(Joint work over several years with: A. Biere, A. Cimatti, X. Zhu, A. Gupta, J. Kukula, D. Kroening, O. Strichman)
Symbolic Model Checking with BDDs

- BDDs traditionally used to represent Boolean functions.
- Can handle much larger designs – hundreds of state variables.
- Uses Boolean encoding for state machine and sets of states.
- Method used by most "industrial strength" model checkers.
Problems with BDDs

BDDs are a canonical representation. Often become too large.

- Often time consuming or needs manual intervention.
- Sometimes, no space efficient variable ordering exists.
- 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 uses a SAT procedure instead of BDDs. We construct Boolean formula that is satisfiable if 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.
We have implemented a tool BMC for our approach. If counterexample exists, a standard SAT solver generates a truth assignment for the formula. Given \( k \), BMC outputs a formula that is satisfiable if counterexample exists of length \( k \). It accepts a subset of the SMV language.
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:
  – Experimental results.

• Directions for future research.
We use linear temporal logic (LTL) for specifications.

Basic Definitions and Notation

We use linear temporal logic (LTL) for specifications.

Basic LTL operators:

- **next time** `\textnext`
- **eventuality** `\texteventually`
- **globally** `\textglobally`
- **release** `\textrelease`
- **until** `\textuntil`
- **ev eventually** `\texteventually`
- **next time** `\textnext`

- **existential** LTL formulas with no path quantifiers:
  - `\exists x . f` is the existential path quantifier, and

- **universal** LTL formulas with path quantifiers:
  - `\forall x . f` is the universal path quantifier.

Recall that `\exists x` is the dual of the universal path quantifier `\forall x`.

Finding a witness for `\exists x . f` is equivalent to finding a counterexample for `\forall x . \neg f`.
System described as a Kripke structure $W = (S, I, \mathcal{L}, \delta)$, where

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

We assume every state has a successor state.
In symbolic model checking, a state is represented by a vector of state variables.

We define propositional formulas as follows: 

\[(s)_f \equiv (s)_f^d \text{ iff } (s)_f^d \quad \text{and} \quad \mathcal{L} \equiv (\tau, s)_f \text{ iff } (\tau, s)_f^d \quad \text{and} \quad I \equiv s \text{ iff } (s)_f^d \]

We write \(\mathcal{L} \equiv \) instead of \(\mathcal{L} \equiv \) \(\tau, s\) \((\tau, s)\_f^d\), etc.

\(12\)
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 Kripke structure.

\[ \forall \exists (0) ( \exists f = \exists \exists \exists \exists \exists \exists \exists \exists ) \text{ is true in } I \text{ for all } i. \]

\[ \text{\exists } f \text{ is true in } I \text{ iff there is a path } \exists \text{ in } \mathcal{W} \text{ with } \exists \mathcal{W} \text{ is a path } \exists \text{ in } \mathcal{W}. \]

\[ \text{\exists } f \text{ is true in } I \text{ iff there is a path } \exists \text{ in } \mathcal{W} \text{ with } \exists \mathcal{W} \text{ is a path } \exists \text{ in } \mathcal{W}. \]

\[ \text{\exists } f \text{ is true in } I \text{ iff there is a path } \exists \text{ in } \mathcal{W} \text{ with } \exists \mathcal{W} \text{ is a path } \exists \text{ in } \mathcal{W}. \]

\[ \text{\exists } f \text{ is true in } I \text{ iff there is a path } \exists \text{ in } \mathcal{W} \text{ with } \exists \mathcal{W} \text{ is a path } \exists \text{ in } \mathcal{W}. \]

\[ \text{\exists } f \text{ is true in } I \text{ iff there is a path } \exists \text{ in } \mathcal{W} \text{ with } \exists \mathcal{W} \text{ is a path } \exists \text{ in } \mathcal{W}. \]

\[ \text{\exists } f \text{ is true in } I \text{ iff there is a path } \exists \text{ in } \mathcal{W} \text{ with } \exists \mathcal{W} \text{ is a path } \exists \text{ in } \mathcal{W}. \]

\[ \text{\exists } f \text{ is true in } I \text{ iff there is a path } \exists \text{ in } \mathcal{W} \text{ with } \exists \mathcal{W} \text{ is a path } \exists \text{ in } \mathcal{W}. \]
Example to Illustrate New Technique

Two-bit counter with an erroneous transition:

\[(0|S) \lor [1|S] \lor [0|S] \lor [1|S]) \land (S, S) = (S, S, J) \]

\[([1|S] \leftrightarrow [1|S]) \lor ([0|S] \leftrightarrow [0|S]) = (S, S, J) \]

In initial state, value of the counter is 0. Thus, \([0|S] \lor [1|S] \lor [0|S] = (S, J)\].

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

Have deliberately added erroneous transition!
There exists a path such that \((s) d\) holds globally along it.

\[
[0]s \land [1]s = (s) d\quad \text{where} \quad \text{EG}\, d
\]

Equivalently, we can check if there is a path on which counter never reaches state \(\cdot\).

On all execution paths, there is a state where \(d\) holds.

\[
[0]s \lor [1]s = (s) d\quad \text{where} \quad \text{AF}\, d
\]

Can specify the property by \(\text{AP}\, d\), where \(d\) holds globally eventually reach state \(\cdot\).

Suppose we want to know if counter will eventually reach state \(\cdot\).

Example (Cont.)
In bounded model checking, we consider paths of length $\infty$. Hence, a counterexample for $AF\varphi$.

Constraints guarantee that $(s_0, s_1, s_2)$ is a witness for $E\varphi$ and, hence, a counterexample for $A\varphi$.

We formulate constraints on $s_0, s_1, s_2$ in propositional logic.

Assume $k$ equals 2. Call the states $s_0, s_1, s_2$.

We start with $k = 0$ and increment $k$ until a witness is found.

In bounded model checking, we consider paths of length $k$. (Cont.)
First, we constrain (s₀, s₁, s₂) to be a valid path starting from the initial state. 

Obtain a propositional formula:

\[
(\{ s₁ \} L \lor \{ s₀ \} L \lor \{ 0s \} I = \llbracket W \rrbracket
\]
Second, we constrain the shape of the path. The sequence of states, $A$, can be a loop. If so, there is a transition from $s_2$ to the initial state $s_0$, $s_1$ or itself. 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 $A$. If no loop exists, $\varphi$ does not hold and $[\varphi]$ is false.

The temporal property $\varphi$ must hold on $(s_0, s_1, s_2)$.

Finally, $d$ must hold at every state on the path previously.

To be a witness for $\varphi$, the path must contain a loop (condition $T$, given $[d \varphi]$).

We combine all the constraints to obtain the propositional formula:

$$
\left( \bigwedge_{z=0}^{0=1} \left( [d \varphi] \lor T^1 \right) \land \left( \varphi \lor T^- \right) \right) \lor [W]
$$

Example (Cont.)
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.

- In this example, the formula is satisfiable.
Model Checking: 16x16 bit sequential shift and add multiplier with overflow flag and 16 output bits.

<table>
<thead>
<tr>
<th>1066</th>
<th>23970</th>
<th>2202</th>
<th>11923</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>183</td>
<td>1</td>
<td>4449</td>
<td>15</td>
</tr>
<tr>
<td>4</td>
<td>917</td>
<td>1</td>
<td>7254</td>
<td>14</td>
</tr>
<tr>
<td>4</td>
<td>186</td>
<td>1</td>
<td>8512</td>
<td>13</td>
</tr>
<tr>
<td>4</td>
<td>916</td>
<td>1</td>
<td>0380</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>121</td>
<td>1</td>
<td>3624</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>916</td>
<td>1</td>
<td>8512</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>889</td>
<td>1</td>
<td>9823</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>69</td>
<td>0</td>
<td>473</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>0</td>
<td>602</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>55</td>
<td>278</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>12</td>
<td>67</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>33</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>8</td>
<td>27</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>8</td>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

| bit | sec | MB  | sec | MB  | sec | MB  | sec | MB  | sec | MB  | sec | MB  | sec | MB  | sec | MB  | sec | MB  | sec | MB  |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| PROVER | SATO | SMV1 | SMV2 | 16x16 | sequential multiplier example |

Sequential Multiplier Example
Model Checking: Liveness for one user in the DME.
Model Checking: Counterexample for liveness in a buggy DME implementation.

<table>
<thead>
<tr>
<th>Cells</th>
<th>SMV1 sec MB</th>
<th>SMV2 sec MB</th>
<th>SATO sec MB</th>
<th>PROVER sec MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>799 14 44 1 0 2</td>
<td>14 24 57 0 1 0 2</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1661 21 40 76 0 1 0 2</td>
<td>24 50 57 0 1 0 0 2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>3155 21 40 76 0 1 0 2</td>
<td>137 76 0 1 0 0 0 2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>5622 38 74 217 0 1 0 0 2</td>
<td>38 74 0 1 0 0 0 2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>9449 73 118 217 0 1 0 0 2</td>
<td>73 118 0 1 0 0 0 2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>172 220 0 1 0 0 1</td>
<td>220 0 1 0 0 1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>224 702 0 1 0 0 2</td>
<td>702 0 1 0 0 2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>11</td>
<td>413 702 0 1 0 0 2</td>
<td>702 0 1 0 0 2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>12</td>
<td>719 702 0 1 0 0 2</td>
<td>702 0 1 0 0 2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>13</td>
<td>843 702 0 1 0 0 2</td>
<td>702 0 1 0 0 2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>14</td>
<td>1060 702 0 1 0 0 2</td>
<td>702 0 1 0 0 2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>15</td>
<td>1429 702 0 1 0 0 2</td>
<td>702 0 1 0 0 2</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
can compute additional $k$ - 1 clauses "for free'.

This symmetry indicates that under certain conditions, for each conflict clause we can compute additional $k$ - 1 clauses "for free'.

Thus, for each conflict clause, we can compute additional clauses without increasing the number of variables in the SAT instance.

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

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

\[ d \wedge \forall \gamma (s^1 \wedge \gamma) \vee \forall \gamma (s^2 \wedge \gamma) \]

Use the regular structure of $\text{AG}$ formulas to replicate conflict clauses.

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

(0. Strichman, CAV00)

Tuning SAT checkers for BMC
UsetheincrementalnatureofBMCtoreuseconflictclauses. SomeoftheclausescomputedwhilesolvingBMCwith\(k=10\)canbereusedwhensolvingthesubsequentinstancewith\(k=11\).

Restrictdecisionstomodelvariablesonly(ignorcNFAuxiliaryvars).

Itispossibletot decidetheroutingformulawithouttheauxiliaryvariables(theywillbereimplied). Inmanysamples,theyare80%-90%ofthevariablesinanCNPinstance.

UsetheincrementalnatureofBMCToreuseconflictcles.

TuningSATcheckersforBMC(cont'd)
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># of Rules</th>
<th>Design Tuned</th>
<th>Grasp</th>
<th>RB1</th>
<th>RB2</th>
</tr>
</thead>
<tbody>
<tr>
<td>312</td>
<td>9035</td>
<td>9035</td>
<td>701</td>
<td>64</td>
<td>6</td>
</tr>
<tr>
<td>25</td>
<td>6317</td>
<td>2671</td>
<td>6</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>20</td>
<td>2 5</td>
<td>356</td>
<td>22</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>24 24</td>
<td>184 803</td>
<td>12</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>12</td>
<td>12 510</td>
<td>269 690</td>
<td>14</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>3 76 375</td>
<td>97 97</td>
<td>12</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>0.8 1.1 6 282</td>
<td>70 70 18</td>
<td>18</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

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

\[(s) \forall \exists \mathcal{L} \leftrightarrow (s) \exists^+ \forall \mathcal{L}\]

Thus, \( \mathcal{L} \) is greater or equal than the diameter \( p \) if

\[\forall 1 \leq 0 \leq \exists \mathcal{L} \]

State \( s \) is reachable in \( \mathcal{L} \) steps:

Finding \( p \) is computationally hard.

Diameter: Least number of steps to reach all reachable states. If the property holds for \( p \) and the property holds for all reachable states, then the diameter is greater or equal than the property. If the property does not hold for \( p \), then the diameter is less than the property.
Theorem: Recurrence Diameter $r_d$ is an upper bound for the Diameter $d$.

- Steps, i.e., $r = \forall$
- Have a minimum length of four
- All paths with at least one cycle
- In two steps, i.e., $\exists$
- All states are reachable from $s_0$

Example:

Reccurrence Diameter $r_d$: Least number of steps $n$ such that all valid paths of length $n$ have at least one cycle

A Compromise: Recurrence Diameter
Testing the Recurrence Diameter

Find cycles by comparing all states with each other.

Recurrent Diameter test in BMC:

Size of CNF: $O(2^\gamma)$

Too expensive for big $\gamma$.
Idea: Look for cycles using a Sorting Network

First, sort the $k + 1$ states symbolically:

$s'_0, \ldots, s'_k$ are permutation of $s_0, \ldots, s_k$ 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:
### Example CNF Size Comparison (without Transition System):

<table>
<thead>
<tr>
<th>Variables</th>
<th>Clauses</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>5,777</td>
</tr>
<tr>
<td>64</td>
<td>22,817</td>
</tr>
<tr>
<td>128</td>
<td>90,689</td>
</tr>
<tr>
<td>256</td>
<td>361,601</td>
</tr>
<tr>
<td>512</td>
<td>1,444,097</td>
</tr>
</tbody>
</table>

Recurrence Diameter Test using Sorting Networks
Future Research Directions

- 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.
- A practical decision procedure for QBF would also be useful.
- Combining bounded model checking with other reduction techniques is also a fruitful direction.

Nevertheless, there are a number of directions for future research:

We believe our techniques may be able to handle much larger designs than is currently possible.