The What, Why, and How of Probabilistic Verification

Part 4: Recent Research Developments

Joost-Pieter Katoen

RWTH AACHEN UNIVERSITY

UNIVERSITY OF TWENTE.

CAV Invited Tutorial 2015, San Francisco

Overview

Recent Research Developments

- Parameter Synthesis
- Model Repair
- Counterexample Generation
- Probabilistic Programming

Epilogue

Overview

Motivation

Fact

Probabilistic model checking is applicable to various areas, e.g.:

- fault-tolerant systems
- randomized algorithms
- systems biology

Motivation

Fact

Probabilistic model checking is applicable to various areas, e.g.:

- fault-tolerant systems
- randomized algorithms
- systems biology

Limitation

Probabilities need to be known a priori

Motivation

Fact

Probabilistic model checking is applicable to various areas, e.g.:

- fault-tolerant systems
- randomized algorithms
- systems biology

Limitation

Probabilities need to be known a priori

Goal

Treat parametric models, synthesize “safe” parameter values

Motivation

Fact

Probabilistic model checking is applicable to various areas, e.g.:

- fault-tolerant systems
- randomized algorithms
- systems biology

Limitation

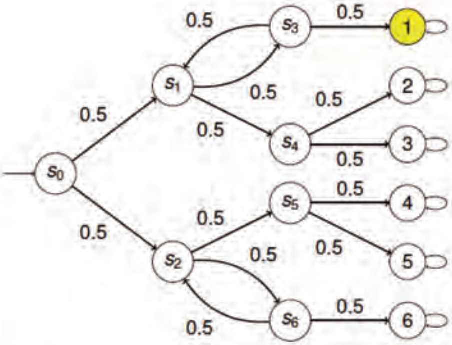
Probabilities need to be known a priori

Goal

Treat parametric models, synthesize "safe" parameter values

New: PROPhESY — A PRObabilistic PharametEr SYnthesis Tool

Knuth-Yao's Die Algorithm



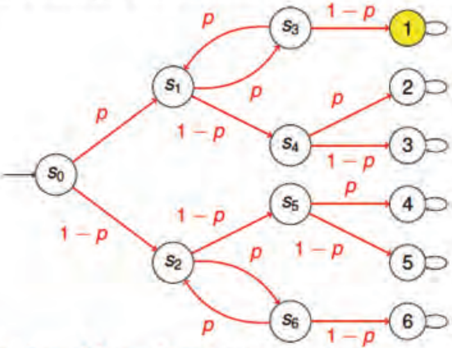
$Pr(\lozenge ●) = \frac{1}{6}$

compute

e.g. reachability probabilities, expected rewards, conditional probabilities

Parametric Markov Chains

idea: enrich discrete-time Markov chains with parameters



$Pr(\lozenge \bullet) = \frac{1}{6}$

$f_{\lozenge \bullet}(p) = \frac{p^2}{p+1}$

$f_{\lozenge \bullet}(0.5) = \frac{1}{6}$

compute **rational functions** representing

e.g. reachability probabilities, expected rewards, conditional probabilities

Parameter Synthesis

Inputs:

1. a (finite) parametric discrete-time Markov chain
2. a property (e.g., reachability, expected reward, conditional reachability)
3. a threshold

Desired output:

For which parameter values does the pMC satisfy the property with the given threshold?

Parameter Synthesis

Inputs:

1. a (finite) parametric discrete-time Markov chain
2. a property (e.g., reachability, expected reward, conditional reachability)
3. a threshold

Desired output:

For which parameter values does the pMC satisfy the property with the given threshold?

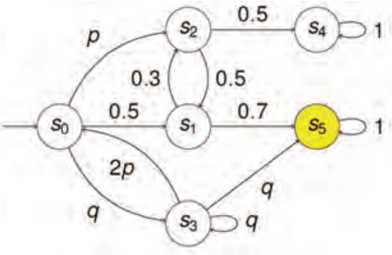
Problem instances:

- What is the maximal tolerable message loss?
- What is the maximal tolerable failure rate for program correctness?
- ......

State Elimination

[Daws, 2004]

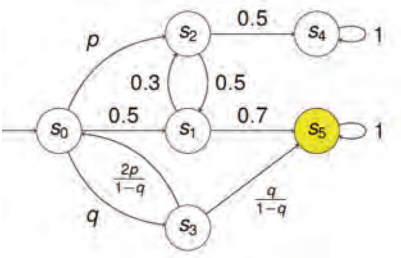
Adapt the automaton-to-regular expression algorithm to parametric MCs.



State Elimination

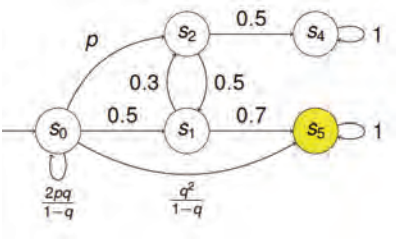
[Daws, 2004]

Adapt the automaton-to-regular expression algorithm to parametric MCs.

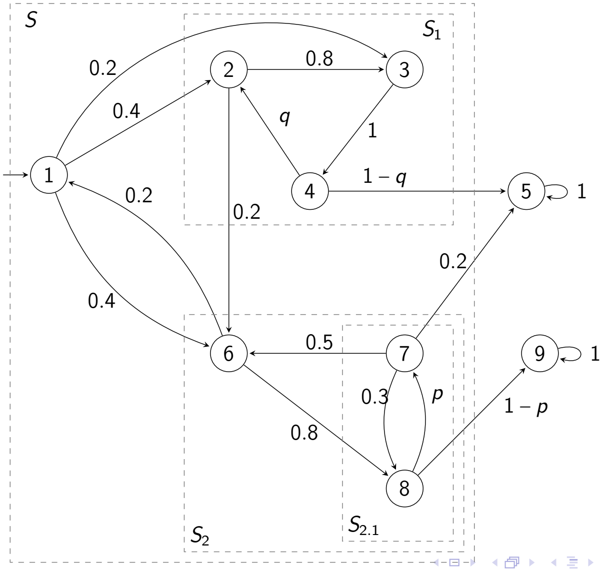


State Elimination [Daws, 2004]

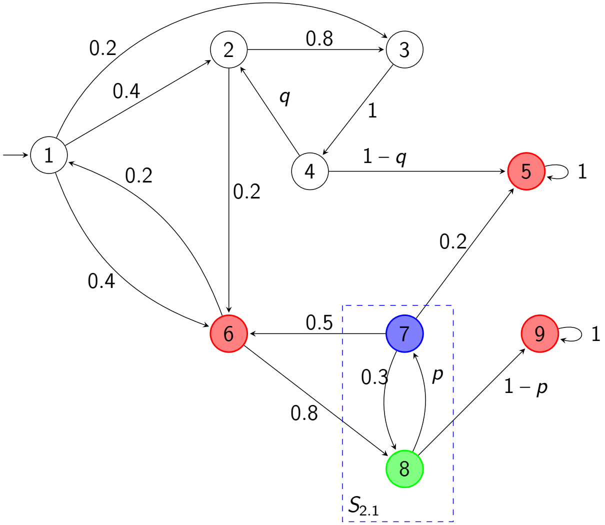
Adapt the automaton-to-regular expression algorithm to parametric MCs.



Hierarchical SCC Decomposition [Jansen *et al.*, 2014]

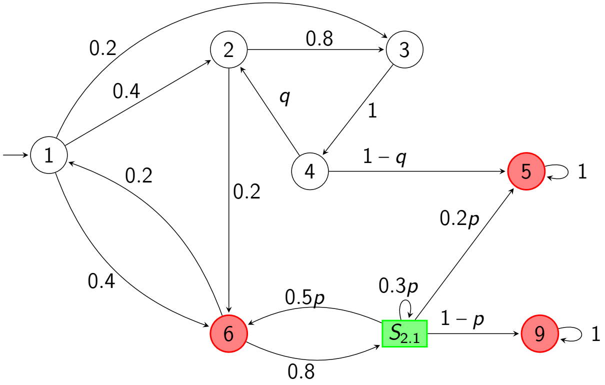


Hierarchical SCC Decomposition [Jansen *et al.*, 2014]

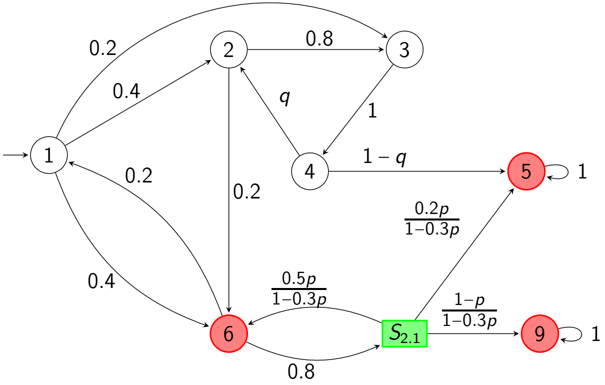


Hierarchical SCC Decomposition

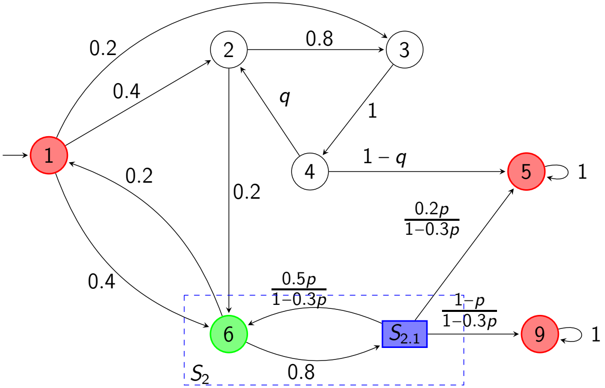
[Jansen *et al.*, 2014]



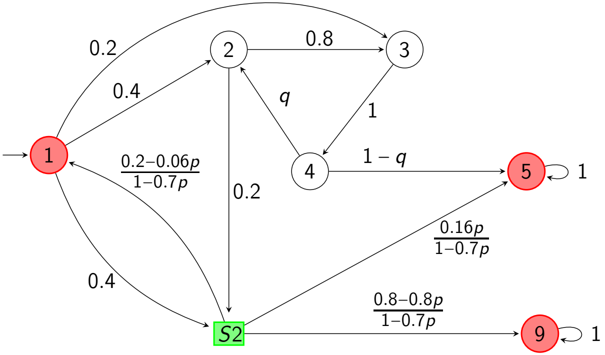
Hierarchical SCC Decomposition [Jansen *et al.*, 2014]



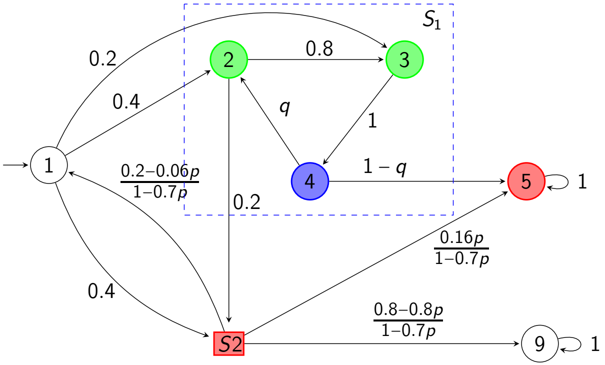
Hierarchical SCC Decomposition [Jansen *et al.*, 2014]



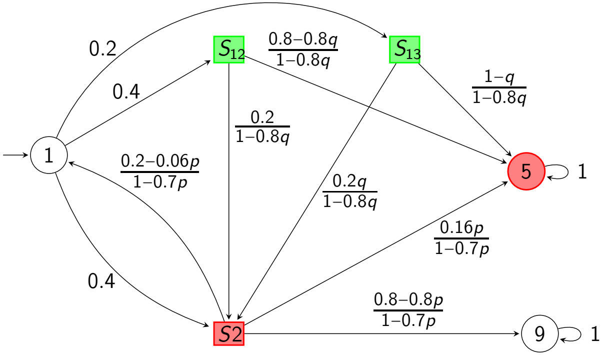
Hierarchical SCC Decomposition [Jansen *et al.*, 2014]



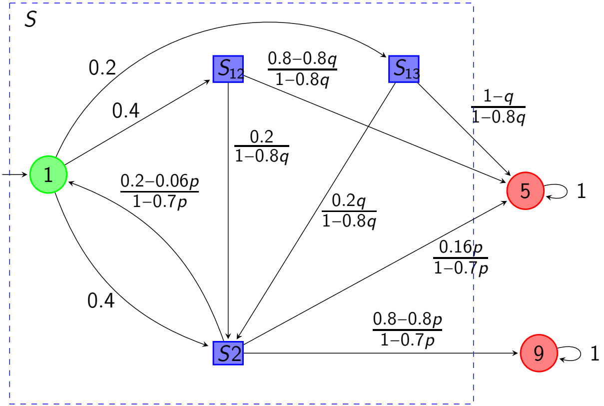
Hierarchical SCC Decomposition [Jansen *et al.*, 2014]



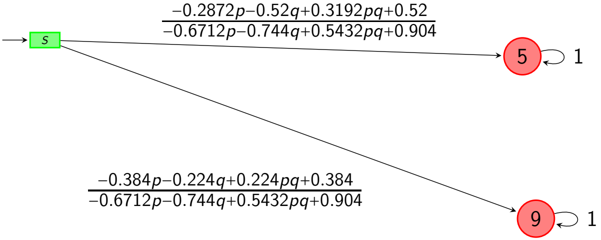
Hierarchical SCC Decomposition [Jansen *et al.*, 2014]



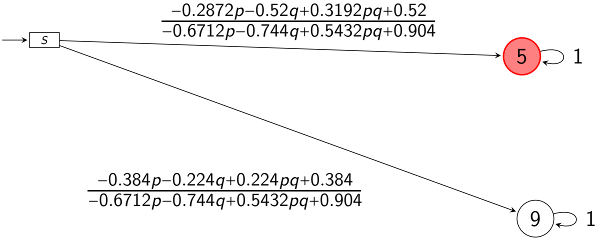
Hierarchical SCC Decomposition [Jansen *et al.*, 2014]



Hierarchical SCC Decomposition [Jansen *et al.*, 2014]



Hierarchical SCC Decomposition [Jansen *et al.*, 2014]



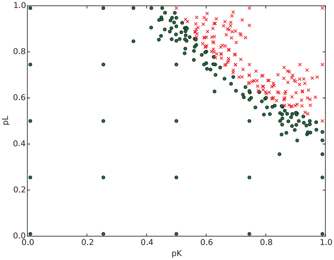
For which (combinations of) values for p and q is the probability of reaching 5 smaller than $c \in [0, 1]$?
$\Rightarrow$ Evaluate rational function.

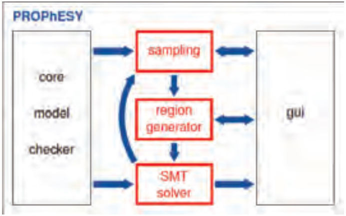
Analysing Parametric Markov Chains

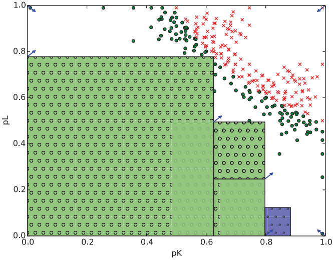
1. Determine the rational function f for the given property-of interest.
   - Use SCC-based state elimination
   - Use dedicated library `CaRL` for treating rational functions
2. In CEGAR-like style determine parameter sub-spaces for which $f <$ bound
   - Sample the parameter space
   - Automatically generate candidate regions
   - Check whether region is completely safe (unsafe) [1])
   - If sub-space contains an invalid point, refine the region and re-check

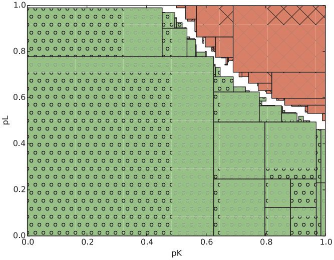
[1] Using SMT techniques for non-linear theories, e.g., Z3 or SMT-RAT.

Parameter Synthesis



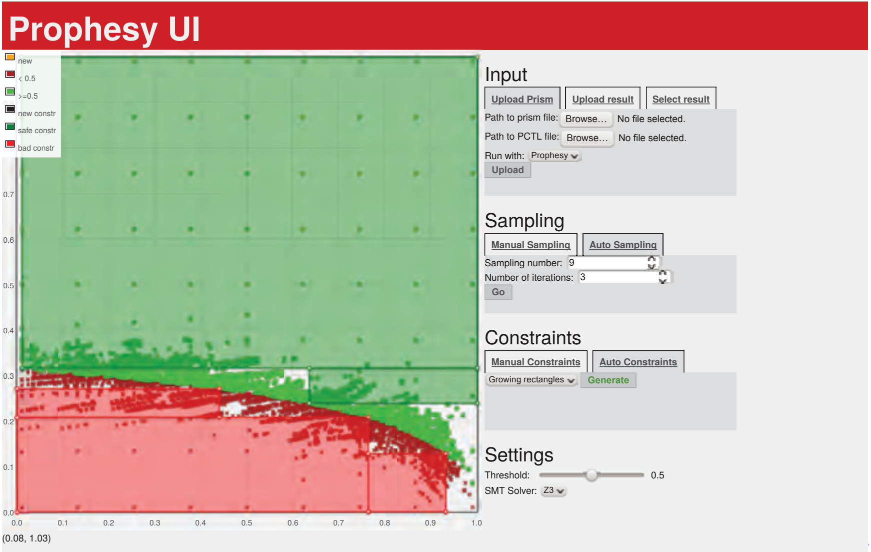






A Live Demo

Sampling and Regions



Experimental Results

[Dehnert *et al.*, 2015]

competitors

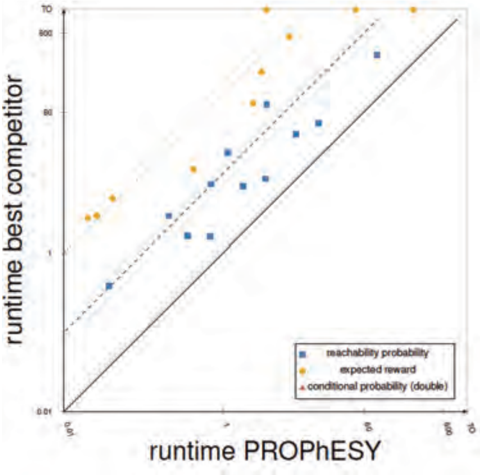
- PARAM [Hahn et al., 2010]
- PRISM [Parker et al., 2011]

models

- Bounded retransmission protocol
- NAND multiplexing
- Zeroconf, Crowds protocol
- 10^4 to $7.5 \cdot 10^6$ states

experiments:

- best set-up for each tool
- log-scale x- and y-axis



http://moves.rwth-aachen.de/research/tools/prophesy/

Experimental Results

[Dehnert *et al.*, 2015]

competitors

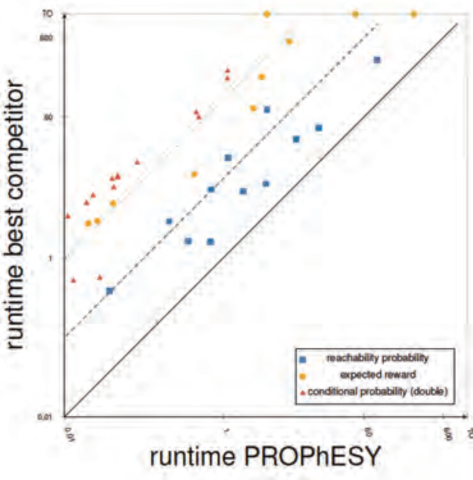
- PARAM [Hahn et al., 2010]
- PRISM [Parker et al., 2011]
- prototype [Baier et al., 2014]

models

- Bounded retransmission protocol
- NAND multiplexing
- Zeroconf, Crowds protocol
- 10^4 to $7.5 \cdot 10^6$ states

experiments:

- best set-up for each tool
- log-scale x- and y-axis



http://moves.rwth-aachen.de/research/tools/prophesy/

Overview

Recent Research Developments

- Parameter Synthesis
- Model Repair
- Counterexample Generation
- Probabilistic Programming

Epilogue

Model Repair: Motivation [Bartocci *et al.*, 2011]

- Assume a reachability probability threshold is not met

Model Repair: Motivation [Bartocci *et al.*, 2011]

- Assume a reachability probability threshold is not met
- What to do? No problem! Counterexamples point to errors

Model Repair: Motivation

[Bartocci *et al.*, 2011]

- Assume a reachability probability threshold is not met
- What to do? No problem! Counterexamples point to errors
- ...... But they are often large and incomprehensible

Model Repair: Motivation [Bartocci *et al.*, 2011]

- Assume a reachability probability threshold is not met
- What to do? No problem! Counterexamples point to errors
- . . . . . . But they are often large and incomprehensible
- Can we "fix" a model when a bad state is reached too often?

Model Repair: Motivation [Bartocci *et al.*, 2011]

- Assume a reachability probability threshold is not met
- What to do? No problem! Counterexamples point to errors
- . . . . . . But they are often large and incomprehensible
- Can we "fix" a model when a bad state is reached too often?
- What is the minimal change to be made?

Model Repair: Motivation

[Bartocci *et al.*, 2011]

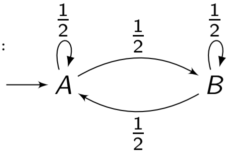
- Assume a reachability probability threshold is not met
- What to do? No problem! Counterexamples point to errors
- . . . . . . But they are often large and incomprehensible
- Can we "fix" a model when a bad state is reached too often?
- What is the minimal change to be made?

Here, automated model repair algorithms come into the play.

A Robotics Scenario

A Robotics Scenario

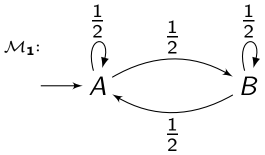
$\mathcal{M}_1$:



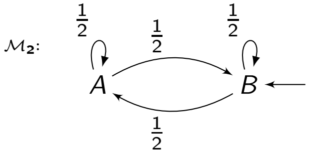
Environment

- Object moves between position A and B according to MC $\mathcal{M}_1$

A Robotics Scenario



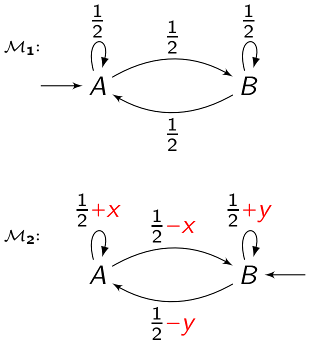
Environment



Strategy

- Object moves between position A and B according to MC $\mathcal{M}_1$
- Robot moves according to strategy given by MC $\mathcal{M}_2$

A Robotics Scenario

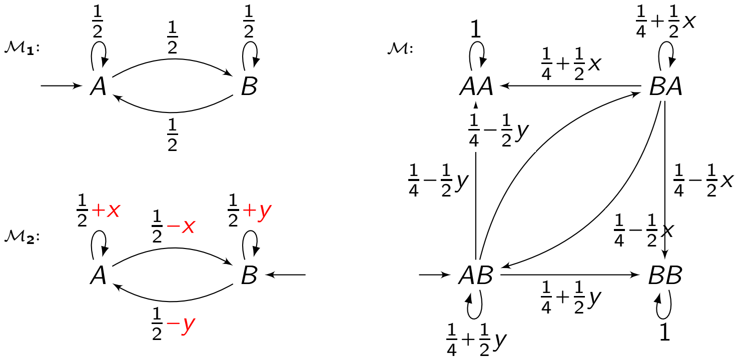


Environment

Strategy

- Object moves between position A and B according to MC $\mathcal{M}_1$
- Robot moves according to strategy given by pMC $\mathcal{M}_2$
  $\Rightarrow$ parameters indicate variability of the robot's strategy

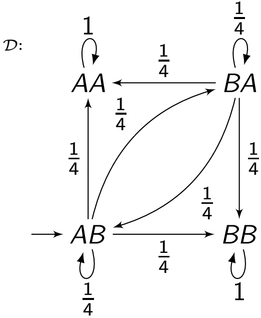
A Robotics Scenario



- Object moves between position A and B according to MC $\mathcal{M}_1$
- Robot moves according to strategy given by pMC $\mathcal{M}_2$
  $\Rightarrow$ parameters indicate variability of the robot's strategy
- In $\mathcal{M}_1 \| \mathcal{M}_2$, robot catches ball at positions AA or BB

Repairing Robotics Example

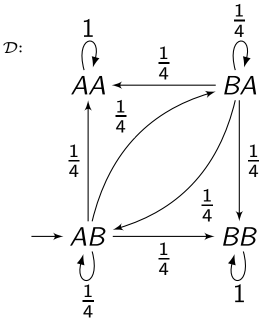
- Assume concrete strategy $\mathcal{M}$ (obtained via reinforcement learning)
- Property φ: The probability to catch at B shall be smaller than 1/2



Valuation $v(x) = v(y) = 0$

Repairing Robotics Example

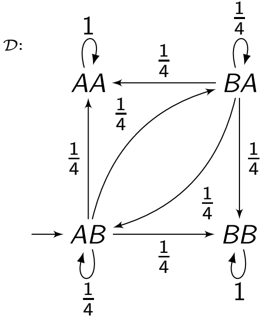
- Assume concrete strategy $\mathcal{M}$ (obtained via reinforcement learning)
- Property φ: The probability to catch at B shall be smaller than $1/2$
- $p_{BB} = 1/2$ ☹

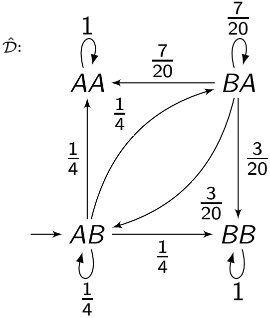


Valuation $v(x) = v(y) = 0 \Rightarrow v \not\models \varphi$

Repairing Robotics Example

- Assume concrete strategy $\mathcal{M}$ (obtained via reinforcement learning)
- Property φ: The probability to catch at B shall be smaller than $1/2$
- $p_{BB} = 1/2$ ☹ but $\hat{p}_{BB} = 1/3$ ☺





Valuation $v(x) = v(y) = 0 \Rightarrow v \not\models \varphi$ $\qquad$ $\hat{v}(x) = 0.2,\ \hat{v}(y) = 0 \Rightarrow \hat{v} \models \varphi$

A Local Repair Approach [Pathak *et al.*, 2015]

Local repair strategy for pMC and $\varphi = \Pr(\Diamond B) < p$:

[2] For simple parameter dependencies, $|R| = 1$ can be taken.

A Local Repair Approach [Pathak *et al.*, 2015]

Local repair strategy for pMC and $\varphi = \Pr(\Diamond B) < p$:

- Maintain graph structure of the pMC

[2] For simple parameter dependencies, $|R| = 1$ can be taken.

A Local Repair Approach [Pathak *et al.*, 2015]

Local repair strategy for pMC and $\varphi = \Pr(\Diamond B) < p$:

- Maintain graph structure of the pMC
- Heuristically select a set $R \subseteq S$ of states[2] to be repaired

[2] For simple parameter dependencies, $|R| = 1$ can be taken.

A Local Repair Approach

[Pathak *et al.*, 2015]

Local repair strategy for pMC and $\varphi = \Pr(\Diamond B) < p$:

- Maintain graph structure of the pMC
- Heuristically select a set $R \subseteq S$ of states[2] to be repaired
- Shift probability mass of some edges of $r \in R$ to its other edges, s.t.
  1. $\mathbf{P}'(s,u) = \mathbf{P}(s,u)$ for all $s \notin R$ and $u \in S$, and
  2. $\underbrace{\sum_{u \in S} \mathbf{P}'(r,u)}_{\text{valuation } v'} \cdot \Pr(u \models \Diamond B) < \underbrace{\sum_{u \in S} \mathbf{P}(r,u)}_{\text{valuation } v} \cdot \Pr(u \models \Diamond B)$ for all r in R

[2] For simple parameter dependencies, $|R| = 1$ can be taken.

A Local Repair Approach

[Pathak *et al.*, 2015]

Local repair strategy for pMC and $\varphi = \Pr(\Diamond B) < p$:

- Maintain graph structure of the pMC
- Heuristically select a set $R \subseteq S$ of states[2] to be repaired
- Shift probability mass of some edges of $r \in R$ to its other edges, s.t.
  1. $\mathbf{P}'(s,u) = \mathbf{P}(s,u)$ for all $s \notin R$ and $u \in S$, and
  2. $\underbrace{\sum_{u \in S} \mathbf{P}'(r,u)}_{\text{valuation } v'} \cdot \Pr(u \models \Diamond B) < \underbrace{\sum_{u \in S} \mathbf{P}(r,u)}_{\text{valuation } v} \cdot \Pr(u \models \Diamond B)$ for all r in R
- This yields a strict partial order on valuations $v' \prec v$

[2] For simple parameter dependencies, $|R| = 1$ can be taken.

A Local Repair Approach [Pathak *et al.*, 2015]

Local repair strategy for pMC and $\varphi = \Pr(\Diamond B) < p$:

- Maintain graph structure of the pMC
- Heuristically select a set $R \subseteq S$ of states[2] to be repaired
- Shift probability mass of some edges of $r \in R$ to its other edges, s.t.
  1. $\mathbf{P}'(s,u) = \mathbf{P}(s,u)$ for all $s \notin R$ and $u \in S$, and
  2. $\underbrace{\sum_{u \in S} \mathbf{P}'(r,u) \cdot \Pr(u \models \Diamond B)}_{\text{valuation } v'} < \underbrace{\sum_{u \in S} \mathbf{P}(r,u) \cdot \Pr(u \models \Diamond B)}_{\text{valuation } v}$ for all r in R
- This yields a strict partial order on valuations $v' \prec v$
- Iterations yield a repair sequence $v_n, \ldots, v_0$ with $n > 0$ such that:

  $v_{i+1} \prec v_i$ and $v_i \not\models \varphi$ for all $i < n$ and $v_n \models \varphi$

[2] For simple parameter dependencies, $|R| = 1$ can be taken.

A Local Repair Approach

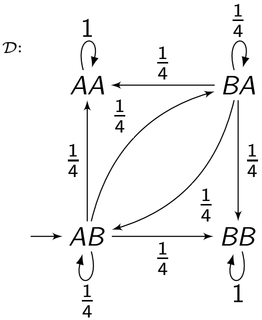
[Pathak *et al.*, 2015]

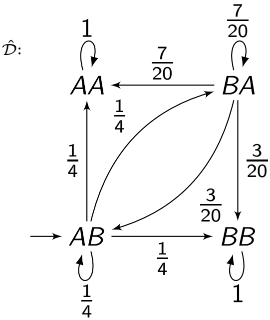
Local repair strategy for pMC and $\varphi = \Pr(\Diamond B) < p$:

- Maintain graph structure of the pMC
- Heuristically select a set $R \subseteq S$ of states[2] to be repaired
- Shift probability mass of some edges of $r \in R$ to its other edges, s.t.
  1. $\mathbf{P}'(s,u) = \mathbf{P}(s,u)$ for all $s \notin R$ and $u \in S$, and
  2. $$\underbrace{\sum_{u \in S} \mathbf{P}'(r,u) \cdot \Pr(u \models \Diamond B)}_{\text{valuation } v'} < \underbrace{\sum_{u \in S} \mathbf{P}(r,u) \cdot \Pr(u \models \Diamond B)}_{\text{valuation } v} \quad \text{for all } r \text{ in } R$$
- This yields a strict partial order on valuations $v' \prec v$
- Iterations yield a repair sequence $v_n, \ldots, v_0$ with $n > 0$ such that:
  $$v_{i+1} \prec v_i \quad \text{and} \quad v_i \not\models \varphi \text{ for all } i < n \quad \text{and} \quad v_n \models \varphi$$
- If no finite repair sequence exists, the pMC is not repairable!

[2] For simple parameter dependencies, $|R| = 1$ can be taken.

Local Repair in Action

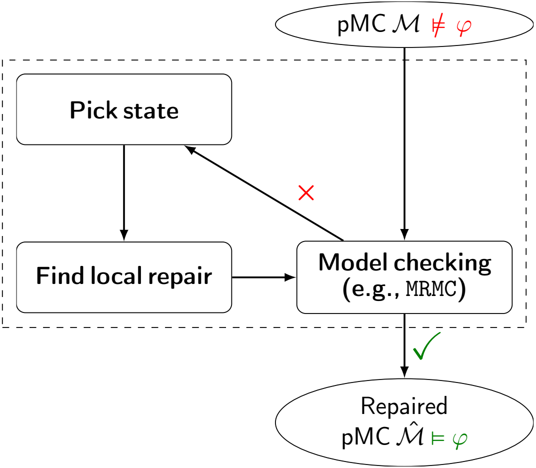




Valuation $v(x) = v(y) = 0 \Rightarrow v \not\models \varphi$ $\qquad$ $\hat{v}(x) = 0.2,\ \hat{v}(y) = 0 \Rightarrow \hat{v} \models \varphi$

We have $\hat{v} \prec v$ and BA is only repaired state.

Local Repair Algorithm



Local Repair Achievements

Soundness

A local repair step repairs at least one state and does not “un-repair” others.

Intuition: Reachability probabilities cannot be increased by local decreasing. (Induction over transient probabilities).

Completeness

If each repair has a certain mass, termination with a minimal result is ensured.

Intuition: The repair mass of infinite sequences converges to zero.

Cost-Optimal Repairs

Cost-Optimal Repairs

- Greedy strategy implies cost functions depending on local changes
- Non-linear cost functions would significantly affect computation time

Cost-Optimal Repairs

- Greedy strategy implies cost functions depending on local changes
- Non-linear cost functions would significantly affect computation time
- Aim: keep changes in parameter values of initial valuation v_0 small:

$$\sum_{x_i \in \mathrm{Var}} |v(x_i) - v_0(x_i)|$$

Cost-Optimal Repairs

- Greedy strategy implies cost functions depending on local changes
- Non-linear cost functions would significantly affect computation time
- Aim: keep changes in parameter values of initial valuation v_0 small:

$$\sum_{x_i \in \mathrm{Var}} |v(x_i) - v_0(x_i)|$$

- Prefer local repair step in state s that minimises

$$\sum_{x_i \in \mathrm{Var}(s)} |(v(x_i) + \delta_i) - v_0(x_i)|$$

Cost-Optimal Repairs

- Greedy strategy implies cost functions depending on local changes
- Non-linear cost functions would significantly affect computation time
- Aim: keep changes in parameter values of initial valuation v_0 small:

$$\sum_{x_i \in \mathrm{Var}} |v(x_i) - v_0(x_i)|$$

- Prefer local repair step in state s that minimises

$$\sum_{x_i \in \mathrm{Var}(s)} |(v(x_i) + \delta_i) - v_0(x_i)|$$

As we use heuristics to pick a state, no global optimal cost is achieved.

Experimental Results for Robot Case Study

Strategies obtained via reinforcement learning from MDP environment

	model		time		quality					
N	states	trans	mc	pick	$\Pr^{\mathcal{D}}$	$\Pr^{\hat{\mathcal{D}}}$	$\sum\Delta$	mx_{Δ}	$\lvert E\rvert$	steps
48	2305	17859		1.05	.159	.001	33.4	.72	621	
64	4097	32003		1.66	.182	.001	18.0	.65	427	
96	9217	72579		6.00	.189	.001	28.0	.68	657	
128	16385	129539		8.46	.150	.001	20.2	.45	640	
256	65537	521219		63.6	.130	.000	28.0	.27	888	
512	262145	2091011		–	.168	.101	19.9	.26	480	
512[a]	262145	2091011		21.7	.168	.000	54.8	.26	1760	
1024	1048577	8376323		–	.105	.104	1.1	.19	24	
1024[a]	1048577	8376323		28.9	.105	.036	80.8	.25	2400	

C++ implementation, Intel I7 CPU 3.4 GHz with 32GB RAM; TO = 2700 sec

[a] = repairing $\lvert R\rvert = 20$ states simultaneously.

Experimental Results for Robot Case Study

Strategies obtained via reinforcement learning from MDP environment

model			time		quality					
N	states	trans	**mc**	pick	$\Pr^{\mathcal{D}}$	$\Pr^{\hat{\mathcal{D}}}$	$\sum\Delta$	mx_Δ	$\lvert E\rvert$	steps
48	2305	17859	0.76	1.05	.159	.001	33.4	.72	621	
64	4097	32003	1.58	1.66	.182	.001	18.0	.65	427	
96	9217	72579	7.17	6.00	.189	.001	28.0	.68	657	
128	16385	129539	15.3	8.46	.150	.001	20.2	.45	640	
256	65537	521219	129	63.6	.130	.000	28.0	.27	888	
512	262145	2091011	TO	–	.168	.101	19.9	.26	480	
512[a]	262145	2091011	145	21.7	.168	.000	54.8	.26	1760	
1024	1048577	8376323	TO	–	.105	.104	1.1	.19	24	
1024[a]	1048577	8376323	378	28.9	.105	.036	80.8	.25	2400	

C++ implementation, Intel I7 CPU 3.4 GHz with 32GB RAM; TO = 2700 sec

[a] = repairing $|R| = 20$ states simultaneously.

Experimental Results for Robot Case Study

Strategies obtained via reinforcement learning from MDP environment

model			time		quality					
N	states	trans	mc	pick	$\Pr^{\mathcal{D}}$	$\Pr^{\hat{\mathcal{D}}}$	$\sum\Delta$	mx_{Δ}	$\lvert E\rvert$	steps
48	2305	17859	0.76	1.05	.159	.001	33.4	.72	621	77
64	4097	32003	1.58	1.66	.182	.001	18.0	.65	427	53
96	9217	72579	7.17	6.00	.189	.001	28.0	.68	657	82
128	16385	129539	15.3	8.46	.150	.001	20.2	.45	640	80
256	65537	521219	129	63.6	.130	.000	28.0	.27	888	111
512	262145	2091011	TO	–	.168	.101	19.9	.26	480	60
512[a]	262145	2091011	145	21.7	.168	.000	54.8	.26	1760	11
1024	1048577	8376323	TO	–	.105	.104	1.1	.19	24	3
1024[a]	1048577	8376323	378	28.9	.105	.036	80.8	.25	2400	3

C++ implementation, Intel I7 CPU 3.4 GHz with 32GB RAM; TO = 2700 sec

[a] = repairing $|R| = 20$ states simultaneously.

Experimental Results for Robot Case Study

Strategies obtained via reinforcement learning from MDP environment

model			time		quality					
N	states	trans	**mc**	pick	$\Pr^{\mathcal{D}}$	$\Pr^{\hat{\mathcal{D}}}$	$\sum \Delta$	mx_{Δ}	$\lvert E \rvert$	**steps**
48	2305	17859	0.76	1.05	.159	.001	33.4	.72	621	77
64	4097	32003	1.58	1.66	.182	.001	18.0	.65	427	53
96	9217	72579	7.17	6.00	.189	.001	28.0	.68	657	82
128	16385	129539	15.3	8.46	.150	.001	20.2	.45	640	80
256	65537	521219	129	63.6	.130	.000	28.0	.27	888	111
512	262145	2091011	TO	–	.168	.101	19.9	.26	480	60
512[a]	262145	2091011	145	21.7	.168	.000	54.8	.26	1760	11
1024	1048577	8376323	TO	–	.105	.104	1.1	.19	24	3
1024[a]	1048577	8376323	378	28.9	.105	.036	80.8	.25	2400	3

C++ implementation, Intel I7 CPU 3.4 GHz with 32GB RAM; TO = 2700 sec

[a] = repairing $|R| = 20$ states simultaneously.

Experiments on Parametric PRISM Benchmarks

Model				time[1]			quality					
	N	K	states	trans	mc	sn	$\Pr^{\mathcal{D}}$	$\Pr^{\hat{\mathcal{D}}}$	$\hat{p}_f$ / $\hat{p}_e$	$\Pr_{real}$	$\lvert E \rvert$	steps
Crowds	5	4	3515	6035	0.23	0.20	.316	.27	.81	.26	32	16
Crowds	6	8	164,308	308,452	5.1	11.2	.519	.327	.813	.316	32	16
Crowds	8	10	3,058,199	6,558,839	75.8	1194	.59	.416	.64	.332	32	16
Crowds	9	10	6,534,529	1,484,8549	237	452	.589	.332	.82	.323	32	16
Crowds	10	6	352,535	833,015	11.5	21.1	.424	.249	.807	.231	32	16
Crowds	12	6	829,669	2,166,277	32.6	55.9	.423	.239	.807	.22	32	16
NAND	6	6	8426	12,209	0.99	0.63	.746	.583	.020	.586	54	29
NAND	10	8	55,902	83,727	14.4	4.89	.727	.514	.020	.519	54	29
NAND	12	6	77,294	116,972	19.4	7.03	.800	.621	.020	.625	54	29
NAND	12	8	102,842	155,564	33.1	9.43	0.808	.623	.020	.628	54	29
NAND	12	10	128,390	194,156	50.2	12.0	.810	.623	.020	.627	54	29
NAND	12	12	153,938	232,748	71.7	14.8	.811	.621	.020	.625	54	29

- "False" valuations introduced, repaired towards original result
- Correct model probabilities and the repaired ones are quite close

Overview

Motivation

> It is impossible to overestimate the importance of counterexamples.
> The counterexamples are invaluable in debugging complex systems.
> Some people use model checking just for this feature.
>
> Ed Clarke, 25 Years of Model Checking, FLOC 2008

Relevance for CEGAR, model repair, scheduling problems, model analysis ...

Counterexamples

- LTL counterexamples are finite paths
  - $\Box\Phi$: a path ending in a $\neg\Phi$-state
  - $\Diamond\Phi$: a $\neg\Phi$-path leading to a $\neg\Phi$ cycle
  - BFS yields shortest counterexamples
- CTL counterexamples are (mostly) finite trees
  - universal CTL$\setminus$LTL: trees or proof-like counterexample
  - existential CTL: witnesses, annotated counterexample
- What are counterexamples for probabilistic reachability?
  - a set of finite paths whose probability mass exceeds a threshold
  - represented as minimal (critical) sub-models

Minimal Critical Subsystems

Given a DTMC $\mathcal{D}$ refuting $\Pr(\Box G) > p$, that is $\Pr(\Diamond \neg G) \leqslant 1-p$

Minimal Critical Subsystems

Given a DTMC $\mathcal{D}$ refuting $\Pr(\Box G) > p$, that is $\Pr(\Diamond \neg G) \leqslant 1-p$

Critical subsystem

A subset $C \subseteq S$ such that the probability of reaching a $\neg G$-state by only visiting states in C is already beyond $1-p$.

Minimal Critical Subsystems

Given a DTMC $\mathcal{D}$ refuting $\Pr(\Box G) > p$, that is $\Pr(\Diamond \neg G) \leqslant 1{-}p$

Critical subsystem

A subset $C \subseteq S$ such that the probability of reaching a $\neg G$-state by only visiting states in C is already beyond $1{-}p$.

Goal

Compute a critical subsystem with a minimum number of states. This is a minimal critical subsystem.

Minimal Critical Subsystems

Given a DTMC $\mathcal{D}$ refuting $\Pr(\Box G) > p$, that is $\Pr(\Diamond\neg G) \leqslant 1-p$

Critical subsystem

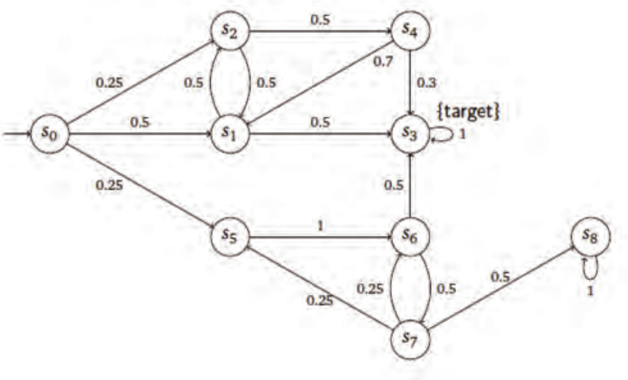
A subset $C \subseteq S$ such that the probability of reaching a $\neg G$-state by only visiting states in C is already beyond $1-p$.

Goal

Compute a critical subsystem with a minimum number of states. This is a minimal critical subsystem.

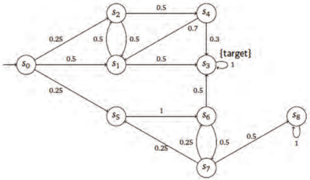
For arbitrary PCTL-formulas, finding a minimal critical subsystem is NP-complete.

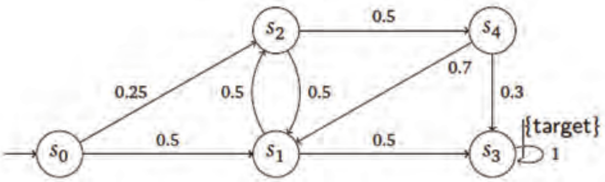
Example MCS



Property: `target` is reachable with probability $< 7/10$

Example MCS





MCS for which `target` is reachable with probability $> \, ^7/_{10}$

MILP formulation for MCS [Jansen *et al.*, 2012]

MILP formulation for MCS [Jansen *et al.*, 2012]

Variables

- $x_s \in \{0, 1\}$, a decision variable for each state s
- $p_s \in [0, 1]$, reachability probability for state s within the subsystem

MILP formulation for MCS [Jansen *et al.*, 2012]

Variables

- $x_s \in \{0, 1\}$, a decision variable for each state s
- $p_s \in [0, 1]$, reachability probability for state s within the subsystem

Constraints

$$
\begin{aligned}
\text{minimize} \quad & \sum_{s \in S} x_s \\
\text{such that} \quad & \\
\text{initial state } s_0: \quad & p_{s_0} > 1 - p \\
\text{target states } s: \quad & p_s = x_s \\
\text{non-target states } s: \quad & p_s \leqslant x_s \\
\text{non-target states } s: \quad & p_s \leqslant \textstyle\sum_{u \in S} \mathbf{P}(s, u) \cdot p_u
\end{aligned}
$$

MILP formulation for MCS [Jansen *et al.*, 2012]

- This yields only a lower bound on required probability p_{s_0}
- Additionally, we want to obtain an MCS with a maximal probability

MILP formulation for MCS [Jansen *et al.*, 2012]

- This yields only a lower bound on required probability p_{s_0}
- Additionally, we want to obtain an MCS with a maximal probability

Adapted constraints (for some $0 < c < 1$)

$$
\begin{aligned}
\text{minimize} \quad & \sum_{s \in S} -c \cdot p_{s_0} + x_s \\
\text{such that} \quad & \\
\text{initial state } s_0: \quad & p_{s_0} > 1 - p \\
\text{target states } s: \quad & p_s = x_s \\
\text{non-target states } s: \quad & p_s \leqslant x_s \\
\text{non-target states } s: \quad & p_s \leqslant \textstyle\sum_{u \in S} \mathbf{P}(s, u) \cdot p_u
\end{aligned}
$$

MILP formulation for MCS [Jansen *et al.*, 2012]

- This yields only a lower bound on required probability p_{s_0}
- Additionally, we want to obtain an MCS with a maximal probability

Adapted constraints (for some $0 < c < 1$)

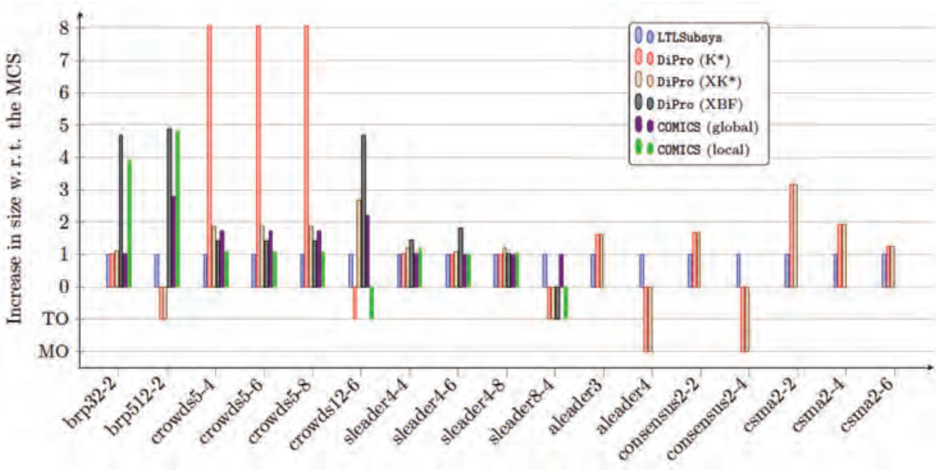
$$\text{minimize} \quad \sum_{s \in S} -c \cdot p_{s_0} + x_s$$

such that

initial state s_0 :	$p_{s_0} > 1-p$
target states s :	$p_s = x_s$
non-target states s :	$p_s \leqslant x_s$
non-target states s :	$p_s \leqslant \sum_{u \in S} \mathbf{P}(s, u) \cdot p_u$

This can be generalised for PCTL, and rewards. MDPs: ω-regular properties.

Experiments



Experiments

Benchmark	States	λ	Subsystem	Time (s)	Memory
crowds	68740	0.1	83	343	< 1 GB
sleader	12302	0.5	6150	22	< 1 GB
consensus	272	0.1	15	733	< 1 GB
csma	66718	0.1	415	2364	< 1 GB

- Hard to proof optimality (NP complete for most of the settings)
- Using intermediate results of MILP solvers gives good heuristic method

PRISM Counterexamples

- Counterexamples on level of state space are typically hard to grasp
- Idea: generate counterexamples directly on model description level!

---

module coin

  f: bool init 0;

  c: bool init 0;

  [flip] $\neg f \rightarrow 0.5 : (f' = 1)\&(c' = 1) + 0.5 : (f' = 1)\&(c' = 0)$;

  [reset] $f \wedge \neg c \rightarrow 1 : (f' = 0)$;

  [proc] $f \rightarrow 0.99 : (f' = 1) + 0.01 : (c' = 1)$;

endmodule

module processor

  p: bool init 0;

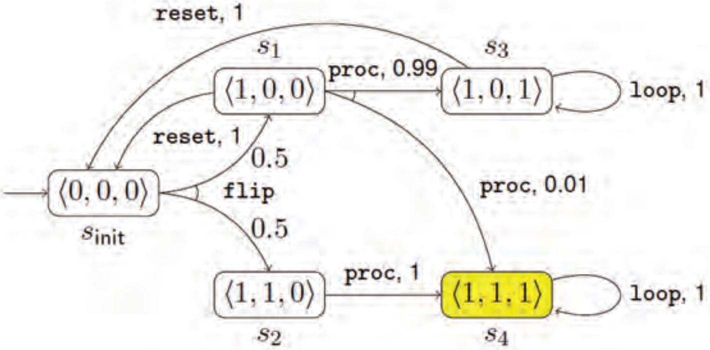
  [proc] $\neg p \rightarrow 1 : (p' = 1)$;

  [loop] $p \rightarrow 1 : (p' = 1)$;

  [reset] $\mathit{true} \rightarrow 1 : (p' = 0)$

endmodule

PRISM Model's State Space



$$\mathcal{P}_{\leq 1-\lambda}(\lozenge \bigcirc) \;\Leftrightarrow\; Pr^{\max}_{\mathcal{A}}(s_{\text{init}} \models \lozenge \bigcirc) \leq 1-\lambda$$

Obtaining PRISM Counterexamples

Minimal set of the PRISM commands whose induced MDP is already buggy!

MILP Approach [Jansen *et al.*, 2013]

1. Assign a unique label to each command.
2. Construct state space, label transitions their originating commands
3. Use a MILP formulation to minimize the number of commands.

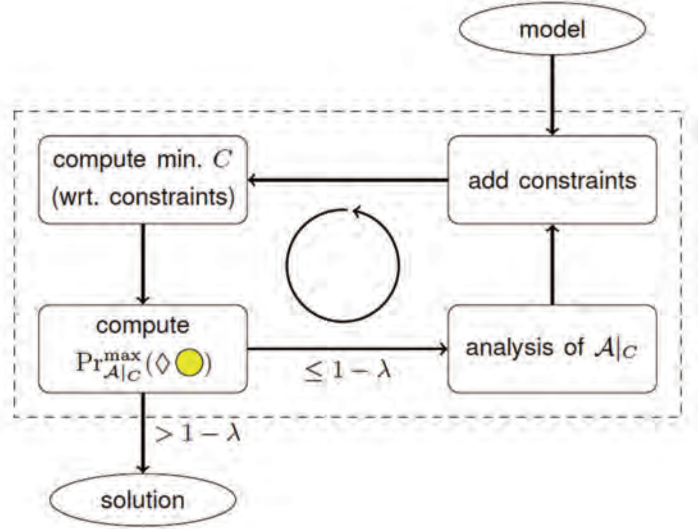
Obtaining PRISM Counterexamples

Minimal set of the PRISM commands whose induced MDP is already buggy!

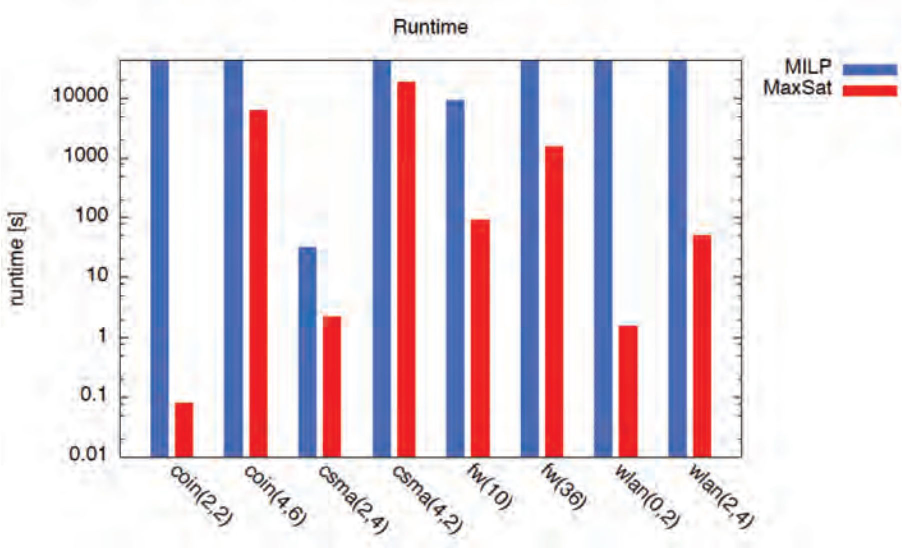
MAXSAT Approach [Dehnert *et al.*, 2014]

1. Use MAX-SAT solver to enumerate possible combinations of commands of minimal size
2. Check their criticality by model checking
3. Analyse non-critical command sets to infer constraints for a "better" solution.

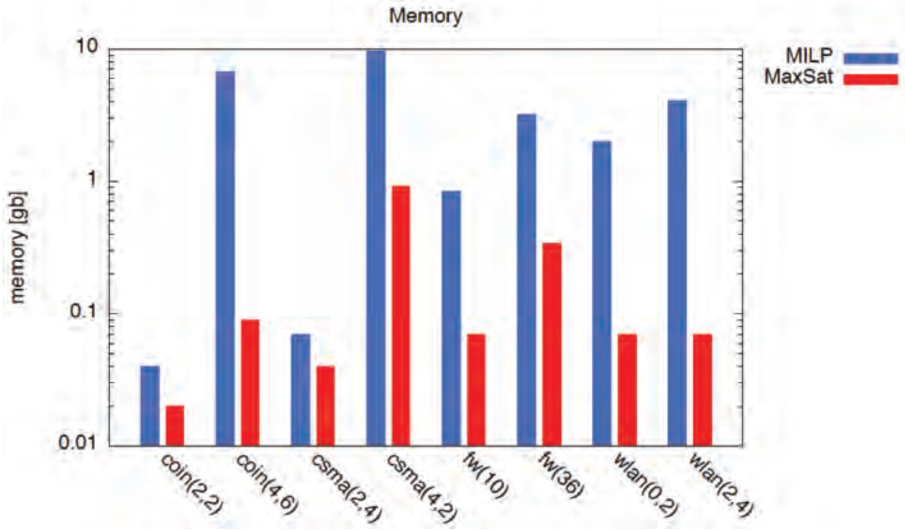
MAX-SAT Approach



Experimental Results



Experimental Results



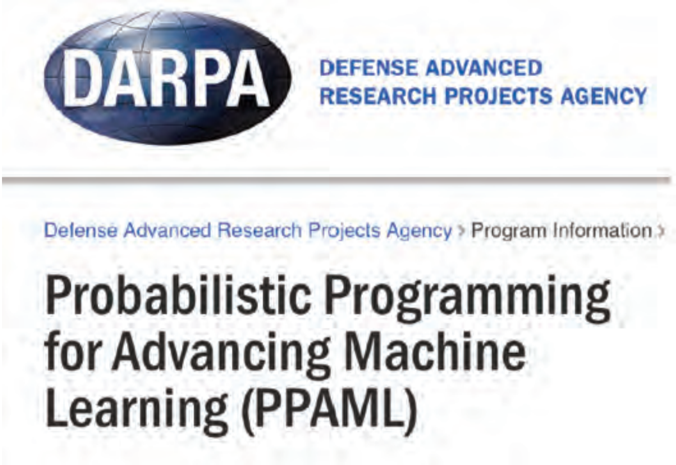
Experiments: Conclusion

Model	Comm.	Cex.	Time (s)	Lower bound	Removed branches
consensus	14	≤ 9	> 600	7	1 / 12
consensus	28	≤ 20	> 600	5	2 / 24
csma	38	36	184.05	—	20 / 90
firewire	68	28	545.68	—	38 / 68
wlan	76	8	0.04	—	6 / 14
wlan	76	≤ 38	> 600	32	31 / 72

- Complimentary technique to minimal critical subsystems
- Extendible to Continuous-time Markov Chains

Overview

Motivation


DARPA
DEFENSE ADVANCED
RESEARCH PROJECTS AGENCY
Defense Advanced Research Projects Agency › Program Information ›
Probabilistic Programming for Advancing Machine Learning (PPAML)

Probabilistic Programs

Probabilistic Programs

What are probabilistic programs?

Sequential, possibly non-deterministic, programs with random assignments.

Probabilistic Programs

What are probabilistic programs?

Sequential, possibly non-deterministic, programs with random assignments.

Applications

Security, machine learning, quantum computing, approximate computing

Probabilistic Programs

What are probabilistic programs?

Sequential, possibly non-deterministic, programs with random assignments.

Applications

Security, machine learning, quantum computing, approximate computing

The scientific challenge

- Such programs are small, but hard to understand and analyse.

Probabilistic Programs

What are probabilistic programs?

Sequential, possibly non-deterministic, programs with random assignments.

Applications

Security, machine learning, quantum computing, approximate computing

The scientific challenge

- Such programs are small, but hard to understand and analyse.
- Problems: infinite variable domains, parameters, and loops.

Probabilistic Programs

What are probabilistic programs?

Sequential, possibly non-deterministic, programs with random assignments.

Applications

Security, machine learning, quantum computing, approximate computing

The scientific challenge

- Such programs are small, but hard to understand and analyse.
- Problems: infinite variable domains, parameters, and loops.

$\Rightarrow$ Aim: **push the limits of their automated analysis**

Program Equivalence

[Kiefer *et al.*, 2012]

```
int XminY1(float p, q){
  int x, f := 0, 0;
  while (f = 0) {
    (x +:= 1 [p] f := 1);
  }
  f := 0;
  while (f = 0) {
    (x -:= 1 [q] f := 1);
  }
  return x;
}
```

```
int XminY2(float p, q){
  int x, f := 0, 0;
  (f := 0 [0.5] f := 1);
  if (f = 0) {
    while (f = 0) {
      (x +:= 1 [p] f := 1);
    }
  } else {
    f := 0;
    while (f = 0) {
      x -:= 1;
      (skip [q] f := 1);
    }
  }
return x;
}
```

Program Equivalence

[Kiefer *et al.*, 2012]

```
int XminY1(float p, q){
  int x, f := 0, 0;
  while (f = 0) {
    (x +:= 1 [p] f := 1);
  }
  f := 0;
  while (f = 0) {
    (x -:= 1 [q] f := 1);
  }
  return x;
}
```

```
int XminY2(float p, q){
  int x, f := 0, 0;
  (f := 0 [0.5] f := 1);
  if (f = 0) {
    while (f = 0) {
      (x +:= 1 [p] f := 1);
    }
  } else {
    f := 0;
    while (f = 0) {
      x -:= 1;
      (skip [q] f := 1);
    }
  }
return x;
}
```

The programs are equivalent for $(p, q) = (\frac{1}{2}, \frac{2}{3})$.

Program Equivalence

[Kiefer *et al.*, 2012]

```
int XminY1(float p, q){
  int  x, f := 0, 0;
  while (f = 0) {
    (x +:= 1 [p] f := 1);
  }
  f := 0;
  while (f = 0) {
    (x -:= 1 [q] f := 1);
  }
  return x;
}
```

```
int XminY2(float p, q){
  int x, f := 0, 0;
  (f := 0 [0.5] f := 1);
  if (f = 0) {
    while (f = 0) {
      (x +:= 1 [p] f := 1);
    }
  } else {
    f := 0;
    while (f = 0) {
      x -:= 1;
      (skip [q] f := 1);
    }
  }
return x;
}
```

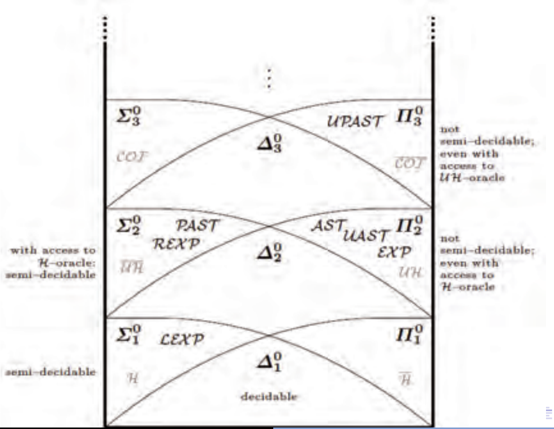
The programs are equivalent for $(p, q) = (\frac{1}{2}, \frac{2}{3})$. Q: No other ones?

Probabilistic Guarded Command Language

▶ `skip`	empty statement
▶ `abort`	abortion
▶ `x := E`	assignment
▶ `prog1 ; prog2`	sequential composition
▶ `if (G) prog1 else prog2`	choice
▶ `prog1 [] prog2`	non-deterministic choice
▶ `prog1 [p] prog2`	probabilistic choice
▶ `while (G) prog`	iteration

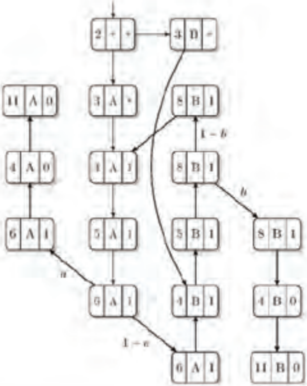
Probabilistic Programs are Hard [Kaminski & Katoen, 2015]

The decision problem whether a pGCL program almost surely terminate on one given input is as hard as the problem whether an ordinary program terminates on all possible inputs.



MDP of duelling cowboys

```
int cowboyDuel(float a, b) {
  int t := A [] t := B;
  bool c := true;
  while (c) {
    if (t = A) {
      (c := false [a] t := B);
    } else {
      (c := false [b] t := A);
    }
  }
  return t;
}
```



This MDP is parameterized but finite. Once we count the number of shots before one of the cowboys dies, the MDP becomes infinite. Our approach however allows to determine e.g., the expected number of shots before success.

Weakest Preconditions

Syntax	Semantics $wp(P, f)$
`skip`	f
`abort`	0
`x := E`	$f[x := E]$
`P1 ; P2`	$wp(P_1, wp(P_2, f))$
`if (G) P1 else P2`	$[G] \cdot wp(P_1, f) + [\neg G] \cdot wp(P_2, f)$
`P1 [] P2`	$\min(wp(P_1, f), wp(P_2, f))$
`P1 [p] P2`	$p \cdot wp(P_1, f) + (1-p) \cdot wp(P_2, f)$
`while (G)P`	$\mu X. ([G] \cdot wp(P, X) + [\neg G] \cdot f)$

μ is the least fixed point operator wrt. the ordering $\leqslant$ on expectations.

Determining Weakest Preconditions is Hard

Correspondence [Gretz *et al.*, 2013]

For `pGCL`-program P, variable valuation η, and post-expectation f, it holds that $wp P, f(\eta)$ equals the expected reward of reaching a terminal state in P's MDP.[3]

[3] All states have reward 0, except terminal states $\langle \varepsilon, \eta' \rangle$ have reward $f(\eta')$.

Determining Weakest Preconditions is Hard

Correspondence [Gretz *et al.*, 2013]

For pGCL-program P, variable valuation η, and post-expectation f, it holds that $wp P, f(\eta)$ equals the expected reward of reaching a terminal state in P's MDP.[3]

Thus: weakest pre-conditions can be obtained as expected rewards in infinite-state parametric MDPs.

This is as hard as solving the universal halting problem!

[3] All states have reward 0, except terminal states $\langle \varepsilon, \eta' \rangle$ have reward $f(\eta')$.

Determining Weakest Preconditions is Hard

Correspondence [Gretz *et al.*, 2013]

For pGCL-program P, variable valuation η, and post-expectation f, it holds that $wpP, f(\eta)$ equals the expected reward of reaching a terminal state in P's MDP.[3]

Thus: weakest pre-conditions can be obtained as expected rewards in infinite-state parametric MDPs.

This is as hard as solving the universal halting problem!

Is there no hope for automation? Well, semi-automation.

[3] All states have reward 0, except terminal states $\langle \varepsilon, \eta' \rangle$ have reward $f(\eta')$.

Loop-Invariant Synthesis [Katoen *et al.*, 2010]

Main steps

Loop-Invariant Synthesis [Katoen *et al.*, 2010]

Main steps

1. Speculatively annotate a program with linear expressions:

$$[\alpha_1 \cdot x_1 + \ldots + \alpha_n \cdot x_n + \alpha_{n+1} \ll 0] \cdot (\beta_1 \cdot x_1 + \ldots + \beta_n \cdot x_n + \beta_{n+1})$$

with real parameters α_i, β_i, program variable x_i, and $\ll \in \{<, \leqslant\}$.

Loop-Invariant Synthesis [Katoen *et al.*, 2010]

Main steps

1. Speculatively annotate a program with linear expressions:

$$[\alpha_1 \cdot x_1 + \ldots + \alpha_n \cdot x_n + \alpha_{n+1} \ll 0] \cdot (\beta_1 \cdot x_1 + \ldots + \beta_n \cdot x_n + \beta_{n+1})$$

   with real parameters α_i, β_i, program variable x_i, and $\ll \in \{<, \leqslant\}$.

2. Transform these numerical constraints into Boolean predicates.

Loop-Invariant Synthesis [Katoen *et al.*, 2010]

Main steps

1. Speculatively annotate a program with linear expressions:

$$[\alpha_1 \cdot x_1 + \ldots + \alpha_n \cdot x_n + \alpha_{n+1} \ll 0] \cdot (\beta_1 \cdot x_1 + \ldots + \beta_n \cdot x_n + \beta_{n+1})$$

   with real parameters α_i, β_i, program variable x_i, and $\ll \in \{ <, \leqslant \}$.

2. Transform these numerical constraints into Boolean predicates.
3. Transform these predicates into non-linear FO formulas.

Loop-Invariant Synthesis [Katoen *et al.*, 2010]

Main steps

1. Speculatively annotate a program with linear expressions:

$$[\alpha_1 \cdot x_1 + \ldots + \alpha_n \cdot x_n + \alpha_{n+1} \ll 0] \cdot (\beta_1 \cdot x_1 + \ldots + \beta_n \cdot x_n + \beta_{n+1})$$

   with real parameters α_i, β_i, program variable x_i, and $\ll \in \{ <, \leqslant \}$.
2. Transform these numerical constraints into Boolean predicates.
3. Transform these predicates into non-linear FO formulas.
4. Use constraint-solvers for quantifier elimination (e.g., Redlog).

Loop-Invariant Synthesis [Katoen *et al.*, 2010]

Main steps

1. Speculatively annotate a program with linear expressions:

$$[\alpha_1 \cdot x_1 + \ldots + \alpha_n \cdot x_n + \alpha_{n+1} \ll 0] \cdot (\beta_1 \cdot x_1 + \ldots + \beta_n \cdot x_n + \beta_{n+1})$$

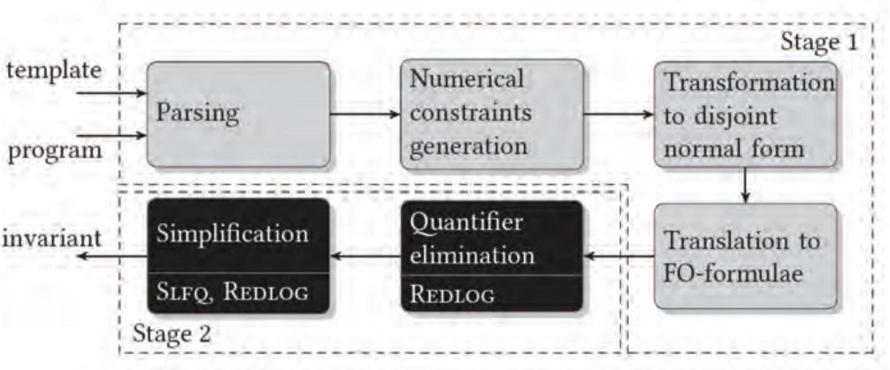
   with real parameters α_i, β_i, program variable x_i, and $\ll \in \{<, \leqslant\}$.
2. Transform these numerical constraints into Boolean predicates.
3. Transform these predicates into non-linear FO formulas.
4. Use constraint-solvers for quantifier elimination (e.g., Redlog).
5. Simplify the resulting formulas (e.g., using Slfq and SMT solving).
6. Exploit resulting assertions to infer program correctness.

Quantitative version of approach by [Colón et al., 2003] for ordinary programs.

Soundness and Completeness

For any linear pGCL program annotated with propositionally linear expressions, this method will find all parameter solutions that make the annotation valid, and no others.

Prinsys: Synthesis Tool of Probabilistic Invariants



download from moves.rwth-aachen.de/prinsys

Program Equivalence

```
int XminY1(float p, q){
  int  x, f := 0, 0;
  while (f = 0) {
    (x +:= 1 [p] f := 1);
  }
  f := 0;
  while (f = 0) {
    (x -:= 1 [q] f := 1);
  }
  return x;
}
```

```
int XminY2(float p, q){
  int x, f := 0, 0;
  (f := 0 [0.5] f := 1);
  if (f = 0) {
    while (f = 0) {
      (x +:= 1 [p] f := 1);
    }
  } else {
    f := 0;
    while (f = 0) {
      x -:= 1;
      (skip [q] f := 1);
    }
  }
return x;
}
```

Program Equivalence

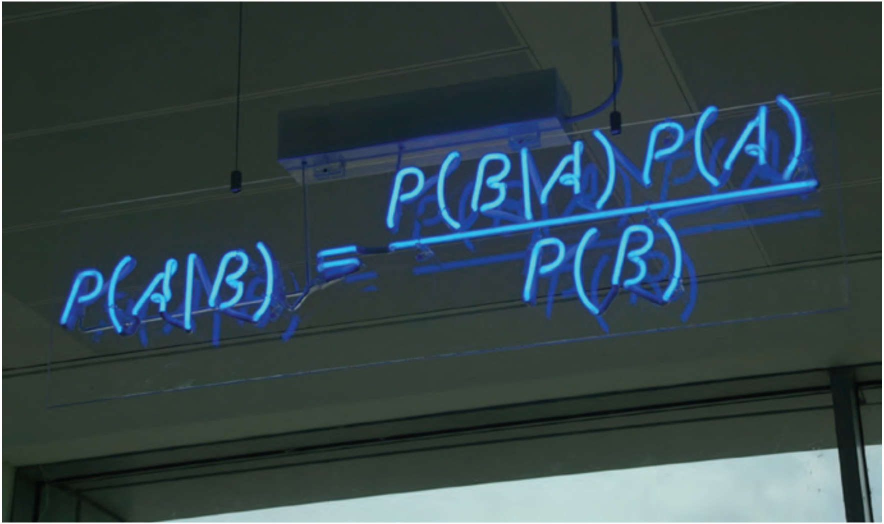
```
int XminY1(float p, q){
  int  x, f := 0, 0;
  while (f = 0) {
    (x +:= 1 [p] f := 1);
  }
  f := 0;
  while (f = 0) {
    (x -:= 1 [q] f := 1);
  }
  return x;
}
```

```
int XminY2(float p, q){
  int x, f := 0, 0;
  (f := 0 [0.5] f := 1);
  if (f = 0) {
    while (f = 0) {
      (x +:= 1 [p] f := 1);
    }
  } else {
    f := 0;
    while (f = 0) {
      x -:= 1;
      (skip [q] f := 1);
    }
  }
return x;
}
```

Analysis with Prinsys yields:

Both programs are equivalent for any q with $q = \frac{1}{2-p}$.

Conditioning



The Piranha Problem

[Tijms, 2004]

One fish is contained within the confines of an opaque fishbowl. The fish is equally likely to be a piranha or a goldfish. A sushi lover throws a piranha into the fish bowl alongside the other fish. Then, immediately, before either fish can devour the other, one of the fish is blindly removed from the fishbowl. The fish that has been removed from the bowl turns out to be a piranha. What is the probability that the fish that was originally in the bowl by itself was a piranha?

The Piranha Problem

```
(f1 := goldfish [0.5] f1 := piranha);
f2 := piranha;
(sample := f1 [0.5] sample := f2);
observe([sample = piranha]);
```

The Piranha Problem

```
(f1 := goldfish [0.5] f1 := piranha);
f2 := piranha;
(sample := f1 [0.5] sample := f2);
observe([sample = piranha]);
```

What is the probability that the original fish in the bowl was a piranha?

The Piranha Problem

```
(f1 := goldfish [0.5] f1 := piranha);
f2 := piranha;
(sample := f1 [0.5] sample := f2);
observe([sample = piranha]);
```

What is the probability that the original fish in the bowl was a piranha?

$$\mathbb{E}(\texttt{f1} \ = \ \texttt{piranha} \mid P \text{ terminates}) = \frac{1 \cdot 1/2 + 0 \cdot 1/4}{1/2 + 1/4} = \frac{1/2}{3/4} = \frac{2}{3}.$$

Infeasible Programs

$$
\begin{aligned}
P:\quad & x := 1;\\
& \mathit{while}(x = 1)\,\{\\
& \quad x := 1\\
& \}
\end{aligned}
\qquad\qquad
\begin{aligned}
Q:\quad & x := 1;\\
& \mathit{while}(x = 1)\,\{\\
& \quad \{x := 1\}\ [0.5]\ \{x := 2\};\\
& \quad \mathit{observe}\,(x = 1);\\
& \}
\end{aligned}
$$

Program P does not terminate. Program Q is infeasible.

Overview

Recent Research Developments

- Parameter Synthesis
- Model Repair
- Counterexample Generation
- Probabilistic Programming

Epilogue

Conclusion

Probabilistic Model Checking . . .

- . . . . . . is a mature automated technique
- . . . . . . focuses on quantitative measures
- . . . . . . has a broad range of applications
- . . . . . . is scalable
- . . . . . . is extendible to costs
- . . . . . . offers many interesting challenges!

Conclusion

Probabilistic Model Checking . . .

- . . . . . . is a mature automated technique
- . . . . . . focuses on quantitative measures
- . . . . . . has a broad range of applications
- . . . . . . is scalable
- . . . . . . is extendible to costs
- . . . . . . offers many interesting challenges!

Current Research

- probabilistic program analysis
- tight game-based abstractions
- parametric verification and synthesis
- stochastic hybrid systems