Bounded Model Checking

Presented by
Weifeng Jiang
Outline

- Introduction
- Semantics
- Translation
- Determining the bound
- Experimental Results
- Conclusion
- NuSMV
1. Introduction
Model Checking (MC)

- Given a:
  - Finite transition system $M(S, I, T)$
  - A temporal property $\varphi$
- The model checking problem:
  - Does $M$ satisfy $\varphi$ ?

$M \models \varphi$
Temporal properties

○ “Safety” properties
  ● “Always x=y”
    \( (G(x=y)) \)
  ● “Every Send is followed by Ack”
    \( (G(Send \rightarrow F \text{ Ack})) \)

○ “Liveness” properties
  ● “Reset can always be reached”
    \( (GF \text{ Reset}) \)
  ● “From some point on, always switch_on”
    \( (FG \text{ switch_on}) \)
What is SAT?

- Given a propositional Formula in CNF, find an assignment to Boolean variables that makes the formula true.
- Example:

  $\omega_1 = (x_2 \lor x_3)$
  $\omega_2 = (\neg x_1 \lor \neg x_4)$
  $\omega_3 = (\neg x_2 \lor x_4)$
  $A = \{x_1=0, x_2=1, x_3=0, x_4=1\}$

- SAT solver: tool that finds a satisfying assignment
Bounded Model Checking (BMC)

- Based on SAT

- ∃ Counterexample of length $k$ $\iff$ Propositional Formula is satisifiable

- BMC for LTL reduced to SAT in poly time

- Example:
Most safety properties can be reduced to “Always $p$” where $p$ is propositional.

Is there a state reachable within $k$ cycles that satisfies $\neg p$?
Example (Continued)

$p$ is preserved up to cycle $k$ iff $\Omega(k)$ is unsatisfiable:

\[
\Omega(k) : \quad I(s_0) \land \bigwedge_{i=1}^{k-1} \rho(s_i, s_{i+1}) \land \bigvee_{i=0}^{k} \neg p_i
\]

$p$ is preserved up to cycle $k$ iff $\Omega(k)$ is unsatisfiable:

\[
\Omega(k) : \quad I(s_0) \land \bigwedge_{i=1}^{k-1} \rho(s_i, s_{i+1}) \land \bigvee_{i=0}^{k} \neg p_i
\]
Example: a two bit counter

Initial state: $I_0: \neg l \land \neg r$

Transition: $\rho: l' = (l \neq r)$
$r' = \neg r$

Property: Always $(\neg l \lor \neg r)$.

$\Omega(2): (\neg l_0 \land \neg r_0) \land l_1 = (l_0 \neq r_0) \land r_1 = \neg r_0 \land (l_0 \land r_0) \lor (l_1 \land r_1) \lor (l_2 \land r_2)$

$\Omega(2)$ is unsatisfiable. $\Omega(3)$ is satisfiable.
What can BMC do?

- A.I. Planning problems:
  *we can reach a desired state in k steps*

- Verification of safety properties:
  *we can find a bad state in k steps*

- Verification:
  *we can find a counterexample in k steps*
Relationship to MC

- **BMC – Advantages**
  - CounterExamples – found fast, minimal length
  - Less space, No manual ordering (vs BDD)
  - The best SAT solvers are capable of handling thousands of state variables

- **BMC – Disadvantages**
  - with the limit \( k \), completeness is naturally sacrificed
2. Semantics
Definition 1: A Kripke structure is a tuple $M = (S, I, T, L)$ with a finite set of states $S$, the set of initial states $I \subseteq S$, a transition relation between states $T \subseteq S \times S$ and the labeling of the states $L: S \rightarrow P(A)$ with atomic propositions $A$.

Each state has a successor state.

$\pi = (s_0, s_1, \ldots) \pi(i) = s_i$ and $\pi^i = (s_i, s_{i+1}, \ldots)$
Semantics

- **Definition 2 (Semantics)**: Let $M$ be a Kripke structure, $\pi$ be a path in $M$ and $f$ be an LTL formula. Then $\pi \models f$ (f is valid along $\pi$) is defined as:

  $\pi \models p$ iff $p \in \mathcal{L}(\pi(0))$
  $\pi \models -p$ iff $p \notin \mathcal{L}(\pi(0))$
  $\pi \models f \land g$ iff $\pi \models f$ and $\pi \models g$
  $\pi \models f \lor g$ iff $\pi \models f$ or $\pi \models g$
  $\pi \models G f$ iff $\forall i. \pi^i \models f$
  $\pi \models F f$ iff $\exists i. \pi^i \models f$
  $\pi \models X f$ iff $\pi^1 \models f$
  $\pi \models f \mathcal{U} g$ iff $\exists i [\pi^i \models g$ and $\forall j, j < i. \pi^j \models f]$
  $\pi \models f \mathcal{R} g$ iff $\forall i [\pi^i \models g$ or $\exists j, j < i. \pi^j \models f]$
Semantics - Validity

Definition 3:

- An LTL formula is **universally valid** in a Kripke structure $M$ (in symbols $M \models Af$) iff $\pi \models f$ for all paths $\pi$ in $M$ with $\pi(0) \in I$.
- An LTL formula $f$ is **existentially valid** in a Kripke structure $M$ (in symbols $M \models Ef$) iff there exists a path $\pi$ in $M$ with $\pi \models f$ and $\pi(0) \in I$.
- Here we consider **existential** model checking problem.
Semantics - Basic Idea of BMC

- Consider only a **finite prefix** of a path (bounded by $k$) and look for possible counterexample.
- Finite Prefix may represent an infinite path if there is a **back loop** from the last state of the prefix to any of the previous states.
- If no back loop, can’t say anything about infinite behavior.
- Example: $Gp$ – Even if $p$ holds from $s_0$ to $s_k$, can’t conclude anything if there is no back loop from $s_k$ to $s_0$. 
Semantics

- **Definition 4**: For $l \leq k$ we call a path $\pi$ a \((k,l)\)-loop if $\pi(k) \rightarrow \pi(l)$ and $\pi = u \cdot v^\omega$ with $u = (\pi(0), \ldots, \pi(l-1))$ and $v = (\pi(l), \ldots, \pi(k))$. We call $\pi$ simply a $k$-loop if $0 \leq l \leq k$ for which $\pi$ is a $(k,l)$-loop.
Definition 5 (Bounded Semantics for a Loop):
Let $k \geq 0$ and $\pi$ be a $k$-loop. Then an LTL formula is valid along the path $\pi$ with bound $k$ (in symbols $\pi \models_k f$) iff $\pi \models f$.

Definition 6 (Bounded Semantics without a Loop): Let $k \geq 0$ and $\pi$ be a path that is not a $k$-loop. Then an LTL formula is valid along the path $\pi$ with bound $k$ (in symbols $\pi \models_k f$) iff $\pi \models_k^0 f$ where:
Semantics

\[ \pi \models^i_k p \quad \text{iff} \quad p \in L(\pi(i)) \]
\[ \pi \models^i_k \neg p \quad \text{iff} \quad p \notin L(\pi(i)) \]
\[ \pi \models^i_k f \land g \quad \text{iff} \quad \pi \models^i_k f \text{ and } \pi \models^i_k g \]
\[ \pi \models^i_k f \lor g \quad \text{iff} \quad \pi \models^i_k f \text{ or } \pi \models^i_k g \]
\[ \pi \models^i_k \text{GF}f \quad \text{is always false} \]
\[ \pi \models^i_k \text{GF}f \quad \text{iff} \quad \exists j, \ i \leq j \leq k. \ \pi \models^j_k f \]
\[ \pi \models^i_k \text{X}f \quad \text{iff} \quad i < k \text{ and } \pi \models^{i+1}_k f \]
\[ \pi \models^i_k f \land g \quad \text{iff} \quad \exists j, \ i \leq j \leq k. \ \pi \models^j_k g \text{ and } \forall n, i \leq n < j. \ \pi \models^n_k f \]
\[ \pi \models^i_k f \land g \quad \text{iff} \quad \exists j, \ i \leq j \leq k. \ \pi \models^j_k f \text{ and } \forall n, i \leq n < j. \ \pi \models^n_k g \]
Semantics

\[ M \vDash Ef \rightarrow M \vDash_{k}Ef \] (reduction)

- **Lemma 7**: Let \( f \) be an LTL formula and \( \pi \) be a path and
  \[ \pi \vDash_{k}f \rightarrow \pi \vDash f \]

- **Lemma 8**: Let \( f \) be an LTL formula and \( M \) a Kripke structure.
  If \( M \vDash Ef \) then there exists \( k \) with \( M \vDash_{k}Ef \)

- **Theorem 9**: Let \( f \) be an LTL formula, \( M \) a Kripke structure.
  Then \( M \vDash Ef \) iff there exists \( k \geq 0 \) with \( M \vDash_{k}Ef \)
3. Translation
Translation

- Given a Kripke structure $M$, LTL formula $f$, bound $k$:
  We need to construct a **Propositional Formula** $[[ M,f ]]_k$ which represents the constraints on $s_0,\ldots,s_k$ (variables denoting a finite sequence of states on a path $\pi$) such that $[[ M,f ]]_k$ is satisfiable iff $f$ is valid along $\pi$.

- **Definition 10 (Unfolding the Transition Relation)**
  For a Kripke structure $M$, $k \geq 0$,
  $$[[ M ]]_k = I(s_0) \land \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1})$$
Translation

- Depending on whether a path is a k-loop or not, two different translations for temporal formula $f$.
  - Translation if path not a k-loop:
    \[
    [[ \cdot ]]^i_k
    \]
  - Translation if path is a k-loop:
    \[
    i[[ \cdot ]]^i_k
    \]
Translation

○ **Definition 11 (Translation of an LTL formula without a Loop):** For an LTL formula $f$ and $k$, with $0 \leq i \leq k$

\[
\begin{align*}
\llbracket p \rrbracket_k^i &= p(s_i) & \llbracket \neg p \rrbracket_k^i &= \neg p(s_i) \\
\llbracket f \land g \rrbracket_k^i &= \llbracket f \rrbracket_k^i \land \llbracket g \rrbracket_k^i & \llbracket f \lor g \rrbracket_k^i &= \llbracket f \rrbracket_k^i \lor \llbracket g \rrbracket_k^i \\
\llbracket Gf \rrbracket_k^i &= \text{false} & \llbracket Ff \rrbracket_k^i &= \lor_{j=i}^k \llbracket f \rrbracket_k^j \\
\llbracket Xf \rrbracket_k^i &= \text{if } i < k \text{ then } \llbracket f \rrbracket_k^{i+1} \text{ else false} \\
\llbracket f U g \rrbracket_k^i &= \lor_{j=i}^k \left( \llbracket g \rrbracket_k^i \land \land_{i=1}^{j-1} \llbracket f \rrbracket_k^n \right) \\
\llbracket f R g \rrbracket_k^i &= \lor_{j=i}^k \left( \llbracket f \rrbracket_k^i \land \land_{i=1}^j \llbracket g \rrbracket_k^n \right)
\end{align*}
\]

○ **Definition 12 (Successor in a Loop):** Let $k,l,i$ non-negative integers, with $l,i \leq k$. Define the successor $\text{succ}(i)$ in a $(k,l)$-loop as $\text{succ}(i) = i+1$ for $i < k$ and $\text{succ}(i) = l$ for $i = k$
Translation

Definition 13 (Translation of an LTL formula for a Loop): Let f be an LTL formula, $k,l,i \geq 0$ with $l,i \leq k$

- $\llbracket p \llbracket^i_k := p(s_i)$
- $\llbracket \neg p \llbracket^i_k := \neg p(s_i)$
- $\llbracket f \land g \llbracket^i_k := \llbracket f \llbracket^i_k \land \llbracket g \llbracket^i_k$
- $\llbracket f \lor g \llbracket^i_k := \llbracket f \llbracket^i_k \lor \llbracket g \llbracket^i_k$
- $\llbracket \Box f \llbracket^i_k := \Box_{j=\min(l,i)} \llbracket f \llbracket^j_k$
- $\llbracket \Diamond f \llbracket^i_k := \Diamond_{j=\min(l,i)} \llbracket f \llbracket^j_k$
- $\llbracket X f \llbracket^i_k := \llbracket f \llbracket^i_k^{\text{suc}(j)}$
- $\llbracket f \land g \llbracket^i_k := \forall_{j=l}^{j=i} \left( \llbracket g \llbracket^j_k \land \bigwedge_{n=l}^{n=j-1} \llbracket f \llbracket_k^n \right) \lor \forall_{j=l}^{j=i} \left( \llbracket g \llbracket^j_k \land \bigwedge_{n=l}^{n=j-1} \llbracket f \llbracket_k^n \land \bigwedge_{n=l}^{n=j-1} \llbracket g \llbracket_k^n \right)$
- $\llbracket f \land g \llbracket^i_k := \forall_{j=l}^{j=i} \left( \llbracket f \llbracket^j_k \land \bigwedge_{n=l}^{n=j-1} \llbracket g \llbracket_k^n \right) \lor \forall_{j=l}^{j=i} \left( \llbracket f \llbracket^j_k \land \bigwedge_{n=l}^{n=j-1} \llbracket g \llbracket_k^n \land \bigwedge_{n=l}^{n=j-1} \llbracket g \llbracket_k^n \right)$
Translation

- **Definition 14 (Loop Condition)**: For \( k, l \geq 0 \), let 
  \[ L_k = \bigvee_{l=0}^{k} L_k \]

- **Definition 15 (General Translation)**: Let \( f \) be an LTL formula, \( M \) a Kripke structure and \( k \geq 0 \)

\[
[[ M, f ]]_k := [[ M ]]_k \land \left( \left( \neg L_k \land [[ f ]]_k^0 \right) \lor \bigvee_{l=0}^{k} \left( L_k \land [[ f ]]_k^0 \right) \right)
\]

- **Theorem 16**: \( [[ M, f ]]_k \) is satisfiable iff \( M \models_k E f \)

- **Corollary 17**: \( M \models A \neg f \) iff \( [[ M, f ]]_k \) is unsatisfiable for all \( k \geq 0 \)
4. Determining The Bound
Determining the Bound

- For every model $M$ and LTL property $\varphi$ there exists $k$ s.t.

$$M \models_k \varphi \rightarrow M \models \varphi$$

- The minimal such $k$ is the Completeness Threshold (CT)
Determining the Bound

- **Diameter** $d = \text{longest ‘shortest path’ from an initial state to any other reachable state.}
- **Recurrence Diameter** $rd = \text{longest loop-free path.}
- $rd \geq d$

$d = 2$

$rd = 3$
Determining the Bound

- **Theorem**: for $Gp$ properties $CT = d$

![Diagram](image-url)

- $s_0$  
  - Arbitrary path  
  - $\neg p$
Determining the Bound

- **Theorem**: for $Fp$ properties $CT = rd$

- Open Problem: The value of $CT$ for general Linear Temporal Logic properties is unknown
5. Experimental Results
Experimental Results

- **BMC – Implement steps**
  - Input language is a subset of the SMV language
  - It takes in a circuit description, a property to be proven, and a user supplied time bound $k$.
  - It then generates the propositional formula.
  - Propositional formula can be solved using SAT solver.
Experimental Results

Benchmark example:

The Intel(R) benchmark: verifying various circuit designs with an in-house BDD model checker (FORECAST) and an in-house SAT solver (THUNDER). Results are given in seconds.
### Experimental Results

<table>
<thead>
<tr>
<th>Model</th>
<th>$k$</th>
<th>FORECAST (BDD)</th>
<th>THUNDER (SAT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Circuit 1</td>
<td>5</td>
<td>114</td>
<td>2.4</td>
</tr>
<tr>
<td>Circuit 2</td>
<td>7</td>
<td>2</td>
<td>0.8</td>
</tr>
<tr>
<td>Circuit 3</td>
<td>7</td>
<td>106</td>
<td>2</td>
</tr>
<tr>
<td>Circuit 4</td>
<td>11</td>
<td>6189</td>
<td>1.9</td>
</tr>
<tr>
<td>Circuit 5</td>
<td>11</td>
<td>4196</td>
<td>10</td>
</tr>
<tr>
<td>Circuit 6</td>
<td>10</td>
<td>2354</td>
<td>5.5</td>
</tr>
<tr>
<td>Circuit 7</td>
<td>20</td>
<td>2795</td>
<td>236</td>
</tr>
<tr>
<td>Circuit 8</td>
<td>28</td>
<td>*</td>
<td>45.6</td>
</tr>
<tr>
<td>Circuit 9</td>
<td>28</td>
<td>*</td>
<td>39.9</td>
</tr>
<tr>
<td>Circuit 10</td>
<td>8</td>
<td>2487</td>
<td>5</td>
</tr>
<tr>
<td>Circuit 11</td>
<td>8</td>
<td>2940</td>
<td>5</td>
</tr>
<tr>
<td>Circuit 12</td>
<td>10</td>
<td>5524</td>
<td>378</td>
</tr>
<tr>
<td>Circuit 13</td>
<td>37</td>
<td>*</td>
<td>195.1</td>
</tr>
<tr>
<td>Circuit 14</td>
<td>41</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Circuit 15</td>
<td>12</td>
<td>*</td>
<td>1070</td>
</tr>
<tr>
<td>Circuit 16</td>
<td>40</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Circuit 17</td>
<td>60</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>
6. Conclusion
Conclusion

- BMC is the first step in applying SAT procedures to symbolic model checking.

- Many models that cannot be solved by BDD symbolic model checkers, can be solved with the optimized SAT Bounded Model Checker.

- Today: BMC with SAT is dominant in finding shallow errors. BDD-based procedures are mainly used for proving their absence.
Recent Work

- New techniques developed to determine the diameter of a system

- A typical methodology applied in the industry today is to use both BMC and BDD based model checkers as complementary methods
7. NuSMV
NuSMV

- **NuSMV** is a software tool for the formal verification of finite state systems. It has been developed jointly by **ITC-IRST** and by **Carnegie Mellon University (CMU)**.

- Version 1 of NUSMV basically implements BDD-based symbolic model checking.

- Version 2 integrated model checking techniques based on propositional satisfiability (**SAT**).

- The newest version is **NuSMV 2.3.1**, We can get it from: [http://nusmv.irst.itc.it](http://nusmv.irst.itc.it)
The basic purpose of the NuSMV language is to describe (using expressions in propositional calculus) the transition relation of a finite Kripke structure.

Since NuSMV is intended to describe finite state machines, the only data types in the language are finite ones, i.e. boolean, scalar and fixed arrays of basic data types.
NuSMV

Functionalities:

- **NUSMV can process files written in SMV**

- **NUSMV can work in batch mode**: processing an input file according to the specified command line options

- **NUSMV has an interactive mode**: it enters a shell performing a read-eval-print loop, and the user can activate the various computation steps (e.g. parsing, model construction, reachability analysis, model checking)