CDA 5416 Computer System Verification
Bounded Model Checking

Hao Zheng

Department of Computer Science and Engineering
University of South Florida
Model Checking is used for exhaustive verification.
- Difficult to scale (state explosion).

OBDDs are a canonical representation.
- Canonicity makes equivalence checking easier.
- A variable ordering is required.

Variable ordering is also a serious restriction.
- Finding an optimal ordering is time consuming.
- No good orderings exist for certain applications.
Bounded Model Checking

- Targeted to find bugs, not to achieve the complete correctness proof.
- Finds bugs in a bounded number of executions.
- Can discover shallow bugs quickly.
  + Always finds the shortest counter-examples.
- Based on the latest advances in Boolean satisfiability (SAT/SMT) solving.
- High memory demand is alleviated, but runtime may be a serious problem.
• Boolean satisfiability answers whether a variable assignment exists to make a Boolean formula be true.
  • A classic NP-complete problem.

• Boolean SAT solving has become very efficient in practice.
  • Can readily handle formulas with tens of thousands of variables.
  • Much more space efficient than OBDDs.

• Many model checking problems can be converted to SAT solving.

• SAT-based BMC
  • Encodes all paths in a TS upto a bound $k$ into a Boolean formula.
  • Encodes negation of properties along the $k$–path formula.
  • Searches counter-examples by using SAT solving on the formula.
BMC: An Illustrating Example

- Check if the circuit satisfies $\forall \square \neg q$.

Initial state: $x=0, y=0$.

$q = (w \oplus y \lor x) \land \neg(x \lor w)$
Circuit Initial State

\[ q^0 = (w^0 \oplus y^0 \lor x^0) \land \neg(x^0 \lor w^0) = 0 \]
Circuit State after Cycle 1

- $q^1 = 1$ if $w^0 = 1$ in the initial state and $w^1 = 0$ in cycle 1.
- A counter-example to $\forall \square \neg q$ is a 2-state sequence.
Big Picture of Bounded Model Checking

Comb. Logic

\[ I \rightarrow S \rightarrow S' \rightarrow O \]

Comb. Logic
Comb. Logic
Comb. Logic

\[ I^0 \rightarrow S^0 \rightarrow S^1 \rightarrow O^0 \]
\[ I^1 \rightarrow S^1 \rightarrow S^2 \rightarrow O^1 \]
\[ I^2 \rightarrow S^2 \rightarrow S^3 \rightarrow O^2 \]

H. Zheng (CSE USF)
How BMC Works

```
k = 0

BMC(M, ¬f, k)

Stop
Yes
k ≥ CT
fo

Resource exhausted
SAT
fail

k++
UNSAT

k
```
Given a $M = (I, \Delta)$, an LTL formula $f$ and a bound $k$, BMC generates a Boolean formula $[M, \neg f]_k$ such that

$$[M, \neg f]_k \text{ is satisfiable } \iff \text{ A count-example of length } k \text{ exists}$$

- $[M]_k$: all $k$–paths in $M(I, \Delta)$.  

$$[M]_k = I(\vec{x}_0) \land \Delta(\vec{x}_0, \vec{x}_1) \land \ldots \land \Delta(\vec{x}_{k-1}, \vec{x}_k) \land \Delta(\vec{x}_k, \vec{x}_l)$$

- Encoding of $\neg f$ over $[M]_k$.
  - $[\neg f]_k$: encoding of $\neg f$ on $k$–paths.
  - $l[\neg f]_k$: encoding of $\neg f$ on $k$–loops.
A $k$−bounded path is a sequence of $k$ state transitions.

$$[M]_k = I(x_0) \land \Delta(x_0, x_1) \land \ldots \land \Delta(x_{k-1}, x_k)$$

\[\text{step 1} \quad \text{step } k\]
**$k$—Bounded Loops**

- A finite path is infinite if it has a back loop.

- A $(k, l)$—loop is a $k$—bounded path $\rho$ such that $R(s_k, s_l)$ holds.

\[
[M]_k = I(\vec{x}_0) \land \Delta(\vec{x}_0, \vec{x}_1) \land \ldots \land \Delta(\vec{x}_{k-1}, \vec{x}_k) \land \Delta(\vec{x}_k, \vec{x}_l)
\]

- A path $\rho$ is a $k$—loop if there exists $0 \leq l \leq k$ such that $\rho$ is a $(k, l)$—loop.

\[
[M]_k = I(\vec{x}_0) \land \Delta(\vec{x}_0, \vec{x}_1) \land \ldots \land \Delta(\vec{x}_{k-1}, \vec{x}_k) \land \forall 0 \leq l \leq k, \Delta(\vec{x}_k, \vec{x}_l)
\]
Bounded Semantics of LTL Formulas

• Let $\rho \models^k f$ denote the truth of the LTL formula $f$ over the $k$—bounded path $\rho$.
  • Evaluate $f$ only in the first $k + 1$ states on $\rho$.
• Let $\rho(i)$ denote the $i^{th}$ state on $\rho$.
• Let $\rho \models^i_k f$ denote the truth of $f$ over the path from state $\rho(i)$ to $\rho(k)$.
• If a path $\rho$ is a $k$—loop,

$$\rho \models^k f \iff \rho \models f$$
Bounded Semantics of LTL Formulas (2)

- \( \rho \models_k f \iff \rho \models^0_k f \) where

\[
\begin{align*}
\rho \models^i_k p & \iff p \in L(\rho(i)) \\
\rho \models^i_k \neg p & \iff p \notin L(\rho(i)) \\
\rho \models^i_k f \land g & \iff \rho \models^i_k f \text{ and } \rho \models^i_k g \\
\rho \models^i_k f \lor g & \iff \rho \models^i_k f \text{ or } \rho \models^i_k g \\
\rho \models^i_k \Box f & \iff \text{false} \\
\rho \models^i_k \Diamond f & \iff \exists i \leq j \leq k, \ \rho \models^j_k f \\
\rho \models^i_k \circ f & \iff i < k \text{ and } \rho \models^{i+1}_k f \\
\rho \models^i_k f \lor g & \iff \exists i \leq j \leq k, \ \rho \models^j_k f \text{ and } \forall i \leq n \leq j. \rho \models^n_k f
\end{align*}
\]

where \( p \) is an atomic proposition.
Bounded Model Checking of LTL

• Let $M \models^k f$ denote a $k$–bounded model checking problem for the LTL formula $f$.
  • Formula $f$ is evaluated on all $k$–bounded path.

• Let $f$ be a LTL formula and $\rho$ a path.

$$\rho \models^k \neg f \implies \rho \models \neg f$$

• If there is a $\rho$ in $M$ such that $\rho \models^k \neg f$, then $M \models f$ does not hold.
  Search for $k$-bounded counter-example.

• $M \models f \iff \exists k \geq 0, M \models^k f$.
  • There always exists a $k$ such that the result of bounded model checking is equivalent to that of the complete one.
  • Finding the completeness threshold is difficult.
An BMC Example: Translation

- $M \models \square \neg (a \land b)$ for $k = 2$.
- $M = (I, \Delta)$ where
  
  
  $I = \neg a \land \neg b$

  $\Delta = (\neg a \land \neg b \land a' \land \neg b') \lor (\neg a \land \neg b \land \neg a' \land b') \lor (\neg a \land b \land \neg a' \land \neg b') \lor (a \land \neg b \land \neg a' \land \neg b') \lor (a \land \neg b \land a' \land b') \lor (a \land b \land \neg a' \land \neg b')$

\[ \begin{array}{ccc}
\neg ab & \neg a \neg b & a \neg b \\
\neg ab & \neg a \neg b & a \neg b \\
\neg ab & \neg a \neg b & a \neg b \\
\neg ab & \neg a \neg b & a \neg b \\
\end{array} \]
- $M \models \Box \neg(a \land b)$.

- BMC checks if there is a bounded path on which $\Diamond (a \land b)$ holds.

Check if $I(a_0, b_0) \land (a_0 \land b_0)$ is satisfiable?
• $M \models_{k=1} \Box \neg (a \land b)$.

• Check if the following formula is satisfiable?

\[ I(a_0, b_0) \land \Delta(a_0, b_0, a_1, b_1) \land (a_1 \land b_1) \]
• $M \models_{k=2} \Box \neg(a \land b)$.

• Check if the following formula is satisfiable?

$$I(a_0, b_0) \land \Delta(a_0, b_0, a_1, b_1) \land \Delta(a_1, b_1, a_2, b_2) \land (a_2 \land b_2)$$
Bounded Model Checking: Overview

\[ k = 0 \]

\[ \text{BMC}(M, \neg f, k) \]

\[ k \geq CT \]

Stop

Resource exhausted

SAT

fail

UNSAT

k++

Yes

pass
Generalization of BMC

- Key idea of BMC: impose bounds on aspects of system behavior.
- Two generalizations:
  - Bounded model checking of sequential software
  - Context bounded model checking of concurrent software
CBMC is a bounded model checker for ANSI-C programs.

- Handles function calls using inlining.
- Unwinds the loops a fixed number of times.
- Allows user input to be modeled using non-determinism.
  - So that a program can be checked for a set of inputs rather than a single input
- Allows specification of assertions which are checked using the bounded model checking
- It targets sequential programs
Loops and Recursive Functional Calls

- Unwind the loop $n$ times by duplicating the loop body $n$ times
  - Each copy is guarded using an if statement that checks the loop condition.
- At the end of the $n$ repetitions an unwinding assertion is added which is the negation of the loop condition
  - Hence if the loop iterates more than $n$ times in some execution, the unwinding assertion will be violated and we know that we need to increase the bound in order to guarantee correctness
- A similar strategy is used for recursive function calls.
  - The recursion is unwound up to a certain bound and then an assertion is generated stating that the recursion does not go any deeper.
A Simple Loop Example

x = 0;
while (x < 2) {
    y = y+x;
    x++;
}

x=0;
if (x < 2) {
    y=y+x;
    x++;}
if (x < 2) {
    y=y+x;
    x++;}
if (x < 2) {
    y=y+x;
    x++;}
assert(x >= 2);
After eliminating loops and recursion, CBMC converts the input program to the static single assignment (SSA) form:

- In SSA each variable appears at the left hand side of an assignment only once.
- This is a standard program transformation that is performed by creating new variables.

In the resulting program each variable is assigned a value only once and all the branches are forward branches (there is no backward edge in the control flow graph).

CBMC generates a Boolean logic formula from the program using bit vectors to represent variables.
Encoding: A Simple Example

Original Code

```c
x = x + y;
if (x != 1) {
    x = 2;
} else
    x++;  
Assert (x <= 3);
```

Code in SSA format

```c
x1 = x0 + y0;
if (x1 != 1) {
    x2 = 2;;
} else
    x3 = x1 + 1;
x4 = (x1 != 1) ? x2 : x3
assert (x4 <= 3);
```

- Generated Constraints:

  \[\text{Program C} \quad x_1 = x_0 + y_0 \land (x_1 \neq 1 \rightarrow x_2 = 2) \land (x_1 = 1 \rightarrow x_3 = x_1 + 1) \land (x_1 \neq 1 \land x_4 = x_2 \lor x_1 = 1 \land x_4 = x_3)\]

  \[\text{Assertion P} \quad x_4 \leq 3\]
Encoding: A Simple Example

Original Code

```c
x = x + y;
if (x != 1) {
    x = 2;
} else
    x++;
Assert (x <= 3);
```

Code in SSA format

```c
x1 = x0 + y0;
if (x1 != 1) {
    x2 = 2;;
} else
    x3 = x1 + 1;
x4 = (x1 != 1) ? x2 : x3
assert (x4 <= 3);
```

- BMC checks $C \land \neg P$. Assertion $P$ is violated if $C \land \neg P$ is satisfiable.
BMC of Multi-Threaded Programs

- First, convert a MT program into an equivalent sequential program.
- Next, apply encoding previous techniques to generate a BMC problem.
- Complexity is much higher.