Lecture 2: Symbolic Model Checking With SAT
Symbolic Model Checking with BDDs

Method used by most “industrial strength” model checkers:

- uses Boolean encoding for state machine and sets of states.
- can handle much larger designs – hundreds of state variables.
- BDDs traditionally used to represent Boolean functions.
Problems with BDDs

BDDs are a canonical representation. Often become too large.

- Oftentimes, no space efficient variable ordering exists.
- Often time consuming or needs manual intervention.

Selecting right variable ordering very important for obtaining small BDDs.

- Variable ordering must be uniform along paths.

We describe an alternative approach to symbolic model checking that uses SAT procedures.
Advantages of SAT Procedures

- Very efficient implementations available.
- Different split orderings possible on different branches.
- Do not suffer from the potential space explosion of BDDs.
- SAT procedures also operate on Boolean expressions but do not use canonical forms.

SAT procedures also operate on Boolean expressions but do not use canonical forms.
Bounded Model Checking
(Clark, Biere, Cimatti, Fujita, Zhu)

Bounded model checking uses a SAT procedure instead of BDDs.

We construct Boolean formula that is satisfiable iff there is a counterexample of length $k$. We look for longer and longer counterexamples by incrementing the bound $k$. After some number of iterations, we may conclude no counterexample exists and specification holds.

For example, to verify safety properties, number of iterations is bounded by diameter of finite state machine.
Main Advantages of Our Approach

- Bounded model checking of LTL formulas does not require a tableau or automaton construction.
- Bounded model checking finds counterexamples fast. This is due to depth first nature of SAT search procedures.
- Splitting heuristics usually sufficient. Default splitting heuristics of minimal length. This feature helps user understand nature of SAT search procedures.
- It finds counterexamples of minimal length. This feature helps user understand counterexample more easily.
- Does not need manually selected variable order or costly reordering. Default splitting heuristics usually sufficient.
- It uses much less space than BDD based approaches.
We have implemented a tool BMC for our approach. It accepts a subset of the SMV language. Given $k$, BMC outputs a formula that is satisfiable iff counterexample exists of length $k$. If counterexample exists, a standard SAT solver generates a truth assignment for the formula.
Performance

We give examples where BMC significantly outperforms BDD based model.

In some cases BMC detects errors instantly, while SMV fails to construct BDD for initial state.
Outline

- Bounded Model Checking:
  - Definitions and notation.
  - Example to illustrate bounded model checking.
  - Reduction of bounded model checking for LTL to SAT.
  - Tuning SAT checkers for bounded model checking.
  - Efficient computation of diameters.

- Abstraction/refinement with SAT

- Directions for future research.

- Tuning SAT checkers for bounded model checking.

- Experimental results.
We use linear temporal logic (LTL) for specifications.

Basic Definitions and Notation
System described as a Kripke structure $\langle \mathcal{S}, \mathcal{T}, I, \mathcal{L}, W \rangle$, where

- $\mathcal{S}$ is a finite set of states,
- $I$ is the set of initial states,
- $\mathcal{T}$ is the transition relation, and
- $\mathcal{L}$ is the state labeling.

We assume every state has a successor state.
Definitions and Notation (Cont.)

In symbolic model checking, a state is represented by a vector of state variables.

We define propositional formulas as follows:

1. $(s)\neg \equiv d$ iff $(s)^{df}$
2. $(s)\lor \equiv (s)\land (s)^{df}$
3. $(s)\land \equiv (s)^{df}$
4. $(s)^{df}$
5. $(\neg s)$

We write $\neg$ instead of $\neg$(s), etc.

In symbolic model checking, a state is represented by a vector of state variables...
Does a witness exist for the LTL formula?

Kripke structure, equivalently,

Model checking is the problem of determining the truth of an LTL formula in a

\( I \models (f \varphi \models I_{\mathcal{W}}) \) if there is a path \( \varphi \) in \( \mathcal{W} \) with \( \mathcal{W} \models \varphi \)

\( I \models (f \models I_{\mathcal{W}}) \) if for all \( I \)

\( \not \exists \text{ a path in } I \)

\( (s_0, s_1, \ldots, s_{|f|}) = I \) (then \( f = \) when \( s \not\models \) for all \( I \) and

\( \not\exists \text{ a witness } \)
Example to Illustrate New Technique

Have deliberately added erroneous transition!

\[(\lnot 0, s) \lor (1, s) \lor (0, s) \lor (1, s) \land (s, inc) \quad = 
\]

Define \( T(s, s) \cdot (s, inc) \quad = 
\]

\[(\lnot [1 \oplus s] \leftrightarrow [1, s]) \lor ([0, s] \leftrightarrow [0, s]) = (s, s) \quad = 
\]

In initial state, value of the counter is 0. Thus, \( [1, s] \)

Each state \( s \) is represented by two state variables \( s \) and \( s \).

Two-bit counter with an erroneous transition:

\[
\begin{array}{c}
00 \\
| \\
01 \\
| \\
11 \\
| \\
01 \\
\end{array}
\]
Suppose we want to know if counter will eventually reach state $s$. Can specify the property by $\varphi = \text{counter} \in [s]$. This is expressed by $\mathbf{EG} \varphi$, where $\mathbf{EG} \varphi \equiv [0] \varphi \land [1] \varphi$. On all execution paths, there is a state where $\varphi$ holds. Equivalently, we can check if there is a path on which counter never reaches state $s$.

There exists a path such that $\varphi$ holds globally along it.

$\mathbf{EF} \varphi \equiv [0] \varphi \lor [1] \varphi$. Suppose we want to know if counter will eventually reach state $s$. (Cont.)
In bounded model checking, we consider paths of length $k$. We start with $k = 0$ and increment $k$ until a witness is found. Assume $k$ equals $2$. Call the states $s_0$, $s_1$, and $s_2$. We formulate constraints on $s_0$, $s_1$, and $s_2$ in propositional logic.

We formulate constraints that $(s_0, s_1, s_2)$ is a witness for $Bp$ and, hence, a counterexample for $Af$.
First, we constrain $(s^0, s_1, s_2)$ to be a valid path starting from the initial state.

Obtain a propositional formula

\[(\mathcal{L} \lor (s^1, s_1, s_2)) \land \mathcal{L} \lor (s^0) I = [W]\]
Second, we constrain the shape of the path.

The sequence of states $s_0, s_1, s_2$ can be a loop.

The sequence of states $s_0, s_1, s_2$ can be a loop.

Second, we constrain the shape of the path.

Example (Cont.)
The temporal property must hold on $\exists z$. If no loop exists, $C_{P}$ does not hold and $[\neg true]$ is false. Finally, $P$ must hold at every state on the path previously given.

To be a witness for $C_{P}$, the path must contain a loop (condition $L$ given). The temporal property $C_{P}$ must hold on $(s_{0}, s_{1}, s_{2}, \ldots)$.

Example (Cont.)
Example (Cont.)

In this example, the formula is satisfiable.

° Self-loop at (10).

° Truth assignment corresponds to counterexample path (00), (10), (01), (10) followed by self-loop at (10).

° If self-loop at (10) is removed, then formula is unsatisfiable.

Ex:ample (Cont.)
Model Checking: 16x16 bit sequential shift and add multiplier with overflow flag and

### Sequential Multiplier Example

<table>
<thead>
<tr>
<th>106</th>
<th>23970</th>
<th>2202</th>
<th>71923</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1</td>
<td>4449</td>
<td></td>
<td>15</td>
</tr>
<tr>
<td>4</td>
<td>226</td>
<td>4752</td>
<td></td>
<td>14</td>
</tr>
<tr>
<td>4</td>
<td>186</td>
<td>712</td>
<td></td>
<td>13</td>
</tr>
<tr>
<td>4</td>
<td>156</td>
<td>1080</td>
<td></td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>125</td>
<td>367</td>
<td></td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>187</td>
<td>1</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>58</td>
<td>856</td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>93</td>
<td>473</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>584</td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>12</td>
<td>102</td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>92</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>13</td>
<td>82</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>8</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>6</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

<1GB

17h

18371

23671

4449

1792

Model Checking: 16x16 bit sequential shift and add multiplier with overflow flag and
Model Checking: Liveness for one user in the DME.

<table>
<thead>
<tr>
<th>$\gamma$</th>
<th>$\kappa$</th>
<th>168</th>
<th>117</th>
<th>8</th>
<th>11</th>
<th>1</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>164</td>
<td>14</td>
<td>10</td>
<td>8</td>
<td>22</td>
<td>10</td>
<td>14</td>
</tr>
<tr>
<td>12</td>
<td>131</td>
<td>12</td>
<td>91</td>
<td>8</td>
<td>61</td>
<td>6</td>
<td>13</td>
</tr>
<tr>
<td>11</td>
<td>104</td>
<td>11</td>
<td>80</td>
<td>9</td>
<td>16</td>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td>11</td>
<td>100</td>
<td>11</td>
<td>38</td>
<td>9</td>
<td>13</td>
<td>8</td>
<td>11</td>
</tr>
<tr>
<td>10</td>
<td>64</td>
<td>10</td>
<td>36</td>
<td>5</td>
<td>10</td>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>44</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>32</td>
<td>12</td>
<td>16</td>
<td>6</td>
<td>5</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>22</td>
<td>15</td>
<td>9</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>19</td>
<td>7</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>6</td>
<td>9</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

$\text{secMB} = \frac{\text{sec}}{\text{MB}}$

$\gamma = 10$

$\kappa = 10$

$\kappa = 7$

$\kappa = 5$

$\kappa = 2$

$\text{cells}$

$\text{DME Example}$
Model Checking: Counterexample for liveness in a buggy DME implementation.

<table>
<thead>
<tr>
<th>Cells</th>
<th>SMT2</th>
<th>SMV</th>
<th>SatO</th>
<th>ProvE</th>
<th>sec MB</th>
<th>sec MB</th>
<th>sec MB</th>
<th>sec MB</th>
<th>sec MB</th>
<th>sec MB</th>
<th>sec MB</th>
<th>sec MB</th>
<th>sec MB</th>
<th>sec MB</th>
<th>sec MB</th>
<th>sec MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

"Buggy" DME Example
Usethevariabledependencygraphfor
derivingastaticvariableordering.

The transition relation appears \( k \) times in \( \varphi \), each time with different variables.

\[
\begin{align*}
&\exists \vec{d} \, \wedge \forall \vec{v} \, \left( (1^{+} \vec{s}, \vec{s}, \vec{s}) \cup \left( \bigvee_{\vec{v}} \forall \vec{v} \vdash \vec{d} : \varphi \right) \right) \\
&\wedge \bigvee_{\vec{v}} \forall \vec{v} \vdash \vec{d} : \varphi
\end{align*}
\]

This symmetry indicates that under certain conditions, for each conflict clause we can compute additional \( k - 1 \) clauses, for free.

Use the regular structure of \( \text{AGP} \) formulas to replicate conflict clauses.

Use the variable dependency graph for deriving a static variable ordering.
Use the incremental nature of BMC to reuse conflict clauses.

Restrict decisions to model variables only (ignore CNF auxiliary vars).

It is possible to decide the formula without the auxiliary variables (they will be implied). In many examples they are 80%-90% of the variables in the CNF instance. Some of the clauses that were computed while solving BMC with $k=9$ can be reused when solving the subsequent instance with $k=10$. 

Restrict decisions to model variablesonly (ignore CNF auxiliary vars).
**RB2** - RuleBase second run (without BDD dynamic reordering).

**RB1** - RuleBase first run (with BDD dynamic reordering).

RuleBase is IBM’s BDD based symbolic model-checker.

<table>
<thead>
<tr>
<th>Design #</th>
<th>Grasp</th>
<th>RB1</th>
<th>RB2</th>
<th>Grasp</th>
<th>Tuned</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>3626</td>
<td>1419</td>
<td>142</td>
<td>131</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9035</td>
<td>803</td>
<td>803</td>
<td>131</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6317</td>
<td>690</td>
<td>690</td>
<td>131</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2671</td>
<td>2671</td>
<td>2671</td>
<td>131</td>
</tr>
<tr>
<td></td>
<td></td>
<td>356</td>
<td>356</td>
<td>356</td>
<td>131</td>
</tr>
<tr>
<td></td>
<td></td>
<td>984</td>
<td>984</td>
<td>984</td>
<td>131</td>
</tr>
<tr>
<td>2</td>
<td>24</td>
<td>510</td>
<td>510</td>
<td>510</td>
<td>131</td>
</tr>
<tr>
<td>3</td>
<td>76</td>
<td>375</td>
<td>375</td>
<td>375</td>
<td>131</td>
</tr>
<tr>
<td>0.8</td>
<td>1.1</td>
<td>70</td>
<td>70</td>
<td>70</td>
<td>131</td>
</tr>
<tr>
<td>3</td>
<td>282</td>
<td>184</td>
<td>184</td>
<td>184</td>
<td>131</td>
</tr>
</tbody>
</table>

BMC of some hardware designs w/wo tuning SAT
This requires an efficient QBF checker:

\[(s)^R : \forall y \geq f \exists s \iff (s)^{1,\infty} R : s \land \exists f \exists s \iff s \implies \text{State } s \text{ is reachable in } f \text{ steps}.

\[
\frac{0=\gamma}{(1+\gamma)} \bigvee_{i=1}^{0} \forall s^{0} I \lor \forall s = s : \forall s^{1} \ldots \forall s^{0} E =: (s)^{R}
\]

Thus, \( \gamma \) is greater or equal than the diameter \( d \).

Finding \( d \) is computationally hard:

- Finding \( d \) is computationally hard.
- Finding \( d \) is computationally hard.
- Finding \( d \) is computationally hard.
- Finding \( d \) is computationally hard.

Diameter \( d \): Least number of steps to reach all reachable states. If the property holds for \( k \geq d \), the property holds for all reachable states.
A Compromise: Recurrence Diameter $d$ is an upper bound for the Diameter $d$

\[ a = p \]

steps, i.e., $d = 4$

have a minimum length of four.

- All paths with at least one cycle

\[ b = p \]

in two steps, i.e., $d = 2$

- All states are reachable from $s_0$

Example:

- Have at least one cycle

Recurrent Diameter $d$: Least number of steps $u$ such that all valid paths of length
Testing the Recurrence Diameter

Too expensive for big $k$

Size of CNF: $O(2^k)$

Find cycles by comparing all states with each other

Recurrence Diameter test in BMC:
Recurrence Diameter Test using Sorting Networks (D. Kroening)

Idea: Look for cycles using a Sorting Network

First, sort symbolically:

\[
(\forall s = \beta s \lor \not s \neq 1 : \not s, \not E) \iff (1+\bar{s} = \bar{s} : \not E)
\]

Now only check neighbors in the sorted sequence:

\[
\begin{align*}
\beta s & \leq \cdots \leq 1_s & 0_s & \text{s.t. } s_0, \ldots, s_0 \text{ are permutation of } 0, \ldots, 0
\end{align*}
\]

Biconic sort, have size \(O(\log \log n)\).

Practical implementations are done with CNF of size \(O(\log \log n)\).

First, sort the \(k + 1\) states symbolically:

Sorting can be done with CNF of size \(O(\log n)\).
Recurrence Diameter Test using Sorting Networks

Example CNF size comparison (without transition system):

<table>
<thead>
<tr>
<th>$2^k$</th>
<th>Variables</th>
<th>Clauses</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>64,443</td>
<td>9,535</td>
</tr>
<tr>
<td>128</td>
<td>2,104,723</td>
<td>104,833</td>
</tr>
<tr>
<td>256</td>
<td>11,242,601</td>
<td>356,343</td>
</tr>
<tr>
<td>512</td>
<td>45,687,493</td>
<td>1,177,799</td>
</tr>
</tbody>
</table>

$k = 31$
Future Research Directions

- Combining bounded model checking with other reduction techniques is also a fruitful direction.

  - Techniques for generating short propositional formulas need to be studied.

  - Want to investigate further the use of domain knowledge to guide search in SAT procedures.

  - A practical decision procedure for QBF would also be useful.

  - Combining bounded model checking with other reduction techniques is also a fruitful direction.

  - Techniques for generating short propositional formulas need to be studied.

We believe our techniques may be able to handle much larger designs than is currently possible. Nevertheless, there are a number of directions for future research.