Fundamentals of Probabilistic Model Checking

Sebastian Junges, **Joost-Pieter Katoen**

Radboud Universiteit

Software Modeling and Verification Chair

RWTH AACHEN UNIVERSITY

UNIVERSITY OF TWENTE.

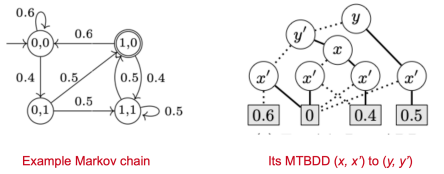
Tutorial Overview

1. 

Fundamentals of
Probabilistic Model Checking

2. 

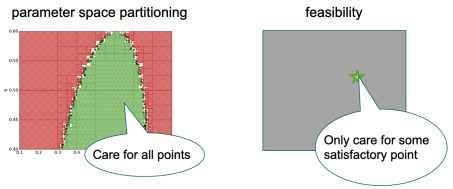
Probabilistic Model Checking with Storm:
Hands-on Slides

3. 

Example Markov chain

Its MTBDD (x, x') to (y, y')

Automated Symbolic Reasoning

4. 

Parameter Synthesis in Markov Models

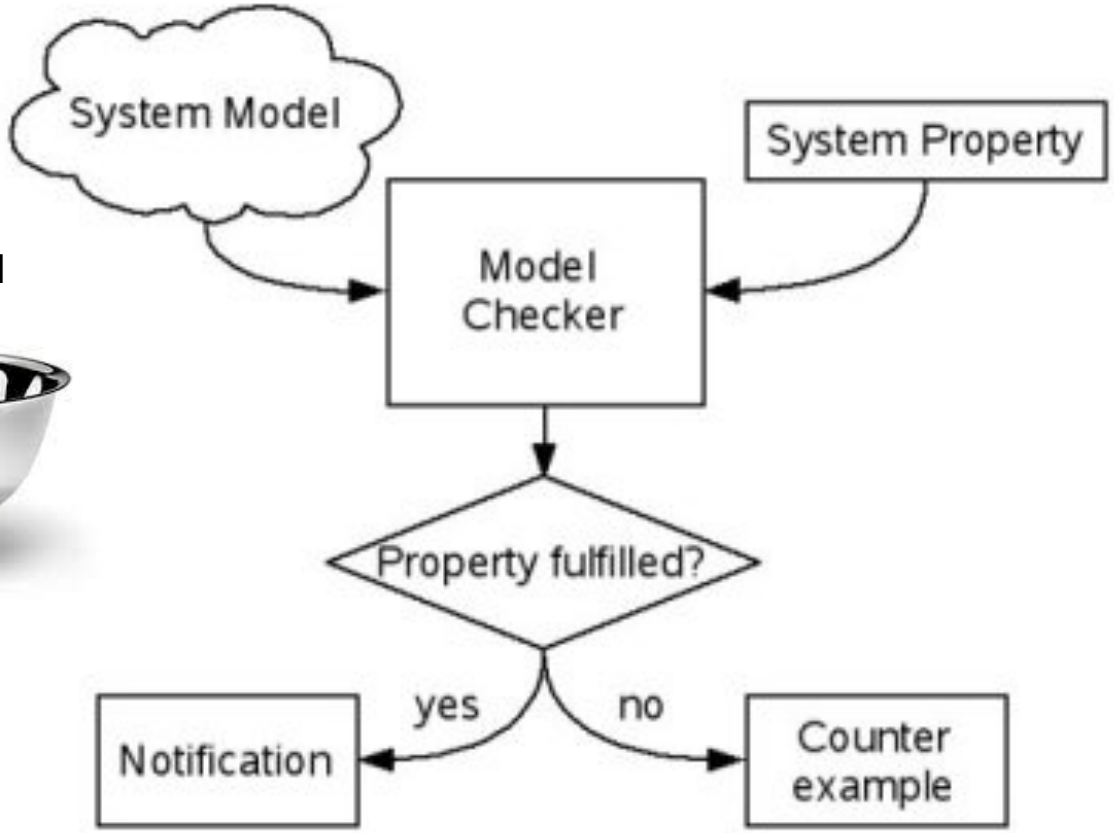
Model Checking

Edmund Clarke

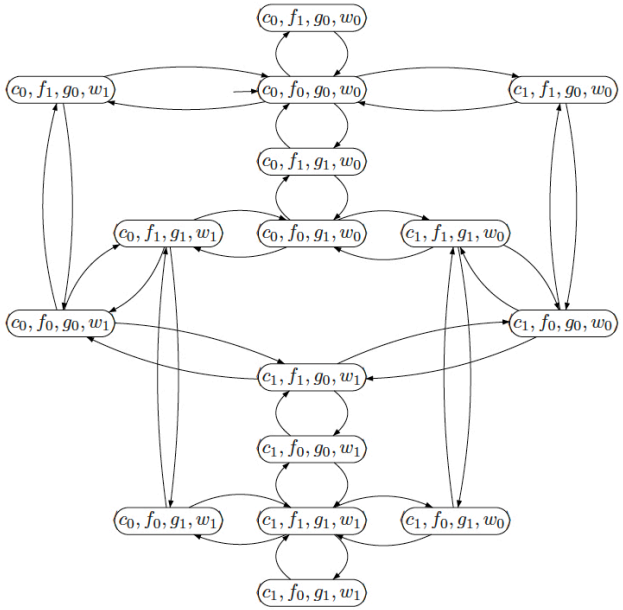
Allen Emerson

Joseph Sifakis

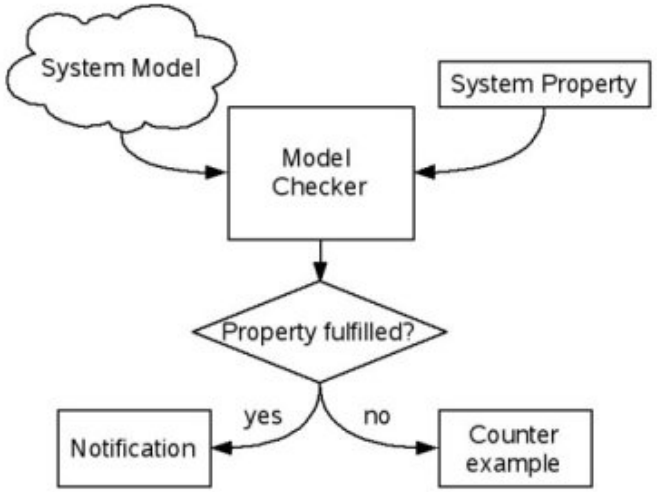
Turing Award
2007



A Toy Example: The Wolf-Goat-Cabbage Puzzle



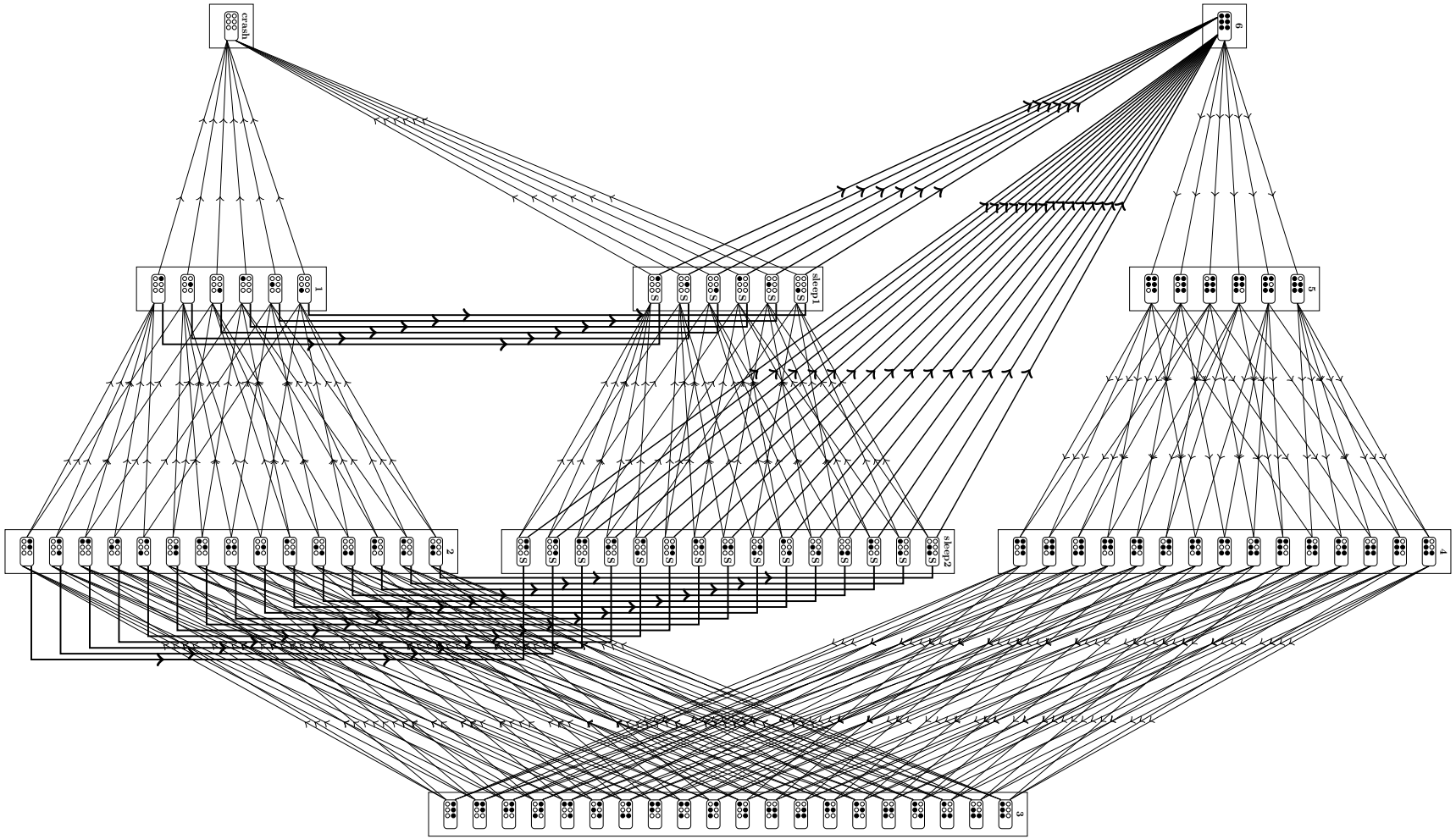
$$\Big(\bigwedge_{i=0,1} (w_i \wedge g_i \rightarrow f_i) \ \wedge \ (c_i \wedge g_i \rightarrow f_i) \Big) \ \mathsf{U} \ (c_1 \wedge f_1 \wedge g_1 \wedge w_1).$$



$\langle c_0, f_0, g_0, w_0 \rangle$	goat to riverbank 1
$\langle c_0, f_1, g_1, w_0 \rangle$	ferryman comes back to riverbank 0
$\langle c_0, f_0, g_1, w_0 \rangle$	cabbage to riverbank 1
$\langle c_1, f_1, g_1, w_0 \rangle$	goat back to riverbank 0
$\langle c_1, f_0, g_0, w_0 \rangle$	wolf to riverbank 1
$\langle c_1, f_1, g_0, w_1 \rangle$	ferryman comes back to riverbank 0
$\langle c_1, f_0, g_0, w_1 \rangle$	goat to riverbank 1
$\langle c_1, f_1, g_1, w_1 \rangle$	

al

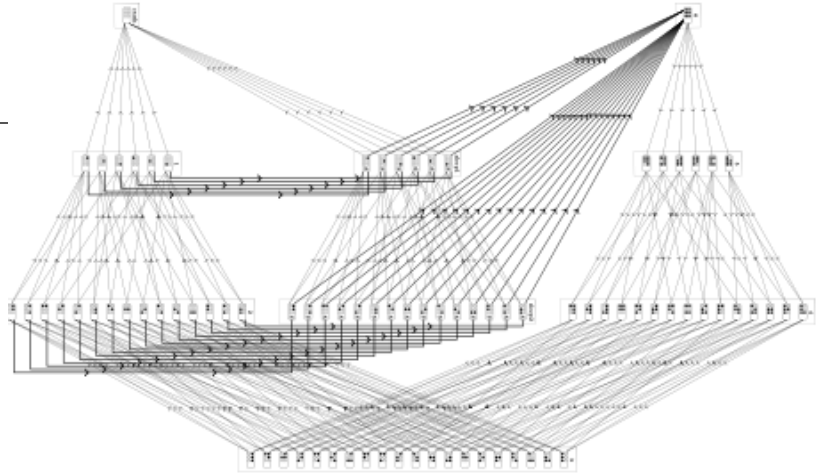
Models are Huge

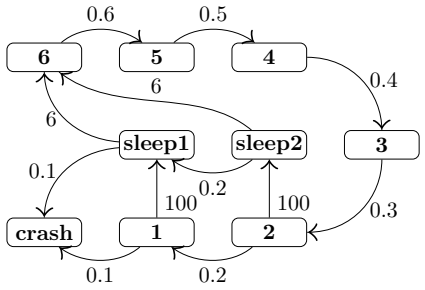


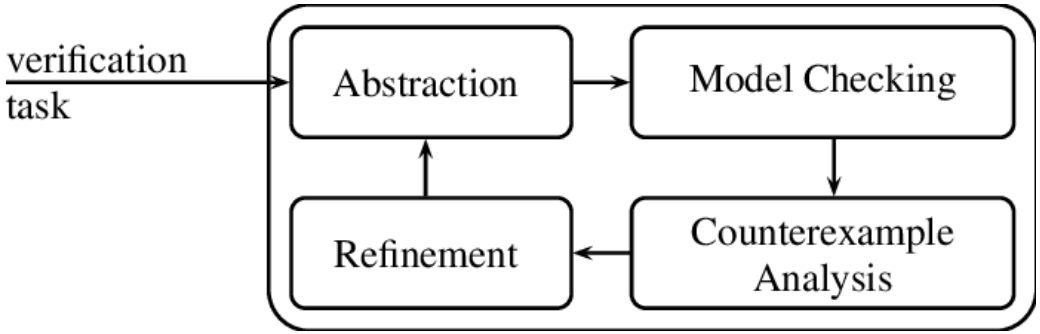
Simplified model of a Hubble telescope

Treating Gigantic Models

- Use compact data structures
- Make models smaller prior to checking them
- Try to make them even smaller
- Until obtaining the smallest possible model
- While preserving the properties of interest
- Do this all algorithmically: full automation



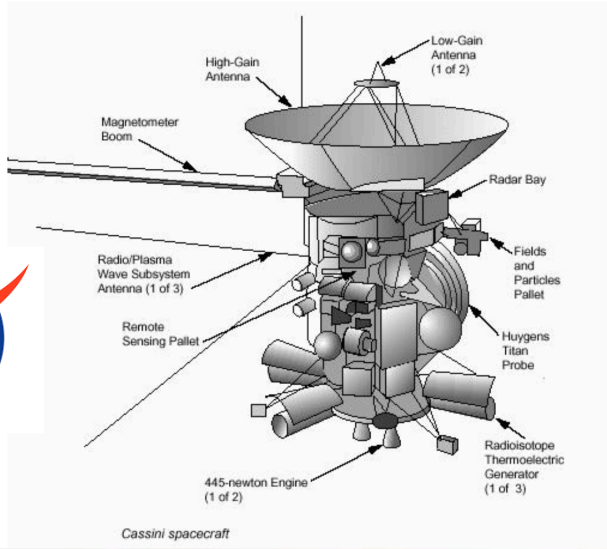


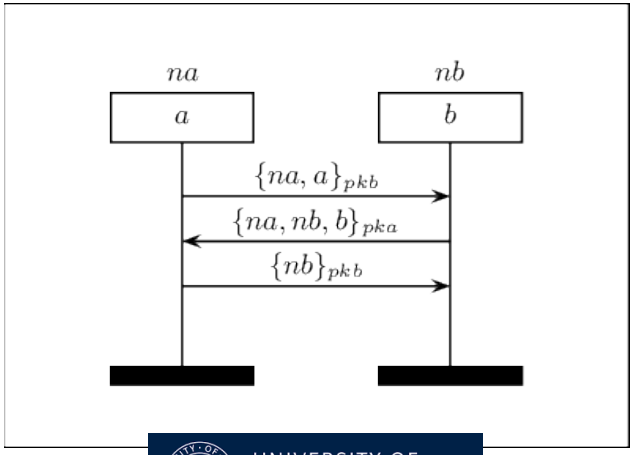


Striking Examples

Microsoft

NASA





UNIVERSITY OF OXFORD

PHILIPS

Probabilistic Model Checking

Probabilistic Model Checking

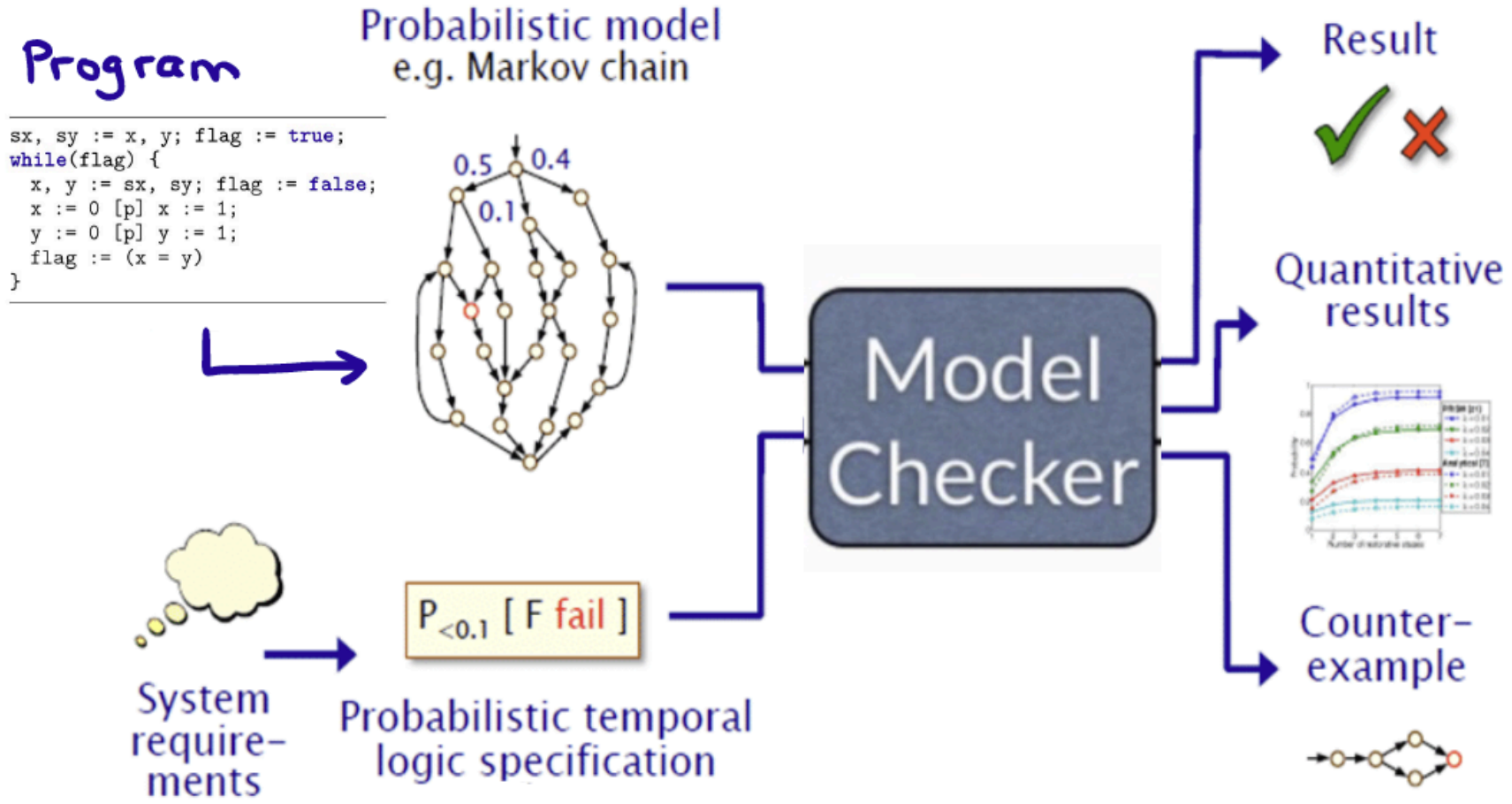
Take-home message:

Model Checking of Markov Chains :=

Computing Reachability Probabilities.

Efficient. Sound.

Probabilistic Model Checking



https://www.stormchecker.org

A Toy Example: The Lost Passenger Ticket Problem

- Passengers are queueing to board a fully-booked airplane
- All – except only the first – passengers have their boarding pass
- The first (pass-less) passenger randomly picks a seat
- All other passengers proceed as:
  - take your seat, if the seat on boarding pass is free
  - otherwise, randomly pick a seat

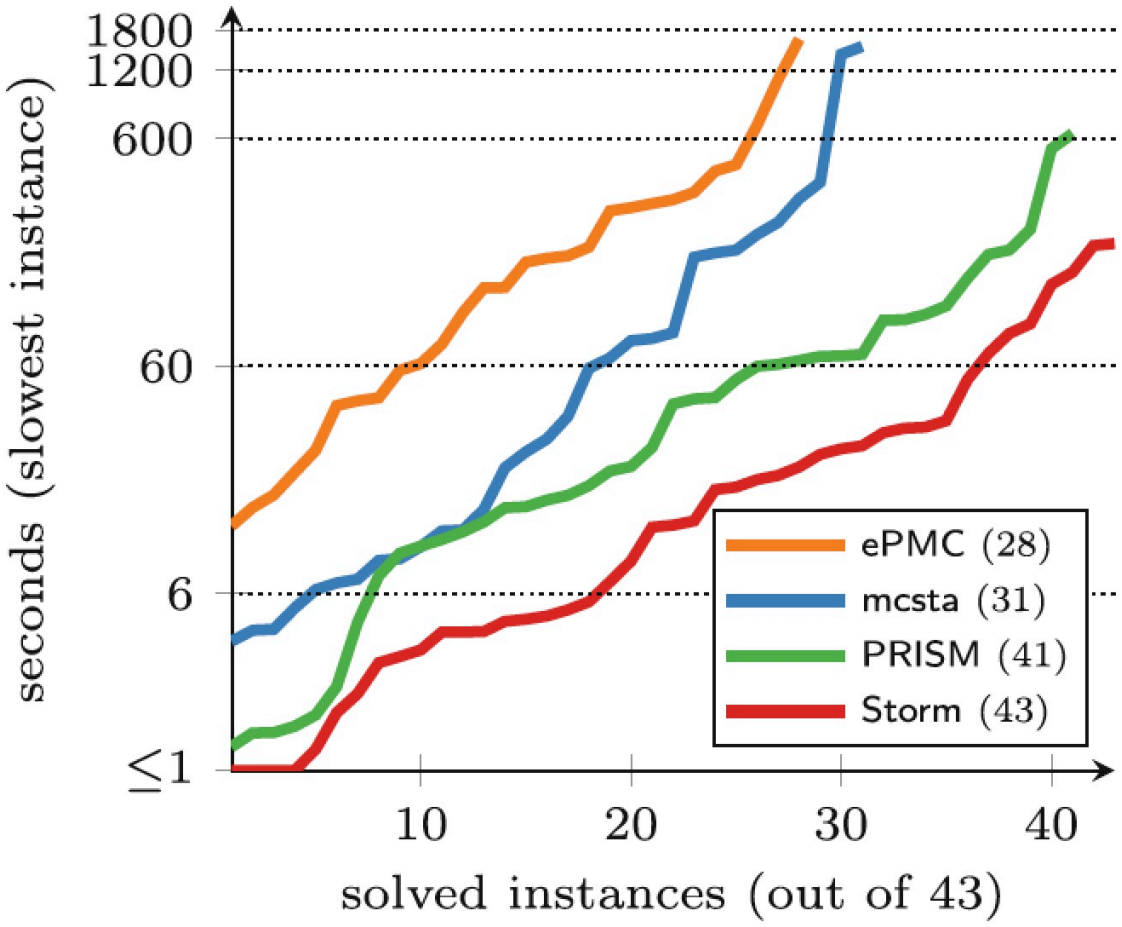
Q: how likely does the last passenger get the seat on her boarding pass?

A Toy Example: The Lost Passenger Ticket Problem

```
E := 1000;
roll{
  1/E    :: b := true;
  1-1/E :: b := false;}
E := E-1;

while (E > 1){
  if (!b) {
    roll {
      1/E    :: roll{
                  1/E    :: b := true;
                  1-1/E :: b := false;}
      1-1/E :: skip}}
  E := E-1;
}
return b
```

E	time (in s)
100	0.1
1,000	0.1
10,000	0.2
1,000,000	6.4
10,000,000	66.8

Efficiency


1800
1200
600
60
6
≤1
seconds (slowest instance)
10
20
30
40
solved instances (out of 43)
ePMC (28)
mcsta (31)
PRISM (41)
Storm (43)

Probabilistic Models

	Discrete	Continuous
Deterministic	discrete-time Markov chain (DTMC)	continuous-time MC
Nondeterministic	Markov decision process (MDP)	CTMDP
Compositional	Segala's probabilistic automata (PA)	Markov automata (MA)

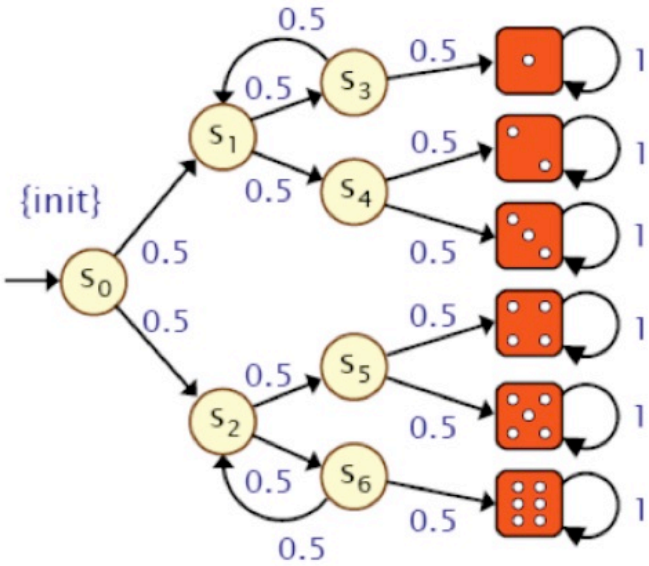
Other models: stochastic games, probabilistic pushdown automata, probabilistic timed automata

Objectives

	Logic	Monitors
Discrete time	probabilistic CTL	deterministic automata (safety and LTL)
Continuous time	probabilistic timed CTL	deterministic timed automata

Core problem: computing (timed) reachability probabilities

Key: Computing Reachability Probabilities (Knuth-Yao Die)



- Consider the event $\Diamond 4$
- We obtain:

$$x_1 = x_2 = x_3 = x_5 = x_6 = 0 \text{ and } x_4 = 1$$

$$x_{s_1} = x_{s_3} = x_{s_4} = 0$$

$$x_{s_0} = \frac{1}{2}x_{s_1} + \frac{1}{2}x_{s_2}$$

$$x_{s_2} = \frac{1}{2}x_{s_5} + \frac{1}{2}x_{s_6}$$

$$x_{s_5} = \frac{1}{2}x_5 + \frac{1}{2}x_4$$

$$x_{s_6} = \frac{1}{2}x_{s_2} + \frac{1}{2}x_6$$

- Gaussian elimination yields:

$$x_{s_5} = \frac{1}{2},\ x_{s_2} = \frac{1}{3},\ x_{s_6} = \frac{1}{6},\ \text{and}\ \boxed{x_{s_0} = \frac{1}{6}}$$

Linear Equation System

- Let $S_? = Pre^*(G) \setminus G$, the states that can reach G by > 0 steps
- $\mathbf{A} = \big(\mathbf{P}(s,t)\big)_{s,t \in S_?}$, the transition probabilities in $S_?$
- $\mathbf{b} = (b_s)_{s \in S_?}$, the probs to reach G in 1 step, i.e., $b_s = \sum_{s' \in G} \mathbf{P}(s,s')$

Theorem

The vector $\mathbf{x} = (x_s)_{s \in S_?}$ with $x_s = Pr(s \models \Diamond G)$ is the unique solution of the linear equation system:

$$\mathbf{x} = \mathbf{A}\cdot\mathbf{x} + \mathbf{b} \quad \text{or, equivalently} \quad (\mathbf{I} - \mathbf{A})\cdot\mathbf{x} = \mathbf{b}$$

where $\mathbf{I}$ is the identity matrix of cardinality $|S_?|\cdot|S_?|$.

Value Iteration

- Reachability probabilities are typically obtained iteratively:

$$\mathbf{x}^{(n+1)} = \mathbf{A} \cdot \mathbf{x}^{(n)} + \mathbf{b}$$

- Then: reachability probability $Pr(\Diamond\, G)$ equals $\lim_{n\to\infty} \mathbf{x}^{(n)}$
- Question: when to halt this iterative process?
- Typical approach:

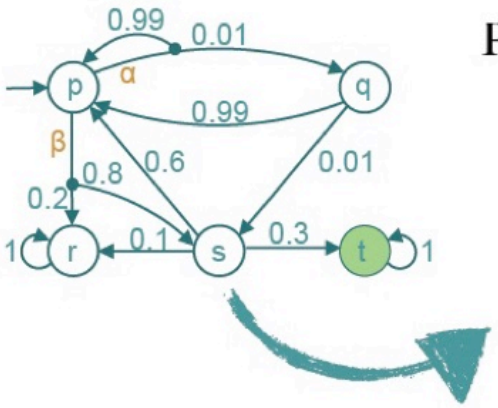
$$|\mathbf{x}^{(n+1)} - \mathbf{x}^{(n)}| \leqslant \varepsilon$$

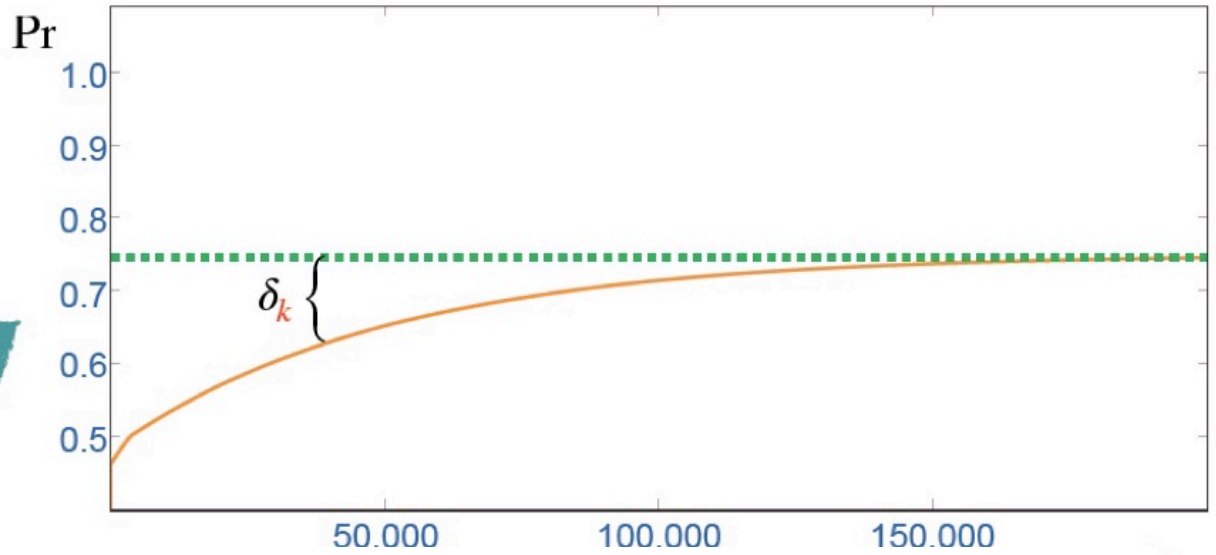
  for some ε, e.g.,10^{-6}
- Potential problem: premature convergence

  That is: iterations are stopped too early
- Verification results are obtained without guarantees

Value Iteration

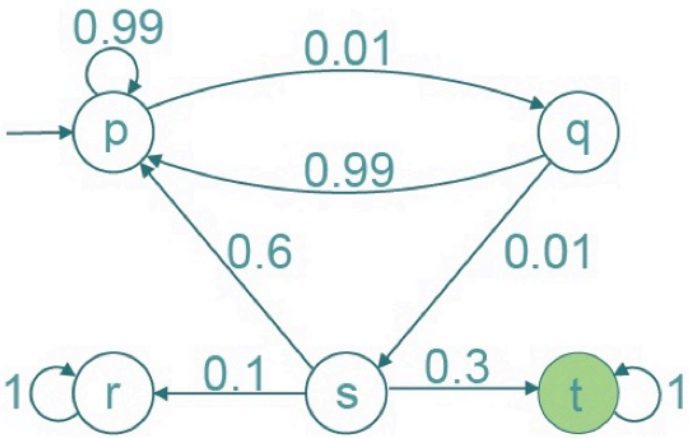
Idea: approach $Pr(\Diamond\, G)$ by computing $Pr(\Diamond^{\leqslant k} G)$ for increasing k





- Problem: δ_k is unknown
- Stopping criterion: $|Pr(\Diamond^{\leqslant k+1} G) - Pr(\Diamond^{\leqslant k} G)| \leqslant \varepsilon$
- But this is independent from the aim: $\underbrace{Pr(\Diamond\, G) - Pr(\Diamond^{\leqslant k} G)}_{\delta_k} \leqslant \varepsilon$

Value Iteration



- Exact answer: $Pr(\Diamond\, t) = \frac{3}{4}$
- Value iteration with $\varepsilon = 0,000001$ yields 0.7248
- True error: 0.0252

Sandwich the Iteration

Idea: provide bounds $\ell_k \leqslant \delta_k \leqslant u_k$ for $\delta_k = Pr(\Diamond\, G) - Pr(\Diamond^{\leqslant k}\, G)$

How to obtain these bounds? Towards an upper bound observe:

$$\delta_k = \underbrace{Pr(\Diamond\, G) - Pr(\Diamond^{\leqslant k}\, G)}_{\text{probability to reach } G \text{ in } > k \text{ steps}} \leqslant Pr(\Box^{\leqslant k}\, S_?) \cdot \max_{s \in S_?} Pr_s(\Diamond\, G)$$

For a lower bound:

$$\delta_k = \underbrace{Pr(\Diamond\, G) - Pr(\Diamond^{\leqslant k}\, G)}_{\text{probability to reach } G \text{ in } > k \text{ steps}} \geqslant Pr(\Box^{\leqslant k}\, S_?) \cdot \min_{s \in S_?} Pr_s(\Diamond\, G)$$

Sound Value Iteration

For DTMC $\mathcal{D}$, goal states $G \subseteq S$ and $k \in \mathbb{N}$:

$$Pr(\Diamond^{\leqslant k} G) + \ell_k \leqslant Pr(\Diamond G) \leqslant Pr(\Diamond^{\leqslant k} G) + u_k$$

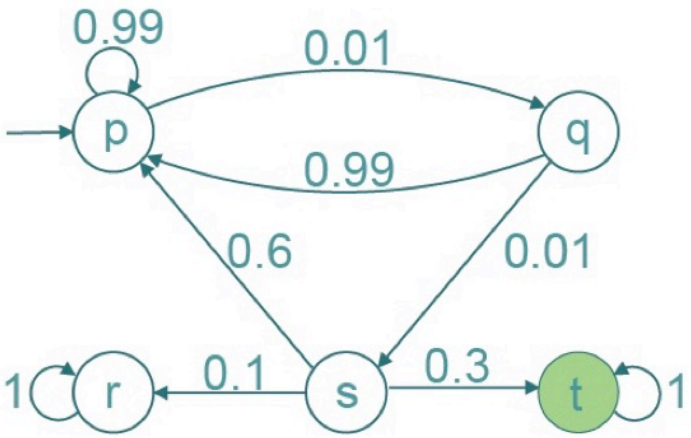
where:

$$u_k = Pr(\Box^{\leqslant k} S_?) \cdot \max_{s \in S_?} \frac{Pr_s(\Diamond^{\leqslant k} G)}{1 - Pr_s(\Box^{\leqslant k} S_?)}$$

and

$$\ell_k = Pr(\Box^{\leqslant k} S_?) \cdot \min_{s \in S_?} \frac{Pr_s(\Diamond^{\leqslant k} G)}{1 - Pr_s(\Box^{\leqslant k} S_?)}$$

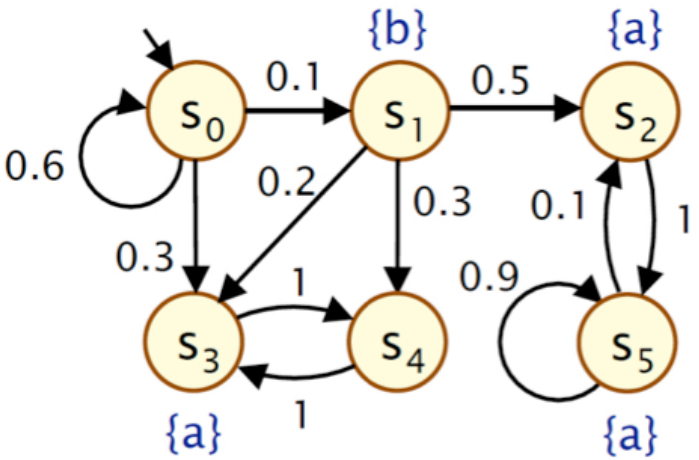
Sound Value Iteration



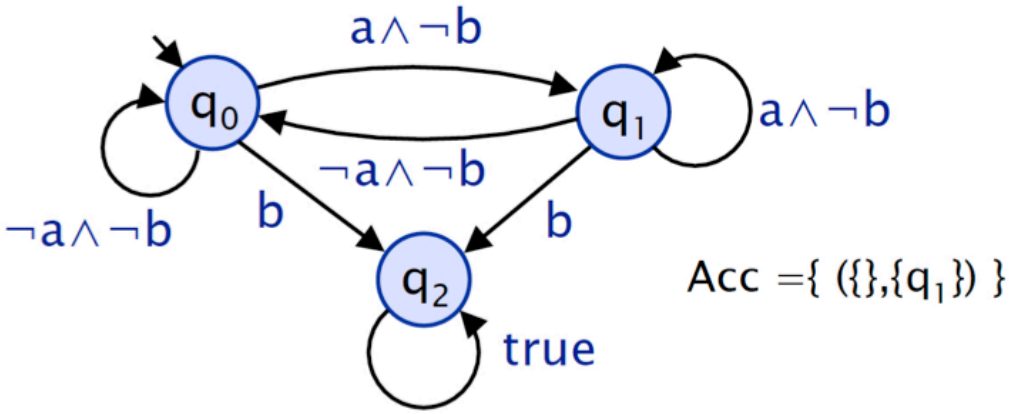
- Exact answer: $Pr(\Diamond\, t) = \frac{3}{4}$
- Value iteration with $\varepsilon = 0,000001$ yields 0.7248
- True error: 0.0252

- We have $\mathbf{l}_3 = (0.00003, 0.003, 0.3)$
- and $\mathbf{u}_3 = (0.99996, 0.996, 0.6)$
- For all $s \in S_?$ we have $\frac{\ell_3(s)}{1-u_3(s)} = \frac{3}{4}$
- Thus $\ell_3 = u_3 = \frac{3}{4}$
- Three iterations suffice for the exact answer

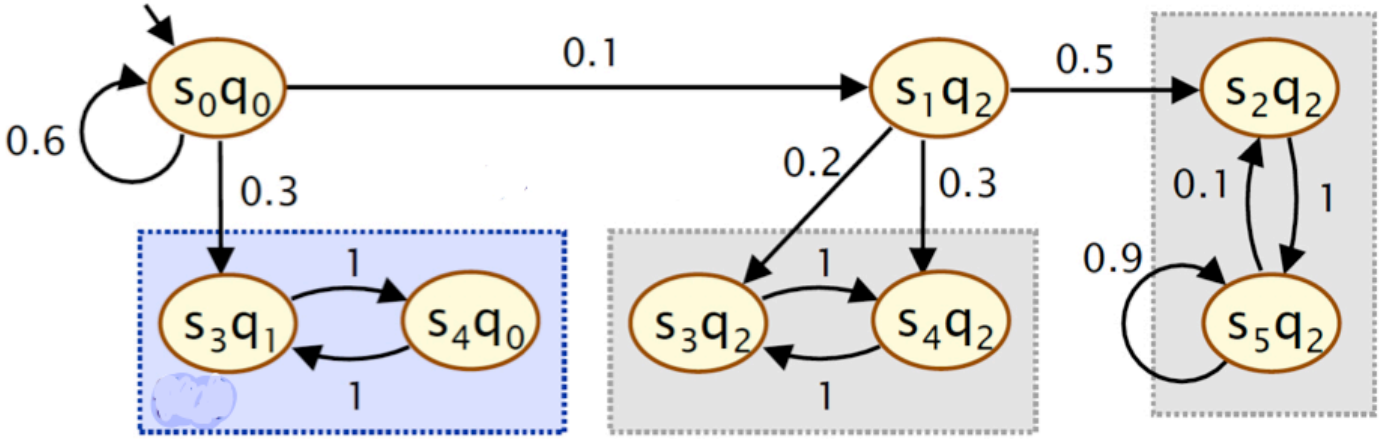
Beyond Reachability



Markov chain



“never *b* and infinitely often *a*”



probability to reach an accepting bottom SCC

(courtesy: David Parker)

Linear Temporal Logic

Linear Temporal Logic: Syntax [Pnueli 1977]

LTL formulas over the set AP obey the grammar:

$$\varphi \ ::= \ a \ \Big| \ \neg\varphi \ \Big| \ \varphi_1 \wedge \varphi_2 \ \Big| \ \bigcirc \varphi \ \Big| \ \varphi_1 \, \mathsf{U} \, \varphi_2$$

where $a \in AP$ and φ, φ_1, and φ_2 are LTL formulas.

LTL Semantics

LTL semantics

The LT-property induced by LTL formula φ over AP is:

$$Words(\varphi) = \left\{\sigma \in \left(2^{AP}\right)^{\omega} \mid \sigma \models \varphi\right\}, \text{ where } \models \text{ is the smallest relation satisfyin}$$

$$\begin{array}{lcll}
\sigma & \models & \text{true} & \\
\sigma & \models & a & \text{iff} \quad a \in A_0 \quad (\text{i.e., } A_0 \models a) \\
\sigma & \models & \varphi_1 \wedge \varphi_2 & \text{iff} \quad \sigma \models \varphi_1 \text{ and } \sigma \models \varphi_2 \\
\sigma & \models & \neg\varphi & \text{iff} \quad \sigma \not\models \varphi \\
\sigma & \models & \bigcirc \varphi & \text{iff} \quad \sigma^1 = A_1 A_2 A_3 \ldots \models \varphi \\
\sigma & \models & \varphi_1 \,\mathsf{U}\, \varphi_2 & \text{iff} \quad \exists j \geqslant 0.\ \sigma^j \models \varphi_2 \ \text{ and } \ \sigma^i \models \varphi_1,\ 0 \leqslant i < j
\end{array}$$

for $\sigma = A_0 A_1 A_2 \ldots$ we have $\sigma^i = A_i A_{i+1} A_{i+2} \ldots$ is the suffix of σ from index i on.

Relating LTL and DRA

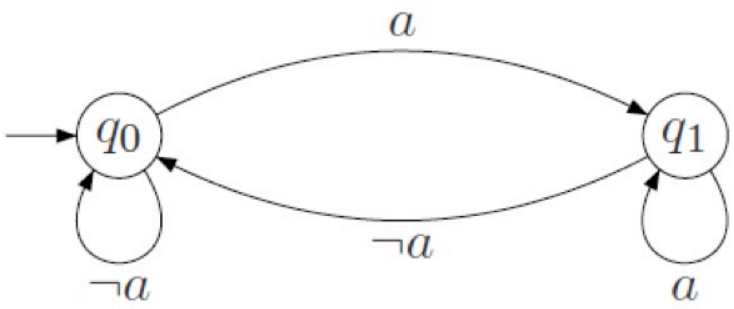
LTL are DRA-definable

For any LTL formula φ, there exists a DRA $\mathcal{A}$ such that $\mathcal{L}_\omega = \mathit{Words}(\varphi)$ where the number of states in $\mathcal{A}$ lies in $2^{2^{|\varphi|}}$.

Deterministic Rabin Automaton: “From Some Point On Always Safe”

Acceptance condition

A run of an infinite word on a DRA is accepting iff $\bigvee_{0<i\leqslant k} (\Diamond\Box\neg L_i \wedge \Box\Diamond K_i)$.



For $\mathcal{F} = \{ (L, K) \}$ with $L = \{ q_0 \}$ and $K = \{ q_1 \}$, this DRA accepts $\Diamond\Box a$

There does not exist a deterministic Büchi automaton for $\Diamond\Box a$.

LTL Model Checking = Computing Reachability Probabilities

Accepting BSCC

A BSCC T in $\mathcal{D} \otimes \mathcal{A}$ is *accepting* iff for some index $i \in \{1, \ldots, k\}$ we have:

$$T \cap (S \times L_i) = \varnothing \quad \text{and} \quad T \cap (S \times K_i) \neq \varnothing.$$

Thus, once such an accepting BSCC T is reached in $\mathcal{D} \otimes \mathcal{A}$, the acceptance criterion for the DRA $\mathcal{A}$ is fulfilled almost surely.

DRA probabilities = reachability probabilities

Let $\mathcal{D}$ be a finite DTMC, s a state in $\mathcal{D}$, $\mathcal{A}$ a DRA, and let U be the union of all accepting BSCCs in $\mathcal{D} \otimes \mathcal{A}$. Then:

$$Pr^{\mathcal{D}}(s \models \mathcal{A}) = Pr^{\mathcal{D} \otimes \mathcal{A}}(\langle s, q_s \rangle \models \Diamond U) \quad \text{where} \quad q_s = \delta(q_0, L(s)).$$

Remind This

Take-home message

Model checking a finite DTMC against various automata models reduces to computing reachability probabilities in a synchronous product of the DTMC and the automaton.

Probabilistic Models

	Discrete	Continuous
Deterministic	discrete-time Markov chain (DTMC)	continuous-time MC
Nondeterministic	Markov decision process (MDP)	CTMDP
Compositional	Segala's probabilistic automata (PA)	Markov automata (MA)

Probabilistic Model Checking

Take-home message:

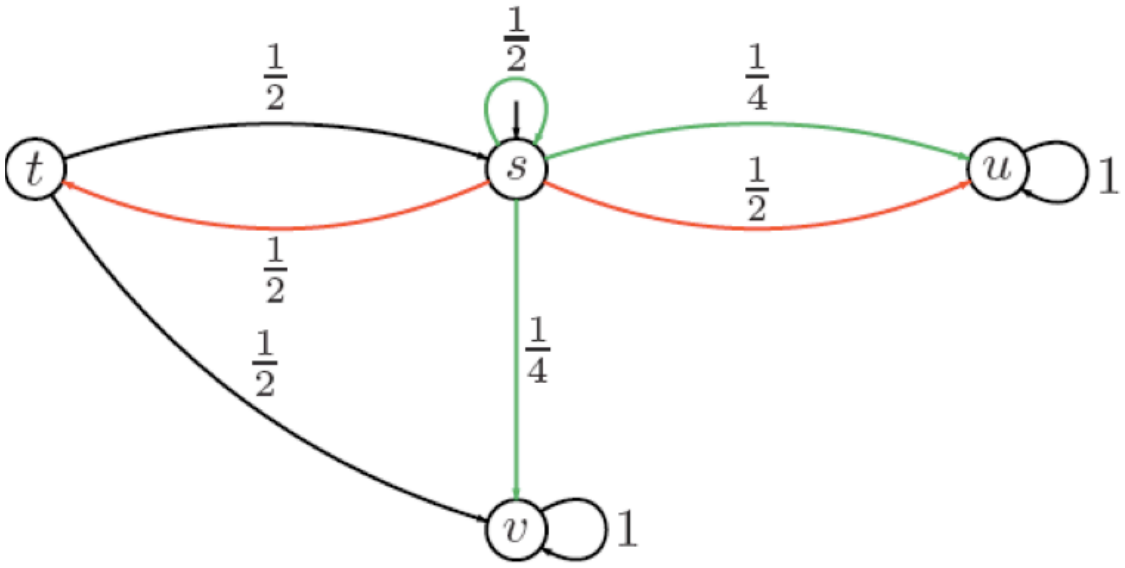
MDP Model Checking :=

Computing Extremal Reachability Probabilities.

Efficient. Sound.

Markov Decision Processes

An MDP is a DTMC in which in any state a non-deterministic choice between probability distributions exists.



Policies

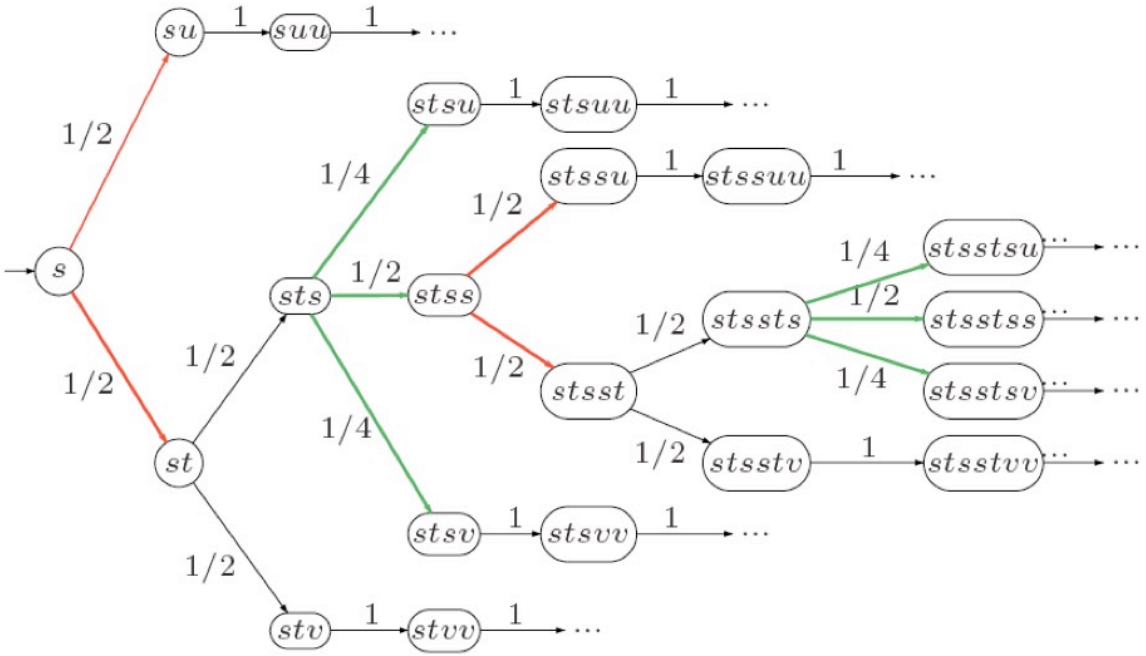
To solve MDPs, non-determinism is resolved by an oracle, called a policy.

Policy

A policy for MDP M is a function $\mathfrak{S}$ that for a give finite sequence of states through $\mathcal{M}$ yields an action (= color) to take next.

Different types of policies exist: history-dependent (as above) versus positional (or bounded memory), deterministic versus randomised.

MDP + Policy = Markov Chain



Induced DTMC for a policy that alternates between selecting red and green starting with red.

Reachability Probabilities in MDPs

Maximal and minimal reachability probabilities

The minimal reachability probability of $G \subseteq S$ from $s \in S$ is:

$$Pr^{\min}(s \models \Diamond G) = \inf_{\mathfrak{S}} Pr^{\mathfrak{S}}(s \models \Diamond G)$$

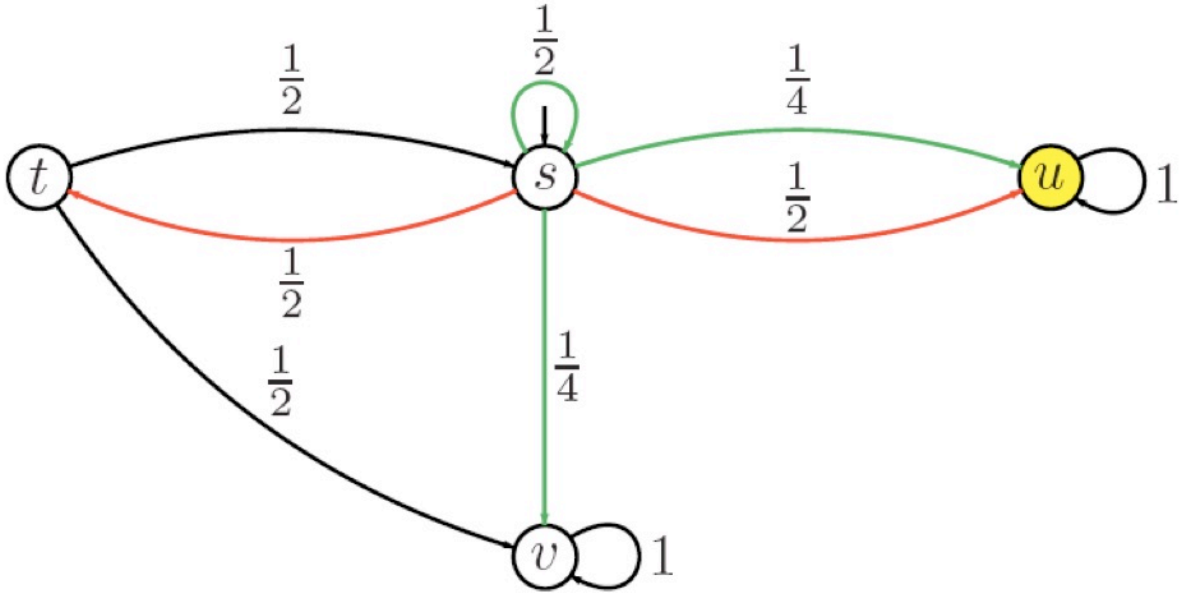
In a similar way, the maximal reachability probability of $G \subseteq S$ is:

$$Pr^{\max}(s \models \Diamond G) = \sup_{\mathfrak{S}} Pr^{\mathfrak{S}}(s \models \Diamond G).$$

where policy $\mathfrak{S}$ ranges over all, infinitely (countably) many, policies.

Typically no discounting is considered

Maximal Reachability Probabilities in MDPs



equation system for reachability objective $\Diamond\{ u \}$ is:

$$x_u = 1 \text{ and } x_v = 0$$

$$x_s = \max\{ \tfrac{1}{2}x_s + \tfrac{1}{4}x_u + \tfrac{1}{4}x_v, \tfrac{1}{2}x_u + \tfrac{1}{2}x_t \} \quad \text{and} \quad x_t = \tfrac{1}{2}x_s + \tfrac{1}{2}x_v$$

Positional Policies Suffice for Reachability

A positional policy selects the next action only based on the current state.

Existence of optimal positional policies

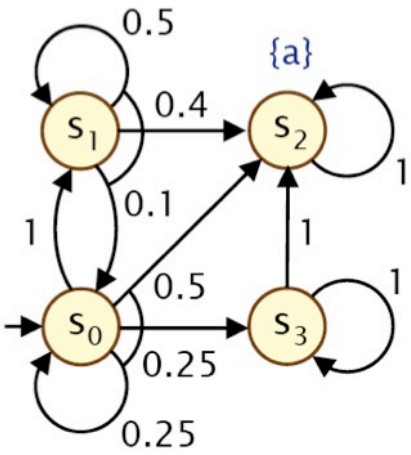
Let $\mathcal{M}$ be a finite MDP with state space S, and $G \subseteq S$. There exists a positional policy $\mathfrak{S}$ such that for any $s \in S$ it holds:

$$Pr^{\mathfrak{S}}(s \models \Diamond G) = Pr^{\max}(s \models \Diamond G).$$

A similar result holds for minimal reachability probabilities.

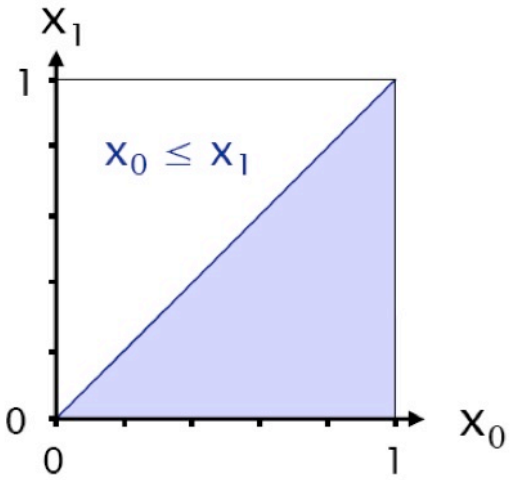
Techniques to obtain these policies: value or policy iteration, linear programming

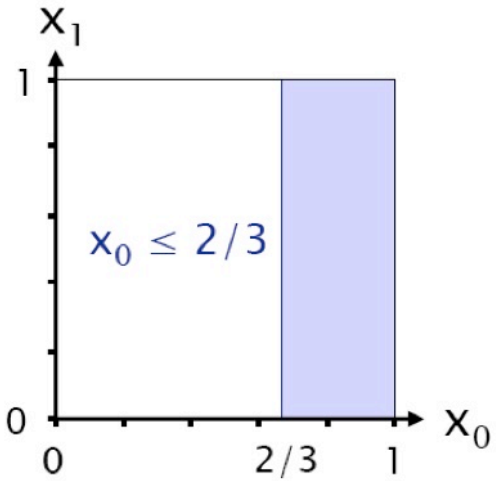
Linear Programming

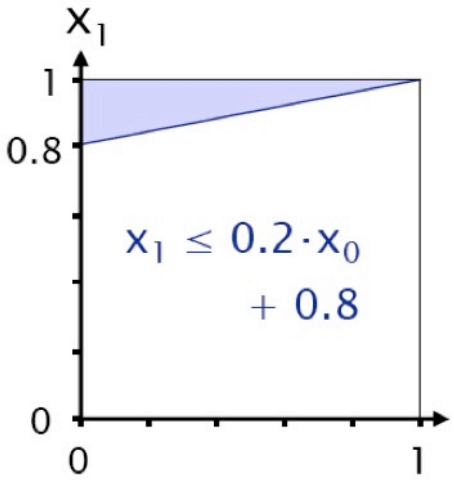


- $G = \{ s_2 \}, S^{\min}_{=0} = \{ s_3 \}, S \setminus (G \cup S^{\min}_{=0}) = \{ s_0, s_1 \}.$
- Maximise $x_0 + x_1$ subject to the constraints:

$$\begin{aligned} x_0 &\leqslant x_1 \\ x_0 &\leqslant \tfrac{2}{3} \\ x_1 &\leqslant \tfrac{2}{5} \cdot x_0 + \tfrac{4}{5} \end{aligned}$$

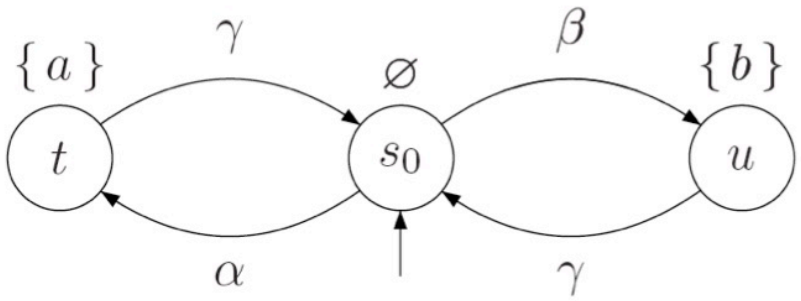






(courtesy: David Parker)

Policies for LTL



Positional policy $\mathfrak{S}_\alpha$ always chooses α in state s_0
Positional policy $\mathfrak{S}_\beta$ always chooses β in state s_0. Then:

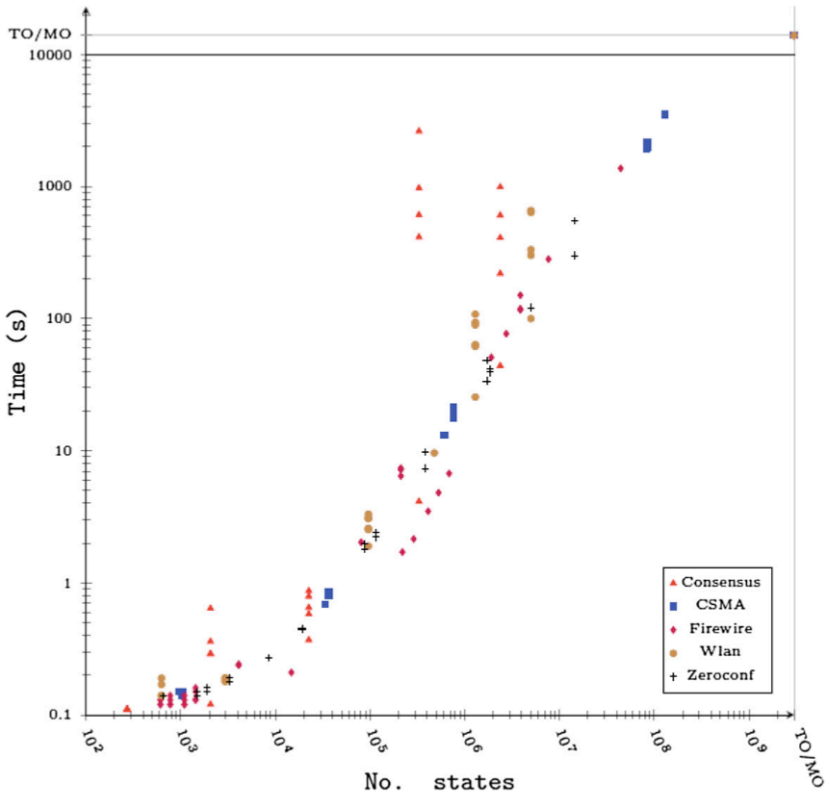
$$Pr_{\mathfrak{S}_\alpha}(s_0 \models \Diamond a \wedge \Diamond b) \;=\; Pr_{\mathfrak{S}_\beta}(s_0 \models \Diamond a \wedge \Diamond b) \;=\; 0.$$

Now consider the policy $\mathfrak{S}_{\alpha\beta}$ which alternates between selecting α and β. Then:

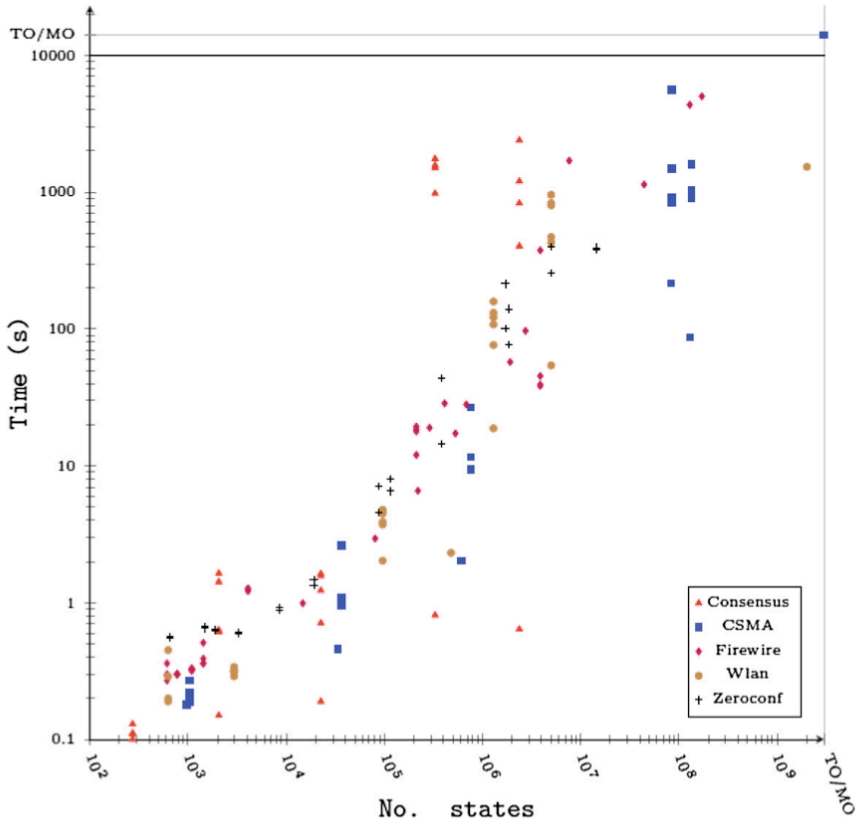
$$Pr_{\mathfrak{S}_{\alpha\beta}}(s_0 \models \Diamond a \wedge \Diamond b) \;=\; 1.$$

Positional policies do not suffice.
LTL requires finite memory policies

MDP Model Checking Statistics



Explicit state model checking



Symbolic model checking

https://www.stormchecker.org

Multi-Objective Model Checking

Take-home message:

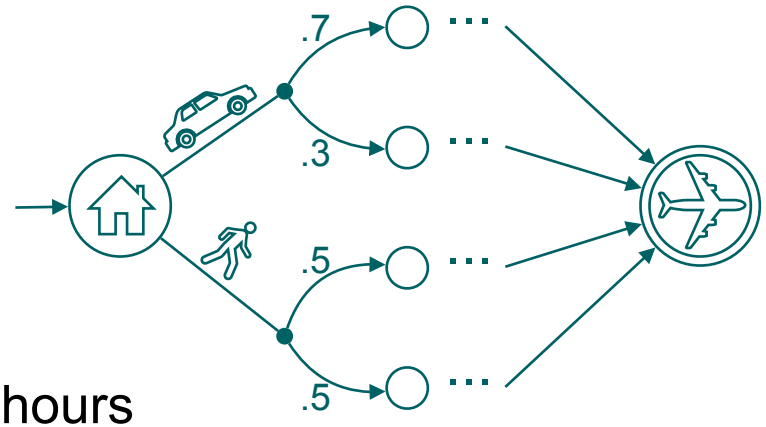
MDP Multi-Objective Model Checking :=

Verifying Arbitrary Combinations of Objectives

Efficient. Using Dynamic Programming.

Multi-Objective Model Checking

- Model trip to airport as a Markov decision process (MDP)
  - Nondeterminism
  - Probabilistic branching
  - Costs (here: travel time)

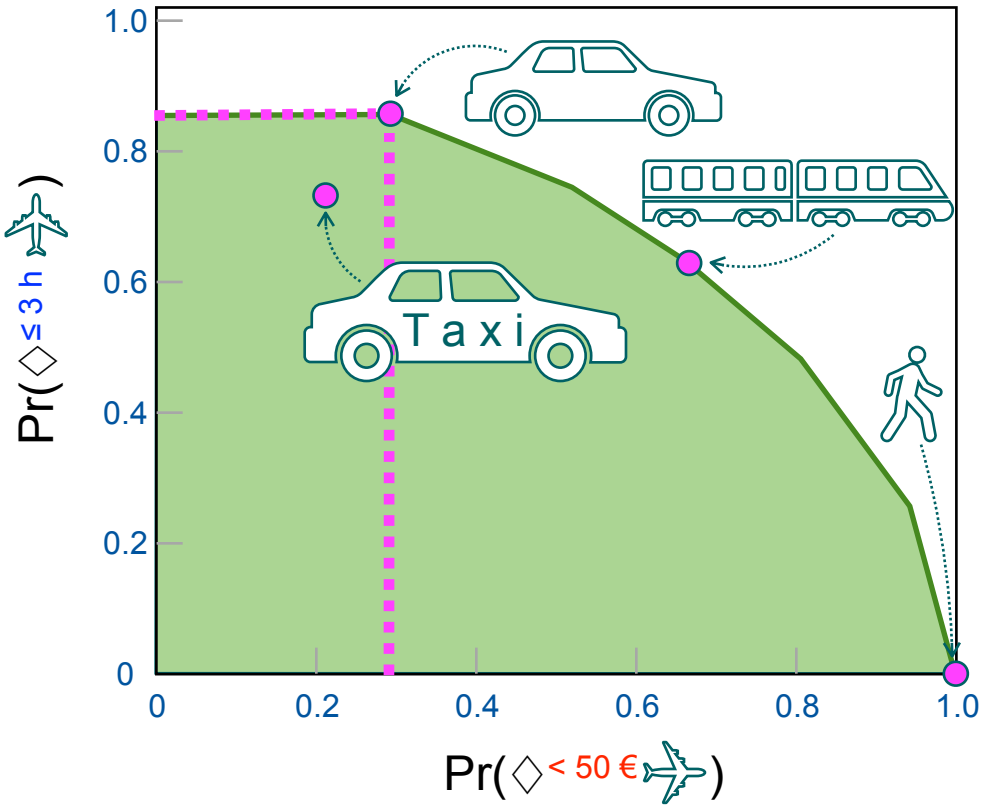


- Minimise expected travel time
- E.g., Exp(♢ ✈) = 2 hours; departure in 3 hours
- **Better:** Maximise probability to arrive before the flight departs
- Other types of cost can play a role too: money, fuel, pollution, stress, ...

Sound Value Iteration

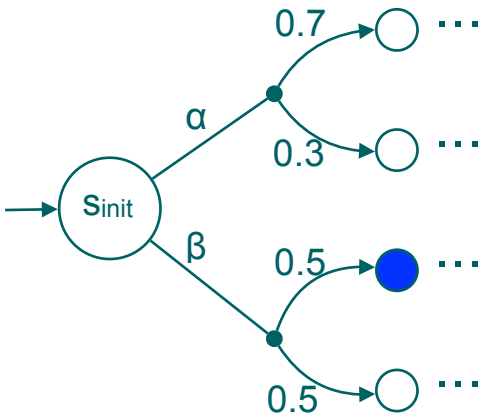
Analyse Tradeoffs Between Objectives

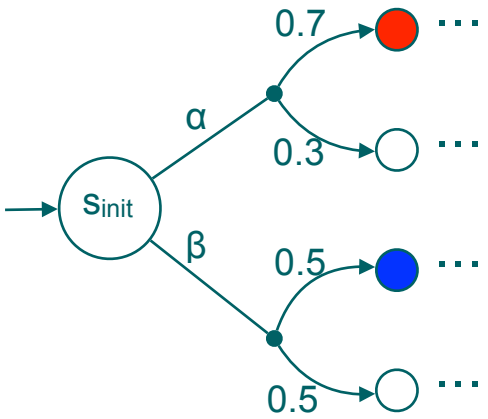
Arrive within 3 hours
vs.
Invest less than 50 €



Single vs. Multiple Objectives

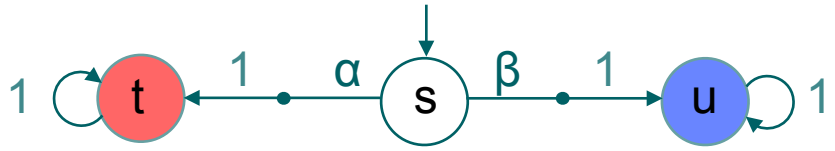
- Single-objective: maximal probability
  - $Pr_{max}(\Diamond G) \coloneqq \max_{\mathfrak{S}} Pr^{\mathfrak{S}}(\Diamond G)$
- Positional policy $\mathfrak{S}$ resolves nondeterminism:
  - $\mathfrak{S}(s)(\alpha)$ = “prob. to pick action α in state s"
- Multi-objective: trade-off
  - $Pr_{max}(\Diamond G_1)$ vs. $Pr_{max}(\Diamond G_2)$ vs. ...
  - No single policy maximises all probabilities
- Randomised policy $\mathfrak{S}$ resolves nondeterminism:
  - $\mathfrak{S}(\pi)(\alpha)$ = “prob. to pick action α after finite path π"





Policy Requirements

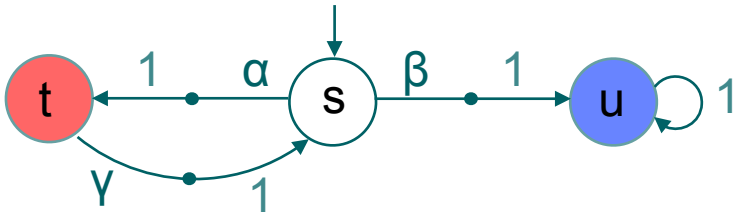
In general, we need policies with randomisation and finite memory, e.g.:



$(0.5, 0.5) \in \mathfrak{A}(\Diamond\{t\}, \Diamond\{u\})$?

achievable points

Only with randomised policy $\mathfrak{S}(s)(\alpha) = \mathfrak{S}(s)(\beta) = 0.5$



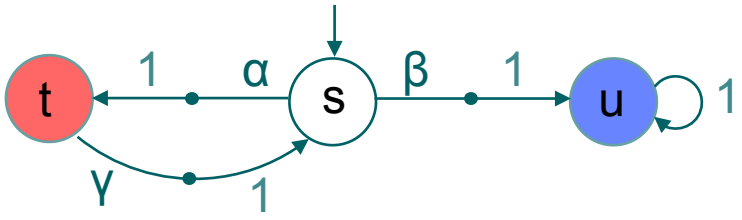
$(1, 1) \in \mathfrak{A}(\Diamond\{t\}, \Diamond\{u\})$?

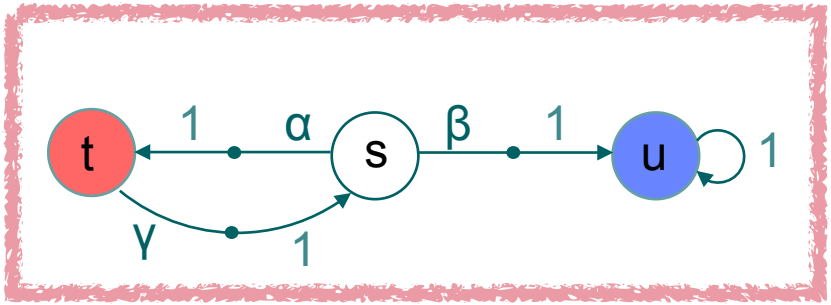
achievable points

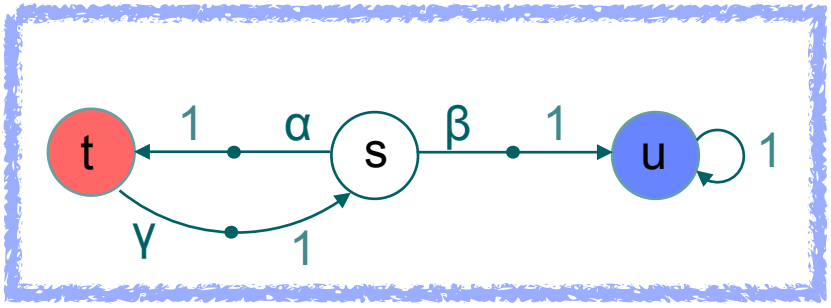
Only with finite-memory policy $\mathfrak{S}(\pi)(\alpha) = \begin{cases} 1, \text{ if } \pi \text{ has not visited } t, \text{ yet} \\ 0, \text{ otherwise} \end{cases}$

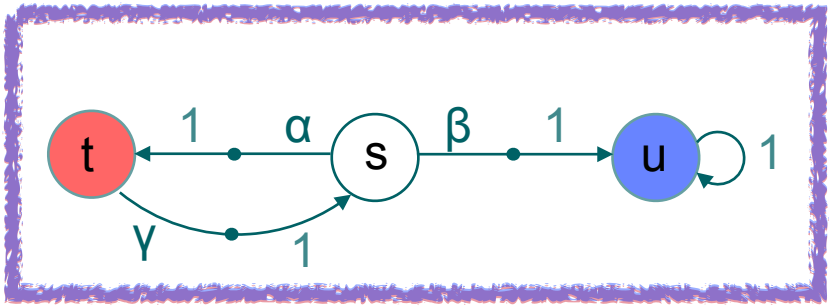
Goal Unfolding

- A policy may need to memorise which set G_i has been reached already
- Idea: encode this information into the state space
- Then: positional (randomised) policies suffice

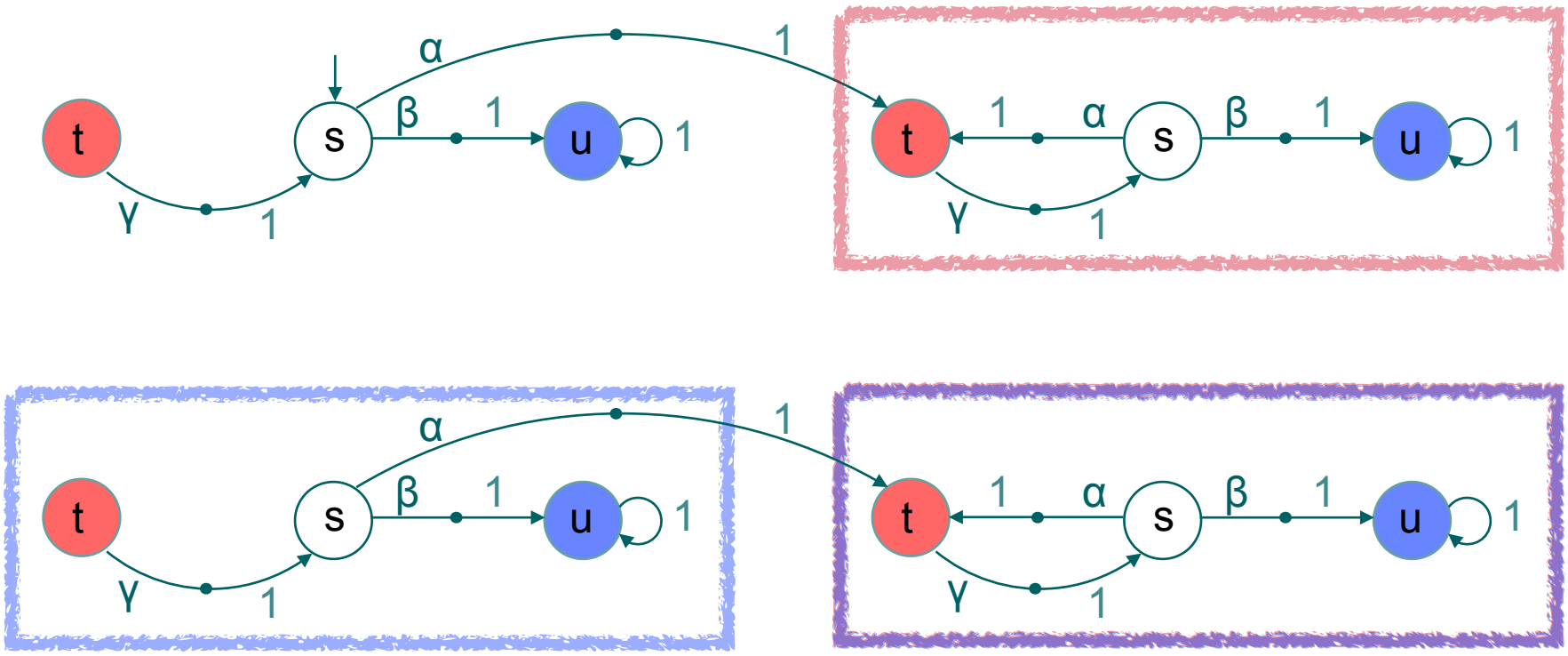




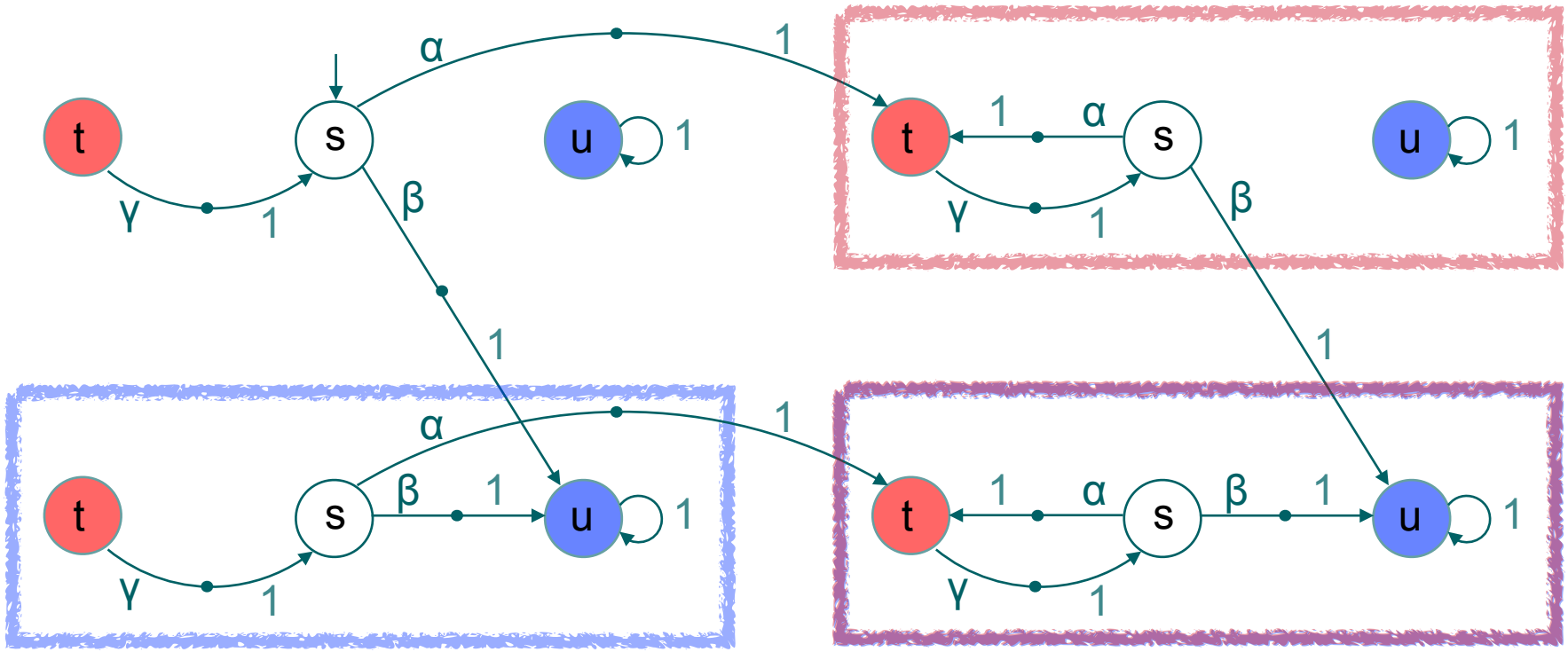




Goal Unfolding

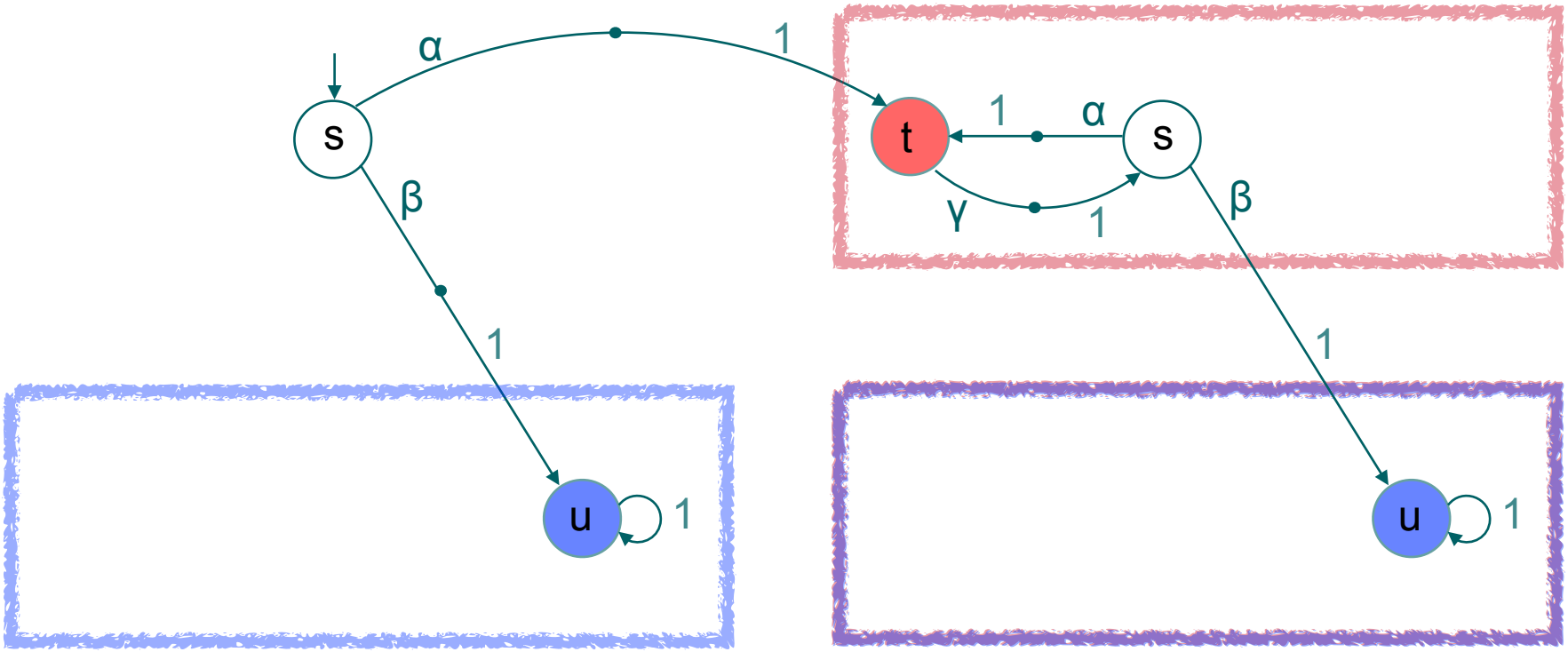


Goal Unfolding



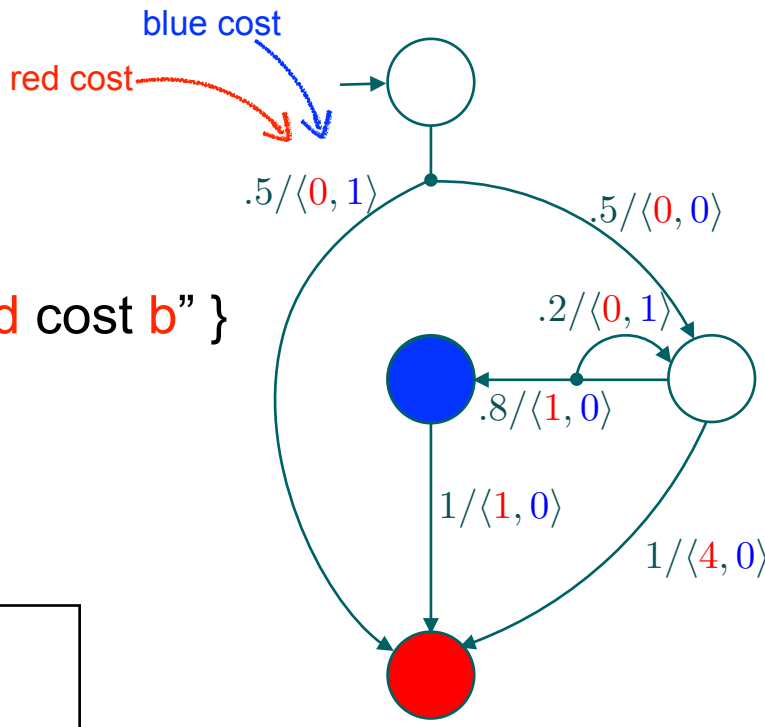
Goal Unfolding

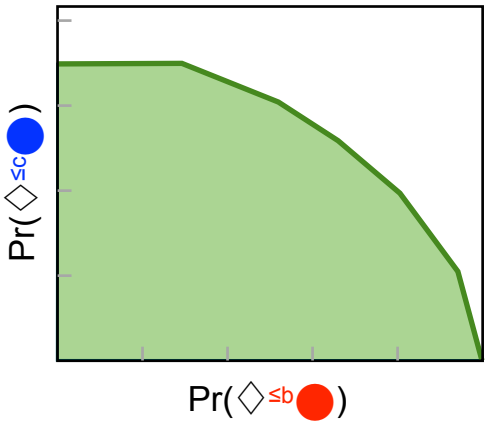
- A policy may need to memorise which set G_i has been reached already
- Idea: encode this information into the state space
- Then: positional (randomised) policies suffice



Cost-Bounded MDPs

- MDPs + multiple cost structures
- Policy σ resolves nondeterminism
- $\Pr^{\sigma}(\Diamond^{\leq b}\bullet)$ = "Pr { reach $\bullet$ with at most red cost b" }
- Multi-objective: tradeoff
  - *multi* [$\Pr_{max}(\Diamond^{\leq b}\bullet)$, $\Pr_{max}(\Diamond^{\leq c}\bullet)$]
  - yields a Pareto curve





Complexity

Multi-Cost Bounded Reachability in MDPs is PSPACE-hard

Almost-Sure Multi-Cost Bounded Reachability is PSPACE-complete

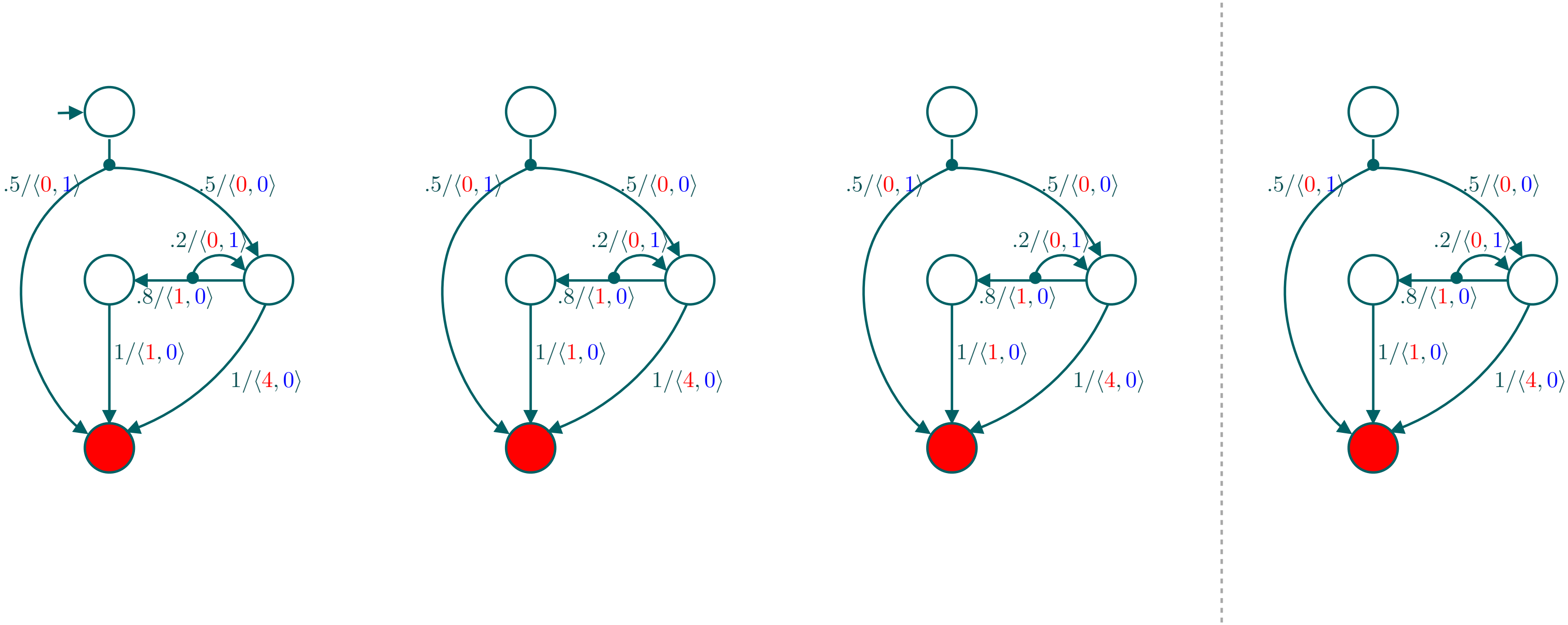
Form Methods Syst Des (2017) 50:207–248
DOI 10.1007/s10703-016-0262-7

Percentile queries in multi-dimensional Markov decision processes

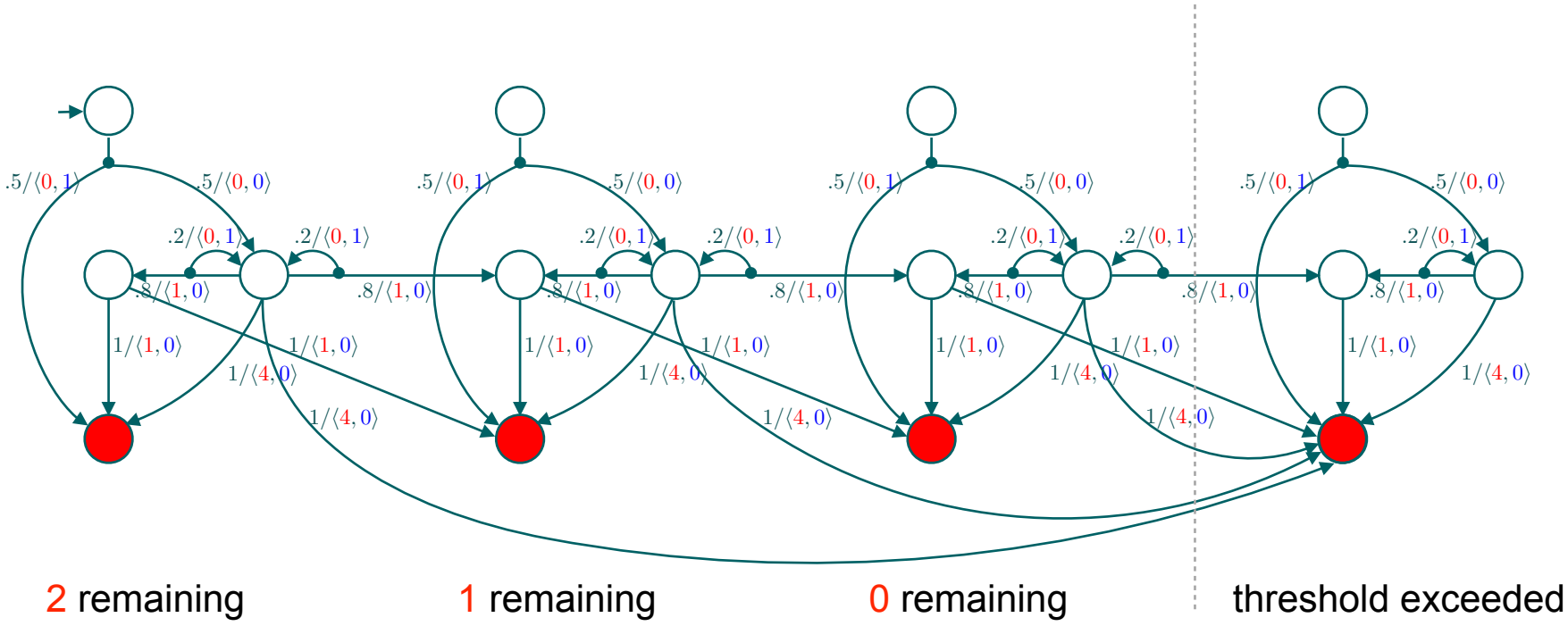
Mickael Randour[1] · Jean-François Raskin[1] · Ocan Sankur[2]

Let's take a ``practical'' view: how to solve this efficiently?

Unfolding for $\mathrm{Pr}_{max}(\Diamond^{\leq 2}$ ●$)$



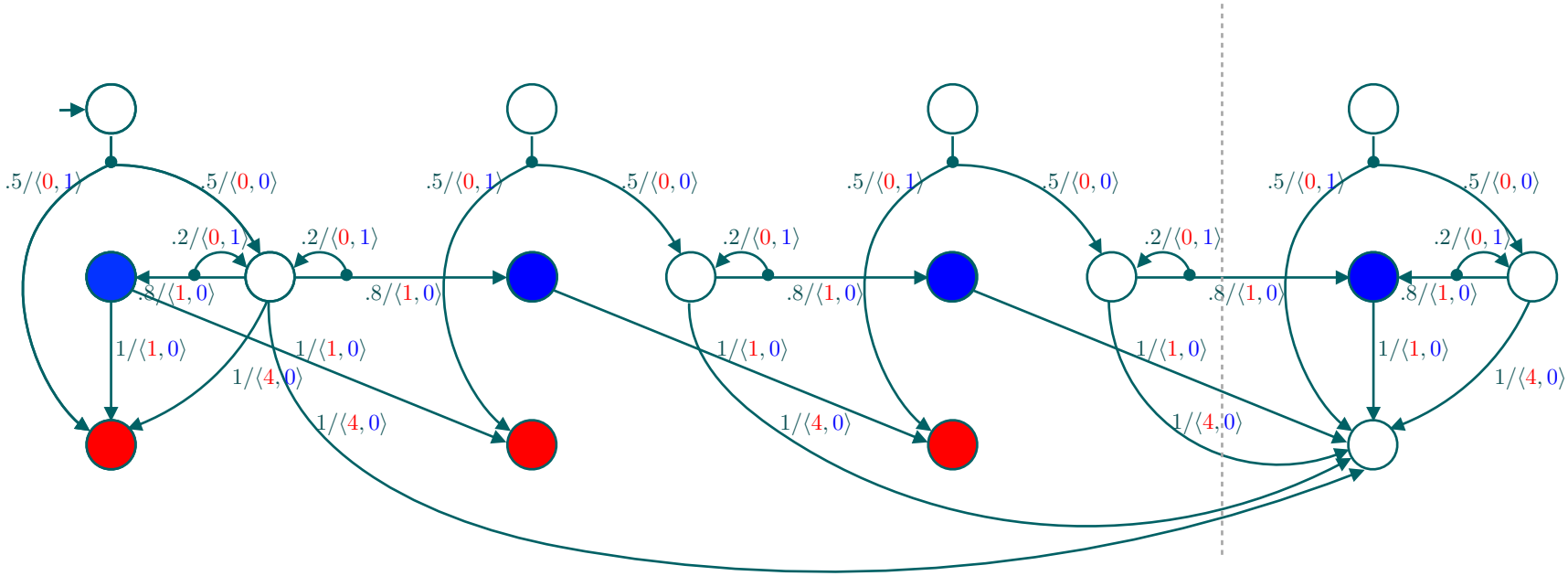
Unfolding for $\Pr_{max}(\Diamond^{\leq 2} \bullet)$



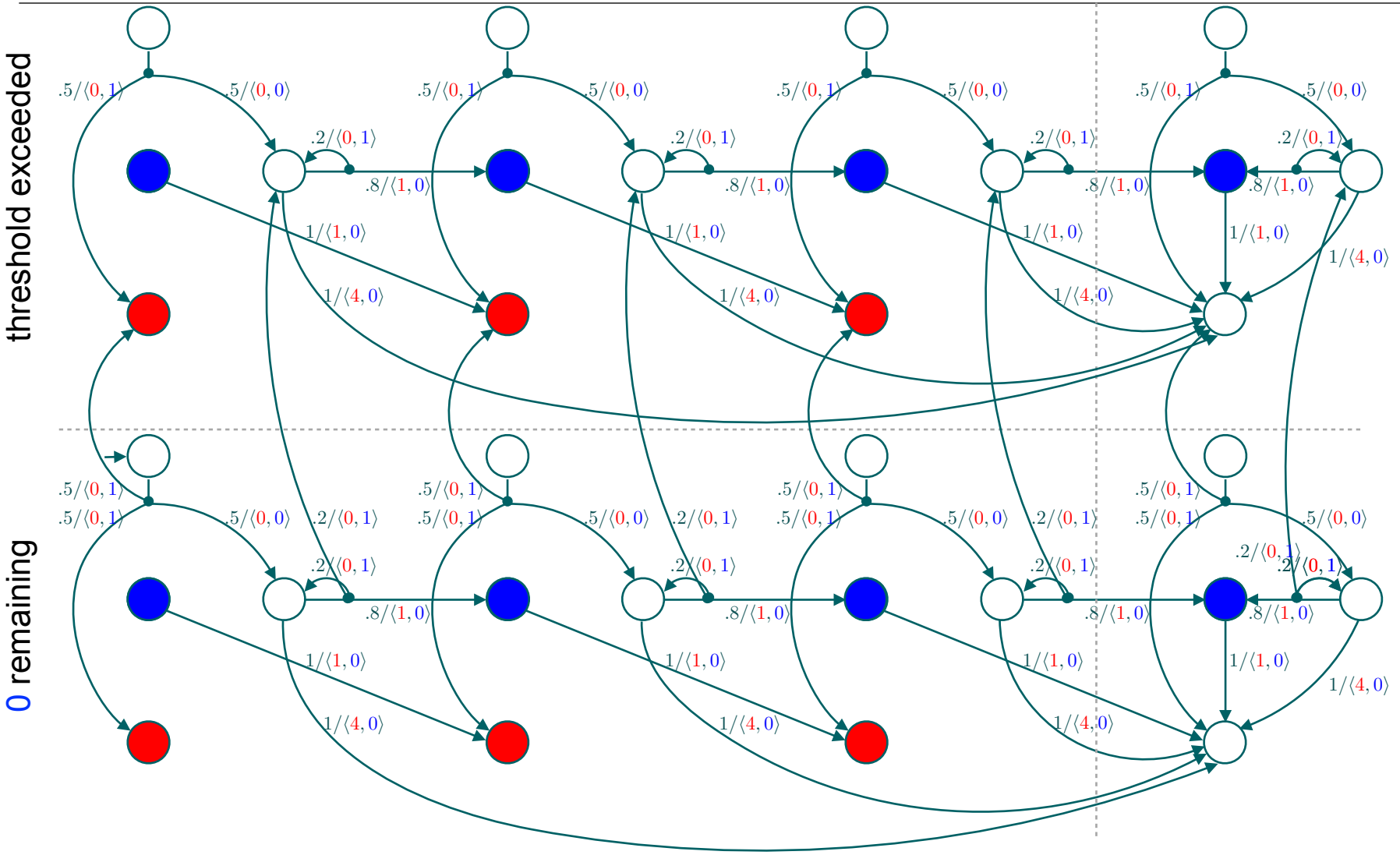
$\Pr_{max}(\Diamond^{\leq b} \bullet)$ for original MDP coincides with $\Pr_{max}(\Diamond \bullet)$ for unfolded MDP

Unfolding for *multi* [$\Pr_{max}(\Diamond^{\leq 2}$  $)$, $\Pr_{max}(\Diamond^{\leq 0}$  $)$]

- **Idea:** Unfold in multiple dimensions



Unfolding for *multi* [$\Pr_{max}(\Diamond^{\leq 2}$ ●$)$, $\Pr_{max}(\Diamond^{\leq 0}$ ●$)$]

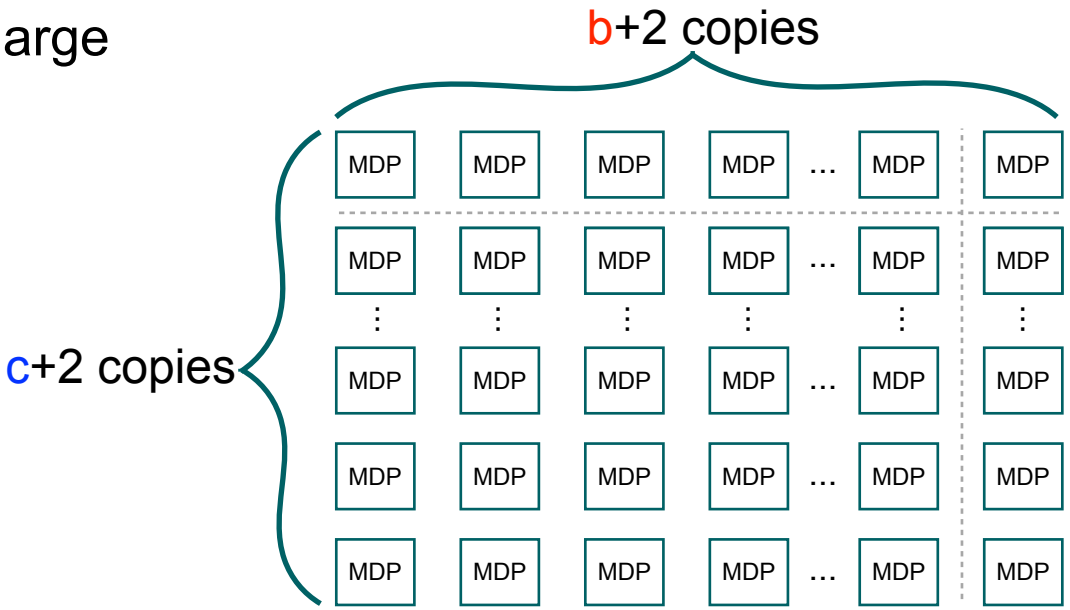


Dynamic Programming

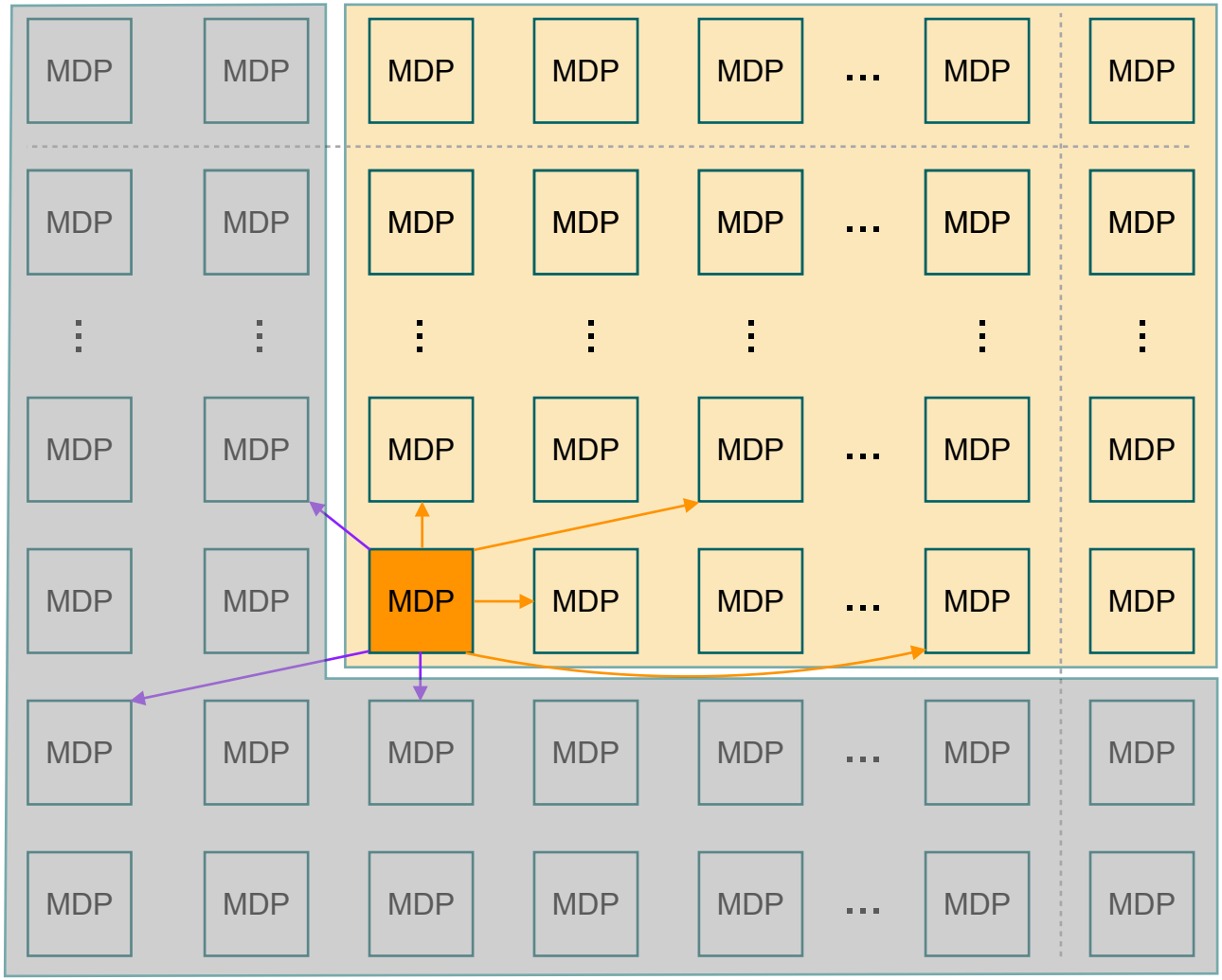
multi[$\Pr_{max}(\Diamond^{\leq b}$  $)$, $\Pr_{max}(\Diamond^{\leq c}$  $)$] for original MDP

coincides with multi[$\Pr_{max}(\Diamond$  $)$, $\Pr_{max}(\Diamond$  $)$] for unfolded MDP

- Obtain Pareto curve with existing algorithms for unbounded reachability
- Unfolding is large



Dynamic Programming



Dynamic Programming

- Analyse 

copies sequentially (one after the other)
- No need to consider the entire unfolding at once
- Efficient implementation as 

copies are structurally similar
- Naturally extends approaches from single cost-bounded analysis

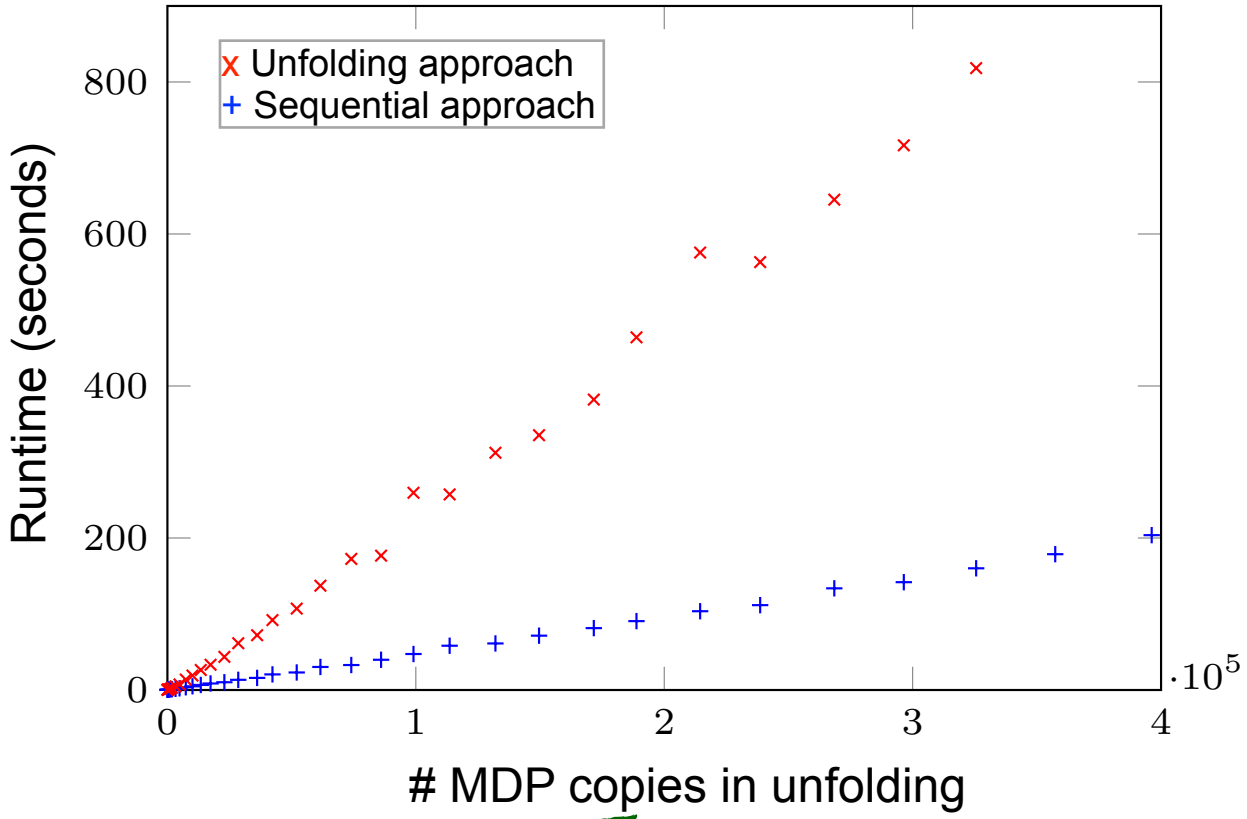
  [Hahn and Hartmanns, SETTA 2016], [Klein *et al.*, STTT 2017]

Experiments

Mars Rover

- Schedule tasks the rover should perform within a time and energy limit

- 16 states
- 2 objectives
- 3 cost structures



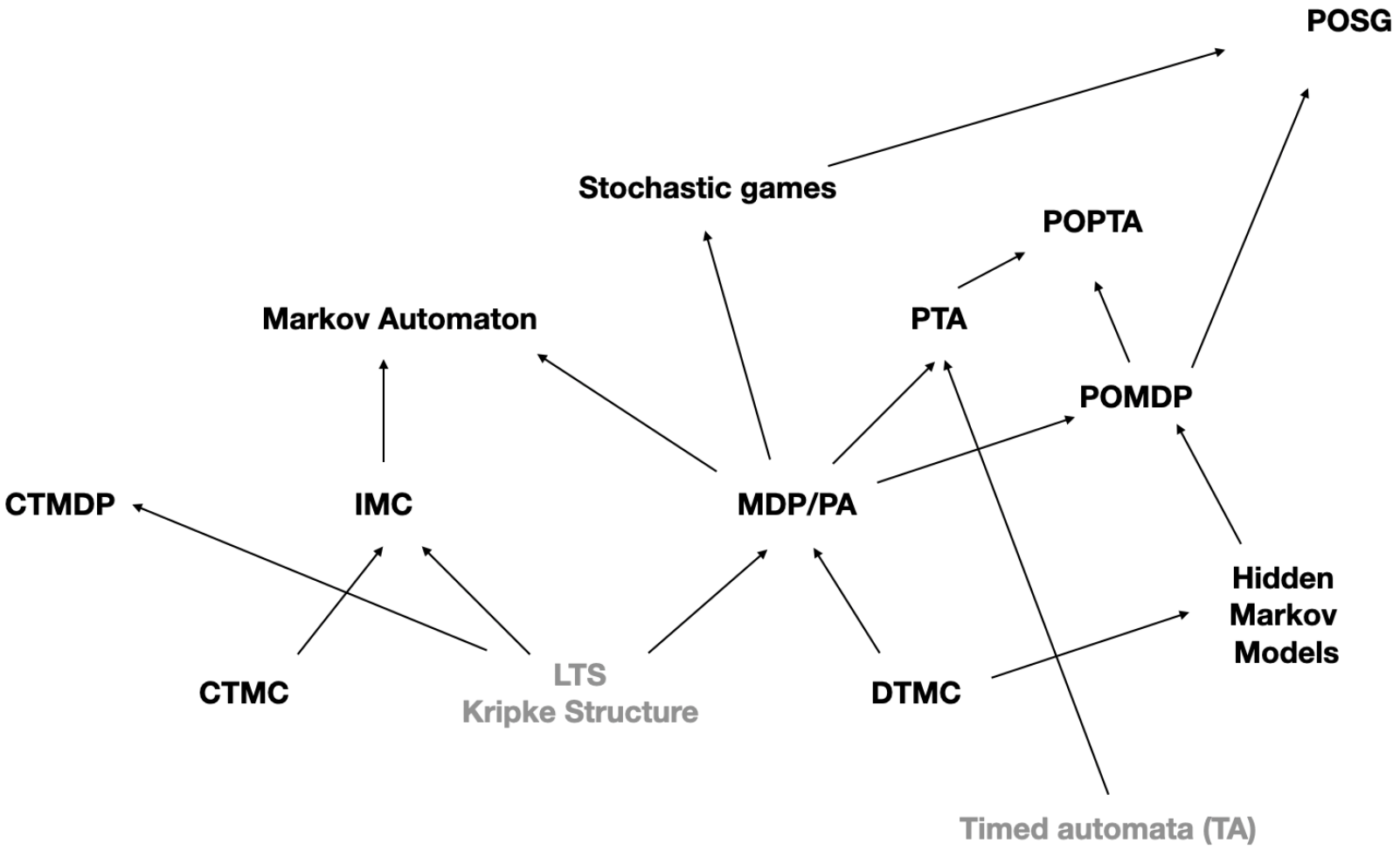
reflects magnitude of cost bounds

Verification Times Multiple Cost-Objectives

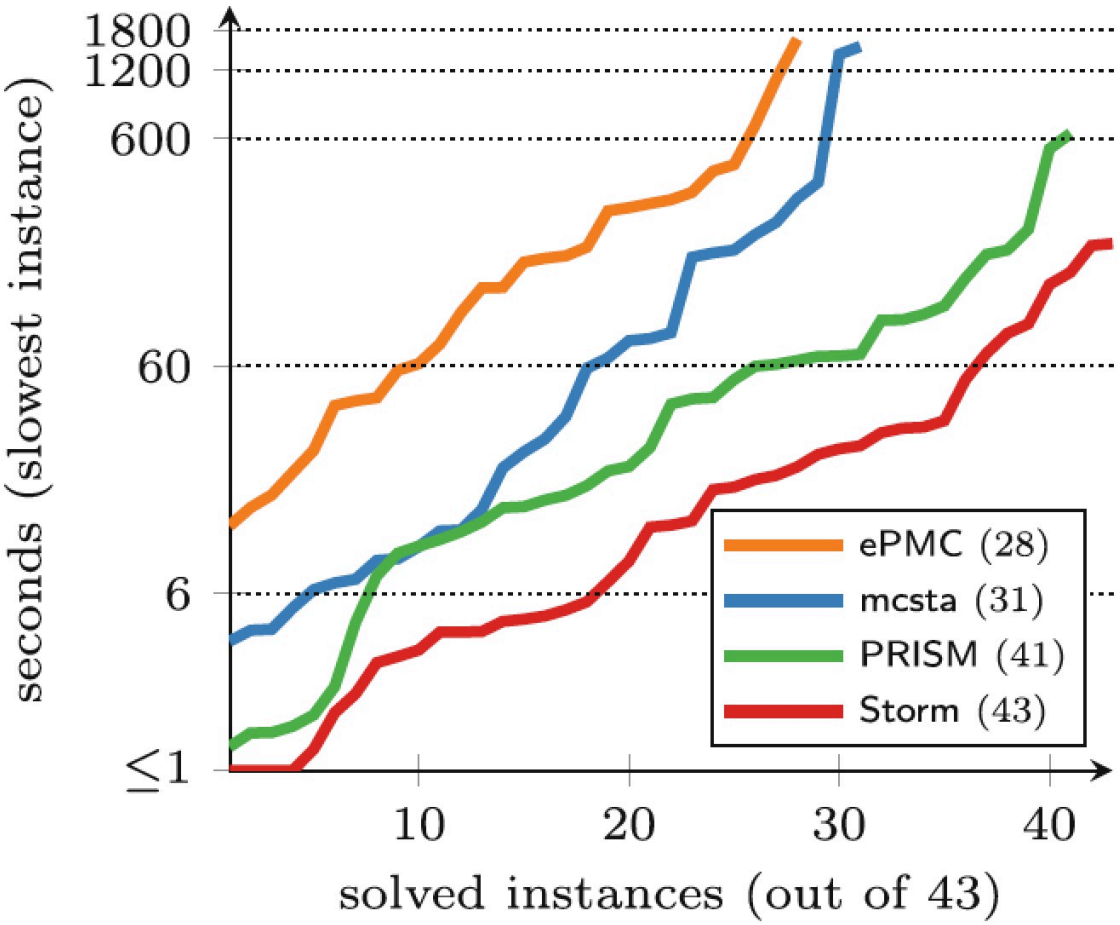
Benchmark instance							Interval It.		Policy It.	
Case Study	$\|S\|$	$\|T\|$	ℓ-r-m	$\|\mathfrak{E}\|$	#w	$\|S_{unf}\|$	UNF-sp	SEQ	UNF-sp	SEQ
Service	$8 \cdot 10^4$	$2 \cdot 10^5$	2–1–2	162	34	$6 \cdot 10^6$	1918	**543**	TO	**4679**
JobSched2	349	660	2–4–4	$4 \cdot 10^4$	2	$1 \cdot 10^5$	**3**	54	**15**	183
JobSched3	4584	$1 \cdot 10^5$	2–4–4	$1 \cdot 10^6$	35	$2 \cdot 10^6$	**96**	TO	**6239**	TO
JobSched5	$1 \cdot 10^6$	$4 \cdot 10^6$	2–4–4	$3 \cdot 10^5$	?	?	TO	TO	TO	TO
FireWire	776	1411	2–2–2	6024	3	$7 \cdot 10^5$	32	**17**	TO	**1159**
FireWire	776	1411	2–2–2	$1 \cdot 10^5$	2	$1 \cdot 10^7$	863	**225**	TO	TO
Resources	94	326	2–3–4	$2 \cdot 10^5$	3	$6 \cdot 10^5$	25	**16**	2047	**52**
Resources	94	326	2–3–4	$1 \cdot 10^8$	?	?	TO	TO	TO	TO
Rover	16	30	2–3–3	$9 \cdot 10^5$	7	$1 \cdot 10^6$	177	**39**	5817	**3328**
Rover	16	30	2–3–3	$1 \cdot 10^8$	7	$2 \cdot 10^8$	TO	**5785**	TO	TO
UAV	$1 \cdot 10^5$	$6 \cdot 10^4$	2–1–2	52	18	$4 \cdot 10^4$	**2**	24	**102**	1098
UAV	$1 \cdot 10^5$	$6 \cdot 10^4$	2–1–2	102	22	$4 \cdot 10^5$	70	**39**	**2282**	3062
Wlan3	$1 \cdot 10^5$	$2 \cdot 10^5$	3–1–2	82	68	$3 \cdot 10^6$	5239	**2231**	TO	TO
Wlan3	$1 \cdot 10^5$	$2 \cdot 10^5$	3–1–2	202	4	$1 \cdot 10^7$	1769	**185**	TO	TO
Wlan6	$5 \cdot 10^6$	$1 \cdot 10^7$	3–1–2	82	?	$2 \cdot 10^7$	TO	TO	TO	TO

l = # queries, r = # different cost structures, m = # bounds

Beyond Markov Chains



Efficiency



https://www.stormchecker.org

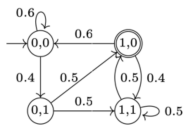
Tutorial Overview

1. 

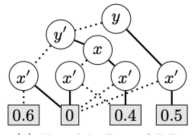
Fundamentals of
Probabilistic Model Checking

2. 

Probabilistic Model Checking with Storm:
Hands-on Slides

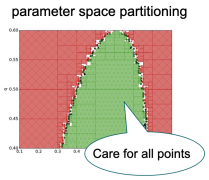
3. 

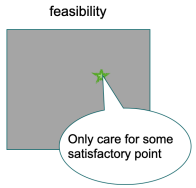
Example Markov chain



Its MTBDD (x, x') to (y, y')

Automated Symbolic Reasoning

4. 



Parameter Synthesis in Markov Models

Probabilistic Model Checking with Storm

Sebastian Junges, **Joost-Pieter Katoen**

Radboud Universiteit

Software Modeling and Verification Chair

RWTH AACHEN UNIVERSITY

UNIVERSITY OF TWENTE.

Hands-on presentation

- See

  https://github.com/moves-rwth/stormpyter/tree/master/tutorial_uai

  for the material. The material includes information how-to run the interactive slides on your on own machine! Below, you can find the first few slides.

Introduction to Storm

Sebastian Junges and Joost-Pieter Katoen

using material by the Storm Developers

www.stormchecker.org

Press *spacebar* to navigate

Storm

A modern probabilistic model checker

- **State-of-the-art**: best performance at QComp 2020
- **Modular**: dedicated solvers for each task, interchangable libraries
- Written in **C++**, **Python interface** via stormpy
- **Open-source**, developed since 2012, over 230,000 lines of code

Getting Storm

- Native support for *Linux* and *macOS* (homebrew formula)
- Virtual machine and Docker containers (also for *Windows*)

Getting Storm for this presentation

We use a Docker container based on Jupyter Notebook throughout this presentation.

Installation steps:

1. Install Docker for your OS
2. Download (>1 GB) and start the container:

```
docker run -it -p 8080:8080 --name stormpyter sjunges/stormpyter:uai22
```

3. Open the Jupyter website indicated in the command line: 127.0.0.1:8080/...
4. Open file **tutorial.ipynb**
5. The presentation should start automatically

Hands-on presentation

- The PDF includes an non-interactive version.
  The interactive version can be found at

  https://github.com/moves-rwth/stormpyter/tree/master/tutorial_uai

Introduction to Storm

Sebastian Junges and Joost-Pieter Katoen

using material by the Storm Developers

www.stormchecker.org

Press *spacebar* to navigate

Storm

A modern probabilistic model checker

- **State-of-the-art**: best performance at QComp 2020
- **Modular**: dedicated solvers for each task, interchangable libraries
- Written in **C++**, **Python interface** via stormpy
- **Open-source**, developed since 2012, over 230,000 lines of code

Getting Storm

- Native support for *Linux* and *macOS* (homebrew formula)
- Virtual machine and Docker containers (also for *Windows*)

Getting Storm for this presentation

We use a Docker container based on Jupyter Notebook throughout this presentation.

Installation steps:

1. Install Docker for your OS
2. Download (>1 GB) and start the container:

   ```
   docker run -it -p 8080:8080 --name stormpyter sjunges/stormpyter:uai22
   ```

3. Open the Jupyter website indicated in the command line: 127.0.0.1:8080/...
4. Open file **tutorial.ipynb**
5. The presentation should start automatically

Hands-on presentation

- This is an interactive presentation. You can **execute all commands by yourself**!
- Navigate with *spacebar* and *shift+spacebar*
- All interactive commands can be executed with *shift+enter*

- Switch between presentation and notebook with *alt+r*

Example

In [4]:
```
%alias storm "./storm"
```

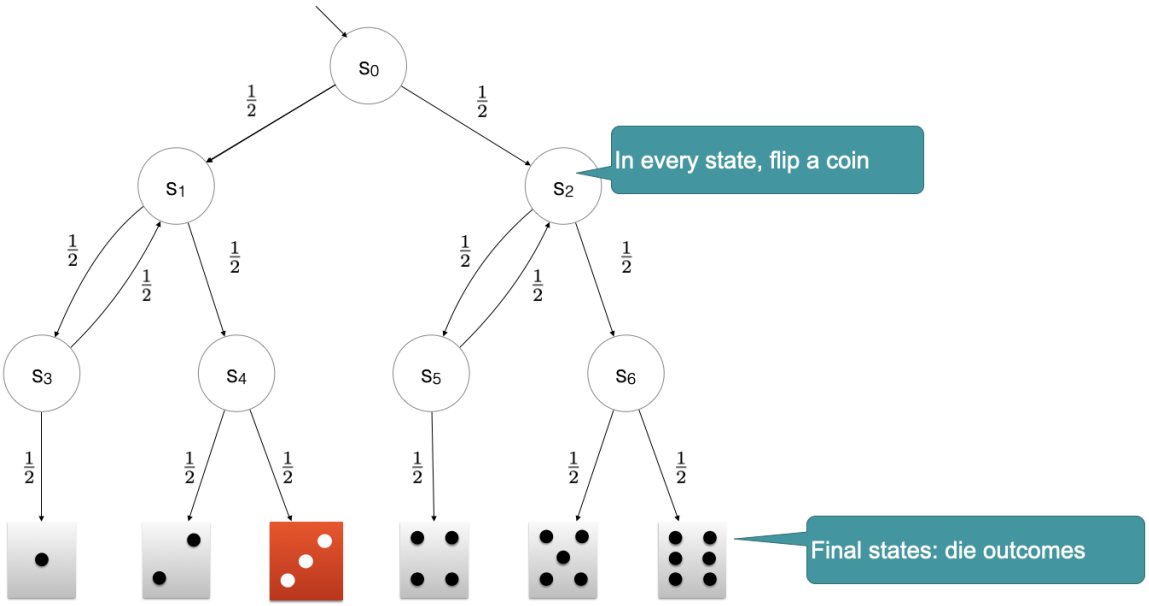
In [6]:
```
storm --version
```

```
Storm 1.7.1 (dev)

Date: Mon Aug  1 10:00:01 2022
Command line arguments: --version
Current working directory: /Users/junges/stormpyter/tutorial_uai

Version 1.7.1 (dev) (+ 6 commits) build from revision 704e8066486de8cef0362cf98a36404b68
d7c187 (clean)
Compiled on Darwin 21.6.0 using AppleClang 13.1.6.13160021 with flags ' -stdlib=libc++ -
ftemplate-depth=1024 -O3 -DNDEBUG -fno-stack-check -march=native -fomit-frame-pointer'
Linked with GNU Linear Programming Kit v5.0.
Linked with Gurobi Optimizer v9.5.1.
Linked with Microsoft Z3 Optimizer v4.9 Build 1 Rev 0.
Linked with CArL.
```

You should see the Storm version information.

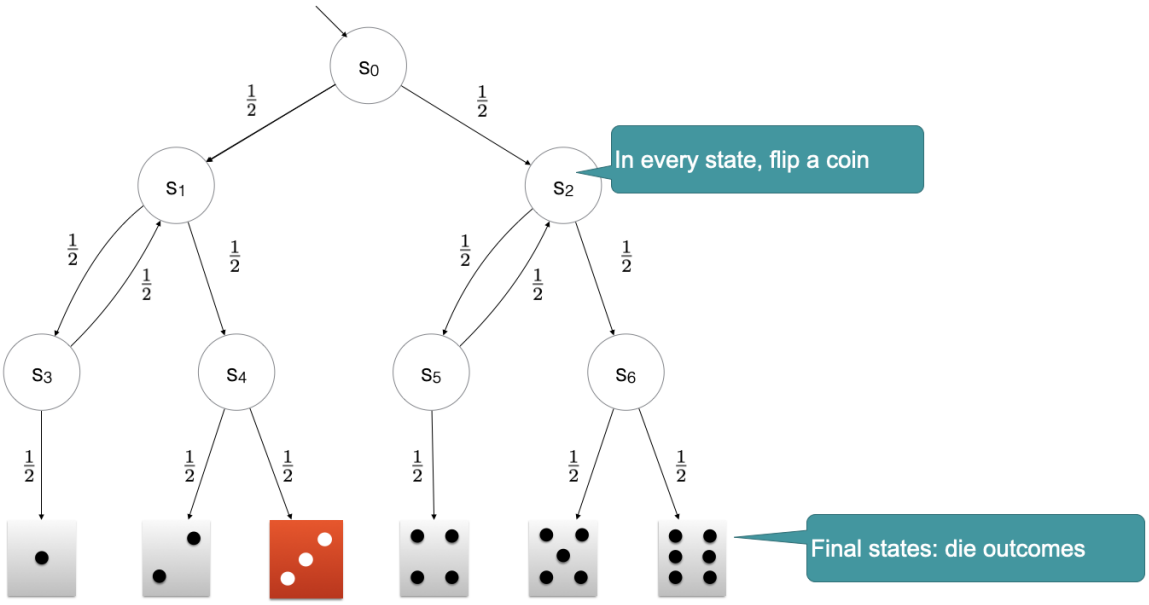
You can even **change the code**! Try adding the flag `--version`.

Knuth-Yao Die



This models a dice roll with a fair coin

Knuth-Yao Die



In [3]:
```
!tail -n+3 examples/kydie.drn | head -n 25
```

```
@type: DTMC
@parameters

@reward_models

@nr_states
13
@nr_choices
13
@model
state 0 init
        action 0
                1 : 0.5
                2 : 0.5
state 1
        action 0
                3 : 0.5
                4 : 0.5
state 2
        action 0
                5 : 0.5
                6 : 0.5
state 3
        action 0
                1 : 0.5
```

Running Storm on KY-Die

In [4]:
```
storm --explicit-drn examples/kydie.drn --buildfull
```

```
Storm 1.7.1 (dev)

Date: Sun Jul 31 21:59:49 2022
Command line arguments: --explicit-drn examples/kydie.drn --buildfull
Current working directory: /Users/junges/stormpyter/tutorial_uai

Time for model construction: 0.001s.

-------------------------------------------------------------
Model type:     DTMC (sparse)
```

```
States:          13
Transitions:     20
Reward Models:   none
State Labels:    8 labels
   * init -> 1 item(s)
   * out5 -> 1 item(s)
   * done -> 6 item(s)
   * out3 -> 1 item(s)
   * out1 -> 1 item(s)
   * out6 -> 1 item(s)
   * out2 -> 1 item(s)
   * out4 -> 1 item(s)
Choice Labels:   none
---------------------------------------------------------------
```

Computing Reachability Probabilities

In [86]:
```
storm --explicit-drn examples/kydie.drn --prop 'P=? [F "out3" ]' | tail -n+7
```

```
Time for model construction: 0.000s.

---------------------------------------------------------------
Model type:      DTMC (sparse)
States:          13
Transitions:     20
Reward Models:   none
State Labels:    8 labels
   * init -> 1 item(s)
   * out5 -> 1 item(s)
   * done -> 6 item(s)
   * out3 -> 1 item(s)
   * out1 -> 1 item(s)
   * out6 -> 1 item(s)
   * out2 -> 1 item(s)
   * out4 -> 1 item(s)
Choice Labels:   none
---------------------------------------------------------------

Model checking property "1": P=? [F "out3"] ...
Result (for initial states): 0.1666666667
Time for model checking: 0.000s.
```

Computing Exected Time (steps)

In [88]:
```
storm --explicit-drn examples/kydie.drn --prop 'T=? [F "done" ]' | tail -n+7
```

```
Time for model construction: 0.000s.

---------------------------------------------------------------
Model type:      DTMC (sparse)
States:          13
Transitions:     20
Reward Models:   none
State Labels:    8 labels
   * init -> 1 item(s)
   * out5 -> 1 item(s)
   * done -> 6 item(s)
   * out3 -> 1 item(s)
   * out1 -> 1 item(s)
   * out6 -> 1 item(s)
   * out2 -> 1 item(s)
   * out4 -> 1 item(s)
Choice Labels:   none
---------------------------------------------------------------
```

```
Model checking property "1": T[exp]=? [F "done"] ...
Result (for initial states): 3.666666667
Time for model checking: 0.000s.
```

Input formats

- Storm supports a variety of input formats:
  - Prism language
  - Jani modelling language
  - Explicit format
  - Generalized stochastic Petri nets
  - Domain specific languages, e.g., Dynamic Fault Trees

Intro to Prism format

KY-die with Prism

Naive encoding:

- Variable *s* encodes the state
- Describe an update for every state
- Single module

In [7]:
```
!tail -n20 examples/kydie-naive.prism | head -n 16
```

```
module main
    s : [0..12] init 0;
    [] s=0 -> 0.5:(s'=1) + 0.5:(s'=2);
    [] s=1 -> 0.5:(s'=3) + 0.5:(s'=4);
    [] s=2 -> 0.5:(s'=5) + 0.5:(s'=6);
    [] s=3 -> 0.5:(s'=1) + 0.5:(s'=7);
    [] s=4 -> 0.5:(s'=8) + 0.5:(s'=9);
    [] s=5 -> 0.5:(s'=2) + 0.5:(s'=10);
    [] s=6 -> 0.5:(s'=11) + 0.5:(s'=12);
    [] s>6 -> 1:(s'=s);
endmodule

label "out1" = s=7;
label "out2" = s=8;
label "out3" = s=9;
label "out4" = s=10;
```

Single-Module Prism

- A module contains declarations of (local) variables and commands.
- Commands

$$[action]\ \text{guard}\ \texttt{->}\ \text{probability}\texttt{:}\text{update} + \text{probability}\texttt{:}\text{update} + \ldots$$

  - *action*: Synchronization label (may be empty)
  - *guard*: Boolean combination of inequalities involving local and global variables

- *probability*: expression between 0 and 1. The probabilities of each command have to sum up to 1.
- *update*: Assigns new values to (a subset of) the variables.

$$\mathbf{var}' = f(\mathbf{vars})$$

- Every action refers to an action in the MDP

Gridworld with Prism

- Robot in an N by N grid
- Own position given by x,y
- Up to 4 actions

In [8]:
```
!cat examples/grid.prism
```

```
mdp

const int N;

module main
   x : [0..N] init 3;
   y : [0..N] init 3;

   [north] x > 1 -> 0.9:(x'=x-1) + 0.1:(x'=x);
   [south] x < N -> 0.9:(x'=x+1) + 0.1:(x'=x);
   [west] y > 1 -> 0.8:(y'=y-1) + 0.2:(y'=y);
   [east] y < N -> 0.8:(y'=y+1) + 0.2:(y'=y);
endmodule
```

Gridworld with Prism

- Robot in an N by N grid
- Own position given by x,y
- Up to 4 actions

In [9]:
```
storm --prism examples/grid.prism -const N=6 | tail -n+6 | head -n 10
```

```
Time for model input parsing: 0.001s.

Time for model construction: 0.048s.

-------------------------------------------------------------
Model type:     MDP (sparse)
States:         36
Transitions:    240
Choices:        120
```

In [10]:
```
storm --prism examples/grid.prism -const N=60 --prop "Tmin=? [F y=N & x=N]" | tail -n 3
```

```
Model checking property "1": T[exp]min=? [F ((y = 60) & (x = 60))] ...
Result (for initial states): 134.583334
Time for model checking: 0.007s.
```

Advanced properties

We consider a selection of slightly more advanced queries

We therefore also extend the MDP description...

In [11]:
```
!tail -n 10 examples/grid_complete.prism
```

```
label "station" = x=4 & y=1;
label "castle" = x=1 & y=N-1;
formula grass =  y = 3 | x=2;

rewards "movementcost"
    grass : 2;
    !grass : 1;
endrewards
```

Cost-bounded reachability

- Simple extension to reachability
- Storm can handle multi-reward-bounded formulas and compute quantiles

In [12]:
```
storm --prism examples/grid_complete.prism -const N=30 --prop "Pmax=? [F{\"movementcost\
```

```
Time for model construction: 0.023s.

-------------------------------------------------------------
Model type:     MDP (sparse)
States:         900
Transitions:    6955
Choices:        3478
Reward Models:  movementcost
State Labels:   3 labels
   * deadlock -> 0 item(s)
   * init -> 1 item(s)
   * castle -> 1 item(s)
Choice Labels:  none
-------------------------------------------------------------

Model checking property "1": Pmax=? [true Urew{"movementcost"}<=40 "castle"] ...
Result (for initial states): 0.9060504552
Time for model checking: 0.012s.
```

LTL properties

In [13]:
```
storm --prism examples/grid_complete.prism -const N=6 --prop "Pmax=? [GF \"station\" & G
```

```
Model checking property "1": Pmax=? [G F ("station" & (G F "castle"))] ...
Result (for initial states): 1
Time for model checking: 0.014s.
```

In [14]:
```
storm --prism examples/grid_complete.prism -const N=6 --prop "Pmax=? [F<=7 \"station\" &
```

```
Model checking property "1": Pmax=? [true U<=7 ("station" & (true U<=7 "castle"))] ...
Result (for initial states): 0.4255827145
Time for model checking: 0.017s.
```

In [15]:
```
storm --prism examples/grid_complete.prism -const N=6 --prop "Pmax=? [F<=7 \"station\"];
```

```
Model checking property "1": Pmax=? [true U<=7 "station"] ...
Result (for initial states): 0.9998235
Time for model checking: 0.000s.
```

```
Model checking property "2": Pmax=? [true U<=7 "castle"] ...
Result (for initial states): 0.966656
Time for model checking: 0.000s.
```

Extracting Policies ('Schedulers')

In [16]:
```
storm --prism examples/grid.prism -const N=6 --prop "Tmin=? [F x=3 & y > 4]" --exportsch
```

```
Model checking property "1": T[exp]min=? [F ((x = 3) & (y > 4))] ...
Exporting scheduler ...  WARN (model-handling.h:1133): No information of state valuation
s available. The scheduler output will use internal state ids. You might be interested i
n building the model with state valuations using --buildstateval.
 WARN (model-handling.h:1137): No symbolic choice information is available. The schedule
r output will use internal choice ids. You might be interested in building the model wit
h choice labels or choice origins using --buildchoicelab or --buildchoiceorig.
 WARN (model-handling.h:1140): Only partial choice information is available. You might w
ant to build the model with choice origins using --buildchoicelab or --buildchoiceorig.
Write to file plain.sched.
Result (for initial states): 2.5
Time for model checking: 0.000s.
```

In [17]:
```
!head -n10 plain.sched
```

```
_____________________________________________________________________
Fully defined memoryless deterministic scheduler:
model state:    choice(s)
           0    3
           1    3
           2    3
           3    3
           4    3
           5    2
           6    1
```

Extracting Policies

In [18]:
```
storm --prism examples/grid.prism -const N=6 --prop "Tmin=? [F x=3 & y > 4]" --exportsch
```

```
Model checking property "1": T[exp]min=? [F ((x = 3) & (y > 4))] ...
Exporting scheduler ... Write to file high.sched.
Result (for initial states): 2.5
Time for model checking: 0.000s.
```

In [19]:
```
!head -n10 high.sched
```

```
_____________________________________________________________________
Fully defined memoryless deterministic scheduler:
      model state:    choice(s)
    0: [x=3     & y=3]    3 {east}
    1: [x=2     & y=3]    3 {east}
    2: [x=4     & y=3]    3 {east}
    3: [x=3     & y=2]    3 {east}
    4: [x=3     & y=4]    3 {east}
    5: [x=1     & y=3]    2 {east}
    6: [x=2     & y=2]    1 {south}
```

Methods and Engines

Computing exact values

- Floating point arithmetic and convergence criteria might not be precise
  ➜ use exact (rational) numbers and sound algorithms off-the-shelf
- Drawback: performance decreases

In [20]:
```
storm --explicit-drn examples/kydie.drn --prop 'P=? [F "out1"]' | tail -n2
```

```
Result (for initial states): 0.1666666667
Time for model checking: 0.000s.
```

In [21]:
```
storm --explicit-drn examples/kydie.drn --prop 'P=? [F "out1"]' --exact | tail -n2
```

```
Result (for initial states): 1/6 (approx. 0.1666666667)
Time for model checking: 0.000s.
```

Different methods

- Variations of value iteration with stronger guarantees
- Policy iteration, hybrid methods
- Abstraction-based approaches

In [22]:
```
storm --prism examples/grid_complete.prism -const N=40 --prop "Tmin=? [F x=3 & y>4]" --m
```

```
Model checking property "1": T[exp]min=? [F ((x = 3) & (y > 4))] ...
Result (for initial states): 7.222222222
Time for model checking: 0.488s.
```

In [23]:
```
storm --prism examples/grid_complete.prism -const N=40 --prop "Tmin=? [F x=3 & y>4]" --m
```

```
Model checking property "1": T[exp]min=? [F ((x = 3) & (y > 4))] ...
Result (for initial states): 7.222222222
Time for model checking: 0.002s.
```

Engines in Storm

Storm supports different model representations:

- **Sparse matrix**:
  - Model building is time and memory intensive
  - Fast numerical computations
- **Binary Decision Diagrams (BDD)**:
  - Fast and memory efficient model building if model is structured
  - Slower numerical computations
- **Hybrid** approach:
  - Build with BDD, use sparse matrix for numerical computations
- ...

Engines in Storm

In [13]:
```
storm --prism examples/grid.prism -const N=800 --prop 'Tmin=? [F x=3 & y > 4]' -e dd --m
```

```
Model checking property "1": T[exp]min=? [F ((x = 3) & (y > 4))] ...
Result (for initial states): 2.5
Time for model checking: 0.452s.
```

In [25]:
```
storm --prism examples/grid.prism -const N=800 --prop 'Tmin=? [F x=3 & y > 4]' -e sparse
```

```
Model checking property "1": T[exp]min=? [F ((x = 3) & (y > 4))] ...
Result (for initial states): 2.5
Time for model checking: 10.921s.
```

Bisimulation minimization

- Reduces state-space size without loosing information
- Exploits symmetries

In [26]:
```
storm --bisimulation --prism examples/phil-nofair3.nm --prop 'filter(forall, P>=1 [F "ea
```

```
Time for model construction: 0.025s.

-------------------------------------------------------------
Model type:     MDP (sparse)
States:         956
Transitions:    3048
Choices:        2694
Reward Models:  none
State Labels:   4 labels
   * deadlock -> 0 item(s)
   * init -> 1 item(s)
   * hungry -> 922 item(s)
   * eat -> 240 item(s)
Choice Labels:  none
-------------------------------------------------------------

Time for model preprocessing: 0.008s.

-------------------------------------------------------------
Model type:     MDP (sparse)
States:         172
Transitions:    517
Choices:        461
Reward Models:  none
State Labels:   3 labels
   * init -> 1 item(s)
   * hungry -> 164 item(s)
   * eat -> 44 item(s)
Choice Labels:  none
-------------------------------------------------------------
```

Summary

- Basic introduction to **Storm command-line interface**
- Storm supports broad range of **models and properties** given in high-level description languages
- Different **engines** allow tailored analysis per model and property

Next:

Model checking **in the loop** using Stormpy

Intro to Stormpy

- Basic model checking
- Tradeoff analysis

- Robustness analysis
- Simulation engines

In [14]:
```
import stormpy
import stormpy.info

print("Stormpy version: " + stormpy.__version__ + " using Storm in version: " + stormpy.
```

```
Stormpy version: 1.7.0 using Storm in version: 1.7.1
```

Basic Model Checking

- Parse input-description and translate into MDP

In [28]:
```
orig_program = stormpy.parse_prism_program("examples/grid_slip_forward.prism")
program = orig_program.define_constants(stormpy.parse_constants_string(orig_program.expr
```

In [29]:
```
options = stormpy.BuilderOptions(True, True)
options.set_build_state_valuations()
options.set_build_choice_labels()
model = stormpy.build_sparse_model_with_options(program, options)
print("Number of states: {}".format(model.nr_states))
print("Labels: {}".format(model.labeling.get_labels()))
```

```
Number of states: 48
Number of transitions: 258
Labels: {'castle', 'deadlock', 'init', 'station'}
```

- Run model checking query

In [30]:
```
properties = stormpy.parse_properties("Pmax=? [F<=8 \"station\"]")
result = stormpy.model_checking(model, properties[0])
print(result.at(model.initial_states[0]))
```

```
0.86810446
```

Tradeoff analysis

- Find values that can be achieved by a single policy

In [31]:
```
from stormpy.utility.multiobjective_plotting import prepare_multiobjective_result_for_pl
import matplotlib.pyplot as plt

properties = stormpy.parse_properties_for_prism_program("multi(Pmax=? [ F<=11 \"castle\"
result = stormpy.model_checking(model,properties[0])
print(result.get_underapproximation().vertices)
```

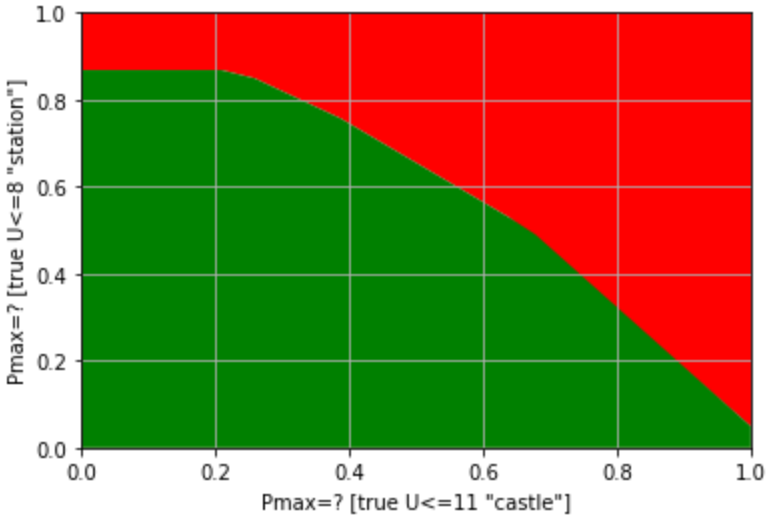
```
[[0.2095549336299997, 0.86810446], [0.22331658280000125, 0.8629031799999994], [0.3872408
782000002, 0.7571695599999997], [0.6394090000000007, 0.5291019999999993], [0.25903569549
99999, 0.8492238999999998], [1.0, 0.04899999999999984], [0.6785109999999998, 0.490000000
00000016]]
```

- Plot the outcome

In [32]:
```
lower_left = [0,0]
upper_right = [1,1]
formula = properties[0].raw_formula
```

```
fig, ax = plt.subplots()

underapprox_points, overapprox_points = prepare_multiobjective_result_for_plotting(resul
plot_convex_pareto_curve_demo(ax, underapprox_points, overapprox_points, lower_left, upp
ax.set_xlabel(formula.subformulas[0])
ax.set_ylabel(formula.subformulas[1])
```

Out[32]:
```
Text(0, 0.5, 'Pmax=? [true U<=8 "station"]')
```



Robustness with Parametric Checking

Step 1: Compute an optimal policy

In [33]:
```
properties = stormpy.parse_properties("R{\"movecost\"}min=? [ F \"station\"]")
result = stormpy.model_checking(model, properties[0], extract_scheduler=True)
scheduler = result.scheduler
print(result.at(model.initial_states[0]))
```

```
5.766754302112218
```

Step 2: Set-up an MDP with parametric slipiness

In [34]:
```
program2 = orig_program.define_constants(stormpy.parse_constants_string(orig_program.exp
parametric_model = stormpy.build_sparse_parametric_model_with_options(program2, options)
print("Number of states: {}".format(model.nr_states))
print("Number of transitions: {}".format(model.nr_transitions))
print("Labels: {}".format(model.labeling.get_labels()))
```

```
Number of states: 48
Number of transitions: 258
Labels: {'castle', 'deadlock', 'init', 'station'}
```

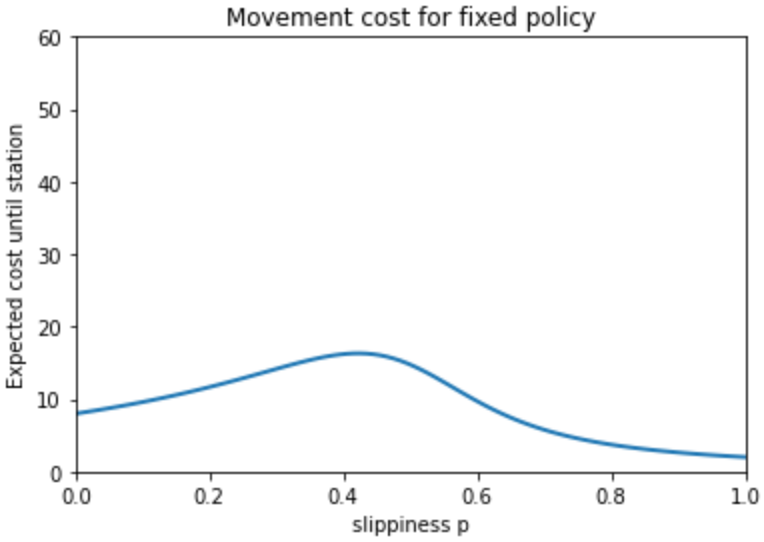
Step 3: Apply the policy on this parametetric MDP and evaluate performance

In [35]:
```
induced_model = parametric_model.apply_scheduler(scheduler.cast_to_parametric_datatype()
```

In [36]:
```
result = stormpy.model_checking(induced_model, properties[0]).at(induced_model.initial_s
print(result)
```

```
(p^6+62*p^2+(-58)*p^3+30*p^4+(-7)*p^5+(-34)*p+8)/((2*p^2+(-2)*p+1) * (2*p^4+6*p^2+(-4)*p
^3+(-4)*p+1))
```

In [37]:
```
import numpy as np
plt.xlabel('slippiness p')
plt.ylabel('Expected cost until station')
plt.title("Movement cost for fixed policy")

p = np.linspace(0.001, 0.999, 100)
plt.axis((0,1,0,60))
plot_func = eval(str(result).replace("^", "**"))
plt.plot(p, plot_func, linewidth=2);
```



Step 4: Computing the regret

- First, we find the best-case slipiness for p in [0.1, 0.2]
- We also compute the associated expected movement cost

In [51]:
```
import stormpy.pars as spp
stormpy.set_loglevel_error()
env = stormpy.Environment()
checker = spp.create_region_checker(env, parametric_model, properties[0].raw_formula)
region_string = "0.1<=p<=0.2"
region = spp.ParameterRegion.create_from_string(region_string, parametric_model.collect_
precision = stormpy.RationalRF(0.01)
best_case_result = checker.compute_extremum(env, region, stormpy.OptimizationDirection.M
print(str(best_case_result[0]) + " or approx " + str(float(best_case_result[0].constant_
```

```
4938860038855867/562949953421312 or approx 8.773177808862206
```

- We can now compare this value to the value by the scheduler

In [41]:
```
print(float(result.evaluate(best_case_result[1])))
```

```
9.713210126685812
```

Simulation

Storm supports gym-inspired bindings for simulations

Variant 1: High-level simulators

In [66]:
```
import stormpy.simulator as ssim
```

```
import random
random.seed(3)
simulator = ssim.create_simulator(program, seed=42)
simulator.set_action_mode(ssim.SimulatorActionMode.GLOBAL_NAMES)

paths = []
for m in range(3):
    path = []
    state, reward, labels = simulator.restart()
    path = [f"({state['x']},{state['y']})"]
    for n in range(6):
        actions = simulator.available_actions()
        select_action = random.randint(0,len(actions)-1)
        path.append(f"--{actions[select_action]}-->")
        state, reward, labels = simulator.step(actions[select_action])
        path.append(f"({state['x']},{state['y']})")
        if simulator.is_done():
            break
    paths.append(path)
for path in paths:
    print(" ".join(path))
```

```
(1,3) --south--> (2,3) --south--> (3,3) --west--> (3,4) --east--> (3,5) --north--> (2,5)
--north--> (1,5)
(1,3) --west--> (1,2) --west--> (1,1) --south--> (2,1) --north--> (1,1) --east--> (1,2)
--east--> (1,3)
(1,3) --east--> (1,4) --west--> (1,3) --west--> (1,4) --east--> (1,5) --south--> (2,5) -
-south--> (3,5)
```

Variant 2: Model-level simulators

In [82]:

```
import stormpy.simulator as ssim
import random
random.seed(3)
induced_model = model.apply_scheduler(scheduler)
simulator = ssim.create_simulator(model.apply_scheduler(scheduler), seed=42)
simulator.set_observation_mode(ssim.SimulatorObservationMode.PROGRAM_LEVEL)

paths = []
for m in range(3):
    path = []
    state, reward, labels = simulator.restart()
    path = [f"({state['x']},{state['y']})"]
    for n in range(6):
        path.append(f"-->")
        state, reward, labels = simulator.step()
        path.append(f"({state['x']},{state['y']})")
        if simulator.is_done():
            break
    paths.append(path)
for path in paths:
    print(" ".join(path))
```

```
(1,3) --> (1,4) --> (1,3) --> (1,4) --> (1,3) --> (1,2) --> (1,3)
(1,3) --> (1,2) --> (1,1) --> (2,1) --> (1,1) --> (2,1) --> (1,1)
(1,3) --> (1,4) --> (1,3) --> (1,4) --> (1,3) --> (1,2) --> (1,3)
```

Summary

- Support for wide variety of models and queries
- Variety of methods and engines for advanced users

- Python API (or C++ API) allows for flexible use of model checker as backend
- Actively maintained and extended

Automated Symbolic Reasoning

Sebastian Junges, **Joost-Pieter Katoen**

Radboud Universiteit

Software Modeling and Verification Chair

RWTH AACHEN UNIVERSITY

UNIVERSITY OF TWENTE.

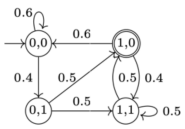
Tutorial Overview

1. 

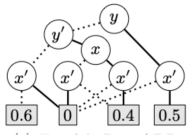
Fundamentals of
Probabilistic Model Checking

2. 

Probabilistic Model Checking with Storm:
Hands-on Slides

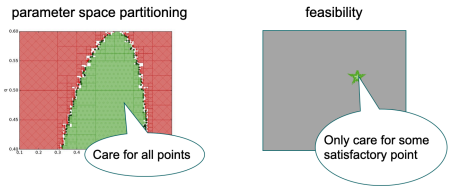
3. 

Example Markov chain



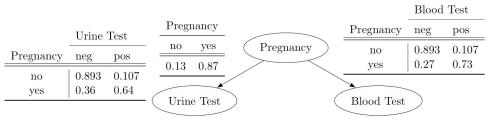
Its MTBDD (x, x') to (y, y')

Automated Symbolic Reasoning

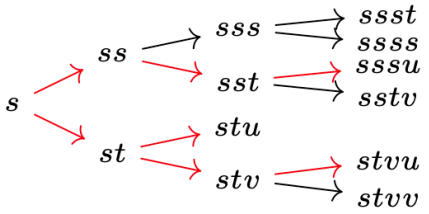
4. 

Parameter Synthesis in Markov Models

Tutorial Part 3: Automated Symbolic Reasoning

3a. 

Symbolic Probabilistic Model Checking of Bayesian Networks

3b. 

Probabilistic Model Checking by Inference

3c. $\Phi\colon [0,1]^S \to [0,1]^S\,,$

$$\Phi(F)[s] = \begin{cases} 1, & \text{if } s \in \text{Bad} \\ \sum_{s' \in S} P(s,s') \cdot F[s'], & \text{else} \end{cases}$$

Inductive Invariants (aka: 1-Induction)

3d. $\underbrace{\Phi\left(\Psi_f^{k-1}(f)\right) \sqsubseteq f}_{f \text{ is } k\text{-inductive invariant}}$ *iff* $\underbrace{\Phi\left(\Psi_f^{k-1}(f)\right) \sqsubseteq \Psi_f^{k-1}(f)}_{\Psi_f^{k-1}(f) \text{ is inductive invariant}}$

k-Induction

Bayesian Networks

Take-home message:

Inference = Computing Reachability Probabilities

Bayesian Networks

"Bayesian networks are as important to AI and machine learning as Boolean circuits are to computer science."

[Stuart Russell (Univ. of California, Berkeley), 2009]

Judea Pearl

Pregnancy	Urine Test neg	Urine Test pos
no	0.893	0.107
yes	0.36	0.64

Pregnancy no	Pregnancy yes
0.13	0.87

Pregnancy	Blood Test neg	Blood Test pos
no	0.893	0.107
yes	0.27	0.73

Turing Award 2011: "for fundamental contributions to AI through the development of a calculus for probabilistic and causal reasoning".

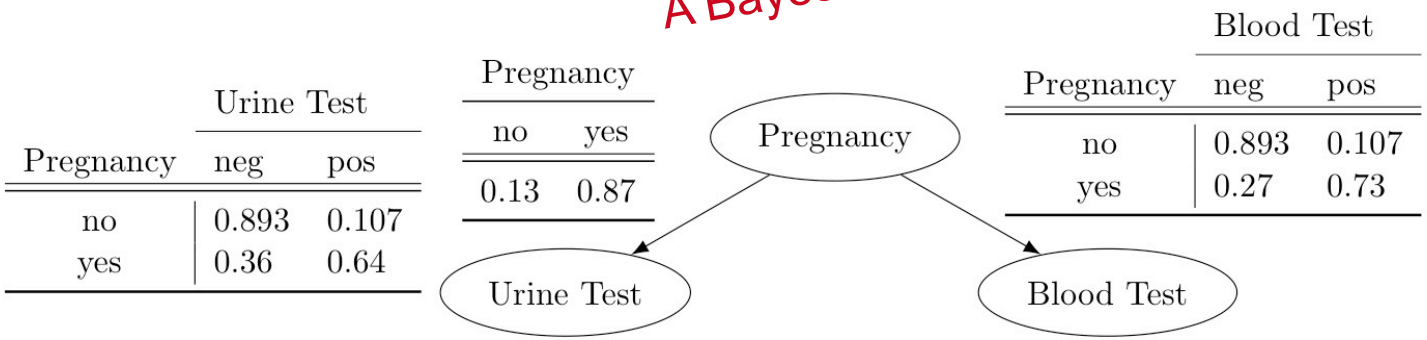
Bayesian Networks

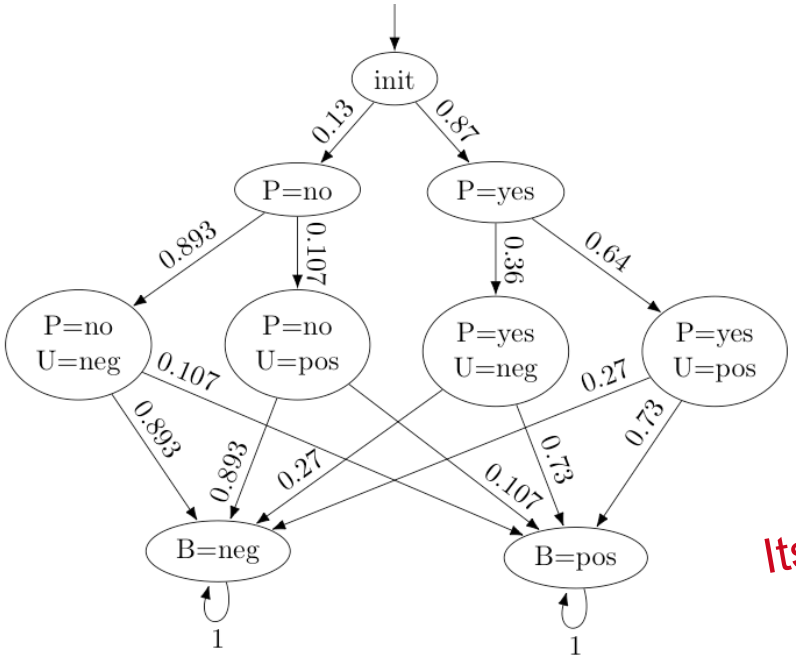
A Bayes Network

Pregnancy	Urine Test neg	Urine Test pos
no	0.893	0.107
yes	0.36	0.64

Pregnancy no	Pregnancy yes
0.13	0.87

Pregnancy	Blood Test neg	Blood Test pos
no	0.893	0.107
yes	0.27	0.73





Its Markov Chain

Inference = Reachability Probabilities

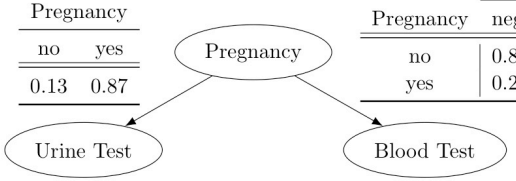
[Salmani and K, QEST 2020]

$$\underbrace{\Pr_{\mathcal{B}}(E)}_{\text{@Bayes network}} = \underbrace{1 - \Pr_{\mathcal{M}_{\mathcal{B}}^{\varrho}}(\Diamond\, \neg E)}_{\text{@Markov chain}} \quad \text{and} \quad \underbrace{\Pr_{\mathcal{B}}(H \mid E)}_{\text{@Bayes network}} = \underbrace{\frac{1 - \Pr_{\mathcal{M}_{\mathcal{B}}^{\varrho}}(\Diamond\, (\neg H \vee \neg E))}{1 - \Pr_{\mathcal{M}_{\mathcal{B}}^{\varrho}}(\Diamond\, \neg E)}}_{\text{@Markov chain}}$$

Pregnancy	Urine Test neg	Urine Test pos
no	0.893	0.107
yes	0.36	0.64

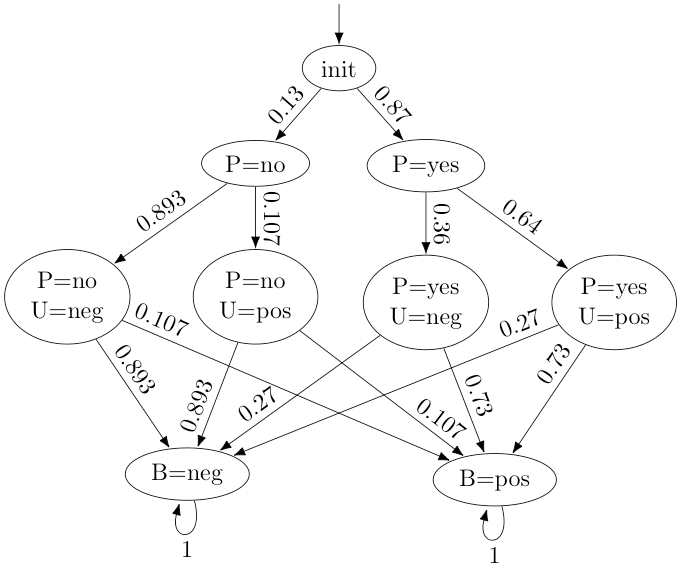
Pregnancy	no	yes
	0.13	0.87

Pregnancy	Blood Test neg	Blood Test pos
no	0.893	0.107
yes	0.27	0.73



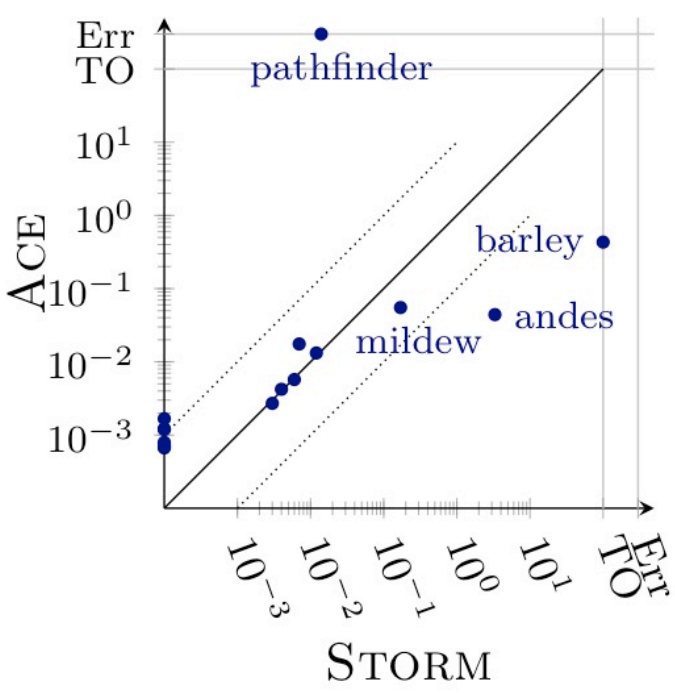
Pr { both tests are neg }

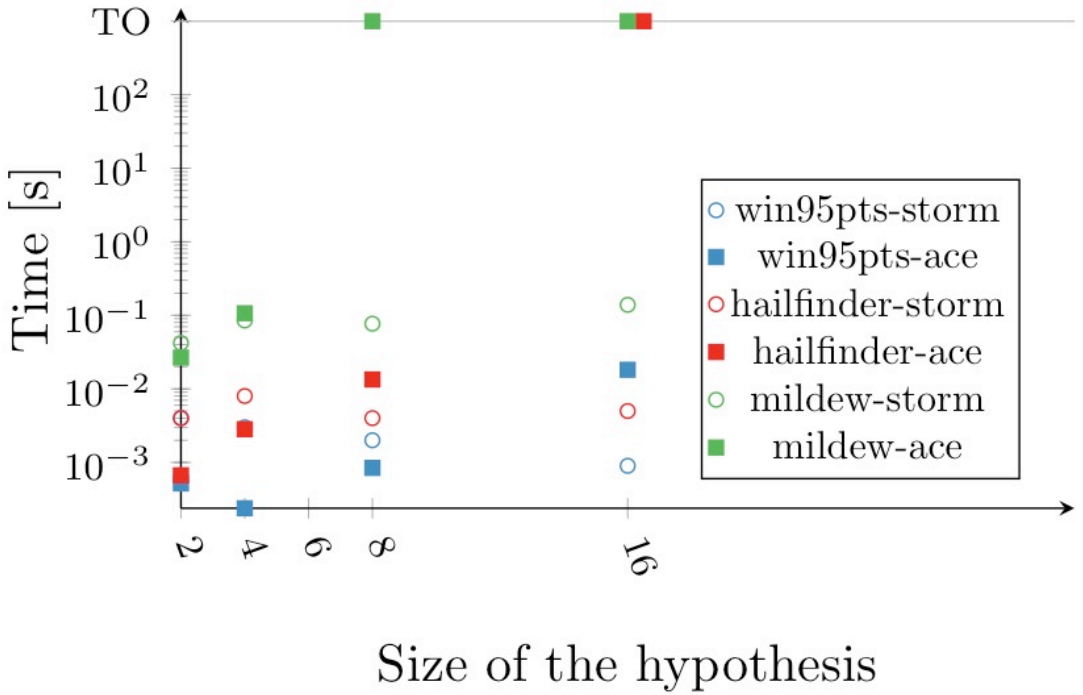
Pr { pregnant | both tests are neg }



Recall: inference in BNs is PP-complete

Bayesian Inference by Explicit-State Model Checking



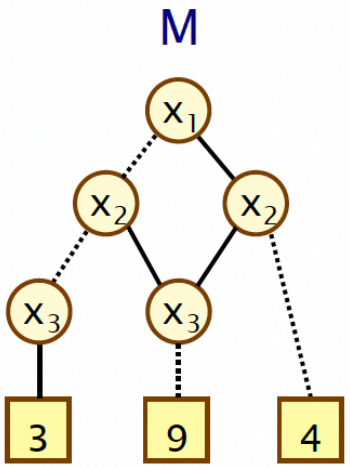


Multi-Terminal Binary Decision Diagrams

- Multi-terminal BDDs (MTBDDs), sometimes called ADDs
  - extension of BDDs to represent real-valued functions
  - like BDDs, an MTBDD M is associated with n Boolean variables
  - MTBDD M represents a function $f_M(x_1,\ldots,x_n) : \{0,1\}^n \rightarrow \mathbb{R}$

For clarity, we omit the zero terminal node and any incoming edges

e.g.



x_1	x_2	x_3	f_M
0	0	0	0
0	0	1	3
0	1	0	9
0	1	1	0
1	0	0	4
1	0	1	4
1	1	0	9
1	1	1	0

Size heavily depends on variable ordering

(courtesy: David Parker)

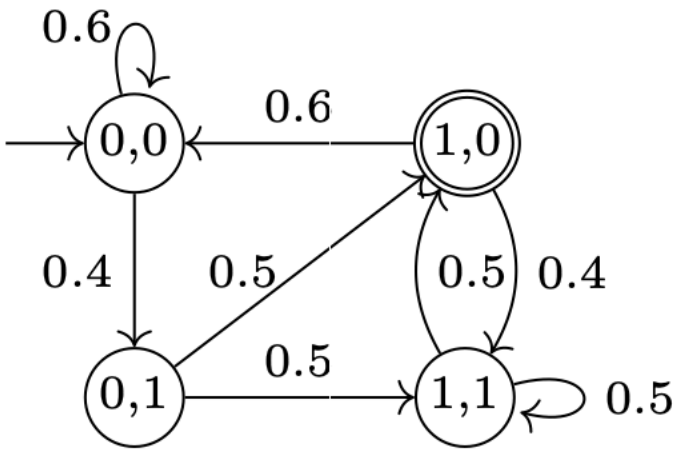
MTBDDs Represent Matrices

- MTBDDs can be used to represent real-valued matrices indexed over a set of states S
  - e.g. the transition probability/rate matrix of a DTMC/CTMC
- For an encoding of state space S into n Boolean variables
  - a matrix **M** maps pairs of states to reals i.e. $\mathbf{M} : S \times S \rightarrow \mathbb{R}$
  - this becomes: $f_M(x_1,\ldots,x_n,y_1,\ldots,y_n) : \{0,1\}^{2n} \rightarrow \mathbb{R}$
- Row and column variables
  - for efficiency reasons, we interleave the row variables $x_1,..,x_n$ and column variables $y_1,\ldots,y_n$
  - i.e. we use function $f_M(x_1,y_1,\ldots,x_n,y_n) : \{0,1\}^{2n} \rightarrow \mathbb{R}$

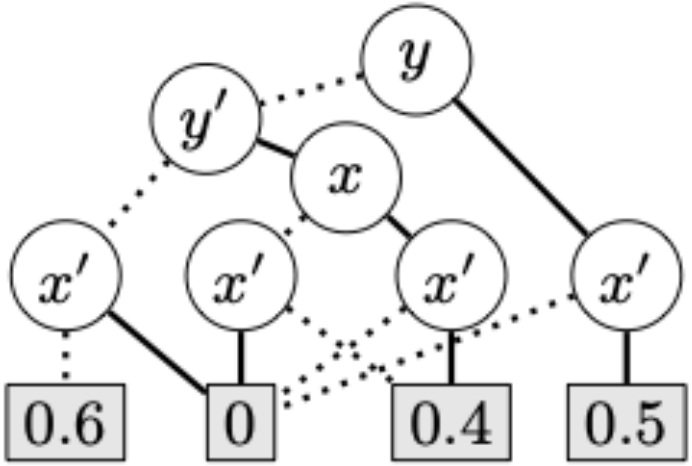
(courtesy: David Parker)

MTBDD Representation of Markov Chains

[Baier et al., ICALP 1997]

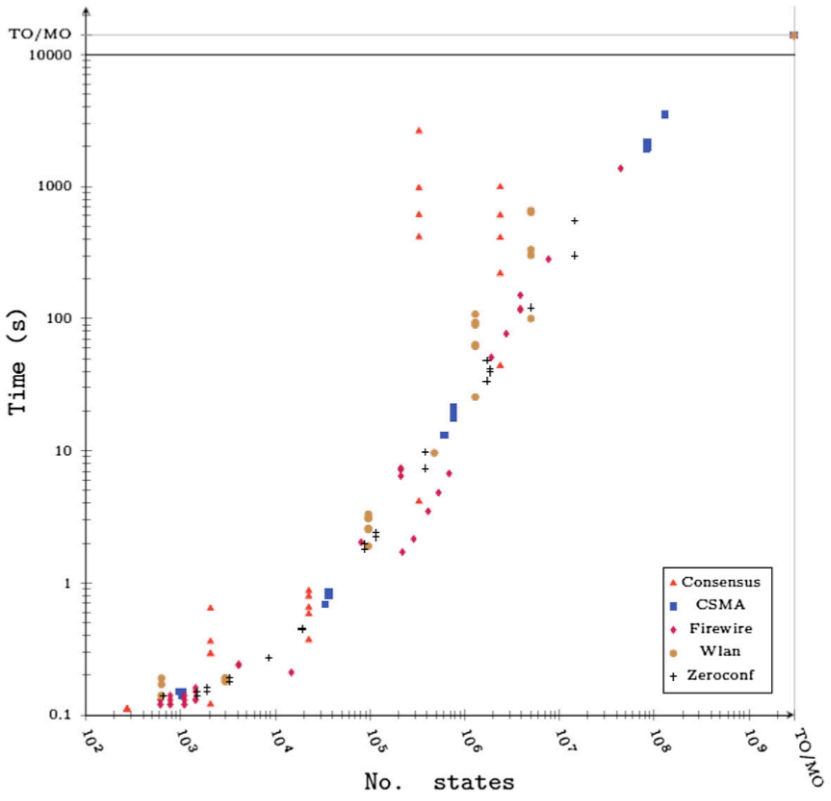


Example Markov chain

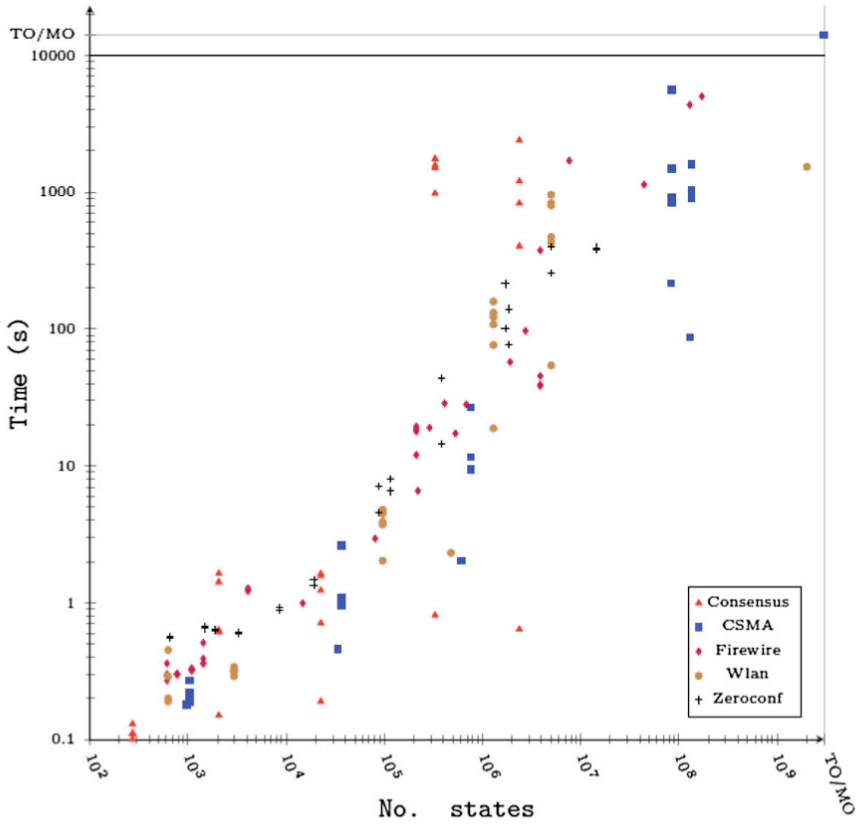


Its MTBDD (*x, x'*) to (*y, y'*)

MDP Model Checking Statistics: Explicit vs MTBDD-Based



Explicit state model checking



Symbolic model checking

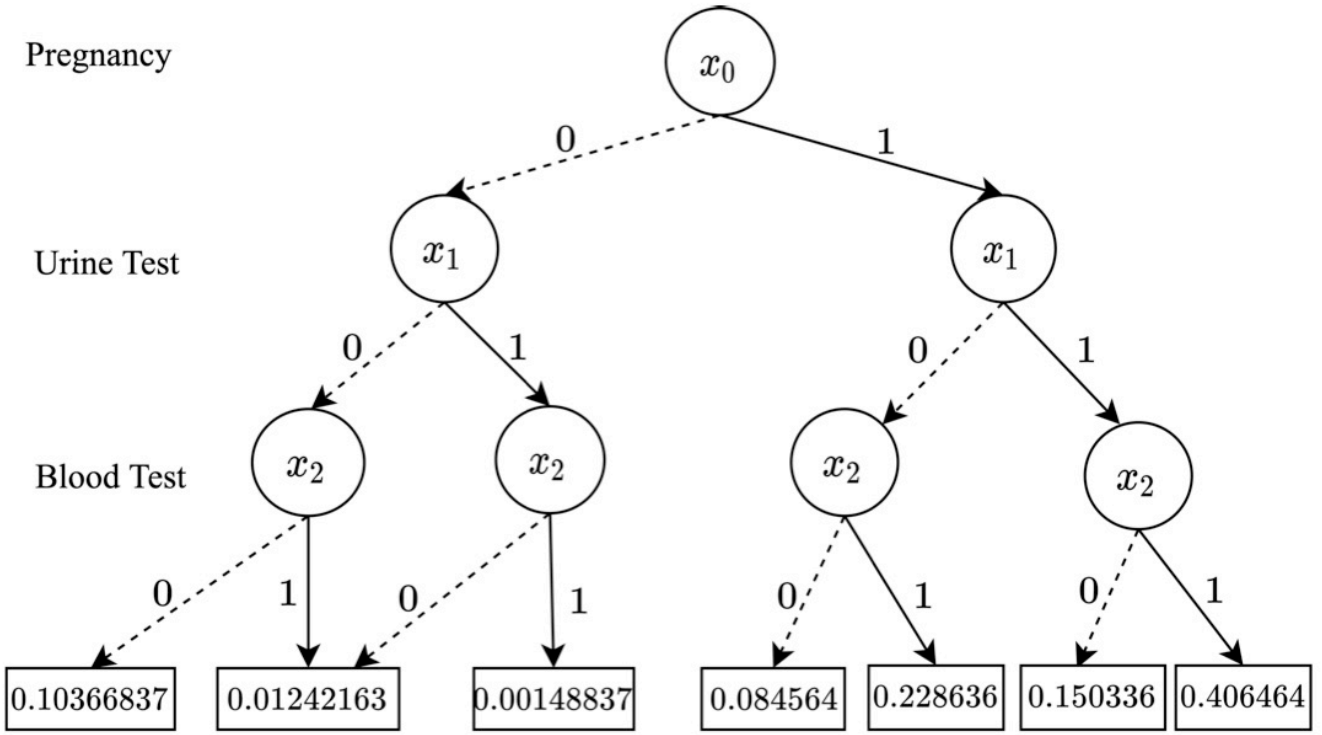
https://www.stormchecker.org

MTBDDs for Bayesian Networks

Pregnancy	Urine Test: neg	Urine Test: pos
no	0.893	0.107
yes	0.36	0.64

Pregnancy: no	Pregnancy: yes
0.13	0.87

Pregnancy	Blood Test: neg	Blood Test: pos
no	0.893	0.107
yes	0.27	0.73



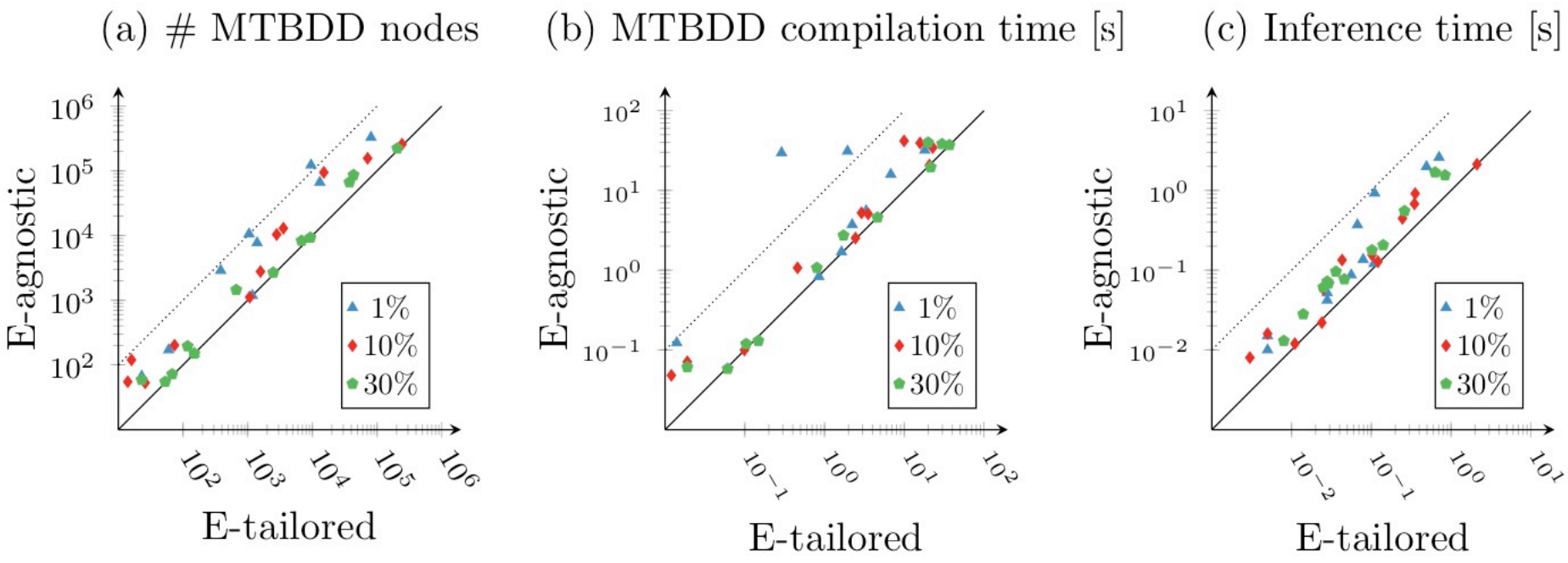
Symbolic Model Checking

	Construction time (in s)	Inference time (in s)
`andes` - Storm, bisimulation	154.66	0.583
`andes` - Storm, MTBDD	303.15	avg: 3.298
`andes` - PSDD, minfill vtree	4.724	3.423
`win95pts` - Storm, bisimulation	0.149	0.002
`win95pts` - Storm, MTBDD	15.740	avg: 0.077
`win95pts` - PSDD, minfill vtree	0.047	0.017

Smart Markov Chain Generation

[Salmani and K, arXiV 2022

Idea: tailor the state-space generation to the evidence of the BN



Model Checking By Inference

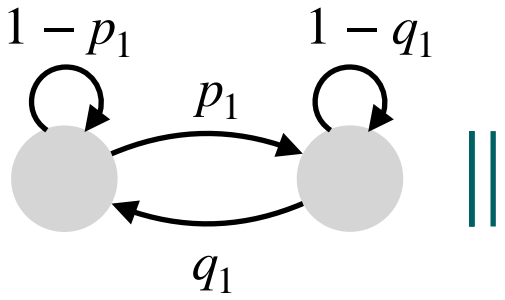
Take-home message:

Computing Finite-Horizon Reachability Probabilities

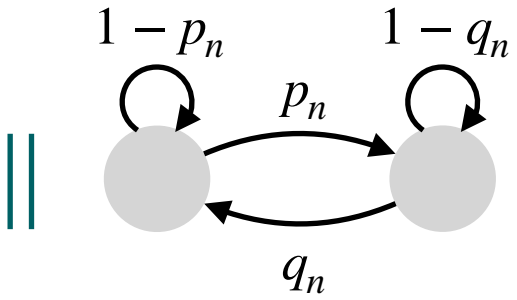
can be Sometimes Efficient Using Inference

Finite-Horizon Reachability by Inference

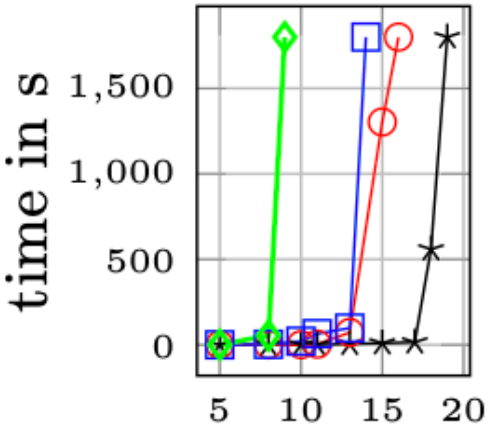
[Holtzen et al., CAV 2021]



Motivating example



Given *n* factories that are either operational or on strike,
what is the probability that all factories are on strike within, say, 10 days?



Prism

Storm DD

Storm Explicit

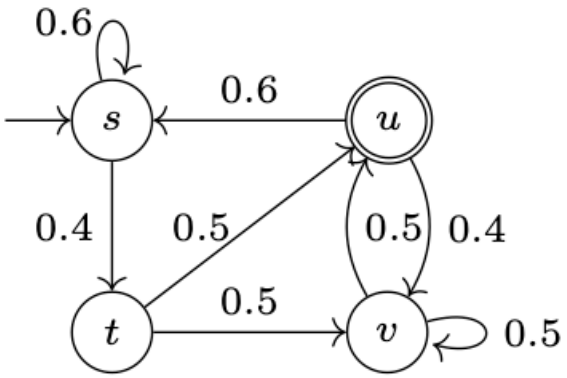
Rubicon

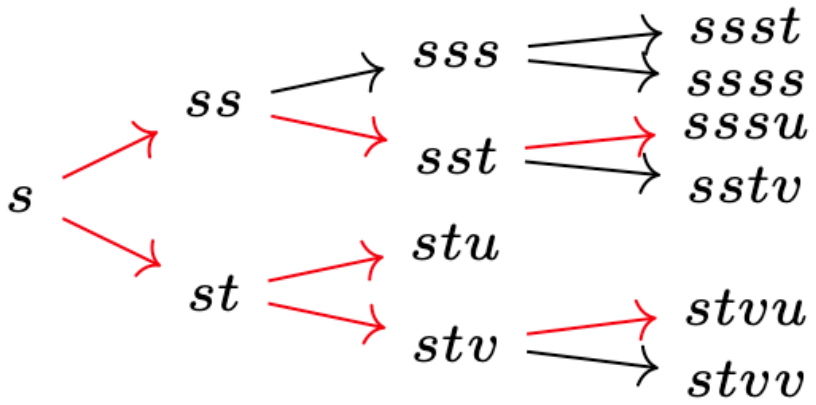
Compact Computation Trees

Idea:

Given a symbolic description of a MC *M*, compute an MC *L*

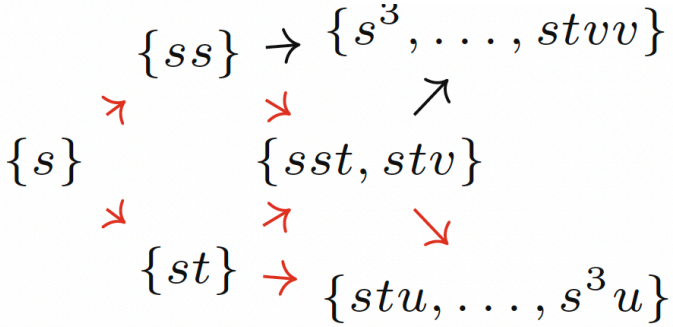
that is bisimulation (aka: lumping) equivalent to *M*'s computation tree up to depth *k*





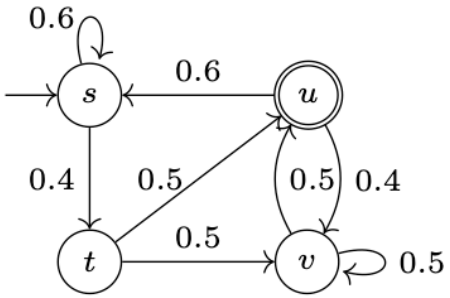
k = 3

Build on-the-fly



k = 3

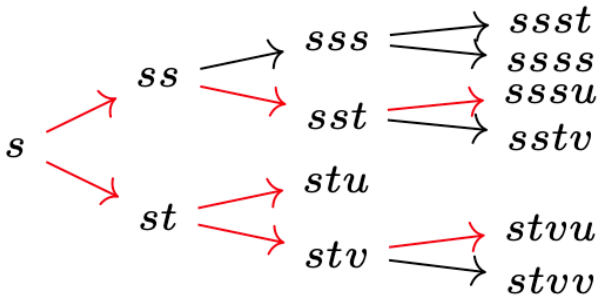
Inference using Weighted Model Counting



Idea:

Given an MC *M*, provide a weighted formula *f* such that

Pr { *M* reaches *G* within *k* steps } = weighted model counting of *f*



$$s = \langle 0,0 \rangle \;<\; t = \langle 0,1 \rangle \;<\; u = \langle 1,0 \rangle \;<\; v = \langle 1,1 \rangle$$

$$\varphi^{C}_{\mathcal{M},3} = (c_{s,0} \wedge \neg c_{s,1} \wedge c_{t,2}) \vee (\neg c_{s,0} \wedge c_{t,1}) \vee (\neg c_{s,0} \wedge \neg c_{t,1} \wedge c_{v,2})$$

Each model of this formula is a single path to the goal

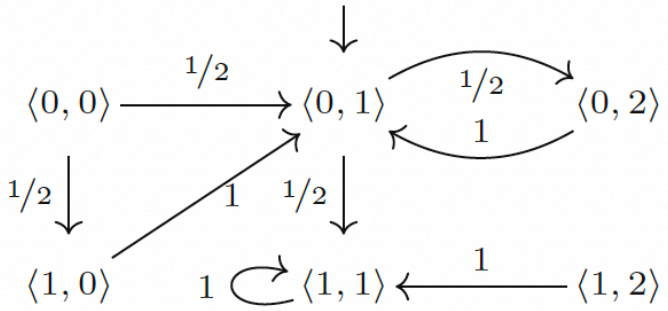
$$W(c_{s,i}) = 0.6 \text{ and } W(c_{t,i}) = W(c_{v,i}) = 0.5$$

Then: $W(f) = 0.42 = \Pr\{M \text{ reaches } u \leq 3 \text{ steps}\}$

From PRISM Models to Dice Programs

```
module main
   x : [0..1] init 0;
   y : [0..2] init 1;
   [] x=0 & y<2 -> 0.5:x'=1 + 0.5:y'=y+1;
   [] y=2 -> 1:y'=y-1;
   [] x=1 & y!=2 -> 1:x'=y & y'=x;
endmodule
property: P=? [F<=2 (x=0 & y=2)]
```

(a) PRISM program with reachability query



(b) Underlying MC

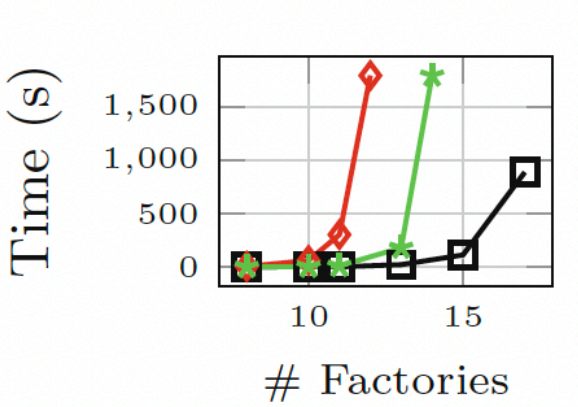
```
let s = init() in // init state
let T = hit(s) in  // init target
let (s, T) = if !T
   then let s' = step(s) in (s', hit(s'))
   else (s, T) in
let (s, T) = if !T then
   then let s' = step(s) in (s', hit(s'))
   else (s, T) in
T
```

(c) Main `Dice` program for h=2

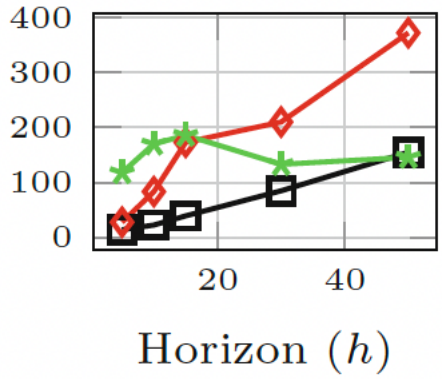
```
fun init() { (0,1) }
fun hit((x,y)) { x ==0  &&  y ==  2 }
fun step((x,y)) {
  if x==0 && y<2 then
     if flip 0.5 then (1,y) else (x,y+1)
  else if y==2 then (x,y-1)
  else if x==1 &&y!=1 then (y,x)
  else (x,y)
}
```

(d) `Dice` auxiliary functions

Experimental Evaluation



(a) Weather Factory



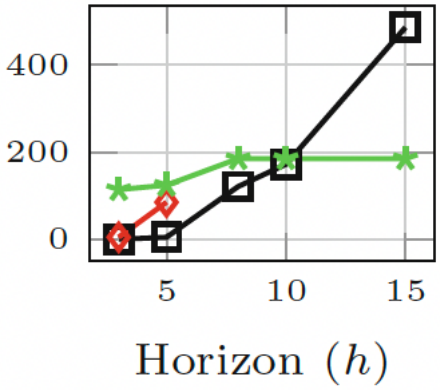
(e) Herman-17

each process
same bias

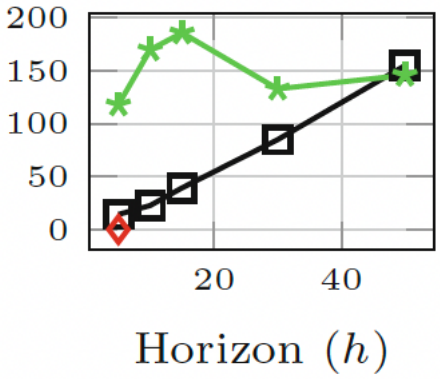
Storm DD

Storm Explicit

Rubicon



(h) Queues



(f) Herman-17 (R)

each process
different bias

Inductive Invariants

Take-home message:

Powerful Alternative for

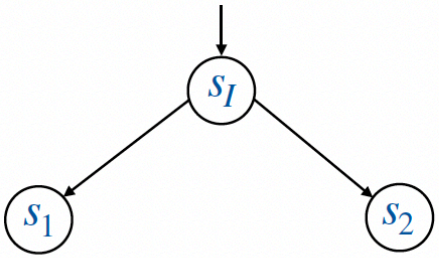
Computing (Un)Reachability Probabilities

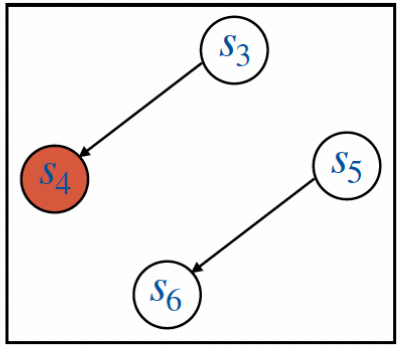
(Even for infinite-state MDPs)

Proving Unreachability

[Batz et al., CAV 2020]

TS $= (S, s_I, T)$ Bad $\subseteq S$





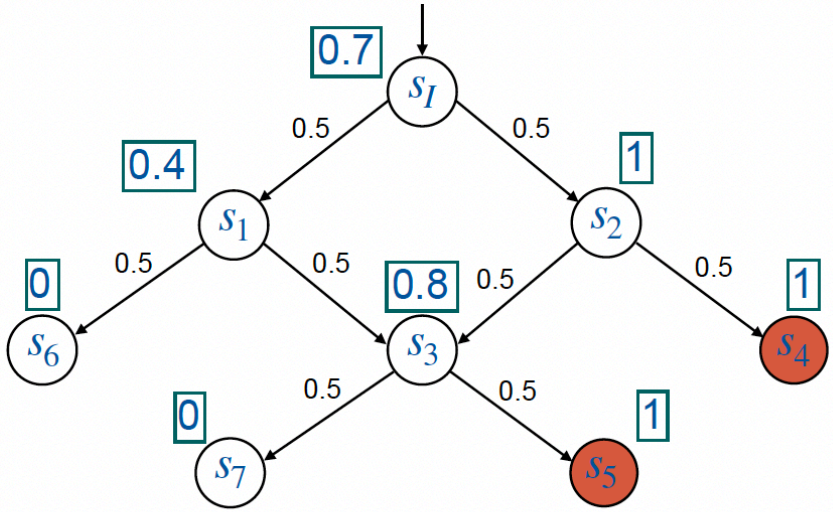
Inductive invariant
$F \subseteq S$ with $s_I \notin F$

or:
$F\colon S \to \{0,1\}$ with $F[s_I] = 0$

Boolean setting

MC $= (S, s_I, P)$ Bad $\subseteq S$ $\lambda = 0.7$

Pr $(s_I \models \Diamond \text{Bad}) = 0.5$



$$\sum_{s' \in S} P(s, s') \cdot F[s'] \leq F[s]$$

Inductive invariant
$F\colon S \to [0,1]$ with $F[s_I] \leq \lambda$

Probabilistic setting

Foundations

TS $= (S, s_I, T)$ Bad $\subseteq S$

Call $F\colon S \to \{0,1\}$ a frame.

Frames are partially ordered by

$$F \leq F' \quad \text{iff} \quad \forall s\colon F[s] \leq F'[s]$$

$$\Phi\colon 2^S \to 2^S,$$
$$\Phi(F) = \text{Bad} \cup \text{Pred}(F)$$

Then:

$s \vDash \Diamond \text{Bad}$ iff $s \in \Phi^\omega(\varnothing)$ iff $s \in \text{lfp}.\Phi$

If $\Phi(F) \leq F$, then F is an inductive invariant.

MC $= (S, s_I, P)$ Bad $\subseteq S$ $\lambda \in [0,1]$

Call $F\colon S \to [0,1]$ a frame.

Frames are partially ordered by

$$F \leq F' \quad \text{iff} \quad \forall s\colon F[s] \leq F'[s]$$

$$\Phi\colon [0,1]^S \to [0,1]^S,$$
$$\Phi(F)[s] = \begin{cases} 1, & \text{if } s \in \text{Bad} \\ \sum_{s' \in S} P(s,s') \cdot F[s'], & \text{else} \end{cases}$$

Then:

$\Pr\left(s \vDash \Diamond \text{Bad}\right) = (\Phi^\omega(\mathbf{0}))[s] = (\text{lfp}.\Phi)[s]$

If $\Phi(F) \leq F$, then F is an inductive invariant.

The Boolean Setting

TS $= (S, s_I, T)$ Bad $\subseteq S$

Call $F\colon S \to \{0,1\}$ a frame.

For increasing $k = 0,1,\ldots$, compute sequence

$$F_0, \ldots, F_k$$

such that

1. Initiality: $F_0 = [\text{Bad}] = \Phi(\varnothing)$
2. Chain-Property: $\forall 0 \leq i < k\colon F_i \leq F_{i+1}$
3. Frame-safety: $\forall 0 \leq i \leq k\colon F_i[s_I] = 0$
4. Relative inductivity: $\forall 0 \leq i < k\colon \Phi(F_i) \leq F_{i+1}$

- $\Diamond^{\leq i}\text{Bad} \leq F_i$
- If $F_i = F_{i+1}$, then $\Phi(F_i) \leq F_i$ hence $s_I \not\models \Diamond\text{Bad}$

The Probabilistic Setting

MC $= (S, s_I, P)$ Bad $\subseteq S$ $\lambda \in [0,1]$

Call $F\colon S \to [0,1]$ a frame.

For increasing $k = 0,1,\ldots$, compute sequence

$$F_0, \ldots, F_k$$

such that

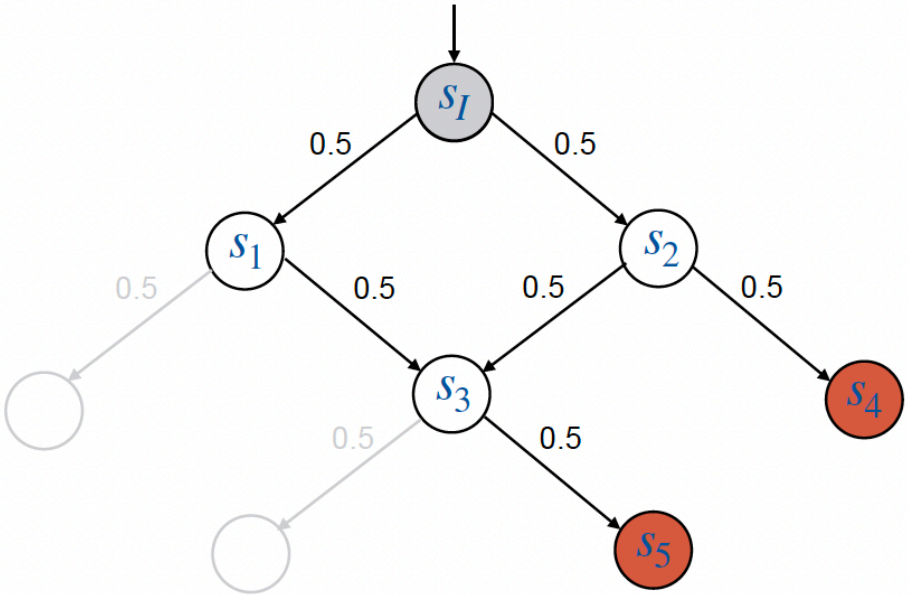
1. Initiality: $F_0 = [\text{Bad}] = \Phi(\mathbf{0})$
2. Chain-Property: $\forall 0 \leq i < k\colon F_i \leq F_{i+1}$
3. Frame-safety: $\forall 0 \leq i \leq k\colon F_i[s_I] \leq \lambda$
4. Relative inductivity: $\forall 0 \leq i < k\colon \Phi(F_i) \leq F_{i+1}$

- $\Pr\left(s \vDash \Diamond^{\leq i} \text{Bad}\right) \leq F_i[s]$
- If $F_i = F_{i+1}$, then $\Phi(F_i) \leq F_i$ hence $\Pr\left(s_I \vDash \Diamond \text{Bad}\right) \leq \lambda$

Example Computing Reach-Probabilities

$\text{MC} = (S, s_I, P)$ $\quad \text{Bad} \subseteq S \quad \lambda = 0.7$

1. Initiality: $F_0 = [\text{Bad}] = \Phi(\mathbf{0})$
2. Chain-Property: $\forall 0 \leq i < k\colon F_i \leq F_{i+1}$
3. Frame-safety: $\forall 0 \leq i \leq k\colon F_i[s_I] \leq 0.7$
4. Relative inductivity: $\forall 0 \leq i < k\colon \Phi(F_i) \leq F_{i+1}$



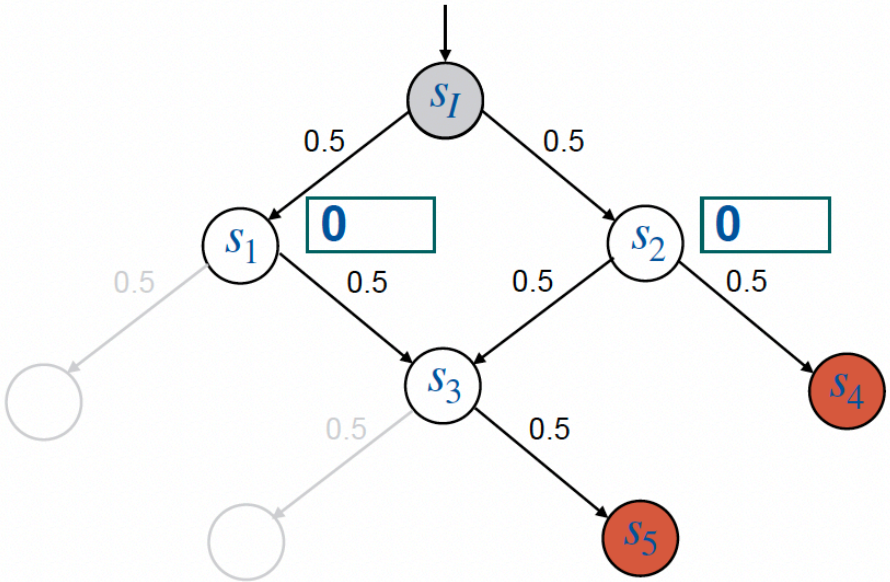
$$F_1 = (\underset{s_I}{1}, \underset{s_1}{1}, \underset{s_2}{1}, \underset{s_3}{1}, \underset{s_4}{1}, \underset{s_5}{1})$$

$$F_0[s] = \begin{cases} 0, & \text{if } s \notin \text{Bad} \\ 1, & \text{if } s \in \text{Bad} \end{cases}$$

Example Computing Reach-Probabilities

$\text{MC} = (S, s_I, P)$ $\quad \text{Bad} \subseteq S \quad \lambda = 0.7$

1. Initiality: $F_0 = [\text{Bad}] = \Phi(\mathbf{0})$
2. Chain-Property: $\forall 0 \leq i < k\colon F_i \leq F_{i+1}$
3. Frame-safety: $\forall 0 \leq i \leq k\colon F_i[s_I] \leq 0.7$
4. Relative inductivity: $\forall 0 \leq i < k\colon \Phi(F_i) \leq F_{i+1}$



Check: $0.5 \cdot F_0[s_1] + 0.5 \cdot F_0[s_2] > 0.7$?

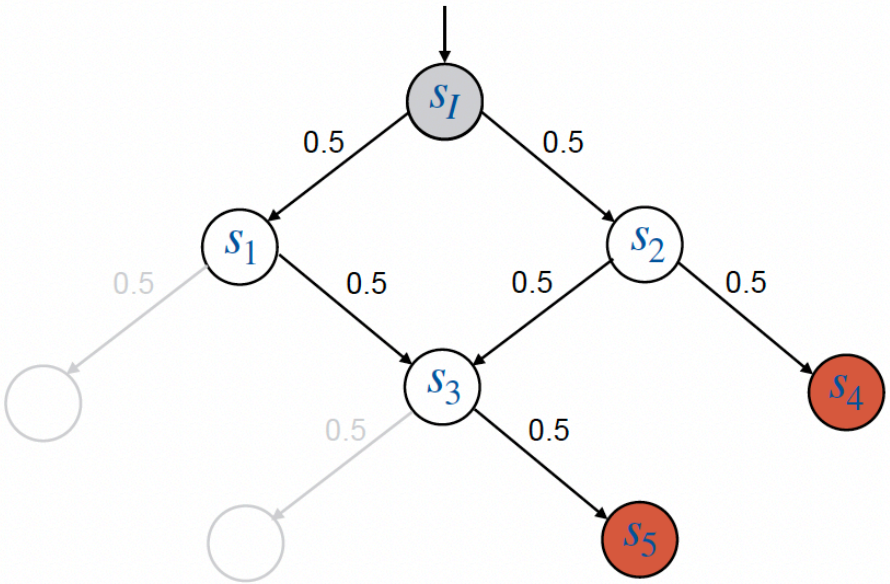
$$F_1 = (\underset{s_I}{1}, \underset{s_1}{1}, \underset{s_2}{1}, \underset{s_3}{1}, \underset{s_4}{1}, \underset{s_5}{1})$$

$$\mathbf{F_0(s)} = \begin{cases} \mathbf{0}, & \text{if } s \notin \text{Bad} \\ 1, & \text{if } s \in \text{Bad} \end{cases}$$

Example Computing Reach-Probabilities

$\text{MC} = (S, s_I, P)$ $\quad \text{Bad} \subseteq S \quad \lambda = 0.7$

1. Initiality: $F_0 = [\text{Bad}] = \Phi(\mathbf{0})$
2. Chain-Property: $\forall 0 \leq i < k\colon F_i \leq F_{i+1}$
3. Frame-safety: $\forall 0 \leq i \leq k\colon F_i[s_I] \leq 0.7$
4. Relative inductivity: $\forall 0 \leq i < k\colon \Phi(F_i) \leq F_{i+1}$



$$F_2 = (\ \underset{s_I}{1}\ ,\ \underset{s_1}{1}\ ,\ \underset{s_2}{1}\ ,\ \underset{s_3}{1}\ ,\ \underset{s_4}{1}\ ,\ \underset{s_5}{1}\ )$$

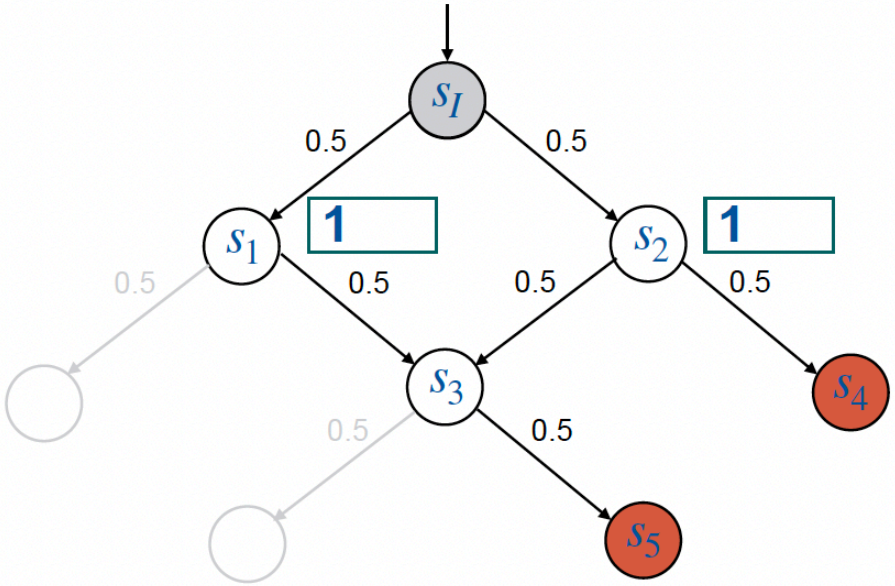
$$F_1 = (\ \underset{s_I}{0.7}\ ,\ \underset{s_1}{1}\ ,\ \underset{s_2}{1}\ ,\ \underset{s_3}{1}\ ,\ \underset{s_4}{1}\ ,\ \underset{s_5}{1}\ )$$

$$F_0(s) = \begin{cases} 0, & \text{if } s \notin \text{Bad} \\ 1, & \text{if } s \in \text{Bad} \end{cases}$$

Example Computing Reach-Probabilities

$\text{MC} = (S, s_I, P)$ $\quad \text{Bad} \subseteq S \quad \lambda = 0.7$

1. Initiality: $F_0 = [\text{Bad}] = \Phi(\mathbf{0})$
2. Chain-Property: $\forall 0 \leq i < k\colon F_i \leq F_{i+1}$
3. Frame-safety: $\forall 0 \leq i \leq k\colon F_i[s_I] \leq 0.7$
4. Relative inductivity: $\forall 0 \leq i < k\colon \Phi(F_i) \leq F_{i+1}$



Check: $0.5 \cdot F_1[s_1] + 0.5 \cdot F_1[s_2] > 0.7$?

$$F_2 = \begin{pmatrix} s_I & s_1 & s_2 & s_3 & s_4 & s_5 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

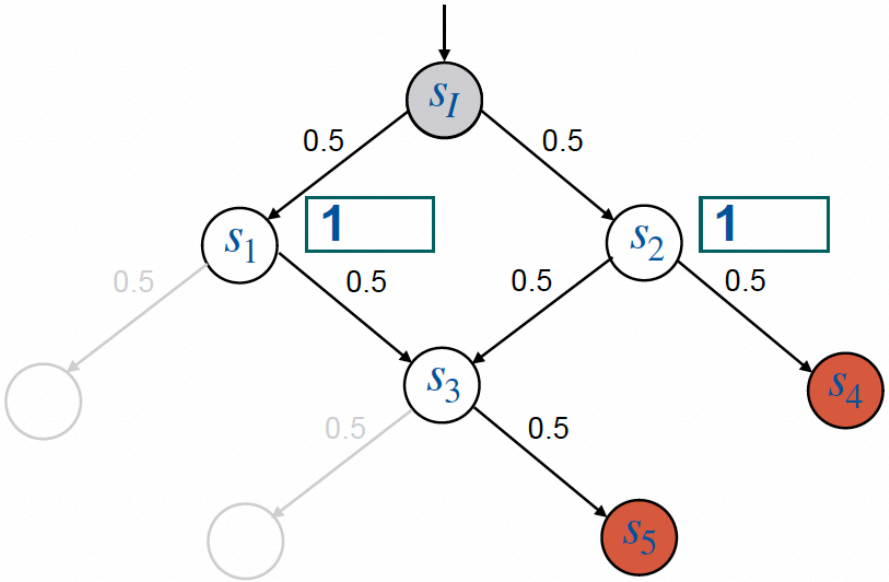
$$F_1 = \begin{pmatrix} s_I & s_1 & s_2 & s_3 & s_4 & s_5 \\ 0.7 & \mathbf{1} & \mathbf{1} & 1 & 1 & 1 \end{pmatrix}$$

$$F_0(s) = \begin{cases} 0, & \text{if } s \notin \text{Bad} \\ 1, & \text{if } s \in \text{Bad} \end{cases}$$

Example Computing Reach-Probabilities

$\text{MC} = (S, s_I, P)$ Bad $\subseteq S$ $\lambda = 0.7$

1. Initiality: $F_0 = [\text{Bad}] = \Phi(\mathbf{0})$
2. Chain-Property: $\forall 0 \leq i < k : F_i \leq F_{i+1}$
3. Frame-safety: $\forall 0 \leq i \leq k : F_i[s_I] \leq 0.7$
4. Relative inductivity: $\forall 0 \leq i < k : \Phi(F_i) \leq F_{i+1}$



Find $x_1, x_2 \in [0,1]$ such that

$$0.5 \cdot x_1 + 0.5 \cdot x_2 \leq 0.7$$

Problem: Infinitely many choices.
There are "bad" choices.
Requires heuristic/oracle.

$$F_2 = \begin{pmatrix} s_I & s_1 & s_2 & s_3 & s_4 & s_5 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

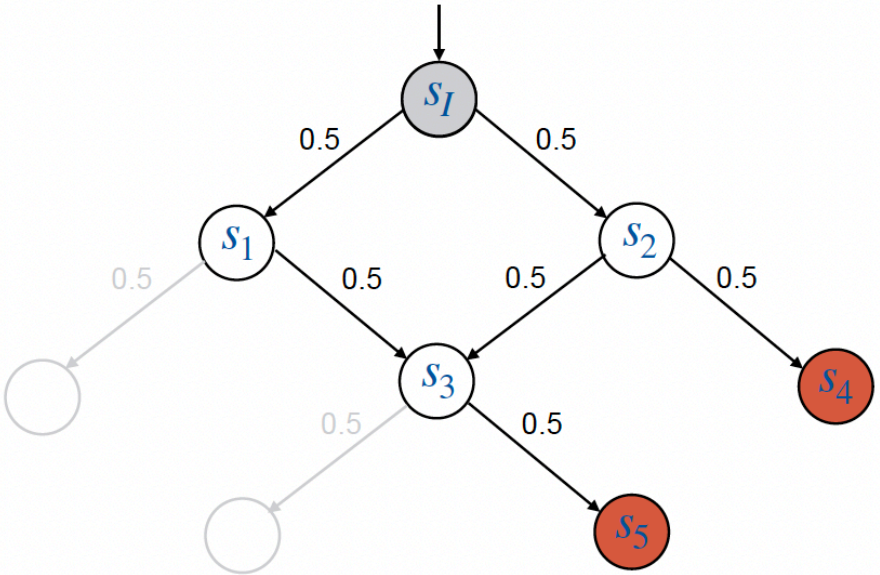
$$F_1 = \begin{pmatrix} s_I & s_1 & s_2 & s_3 & s_4 & s_5 \\ 0.7 & \mathbf{1} & \mathbf{1} & 1 & 1 & 1 \end{pmatrix}$$

$$F_0(s) = \begin{cases} 0, & \text{if } s \notin \text{Bad} \\ 1, & \text{if } s \in \text{Bad} \end{cases}$$

Example Computing Reach-Probabilities

$\text{MC} = (S, s_I, P)$ $\quad \text{Bad} \subseteq S \quad \lambda = 0.7$

1. Initiality: $F_0 = [\text{Bad}] = \Phi(\mathbf{0})$
2. Chain-Property: $\forall 0 \leq i < k\colon F_i \leq F_{i+1}$
3. Frame-safety: $\forall 0 \leq i \leq k\colon F_i[s_I] \leq 0.7$
4. Relative inductivity: $\forall 0 \leq i < k\colon \Phi(F_i) \leq F_{i+1}$



$$F_2 = (\ \underset{s_I}{1}\ ,\ \underset{s_1}{1}\ ,\ \underset{s_2}{1}\ ,\ \underset{s_3}{1}\ ,\ \underset{s_4}{1}\ ,\ \underset{s_5}{1}\ )$$

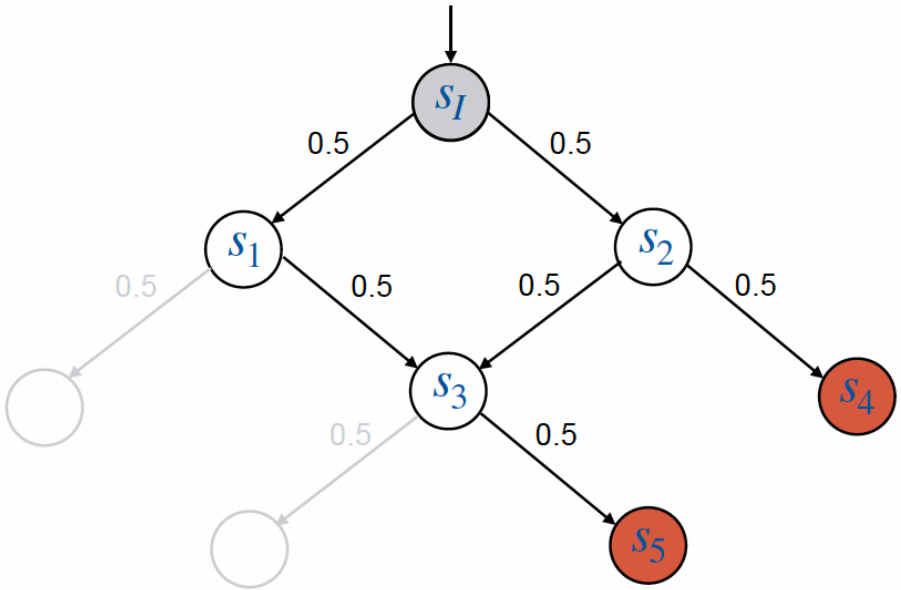
$$F_1 = (\ \underset{s_I}{0.7}\ ,\ \underset{s_1}{0.4}\ ,\ \underset{s_2}{1}\ ,\ \underset{s_3}{1}\ ,\ \underset{s_4}{1}\ ,\ \underset{s_5}{1}\ )$$

$$F_0(s) = \begin{cases} 0, & \text{if } s \notin \text{Bad} \\ 1, & \text{if } s \in \text{Bad} \end{cases}$$

Example Computing Reach-Probabilities

$\text{MC} = (S, s_I, P)$ $\quad \text{Bad} \subseteq S \quad \lambda = 0.7$

1. Initiality: $F_0 = [\text{Bad}] = \Phi(\mathbf{0})$
2. Chain-Property: $\forall 0 \leq i < k\colon F_i \leq F_{i+1}$
3. Frame-safety: $\forall 0 \leq i \leq k\colon F_i[s_I] \leq 0.7$
4. Relative inductivity: $\forall 0 \leq i < k\colon \Phi(F_i) \leq F_{i+1}$



$$F_3 = (\ \underset{s_I}{1}\ ,\ \underset{s_1}{1}\ ,\ \underset{s_2}{1}\ ,\ \underset{s_3}{1}\ ,\ \underset{s_4}{1}\ ,\ \underset{s_5}{1}\ )$$

$$F_2 = (\ \underset{s_I}{0.7}\ ,\ \underset{s_1}{1}\ ,\ \underset{s_2}{1}\ ,\ \underset{s_3}{1}\ ,\ \underset{s_4}{1}\ ,\ \underset{s_5}{1}\ )$$

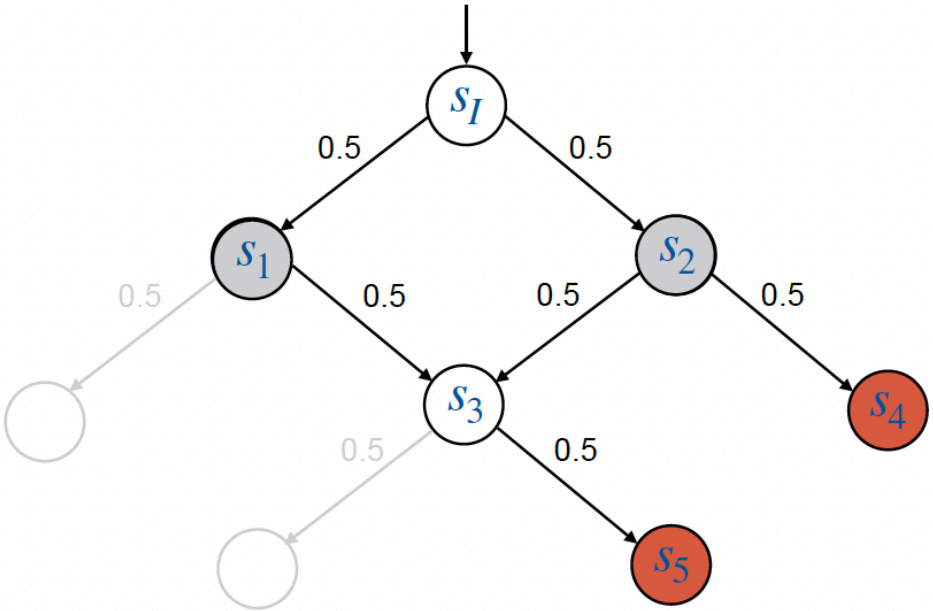
$$F_1 = (\ \underset{s_I}{0.7}\ ,\ \underset{s_1}{0.4}\ ,\ \underset{s_2}{1}\ ,\ \underset{s_3}{1}\ ,\ \underset{s_4}{1}\ ,\ \underset{s_5}{1}\ )$$

$$F_0(s) = \begin{cases} 0, & \text{if } s \notin \text{Bad} \\ 1, & \text{if } s \in \text{Bad} \end{cases}$$

Example Computing Reach-Probabilities

$\text{MC} = (S, s_I, P)$ $\quad \text{Bad} \subseteq S \quad \lambda = 0.7$

1. Initiality: $F_0 = [\text{Bad}] = \Phi(\mathbf{0})$
2. Chain-Property: $\forall 0 \leq i < k\colon F_i \leq F_{i+1}$
3. Frame-safety: $\forall 0 \leq i \leq k\colon F_i[s_I] \leq 0.7$
4. Relative inductivity: $\forall 0 \leq i < k\colon \Phi(F_i) \leq F_{i+1}$



$$F_3 = (\ \underset{s_I}{1}\ ,\ \underset{s_1}{1}\ ,\ \underset{s_2}{1}\ ,\ \underset{s_3}{1}\ ,\ \underset{s_4}{1}\ ,\ \underset{s_5}{1}\ )$$

$$F_2 = (\ \underset{s_I}{0.7}\ ,\ \underset{s_1}{1}\ ,\ \underset{s_2}{1}\ ,\ \underset{s_3}{1}\ ,\ \underset{s_4}{1}\ ,\ \underset{s_5}{1}\ )$$

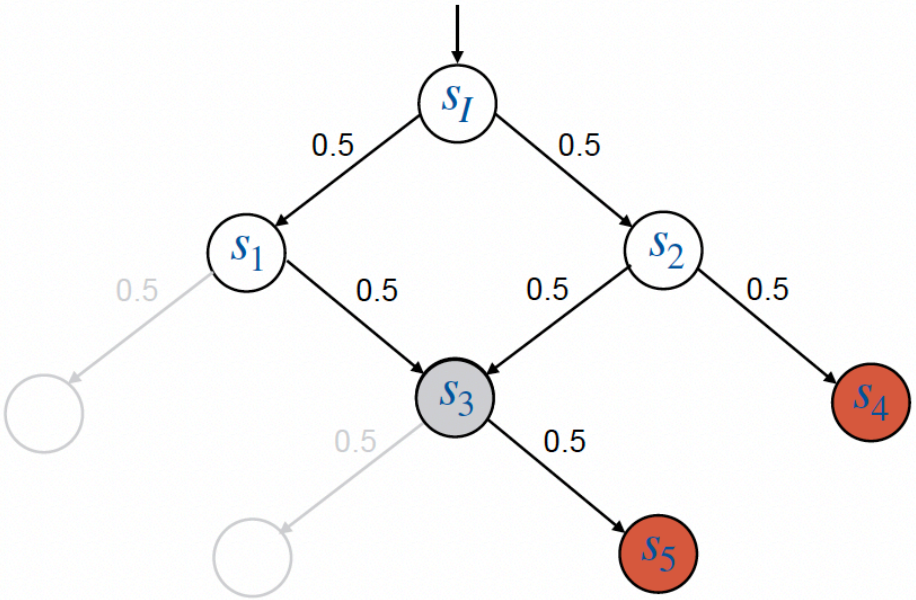
$$F_1 = (\ \underset{s_I}{0.7}\ ,\ \underset{s_1}{0.4}\ ,\ \underset{s_2}{1}\ ,\ \underset{s_3}{1}\ ,\ \underset{s_4}{1}\ ,\ \underset{s_5}{1}\ )$$

$$F_0(s) = \begin{cases} 0, & \text{if } s \notin \text{Bad} \\ 1, & \text{if } s \in \text{Bad} \end{cases}$$

Example Computing Reach-Probabilities

$\text{MC} = (S, s_I, P)$ $\quad \text{Bad} \subseteq S \quad \lambda = 0.7$

1. Initiality: $F_0 = [\text{Bad}] = \Phi(\mathbf{0})$
2. Chain-Property: $\forall 0 \leq i < k\colon F_i \leq F_{i+1}$
3. Frame-safety: $\forall 0 \leq i \leq k\colon F_i[s_I] \leq 0.7$
4. Relative inductivity: $\forall 0 \leq i < k\colon \Phi(F_i) \leq F_{i+1}$



$$F_3 = (\ \underset{s_I}{1}\ ,\ \underset{s_1}{1}\ ,\ \underset{s_2}{1}\ ,\ \underset{s_3}{1}\ ,\ \underset{s_4}{1}\ ,\ \underset{s_5}{1}\ )$$

$$F_2 = (\ \underset{s_I}{0.7}\ ,\ \underset{s_1}{1}\ ,\ \underset{s_2}{1}\ ,\ \underset{s_3}{1}\ ,\ \underset{s_4}{1}\ ,\ \underset{s_5}{1}\ )$$

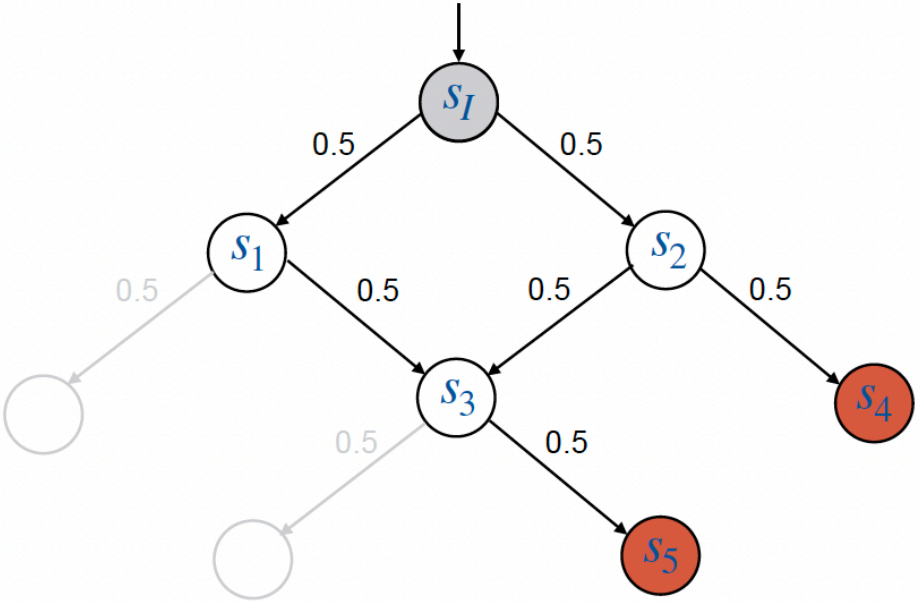
$$F_1 = (\ \underset{s_I}{0.7}\ ,\ \underset{s_1}{0.4}\ ,\ \underset{s_2}{1}\ ,\ \underset{s_3}{1}\ ,\ \underset{s_4}{1}\ ,\ \underset{s_5}{1}\ )$$

$$F_0(s) = \begin{cases} 0, & \text{if } s \notin \text{Bad} \\ 1, & \text{if } s \in \text{Bad} \end{cases}$$

Example Computing Reach-Probabilities

$\text{MC} = (S, s_I, P)$ $\quad \text{Bad} \subseteq S \quad \lambda = 0.7$

1. Initiality: $F_0 = [\text{Bad}] = \Phi(\mathbf{0})$
2. Chain-Property: $\forall 0 \leq i < k\colon F_i \leq F_{i+1}$
3. Frame-safety: $\forall 0 \leq i \leq k\colon F_i[s_I] \leq 0.7$
4. Relative inductivity: $\forall 0 \leq i < k\colon \Phi(F_i) \leq F_{i+1}$



$$F_3 = (\ \overset{s_I}{1}\ ,\ \overset{s_1}{1}\ ,\ \overset{s_2}{1}\ ,\ \overset{s_3}{1}\ ,\ \overset{s_4}{1}\ ,\ \overset{s_5}{1}\ )$$

$$F_2 = (\ \overset{s_I}{0.7}\ ,\ \overset{s_1}{0.4}\ ,\ \overset{s_2}{1}\ ,\ \overset{s_3}{1}\ ,\ \overset{s_4}{1}\ ,\ \overset{s_5}{1}\ )$$

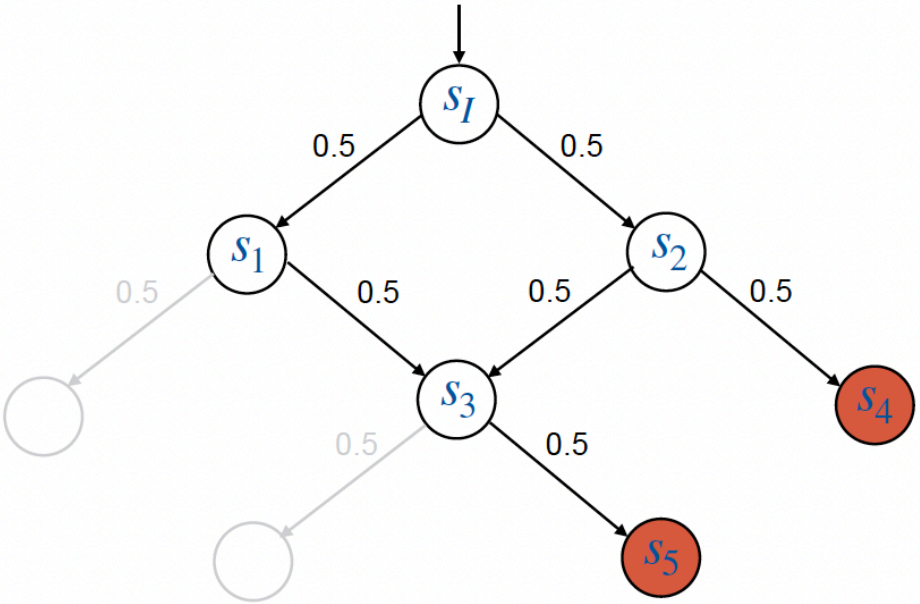
$$F_1 = (\ \overset{s_I}{0.7}\ ,\ \overset{s_1}{0.4}\ ,\ \overset{s_2}{1}\ ,\ \overset{s_3}{0.8}\ ,\ \overset{s_4}{1}\ ,\ \overset{s_5}{1}\ )$$

$$F_0(s) = \begin{cases} 0, & \text{if } s \notin \text{Bad} \\ 1, & \text{if } s \in \text{Bad} \end{cases}$$

Example Computing Reach-Probabilities

$\text{MC} = (S, s_I, P)$ $\quad \text{Bad} \subseteq S \quad \lambda = 0.7$

1. Initiality: $F_0 = [\text{Bad}] = \Phi(\mathbf{0})$
2. Chain-Property: $\forall 0 \leq i < k : F_i \leq F_{i+1}$
3. Frame-safety: $\forall 0 \leq i \leq k : F_i[s_I] \leq 0.7$
4. Relative inductivity: $\forall 0 \leq i < k : \Phi(F_i) \leq F_{i+1}$



$$F_3 = (\ \underset{0.7}{s_I}\ ,\ \underset{1}{s_1}\ ,\ \underset{1}{s_2}\ ,\ \underset{1}{s_3}\ ,\ \underset{1}{s_4}\ ,\ \underset{1}{s_5}\ )$$

$$F_2 = (\ \underset{0.7}{s_I}\ ,\ \underset{0.4}{s_1}\ ,\ \underset{1}{s_2}\ ,\ \underset{1}{s_3}\ ,\ \underset{1}{s_4}\ ,\ \underset{1}{s_5}\ )$$

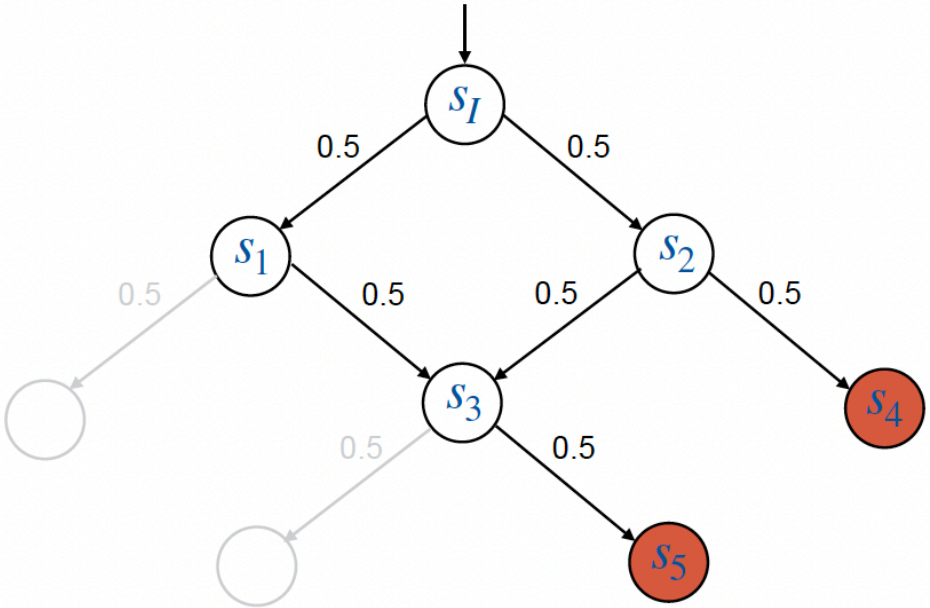
$$F_1 = (\ \underset{0.7}{s_I}\ ,\ \underset{0.4}{s_1}\ ,\ \underset{1}{s_2}\ ,\ \underset{0.8}{s_3}\ ,\ \underset{1}{s_4}\ ,\ \underset{1}{s_5}\ )$$

$$F_0(s) = \begin{cases} 0, & \text{if } s \notin \text{Bad} \\ 1, & \text{if } s \in \text{Bad} \end{cases}$$

Example Computing Reach-Probabilities

$\text{MC} = (S, s_I, P)$ $\quad \text{Bad} \subseteq S \quad \lambda = 0.7$

1. Initiality: $F_0 = [\text{Bad}] = \Phi(\mathbf{0})$
2. Chain-Property: $\forall 0 \leq i < k\colon F_i \leq F_{i+1}$
3. Frame-safety: $\forall 0 \leq i \leq k\colon F_i[s_I] \leq 0.7$
4. Relative inductivity: $\forall 0 \leq i < k\colon \Phi(F_i) \leq F_{i+1}$



$$F_4 = (\ \underset{1}{s_I}\ ,\ \underset{1}{s_1}\ ,\ \underset{1}{s_2}\ ,\ \underset{1}{s_3}\ ,\ \underset{1}{s_4}\ ,\ \underset{1}{s_5}\ )$$

$$F_3 = (\ \underset{0.7}{s_I}\ ,\ \underset{1}{s_1}\ ,\ \underset{1}{s_2}\ ,\ \underset{1}{s_3}\ ,\ \underset{1}{s_4}\ ,\ \underset{1}{s_5}\ )$$

$$F_2 = (\ \underset{0.7}{s_I}\ ,\ \underset{0.4}{s_1}\ ,\ \underset{1}{s_2}\ ,\ \underset{1}{s_3}\ ,\ \underset{1}{s_4}\ ,\ \underset{1}{s_5}\ )$$

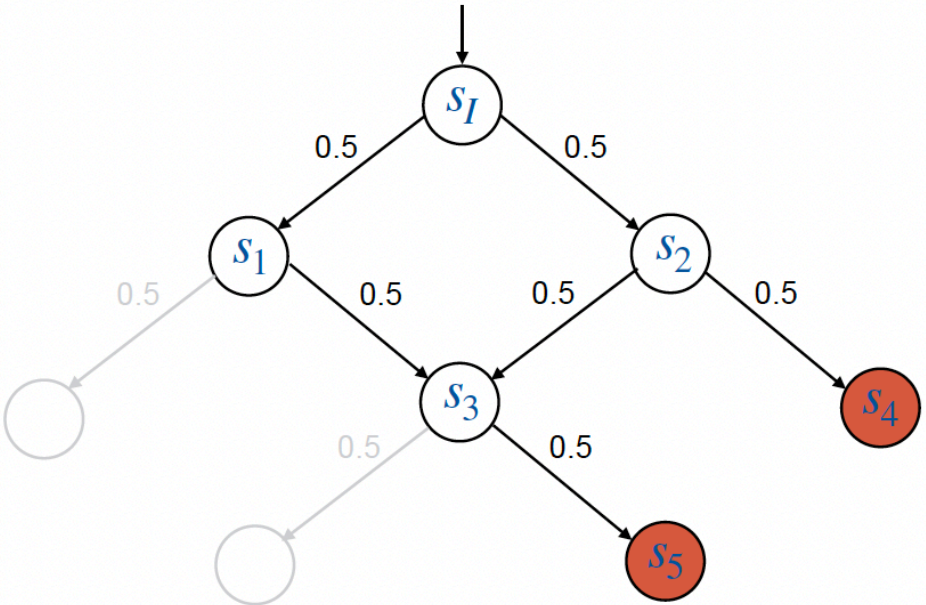
$$F_1 = (\ \underset{0.7}{s_I}\ ,\ \underset{0.4}{s_1}\ ,\ \underset{1}{s_2}\ ,\ \underset{0.8}{s_3}\ ,\ \underset{1}{s_4}\ ,\ \underset{1}{s_5}\ )$$

$$F_0(s) = \begin{cases} 0, & \text{if } s \notin \text{Bad} \\ 1, & \text{if } s \in \text{Bad} \end{cases}$$

Example Computing Reach-Probabilities

$\text{MC} = (S, s_I, P)$ $\quad \text{Bad} \subseteq S \quad \lambda = 0.7$

1. Initiality: $F_0 = [\text{Bad}] = \Phi(\mathbf{0})$
2. Chain-Property: $\forall 0 \leq i < k\colon F_i \leq F_{i+1}$
3. Frame-safety: $\forall 0 \leq i \leq k\colon F_i[s_I] \leq 0.7$
4. Relative inductivity: $\forall 0 \leq i < k\colon \Phi(F_i) \leq F_{i+1}$



$$F_4 = (\ \underset{s_I}{0.7}\ ,\ \underset{s_1}{1}\ ,\ \underset{s_2}{1}\ ,\ \underset{s_3}{1}\ ,\ \underset{s_4}{1}\ ,\ \underset{s_5}{1}\ )$$

$$F_3 = (\ \underset{s_I}{0.7}\ ,\ \underset{s_1}{0.4}\ ,\ \underset{s_2}{1}\ ,\ \underset{s_3}{1}\ ,\ \underset{s_4}{1}\ ,\ \underset{s_5}{1}\ )$$

$$F_2 = (\ \underset{s_I}{0.7}\ ,\ \underset{s_1}{0.4}\ ,\ \underset{s_2}{1}\ ,\ \underset{s_3}{0.8}\ ,\ \underset{s_4}{1}\ ,\ \underset{s_5}{1}\ )$$

$$F_1 = (\ \underset{s_I}{0.7}\ ,\ \underset{s_1}{0.4}\ ,\ \underset{s_2}{1}\ ,\ \underset{s_3}{0.8}\ ,\ \underset{s_4}{1}\ ,\ \underset{s_5}{1}\ )$$

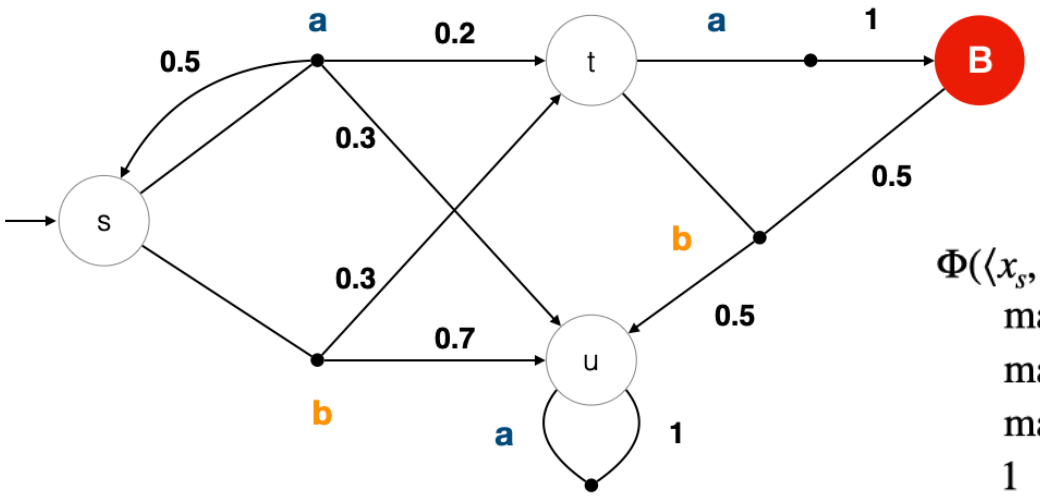
$$F_0(s) = \begin{cases} 0, & \text{if } s \notin \text{Bad} \\ 1, & \text{if } s \in \text{Bad} \end{cases}$$

MDP Frame Transformer := Bellman Operator

Definition 2 (Bellman Operator). *For a set of actions* $A \subseteq \mathrm{Act}$, *we define the* Bellman operator for A *as a frame transformer* $\Phi_A : [0,1]^S \to [0,1]^S$ *with*

$$\Phi_A(F)[s] = \begin{cases} 1, & \text{if } s \in B \\ \max\limits_{a \in A} \sum\limits_{s' \in S} P(s,a,s') \cdot F[s'], & \text{if } s \notin B\,. \end{cases}$$

bad states



$$\Phi(\langle x_s, x_t, x_u, x_B \rangle) =$$
$$\max\{0.5 \cdot x_s + 0.2 \cdot x_t + 0.3 \cdot x_u,\ 0.3 \cdot x_t + 0.7 \cdot x_u\}$$
$$\max\{x_B,\ 0.5 \cdot x_B + 0.5 \cdot x_B\}$$
$$\max\{x_u\}$$
$$1$$

Empirical Results

no generalisation

with generalisation

	$\lvert S\rvert$	$\Pr^{\max}(s_I \models \Diamond B)$	λ	w/o	$\lvert sub\rvert$	pol	$\lvert sub\rvert$	$\text{Storm}_{\text{sparse}}$	Storm_{dd}
BRP	10^3	0.035	0.01	**51.3**	324	TO	–	<0.1	0.18
			0.005	**10.9**	188	TO	–	<0.1	0.1
ZeroConf	10^9	~0.55	0.9	TO	–	**3.7**	0	MO	TO
			0.75	TO	–	**3.4**	0	MO	TO
			0.52	TO	–	TO	– -	MO	TO
			0.45	**<0.1**	1	**<0.1**	1	MO	TO
Chain	10^{12}	0.394	0.9	TO	–	**6.4**	0	MO	TO
			0.4	TO	–	**6.0**	0	MO	TO

15 minutes

https://www.stormchecker.org

More information: [Batz *et al.*, CAV 2020]

k-Inductive Invariants

Take-home message:

An Even More Powerful Alternative for

Computing (Un)Reachability Probabilities

(Even for infinite-state MDPs)

k-Induction

[Sheeran et al., 2000]

- SAT-based technique for verifying invariant properties of finite transition systems
- Later: Verification of *infinite-state* transition systems via SMT solving
- Applications: Hardware- and software model checking

> " [k-induction] easily integrates with existing SAT-solvers [...]. The simplicity of applying k-induction made it the go-to technique for SMT-based infinite-state model checking."[1]

Question:

Is k-induction applicable to
(possibly infinite-state) probabilistic program verification?

Yes. Enables fully automated verification of non-trivial properties.

k-Induction for Infinite-State MDPs

[Batz et al., CAV 2021]

Fully automated k-inductivity checks by SMT solving:

bounded retransmission protocol I

$$\texttt{while}\,(\,sent < toSend \wedge fail < maxFail\,)\,\{$$
$$\{\,\underbrace{fail := 0\,;\,sent := sent + 1}_{\text{successful transmission}}\,\}[0.9]\{\,\underbrace{fail := fail + 1\,;\,totalFail := totalFail + 1}_{\text{failed transmission}}\,\}\}$$

Then:

$$\text{Exp}\,\{\,totalFail\,\} \preceq \underbrace{[toSend \leq 4] \cdot (totalFail + 1) + [toSend > 4] \cdot \infty}_{5\text{-inductive}}$$

Proving this by (1-)induction requires a non-trivial invariant synthesis.

Instead of a stronger proof, use a stronger proof method.

k-Induction - Boolean Formulation

Given: TS $= (S, I, T)$, invariant property $P \subseteq S$

Goal: Prove that P covers all reachable states of TS

If there is $k \geq 1$ such that the following two formulae are valid

$$\underbrace{I(s_1) \wedge \mathsf{T}(s_1, s_2) \wedge \ldots \wedge \mathsf{T}(s_{k-1}, s_k)}_{\text{all states reachable within } k \text{ steps}} \implies \underbrace{P(s_1) \wedge \ldots \wedge P(s_k)}_{\text{are } P\text{-states}}$$

initialisation

$$\underbrace{P(s_1) \wedge \mathsf{T}(s_1, s_2) \wedge \ldots \wedge P(s_k)}_{\text{assuming we stay in } P \text{ for } k \text{ steps,}} \wedge \underbrace{\mathsf{T}(s_k, s_{k+1})}_{\text{after step } k+1,} \implies \underbrace{P(s_{k+1})}_{\text{we end up in } P \text{ again}}$$

induction

then P is a k-inductive invariant covering all reachable states of TS.

k-Induction: Probabilistic Formulation

Let $\mathsf{TS} = (S, I, T)$ and $P \subseteq S$. Define $\Phi \colon 2^S \to 2^S$ on the complete lattice $(2^S, \subseteq)$ by

$$\Phi(F) = I \cup \mathsf{Succs}(F) \,. \qquad \text{Then:} \quad \mathsf{Reach}(\mathsf{TS}) = \mathsf{lfp}\ \Phi$$

Goal: Prove $\mathsf{lfp}\ \Phi \subseteq P$. Now assume $\Phi^k(\emptyset) \subseteq P$.

$$\underbrace{\underbrace{\underbrace{\underbrace{P(s_1) \wedge \mathsf{T}(s_1, s_2)}_{\Phi(P)} \wedge P(s_2)}_{\Phi(P) \cap P} \wedge \mathsf{T}(s_2, s_3)}_{\Phi(\Phi(P) \cap P)} \wedge P(s_3)}_{\Phi(\Phi(P) \cap P) \cap P} \wedge \mathsf{T}(s_3, s_4) \implies P(s_4)$$

k-Induction: Probabilistic Formulation

Let $\mathsf{TS} = (S, I, T)$ and $P \subseteq S$. Define $\Phi \colon 2^S \to 2^S$ on the complete lattice $(2^S, \subseteq)$ by

$$\Phi(F) = I \cup \mathrm{Succs}(F) \,. \qquad \text{Then:} \quad \mathrm{Reach}(\mathsf{TS}) = \mathrm{lfp}\ \Phi$$

Goal: Prove $\mathrm{lfp}\ \Phi \subseteq P$. Now assume $\Phi^k(\emptyset) \subseteq P$.

Define $\Psi_P \colon 2^S \to 2^S$ by

$$\boxed{\Psi_P(F) = \Phi(F) \cap P \,.}$$

$$\underbrace{\underbrace{\underbrace{\underbrace{P(s_1) \wedge \mathsf{T}(s_1, s_2)}_{\Phi(P)} \wedge P(s_2)}_{\Phi(P) \cap P = \Psi_P(P)} \wedge \mathsf{T}(s_2, s_3)}_{\Phi(\Phi(P) \cap P) = \Phi(\Psi_P(P))} \wedge P(s_3)}_{\Phi(\Phi(P) \cap P) \cap P = \Psi_P^2(P)} \wedge \mathsf{T}(s_3, s_4) \implies P(s_4)$$

k-Induction: Probabilistic Formulation

Let TS $= (S, I, T)$ and $P \subseteq S$. Define $\Phi \colon 2^S \to 2^S$ on the complete lattice $(2^S, \subseteq)$ by

$$\Phi(F) = I \cup \mathrm{Succs}(F) \,. \qquad \text{Then:} \quad \mathrm{Reach}(\mathrm{TS}) = \mathrm{lfp}\ \Phi$$

Goal: Prove $\mathrm{lfp}\ \Phi \subseteq P$. Now assume $\Phi^k(\emptyset) \subseteq P$.

Define $\Psi_P \colon 2^S \to 2^S$ by

$$\Psi_P(F) = \Phi(F) \cap P \,.$$

$$\underbrace{\underbrace{\underbrace{\underbrace{\underbrace{P(s_1) \wedge \mathsf{T}(s_1, s_2)}_{\Phi(P)} \wedge P(s_2)}_{\Phi(P) \cap P = \Psi_P(P)} \wedge \mathsf{T}(s_2, s_3)}_{\Phi(\Phi(P) \cap P) = \Phi(\Psi_P(P))} \wedge P(s_3)}_{\Phi(\Phi(P) \cap P) \cap P = \Psi_P^2(P)} \wedge \mathsf{T}(s_3, s_4)}_{\Phi(\Psi_P^{k-1}(P))} \implies P(s_4)$$

meaning $\quad \Phi(\Psi_P^{k-1}(P)) \subseteq P$

k-Induction: The Boolean Setting

Let TS $= (S, I, T)$ and $P \subseteq S$. Define $\Phi \colon 2^S \to 2^S$ on the complete lattice $(2^S, \subseteq)$ by

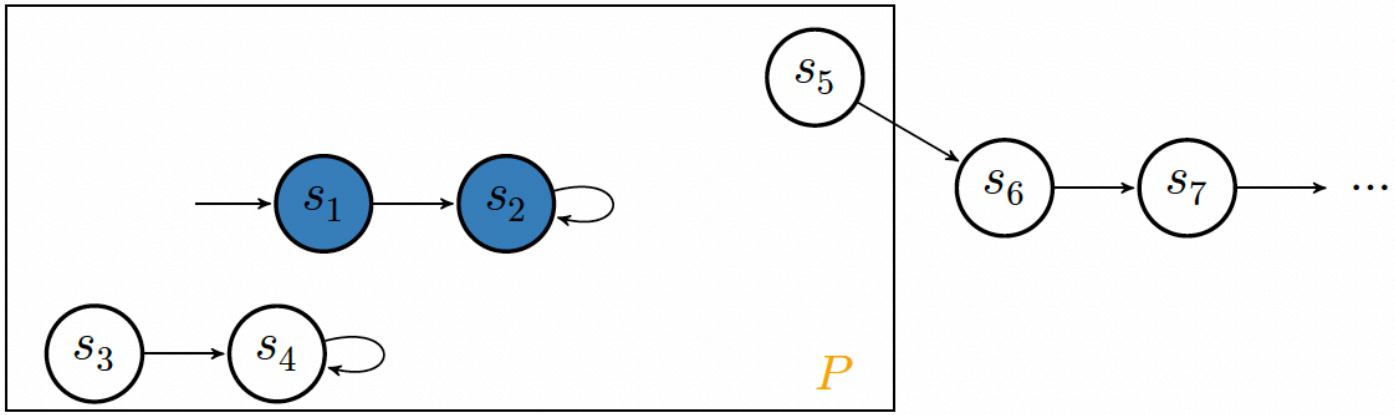
$$\Phi(F) = I \cup \mathrm{Succs}(F) \,. \qquad \text{Then:} \quad \mathrm{Reach}(\mathrm{TS}) = \mathrm{lfp}\ \Phi$$

Goal: Prove $\mathrm{lfp}\ \Phi \subseteq P$.
Define $\Psi_P \colon 2^S \to 2^S$ by

$$\Psi_P(F) = \Phi(F) \cap P \,.$$

If there is $k \geq 1$ such that

$$\Phi\left(\Psi_P^{k-1}(P)\right) \subseteq P, \qquad \text{then} \quad \mathrm{lfp}\ \Phi \subseteq P \,.$$



k-Induction: The Boolean Setting

Let TS $= (S, I, T)$ and $P \subseteq S$. Define $\Phi \colon 2^S \to 2^S$ on the complete lattice $(2^S, \subseteq)$ by

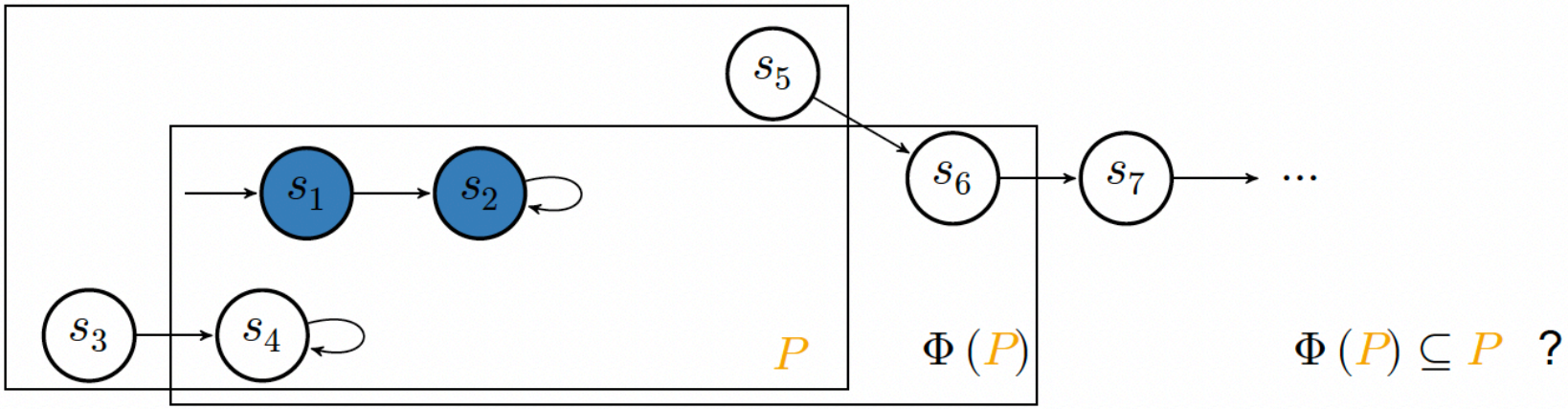
$$\Phi(F) = I \cup \mathrm{Succs}(F) \,. \qquad \text{Then:} \quad \mathrm{Reach}(\mathrm{TS}) = \mathrm{lfp}\ \Phi$$

Goal: Prove $\mathrm{lfp}\ \Phi \subseteq P$.
Define $\Psi_P \colon 2^S \to 2^S$ by

$$\Psi_P(F) = \Phi(F) \cap P \,.$$

If there is $k \geq 1$ such that

$$\Phi\left(\Psi_P^{k-1}(P)\right) \subseteq P, \qquad \text{then} \quad \mathrm{lfp}\ \Phi \subseteq P \,.$$



k-Induction: The Boolean Setting

Let TS $= (S, I, T)$ and $P \subseteq S$. Define $\Phi \colon 2^S \to 2^S$ on the complete lattice $(2^S, \subseteq)$ by

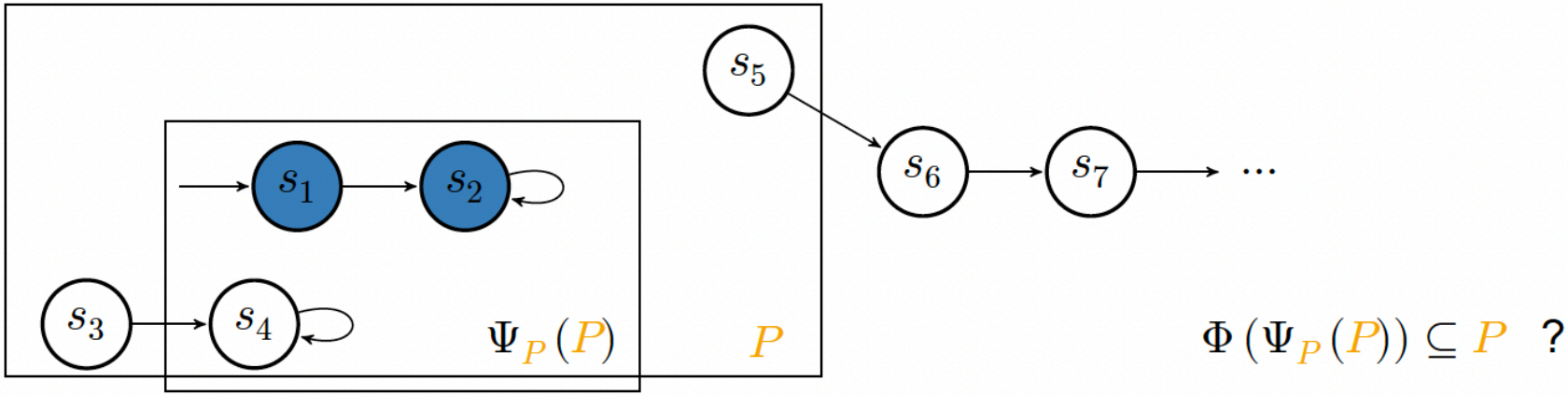
$$\Phi(F) = I \cup \mathrm{Succs}(F) \,. \qquad \text{Then:} \quad \mathrm{Reach}(\mathrm{TS}) = \mathrm{lfp}\ \Phi$$

Goal: Prove $\mathrm{lfp}\ \Phi \subseteq P$.
Define $\Psi_P \colon 2^S \to 2^S$ by

$$\Psi_P(F) = \Phi(F) \cap P \,.$$

If there is $k \geq 1$ such that

$$\Phi\left(\Psi_P^{k-1}(P)\right) \subseteq P, \qquad \text{then} \quad \mathrm{lfp}\ \Phi \subseteq P \,.$$



Generalisation to Lattices

Let $(E, \sqsubseteq)$ be a complete lattice. Furthermore, let $\Phi \colon E \to E$ be monotonic and $f \in E$.

Goal: Prove lfp $\Phi \sqsubseteq f$.

Define $\Psi_f \colon E \to E$ by

$$\Psi_f(g) = \Phi(g) \sqcap f\,.$$

Theorem (Latticed k-Induction)

For every $k \geq 1$,

$$\Phi\left(\Psi_f^{k-1}(f)\right) \sqsubseteq f \quad \textit{implies} \quad \text{lfp}\ \Phi \sqsubseteq f\,.$$

We call such f a k-inductive invariant.

Notice: k-Induction generalizes Park induction $\triangleq$ 1-induction!

k-Induction vs. Inductive Invariants

Theorem (Park Induction from k-Induction)

$$\underbrace{\Phi\left(\Psi_f^{k-1}(f)\right) \sqsubseteq f}_{f \text{ is } k\text{-inductive invariant}} \quad \textit{iff} \quad \underbrace{\Phi\left(\Psi_f^{k-1}(f)\right) \sqsubseteq \Psi_f^{k-1}(f)}_{\Psi_f^{k-1}(f) \text{ is inductive invariant}} .$$

Lemma

Iterating Ψ_f on f yields a descending chain, i.e.,

$$f \sqsupseteq \Psi_f(f) \sqsupseteq \Psi_f^2(f) \sqsupseteq \Psi_f^3(f) \sqsupseteq \dots .$$

Hence, if f is k-inductive invariant, then

- $\Psi_f^{k-1}(f)$ is a (1-)inductive invariant,
- which is stronger than f.

k-Induction-Fixed Point Formulation for Boolean Setting

Let $\mathsf{TS} = (S, I, T)$ and $P \subseteq S$. Define $\Phi \colon 2^S \to 2^S$ on the complete lattice $(2^S, \subseteq)$ by

$$\Phi(F) = I \cup \mathsf{Succs}(F) \,. \qquad \text{Then:} \quad \mathsf{Reach}(\mathsf{TS}) = \mathsf{lfp}\ \Phi$$

Goal: Prove $\mathsf{lfp}\ \Phi \subseteq P$.

$$\underbrace{\underbrace{\underbrace{I(s_1)}_{\Phi(\emptyset)} \wedge \mathsf{T}(s_1, s_2)}_{\Phi^2(\emptyset)} \wedge \mathsf{T}(s_2, s_3)}_{\Phi^3(\emptyset)} \implies P(s_1) \wedge P(s_2) \wedge P(s_3)$$

$$\text{meaning} \quad \Phi^k(\emptyset) \subseteq P$$

k-Induction- Probabilistic Formulation

In order to reason about reachability probabilities in MDPs, we need to:

... leave the Boolean domain and reason about quantities

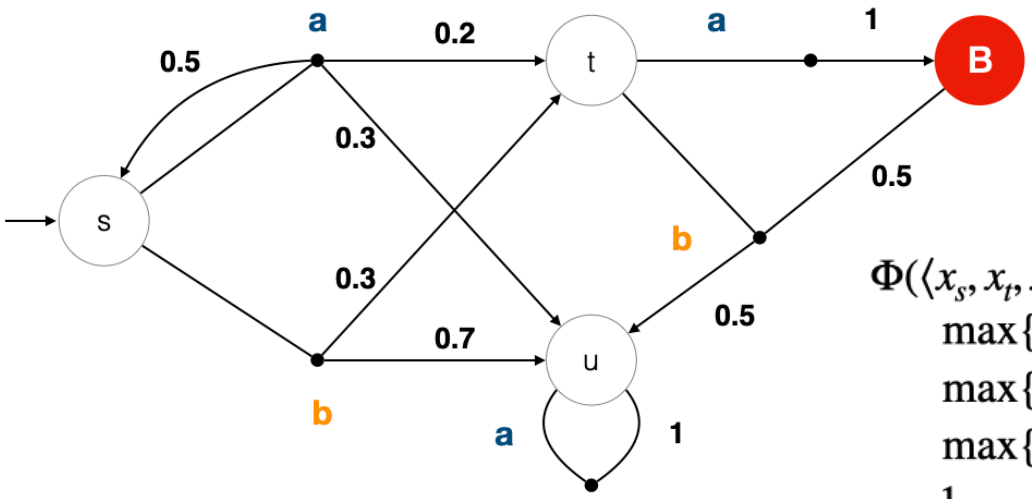
... reason about sets of paths rather than individual paths

MDP Frame Transformer := Bellman Operator

Definition 2 (Bellman Operator). *For a set of actions* $A \subseteq \mathrm{Act}$, *we define the* Bellman operator for A *as a frame transformer* $\Phi_A : [0,1]^S \to [0,1]^S$ *with*

$$\Phi_A(F)[s] = \begin{cases} 1, & \text{if } s \in B \\ \max\limits_{a \in A} \sum\limits_{s' \in S} P(s,a,s') \cdot F[s'], & \text{if } s \notin B\,. \end{cases}$$

bad states



$$\Phi(\langle x_s, x_t, x_u, x_B \rangle) =$$
$$\max\{0.5 \cdot x_s + 0.2 \cdot x_t + 0.3 \cdot x_u,\ 0.3 \cdot x_t + 0.7 \cdot x_u\}$$
$$\max\{x_B,\ 0.5 \cdot x_B + 0.5 \cdot x_B\}$$
$$\max\{x_u\}$$
$$1$$

Experimental Results

peak number of conjuncts

comp. time SMT formulae

SMT solving

total time (incl.preprocessing

infinite-state

	postexpectation	variant	result	k	#formulae	formulae_t	sat_t	total_t
brp	*totalFail*	1	ind	5	285	0.15	0.01	0.28
		2	ind	11	2812	1.77	0.12	2.03
		3	ind	23	26284	17.68	28.09	45.94
		4	TO	–	–	–	–	–
		5	ref	13	949	0.84	14.39	15.28
		6	TO	–	–	–	–	–
		7	TO	–	–	–	–	–
geo	c	1	ind	2	18	0.01	0.00	0.08
		2	ref	11	103	0.04	0.01	0.09
		3	ref	46	1223	0.39	0.04	0.48
rabin	$[i = 1]$	1	ind	1	21	0.01	0.00	0.15
		2	ind	5	1796	1.27	0.03	1.44
		3	TO	–	–	–	–	–
		4	ref	4	458	0.31	0.03	0.40
		5	ref	8	10508	8.76	2.85	11.68
unif_gen	$[c = i]$	1	ind	2	267	0.27	0.02	0.56
		2	ind	3	1402	1.45	0.10	1.81
		3	ind	3	1402	1.48	0.11	1.86
		4	ind	5	40568	47.31	15.70	63.28
		5	TO	–	–	–	–	–

https://github.com/moves-rwth/kipro2.

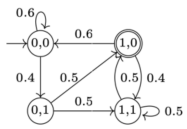
Tutorial Overview

1. 

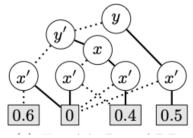
   Fundamentals of Probabilistic Model Checking

2. 

   Probabilistic Model Checking with Storm: Hands-on Slides

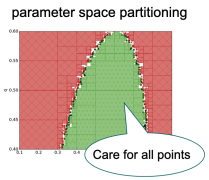
3. 

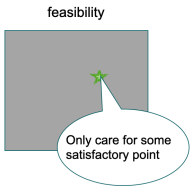
   Example Markov chain



   Its MTBDD (x, x') to (y, y')

   Automated Symbolic Reasoning

4. 



   Parameter Synthesis in Markov Models

Parameter Synthesis in Markov Models

Sebastian Junges, Joost-Pieter Katoen

Radboud Universiteit

UNIVERSITY OF TWENTE.

Software Modeling and Verification Chair

RWTH AACHEN UNIVERSITY

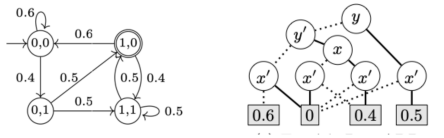
Tutorial Overview

1. 

   Fundamentals of
   Probabilistic Model Checking

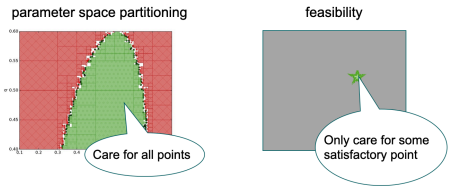
2. 

   Probabilistic Model Checking with Storm:
   Hands-on Slides

3. 

   Example Markov chain Its MTBDD (x, x') to (y, y')

   Automated Symbolic Reasoning

4. 

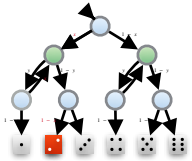
   Parameter Synthesis in Markov Models

Take-home messages

- Parametric Markov chains (pMCs) & what questions to ask about them
- pMCs and finite-state controllers in POMDPs are tightly connected
- pMC analysis as a backbone; e.g., for parameter tuning in Bayesian networks

Overview

Concepts



Encoding

$0 < x < 1 0 < y < 1$

$p_6 = 1$

$p_5 = 0 \quad p_4 = 0 \quad p_3 = 0$

$p_4 = x \cdot p_3 + (1-x) \cdot p_6$

$p_3 = x \cdot p_2 + (1-x) \cdot p_5$

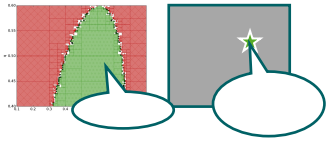
$p_2 = y \cdot p_3 + (1-y) \cdot p_4$

$p_1 = x \cdot x_2 + (1-x) \cdot p_5$

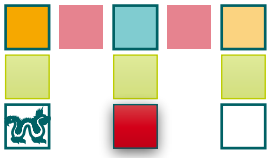
$p_1 > 1/6$

Complexity

Approaches



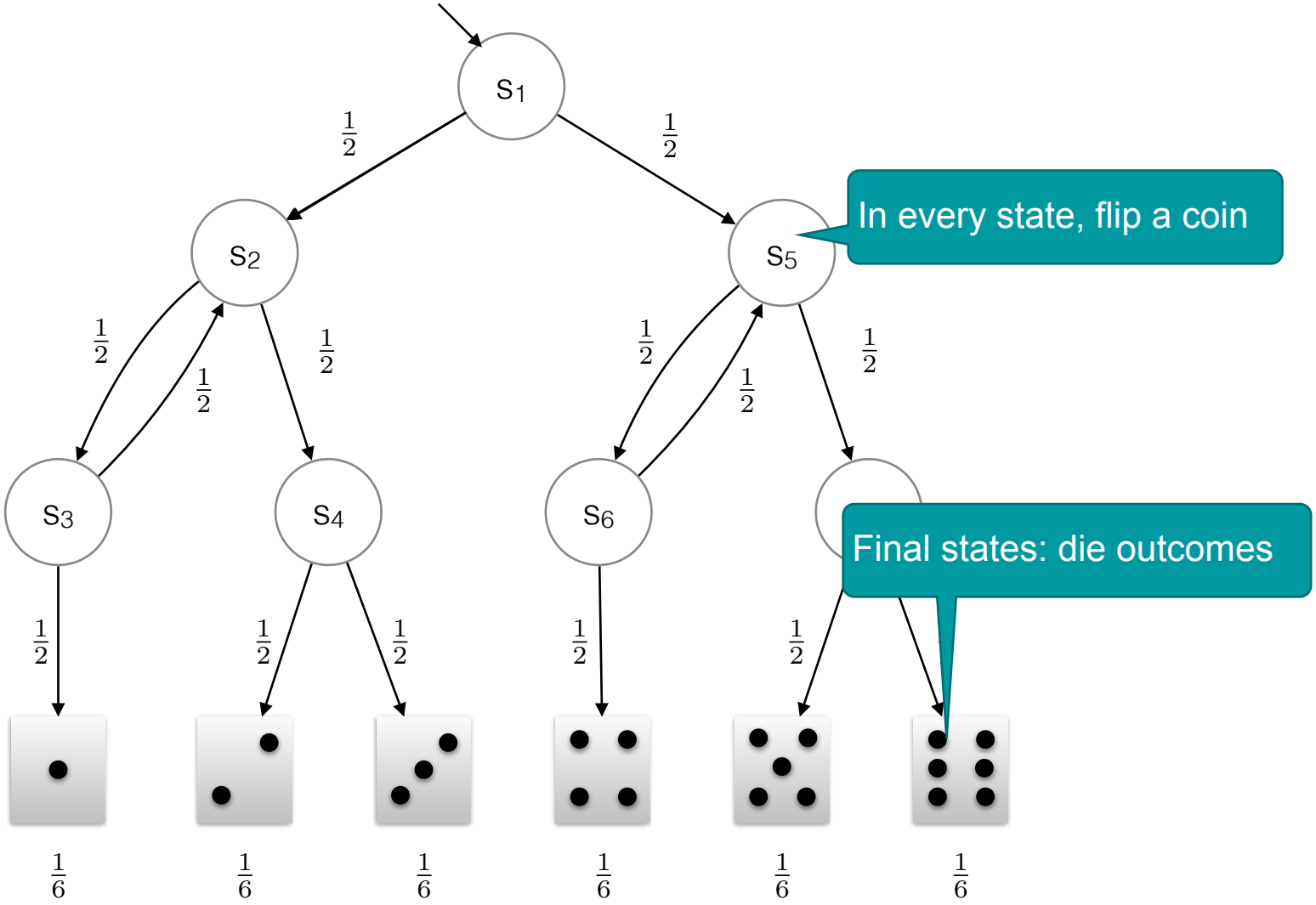
POMDPs



Parametric BNs

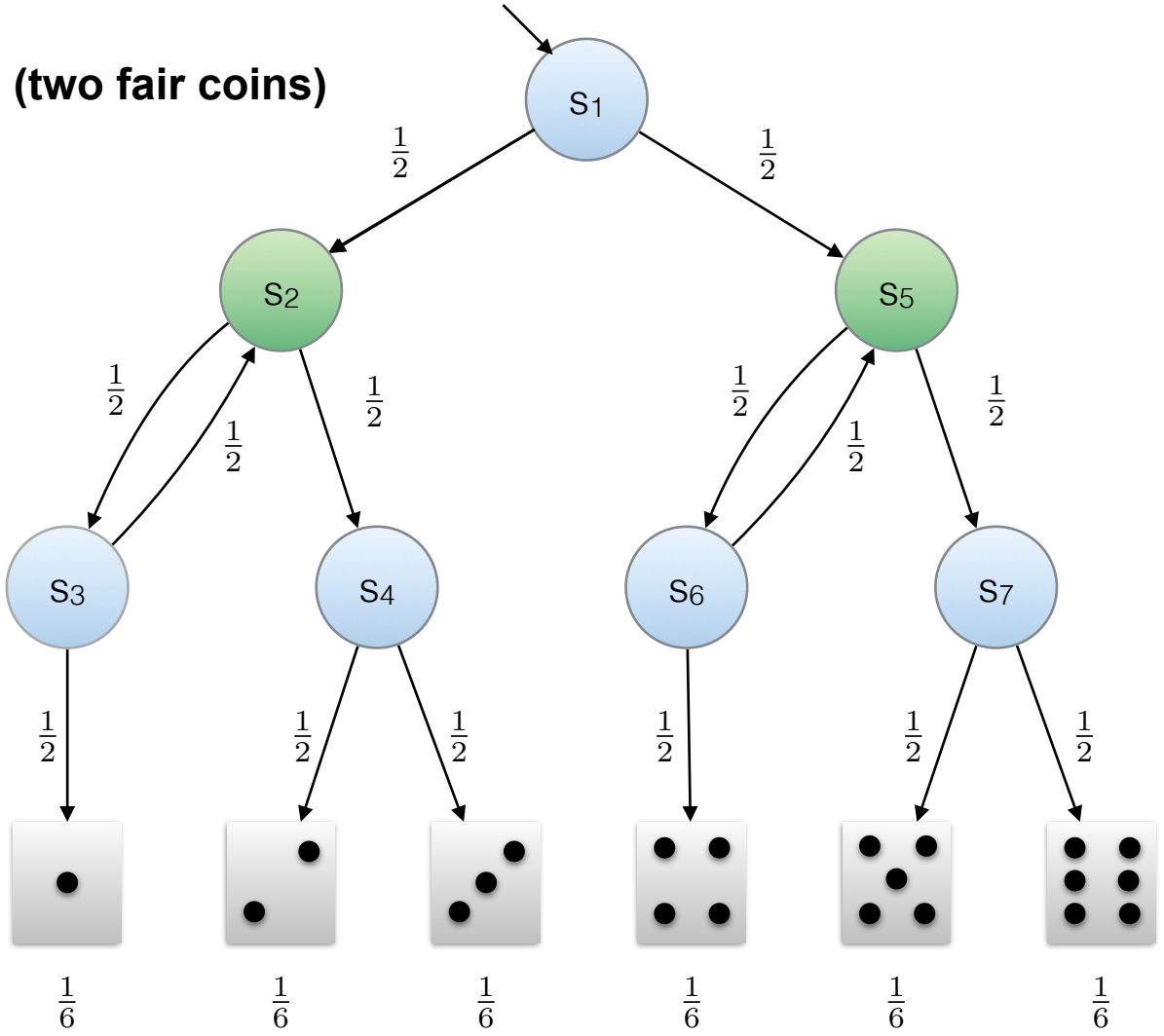
Markov chains

Knuth-Yao die



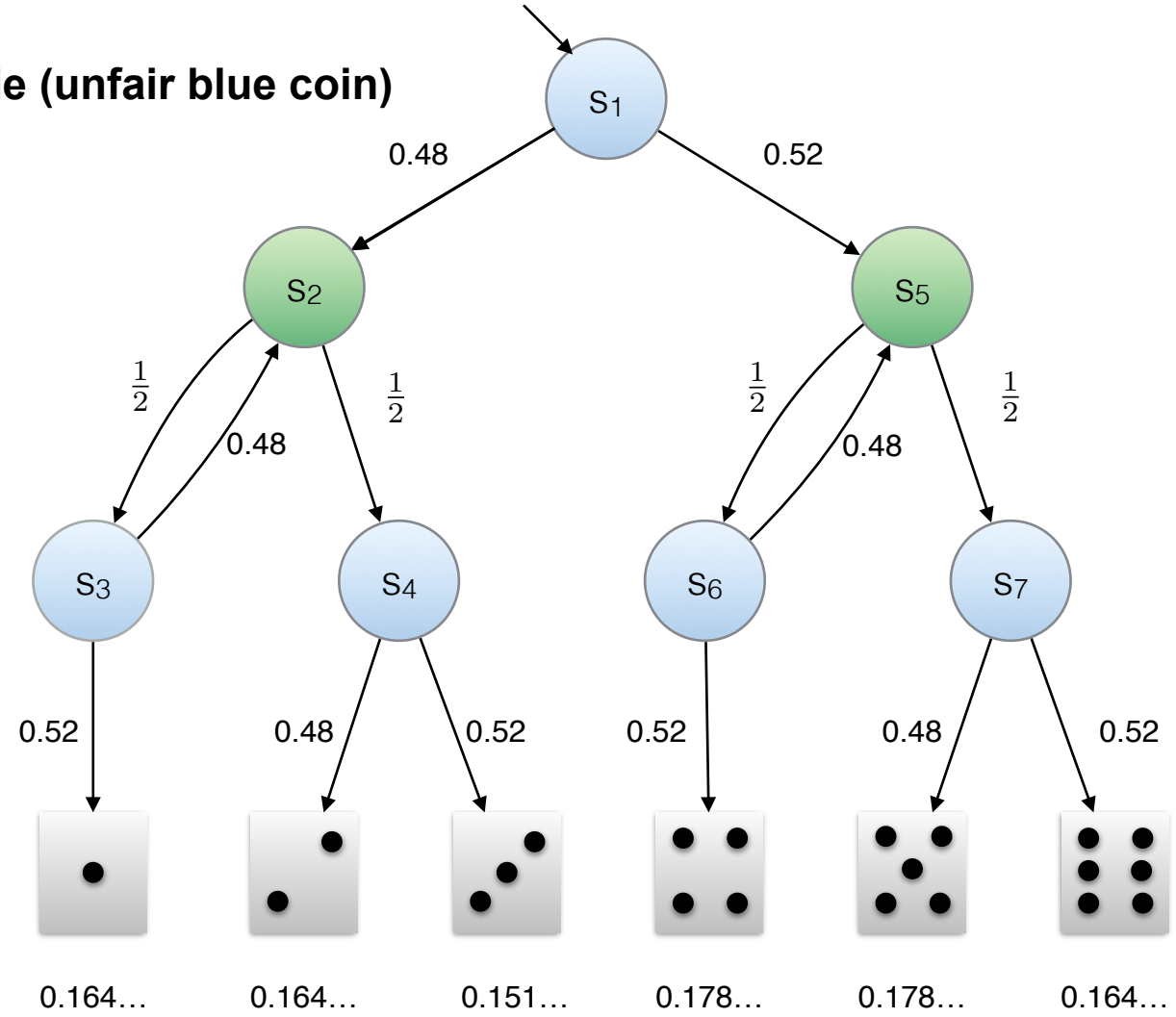
Markov chains

Knuth-Yao die (two fair coins)



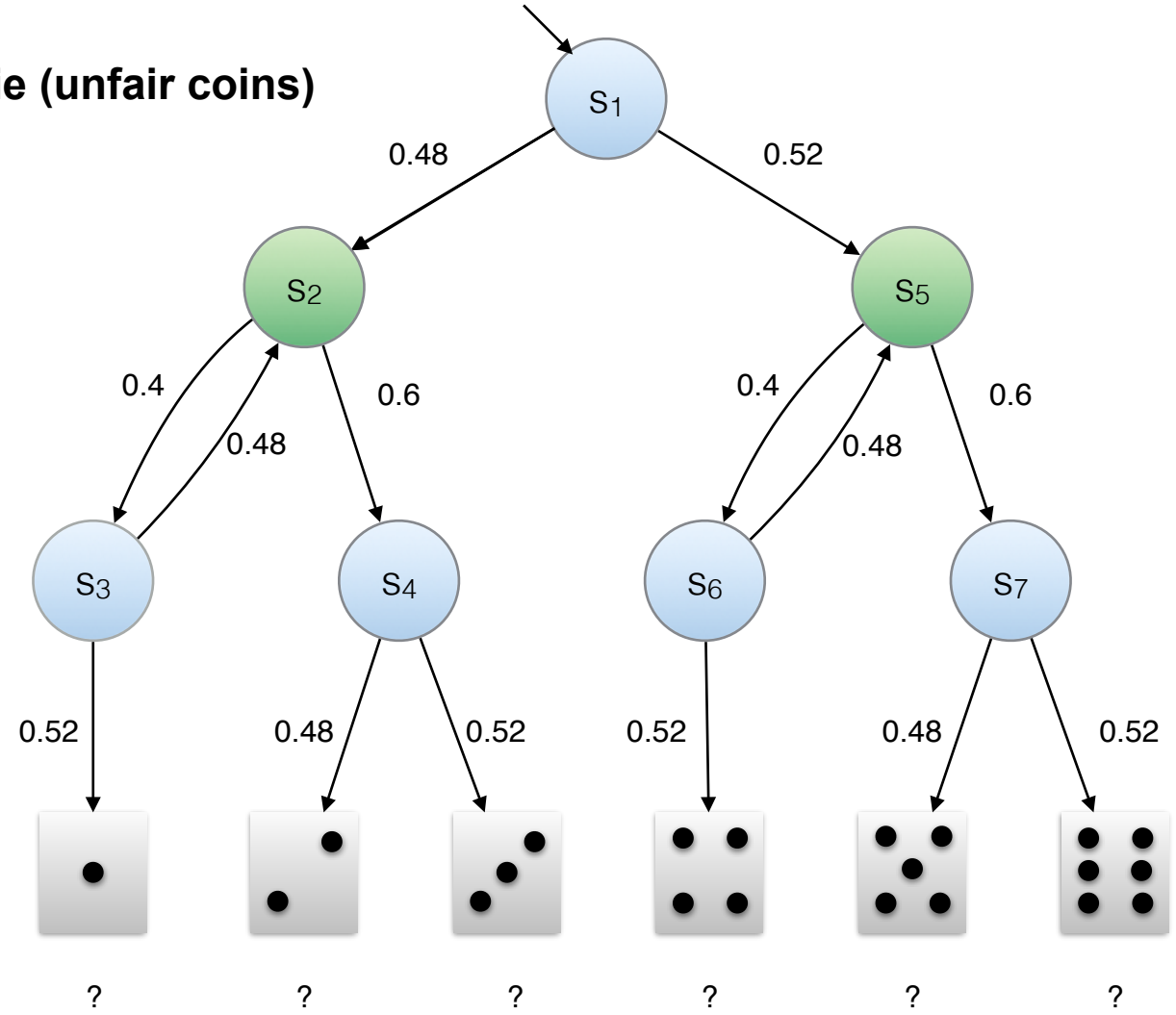
Markov chains

Knuth-Yao die (unfair blue coin)



Markov chains

Knuth-Yao die (unfair coins)



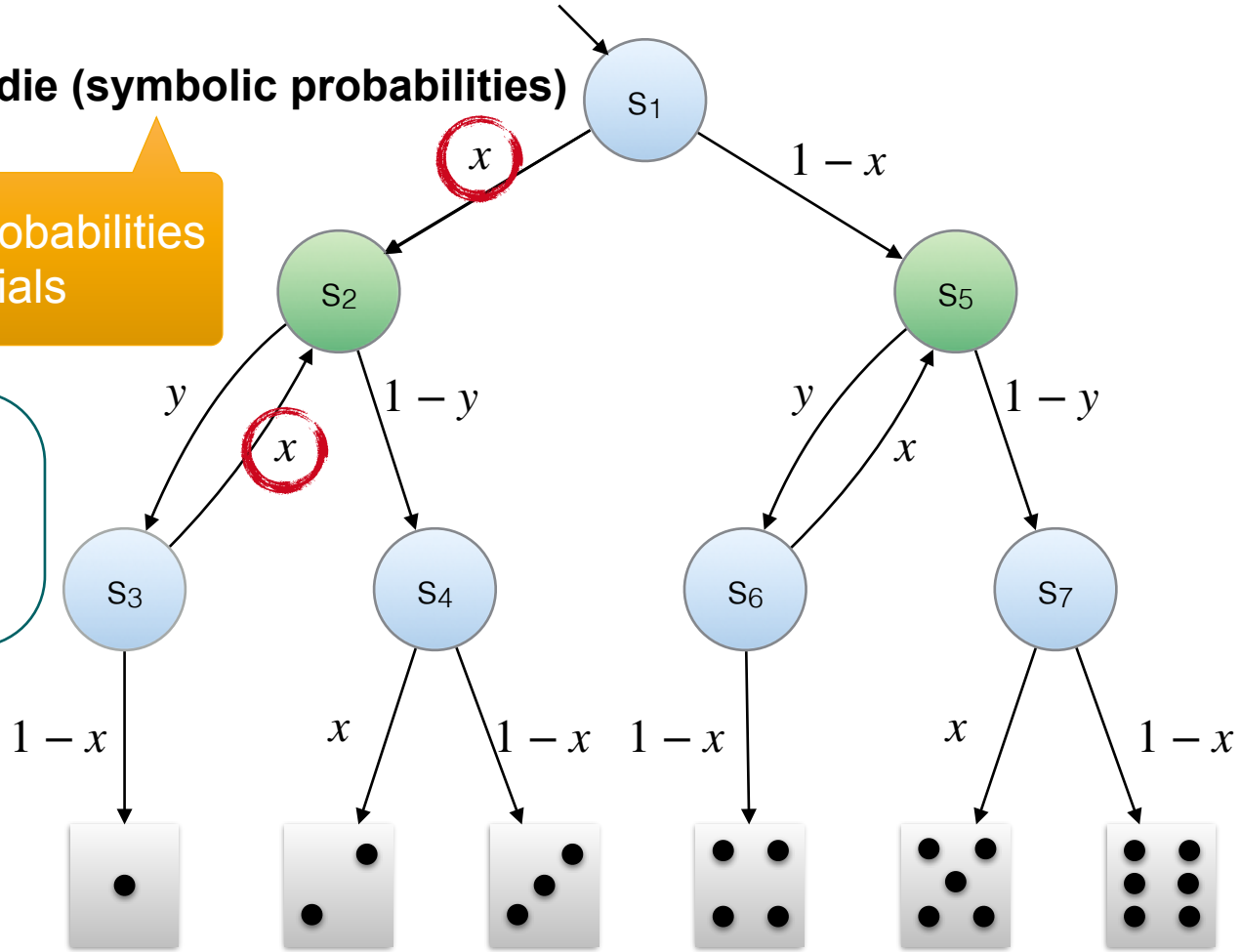
Parametric Markov chains (pMCs)

Knuth-Yao die (symbolic probabilities)

Transition probabilities are polynomials

Unless mentioned otherwise:

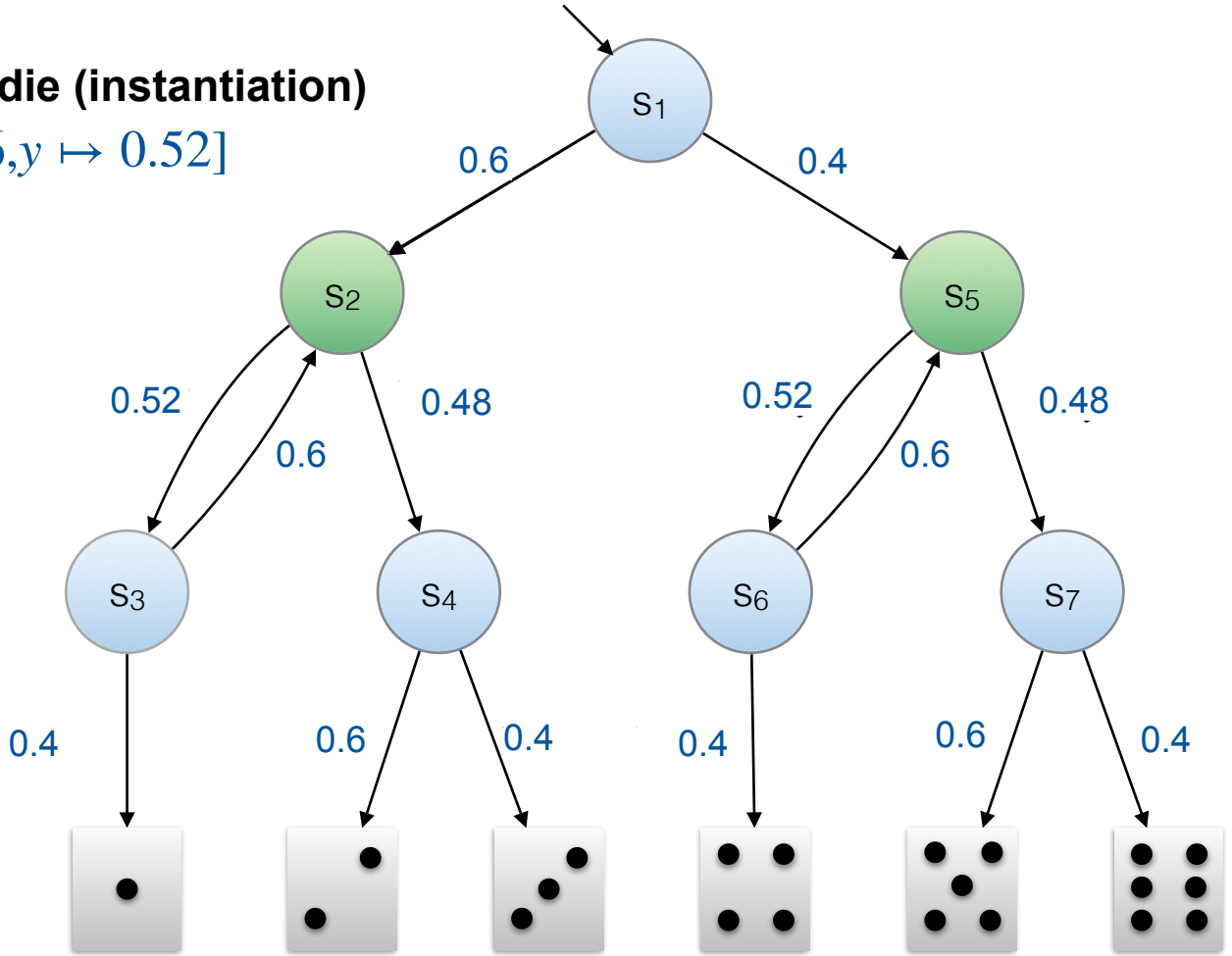
$\{x, 1-x\}$



Parametric Markov chains (pMCs)

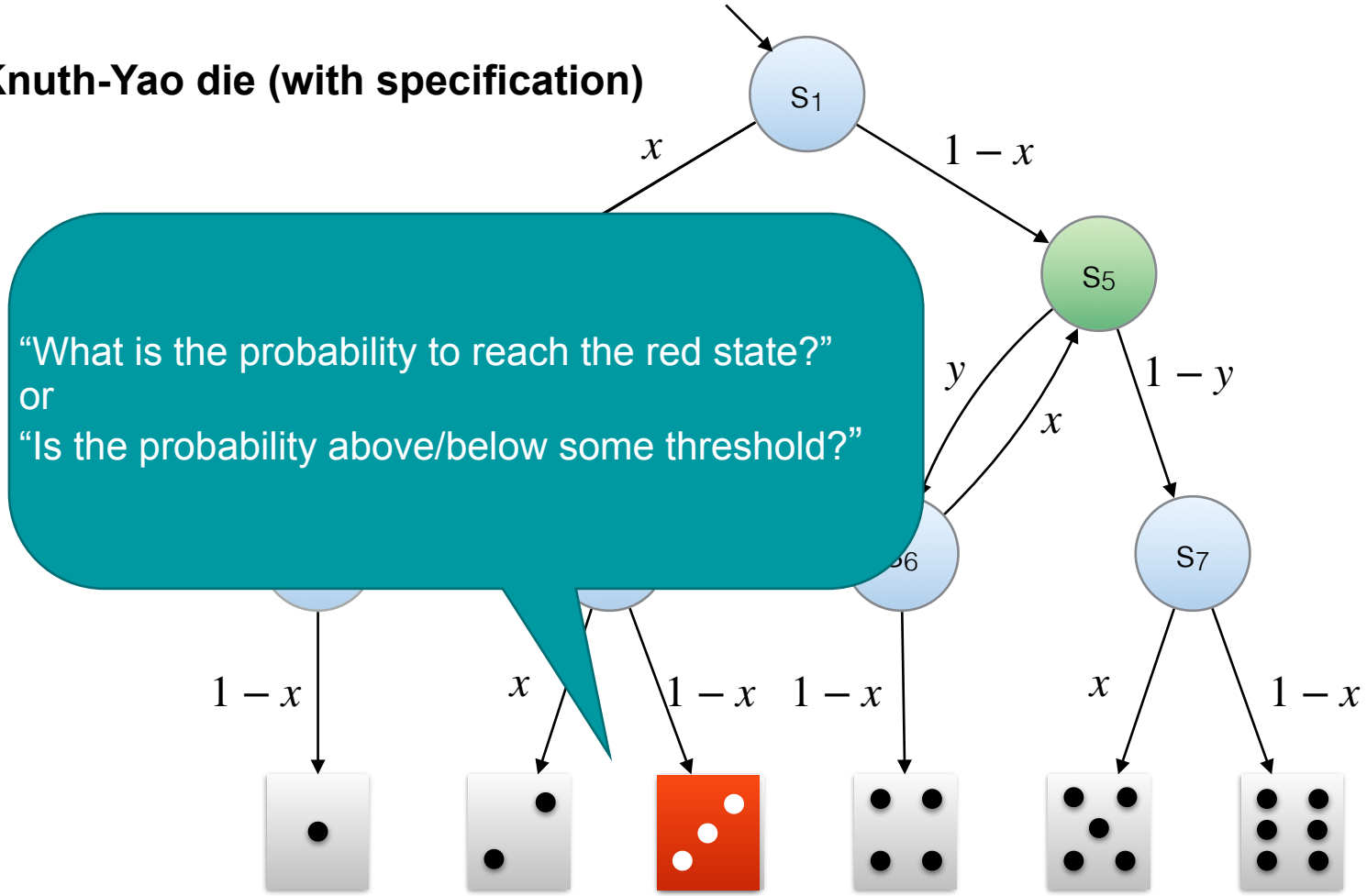
Knuth-Yao die (instantiation)

$\mathscr{M}[x \mapsto 0.6, y \mapsto 0.52]$



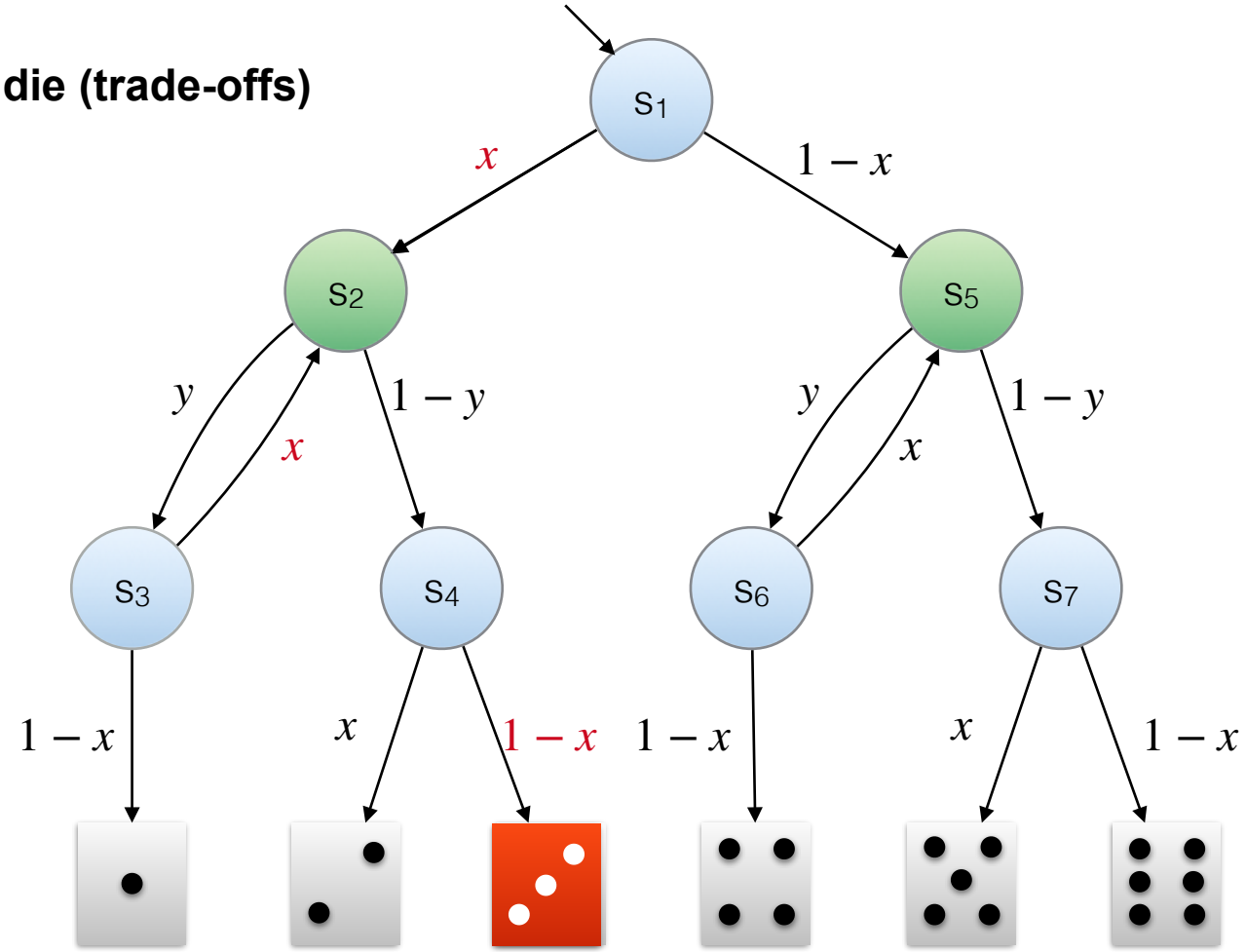
Parametric Markov chains (pMCs)

Knuth-Yao die (with specification)



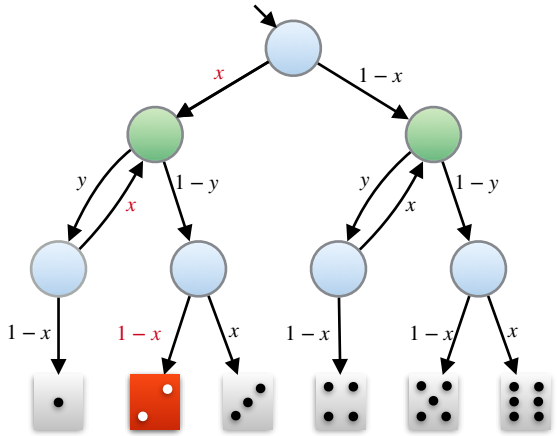
Parametric Markov chains (pMCs)

Knuth-Yao die (trade-offs)



Problem statement: Parameter synthesis

Given:



a parametric MC $\mathscr{M}$
with parameters $\mathbf{x}$

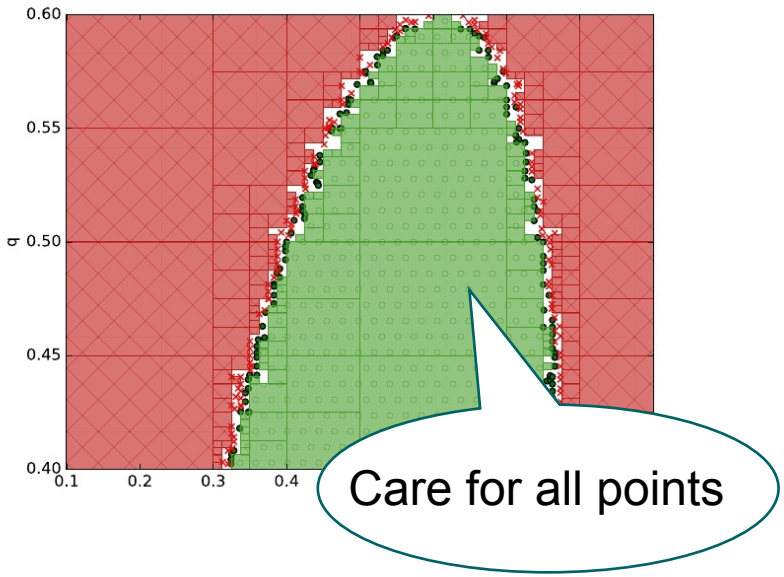
Find: $\text{val}: \mathbf{x} \to [0,1]$

such that: $\mathscr{M}[\text{val}] \models \varphi$, i.e., a red state is reached with probability at least/at most λ

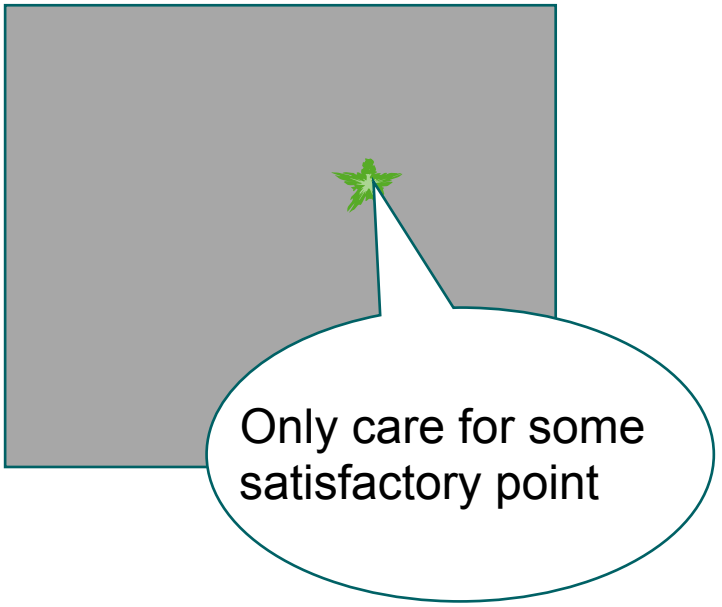
Parameter Synthesis

Various settings

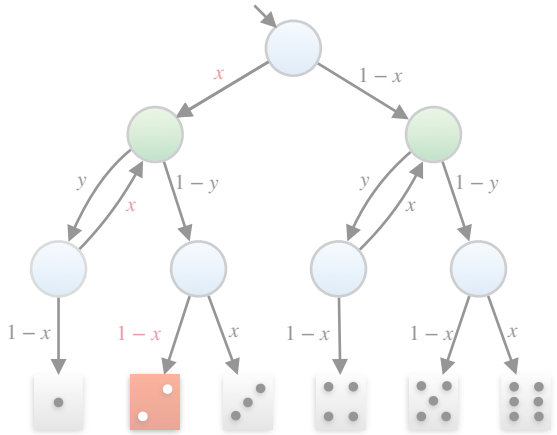
parameter space partitioning



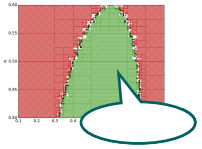
feasibility



Problem statement: Parameter synthesis



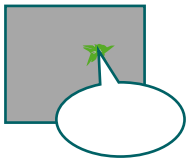
Given: a parametric MC $\mathscr{M}$ with parameters $\mathbf{x}$



all/ many

Find: $\mathrm{val}: \mathbf{x} \to [0,1]$

some

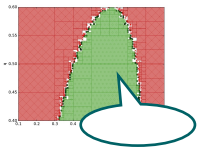


such that: $\mathscr{M}[\mathrm{val}] \models \varphi$, i.e., a red state is reached with probability at least/at most λ

Two types of motivation

Uncontrollable parameters

- “robustness”
  probabilities in environment are only estimates

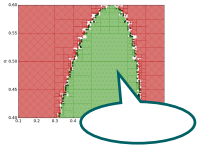


- “effectiveness”
  existence of scenarios that justify redundancy

Controllable parameters

- Randomised algorithms
  to break symmetry in distributed protocols, or
  to maximise entropy

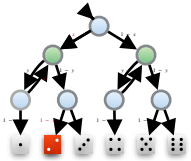
- System configuration, product lines
  E.g, use of higher quality components, or
  use of additional redundancy



- Small strategies
  for partially observable MDPs

Overview

Concepts



Encoding

$0 < x < 1, 0 < y < 1$

$p_6 = 1$

$p_5 = 0 \quad p_4 = 0 \quad p_3 = 0$

$p_4 = x \cdot p_5 + (1-x) \cdot p_6$

$p_3 = x \cdot p_2 + (1-x) \cdot p_4$

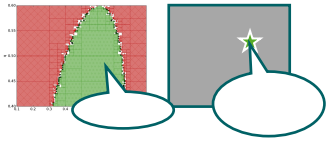
$p_2 = y \cdot p_3 + (1-y) \cdot p_4$

$p_1 = x \cdot x_2 + (1-x) \cdot p_5$

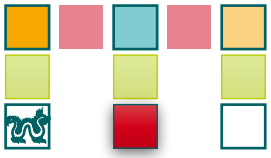
$p_1 > 1/6$

Complexity

Approaches



POMDPs

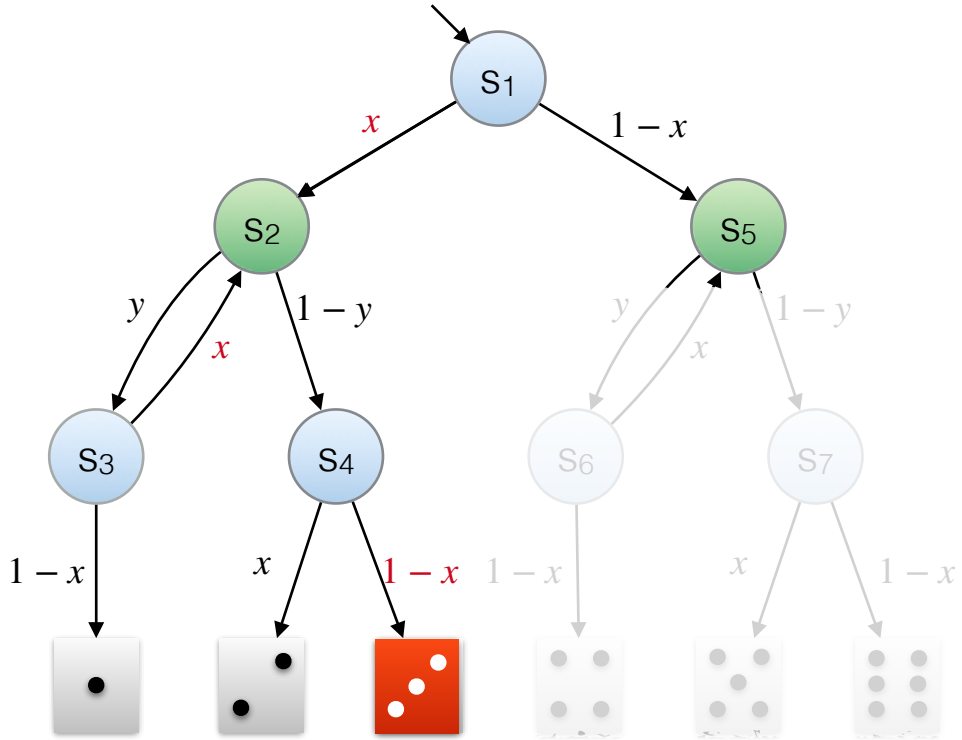


Parametric BNs

Encoding feasibility in Existential Theory of the Reals (ETR)

Does a valuation exist s.t. a red state is reached with probability is more than 1/6?

yes, iff the constraints are satisfiable



$$\exists p_i \exists x, y :$$

$$0 < x < 1,\ 0 < y < 1$$

$$p_{\text{⚅}} = 1$$

$$p_5 = 0 \quad p_{\text{⚀}} = 0 \quad p_{\text{⚁}} = 0$$

$$p_4 = x \cdot p_{\text{⚀}} + (1 - x) \cdot p_{\text{⚅}}$$

$$p_3 = x \cdot p_2 + (1 - x) \cdot p_{\text{⚀}}$$

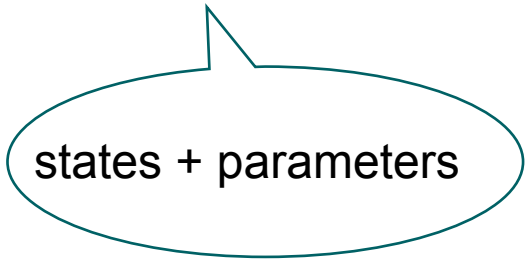
$$p_2 = y \cdot p_3 + (1 - y) \cdot p_4$$

$$p_1 = x \cdot x_2 + (1 - x) \cdot p_5$$

$$p_1 > 1/6$$

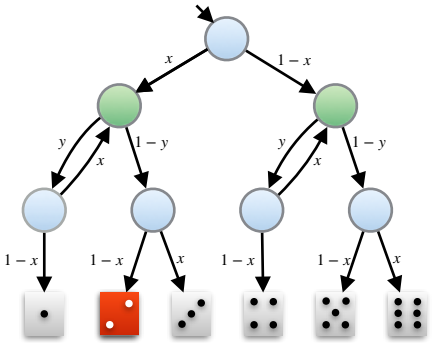
Efficiency?

Solving systems of polynomials — in general — is exponential in number of variables



Eliminating state-variables to get a Solution Function

Results in a rational function $f(\mathbf{x})$ over the parameters $\mathbf{x}$



$$x \cdot (1-x) \cdot \frac{1-y}{1-xy}$$

State elimination (as in NFAs) or Gaussian elimination w/ polynomials

[Daws'04]
[Hahn et al.'11]
[Delgado et al.'11]
[Jansen et al.'14]
[CAV'2015]
[Hutschenreiter et al.'17]
[INFOCOMP'20]

For a pMC with k parameters, n states and linear polynomials as probabilities:

- The rational function can be exponential in k (even for acyclic pMCs)
- For any fixed k, the computation can be done in polynomial time in n

Result of state elimination

41 States - 138 Transitions - 2 Parameters

The numerator has 408 terms,

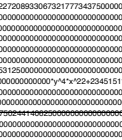
The denominator is the product of 48 linear polynomials

5 seconds

Sebastian Junges & Joost-Pieter Katoen

Probabilistic Verification

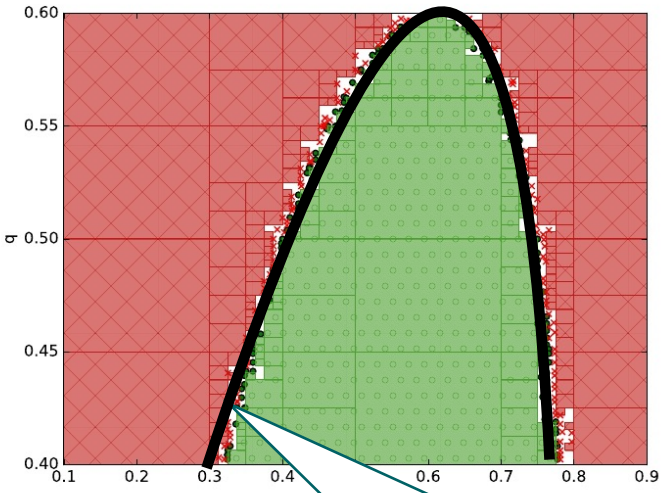
ICALP 2022



Radboud University

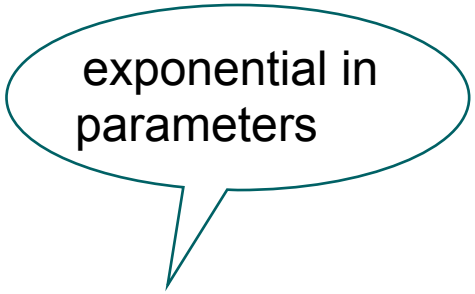
Exact Partitioning

Split R into $R_+ = \{ \text{val} \in R \mid \mathcal{M}[\text{val}] \models \varphi \}$ and $R_- = \{ \text{val} \in R \mid \mathcal{M}[\text{val}v] \not\models \varphi \}$

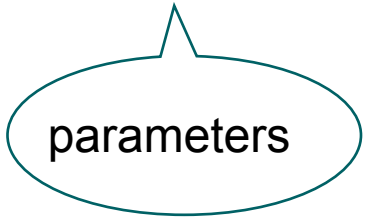


This curve is the solution function

Efficiency?

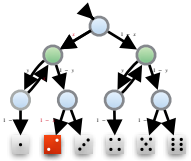


Solving polynomial inequality — in general — is exponential in number of variables



Overview

Concepts



Encoding

$0 < x < 1 0 < y < 1$

$p_6 = 1$

$p_5 = 0 \quad p_4 = 0 \quad p_3 = 0$

$p_4 = x \cdot p_5 + (1-x) \cdot p_6$

$p_3 = x \cdot p_2 + (1-x) \cdot p_4$

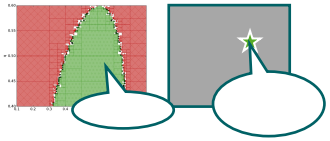
$p_2 = y \cdot p_3 + (1-y) \cdot p_4$

$p_1 = x \cdot x_2 + (1-x) \cdot p_5$

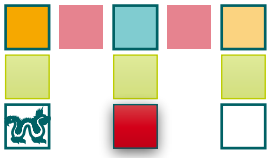
$p_1 > 1/6$

Complexity

Approaches

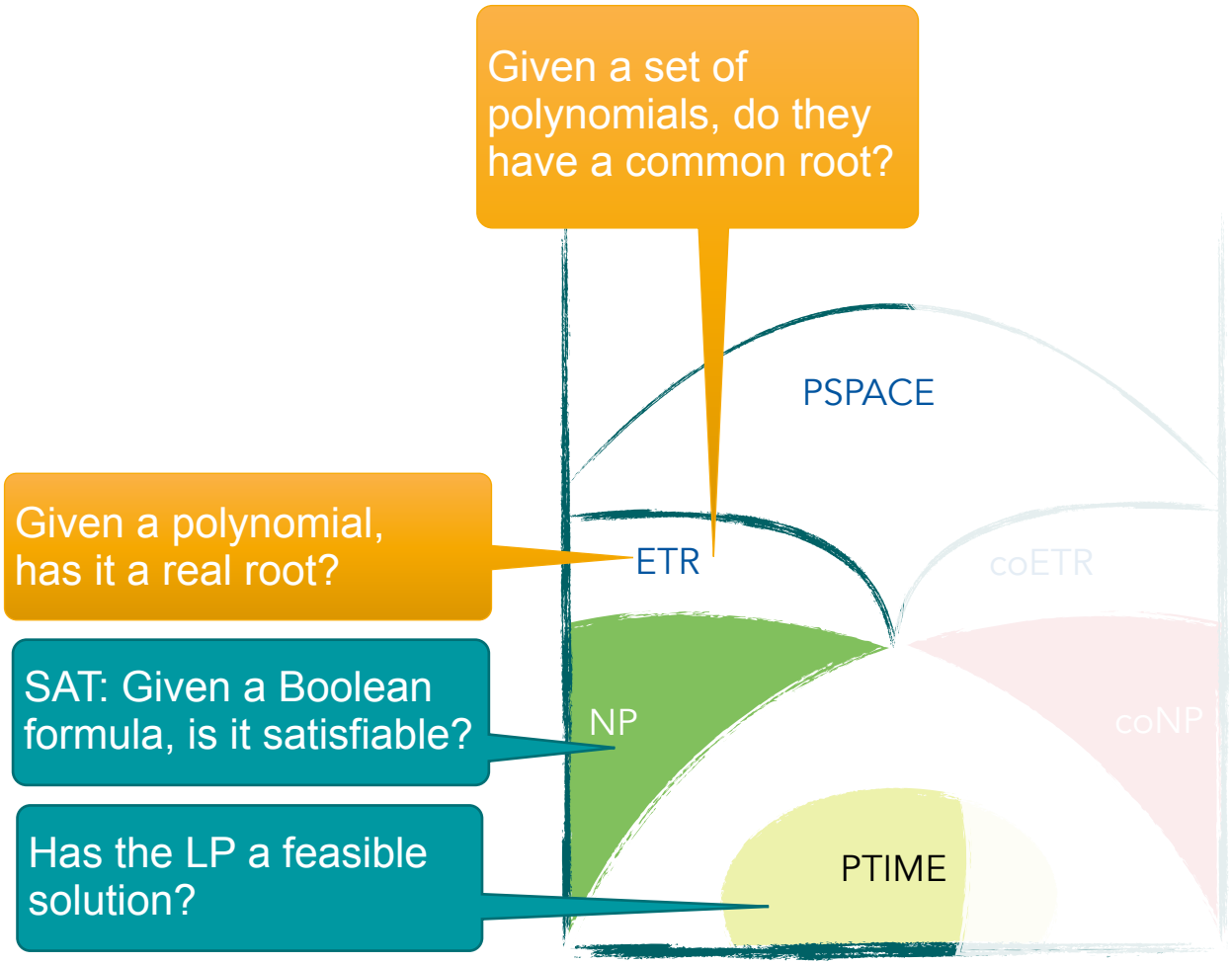


POMDPs

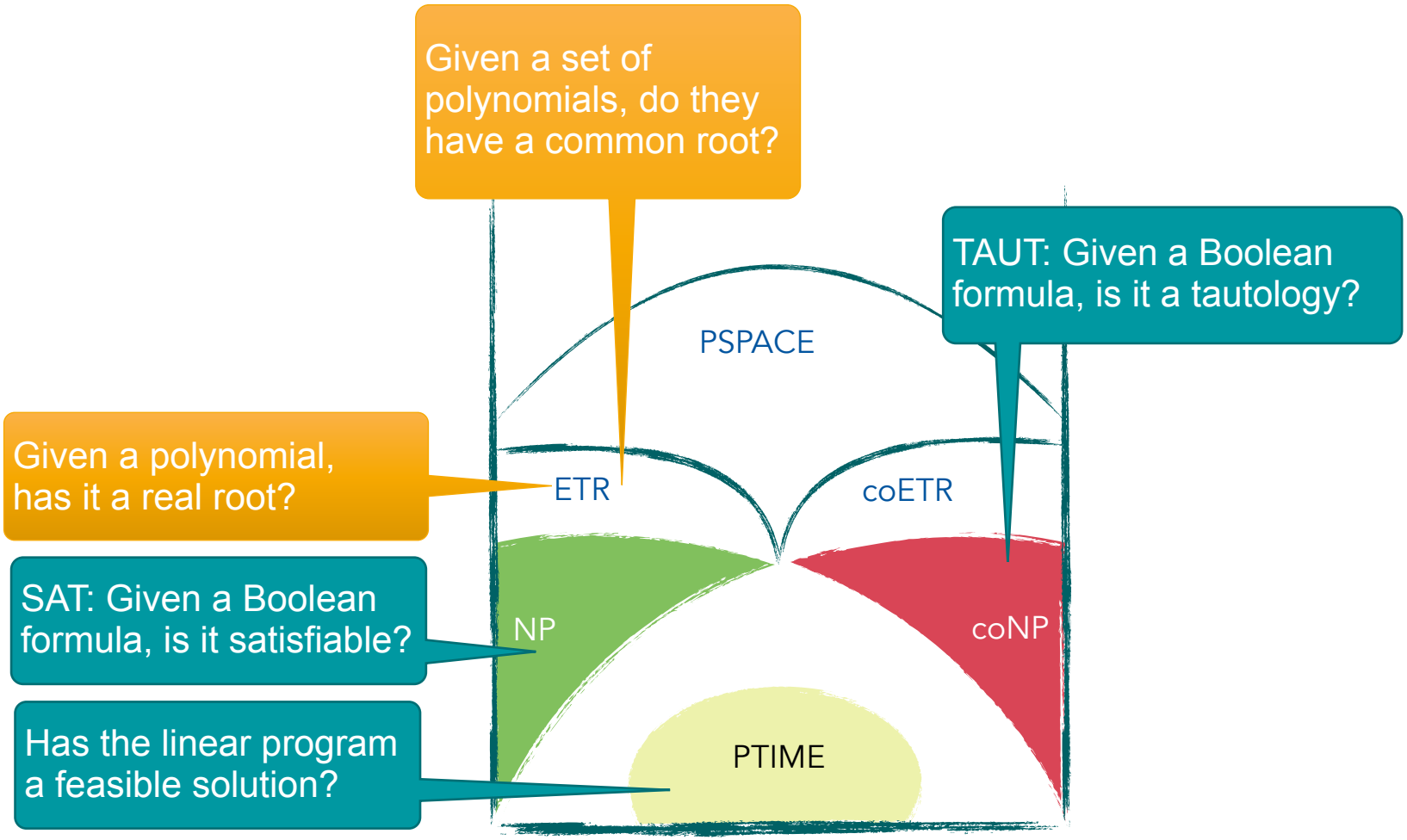


Parametric BNs

Recap: Complexity theory



Recap: Complexity theory

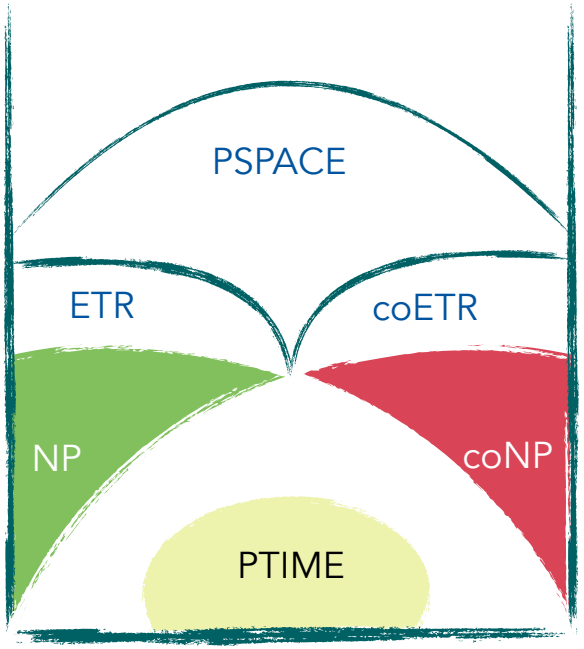


How difficult is parameter synthesis?

[CONCUR'19]

Given: a parametric MC $\mathcal{M}$ with parameters $\mathbf{x}$ **exists:** $\text{val}: \mathbf{x} \to [0,1]$ **s.t.:** in $\mathcal{M}[\text{val}]$ a red state is reached with probability [relation] λ

model	relation	
pMC	$\leq$ $\geq$	ETR-complete
	$<$ $>$	NP-hard in ETR



Encoding polynomial inequalities as pMC

[Chonev'19]
[JCSS'21]

ETR

Given any **polynomial** f
is there a variable valuation val
s.t. $f(\text{val}) \geq \kappa$

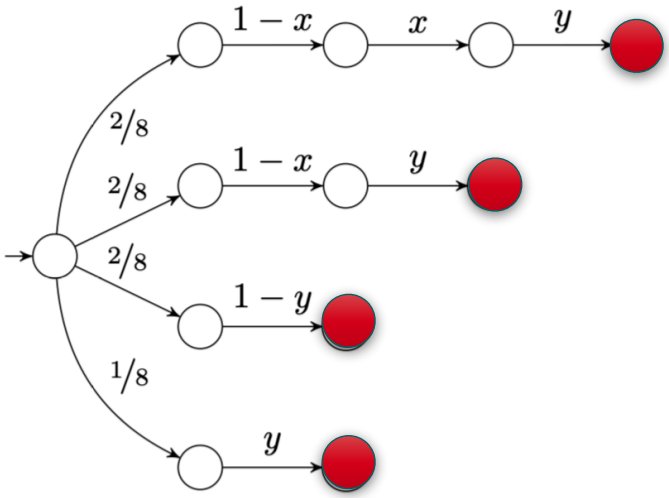
Given any **pMC**
is there a parameter valuation s.t.
the probability reaching ● $\geq \lambda$

$$-2x^2y + y \geqslant 5$$

$$2 \cdot \big((1-x)xy + (1-x)y + (1-y) - 1\big) + y \geqslant 5$$

$$\frac{2 \cdot (1-x)xy + 2 \cdot (1-x)y + 2 \cdot (1-y) + y}{8} \geqslant \frac{7}{8}$$

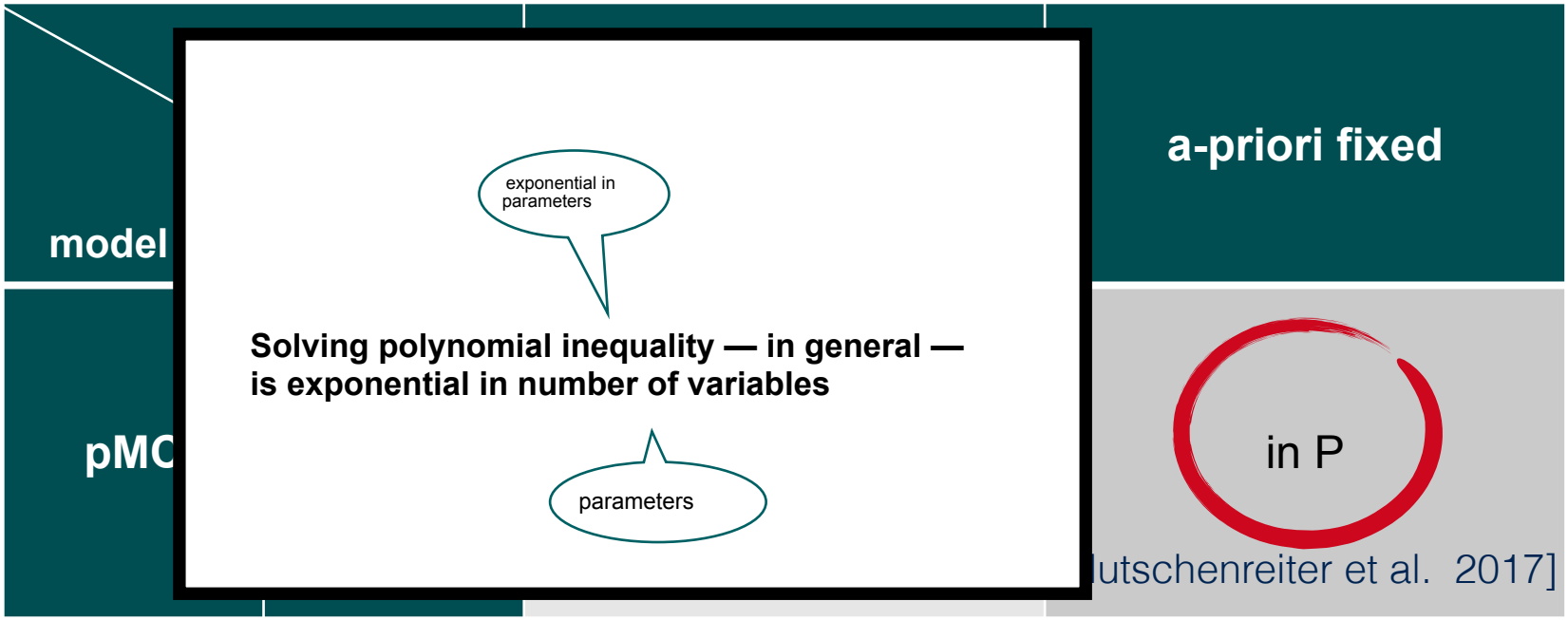
Probability of reaching ● at least 7/8



How difficult is parameter synthesis?

[JCSS'21]

Given: a parametric MC $\mathcal{M}$ with parameters $\mathbf{x}$ **exists:** $\text{val}: \mathbf{x} \to [0,1]$ **s.t.:** in $\mathcal{M}[\text{val}]$ a red state is reached with probability [relation] λ



What about parametric MDPs?

[JCSS'21]

Given: a parametric MDP $\mathcal{M}$ with parameters $\mathbf{x}$

Selecting an action in every state

exists: $\text{val} : \mathbf{x} \to [0,1]$ **such that for all** $\sigma : S \to \text{Act}$: $\mathcal{M}_\sigma[\text{val}] \models \varphi$

How difficult is parameter synthesis?

[JCSS'21]

The complexity landscape for parameter synthesis (simplified)

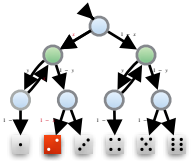
model \ parameters	relation	arbitrarily many	a-priori fixed
pMC	$\leq$ $\geq$	ETR-complete	in P [Hutschenreiter et al. 2017]
pMC	$<$ $>$	NP-hard in ETR	in P [Hutschenreiter et al. 2017]
pMDP	$<$ $\leq$ $>$ $\geq$	ETR-complete	in NP

ETR encoding as extension of the standard LP for MDPs

How to eliminate state variables?

Overview

Concepts



Encoding

$0 < x < 1 0 < y < 1$

$p_6 = 1$

$p_5 = 0 \quad p_4 = 0 \quad p_3 = 0$

$p_4 = x \cdot p_3 + (1-x) \cdot p_6$

$p_3 = x \cdot p_2 + (1-x) \cdot p_5$

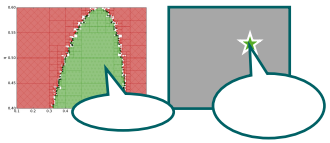
$p_2 = y \cdot p_3 + (1-y) \cdot p_4$

$p_1 = x \cdot x_2 + (1-x) \cdot p_5$

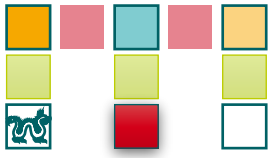
$p_1 > 1/6$

Complexity

Approaches

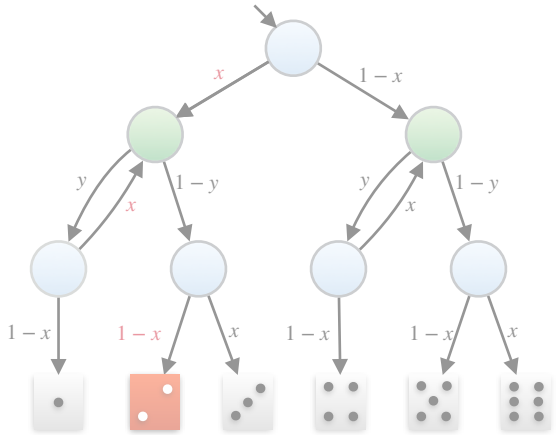


POMDPs

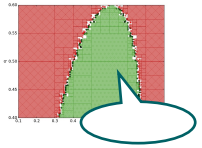


Parametric BNs

Problem statement: Parameter synthesis



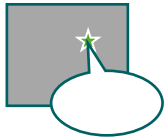
Given: a parametric MDP $\mathcal{M}$ with parameters $\mathbf{x}$



all/ many

Find: $\mathrm{val}: \mathbf{x} \to [0,1]$

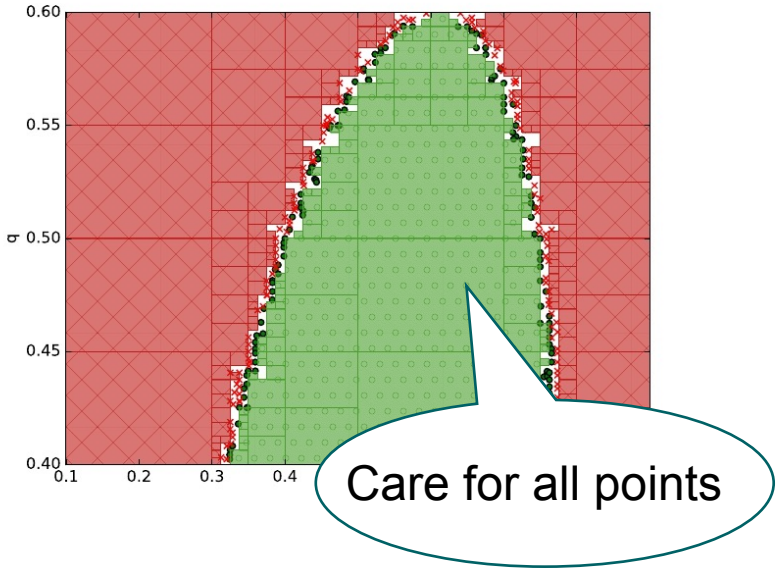
some

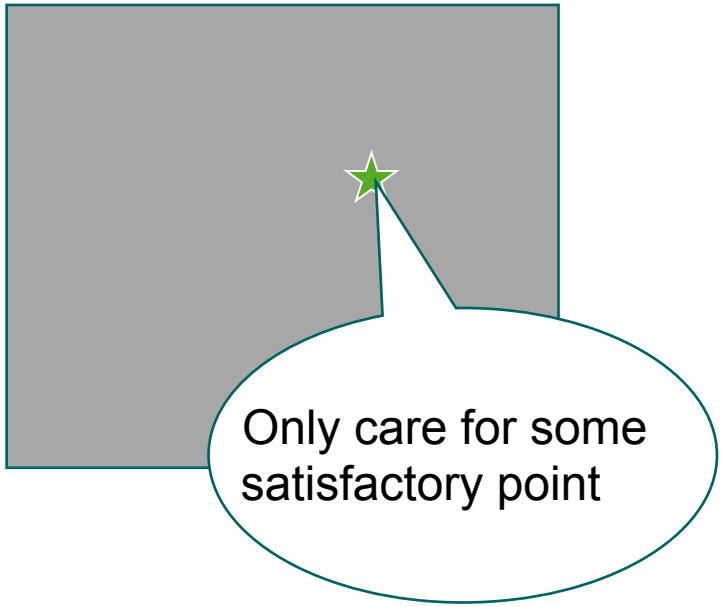


such that: $\mathcal{M}_\sigma[\mathrm{val}] \models \varphi$, i.e., a red state is reached with probability at least/at most λ

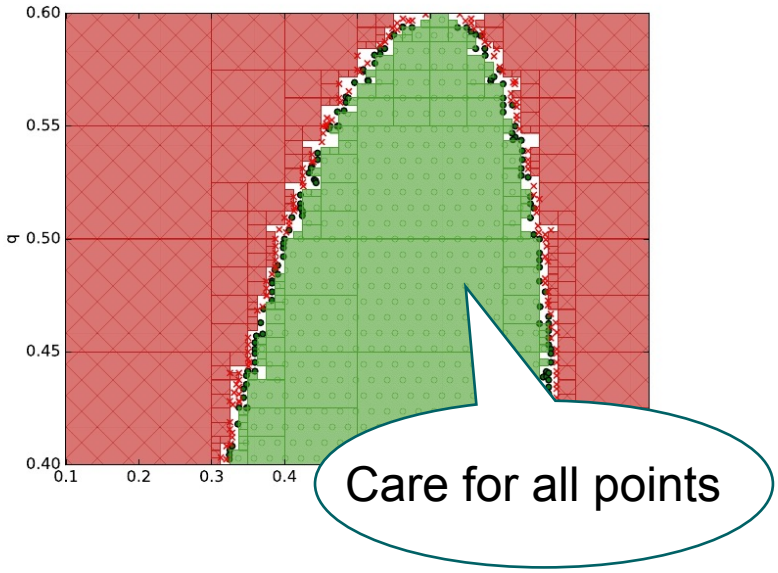
Practical parameter synthesis

Two settings





Practical Parameter Synthesis



Several variants of encoding via SMT solvers [CAV'15]

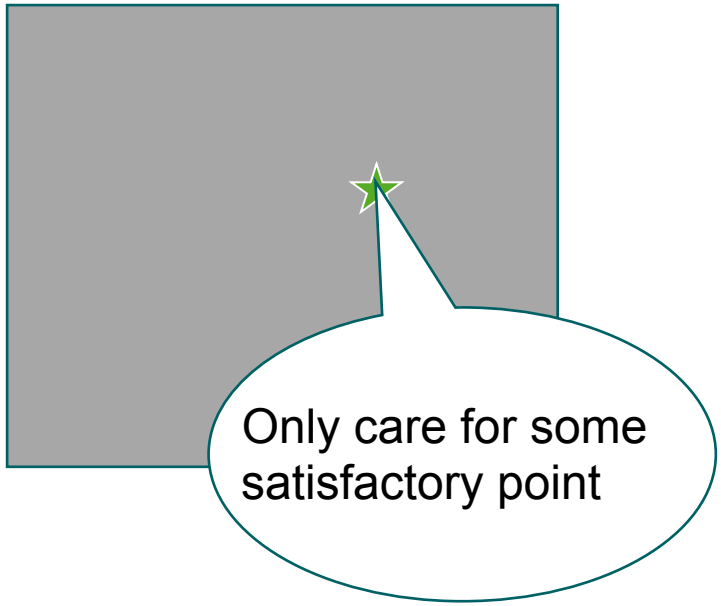
Parameter lifting: abstraction-refinement [ATVA'16]

surveyed in [arXiv'19]

Sampling based methods such as particle swarm [Chen et al.'14]

Iterative convex optimisation schemes [TACAS'17] [ATVA'18]

Practical Parameter Synthesis



Several variants of encoding via SMT solvers [CAV'15]

Parameter lifting: abstraction-refinement [ATVA'16]

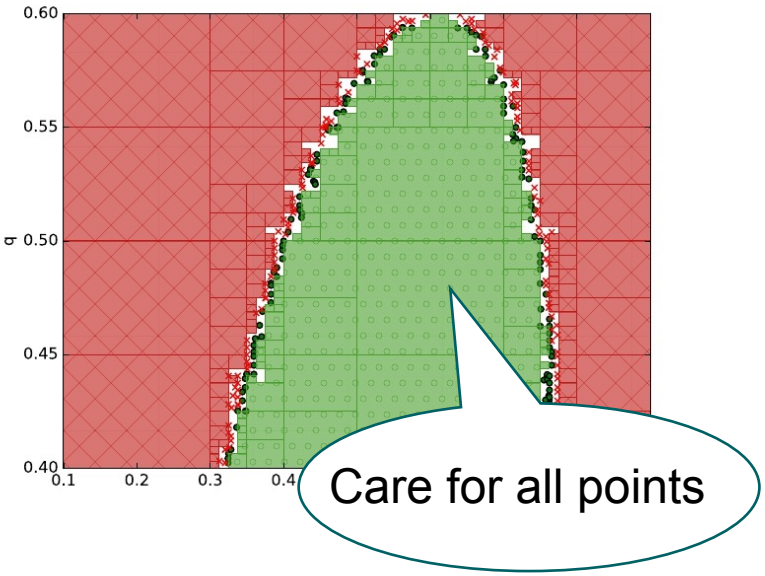
surveyed in [arXiv'19]

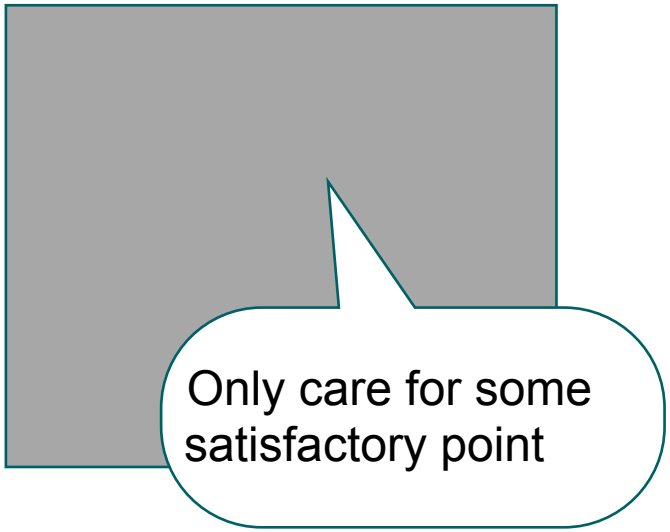
Sampling based methods such as particle swarm

[Chen et al.'14]

Iterative convex optimisation schemes [TACAS'17] [ATVA'18]

Practical Parameter Synthesis





Several variants of encoding via SMT solvers [CAV15]

Parameter lifting: abstraction-refinement [ATVA16]

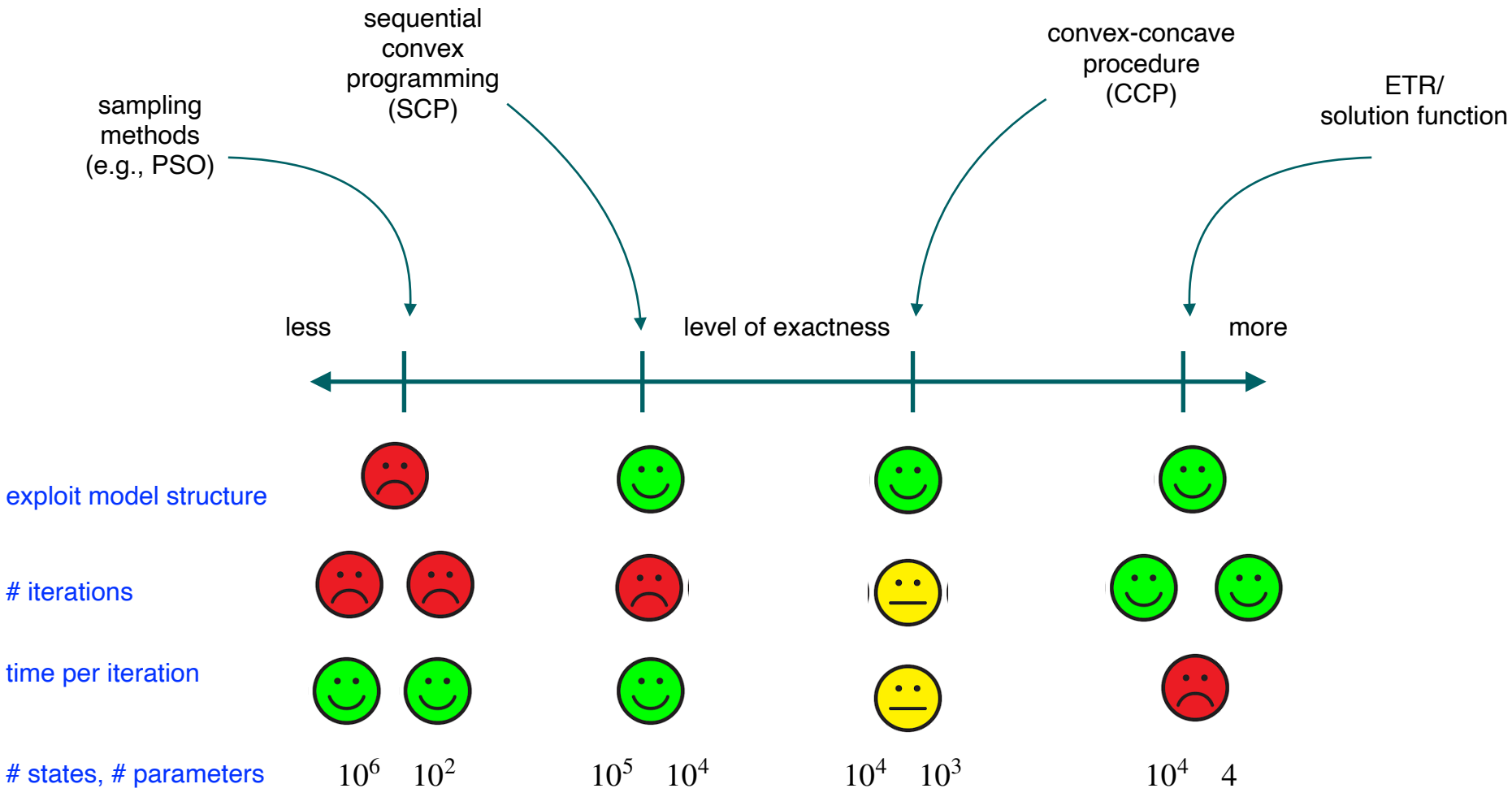
surveyed in [Arxiv19]

Sampling based methods such as particle swarm

[Chen et al.'14]

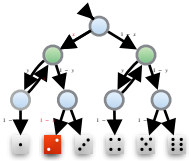
Iterative convex optimisation schemes [TACAS'17] [ATVA'18]

Practical Approaches to Feasibility



Overview

Concepts



Encoding

$0 < x < 1 0 < y < 1$

$p_6 = 1$

$p_5 = 0 \quad p_{\cdot} = 0 \quad p_{\cdot} = 0$

$p_4 = x \cdot p_{\cdot} + (1-x) \cdot p_{\cdot}$

$p_3 = x \cdot p_2 + (1-x) \cdot p_{\cdot}$

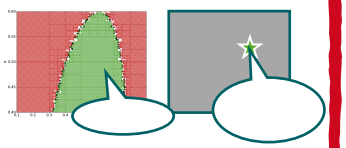
$p_2 = y \cdot p_3 + (1-y) \cdot p_4$

$p_1 = x \cdot x_2 + (1-x) \cdot p_5$

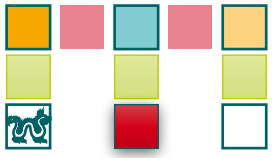
$p_1 > 1/6$

Complexity

Approaches



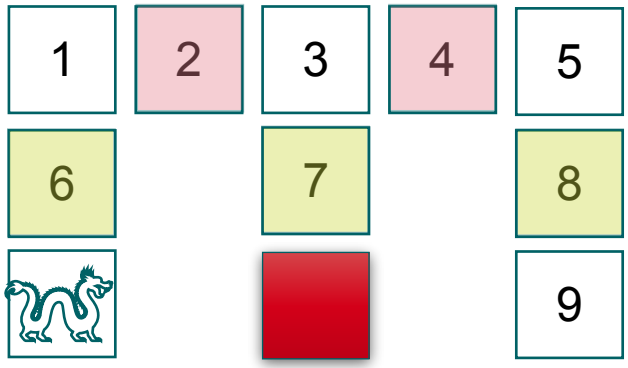
POMDPs

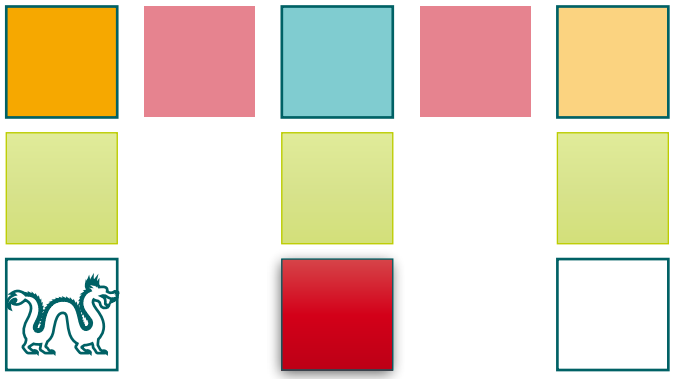


Parametric BNs

Randomisation and memory

POMDP: Reach red state without visiting the dragon.





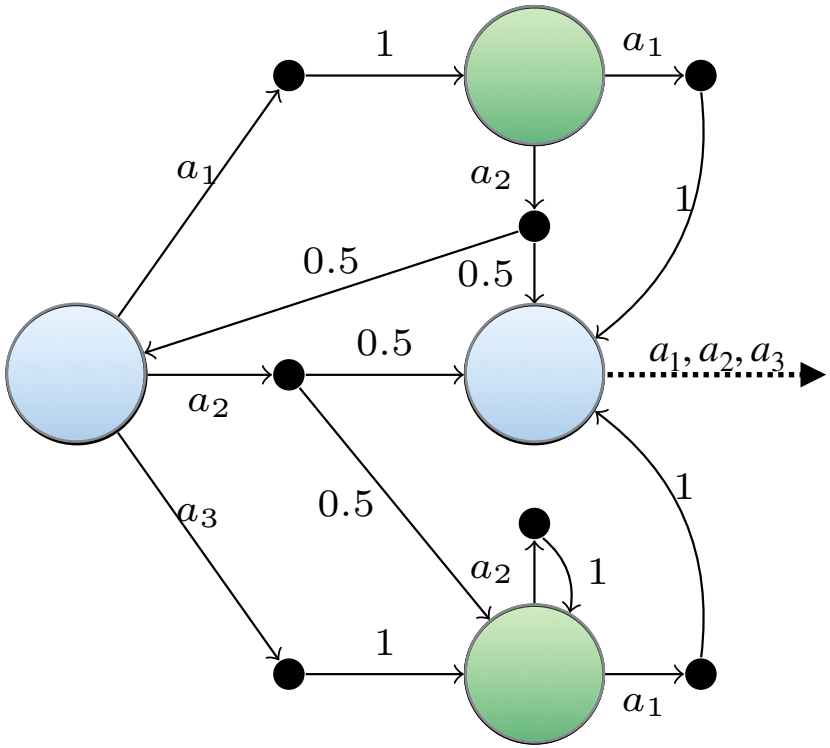
same observations:
- {2,4}
- {6,7,8}

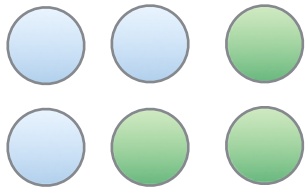
Start in 1 or 5:
Positional policy has to randomise in {2,4}

Start in 6 or 7:
no positional policy
store whether we have been in 3

POMDPs

MDPs with ‘observable colours’



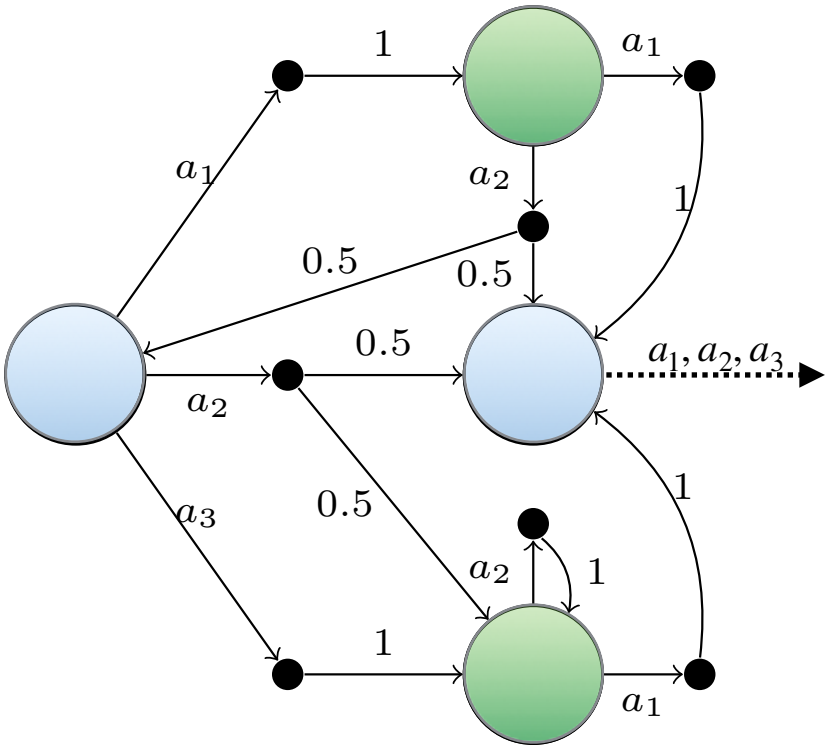


Distribution over actions

Distribution over actions

Given **any** POMDP
is there an observation-based policy s.t.
the probability reaching ● > λ

Partially observable MDPs (POMDPs)



For this talk:
POMDP = MDP with coloured states

Given **any** POMDP
is there a positional policy s.t. the probability reaching ● $>\lambda$

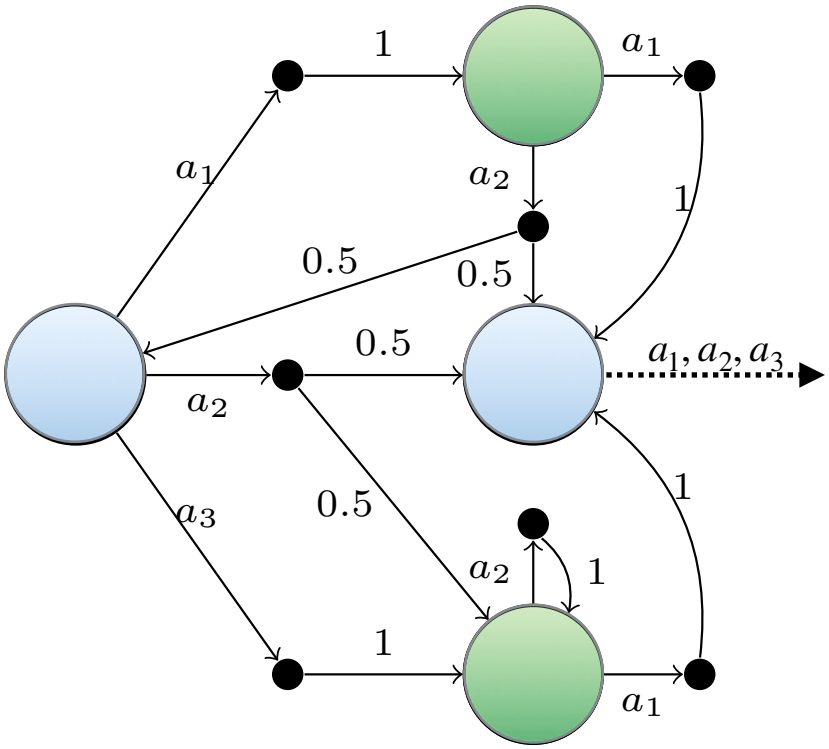
POMDP
Positional policy:
colours to distributions over actions

POMDPs with memoryless strategies

[UAI'18]

Maps observations to distributions over actions

maps observation/action pairs to probabilities



$a_1 \mapsto 1 - x$

$a_2 \mapsto x$

$a_1 \mapsto y_1$

$a_2 \mapsto y_2$

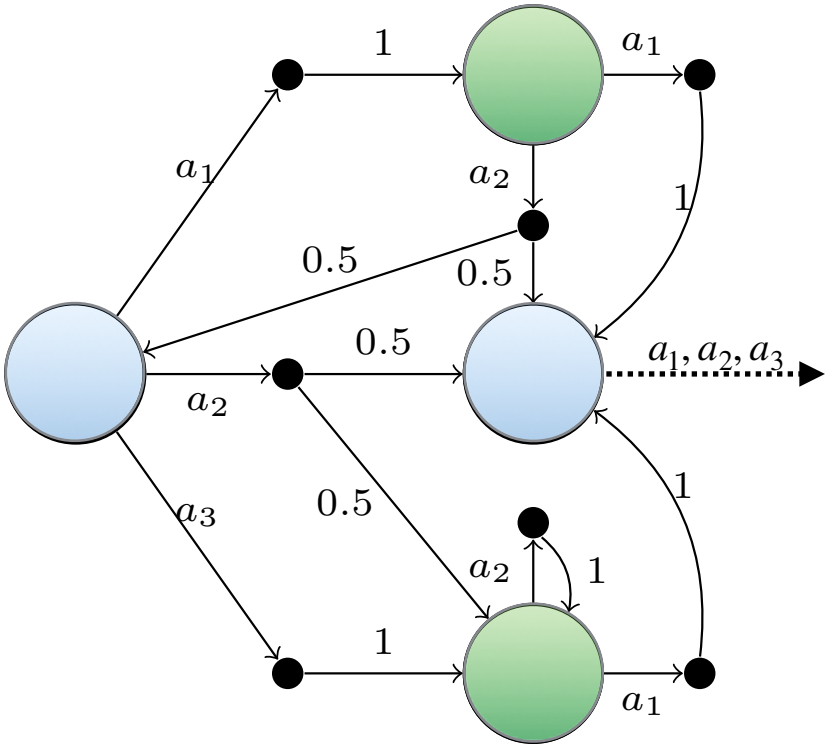
$a_3 \mapsto 1 - y_1 - y_2$

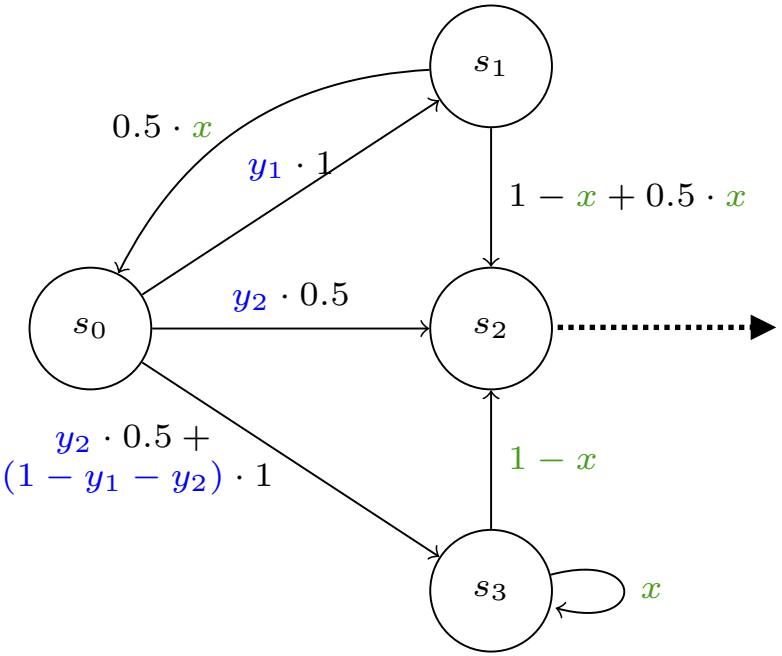
Strategy is uniquely described by values for x, y_1, y_2

Applying the memoryless strategy

[UAI'18]

Induced Markov Chain with unknown probabilities





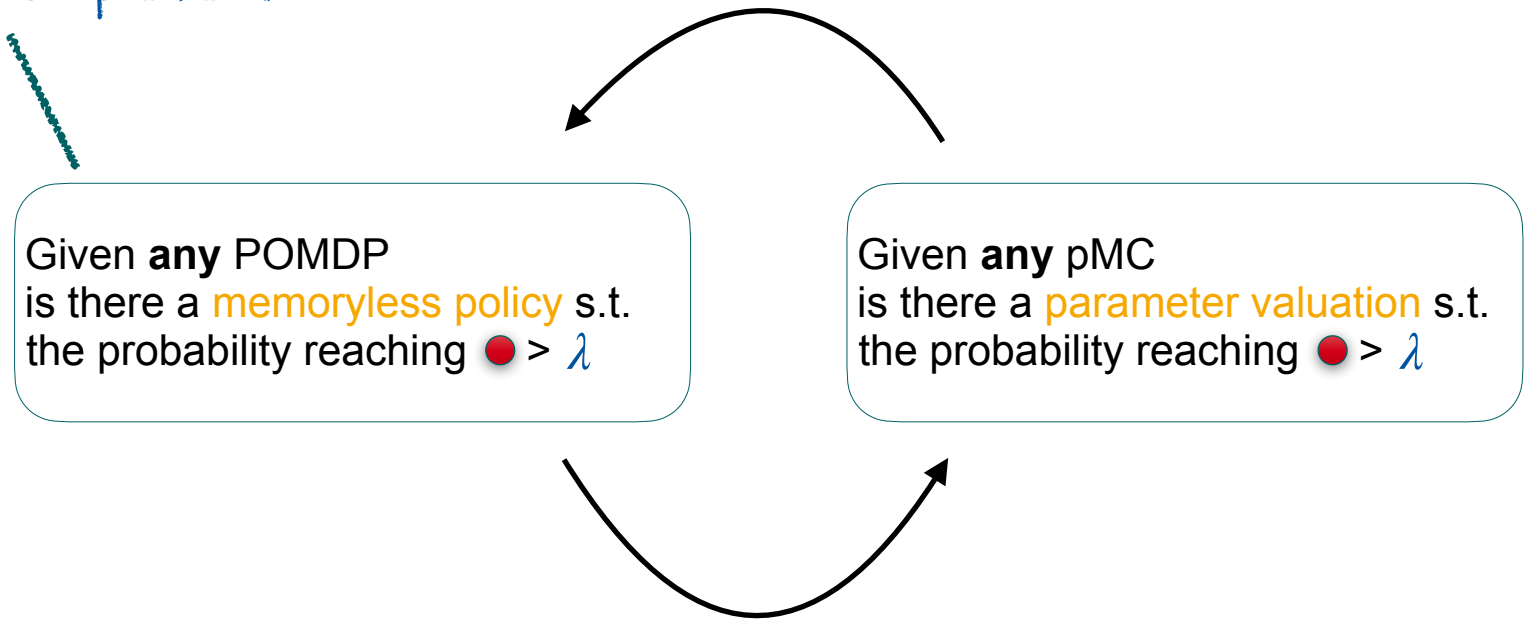
$a_1 \mapsto 1 - x$

$a_2 \mapsto x$

POMDPs and parametric Markov chains

[UAI'18]

Finite-state memory can be supported using a simple reduction



Parameter synthesis yields
new complexity results and **new competitive methods** for POMDPs

Complexity results

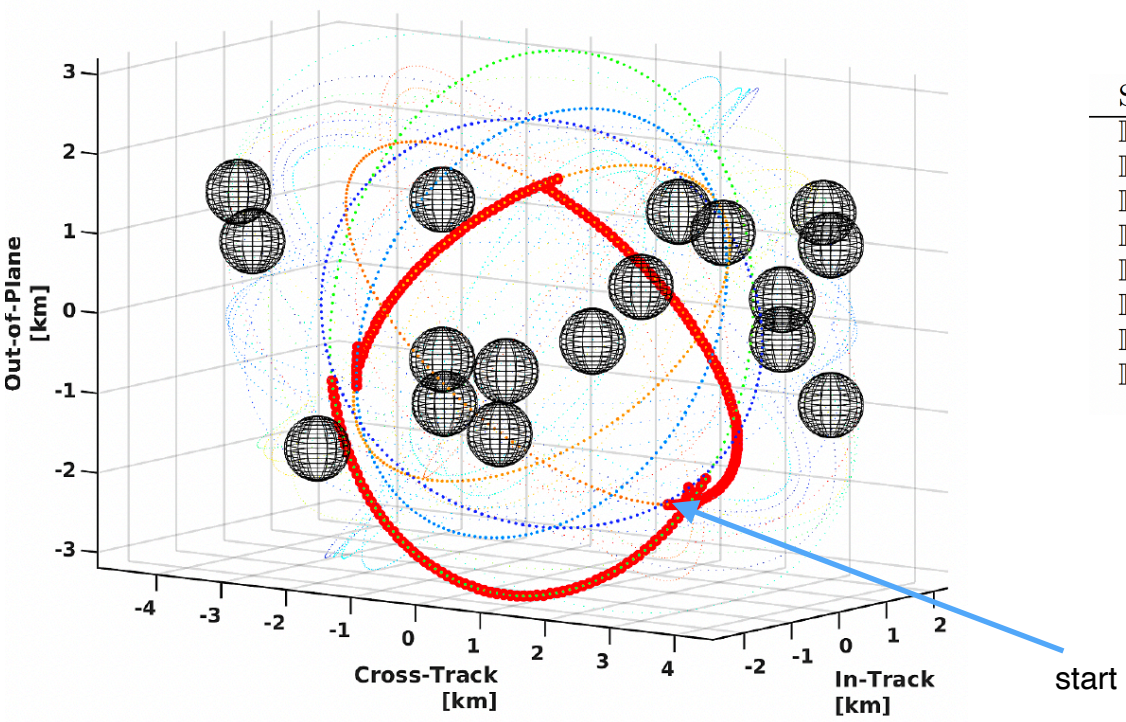
Deciding whether

there exist a memoryless policy for undiscounted expected reward

is ETR-complete.

Empirical Results

[IEEE TAC 2022]



memoryless

Spec	States	Trans.	Par.	SCP t	SCP iter
$\mathbb{P}_{\geq 0.5}$	6265	17436	231	8	6
$\mathbb{P}_{\geq 0.9}$	6265	17436	231	14	12
$\mathbb{P}_{\geq 0.95}$	6265	17436	231	TO	–
$\mathbb{P}_{\geq 0.95}$	31325	156924	2555	146	10
$\mathbb{P}_{\geq 0.995}$	31325	156924	2555	239	18
$\mathbb{P}_{\geq 0.995}$	217561	615433	2248	386	4
$\mathbb{P}_{\geq 0.995}$	217561	615433	5337	336	4
$\mathbb{P}_{\geq 0.995}$	217561	615433	10042	370	4

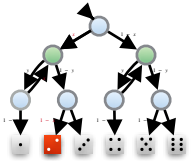
1440, 3600, 7200 observations

Trajectory for a finite-memory policy with memory size five

50% reduction in trajectory length and cost

Overview

Concepts



Encoding

$0 < x < 1 0 < y < 1$

$p_6 = 1$

$p_5 = 0 \quad p_4 = 0 \quad p_3 = 0$

$p_4 = x \cdot p_3 + (1-x) \cdot p_6$

$p_3 = x \cdot p_2 + (1-x) \cdot p_5$

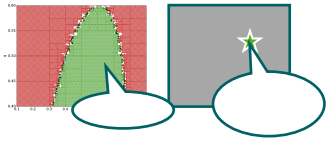
$p_2 = y \cdot p_3 + (1-y) \cdot p_4$

$p_1 = x \cdot x_2 + (1-x) \cdot p_5$

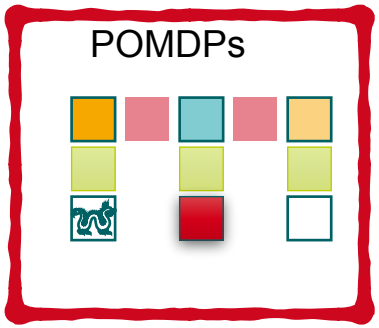
$p_1 > 1/6$

Complexity

Approaches



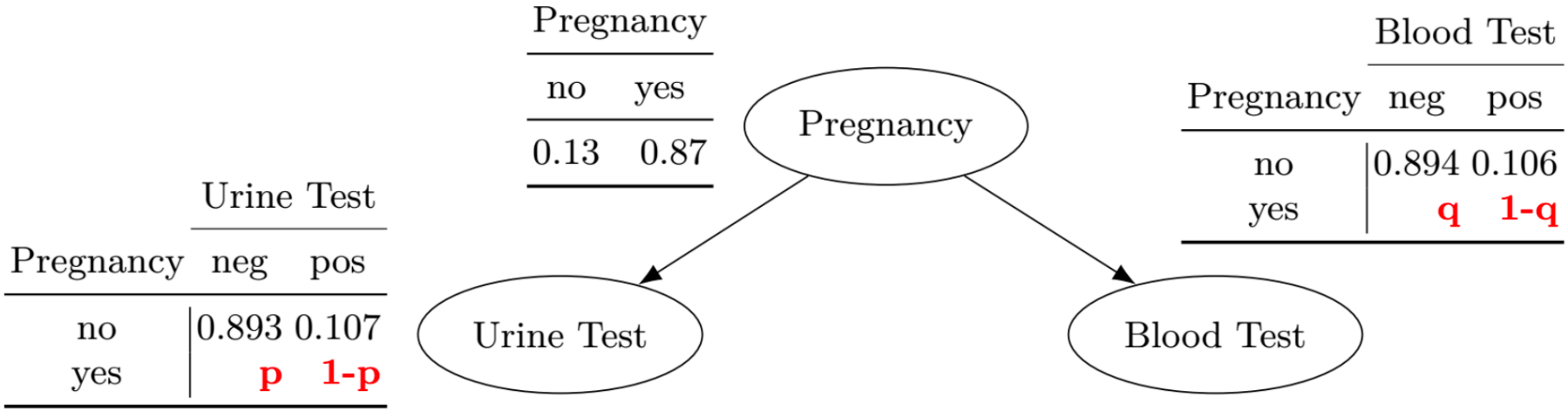
POMDPs



Parametric BNs

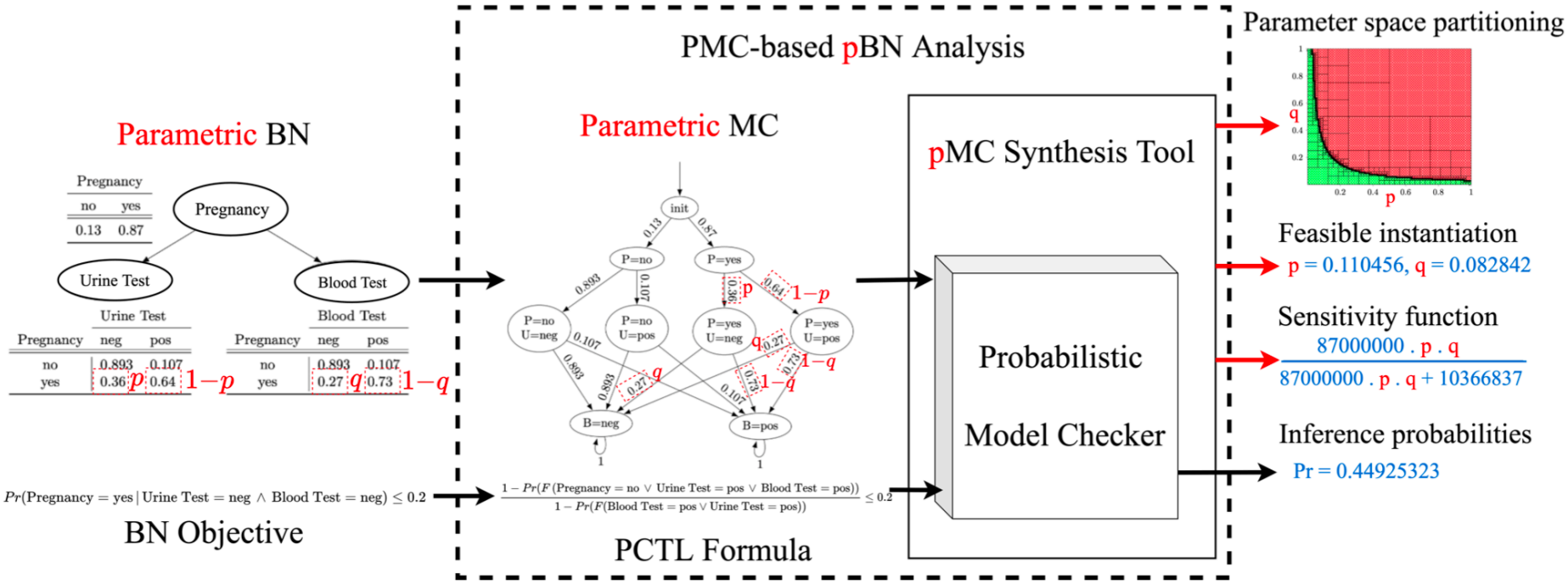
Parametric Bayesian Networks

[Salmani & Katoen, 2022]



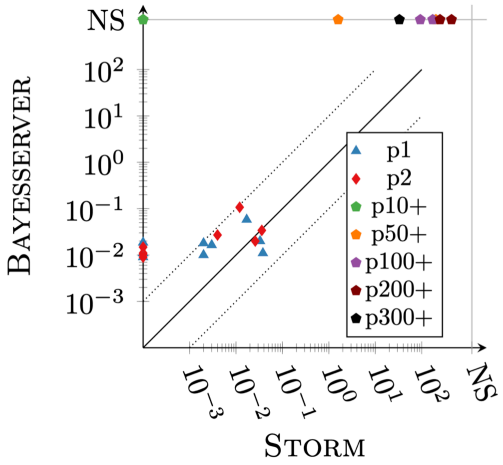
Approach

[Salmani & Katoen, 2022]

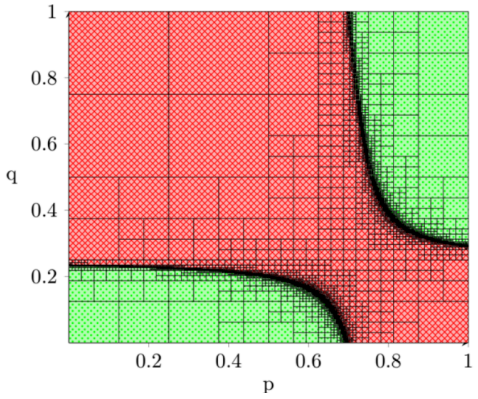


Some Results for pBNs

[Salmani & Katoen, 2022]



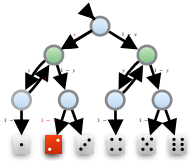
Fast sensitivity analysis with many parameters



Parameter partitioning ("alarm" benchmark)

Overview

Concepts



Encoding

$0 < x < 1 0 < y < 1$

$p_6 = 1$

$p_5 = 0 \quad p_3 = 0 \quad p_2 = 0$

$p_4 = x \cdot p_3 + (1-x) \cdot p_6$

$p_3 = x \cdot p_2 + (1-x) \cdot p_5$

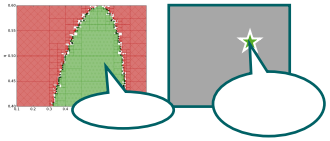
$p_2 = y \cdot p_3 + (1-y) \cdot p_4$

$p_1 = x \cdot x_2 + (1-x) \cdot p_5$

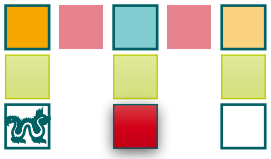
$p_1 > 1/6$

Complexity

Approaches



POMDPs



Parametric BNs

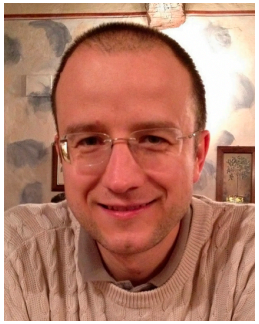
A Big Thanks to Our Co-Authors!

Nils Jansen
(Nijmegen, NL)

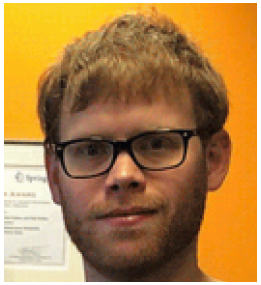
Murat Cubuktepe
(UT Austin US)

Jip Spel
(RWTH, D)

Matthias Volk
(Twente, NL)



Ufuk Topcu
(UT Austin US)



Tim Quatmann
(RWTH, D)

Bahar Salmani, Christian Hensel, Erika Abraham, Harold Bruintjes, Florian Corzilius, Christel Baier, Lisa Hutschenreiter, Joachim Klein, Ralf Wimmer, Leonore Winterer, Bernd Becker

Want to know more?

Parameter Synthesis in Markov Models: A Gentle Survey*

Nils Jansen[1], Sebastian Junges[1] and Joost-Pieter Katoen[2]

[1] Radboud University, Nijmegen, The Netherlands
[2] RWTH Aachen University, Aachen, Germany

Abstract. This paper surveys the analysis of parametric Markov models whose transitions are labelled with functions over a finite set of parameters. These models are symbolic representations of uncountable many concrete probabilistic models, each obtained by instantiating the parameters. We consider various analysis problems for a given logical specification φ: do all parameter instantiations within a given region of parameter values satisfy φ?, which instantiations satisfy φ and which ones do not?, and how can all such instantiations be characterised, either exactly or approximately? We address theoretical complexity results and describe the main ideas underlying state-of-the-art algorithms that established an impressive leap over the last decade enabling the fully automated analysis of models with millions of states and thousands of parameters.

or get in touch!

Outlook

- Monotonicity checking
  - efficient sufficient conditions
  - interplay with region verification
  - gradient descent
- Richer models (e.g., infinite-state, hybrid, AI models)
- Topology synthesis
- Variations
  - robust policies (rather than robust parameters)
  - uncertain (e.g., interval) models
  - parameters governed by distributions

Epilogue

Aim: Mechanically finding the right probabilities

- Feasibility: is there a compliant instantiation?
- Exact partitioning: which instantiations are good/bad?
- Approximate partitioning: using iterative abstraction
- Practical feasibility: using mathematical optimisation

Applications in e.g. Bayesian Networks and POMDPs

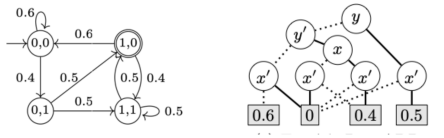
Wrap-Up

1. 

Fundamentals of
Probabilistic Model Checking

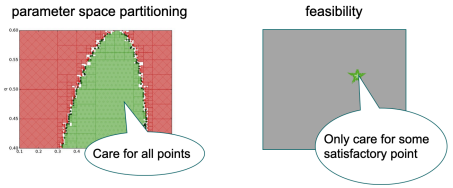
2. 

Probabilistic Model Checking with Storm:
Hands-on Slides

3. 

Example Markov chain Its MTBDD (x, x') to (y, y')

Automated Symbolic Reasoning

4. 

Parameter Synthesis in Markov Models

Wrap-Up

Recent Trends in Probabilistic Model Checking and Verification

- Verification of partially observable MDPs
- Verification of Multi-player MDPs / Stochastic Games
- Synthesis of robust policies
- Synthesis of small policy representations

- Usage in “Safe reinforcement learning”
- Tightening the connection to progress in classical planning

Which features would help you?

What methods should we consider?

katoen@cs.rwth-aachen.de

sjunges@cs.ru.nl