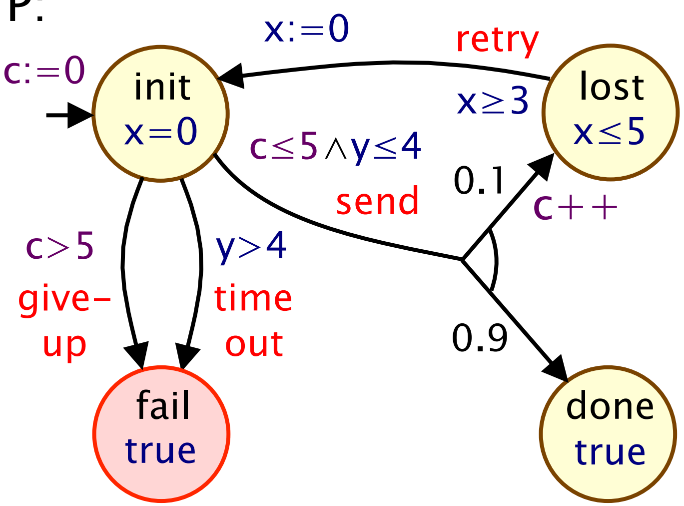


Refined abstraction:
(bounds [0.1,0.1])



Example 2 – Time and data

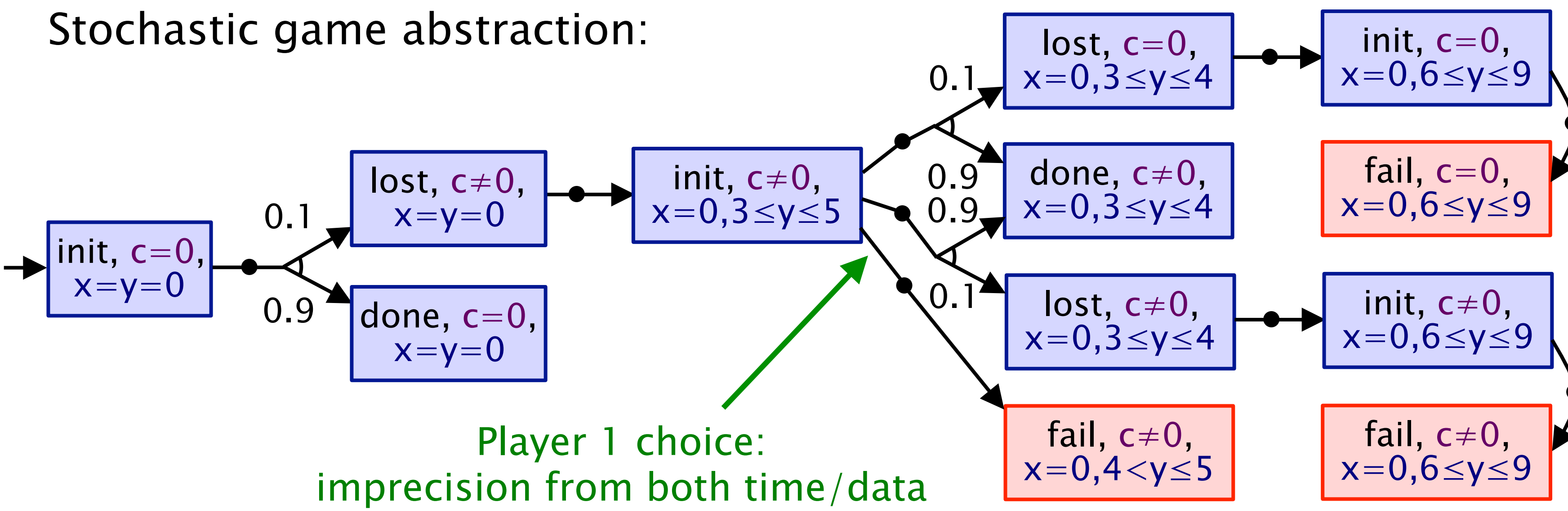
PTP:



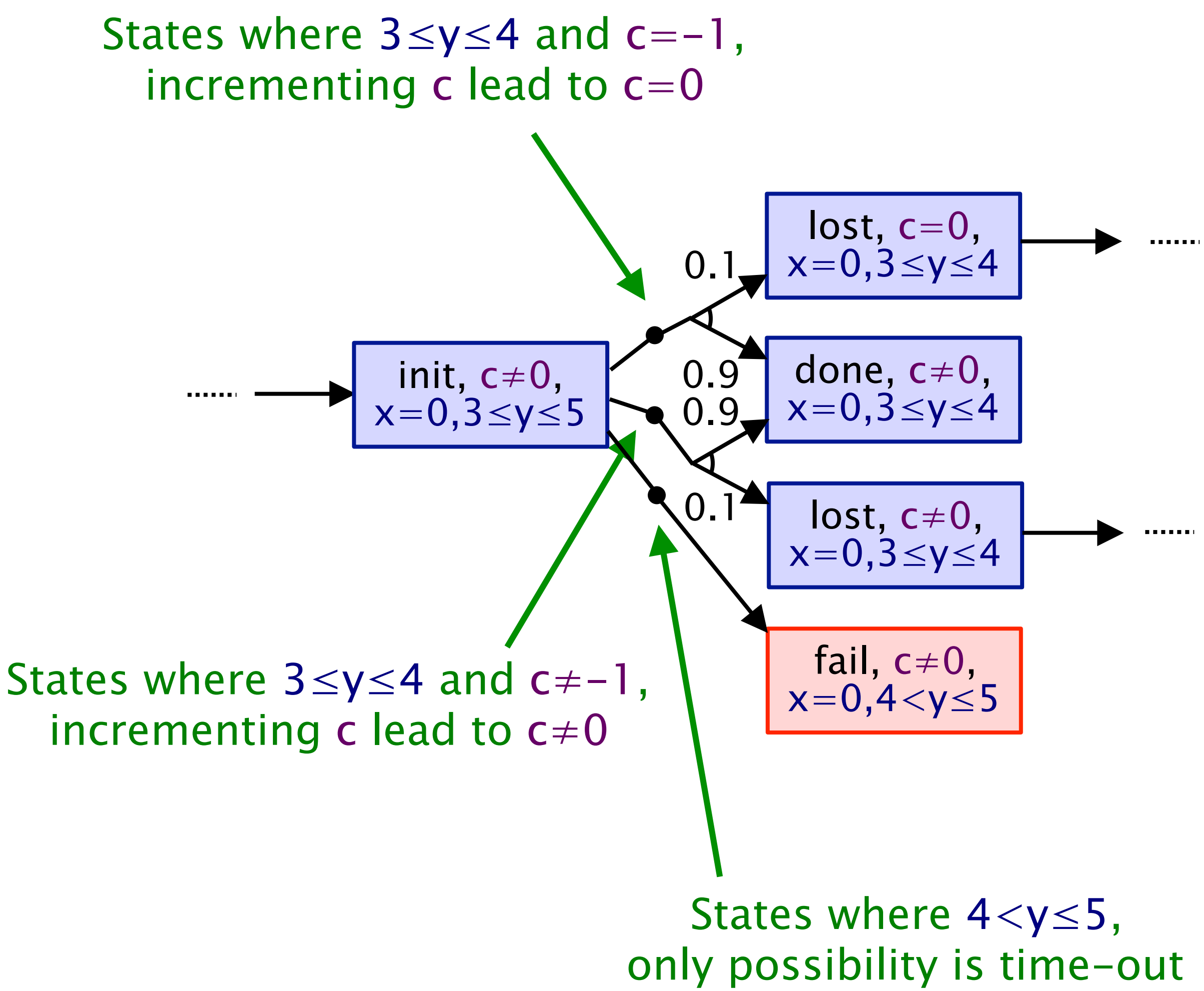
In this example:

- 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



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

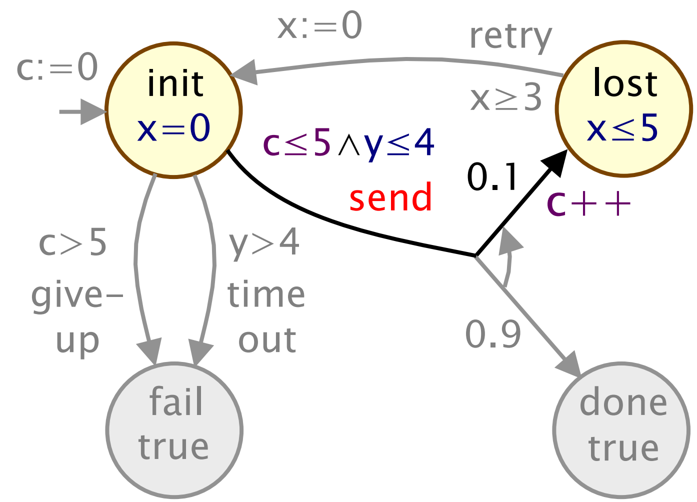
- Time (clocks X)
  - use zone operations, implemented with DBMs
  - for zone $\zeta \in \mathit{Zones}(X)$:
  - $tpost_X[l](\zeta) = inv(l) \wedge \nearrow\zeta$
  - $dpost_X[e](\zeta) = (\zeta \wedge enab(l,\alpha))[Y:=0] \wedge inv(l')$
- Data (variables D)
  - formulate as SAT/SMT problem, use solver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

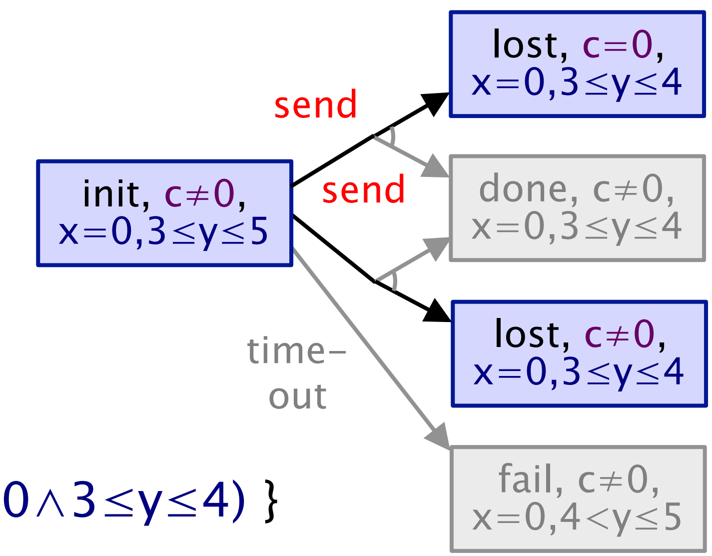
Example: Post operator

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

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

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





Overview

- Quantitative verification
  - discrete-time Markov chains (DTMCs)
  - Markov decision processes (MDPs)
  - probabilistic timed automata (PTAs)
- Quantitative abstraction refinement
  - game-based abstraction of MDPs
  - abstraction-refinement loop
  - verification of PTAs and probabilistic software
- Verifying software with time and probabilities
  - probabilistic timed programs (PTPs)
  - verifying PTPS with abstraction + refinement
- A concrete challenge: Quantitative SystemC verification
- Conclusions, challenges & future work

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]

Overview

- Quantitative verification
  - discrete-time Markov chains (DTMCs)
  - Markov decision processes (MDPs)
  - probabilistic timed automata (PTAs)
- Quantitative abstraction refinement
  - game-based abstraction of MDPs
  - abstraction-refinement loop
  - verification of PTAs and probabilistic software
- Verifying software with time and probabilities
  - probabilistic timed programs (PTPs)
  - verifying PTPS with abstraction + refinement
- A concrete challenge: Quantitative SystemC verification
- Conclusions, challenges & future work

Conclusions

- Probabilistic verification
  - discrete-time Markov chains, Markov decision processes, …
- Abstraction: essential for large/infinite-state systems
  - this talk: abstractions of MDPs as stochastic games
  - yields lower/upper bounds on min/max probabilities
- Quantitative abstraction refinement
  - fully automatic generation of abstractions
  - iterative refinement based on quantitative measure of ‘error’
  - works in practice: probabilistic software & timed automata
- Probabilistic timed programs
  - probability + nondeterminism + real-time + data
  - amenable to verification with abstraction/refinement

Challenges & Future work

- Scalability & efficiency
  - improved abstraction techniques/heuristics
  - compositional verification for PTAs/PTPs
- More realistic modelling of system behaviour
  - e.g. interaction with continuous environment
  - continuous probability distributions
  - probabilistic/stochastic hybrid systems
- Direct verification of modelling/programming languages
  - e.g. SystemC, Simulink
- Beyond verification
  - synthesis of parameters, controllers, designs, …