Lecture 2: Symbolic Model Checking With SAT

Edmund M. Clarke, Jr.
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

(Joint work over several years with: A. Biere, A. Cimatti, Y. Zhu, A. Gupta, J. Kukula, D. Kroening, O. Strichman)
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.
- Variable ordering must be uniform along paths.
- Selecting right variable ordering very important for obtaining small BDDs.
  - Often time consuming or needs manual intervention.
  - Sometimes, no space efficient variable ordering exists.

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

- SAT procedures also operate on Boolean expressions but do not use canonical forms.
- Do not suffer from the potential space explosion of BDDs.
- Different split orderings possible on different branches.
- Very efficient implementations available.
Bounded Model Checking
(Clarke, 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 finds counterexamples fast. This is due to depth first nature of SAT search procedures.

- It finds counterexamples of minimal length. This feature helps user understand counterexample more easily.

- It uses much less space than BDD based approaches.

- Does not need manually selected variable order or costly reordering. Default splitting heuristics usually sufficient.

- Bounded model checking of LTL formulas does not require a tableau or automaton construction.
Implementation

- 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 checking.

- 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.
  - Experimental results.
  - Tuning SAT checkers for bounded model checking
  - Efficient computation of diameters
- Abstraction / refinement with SAT
- Directions for future research.
Basic Definitions and Notation

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

- Basic LTL operators:

  - next time ‘X’
  - eventually ‘F’
  - globally ‘G’
  - until ‘U’
  - release ‘R’

- Only consider existential LTL formulas $\mathbf{E} f$, where

  - $\mathbf{E}$ is the existential path quantifier, and

  - $f$ is a temporal formula with no path quantifiers.

- Recall that $\mathbf{E}$ is the dual of the universal path quantifier $\mathbf{A}$.

- Finding a witness for $\mathbf{E} f$ is equivalent to finding a counterexample for $\mathbf{A} \neg f$. 
System described as a Kripke structure $M = (S, I, T, \ell)$, where

- $S$ is a finite set of states,
- $I$ is the set of initial states,
- $T \subseteq S \times S$ is the transition relation, and
- $\ell : S \to \mathcal{P}(\mathcal{A})$ is the state labeling.

We assume every state has a successor state.
In symbolic model checking, a state is represented by a vector of state variables 
\( s = (s(1), \ldots, s(n)) \).

We define propositional formulas \( f_I(s) \), \( f_T(s, t) \) and \( f_p(s) \) as follows:

\begin{itemize}
  \item \( f_I(s) \) iff \( s \in I \),
  \item \( f_T(s, t) \) iff \( (s, t) \in T \), and
  \item \( f_p(s) \) iff \( p \in \ell(s) \).
\end{itemize}

We write \( T(s, t) \) instead of \( f_T(s, t) \), etc.
Definitions and Notation (Cont.)

- Will sometimes write \( s \rightarrow t \) when \( (s, t) \in T \).
- If \( \pi = (s_0, s_1, \ldots) \), then \( \pi(i) = s_i \) and \( \pi^i = (s_i, s_{i+1}, \ldots) \).
- \( \pi \) is a path if \( \pi(i) \rightarrow \pi(i + 1) \) for all \( i \).
- \( \mathbf{E} \phi \) is true in \( M \) (\( M \models \mathbf{E} \phi \)) iff there is a path \( \pi \) in \( M \) with \( \pi \models \phi \) and \( \pi(0) \in I \).
- Model checking is the problem of determining the truth of an LTL formula in a Kripke structure. Equivalently,

Does a witness exist for the LTL formula?
Example To Illustrate New Technique

Two-bit counter with an erroneous transition:

- Each state $s$ is represented by two state variables $s[1]$ and $s[0]$.
- In initial state, value of the counter is 0. Thus, $I(s) = \neg s[1] \land \neg s[0]$.
- Let $inc(s, s') = (s'[0] \leftrightarrow \neg s[0]) \land (s'[1] \leftrightarrow (s[0] \oplus s[1]))$
- Define $T(s, s') = inc(s, s') \lor (s[1] \land \neg s[0] \land s'[1] \land \neg s'[0])$
- Have deliberately added erroneous transition!!
Example (Cont.)

- Suppose we want to know if counter will eventually reach state \((11)\).
- Can specify the property by \(\mathbf{AF}q\), where \(q(s) = s[1] \land s[0]\).
  
  On all execution paths, there is a state where \(q(s)\) holds.
- Equivalently, we can check if there is a path on which counter never reaches state \((11)\).
- This is expressed by \(\mathbf{EG}p\), where \(p(s) = \neg s[1] \lor \neg s[0]\).
  
  There exists a path such that \(p(s)\) holds globally along it.
Example (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 $s_0$, $s_1$, $s_2$.
- We formulate constraints on $s_0$, $s_1$, and $s_2$ in propositional logic.
- Constraints guarantee that $(s_0, s_1, s_2)$ is a witness for $\text{EG}p$ and, hence, a counterexample for $\text{AF}q$. 
First, we constrain \((s_0, s_1, s_2)\) to be a valid path starting from the initial state.

Obtain a propositional formula

\[
\left[ M \right] = I(s_0) \land T(s_0, s_1) \land T(s_1, s_2).
\]
Example (Cont.)

- Second, we constrain the shape of the path.
- The sequence of states $s_0, s_1, s_2$ can be a loop.
- If so, there is a transition from $s_2$ to the initial state $s_0, s_1$ or itself.
- We write $\ell L = T(s_2, s_l)$ to denote the transition from $s_2$ to a state $s_l$ where $l \in [0, 2]$.
- We define $L$ as $\sqrt{\sum_{l=0}^{2} \ell L}$. Thus $\neg L$ denotes the case where no loop exists.
Example (Cont.)

- The temporal property $Gp$ must hold on $(s_0, s_1, s_2)$.

- If no loop exists, $Gp$ does not hold and $\llbracket Gp \rrbracket$ is $false$.

- To be a witness for $Gp$, the path must contain a loop (condition $L$, given previously).

- Finally, $p$ must hold at every state on the path
  \[
  \llbracket Gp \rrbracket = p(s_0) \land p(s_1) \land p(s_2).
  \]

- We combine all the constraints to obtain the propositional formula
  \[
  \llbracket M \rrbracket \land \left( \neg L \land false \right) \lor \bigvee_{l=0}^{2} \left( L \land \llbracket Gp \rrbracket \right).
  \]
In this example, the formula is satisfiable.

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

If self-loop at \((10)\) is removed, then formula is unsatisfiable.
Model Checking: 16x16 bit sequential shift and add multiplier with overflow flag and 16 output bits.
### DME Example

<table>
<thead>
<tr>
<th>cells</th>
<th>SMV₁ sec</th>
<th>SMV₂ sec</th>
<th>SATO k=5 sec</th>
<th>PROVER k=5 sec</th>
<th>SATO k=10 sec</th>
<th>PROVER k=10 sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>846 11 MB</td>
<td>159 217 MB</td>
<td>0 3 sec</td>
<td>1 3 sec</td>
<td>3 6 sec</td>
<td>54 5 sec</td>
</tr>
<tr>
<td>5</td>
<td>2166 15 MB</td>
<td>530 703 MB</td>
<td>0 4 sec</td>
<td>2 3 sec</td>
<td>9 8 sec</td>
<td>95 5 sec</td>
</tr>
<tr>
<td>6</td>
<td>4857 18 MB</td>
<td>1762 703 MB</td>
<td>0 4 sec</td>
<td>3 3 sec</td>
<td>7 9 sec</td>
<td>149 6 sec</td>
</tr>
<tr>
<td>7</td>
<td>9985 24 MB</td>
<td>6563 833 MB</td>
<td>0 5 sec</td>
<td>4 4 sec</td>
<td>15 10 sec</td>
<td>224 8 sec</td>
</tr>
<tr>
<td>8</td>
<td>19595 31 GB</td>
<td>&gt;1GB</td>
<td>1 6 sec</td>
<td>6 5 sec</td>
<td>16 12 sec</td>
<td>323 8 sec</td>
</tr>
<tr>
<td>9</td>
<td>&gt;10h</td>
<td>1 6 sec</td>
<td>9 5 sec</td>
<td>24 13 sec</td>
<td>444 9 sec</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1 7 sec</td>
<td>10 5 sec</td>
<td>36 15 sec</td>
<td>614 10 sec</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>1 8 sec</td>
<td>13 6 sec</td>
<td>38 16 sec</td>
<td>820 11 sec</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>1 9 sec</td>
<td>16 6 sec</td>
<td>40 18 sec</td>
<td>1044 11 sec</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>1 9 sec</td>
<td>19 8 sec</td>
<td>107 19 sec</td>
<td>1317 12 sec</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>1 10 sec</td>
<td>22 8 sec</td>
<td>70 21 sec</td>
<td>1634 14 sec</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>1 11 sec</td>
<td>27 8 sec</td>
<td>168 22 sec</td>
<td>1992 15 sec</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Model Checking: Liveness for one user in the DME.**
## “Buggy” DME Example

<table>
<thead>
<tr>
<th>cells</th>
<th>SMV1 sec</th>
<th>SMV1 MB</th>
<th>SMV2 sec</th>
<th>SMV2 MB</th>
<th>SATO sec</th>
<th>SATO MB</th>
<th>PROVER sec</th>
<th>PROVER MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>799</td>
<td>11</td>
<td>14</td>
<td>44</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>1661</td>
<td>14</td>
<td>24</td>
<td>57</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>3155</td>
<td>21</td>
<td>40</td>
<td>76</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>5622</td>
<td>38</td>
<td>74</td>
<td>137</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>9449</td>
<td>73</td>
<td>118</td>
<td>217</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>9 segmentation fault</td>
<td>172</td>
<td>220</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 segmentation fault</td>
<td>244</td>
<td>702</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>413</td>
<td>702</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>719</td>
<td>702</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>843</td>
<td>702</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>1060</td>
<td>702</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>1429</td>
<td>702</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Model Checking: Counterexample for liveness in a buggy DME implementation.
Tuning SAT checkers for BMC
(O. Strichman, CAV00)

- Use the variable dependency graph for deriving a static variable ordering.

- Use the regular structure of $\text{AG}^p$ formulas to replicate conflict clauses:

  $$\varphi : I_0 \land \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1}) \land \bigvee_{i=0}^{k} p_i$$

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

  This symmetry indicates that under certain conditions, for each conflict clause we can compute additional $k - 1$ clauses ‘for free’.
Tuning SAT checkers for BMC (cont’d)

- Use the incremental nature of BMC to reuse conflict clauses.
  Some of the clauses that were computed while solving BMC with e.g. \( k=10 \) can be reused when solving the subsequent instance with \( k=11 \).

- **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.

- ...

...
## BMC of some hardware designs w/wo tuning SAT

<table>
<thead>
<tr>
<th>Design #</th>
<th>$\kappa$</th>
<th>RB1</th>
<th>RB2</th>
<th>Grasp</th>
<th>Tuned</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>18</td>
<td>7</td>
<td>6</td>
<td>282</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>70</td>
<td>8</td>
<td>1.1</td>
<td>0.8</td>
</tr>
<tr>
<td>3</td>
<td>14</td>
<td>597</td>
<td>375</td>
<td>76</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>24</td>
<td>690</td>
<td>261</td>
<td>510</td>
<td>12</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td>803</td>
<td>184</td>
<td>24</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>22</td>
<td></td>
<td>356</td>
<td></td>
<td>18</td>
</tr>
<tr>
<td>7</td>
<td>9</td>
<td></td>
<td>2671</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>35</td>
<td></td>
<td></td>
<td>6317</td>
<td>20</td>
</tr>
<tr>
<td>9</td>
<td>38</td>
<td></td>
<td></td>
<td>9035</td>
<td>25</td>
</tr>
<tr>
<td>10</td>
<td>31</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>32</td>
<td>152</td>
<td>60</td>
<td></td>
<td>312</td>
</tr>
<tr>
<td>12</td>
<td>31</td>
<td>1419</td>
<td>1126</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>3626</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

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

RB2 - RuleBase second run (without BDD dynamic reordering).
Diameter

- 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.

- Finding $d$ is computationally hard:
  - State $s$ is reachable in $j$ steps:
    \[
    R_j(s) := \exists s_0, \ldots, s_j : s = s_j \land I(s_0) \land \bigwedge_{i=0}^{j-1} T(s_i, s_{i+1})
    \]
  - Thus, $k$ is greater or equal than the diameter $d$ if
    \[
    \forall s : R_{k+1}(s) \implies \exists j \leq k : R_j(s)
    \]
    This requires an efficient QBF checker!
A Compromise: Recurrence Diameter

- Recurrence Diameter $rd$: Least number of steps $n$ such that all valid paths of length $n$ have at least one cycle

Example:
- All states are reachable from $s_0$ in two steps, i.e., $d = 2$
- All paths with at least one cycle have a minimum length of four steps, i.e., $rd = 4$

- Theorem: Recurrence Diameter $rd$ is an upper bound for the Diameter $d$
Testing the Recurrence Diameter

- Recurrence Diameter test in BMC:
  Find cycles by comparing all states with each other

\[ \forall s_0, \ldots, s_k : I(s_0) \land \bigwedge_{i=0}^{n-1} T(s_i, s_{i+1}) \implies \bigvee_{l=0}^{k-1} \bigvee_{j=l+1}^{k} s_l = s_j \]

- Size of CNF: \( O(k^2) \)
- Too expensive for big \( k \)
Recurrence Diameter Test using Sorting Networks (D. Kroening)

- Idea: Look for cycles using a Sorting Network
- First, sort the $k + 1$ states symbolically:

$$s'_0, \ldots, s'_k \text{ are permutation of } s_0, \ldots, s_k \text{ such that } s'_0 \leq s'_1 \leq \ldots \leq s'_k$$

- Sorting can be done with CNF of size $O(k \log k)$. Practical implementations, e.g., Bitonic sort, have size $O(k \log^2 k)$.
- Now only check neighbors in the sorted sequence:

$$(\exists i : s'_i = s'_{i+1}) \iff (\exists l, j : l \neq j \land s_l = s_j)$$
Recurrence Diameter Test using Sorting Networks

- Example CNF size comparison (without transition system):

<table>
<thead>
<tr>
<th>$k$</th>
<th>$O(k^2)$ Alg.</th>
<th>$O(k \log^2 k)$ Alg.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Variables</td>
<td>Clauses</td>
</tr>
<tr>
<td>32</td>
<td>5,777</td>
<td>25,793</td>
</tr>
<tr>
<td>64</td>
<td>22,817</td>
<td>104,833</td>
</tr>
<tr>
<td>128</td>
<td>90,689</td>
<td>422,657</td>
</tr>
<tr>
<td>256</td>
<td>361,601</td>
<td>1,697,281</td>
</tr>
<tr>
<td>512</td>
<td>1,444,097</td>
<td>6,802,433</td>
</tr>
</tbody>
</table>
Future Research Directions

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:

- 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.