



# On quantitative software verification

Marta Kwiatkowska

Oxford University Computing Laboratory

Joint work with:

Mark Kattenbelt, Gethin Norman, Dave Parker

SPIN 2009, June 2009

# Why quantitative verification?

- **Real** software/systems are **not** qualitative:
  - **Real-time** aspects
    - hard/soft time deadlines
  - **Resource** constraints
    - energy, buffer size, number of unsuccessful transmissions, etc
  - **Randomisation**, e.g. in distributed coordination algorithms
    - random delays/back-off in CSMA/CD, IEEE 802.11, Bluetooth
  - **Uncertainty**, e.g. communication failures/delays
    - prevalence of wireless communication, low-power devices
- **Need formal techniques for quantitative guarantees of:**
  - safety, reliability, performance, dependability, resource usage, security, privacy, trust, anonymity, ...



# Quantitative probabilistic verification

- Models are typically extensions of automata annotated with **quantitative** information, e.g.
  - Probability of taking a transition
  - Time delay
  - Amount of resource
- **Nondeterminism** often essential, to account for environment and parallel composition
- System dynamics (usually under suitable restrictions)
  - Discrete (jumps) or continuous (dense-time)
  - Stochastic (randomly determined targets)
- Efficient **quantitative/probabilistic** model checking tools exist
- Aim to verify **real code**



# Overview

- 1. Quantitative probabilistic model checking
  - Markov decision processes (MDPs), temporal logics, PRISM
- 2. Abstraction & refinement for MDPs
  - abstraction: two-player stochastic games
  - a **quantitative abstraction-refinement loop**
- 3. Verification of probabilistic software
  - probabilistic verification **at the level of source code** (e.g. C)
  - game-based abstraction, predicate abstraction, SAT, ...
- 4. Verification of probabilistic real-time systems
  - abstraction refinement for probabilistic timed automata
  - applicable to **probabilistic real-time software**

# Overview

1. Quantitative probabilistic model checking



# Probabilistic model checking

- Model checking
  - Inputs:
    - finite-state transition system
    - temporal logic specification, e.g. CTL
  - Outputs:
    - “yes”/“no” + counterexample (e.g. trace to error state)
- Quantitative probabilistic model checking
  - Inputs:
    - finite-state **probabilistic** model, e.g. Markov decision process
    - **probabilistic** temporal logic specification, e.g. PCTL
  - Outputs:
    - “yes”/“no” + **quantitative** results/plots

# Discrete-time Markov chains (DTMCs)

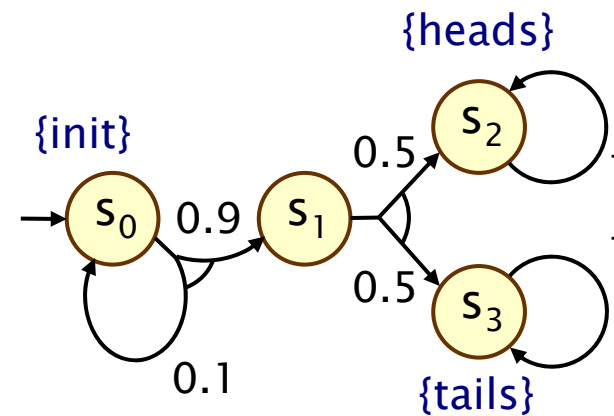
- Model **fully probabilistic** behaviour
  - state-transition systems augmented with probabilistic choice

- Formally, a DTMC is a tuple

- $(S, s_{init}, P, L)$

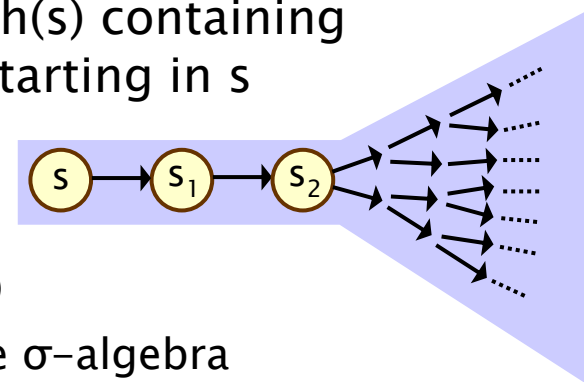
- where:

- $S$  is a set of states
- $s_{init} \in S$  is the initial state
- $P : S \times S \rightarrow [0,1]$  is the transition probability matrix
- $L : S \rightarrow 2^{AP}$  is a labelling function
- $AP$  is a set of atomic propositions



# Paths and probabilities

- Paths through a DTMC:
  - infinite sequences of states  $s_0s_1s_2s_3\dots$  such that  $P(s_i, s_{i+1}) > 0$
  - represent executions of the system being modelled
  - for quantitative reasoning, need probability space over paths
- Probability space (  $\text{Path}(s), \Sigma_{\text{Path}(s)}, \text{Pr}_s$  ) [KSK66]
  - sample space:  $\text{Path}(s)$  = all infinite paths starting in state  $s$
  - event set:  $\Sigma_{\text{Path}(s)}$  = **least  $\sigma$ -algebra** on  $\text{Path}(s)$  containing **cylinder set**  $\text{Cyl}(\omega)$  for all finite paths  $\omega$  starting in  $s$ 
    - $\text{Cyl}(\omega) = \{\omega' \in \text{Path}(s) \mid \omega \text{ is prefix of } \omega'\}$
  - probability measure:  $\text{Pr}_s: \Sigma_{\text{Path}(s)} \rightarrow [0, 1]$ 
    - $\text{Pr}_s(\text{Cyl}(s, s_1, \dots, s_n)) = P(s, s_1) \cdot \dots \cdot P(s_{n-1}, s_n)$
    - extends uniquely to all sets of paths in the  $\sigma$ -algebra
- All omega regular properties are measurable





# Markov decision processes (MDPs)

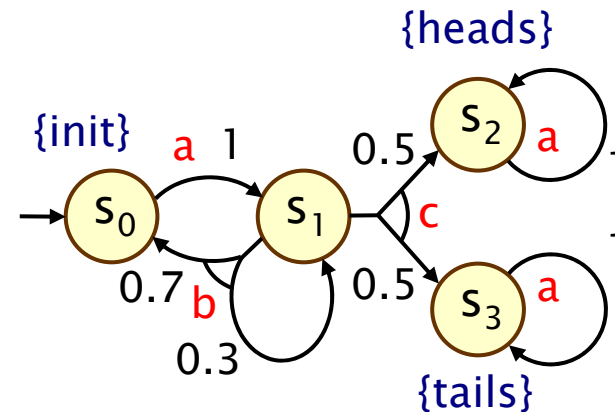
- Model **nondeterministic** as well as **probabilistic** behaviour
  - e.g. for concurrency, under-specification, abstraction...
  - extension of discrete-time Markov chains
  - nondeterministic choice between probability distributions

- Formally, an MDP is a tuple

- $(S, s_{\text{init}}, \text{Steps}, L)$

- where:

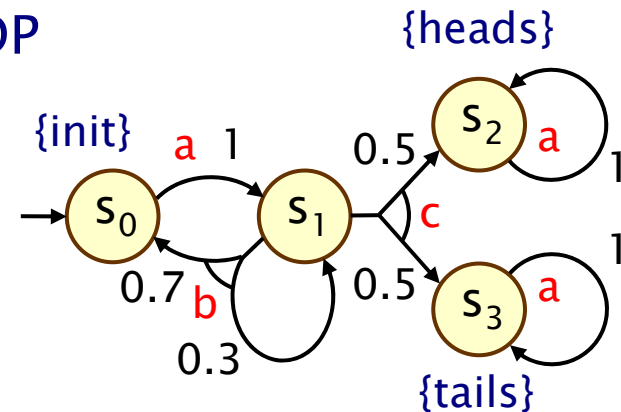
- $S$  is a set of states
- $s_{\text{init}} \in S$  is the initial state
- **Steps** :  $S \rightarrow 2^{\text{Act} \times \text{Dist}(S)}$  is the transition probability function
- $L : S \rightarrow 2^{\text{AP}}$  is a labelling function
  
- **Act** is a set of actions, **AP** is a set of atomic propositions
- **Dist(S)** is the set of discrete probability distributions over  $S$



# Paths and adversaries

- A (finite or infinite) **path** through an MDP

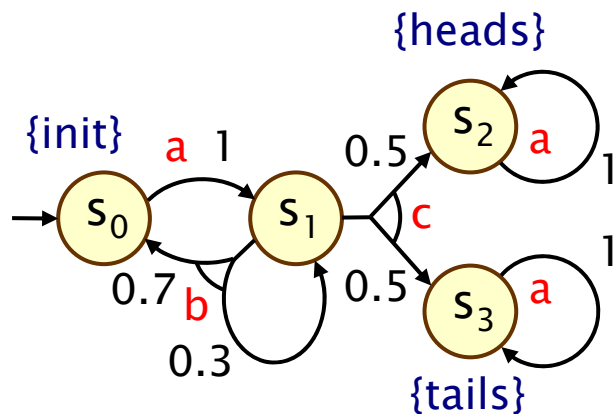
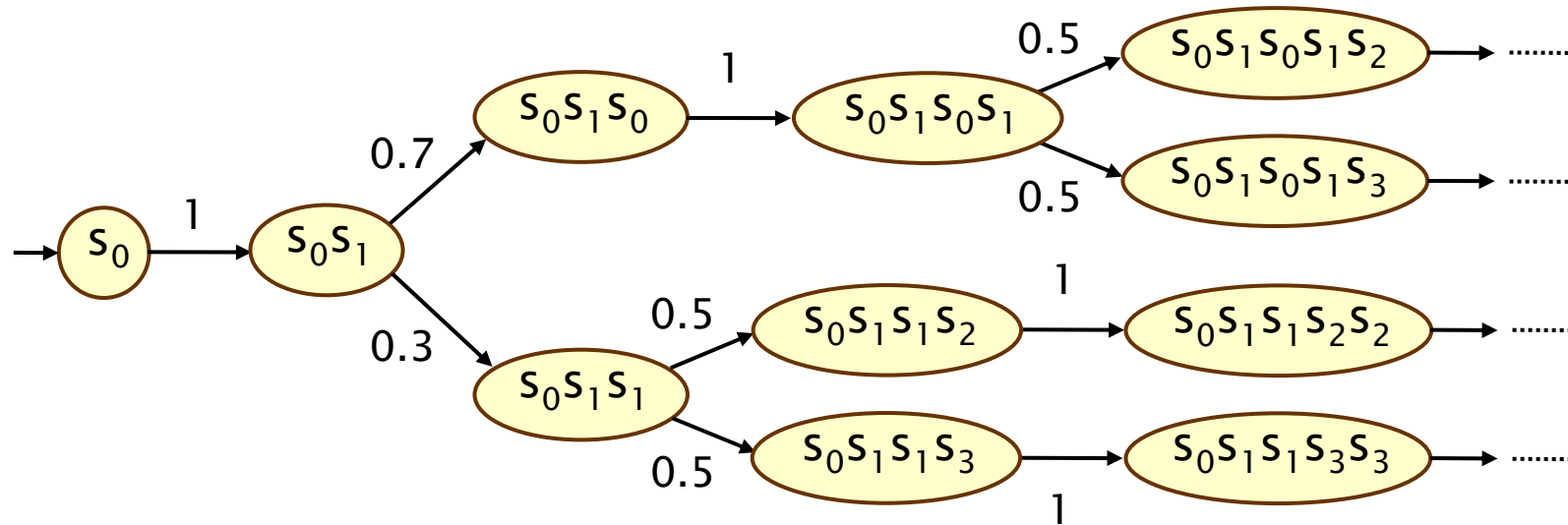
- is a sequence of (connected) states
- represents an execution of the system
- resolves both the probabilistic and nondeterministic choices



- An **adversary** (aka. “scheduler” or “policy”) of an MDP
  - is a resolution of nondeterminism only
  - is (formally) a mapping from finite paths to distributions
  - results in a fully probabilistic model
  - i.e. an (infinite–state) Markov chain over finite paths
  - on which we can define a probability space over infinite paths
- Adversary  $A$  is **simple** iff:  $A(s_1 \dots s_n) = A(s_n)$  for all  $s_1 \dots s_n$ 
  - in this case, resulting model reduces to finite Markov chain

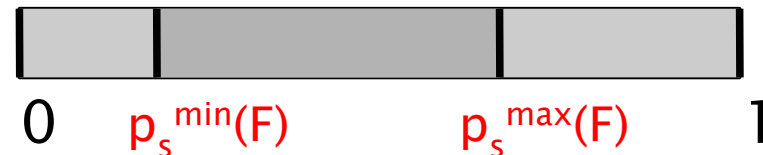
# Example adversary

- Fragment of DTMC for adversary which picks **b** then **c** in  $s_1$



# Probabilistic reachability for MDPs

- An adversary  $A$  induces, for each state  $s$  in the MDP:
  - a set of infinite paths  $\text{Path}^A(s)$
  - a probability space  $\text{Pr}_s^A$  over  $\text{Path}^A(s)$
- Probabilistic reachability (for a set of goal states  $F \subseteq S$ )
  - probability of reaching  $F$  from state  $s$  under adversary  $A$
  - $p_s^A(F) = \text{Pr}_s^A \{ s_0 s_1 s_2 s_3 \dots \in \text{Path}^A(s) \mid s_i \in F \text{ for some } i \}$
- Minimum/maximum probabilities over all adversaries
  - $p_s^{\min}(F) = \inf_A p_s^A(F)$
  - $p_s^{\max}(F) = \sup_A p_s^A(F)$
  - simple adversaries suffice
- Used to reason about best/worst-case behaviour
  - e.g. maximum probability of an error occurring



# Probabilistic model checking for MDPs

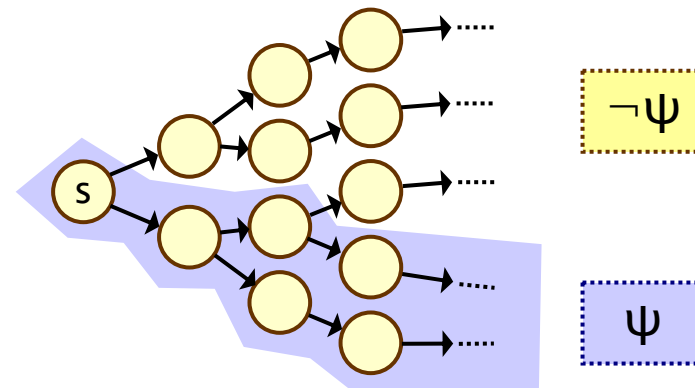
- **Also: Bounded reachability properties**
  - e.g. “min. probability of algorithm termination within T steps”
- **Also: Cost- and reward-based properties**
  - augment states/transitions of MDP with real-valued costs
  - define properties as random variables over  $\text{Path}^A(s)$
  - e.g. “max. expected power consumption for the duration of the protocol”

- **Probabilistic temporal logics**

- e.g. PCTL extends CTL
- existential quantification over paths (E,A operators) replaced with probabilistic P operator

- e.g.  $P_{<0.01} [\diamond \text{error}]$

- $s \models P_{\sim p} [\psi] \Leftrightarrow \Pr^A_s \{ \omega \in \text{Path}^A(s) \mid \omega \models \psi \} \sim p$  for all A



# Probabilistic model checking for MDPs

- Efficient model checking algorithms exist:
  - main component: computation of reachability probabilities
    - **linear optimisation problem** (polynomial complexity)
    - or **value iteration** (dynamic programming) – simple iterative numerical method; more efficient in practice
    - also: graph-based model analysis for qualitative verification
  - **best/worst** case simple adversary can also be generated

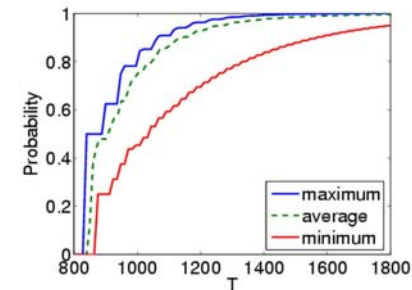
- Focus on quantitative results and analysis

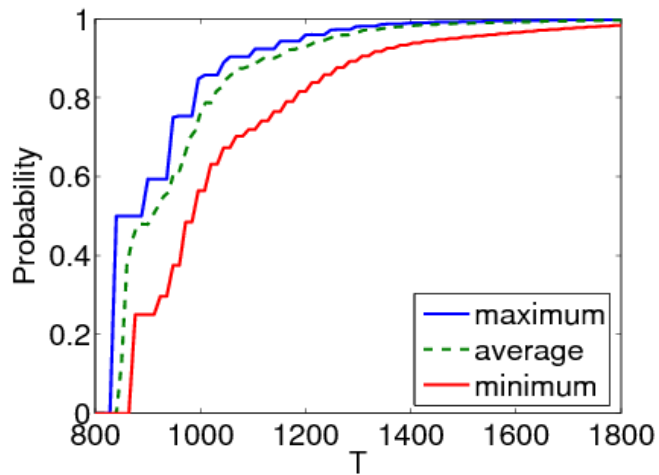
- for PCTL properties with P as the outermost operator, we allow these forms:

- $P_{\min=?} [\psi]$  and  $P_{\max=?} [\psi]$

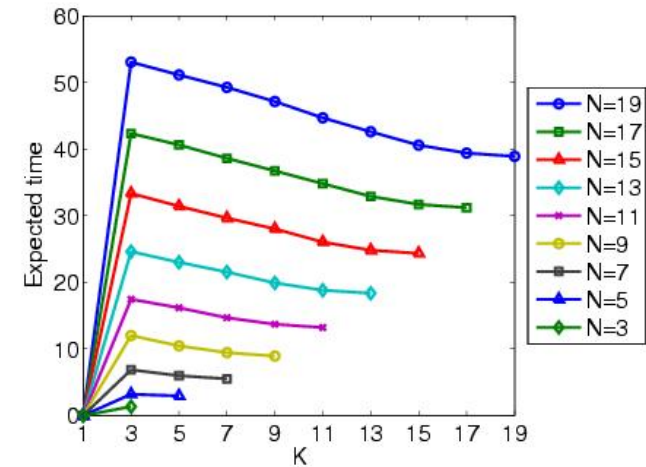
- i.e. “**what is the minimum/maximum probability (over all adversaries) that path formula  $\psi$  is true?**”

- useful to spot patterns/trends

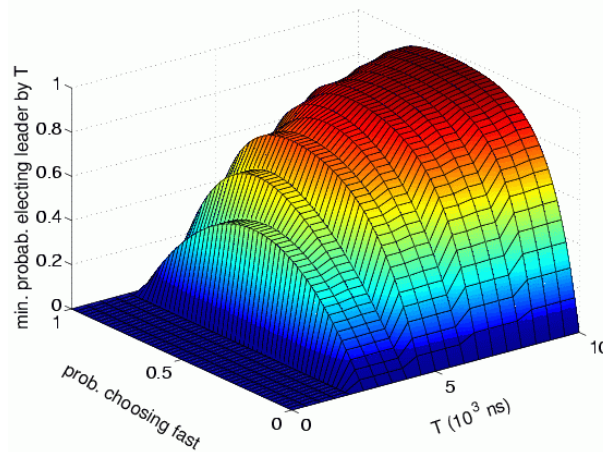




**CSMA/CD protocol:**  
 Min/max/average probability that a message is sent successfully by time T



**Self-stabilisation:**  
 Worst-case expected number of steps to stabilise for initial configurations with K tokens amongst N processes



**Firewire protocol:**  
 Optimum probability of leader election by time T for various coin biases

# PRISM

- **PRISM: Probabilistic model checker**
  - developed at Birmingham, Oxford since approx. 2000
- **Support for MDPs, DTMCs, CTMCs**
  - models specified in probabilistic guarded command language
- **Model checking of PCTL, LTL, rewards, ...**
  - efficient symbolic (BDD-based) implementations
- **Applied to case studies across many application domains**
  - communication protocols, security, biology, ...
  - anomalous behaviour/useful insight obtained in many cases
- **See: [www.prismmodelchecker.org](http://www.prismmodelchecker.org)**
- **But major challenges remain, e.g.**
  - state-space explosion
  - automating model extraction





# Aims

- Combine software verification via model checking
  - efficient verification of C/Java via **abstraction-refinement**
  - e.g. “buffer overflow does **not** occur in this program, for all executions”
  - existing tools: SLAM, BLAST, SATABS, Terminator
- with quantitative probabilistic model checking
  - focus on software where **failure** is unavoidable
    - e.g. network protocols, esp. wireless
  - aim for network utilities and quantitative properties
    - concurrency, user input -> **nondeterministic** choice
    - failure, randomisation -> **probabilistic** choice
  - also include real-time and resources

# Overview

## 2. Abstraction & refinement for MDPs



# Abstraction

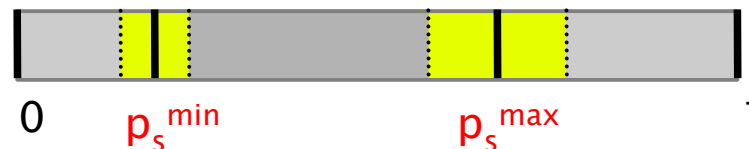
- Very successful in (non-probabilistic) model checking
  - essential for verification of large/infinite-state systems
- Construct abstract model **A** of concrete model **M**
  - details not relevant to some property of interest removed
  - e.g. partition of state space based on a set of predicates
- Non-probabilistic case: existential abstraction
  - conservative: existence of path in **M** implies existence in **A**
  - hence can model check **A** to verify safety properties of **M**
- Abstraction-refinement
  - automate process of constructing abstraction
  - information from model checking process can be used to refine the abstraction (or validate the property)
  - e.g. CEGAR (counterexample-guided abstraction refinement) – check if counterexample is spurious and use to refine

# Abstraction of MDPs

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



- what form does the abstraction of an MDP take?
- Our approach: two-player stochastic games [QEST'06]
- Key idea: separate two forms of nondeterminism
  - (a) from abstraction and (b) from original MDP
  - then generate separate lower/upper bounds for min/max



- gives quantitative measure of utility of abstraction
  - basis of quantitative abstraction-refinement framework

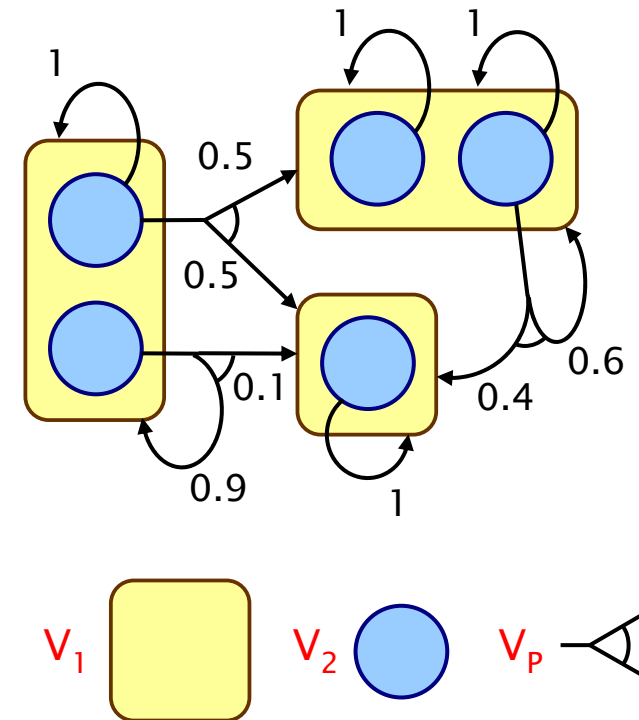
# Stochastic two-player games

- Stochastic two-player games [Shapley, Condon]

- two (nondeterministic) players and probabilistic choice
- (MDPs are 1 player stochastic games)

- Game  $G = ((V, E), v_{init}, (V_1, V_2, V_p), \delta)$

- $(V, E)$  is a finite directed graph
- $v_{init}$  is the initial vertex
- $(V_1, V_2, V_p)$  is a partition of  $V$ : ‘player 1’, ‘player 2’ and ‘probabilistic’
- $\delta : V_p \rightarrow \text{Dist}(V)$  is a probabilistic transition function



- Execution of  $G$ : successor vertex chosen:

- by player 1 / 2 for  $V_1 / V_2$  vertices
- at random ( $\delta$ ) for  $V_p$  vertices

# Properties of stochastic games

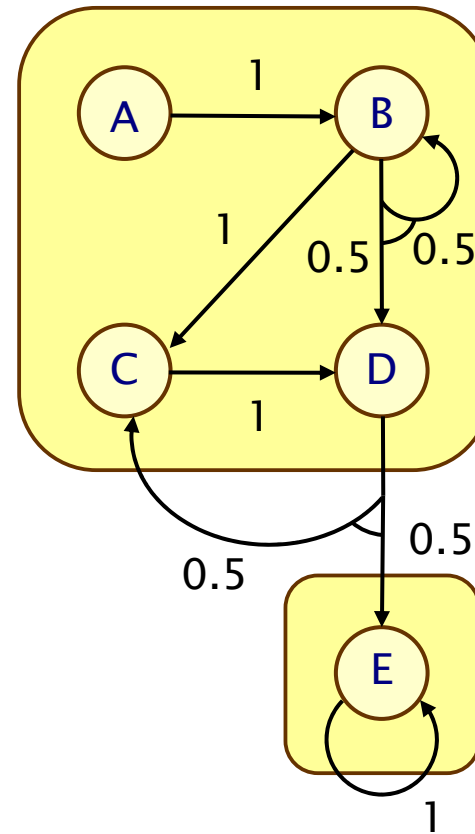
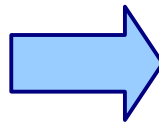
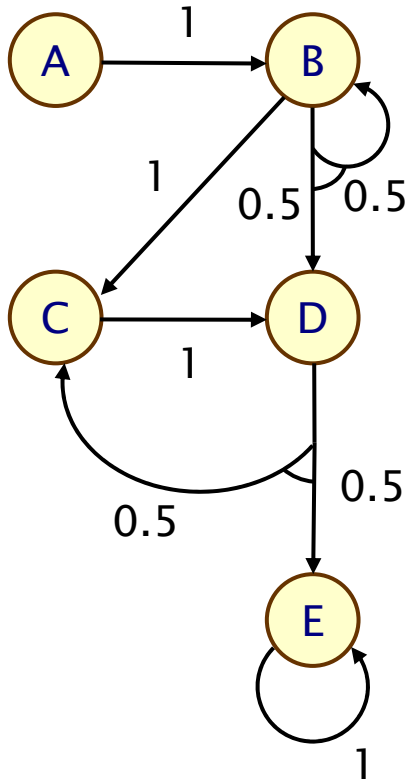
- Resolution of nondeterminism in a stochastic game
  - is done by a pair of **strategies** for players 1 and 2:  $(\sigma_1, \sigma_2)$
  - under which the behaviour of the game is fully probabilistic
  - which induces a probability space over infinite paths
- Probabilistic reachability of vertex goal set  $F \subseteq V$ 
  - $p_v^{\sigma_1, \sigma_2}(F)$  probability of reaching  $F$  from vertex  $v$  under  $(\sigma_1, \sigma_2)$
- Optimal probabilities for player 1 and player 2
  - $\sup_{\sigma_1} \inf_{\sigma_2} p_v^{\sigma_1, \sigma_2}(F)$  and  $\sup_{\sigma_2} \inf_{\sigma_1} p_v^{\sigma_1, \sigma_2}(F)$
  - e.g. “no matter what player 2 does, player 1 can guarantee  $F$  is reached with at least this probability”
  - computable via simple iterative methods, similar to MDPs
  - corresponding strategies also obtained

# Games as abstractions of MDPs

- Abstraction of an MDP is a two-player stochastic game
  - based on a partition  $P_S$  of MDP state space  $S$
  - $V_1$  vertices are elements of  $P_S$  (subsets of  $S$ )
  - $V_2$  vertices are sets of prob. distributions (“states of MDP”)
  - $V_P$  vertices are single probability distributions (over  $V_1$ )
  - strict alternation between  $V_1$ ,  $V_2$ ,  $V_P$  vertices
- Player 1 controls nondeterminism from abstraction
  - selects a state of the original MDP from a subset of  $S$  (in  $P_S$ )
- Player 2 controls nondeterminism from original MDP
  - selects a single probability distribution from a set

# MDP $\rightarrow$ Game

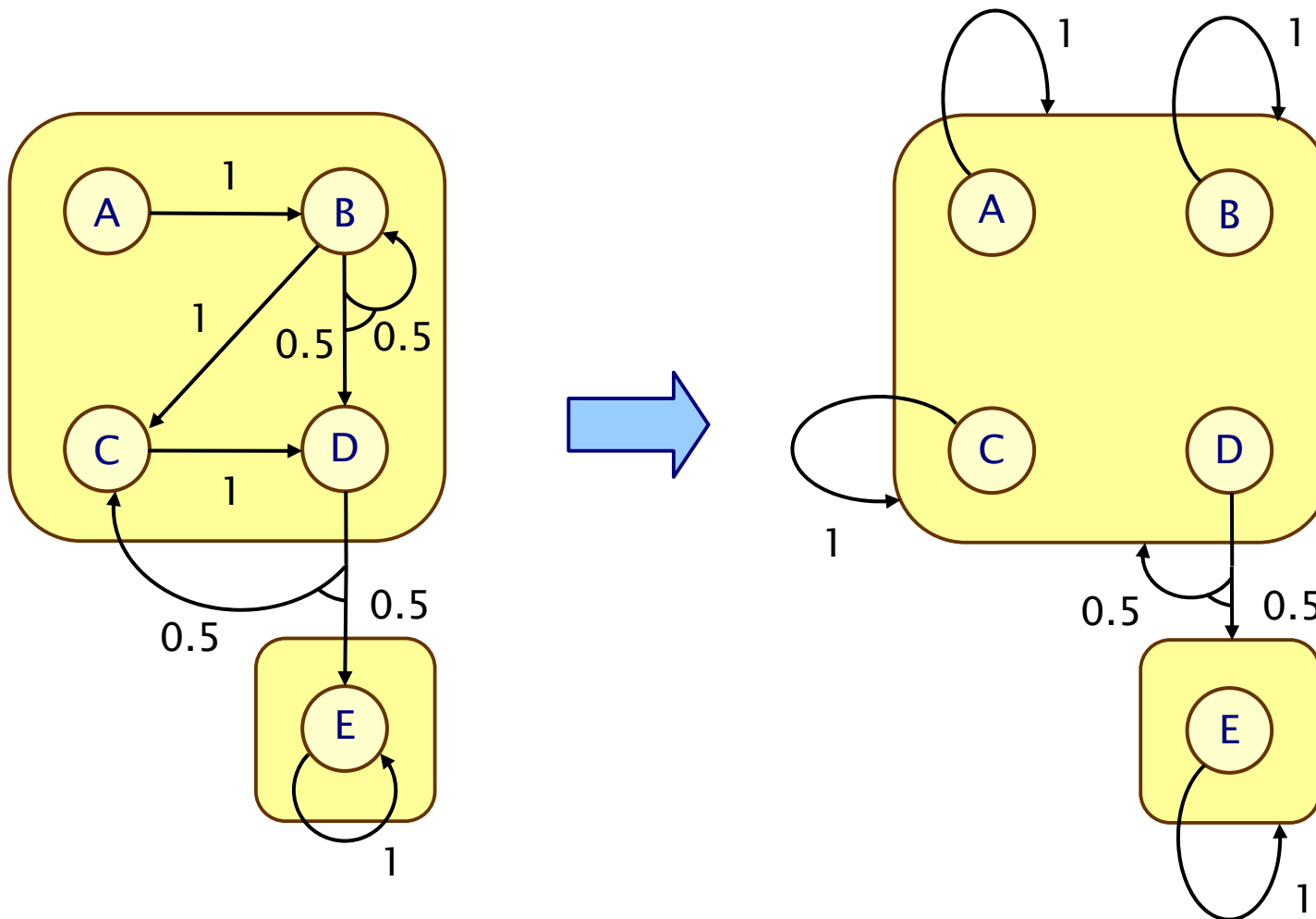
- Player 1 vertices are partition elements (abstract states)





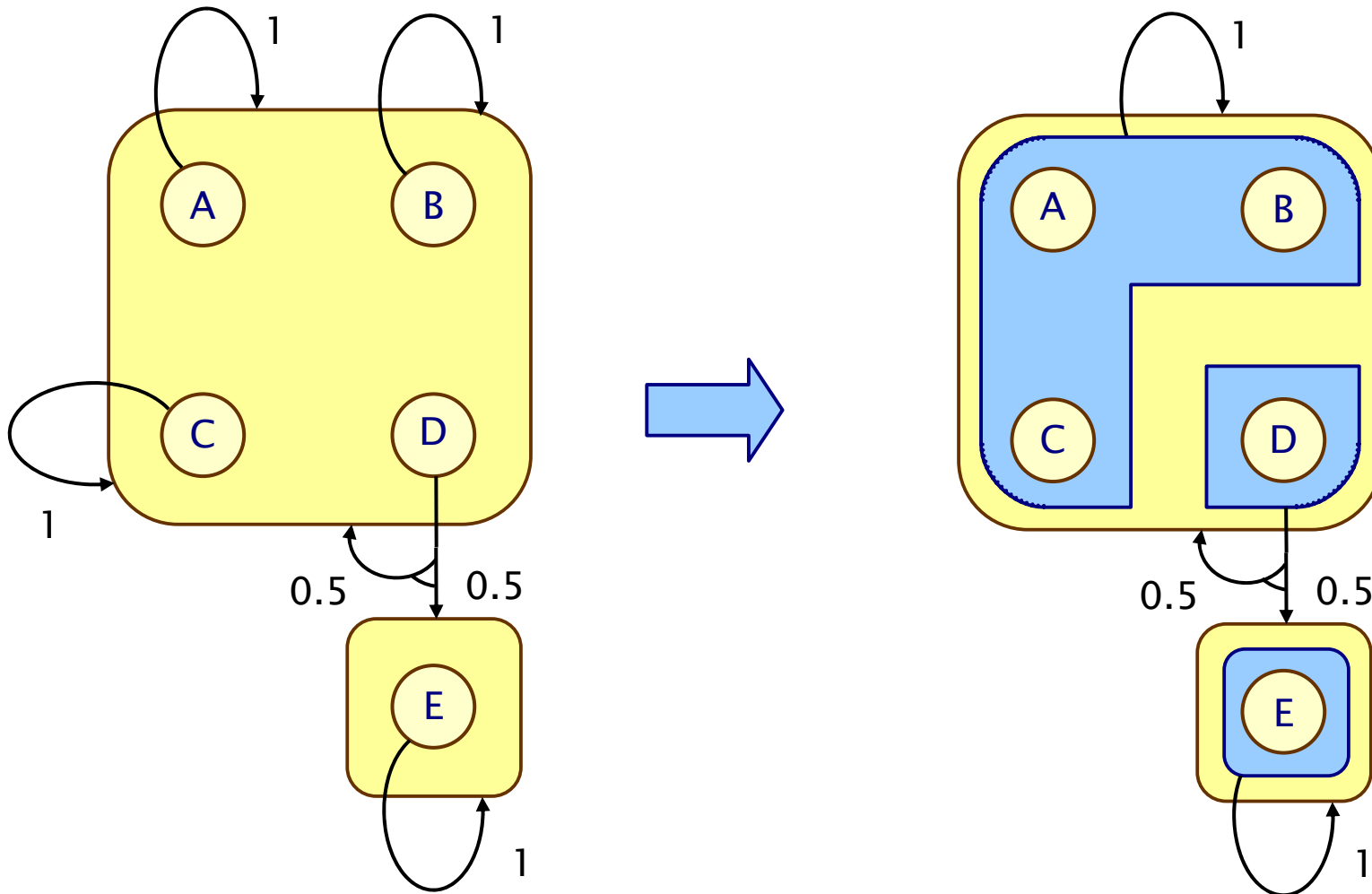
# MDP $\rightarrow$ Game

- (Sets of) distributions are lifted to the abstract state space



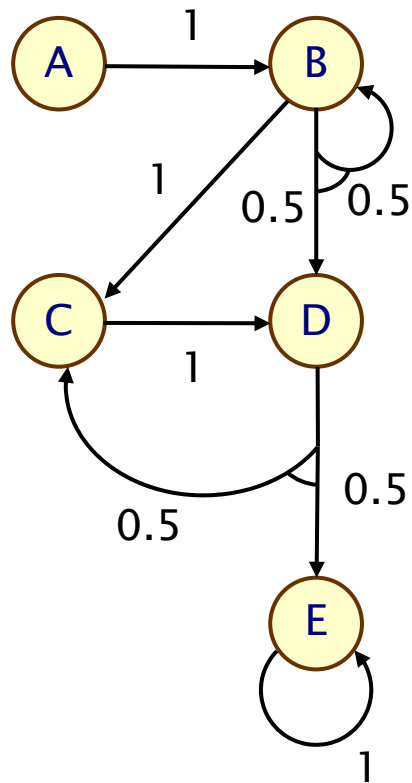
# MDP → Game

- States with same (sets of) choices form player **2** vertices

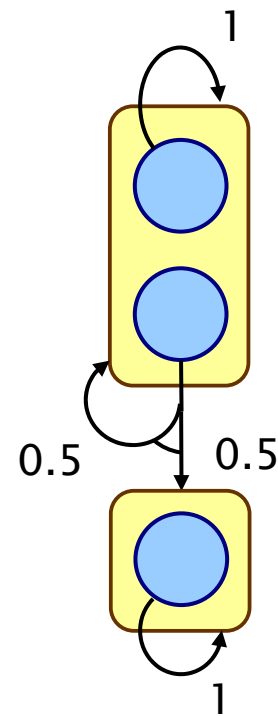
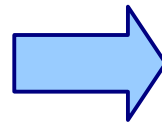


# MDP $\rightarrow$ Game

- Complete transformation:



MDP



Abstraction (game)

# Analysis of the abstraction

- For a stochastic game built from an MDP and partition  $P_S$
- Let  $s \in S$  be an MDP state,  $v \in V$  the corresponding game vertex (i.e.  $s \in v$ ) and  $F \in P_S$  a set of goal states
- Analysis of game yields lower/upper bounds for MDP:

$$\inf_{\sigma_1, \sigma_2} p_v^{\sigma_1, \sigma_2}(F) \leq p_s^{\min}(F) \leq \sup_{\sigma_1} \inf_{\sigma_2} p_v^{\sigma_1, \sigma_2}(F)$$

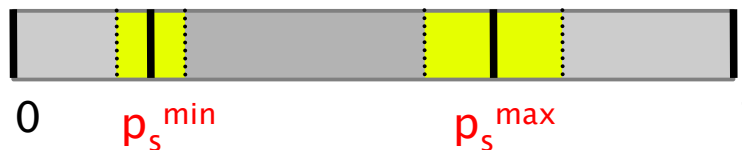
$$\sup_{\sigma_2} \inf_{\sigma_1} p_v^{\sigma_1, \sigma_2}(F) \leq p_s^{\max}(F) \leq \sup_{\sigma_1, \sigma_2} p_v^{\sigma_1, \sigma_2}(F)$$

# Analysis of the abstraction

- For a stochastic game built from an MDP and partition  $P_S$
- Let  $s \in S$  be an MDP state,  $v \in V$  the corresponding game vertex (i.e.  $s \in v$ ) and  $F \in P_S$  a set of goal states
- Analysis of game yields lower/upper bounds for MDP:

$$\inf_{\sigma_1, \sigma_2} p_v^{\sigma_1, \sigma_2}(F) \leq p_s^{\min}(F) \leq \sup_{\sigma_1} \inf_{\sigma_2} p_v^{\sigma_1, \sigma_2}(F)$$
$$\sup_{\sigma_2} \inf_{\sigma_1} p_v^{\sigma_1, \sigma_2}(F) \leq p_s^{\max}(F) \leq \sup_{\sigma_1, \sigma_2} p_v^{\sigma_1, \sigma_2}(F)$$

min/max reachability probabilities for original MDP



# Analysis of the abstraction

- For a stochastic game built from an MDP and partition  $P_S$
- Let  $s \in S$  be an MDP state,  $v \in V$  the corresponding game vertex (i.e.  $s \in v$ ) and  $F \in P_S$  a set of goal states
- Analysis of game yields lower/upper bounds for MDP:

$$\begin{array}{ccccc}
 \inf_{\sigma_1, \sigma_2} p_v^{\sigma_1, \sigma_2}(F) & \leq & p_s^{\min}(F) & \leq & \sup_{\sigma_1} \inf_{\sigma_2} p_v^{\sigma_1, \sigma_2}(F) \\
 \sup_{\sigma_2} \inf_{\sigma_1} p_v^{\sigma_1, \sigma_2}(F) & \leq & p_s^{\max}(F) & \leq & \sup_{\sigma_1, \sigma_2} p_v^{\sigma_1, \sigma_2}(F)
 \end{array}$$

optimal probabilities for player 1, player 2 in game

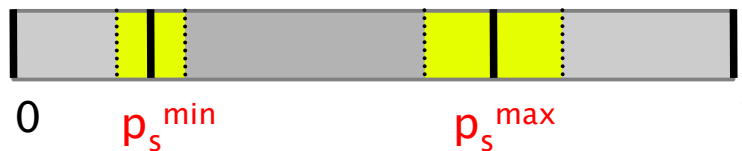


# Analysis of the abstraction

- For a stochastic game built from an MDP and partition  $P_S$
- Let  $s \in S$  be an MDP state,  $v \in V$  the corresponding game vertex (i.e.  $s \in v$ ) and  $F \in P_S$  a set of goal states
- Analysis of game yields lower/upper bounds for MDP:

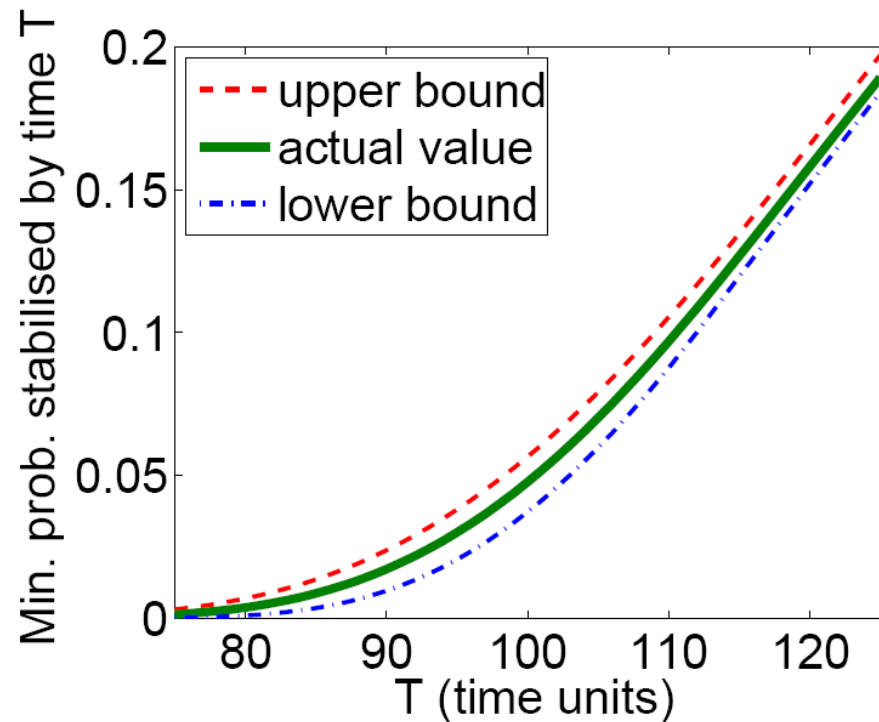
$$\begin{aligned} \inf_{\sigma_1, \sigma_2} p_v^{\sigma_1, \sigma_2}(F) &\leq p_s^{\min}(F) \leq \sup_{\sigma_1} \inf_{\sigma_2} p_v^{\sigma_1, \sigma_2}(F) \\ \sup_{\sigma_2} \inf_{\sigma_1} p_v^{\sigma_1, \sigma_2}(F) &\leq p_s^{\max}(F) \leq \sup_{\sigma_1, \sigma_2} p_v^{\sigma_1, \sigma_2}(F) \end{aligned}$$

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



# Abstraction: Results

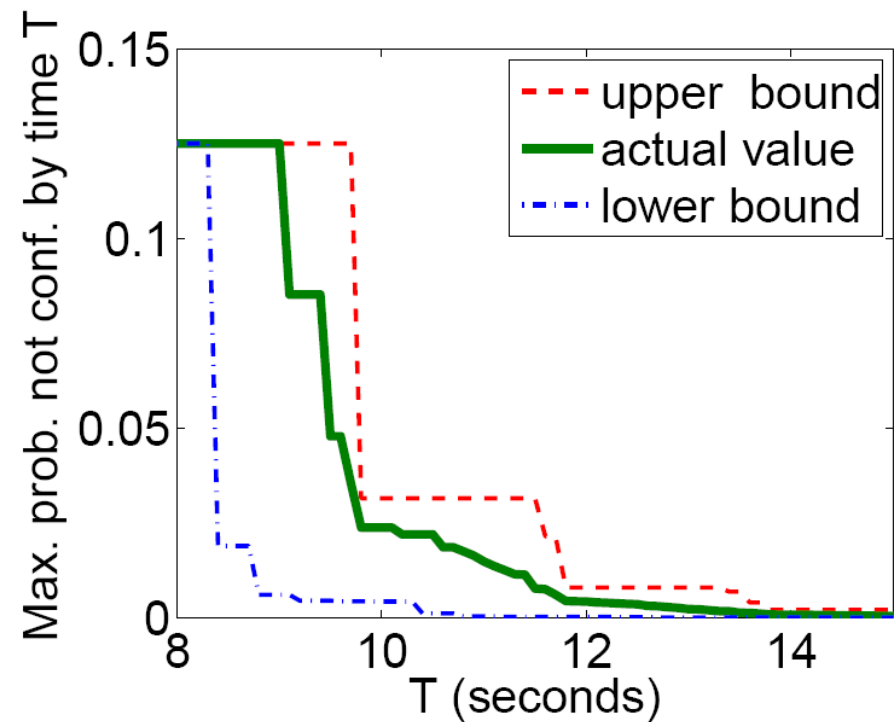
- Israeli & Jalfon's Self Stabilisation [IJ90]
  - protocol for obtaining a stable state in a token ring
  - minimum probability of reaching a stable state by time  $T$





# Abstraction: Results

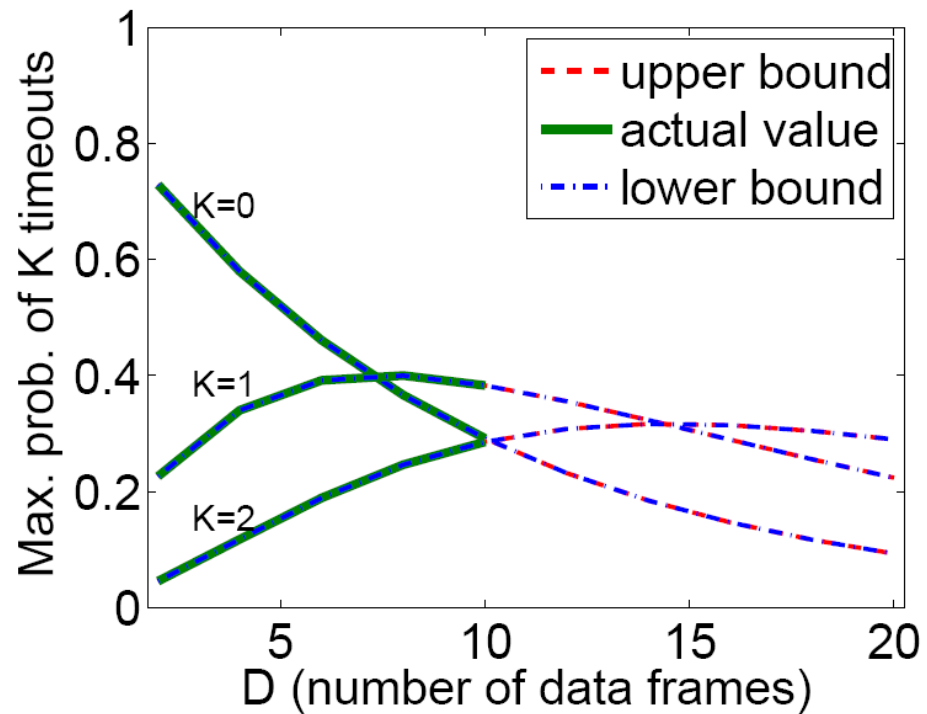
- IPv4 Zeroconf [CAG02]
  - protocol for obtaining an IP address for a new host
  - maximum probability the new host not configured by T



# Abstraction: Results

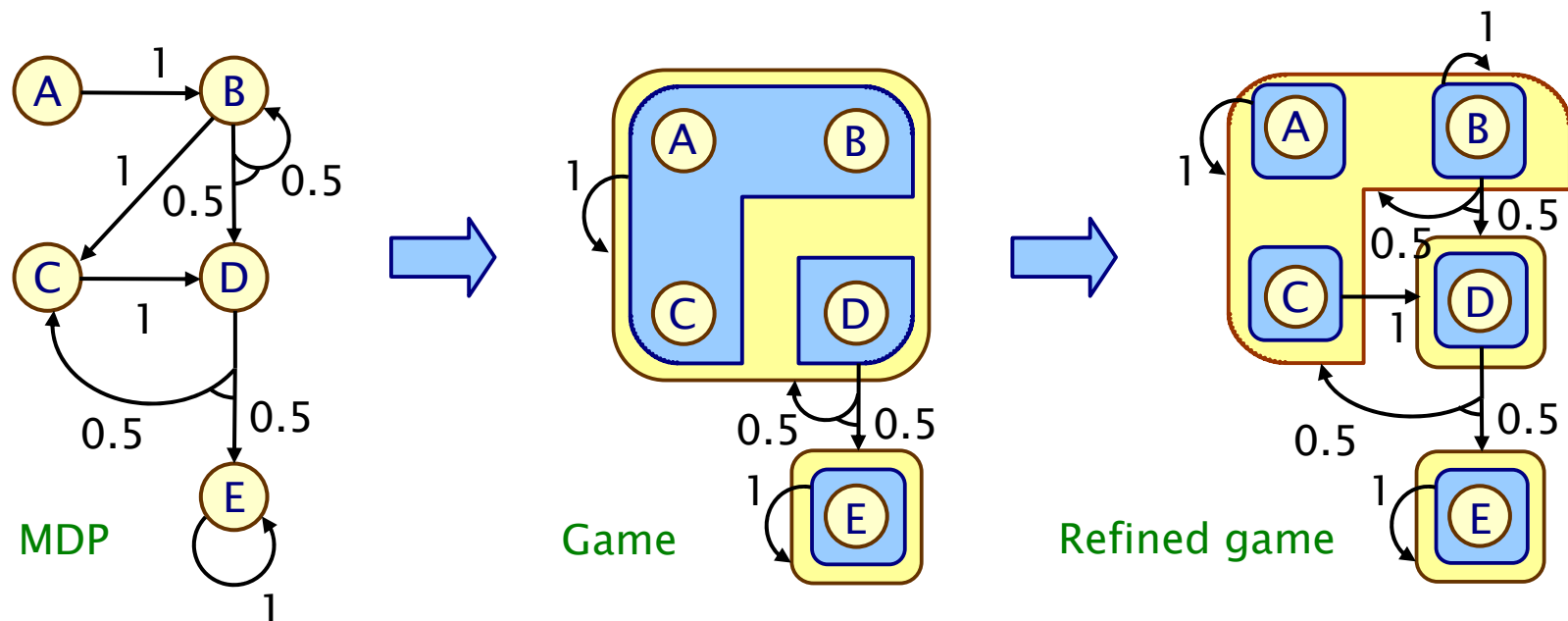
- Sliding Window Protocol

- protocol for sending data over an insecure medium
- maximum probability of K timeouts



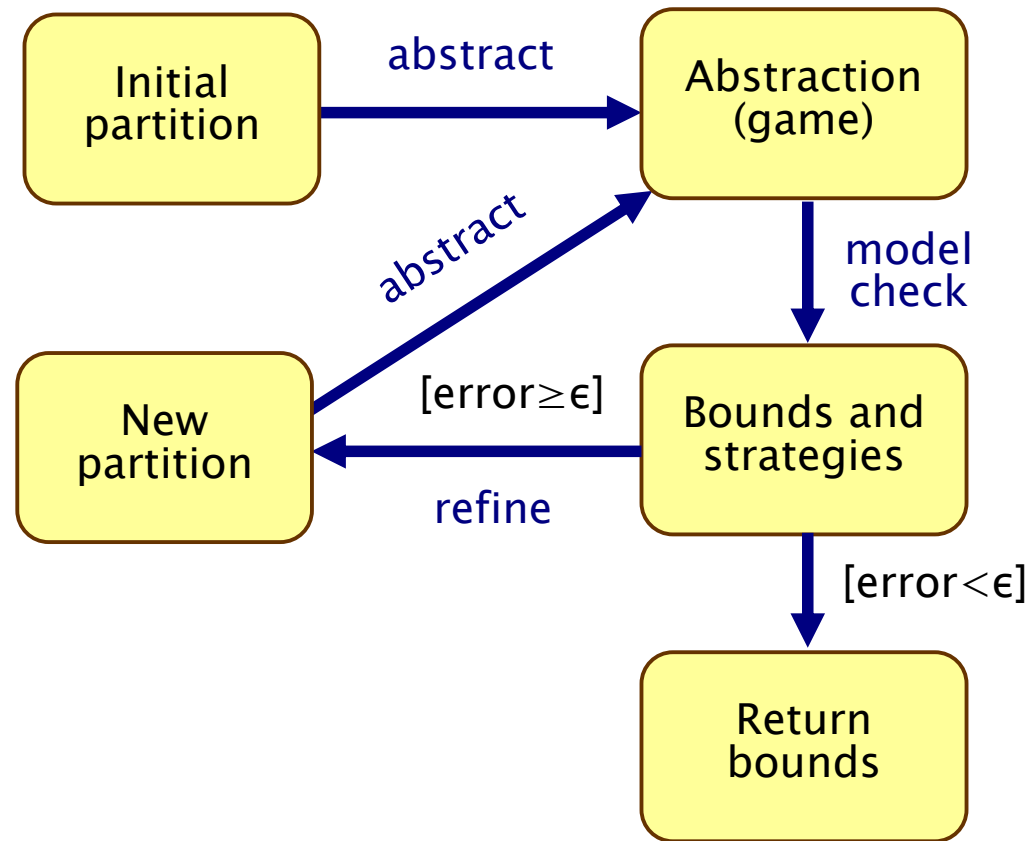
# Abstraction refinement

- 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
- Here, abstraction induced by a partition of the state space
  - a finer partition yields a more precise abstraction
  - bounds and strategies from game guides refinement



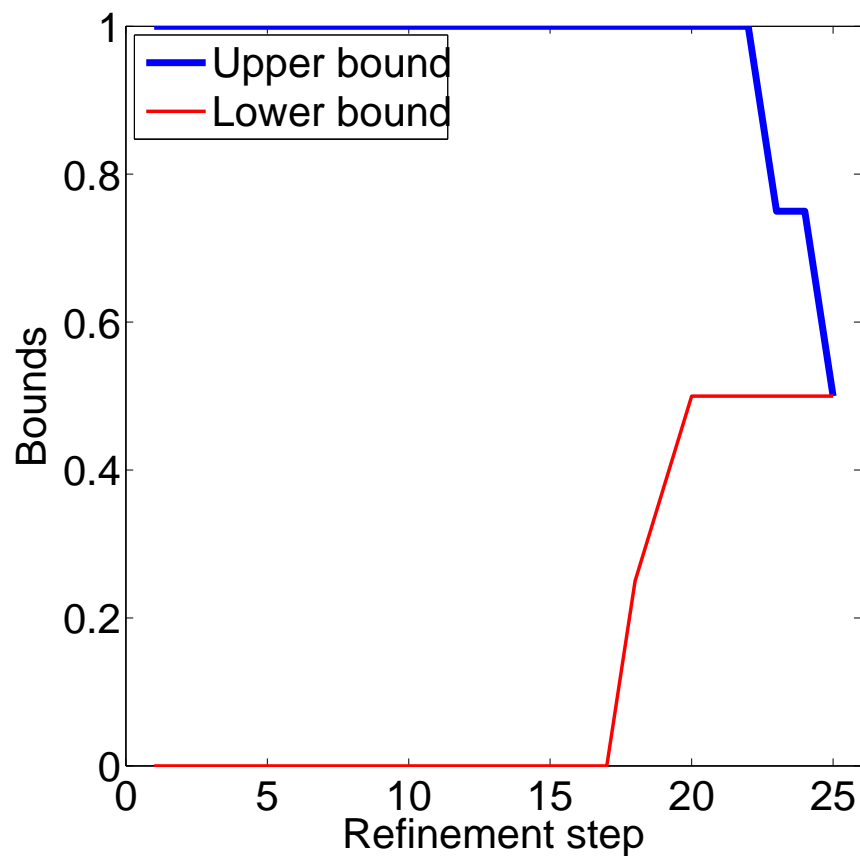
# Abstraction–refinement loop

- **Quantitative** abstraction–refinement loop for MDPs



# Partition refinement – Convergence

- IEEE 802.3 CSMA/CD Ethernet network protocol
  - used to ensure only one node is transmitting
  - minimum probability a station's backoff counter reaches 2



# Abstraction–refinement loop

- Implementations of quantitative abstraction–refinement...
- 1. Explicit–state prototype implementation [QEST'06]
  - confirms viability of game–based abstraction/refinement
- 2. Predicate abstraction of PRISM models [QAPL'08]
  - efficient construction of (manually specified) abstractions
- 3. Abstraction/refinement of software [VMCAI'09]
  - first probabilistic verification of real programming languages
- 4. Abstraction/refinement of PTAs [FORMATS'09]
  - outperforms existing PTA verification techniques
  - applications to real–time properties of software

# Overview

## 3. Verification of probabilistic software



# Probabilistic software

- Consider sequential ANSI C programs
  - support functions, pointers, arrays, but not dynamic memory allocation, unbounded recursion, floating point ops
- Add function `bool coin(double p)` for probabilistic choice
  - for modelling e.g. failures, randomisation
- Add function `int ndet(int n)` for nondeterministic choice
  - for modelling e.g. user input, unspecified function calls
- Focus on software where failure is unavoidable
  - e.g. network protocols/utilities, esp. wireless
- Quantitative properties based on probabilistic reachability
  - e.g. maximum probabilistic of unsuccessful data transmission
  - e.g. minimum expected number of packets sent



# Example – sample target program

```
bool fail = false;
int c = 0;
int main ()
{
    // nondeterministic
    c = num_to_send ();
    while (! fail && c > 0)
    {
        // probabilistic
        fail = send_msg ();
        c --;
    }
}
```

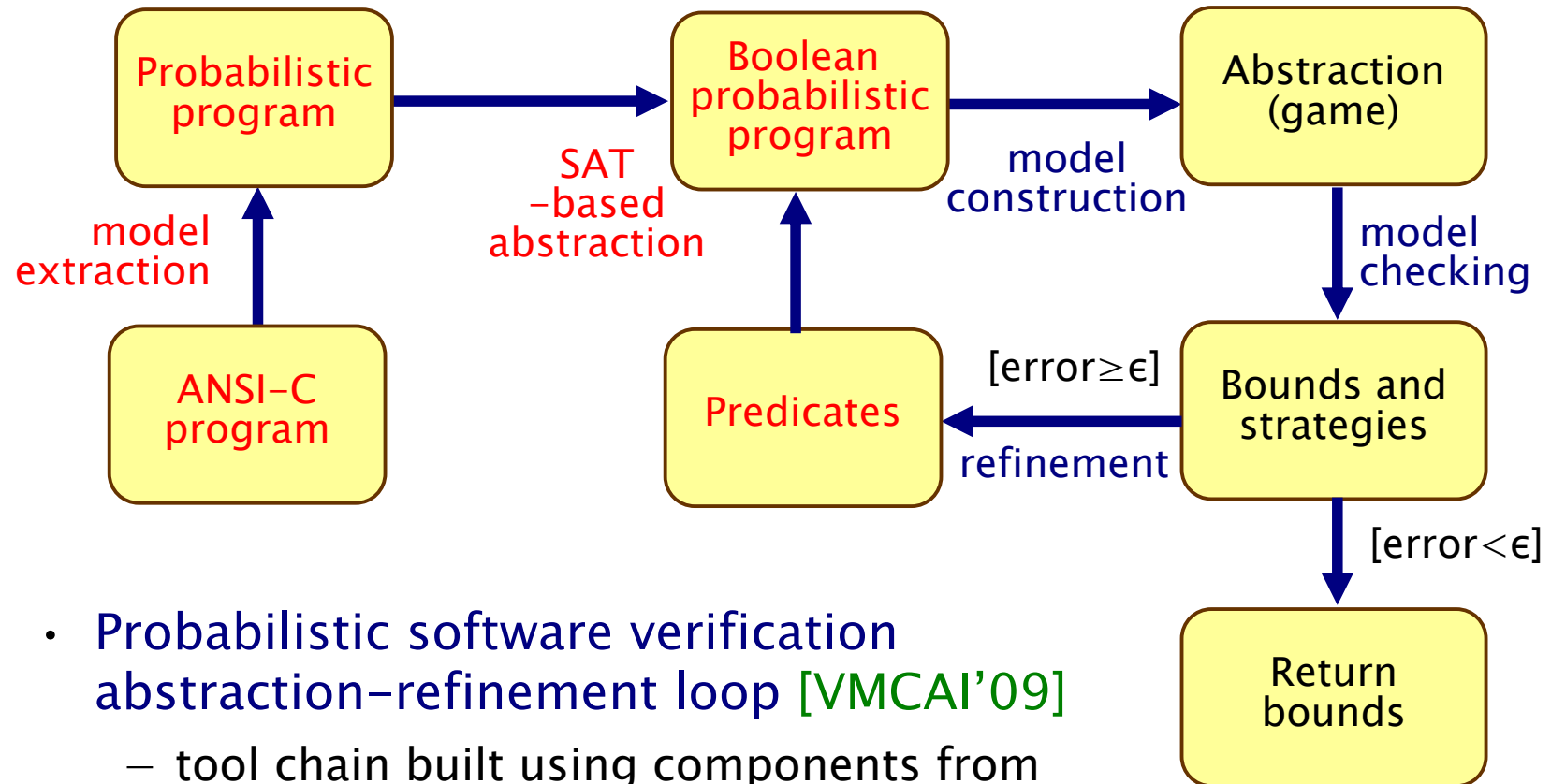
$\Phi$ : “what is the minimum/maximum probability of the program terminating with `fail` being true?”

# Example – simplified

```
bool fail = false;
int c = 0;
int main ()
{
    // nondeterministic
    c = ndet (3);
    while (! fail && c > 0)
    {
        // probabilistic
        fail = coin (0.1);
        c --;
    }
}
```

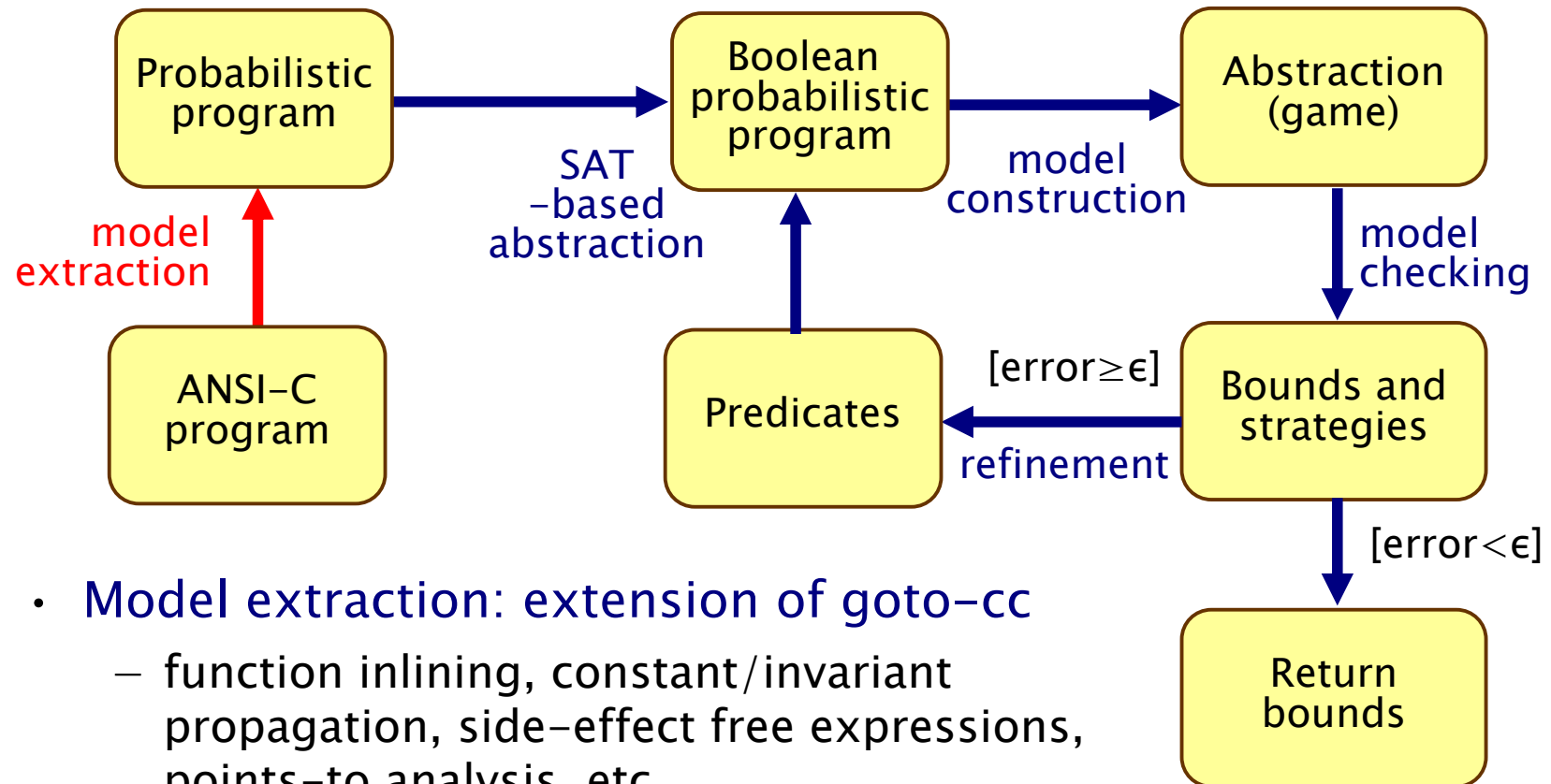
$\Phi$ : “what is the minimum/maximum probability of the program terminating with `fail` being true?”

# Abstraction-refinement loop



- Probabilistic software verification abstraction-refinement loop [VMCAI'09]
  - tool chain built using components from state-of-the-art tools:
  - goto-cc, SATABS, PRISM

# Abstraction-refinement loop

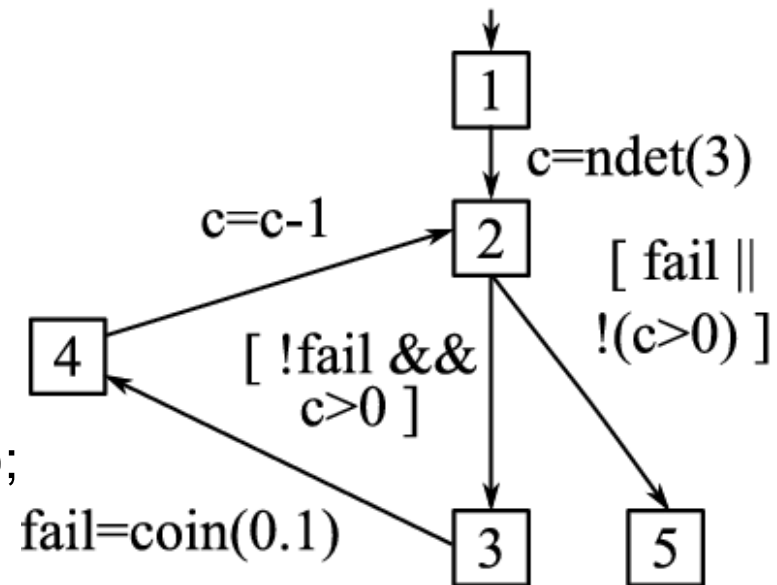


- **Model extraction: extension of goto-cc**
  - function inlining, constant/invariant propagation, side-effect free expressions, points-to analysis, etc.
- **Probabilistic program**
  - probabilistic control flow graph
  - Markov decision process (MDP) semantics

# Back to example

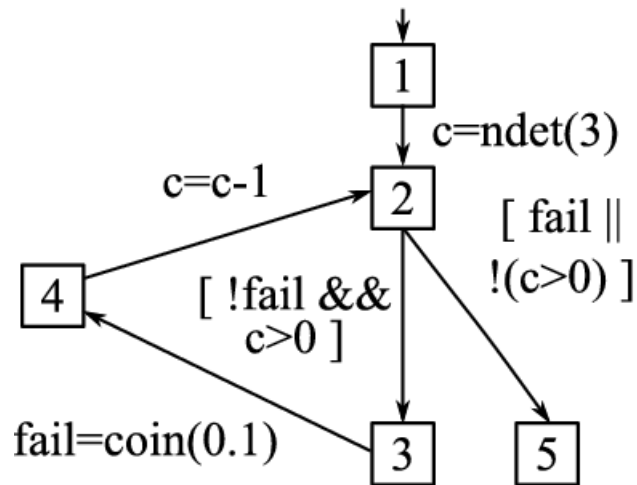
```
bool fail = false;
int c = 0;
int main ()
{
    // nondeterministic
    c = ndet (3);
    while (! fail && c > 0)
    {
        // probabilistic
        fail = coin (0.1);
        c --;
    }
}
```

## Probabilistic program

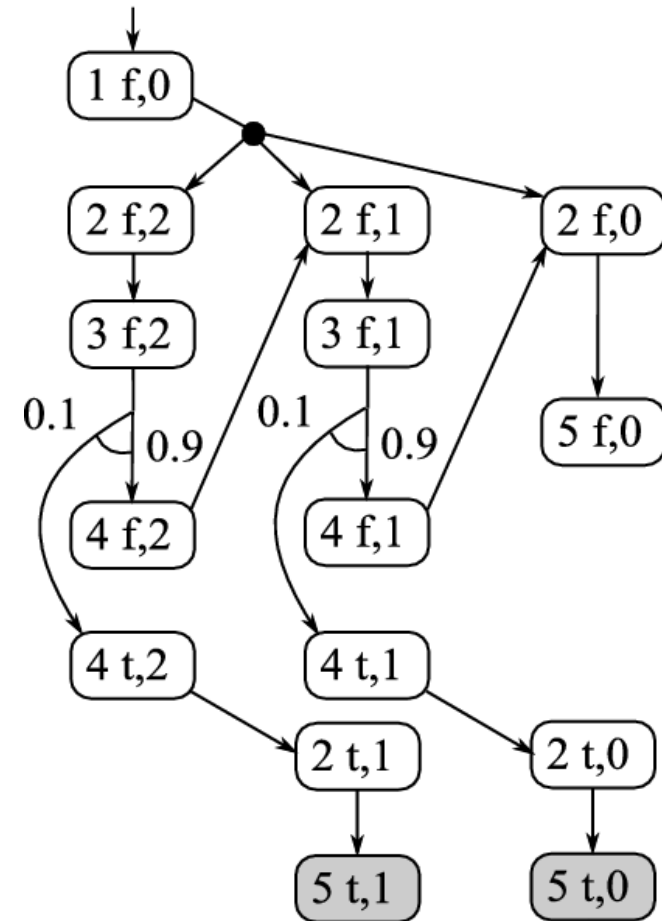


# Probabilistic program as MDP

Probabilistic program

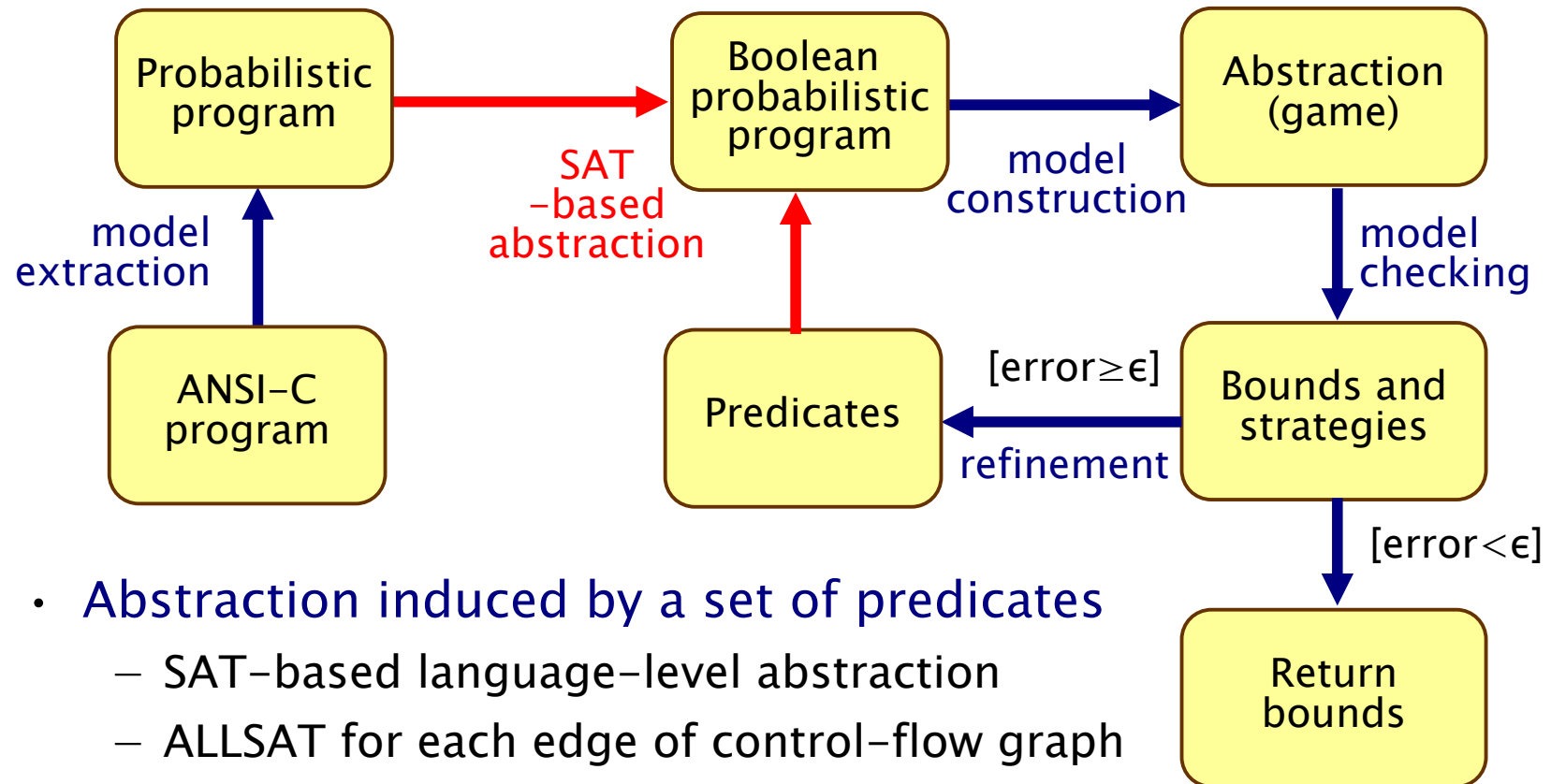


MDP semantics



minimum/maximum probability of the program terminating with **fail** being true is 0 and 0.19, respectively

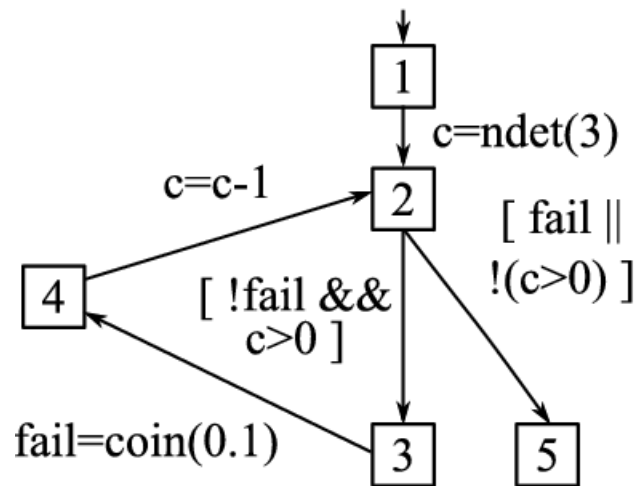
# Abstraction-refinement loop



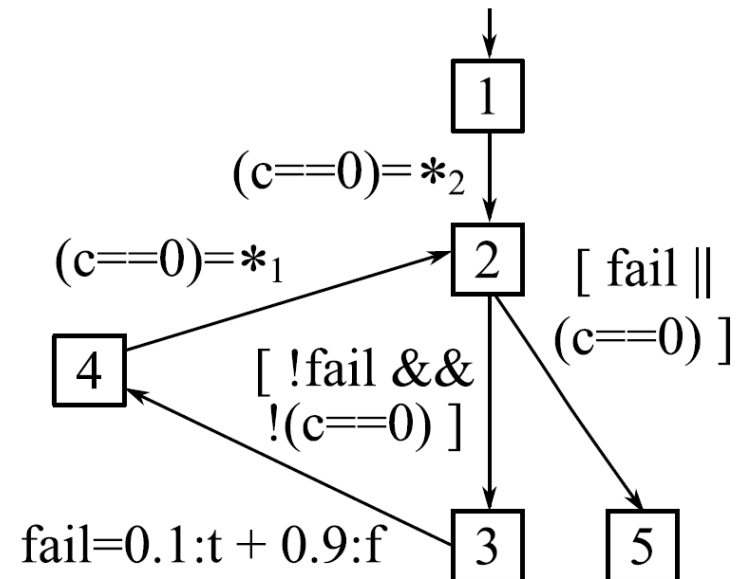
- Abstraction induced by a set of predicates
  - SAT-based language-level abstraction
  - ALLSAT for each edge of control-flow graph
  - implemented in extension of SATABS
- Boolean probabilistic program
  - (predicate) abstraction of probabilistic program
  - stochastic two player game semantics

# Back to example

Probabilistic program



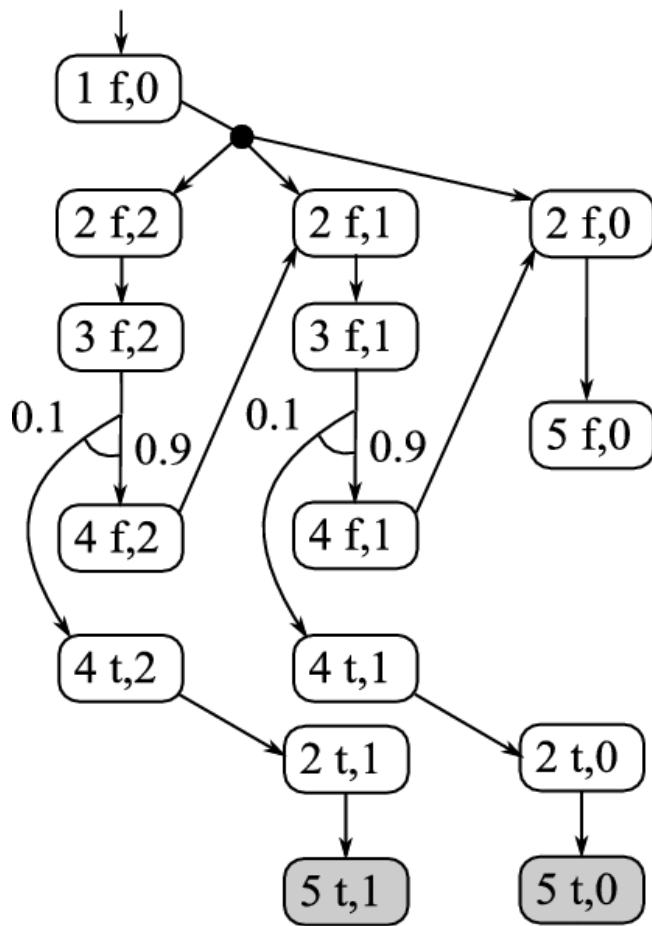
Boolean probabilistic program



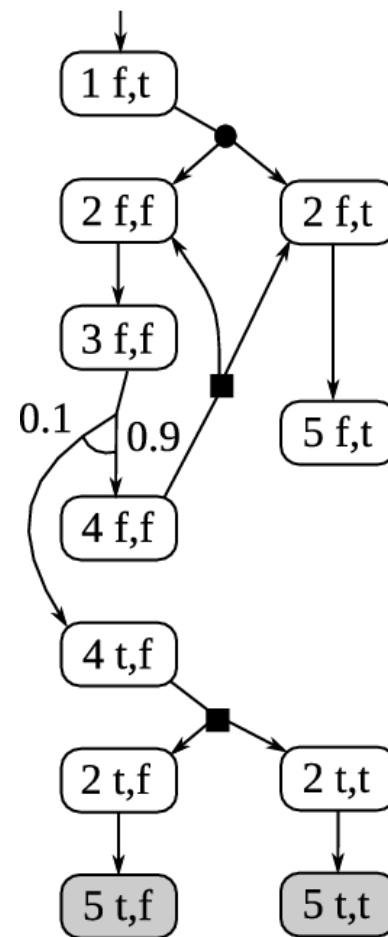


# Back to example

Concrete program (MDP)

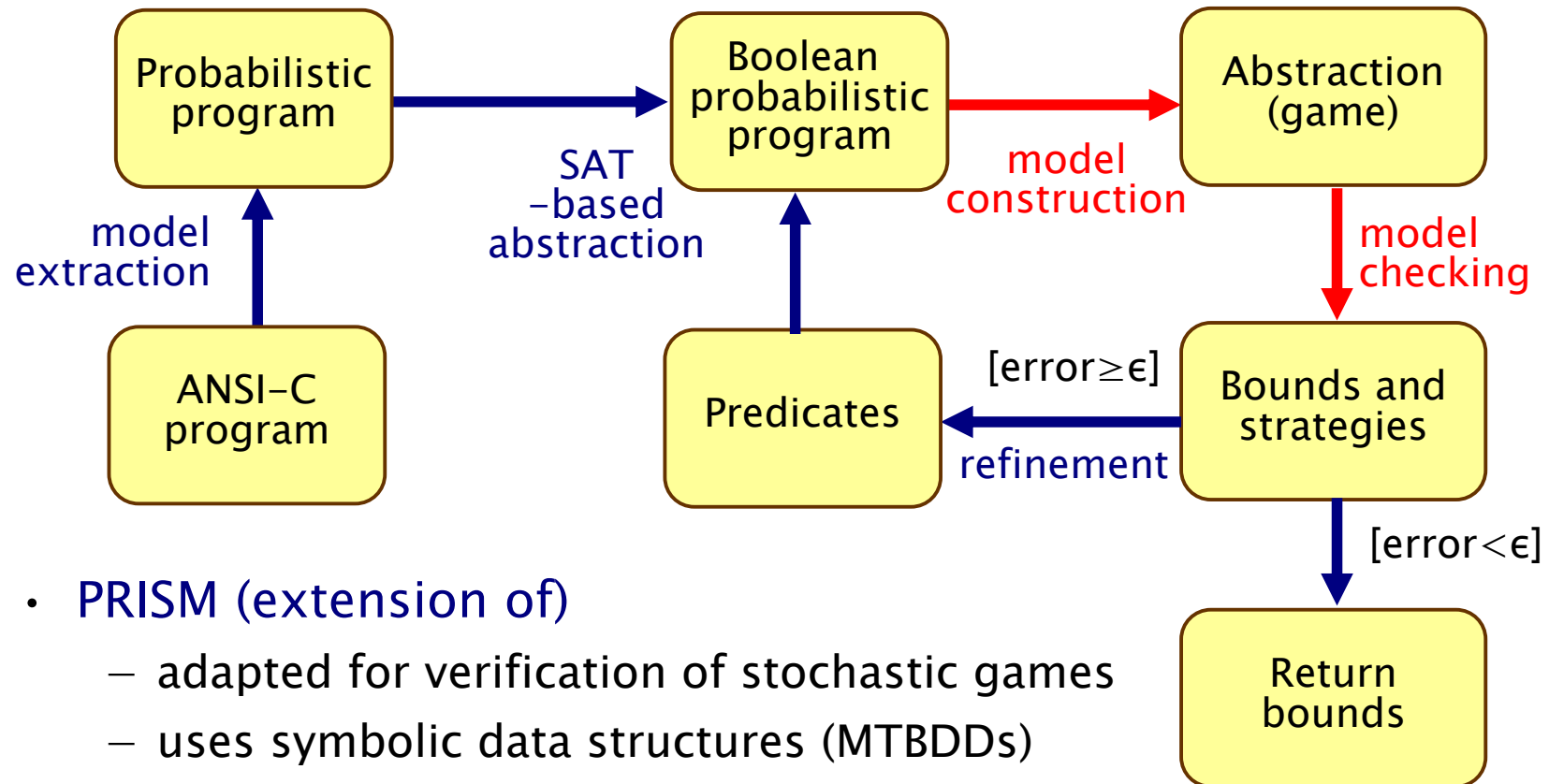


Abstraction (game)



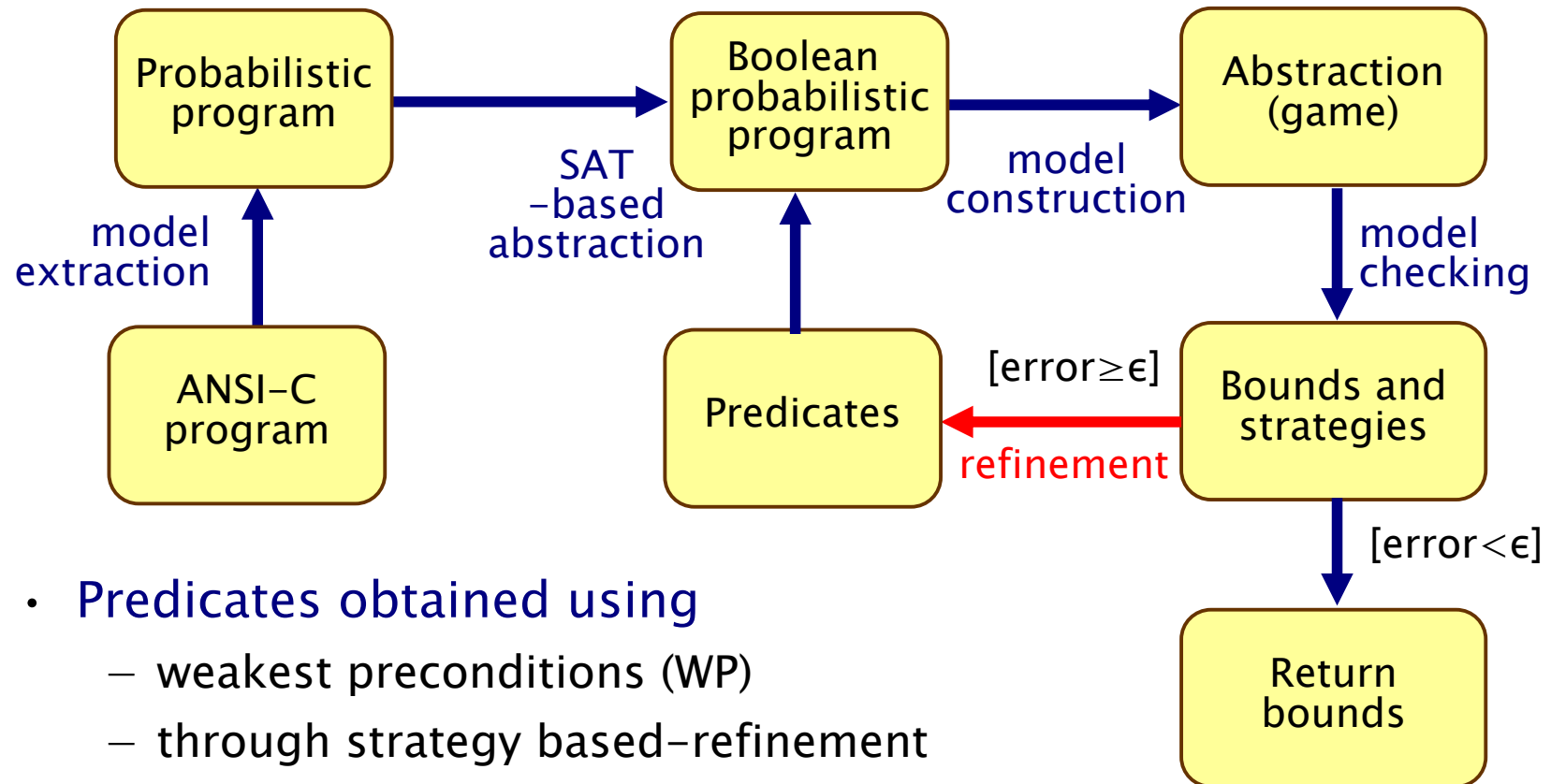
- PC fail, c==0 Abstract state
- Player 1 choice
- Player 2 choice
- Probabilistic choice

# Abstraction-refinement loop



- **PRISM** (extension of)
  - adapted for verification of stochastic games
  - uses symbolic data structures (MTBDDs)
- **Bounds and strategy**
  - returned for a given probabilistic or expected reachability property

# Abstraction–refinement loop



- Predicates obtained using
  - weakest preconditions (WP)
  - through strategy based–refinement
  - includes predicate localisation, reachability analysis, symbolic simulation,...

# Experimental results

- Successfully applied to several Linux network utilities:
  - PING (tool for establishing network connectivity)
  - TFTP (file-transfer protocol client)
- Code characteristics
  - 1 KLOC of non-trivial ANSI-C code
  - Loss of packets modelled by probabilistic choice
  - Linux kernel calls modelled by nondeterministic choice
- Example properties
  - “maximum probability of establishing a write request”
  - “maximum expected amount of data that is sent before timeout”
  - “maximum expected number of echo requests required to establish connectivity”

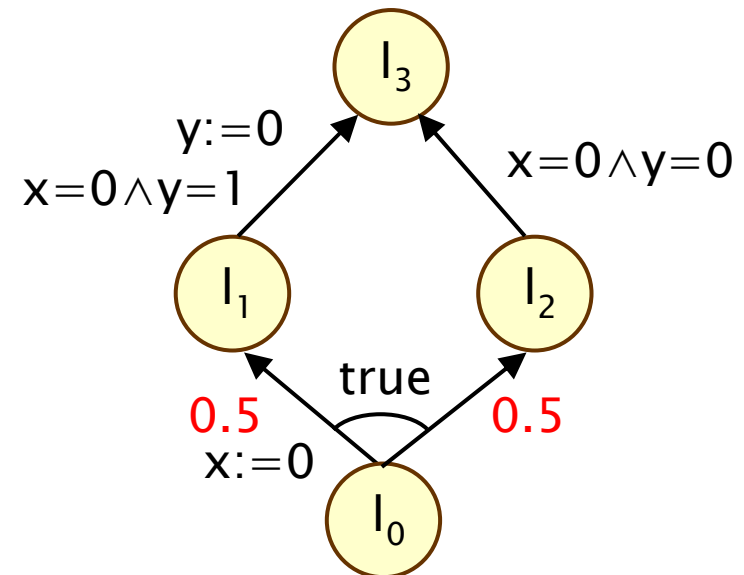
# Overview

## 4. Verification of probabilistic real-time systems



# Probabilistic timed automata

- **Timed automata + discrete probabilistic choice**
  - (or MDPs enriched with real-valued clocks)
  - infinite-state MDP semantics
- **Probability and real-time essential for...**
  - communication protocols
  - security protocols
- **Properties of PTAs:**
  - timed probabilistic reachability
  - e.g. “the minimum probability that a data packet is correctly delivered within T seconds”
  - also PTCTL

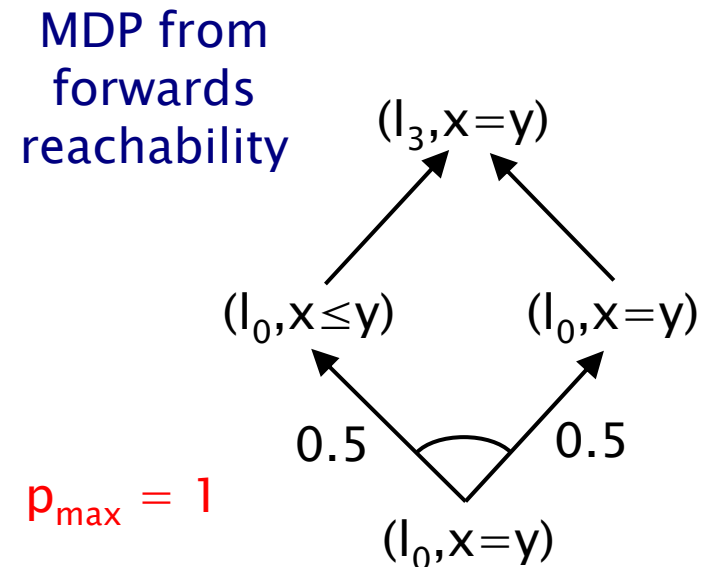
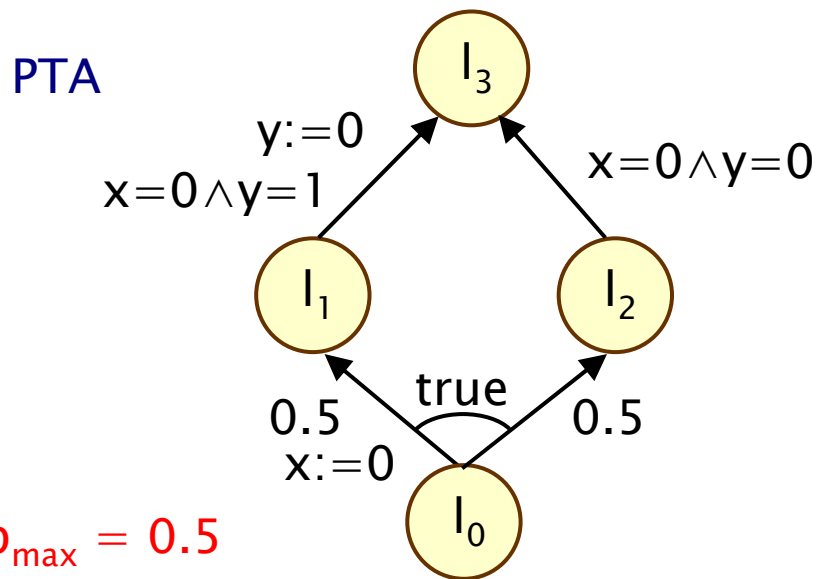


# Probabilistic verification of PTAs

- Region graph construction [TCS'02]
  - exponential blow-up; infeasible in practice
- Forwards reachability [TCS'02]
  - forwards exploration of PTA state space to build finite MDP using symbolic states (zones)
  - only gives upper bound on maximum reachability probabilities
  - MDP is an **abstraction** of the infinite state PTA semantics
- Backwards reachability [I&C'07]
  - backwards traversal; exact results for min/max probabilities
  - operations to implement can be expensive
- Digital clocks (integer semantics) [FMSD'06]
  - language-level translation to model with finite MDP semantics
  - restrictions on applicability; potentially large MDPs

# Forwards reachability

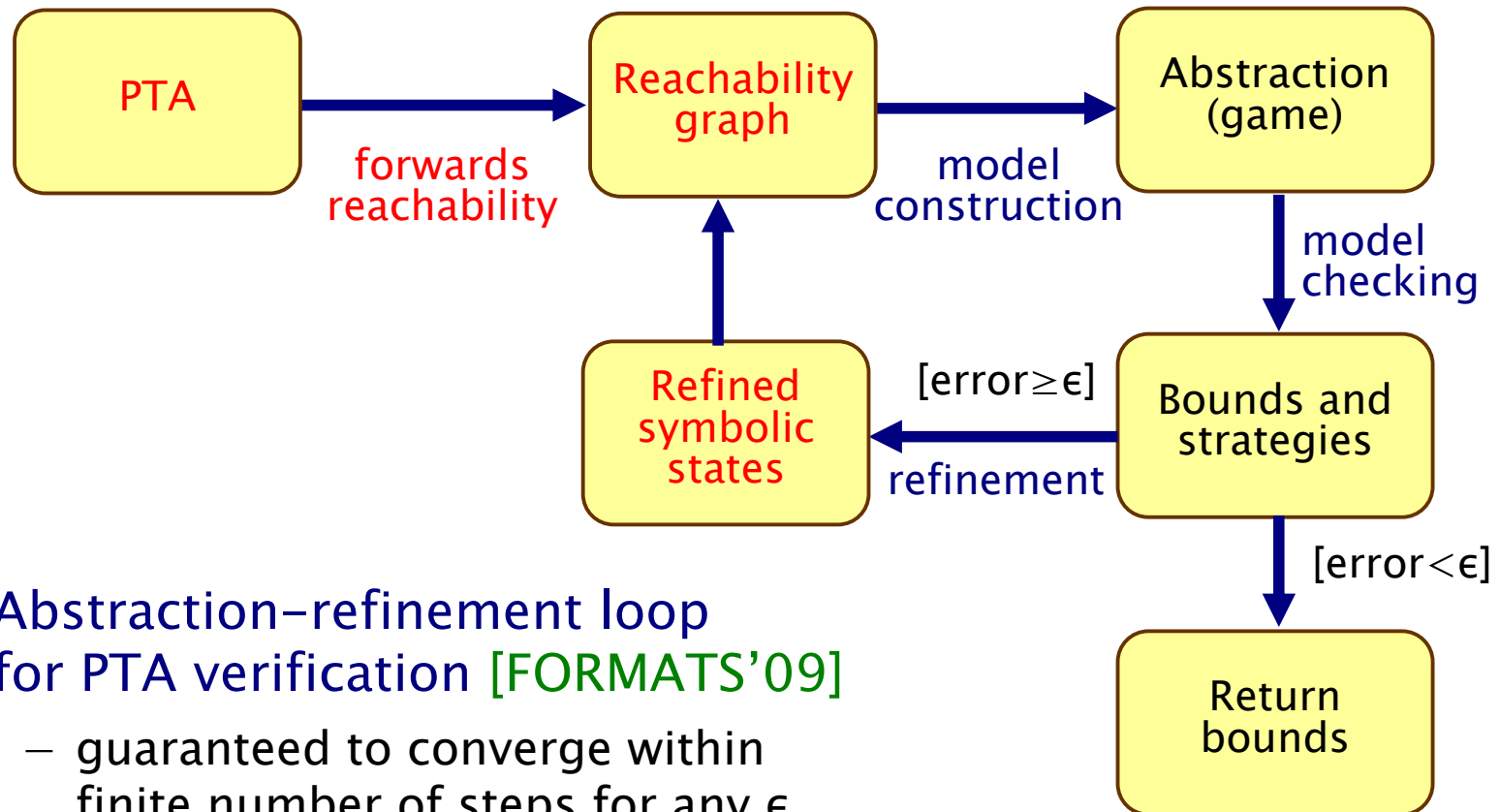
- Why does the MDP constructed through forwards reachability only yield upper bounds?
  - consider maximum probability of reaching  $l_3$  in the PTA below



- We can fix this with an alternative construction that builds a stochastic game abstraction instead [FORMATS'09]



# Abstraction–refinement loop



- Abstraction–refinement loop for PTA verification [FORMATS'09]
  - guaranteed to converge within finite number of steps for any  $\epsilon$
  - (in the worst case, would get region graph, but abstractions are much smaller in practice)

# Experimental results

- Applied to 7 large PTA case studies
  - communication/security protocols: CSMA, FireWire, Zeroconf, non-repudiation, ...
- Comparison of abstraction-refinement approach with other techniques for PTA verification
  - faster than backwards reachability/digital clocks in all cases
  - sometimes by several orders of magnitude
  - handles larger PTAs than the digital clocks approach
  - backwards reachability constructs smaller MDPs, but is still less efficient due to the expensive operations required



# Conclusions

- **Abstraction approach for MDPs**
  - using stochastic two-player games
  - separation of nondeterminism from MDP/abstraction
  - both lower/upper bounds for min/max probabilities/rewards
  - quantitative measure of the utility of abstraction
- **Abstraction refinement for MDPs**
  - quantitative abstraction-refinement loop
  - refinement driven by lower/upper bounds & strategies
- **Quantitative software verification**
  - tool chain using state-of-the-art techniques and tools
- **Real-time probabilistic verification**
  - abstraction refinement outperforms existing techniques



# Quantitative verification – Trends

- Being ‘younger’, generally lags behind conventional verification
  - much smaller models
  - **no** compositional reasoning
  - CEGAR very limited
- Tool usage on the increase, in academic/industrial contexts
  - Real-time verification/synthesis in embedded systems
  - Probabilistic verification in security, reliability, performance
- Shift towards greater automation
  - specification mining, model extraction, synthesis, verification, ...
- **But** many challenges remain!



# Challenges

- **Scalability** remains to be a challenge
  - Develop compositional assume-guarantee methods [TACAS07]
- **Approximate analysis**
  - Broader range of sampling based methods
- **Quantitative software model checking**
  - Heuristics for quantitative abstraction/refinement
- **Adaptive, component-based systems**
  - Online techniques [ICSE09, FASE09]
- **Techniques and tools for continuous (stochastic) dynamics**
  - Embedded systems, controller software (e.g. Simulink), biology (e.g. Bio-PEPA/PRISM), ...
- **Synthesis from specification**
  - Parameter synthesis [Katoen et al]
  - Controller synthesis [Baier et al]