#SAT based Bounded Model Checking

Deep Karkhanis

Summer 2019

Content

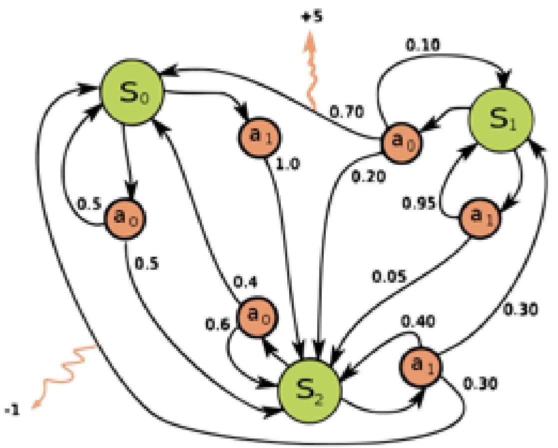
- Background
- BMC for MCs
- Probabilities into #SAT
- BMC for MDPs
- Improvements
- Further Optimizations
- Experiments

Background

- **Markov Decision Process (MDP)**

  A Markov decision process is a 4-tuple (S, A, P_a, R_a)

Background

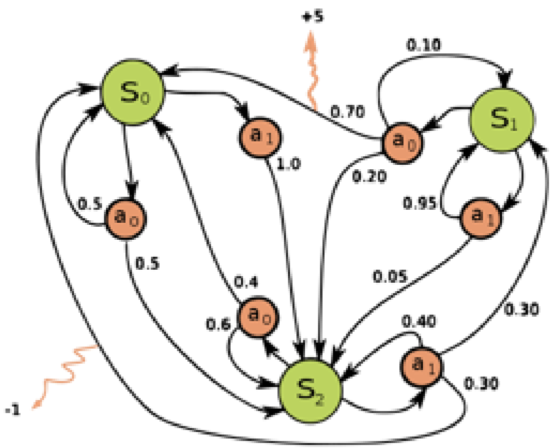


- **Markov Decision Process (MDP)**

A Markov decision process is a 4-tuple (S, A, P_a, R_a) where,

- *S* is a finite set of states
- *A* is a finite set of actions
- $P_a(s, s')$ is the transition probability from state *s* to *s'* on performing action *a*
- R_a is the reward function (immediate reward received on transition from state *s* to *s'* due to action *a*)

Background



➢ Markov Decision Process (MDP)

A Markov decision process is a 4-tuple (S, A, P_a, R_a) where,

- *S* is a finite set of states
- *A* is a finite set of actions
- $P_a(s, s')$ is the transition probability from state *s* to *s'* on performing action *a*
- R_a is the reward function (immediate reward received on transition from state *s* to *s'* due to action *a*)

➢ k-step MDP Policy

A k-step Policy in an MDP is a function:

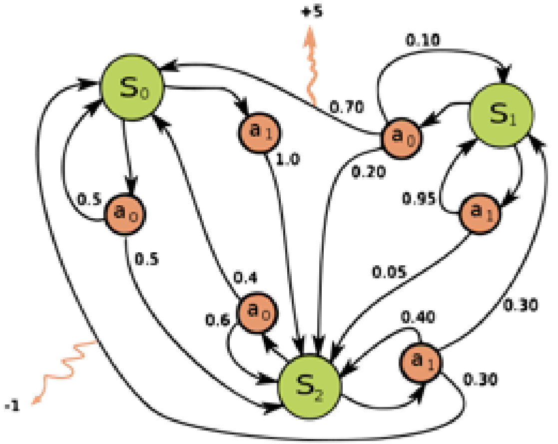
$$\pi : S \times G \to A \quad s.t \;\; G = \{x \mid 0 \leq x \leq k-1, x \,\epsilon\, \mathbb{Z}\}$$

Background

➢ Markov Decision Process (MDP)

A Markov decision process is a 4-tuple (S, A, P_a, R_a) where,

- *S* is a finite set of states
- *A* is a finite set of actions
- $P_a(s, s')$ is the transition probability from state *s* to *s'* on performing action *a*
- R_a is the reward function (immediate reward received on transition from state *s* to *s'* due to action *a*)



➢ k-step MDP Policy

A k-step Policy in an MDP is a function:

$$\pi : S \times G \rightarrow A \quad s.t \;\; G = \{x \mid 0 \leq x \leq k-1, x \in \mathbb{Z}\}$$

Equivalently, a k-step policy specifies the action to be taken at each state for each step made by the agent

A k-step policy induces a Markov Chain with |*S*|*(*k+1*) number of states

Background

- **Model Counting (#SAT)**

  Model Counting, or equivalently the **Sharp-SAT** problem

Background

- **Model Counting (#SAT)**

  Model Counting, or equivalently the **Sharp-SAT** problem

  Counting the number of satisfying assignments of a given propositional formula
  Typically represented as a CNF

Background

- **Model Counting (#SAT)**

Model Counting, or equivalently the **Sharp-SAT** problem

Counting the number of satisfying assignments of a given propositional formula
Typically represented as a CNF

Eg:

No of satisfying assignments of $(a \vee b)$ is 3.
For simplicity, denoted as:

$$\#(a \vee b) = 3$$

Similarly,

$$\#((a \wedge b) \vee c) = 5$$

Software Modeling and Verification Chair | RWTH AACHEN UNIVERSITY

Background

- **Model Counting (#SAT)**

Model Counting, or equivalently the **Sharp-SAT** problem

Counting the number of satisfying assignments of a given propositional formula
Typically represented as a CNF

Eg:

No of satisfying assignments of $(a \vee b)$ is 3.
For simplicity, denoted as:

$$\#(a \vee b) = 3$$

Similarly,

$$\#((a \wedge b) \vee c) = 5$$

Can also specify the variables to count over

Software Modeling and Verification Chair | RWTH AACHEN UNIVERSITY

#SAT for Markov Chains

Problem: Finding Reachability Probability of a target state from a given initial state within k steps

#SAT for Markov Chains

Problem: Finding Reachability Probability of a target state from a given initial state within k steps

Idea: Encode the Markov Chain as a Boolean Formula F such that,

$$\text{Reachability Prob} = \frac{\text{No.of Satisfying Assignments of F}}{\text{No.of possible Assignments to F}} = \frac{\#(F)}{\#(F \vee True)}$$

#SAT for Markov Chains

Problem: Finding Reachability Probability of a target state from a given initial state within k steps
Idea: Encode the Markov Chain as a Boolean Formula F such that,

$$\text{Reachability Prob} = \frac{\text{No.of Satisfying Assignments of F}}{\text{No.of possible Assignments to F}} = \frac{\#(F)}{\#(F \vee True)}$$

- **The SAT formula**

Input : Input variables denote the possible paths that a run of the MC may take
Output: A variable assignment (equivalently, a path) satisfies the formula iff it:

- Starts at the initial state
- Chooses successive states based on the probability distributions
- Ends at a target state

#SAT for Markov Chains

Problem: Finding Reachability Probability of a target state from a given initial state within k steps
Idea: Encode the Markov Chain as a Boolean Formula F such that,

$$\text{Reachability Prob} = \frac{\text{No.of Satisfying Assignments of F}}{\text{No.of possible Assignments to F}} = \frac{\#(F)}{\#(F \vee True)}$$

- **The SAT formula**

Input : Input variables denote the possible paths that a run of the MC may take
Output: A variable assignment (equivalently, a path) satisfies the formula iff it:

- Starts at the initial state
- Chooses successive states based on the probability distributions
- Ends at a target state

Assign a bit $s_{(j,i)}$ each for the $|S|*(k+1)$ locations possible in a MC run
Thus, the formula F can be written as:

Software Modeling and Verification Chair
RWTH AACHEN UNIVERSITY

#SAT for Markov Chains

Problem: Finding Reachability Probability of a target state from a given initial state within k steps
Idea: Encode the Markov Chain as a Boolean Formula F such that,

$$\text{Reachability Prob} = \frac{\text{No.of Satisfying Assignments of F}}{\text{No.of possible Assignments to F}} = \frac{\#(F)}{\#(F \vee True)}$$

- **The SAT formula**

Input : Input variables denote the possible paths that a run of the MC may take
Output: A variable assignment (equivalently, a path) satisfies the formula iff it:

- Starts at the initial state
- Chooses successive states based on the probability distributions
- Ends at a target state

Assign a bit $s_{(j,i)}$ each for the |*S*|*(*k*+*1*) locations possible in a MC run
Thus, the formula *F* can be written as:

$$F = (s_{(0,0)} \wedge (\bigwedge_{j=1}^{|S|-1} \overline{s_{(j,0)}})) \wedge (\bigwedge_{i=0}^{k-1} \bigwedge_{j=0}^{|S|-1} PE_\pi(s_{(j,i)})) \wedge (\bigvee_{i=0}^{k} s_{(|S|-1,i)})$$

where, $PE_\pi(s_{(j,i)})$ encodes the probability distribution for state no *j* at step *i+1*

Encoding Probabilities

- **Biased Coins**

The fraction of assignments that satisfy a given Boolean formula is the probability that the formula encodes

Unbiased Coin: (p=0.5)$\equiv c$ $\quad \frac{\#(c)}{2} = 0.5$

Biased coin: (p=0.75) $\equiv (c_1 \vee c_2)$ $\quad \frac{\#(c_1 \vee c_2)}{2^2} = 0.75$

Encoding Probabilities

- **Biased Coins**

The fraction of assignments that satisfy a given Boolean formula is the probability that the formula encodes

Unbiased Coin: (p=0.5)$\equiv c$ $\quad \frac{\#(c)}{2} = 0.5$

Biased coin: (p=0.75) $\equiv (c_1 \vee c_2)$ $\quad \frac{\#(c_1 \vee c_2)}{2^2} = 0.75$

Equivalently, each variable represents an independent coin flip. (Every input configuration is equi-probable)
Interpret the vector of coin variables as a binary integer, *x*

Encoding Probabilities

- **Biased Coins**

The fraction of assignments that satisfy a given Boolean formula is the probability that the formula encodes

Unbiased Coin: (p=0.5)$\equiv c$ $\quad \frac{\#(c)}{2} = 0.5$

Biased coin: (p=0.75) $\equiv (c_1 \vee c_2)$ $\quad \frac{\#(c_1 \vee c_2)}{2^2} = 0.75$

Equivalently, each variable represents an independent coin flip. (Every input configuration is equi-probable)
Interpret the vector of coin variables as a binary integer, *x*

Allowing the output to be True iff $x < m$: $\Pr(x < m) = \frac{m}{2^n}$ where, n is the number of coins

Every probability having a terminating binary representation can thus be encoded

Encoding Probabilities

- **Biased Coins**

The fraction of assignments that satisfy a given Boolean formula is the probability that the formula encodes

Unbiased Coin: (p=0.5)$\equiv c$ $\quad \dfrac{\#(c)}{2} = 0.5$

Biased coin: (p=0.75) $\equiv (c_1 \vee c_2)$ $\quad \dfrac{\#(c_1 \vee c_2)}{2^2} = 0.75$

Equivalently, each variable represents an independent coin flip. (Every input configuration is equi-probable)
Interpret the vector of coin variables as a binary integer, *x*

Allowing the output to be True iff *x* < *m* : $\Pr(x < m) = \dfrac{m}{2^n}$ where, n is the number of coins

Every probability having a terminating binary representation can thus be encoded

For any rational number *p/q* : $\Pr(x < p \mid x < q)$ = $\dfrac{\Pr(x < p)}{\Pr(x < q)} = \dfrac{\frac{p}{2^n}}{\frac{q}{2^n}} = \dfrac{p}{q}$

That is, only allow *q* inputs to be "valid" and *p* of them to be accepting

Encoding Probabilities

- **Knuth-Yao Encoding**

  A concise representation for Probability Mass Functions

Encoding Probabilities

- **Knuth-Yao Encoding**

A concise representation for Probability Mass Functions

- Represent the probabilities as binary numbers
- Create a binary tree such that if the bit of the expansion of $Pr(A)$ is 1, then A appears as a leaf at depth j
  where, $A \in S \quad s.t\, S$ is a Mutually Exclusive and Collective Exhaustive set

Encoding Probabilities

➢ **Knuth-Yao Encoding**

A concise representation for Probability Mass Functions

- Represent the probabilities as binary numbers
- Create a binary tree such that if the bit of the expansion of $Pr(A)$ is 1, then A appears as a leaf at depth j
  where, $A \in S \quad s.t\, S$ is a Mutually Exclusive and Collective Exhaustive set

- Assign an unbiased coin (equivalently, a boolean variable) for each depth of the tree
  The decisions in the tree are modelled by these coin flips

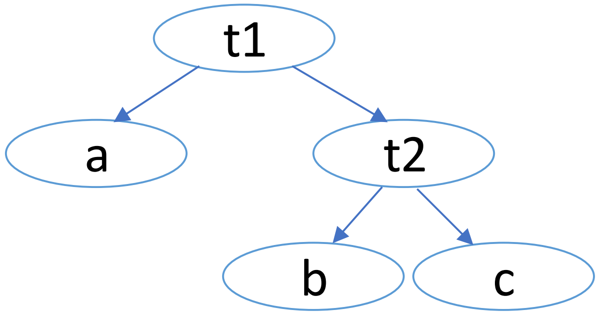
Encoding Probabilities

- **Knuth-Yao Encoding**

A concise representation for Probability Mass Functions

- Represent the probabilities as binary numbers
- Create a binary tree such that if the bit of the expansion of $Pr(A)$ is 1, then A appears as a leaf at depth j
  where, $A \in S \quad s.t\, S$ is a Mutually Exclusive and Collective Exhaustive set

- Assign an unbiased coin (equivalently, a boolean variable) for each depth of the tree
  The decisions in the tree are modelled by these coin flips



Eg: $S = \{A,B,C\} \quad Pr(A)=0.5 \quad Pr(B)=0.25 \quad Pr(C) = 0.25$

In binary, $Pr(A) = 0.1 \quad Pr(B) = 0.01 \quad Pr(C) = 0.01$

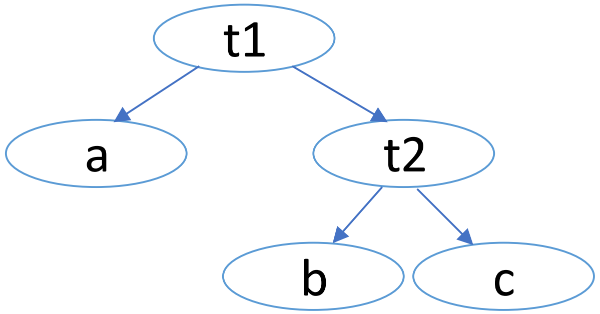
Encoding Probabilities

➢ **Knuth-Yao Encoding**

A concise representation for Probability Mass Functions

- Represent the probabilities as binary numbers
- Create a binary tree such that if the bit of the expansion of *Pr*(*A*) is 1, then A appears as a leaf at depth *j*
  where, $A \in S$ $s.t\, S$ is a Mutually Exclusive and Collective Exhaustive set

- Assign an unbiased coin (equivalently, a boolean variable) for each depth of the tree
  The decisions in the tree are modelled by these coin flips



Eg: *S = {A,B,C}* *Pr(A)=0.5* *Pr(B)=0.25* *Pr(C) = 0.25*

In binary, *Pr(A) = 0.1* *Pr(B) = 0.01* *Pr(C) = 0.01*

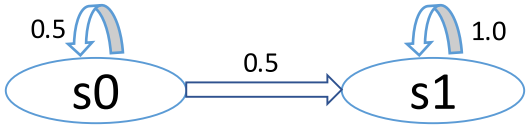
$$F = (a \leftrightarrow (t_1 \wedge \overline{c_1})) \wedge (t_2 \leftrightarrow (t_1 \wedge c_1)) \wedge (b \leftrightarrow (t_2 \wedge \overline{c_2})) \wedge (c \leftrightarrow (t_2 \wedge c_2))$$

#SAT for Markov Chains

- **Example**

|*S*|=2, *k*=2

$$\Pr(s_0, s_0) = 0.5 \equiv 0.1 \quad \Pr(s_0, s_1) = 0.5 \equiv 0.1 \quad \Pr(s_1, s_1) = 1.0 \equiv 1.0$$

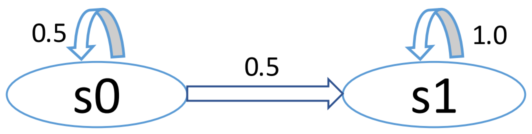


#SAT for Markov Chains

- **Example**

$|S|=2$, $k=2$

$$\Pr(s_0, s_0) = 0.5 \equiv 0.1 \quad \Pr(s_0, s_1) = 0.5 \equiv 0.1 \quad \Pr(s_1, s_1) = 1.0 \equiv 1.0$$



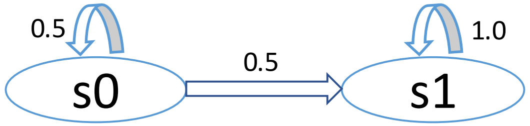
$$F = [(s_{(0,0)} \wedge \overline{s_{(1,0)}})] \bigwedge [s_{(1,0)} \vee s_{(1,1)} \vee s_{(1,2)}]$$

#SAT for Markov Chains

- **Example**

$|S|=2$, $k=2$

$$\Pr(s_0, s_0) = 0.5 \equiv 0.1 \quad \Pr(s_0, s_1) = 0.5 \equiv 0.1 \quad \Pr(s_1, s_1) = 1.0 \equiv 1.0$$



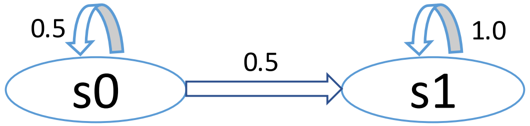
$$\begin{aligned} F = {} & [(s_{(0,0)} \wedge \overline{s_{(1,0)}})] \bigwedge [s_{(1,0)} \vee s_{(1,1)} \vee s_{(1,2)}] \\ & \bigwedge [(s_{(0,1)} \leftrightarrow (s_{(0,0)} \wedge \overline{c_1})) \wedge (t_1 \leftrightarrow (s_{(0,0)} \wedge c_1)) \wedge (t_2 \leftrightarrow s_{(1,0)}) \wedge (s_{(1,1)} \leftrightarrow (t_1 \vee t_2)) \\ & \wedge (s_{(0,2)} \leftrightarrow (s_{(0,1)} \wedge \overline{c_2})) \wedge (t_3 \leftrightarrow (s_{(0,1)} \wedge c_2)) \wedge (t_4 \leftrightarrow s_{(1,1)}) \wedge (s_{(1,2)} \leftrightarrow (t_3 \vee t_4))] \end{aligned}$$

#SAT for Markov Chains

- **Example**

$|S|=2$, $k=2$

$$\Pr(s_0, s_0) = 0.5 \equiv 0.1 \quad \Pr(s_0, s_1) = 0.5 \equiv 0.1 \quad \Pr(s_1, s_1) = 1.0 \equiv 1.0$$



$$\begin{aligned} F = {} & [(s_{(0,0)} \wedge \overline{s_{(1,0)}})] \bigwedge [s_{(1,0)} \vee s_{(1,1)} \vee s_{(1,2)}] \\ & \bigwedge [(s_{(0,1)} \leftrightarrow (s_{(0,0)} \wedge \overline{c_1})) \wedge (t_1 \leftrightarrow (s_{(0,0)} \wedge c_1)) \wedge (t_2 \leftrightarrow s_{(1,0)}) \wedge (s_{(1,1)} \leftrightarrow (t_1 \vee t_2)) \\ & \wedge (s_{(0,2)} \leftrightarrow (s_{(0,1)} \wedge \overline{c_2})) \wedge (t_3 \leftrightarrow (s_{(0,1)} \wedge c_2)) \wedge (t_4 \leftrightarrow s_{(1,1)}) \wedge (s_{(1,2)} \leftrightarrow (t_3 \vee t_4))] \end{aligned}$$

$$F = c_1 \vee c_2$$

$$\frac{\#(F)}{2^2} = \frac{3}{4}$$

#SAT for MDPs

Problem: Does there exist a k-step Policy which ensures that the Reachability Probability of a target state from a given initial state within k steps exceeds a certain threshold?

$$\Pr_{\pi}(\Diamond_{\leq k}\ (s_{|S|-1})) \geq \lambda \quad ?$$

#SAT for MDPs

Problem: Does there exist a k-step Policy which ensures that the Reachability Probability of a target state from a given initial state within k steps exceeds a certain threshold?

$$\Pr_{\pi}(\Diamond_{\leq k}\ (s_{|S|-1})) \geq \lambda \quad ?$$

Idea: Allow for control bits which enforce policies. Given a policy, these action bits are uniquely set. Upon enforcing these action bits, formula represents the induced Markov Chain

#SAT for MDPs

Problem: Does there exist a k-step Policy which ensures that the Reachability Probability of a target state from a given initial state within k steps exceeds a certain threshold?

$$\Pr_{\pi}(\Diamond_{\leq k}\ (s_{|S|-1})) \geq \lambda\ \ ?$$

Idea: Allow for control bits which enforce policies. Given a policy, these action bits are uniquely set. Upon enforcing these action bits, formula represents the induced Markov Chain

Each sub-formula corresponding to the pmf will be relevant iff the action bits corresponding to it are set

$$\bigwedge_{h=0}^{|A|-1} \left(PE(s_{(j,i,h)}) \leftrightarrow a_{(j,i,h)}\right)$$

The values of action bits which give the maximum SAT-count will represent the optimal policy

#SAT for MDPs

Problem: Does there exist a k-step Policy which ensures that the Reachability Probability of a target state from a given initial state within k steps exceeds a certain threshold?

$$\Pr_{\pi}(\Diamond_{\leq k}\ (s_{|S|-1})) \geq \lambda \ \ ?$$

Idea: Allow for control bits which enforce policies. Given a policy, these action bits are uniquely set. Upon enforcing these action bits, formula represents the induced Markov Chain

Each sub-formula corresponding to the pmf will be relevant iff the action bits corresponding to it are set

$$\bigwedge_{h=0}^{|A|-1} (PE(s_{(j,i,h)}) \leftrightarrow a_{(j,i,h)})$$

The values of action bits which give the maximum SAT-count will represent the optimal policy

Essentially the **Max#SAT** problem

Policy Iteration is a more scalable approach

Optimizing the Encoding

- Binary encoding for states
  - Specifying that a particular state is reached would require a $log|S|$ length long formula

Optimizing the Encoding

- Binary encoding for states
  - Specifying that a particular state is reached would require a $log|S|$ length long formula
  - Most #SAT solvers only accept CNF formulas
  - Need to efficiently convert the formula into CNF
    - Tseitin Encoding

Optimizing the Encoding

- Binary encoding for states
  - Specifying that a particular state is reached would require a $log|S|$ length long formula
  - Most #SAT solvers only accept CNF formulas
  - Need to efficiently convert the formula into CNF
    - Tseitin Encoding

- Do not encode action bits
  - Precompute and generate encoding for each of the $|S|*|A|$ Probability distributions

Optimizing the Encoding

- Binary encoding for states
  - Specifying that a particular state is reached would require a $log|S|$ length long formula
  - Most #SAT solvers only accept CNF formulas
  - Need to efficiently convert the formula into CNF
    - Tseitin Encoding

- Do not encode action bits
  - Precompute and generate encoding for each of the $|S|*|A|$ Probability distributions
  - Given a policy, stitch the Boolean formula on the fly
  - Formula generation is the bottleneck

Further Improvements

➢ **Binary Decision Diagrams**

For a more concise encoding of the pmfs, use the BDD representation of the transition function.
The BDD has leaves as probabilities.

Further Improvements

- **Binary Decision Diagrams**

  For a more concise encoding of the pmfs, use the BDD representation of the transition function.
  The BDD has leaves as probabilities.
  A bit string (which encodes the transition to be taken) is used as input and the BDD is traversed.
  The leaf reached gives the transition probability.

Further Improvements

- **Binary Decision Diagrams**

For a more concise encoding of the pmfs, use the BDD representation of the transition function.
The BDD has leaves as probabilities.
A bit string (which encodes the transition to be taken) is used as input and the BDD is traversed.
The leaf reached gives the transition probability.

Thus a single highly optimized tree with max-depth = $2*\log|S|$
Rather than $|S|$ trees (depths relying on precision of probability distribution).

Further Improvements

- **Binary Decision Diagrams**

  For a more concise encoding of the pmfs, use the BDD representation of the transition function.
  The BDD has leaves as probabilities.
  A bit string (which encodes the transition to be taken) is used as input and the BDD is traversed.
  The leaf reached gives the transition probability.

  Thus a single highly optimized tree with max-depth = $2*\log|S|$
  Rather than $|S|$ trees (depths relying on precision of probability distribution).

- **Incremental #SAT solvers**

  Most #SAT solvers keep counts for sub-formulas.
  Can potentially reuse them

Further Improvements

- **Binary Decision Diagrams**

  For a more concise encoding of the pmfs, use the BDD representation of the transition function.
  The BDD has leaves as probabilities.
  A bit string (which encodes the transition to be taken) is used as input and the BDD is traversed.
  The leaf reached gives the transition probability.

  Thus a single highly optimized tree with max-depth = $2*\log|S|$
  Rather than $|S|$ trees (depths relying on precision of probability distribution).

- **Incremental #SAT solvers**

  Most #SAT solvers keep counts for sub-formulas.
  Can potentially reuse them
  Policy Iteration primarily only changes a single action choice => only a few clauses are altered

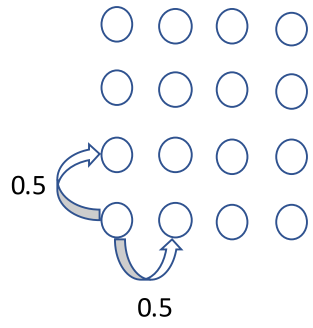
Software Modeling and Verification Chair | RWTH AACHEN UNIVERSITY

Experimental Results

PC Specs: Intel i5, 8GB RAM

	States in k-MDP	Variable Count in Formula	Clauses in Formula
2-state MDP (k=10)	22	37	78
Finite Reachability (k=10)	44	81	131
Grid MDP (5x5)	200	312	508
Grid MDP optimized (5x5)	200	24	126

	Storm (DRN input)	#SAT with approxMC3
5x5	0.012s	0.023s
15x15	0.027s	0.114
100x100	>1min	7.213s



Making use of Specificity

- **Easy jump into Specificity**

  The #SAT approach allows a lot of freedom in specifying the encoding.
  Most MDPs have special patterns and properties.

Making use of Specificity

- **Easy jump into Specificity**

  The #SAT approach allows a lot of freedom in specifying the encoding.
  Most MDPs have special patterns and properties.

  Thus allowing much more concise formulas while also retaining the power of well-known MDP solving techniques

Making use of Specificity

- **Easy jump into Specificity**

  The #SAT approach allows a lot of freedom in specifying the encoding.
  Most MDPs have special patterns and properties.

  Thus allowing much more concise formulas while also retaining the power of well-known MDP solving techniques

  Eg: **Simplified Grid Problem**

  The formula can be as simple as a circuit which adds k-bits