Verification of probabilistic software

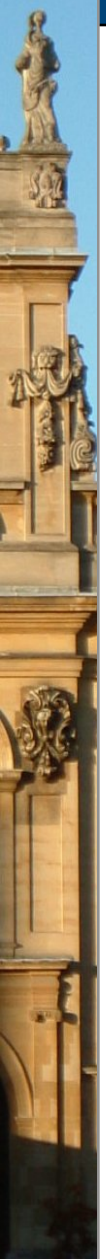
Dave Parker

Oxford University Computing Laboratory

Joint work with:

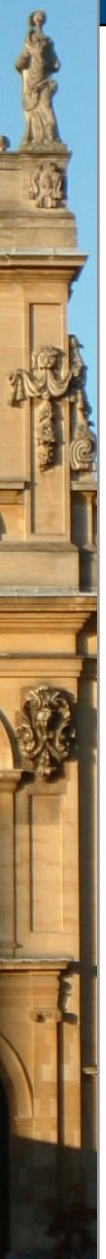
Mark Kattenbelt, Marta Kwiatkowska, Gethin Norman

Queen Mary University, April 2009



Motivation

- Why probability?
  - many systems we want to verify are inherently probabilistic
- Randomisation, e.g. in distributed coordination algorithms
  - random delays/back-off in CSMA/CD, IEEE 802.11, Bluetooth
  - random IP address selection in Zeroconf/Bonjour
  - randomised algorithms for anonymity, contract signing, …
- 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, fairness, …



Overview

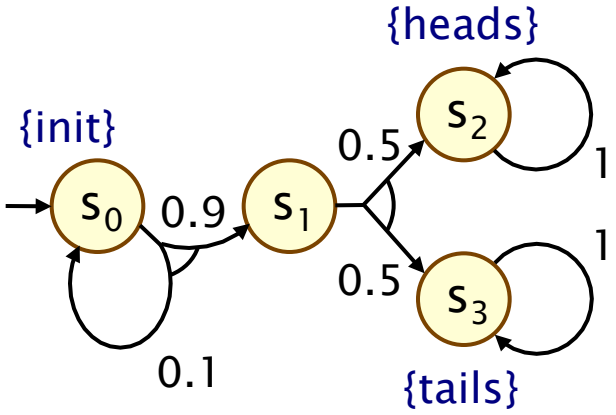
- Probabilistic model checking
  - Markov decision processes (MDPs)
  - probabilistic reachability, temporal logics
  - tool support: PRISM
- Abstraction for MDPs
  - two-player stochastic games
  - abstraction-refinement loop
- Verification of probabilistic software
  - probabilistic verification at the level of source code (e.g. C)
  - game-based abstraction, predicate abstraction, SAT
  - tool chain: (extensions of) goto-cc, SATABS, PRISM

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)
- 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

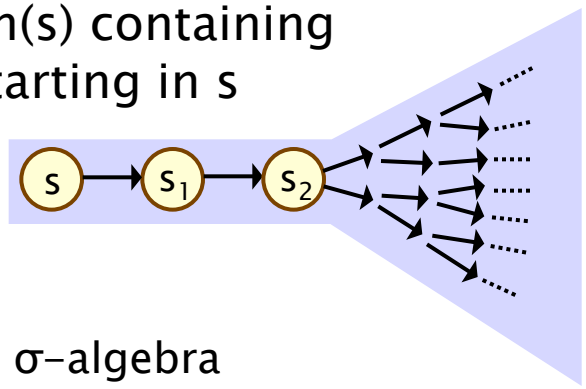


- Formally, a DTMC is a tuple
  - $(S, s_{init}, \mathbf{P}, L)$
- where:
  - S is a set of states
  - $s_{init} \in S$ is the initial state
  - $\mathbf{P} : S \times S \rightarrow [0,1]$ is the transition probability matrix
  - $L : S \rightarrow 2^{AP}$ is a labelling function
  - AP is a set of atomic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

Paths and probabilities

- Paths through a DTMC:
  - infinite sequences of states $s_0 s_1 s_2 s_3 \ldots$ such that $\mathbf{P}(s_i, s_{i+1}) > 0$
  - represent executions of the system being modelled
  - for quantitative reasoning, need probability space over paths

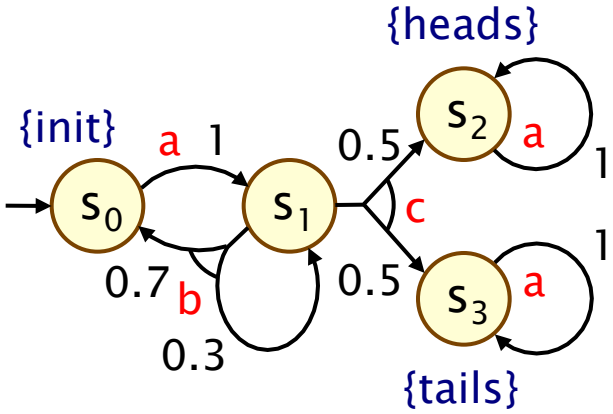
- Probability space $(\text{Path}(s), \Sigma_{\text{Path}(s)}, \Pr_s)$ [KSK66]
  - sample space: Path(s) = all infinite paths starting in state s
  - event set: $\Sigma_{\text{Path}(s)}$ = least σ-algebra on Path(s) containing cylinder set Cyl(ω) for all finite paths ω starting in s
    - $\text{Cyl}(\omega) = \{\omega' \in \text{Path}(s) \mid \omega \text{ is prefix of } \omega'\}$
  - probability measure: $\Pr_s : \Sigma_{\text{Path}(s)} \rightarrow [0,1]$
    - $\Pr_s(\text{Cyl}(s, s_1, \ldots, s_n)) = \mathbf{P}(s,s_1) \cdot \ldots \cdot \mathbf{P}(s_{n-1}, s_n)$
    - extends uniquely to all sets of paths in the σ-algebra



- All omega regular properties are measurable

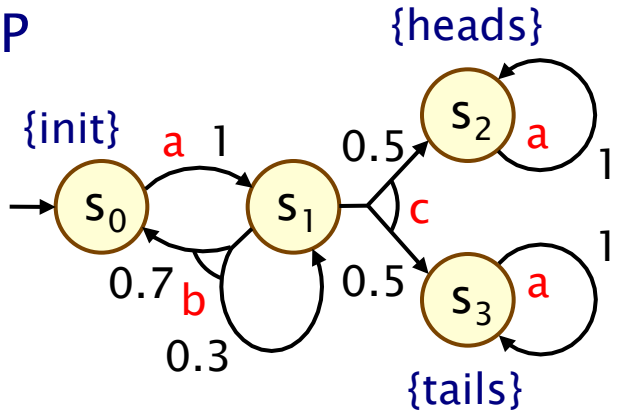
Markov decision processes (MDPs)

- 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
  - $(S, s_{init}, \mathbf{Steps}, L)$
- where:
  - S is a set of states
  - $s_{init} \in S$ is the initial state
  - $\mathbf{Steps} : S \rightarrow 2^{Act \times Dist(S)}$ is the transition probability function
  - $L : S \rightarrow 2^{AP}$ 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

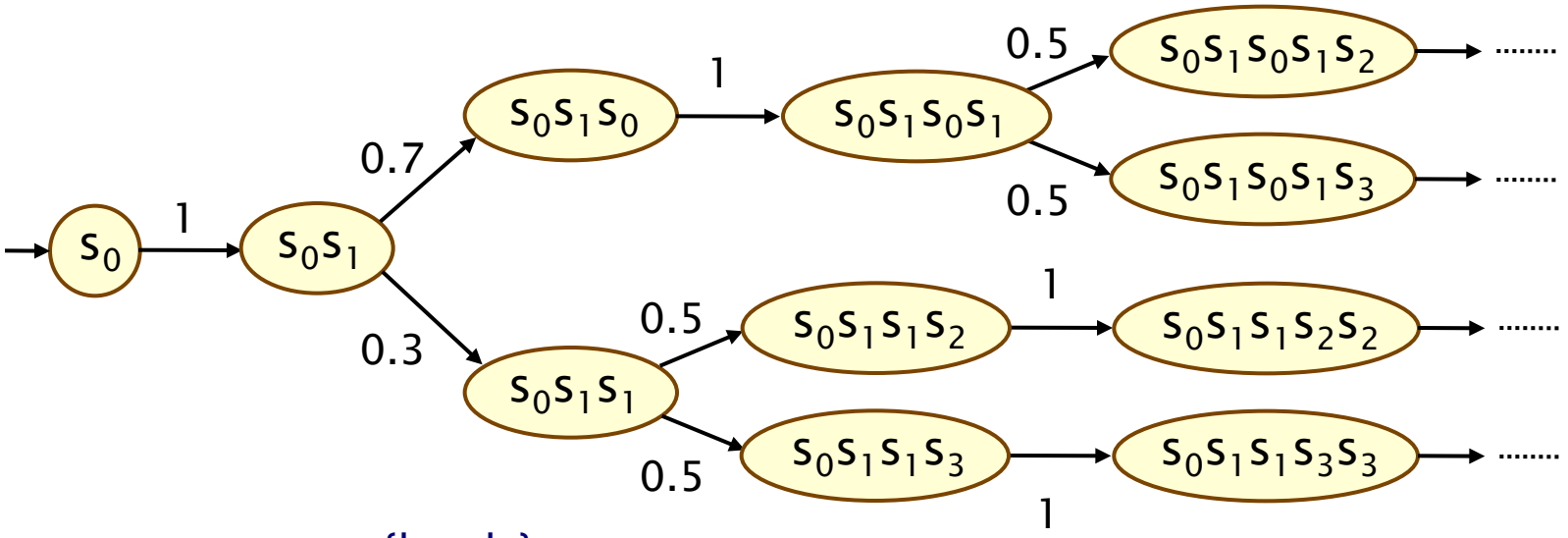


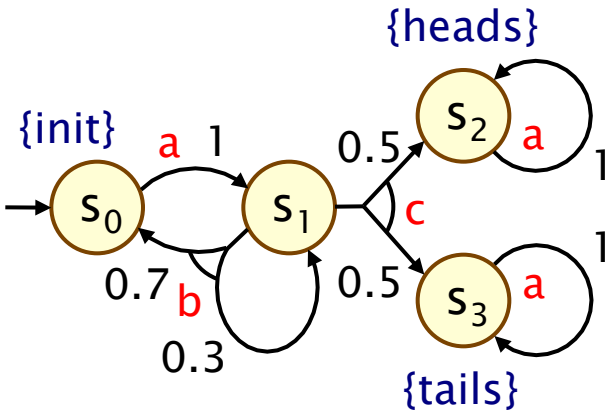
- An adversary (aka. "scheduler" or "policy") of an MDP
  - is a resolution of nondeterminism only
  - is (formally) a mapping from finite paths to distributions
  - results in a fully probabilistic model
  - i.e. an (infinite-state) Markov chain over finite paths
  - on which we can define a probability space over infinite paths

- Adversary A is simple iff: $A(s_1...s_n) = A(s_n)$ for all $s_1...s_n$
  - in this case, resulting model reduces to finite Markov chain

Example adversary

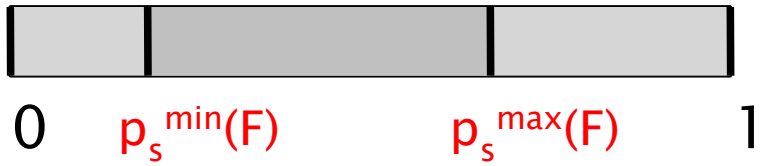
- Fragment of DTMC for adversary which picks b then c in s_1





Probabilistic reachability for MDPs

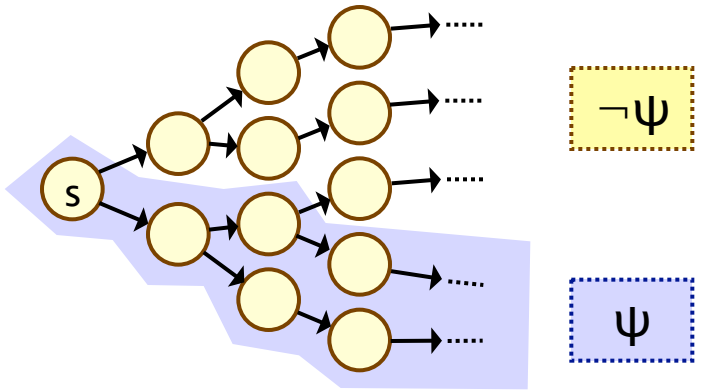
- An adversary A induces, for each state s in the MDP:
  - a set of infinite paths $Path^A(s)$
  - a probability space Pr^A_s over $Path^A(s)$
- Probabilistic reachability (for a set of goal states $F \subseteq S$)
  - probability of reaching F from state s under adversary A
  - $p_s^A(F) = Pr^A_s \{ s_0 s_1 s_2 s_3 \ldots \in Path^A(s) \mid s_i \in F \text{ for some } i \}$
- 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

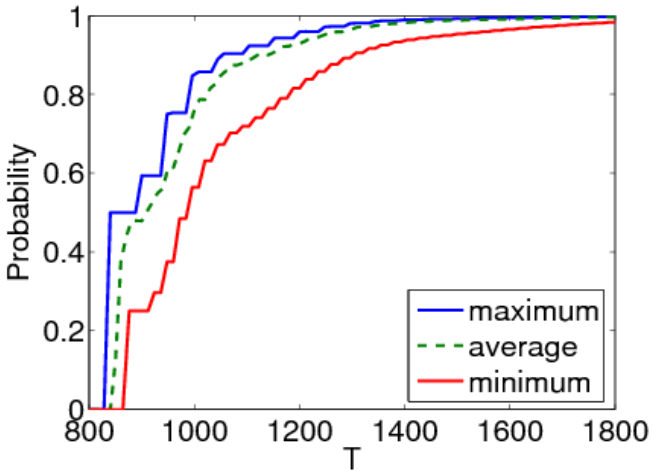
- Also: Bounded reachability properties
  - e.g. “min. probability of algorithm termination within T steps”
- Also: Cost- and reward-based properties
  - augment states/transitions of MDP with real-valued costs
  - define properties as random variables over $\text{Path}^A(s)$
  - e.g. “max. expected 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. $P_{<0.01}$ [◇ 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

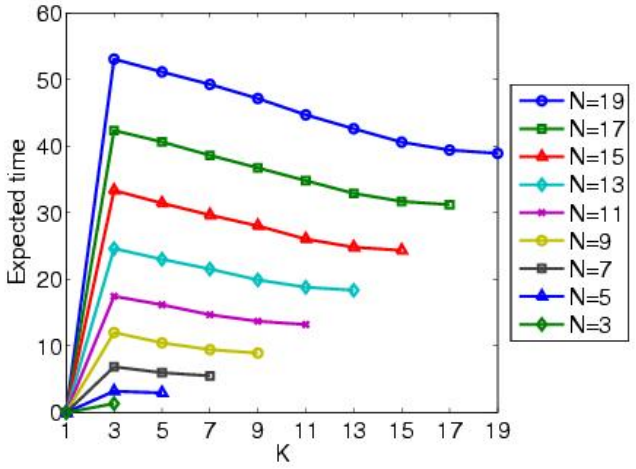
- Efficient model checking algorithms exist:
  - main component: computation of reachability probabilities
    - linear optimisation problem (polynomial complexity)
    - or value iteration (dynamic programming) – simple iterative numerical method; more efficient in practice
    - also: graph-based model 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:
  - $P_{min=?}$ [ψ] and $P_{max=?}$ [ψ]
  - i.e. "what is the minimum/maximum probability (over all adversaries) that path formula ψ is true?"
  - useful to spot patterns/trends



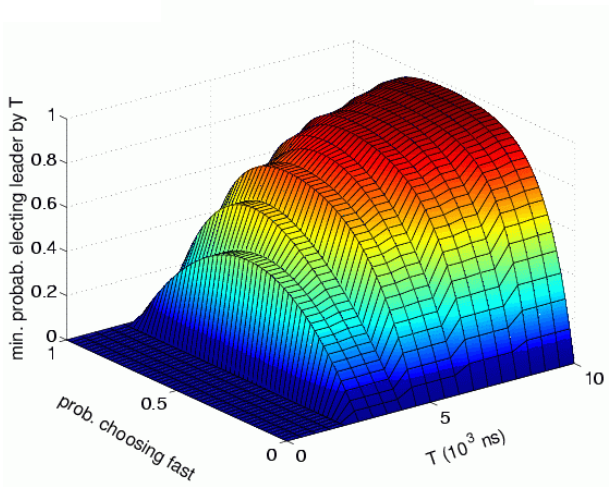
CSMA/CD protocol:

Min/max/average probability that a message is sent successfully by time T



Self-stabilisation:

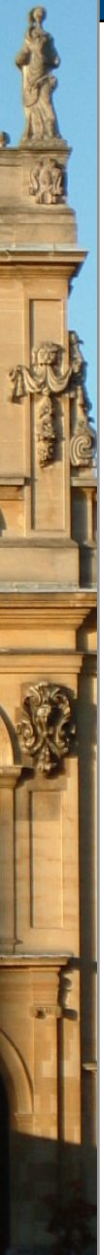
Worst-case expected number of steps to stabilise for initial configurations with K tokens amongst N processes



Firewire protocol:

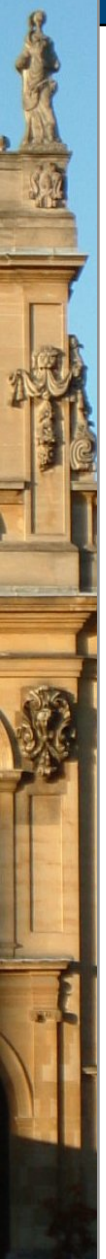
Optimum probability of leader election by time T for various coin biases

PRISM



- PRISM: Probabilistic model checker
  - developed at Birmingham, Oxford since approx. 2001
- 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

- But major challenges remain, e.g.
  - state-space explosion
  - automating model extraction



Overview

- Probabilistic model checking
  - Markov decision processes (MDPs)
  - probabilistic reachability, temporal logics
  - tool support: PRISM
- Abstraction for MDPs
  - two-player stochastic games
  - abstraction-refinement loop
- Verification of probabilistic software
  - probabilistic verification at the level of source code (e.g. C)
  - game-based abstraction, predicate abstraction, SAT
  - tool chain: (extensions of) goto-cc, SATABS, PRISM

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

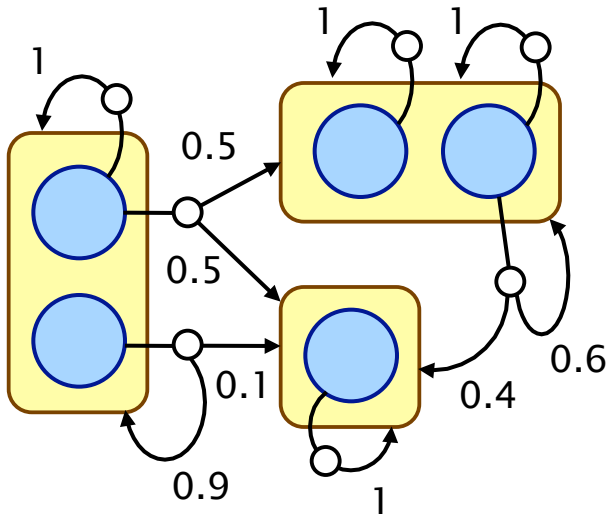
- Abstraction increases degree of nondeterminism
  - i.e. minimum probabilities are lower and maximums higher

  - 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

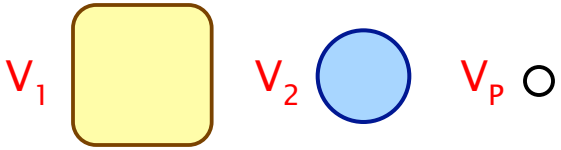
  - gives quantitative measure of utility of abstraction
  - basis of quantitative abstraction-refinement framework

Stochastic two-player games

- Simple stochastic games [Shapley,Condon]
- Game G = ((V,E), v_{init}, (V_1,V_2,V_P), δ)
  - (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'
  - δ : V_P → Dist(V) is a probabilistic transition function



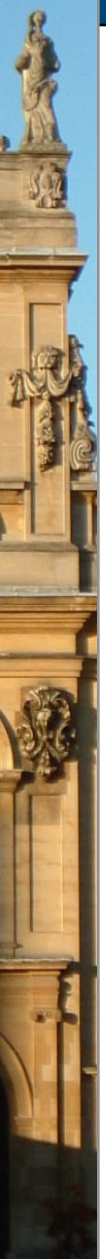
- Execution of G: successor vertex chosen:
  - by player 1/2 for V_1/V_2 vertices
  - at random (δ) for V_P vertices



- MDPs can be thought of as stochastic two-player games with no V_1 vertices and strict alternation between V_2/V_P

Properties of stochastic games

- Resolution of nondeterminism in a stochastic game
  - is done by a pair of strategies for players 1 and 2: (σ_1,σ_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 (σ_1,σ_2)
- Optimal probabilities for player 1 and player 2
  - $\sup_{\sigma 1} \inf_{\sigma 2} p_v^{\sigma 1,\sigma 2}(F)$ and $\sup_{\sigma 2} \inf_{\sigma 1} p_v^{\sigma 1,\sigma 2}(F)$
  - computable via simple iterative methods, similar to MDPs

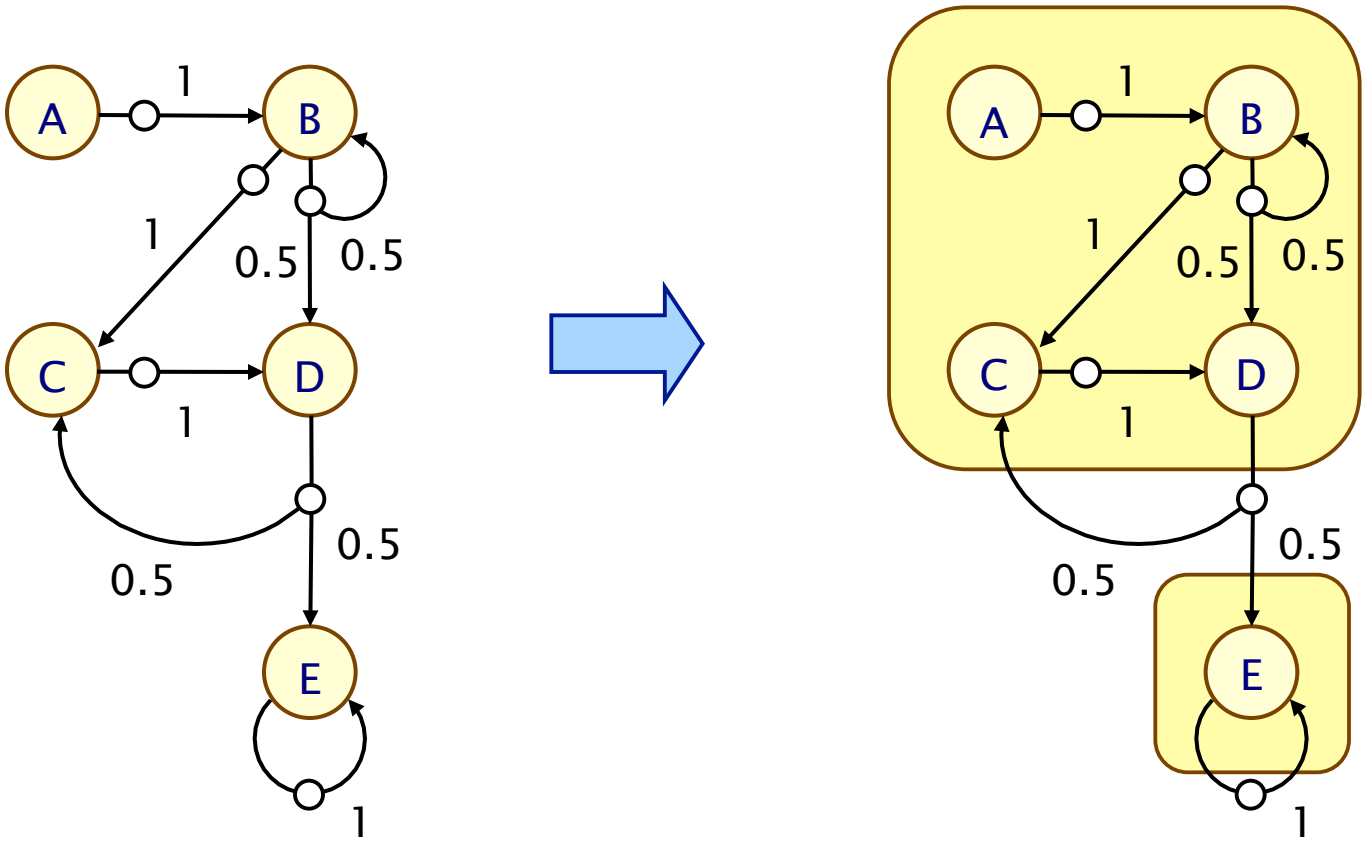


Games as abstractions of MDPs

- 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

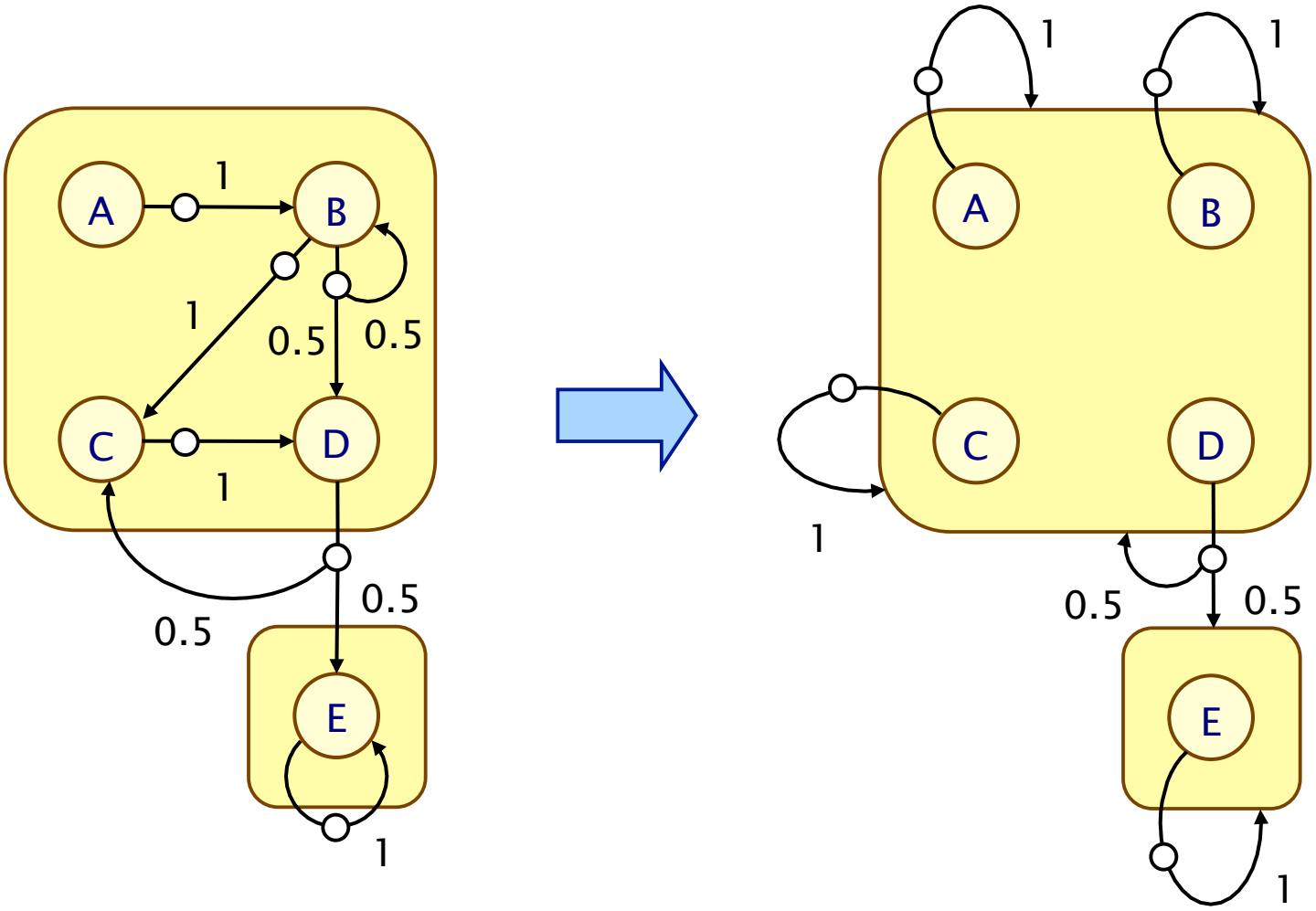
MDP → Game

- Player 1 vertices are partition elements (abstract states)



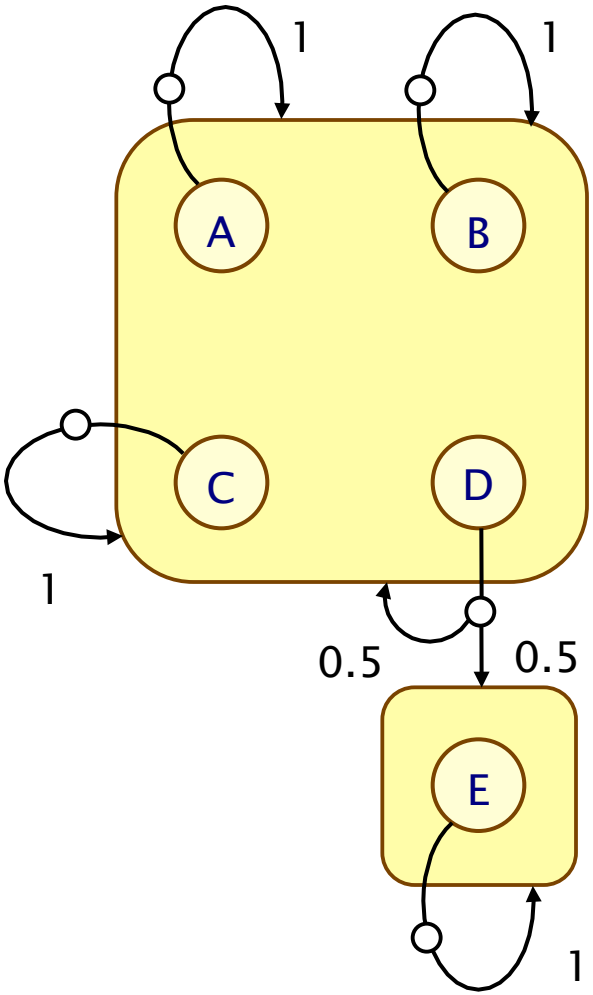
MDP → Game

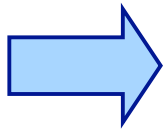
- (Sets of) distributions are lifted to the abstract state space

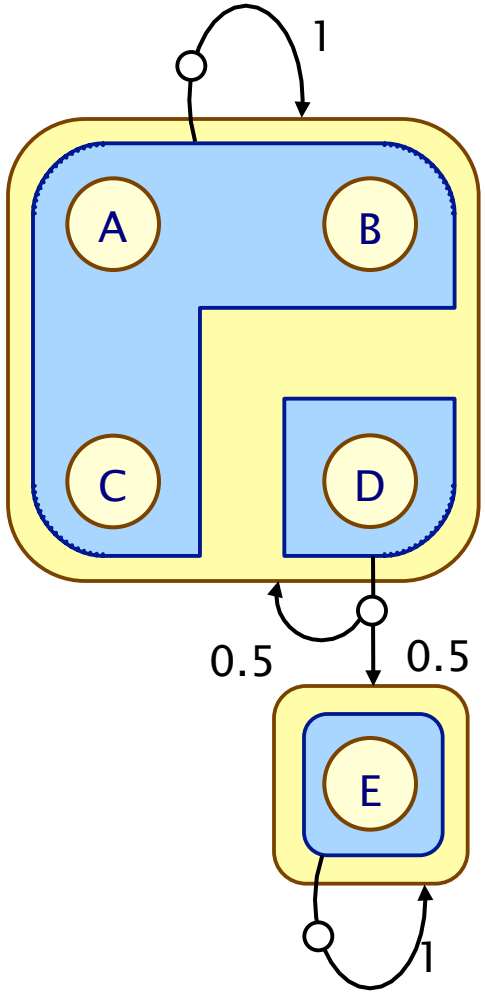


MDP → Game

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

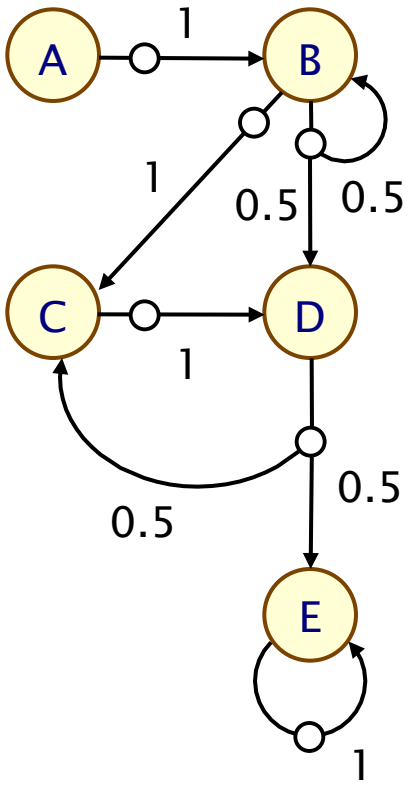


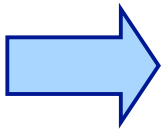


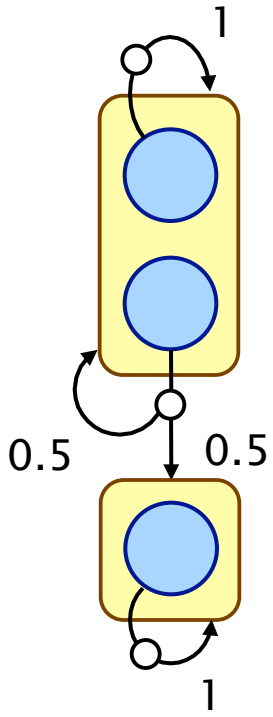


MDP → Game

- 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:

$$\inf_{\sigma 1,\sigma 2} p_v^{\sigma 1,\sigma 2}(F) \leq p_s^{min}(F) \leq \sup_{\sigma 1} \inf_{\sigma 2} p_v^{\sigma 1,\sigma 2}(F)$$

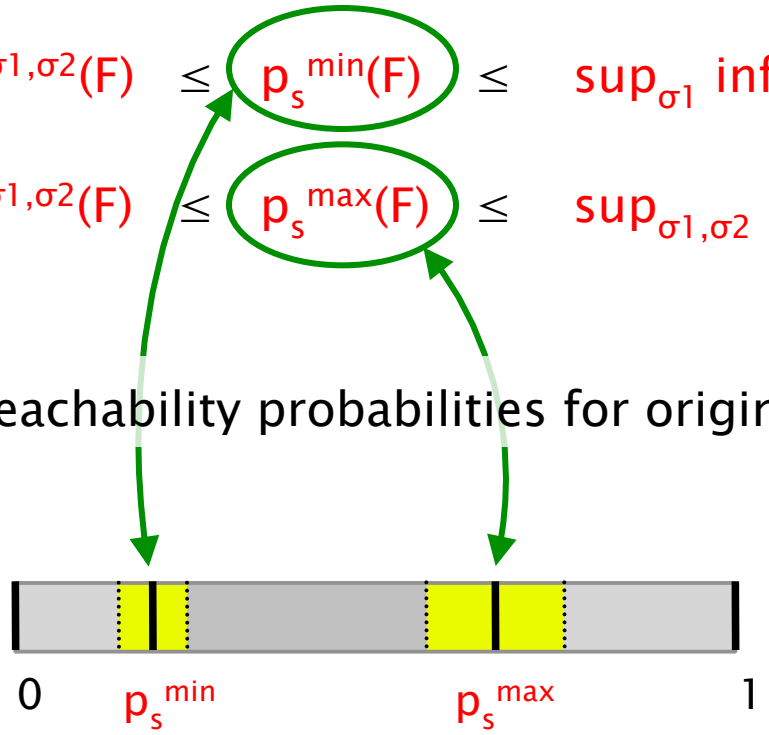
$$\sup_{\sigma 2} \inf_{\sigma 1} p_v^{\sigma 1,\sigma 2}(F) \leq p_s^{max}(F) \leq \sup_{\sigma 1,\sigma 2} p_v^{\sigma 1,\sigma 2}(F)$$

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:

$$\inf_{\sigma 1,\sigma 2} p_v^{\sigma 1,\sigma 2}(F) \leq p_s^{min}(F) \leq \sup_{\sigma 1} \inf_{\sigma 2} p_v^{\sigma 1,\sigma 2}(F)$$

$$\sup_{\sigma 2} \inf_{\sigma 1} p_v^{\sigma 1,\sigma 2}(F) \leq p_s^{max}(F) \leq \sup_{\sigma 1,\sigma 2} p_v^{\sigma 1,\sigma 2}(F)$$



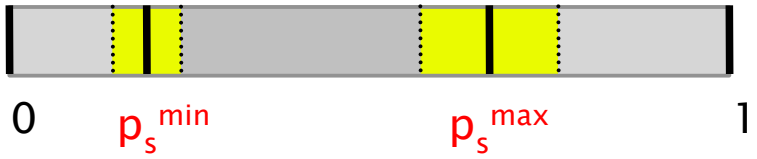
Analysis of the abstraction

- 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:

$$\inf_{\sigma 1,\sigma 2} p_v^{\sigma 1,\sigma 2}(F) \leq p_s^{min}(F) \leq \sup_{\sigma 1} \inf_{\sigma 2} p_v^{\sigma 1,\sigma 2}(F)$$

$$\sup_{\sigma 2} \inf_{\sigma 1} p_v^{\sigma 1,\sigma 2}(F) \leq p_s^{max}(F) \leq \sup_{\sigma 1,\sigma 2} p_v^{\sigma 1,\sigma 2}(F)$$

optimal probabilities for player 1, player 2 in game



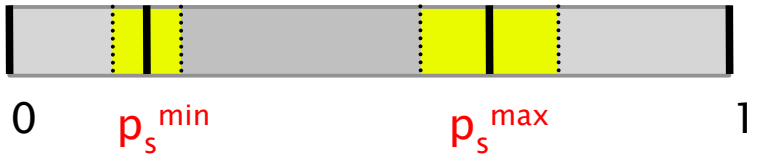
Analysis of the abstraction

- 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:

$$\inf_{\sigma 1,\sigma 2} p_v^{\sigma 1,\sigma 2}(F) \leq p_s^{min}(F) \leq \sup_{\sigma 1} \inf_{\sigma 2} p_v^{\sigma 1,\sigma 2}(F)$$

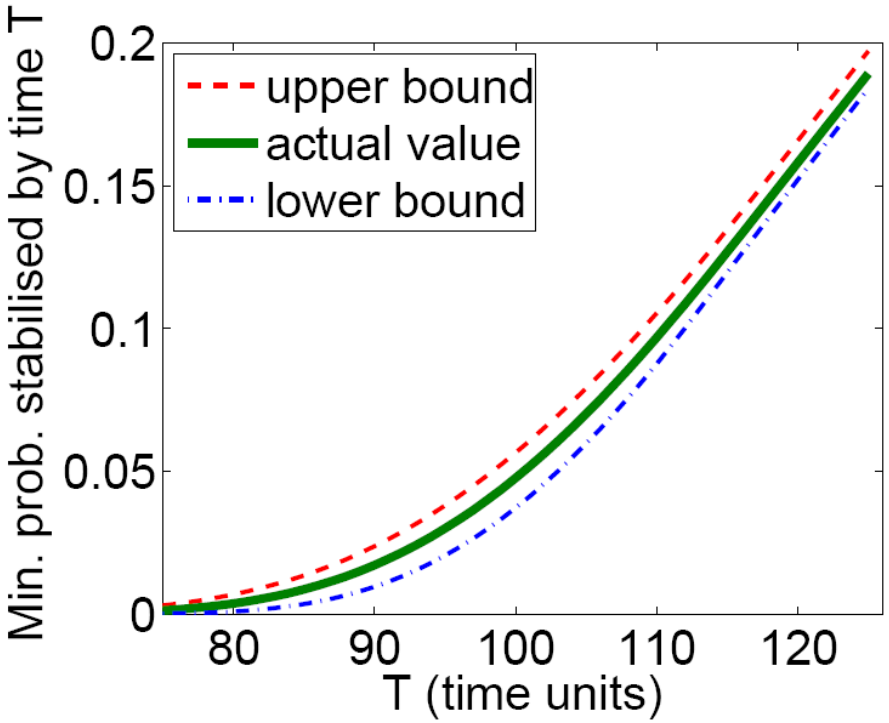
$$\sup_{\sigma 2} \inf_{\sigma 1} p_v^{\sigma 1,\sigma 2}(F) \leq p_s^{max}(F) \leq \sup_{\sigma 1,\sigma 2} p_v^{\sigma 1,\sigma 2}(F)$$

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



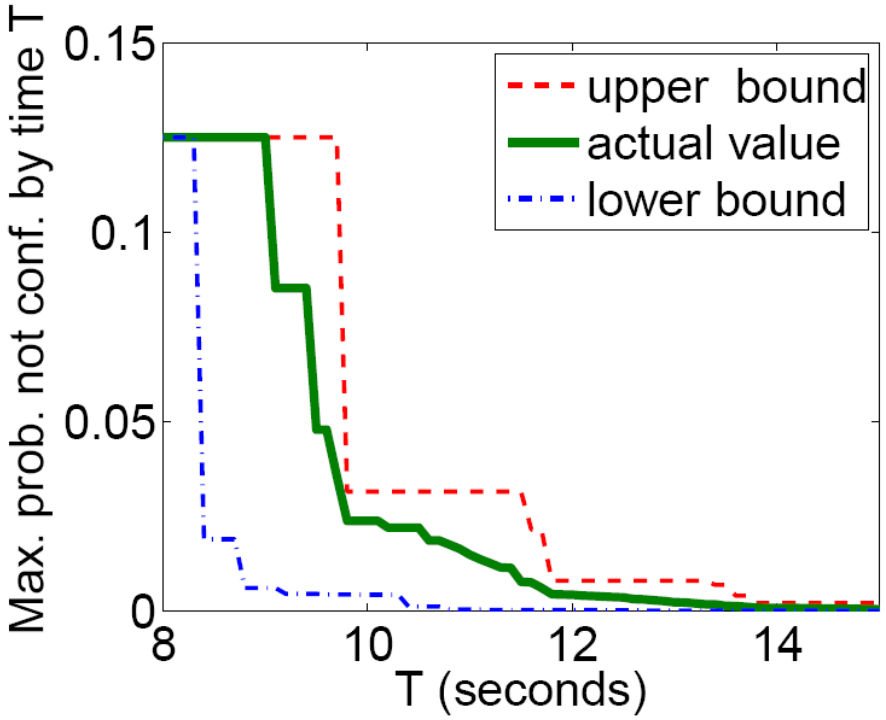
Abstraction: Results

- 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



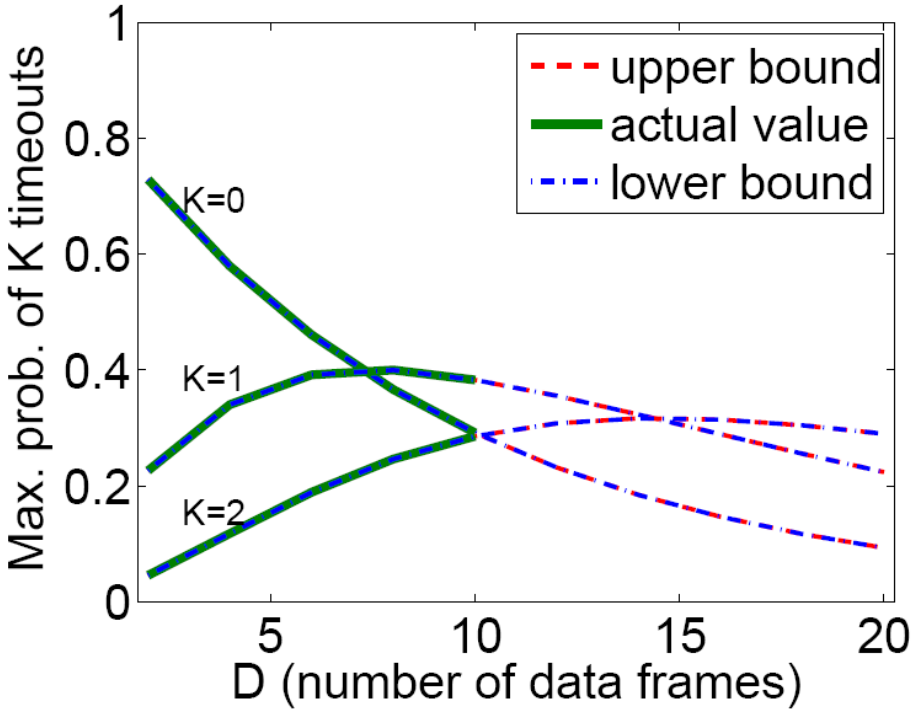
Abstraction: Results

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



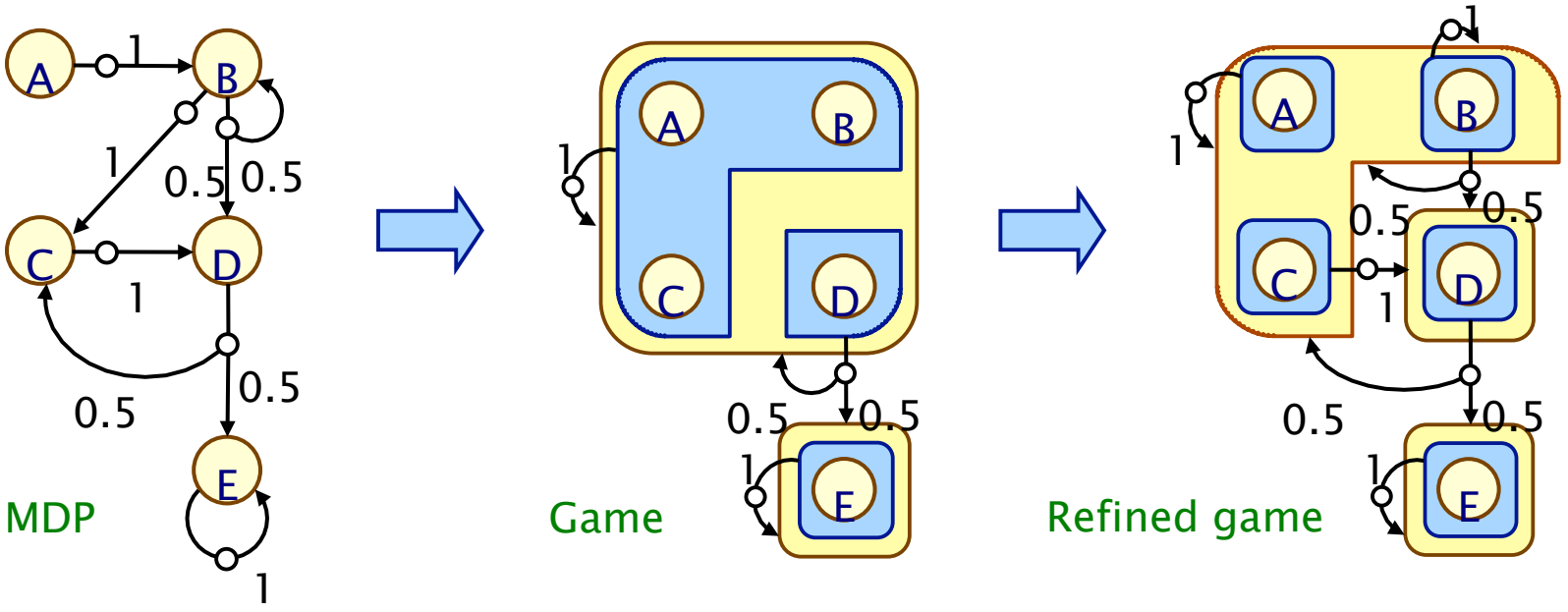
Abstraction: Results

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



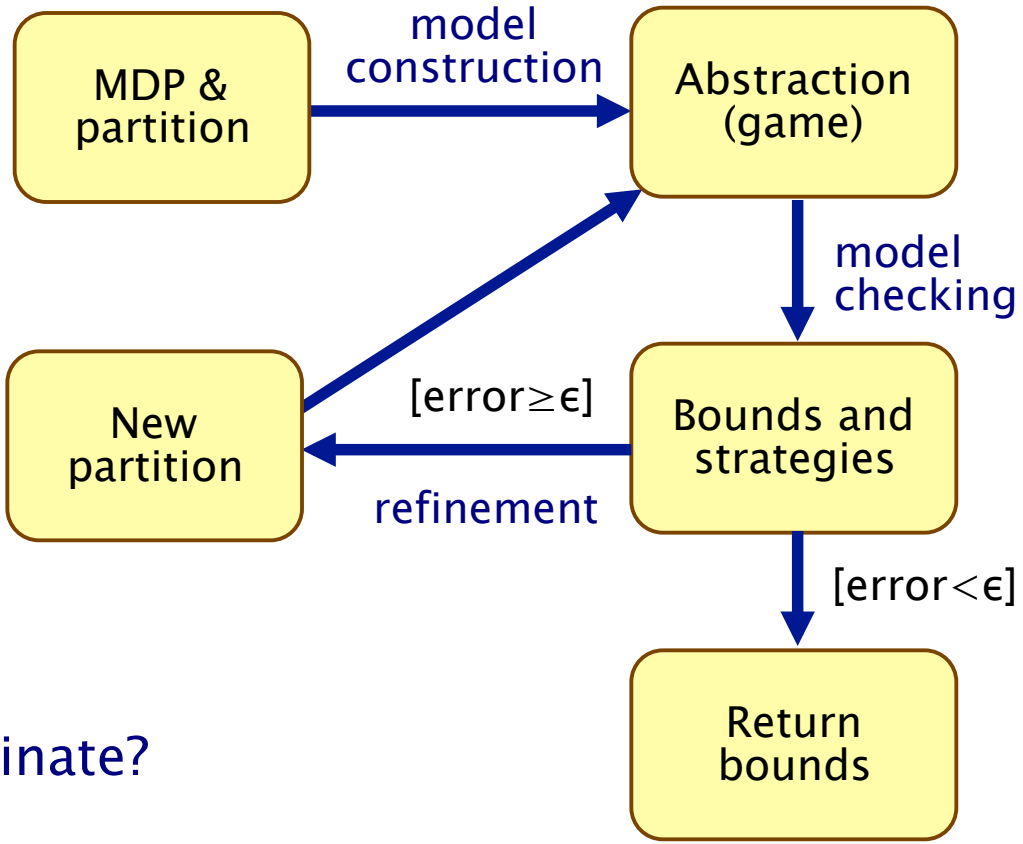
Abstraction-refinement

- 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

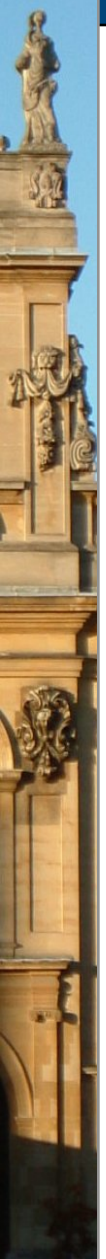


Abstraction-refinement loop

- Quantitative abstraction-refinement loop for MDPs



- Does the loop terminate?



Overview

- Probabilistic model checking
  - Markov decision processes (MDPs)
  - probabilistic reachability, temporal logics
  - tool support: PRISM
- Abstraction for MDPs
  - two-player stochastic games
  - abstraction-refinement loop
- Verification of probabilistic software
  - probabilistic verification at the level of source code (e.g. C)
  - game-based abstraction, predicate abstraction, SAT
  - tool chain: (extensions of) goto-cc, SATABS, PRISM

Probabilistic software

- Consider sequential ANSI C programs
  - support functions, pointers, arrays, but not dynamic memory allocation, unbounded recursion, floating point op.s
- 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 --;
      }
}
```

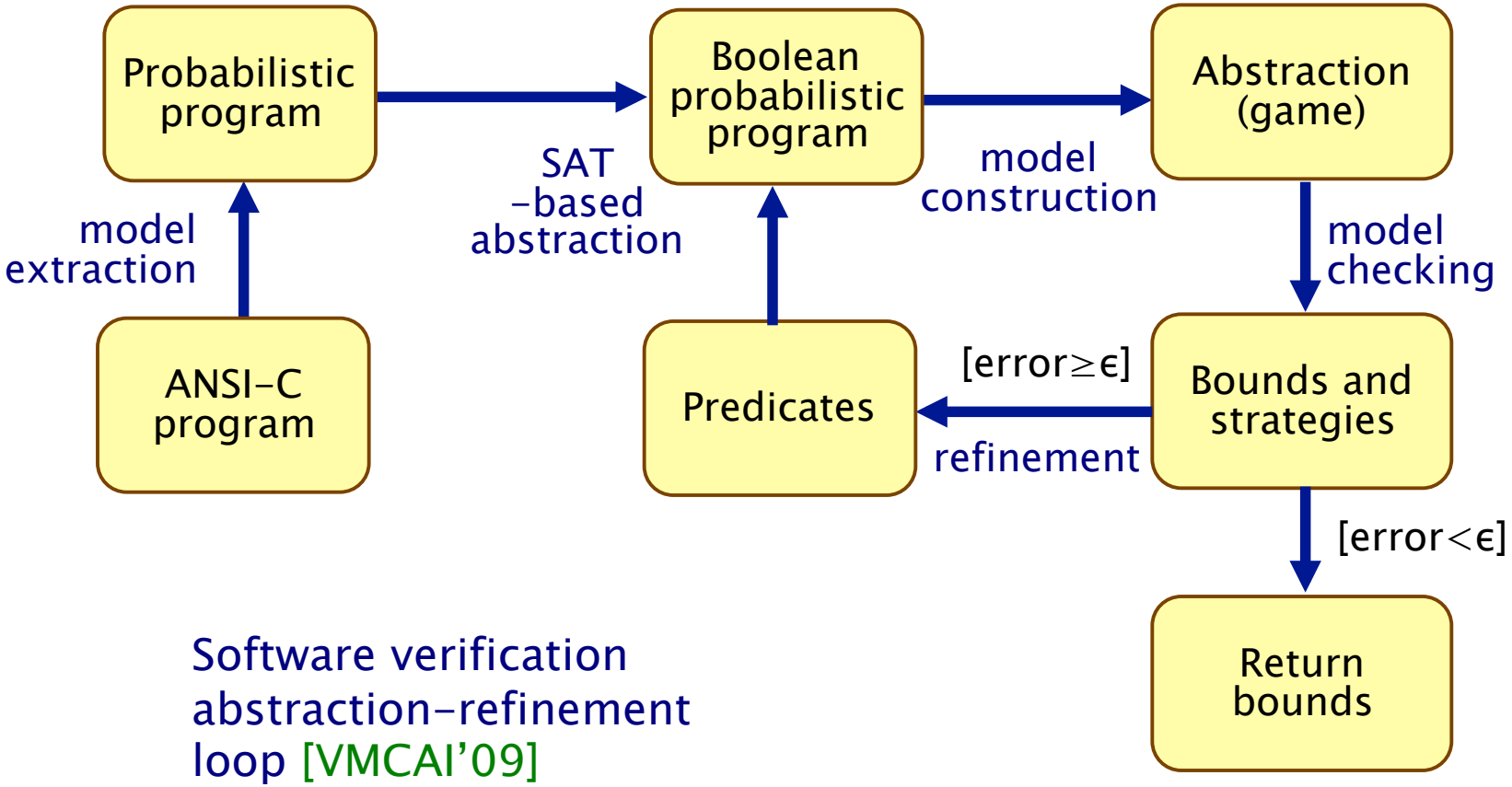
Φ: “what is the minimum/maximum probability of the program terminating with fail being true?”

Example – simplified

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

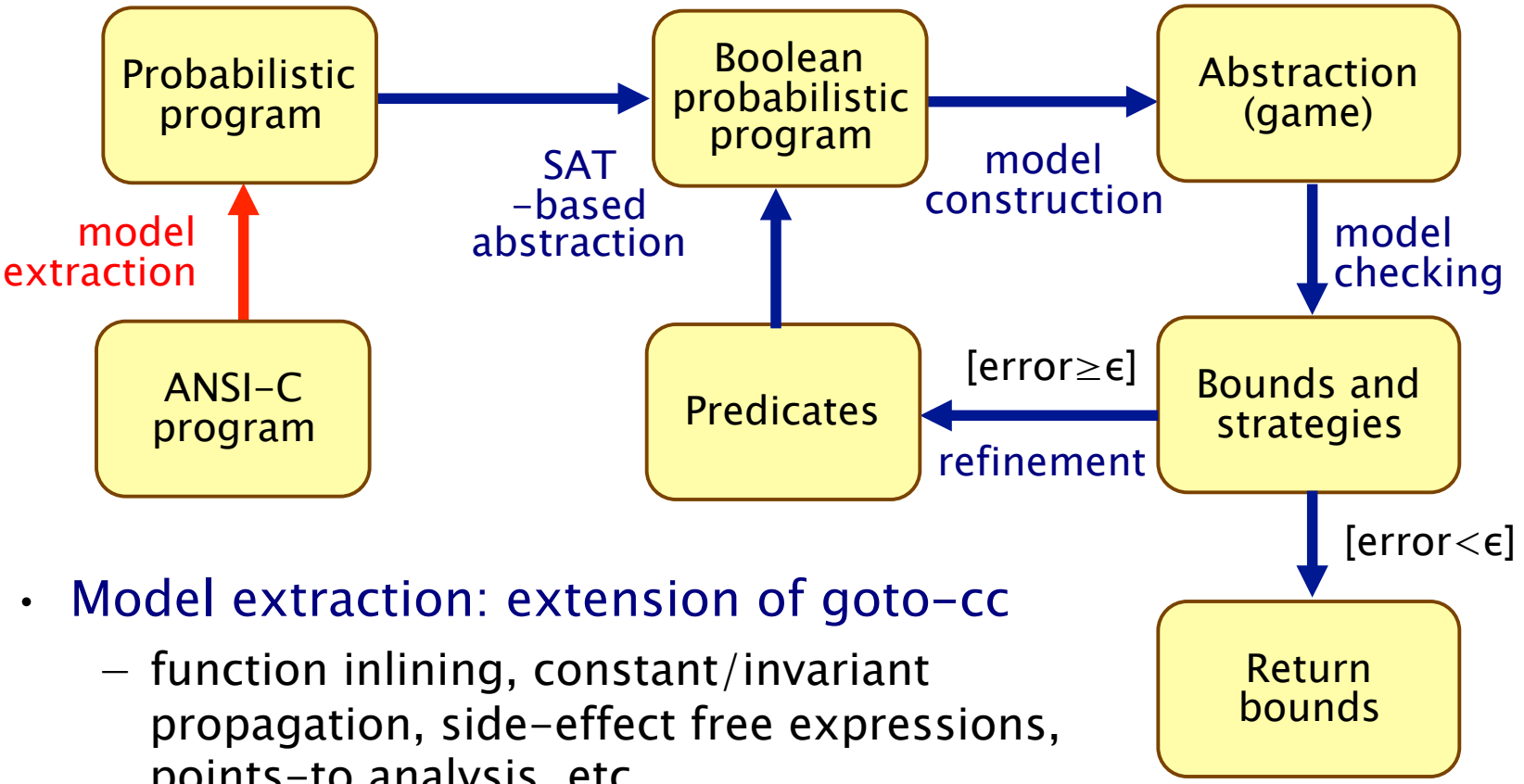
Φ: “what is the minimum/maximum probability of the program terminating with fail being true?”

Abstraction-refinement loop



Software verification abstraction-refinement loop [VMCAI'09]

Abstraction-refinement loop

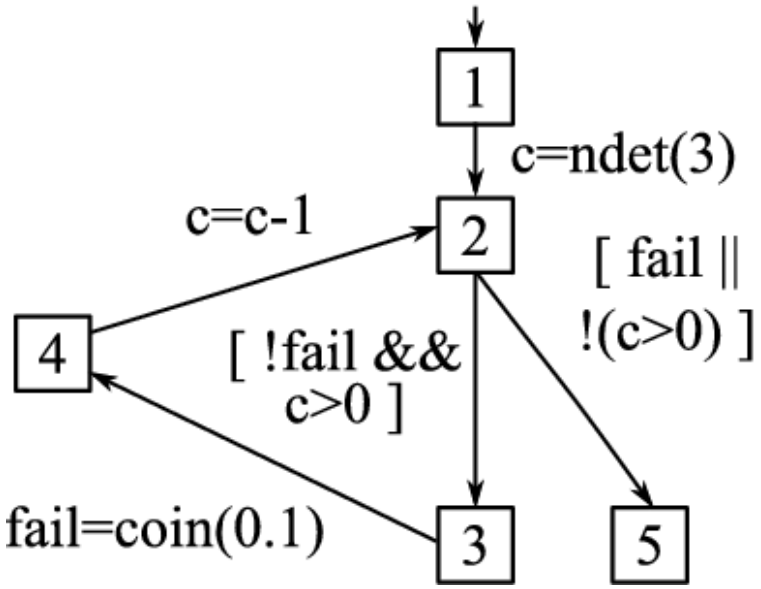


- Model extraction: extension of goto-cc
  - function inlining, constant/invariant propagation, side-effect free expressions, points-to analysis, etc.
- Probabilistic program
  - probabilistic control flow graph
  - Markov decision process (MDP) semantics

Back to example

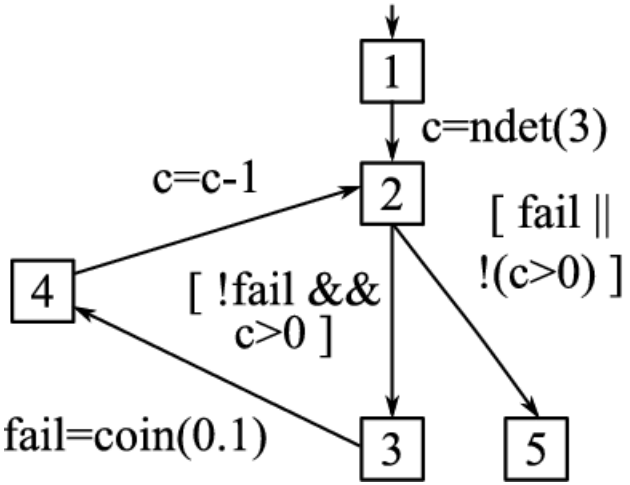
```
bool fail = false;
int c = 0;
int main ()
{
      // nondeterministic
      c = ndet (3);
      while (! fail && c > 0)
      {
            // probabilistic
            fail = coin (0.1);
            c --;
      }
}
```

Probabilistic program



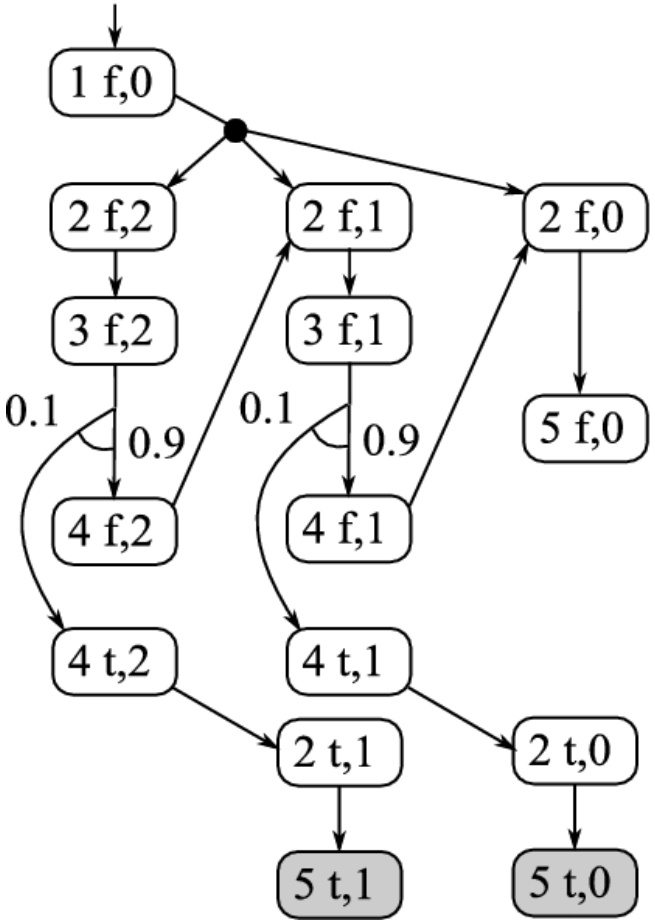
Probabilistic program as MDP

Probabilistic program

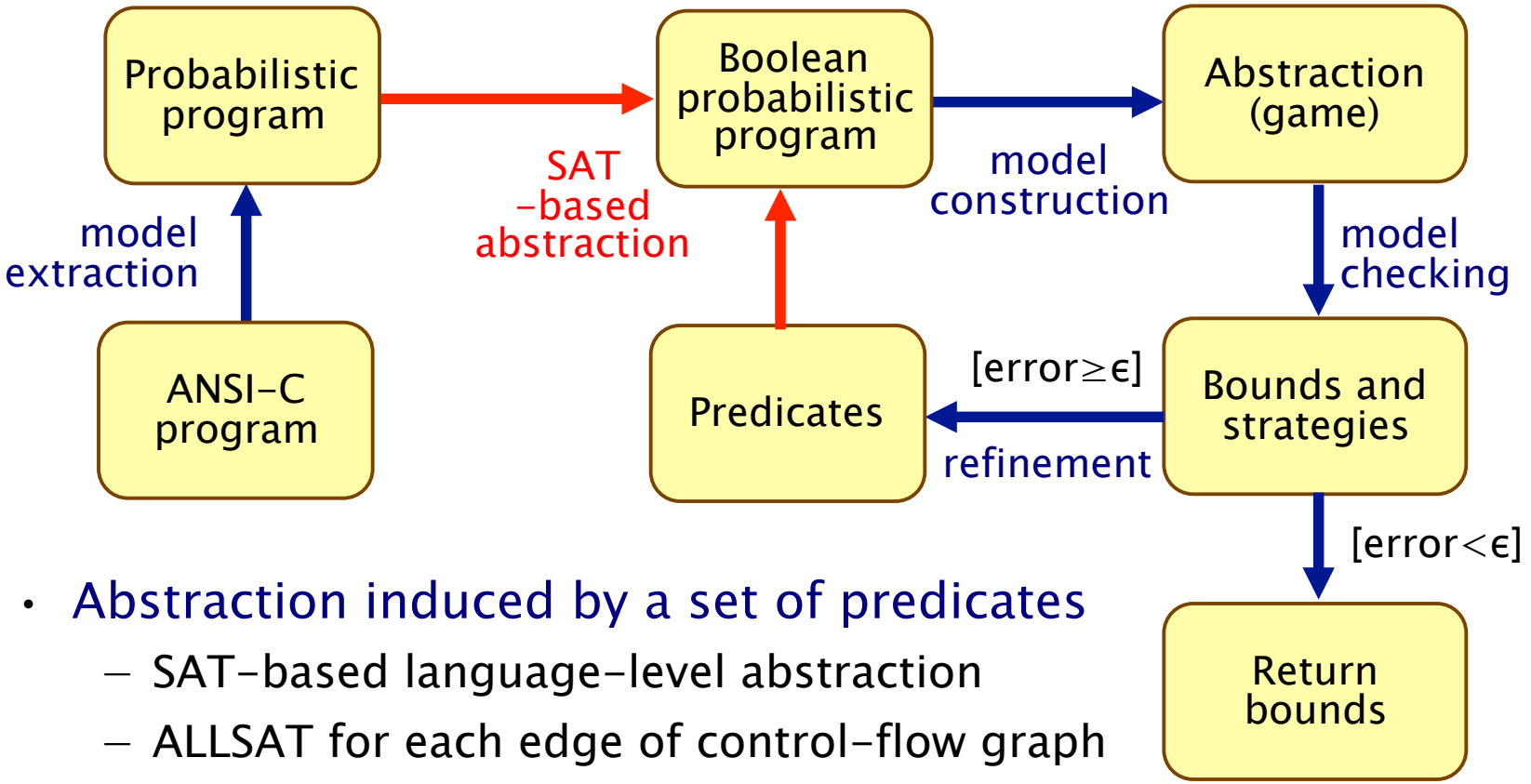


minimum/maximum probability of the program terminating with fail being true is 0 and 0.19, respectively

MDP semantics



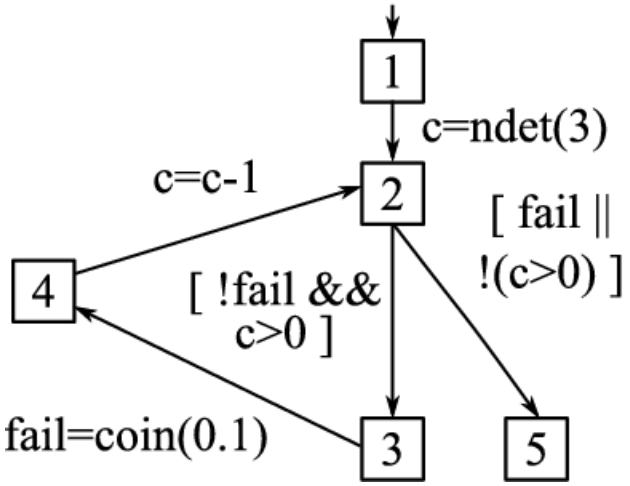
Abstraction-refinement loop



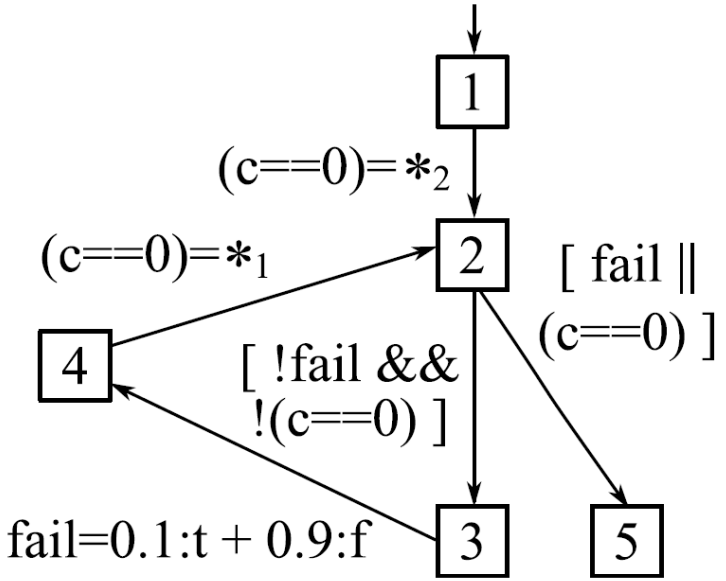
- Abstraction induced by a set of predicates
  - SAT-based language-level abstraction
  - ALLSAT for each edge of control-flow graph
  - implemented in extension of SATABS
- Boolean probabilistic program
  - (predicate) abstraction of probabilistic program
  - stochastic two player game semantics

Back to example

Probabilistic program

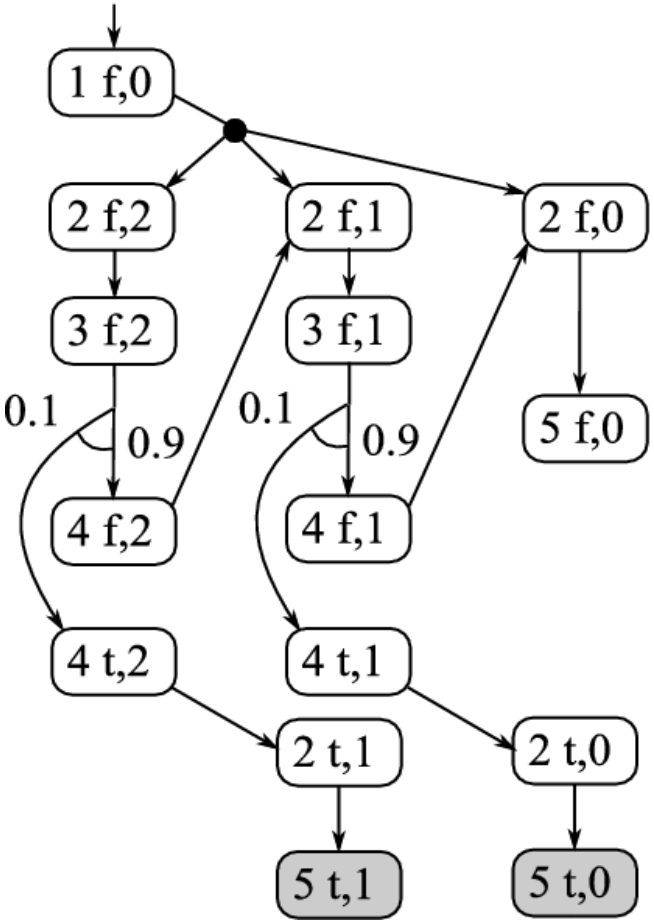


Boolean probabilistic program

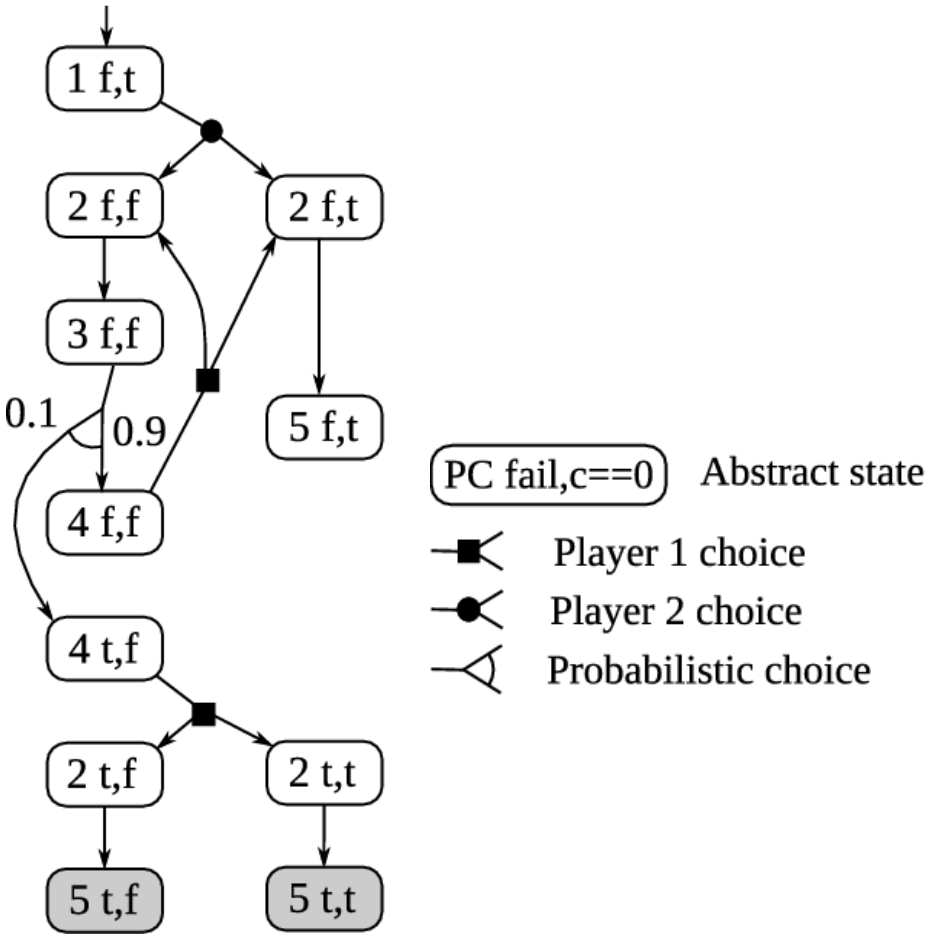


Back to example

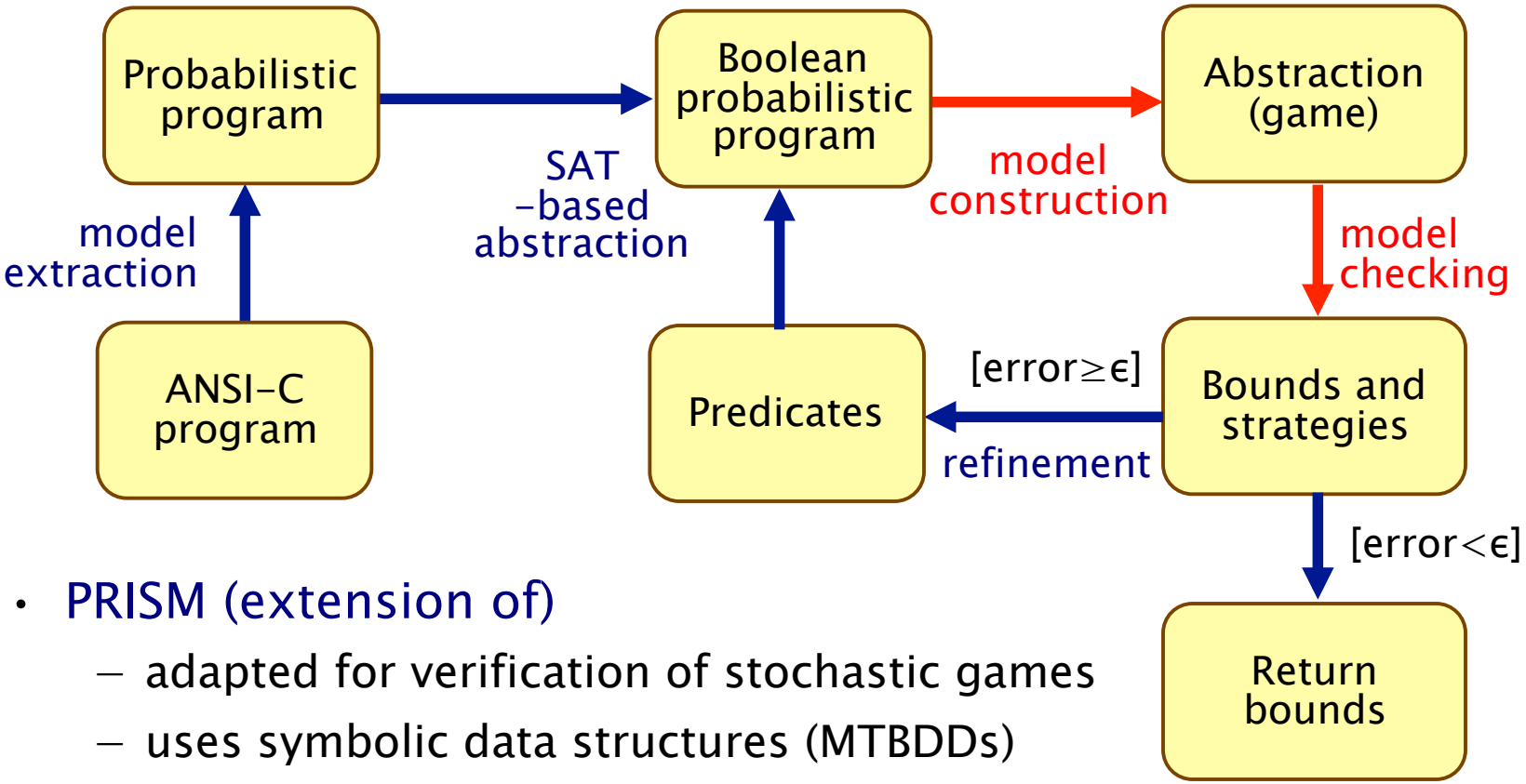
Concrete program (MDP)



Abstraction (game)

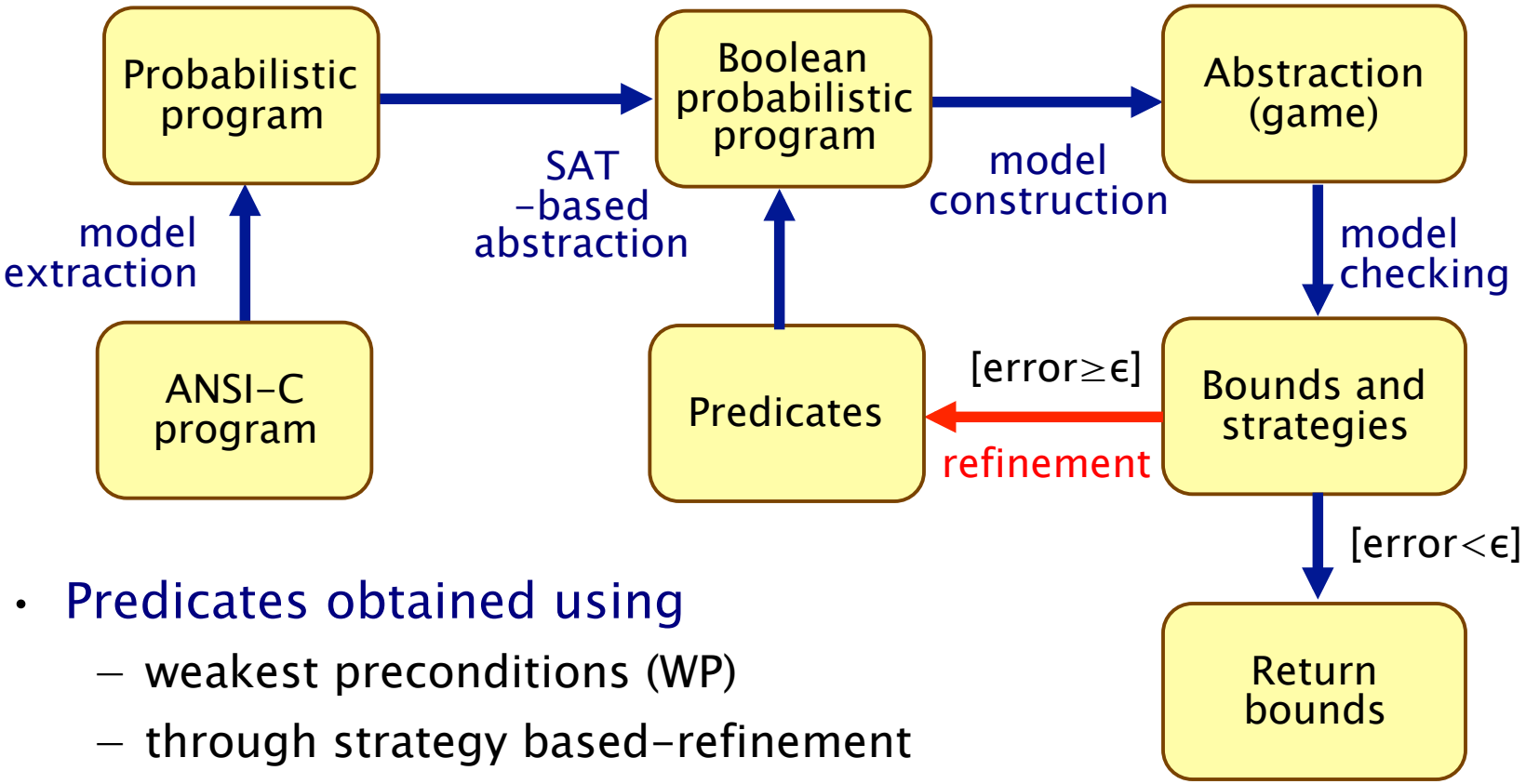


Abstraction-refinement loop



- 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

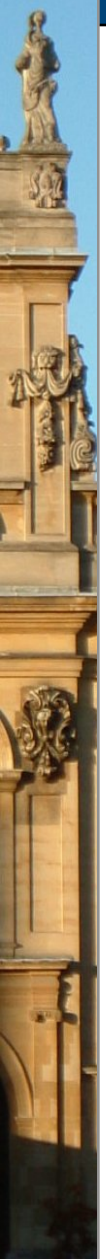
Abstraction-refinement loop



- 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”



Conclusions

- Probabilistic model checking using MDPs
  - automated formal verification of systems exhibiting both probabilistic and nondeterministic behaviour
- Abstraction approach for MDPs using 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
- Quantitative software verification
  - tool chain using state-of-the-art techniques and tools
- Current & future work
  - improved refinement heuristics, better handling of loops
  - extend to allow imprecise abstractions