# CDA 5416 Computer System Verification

## Bounded Model Checking

Hao Zheng

Department of Computer Science and Engineering
University of South Florida

# Introduction

- 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.

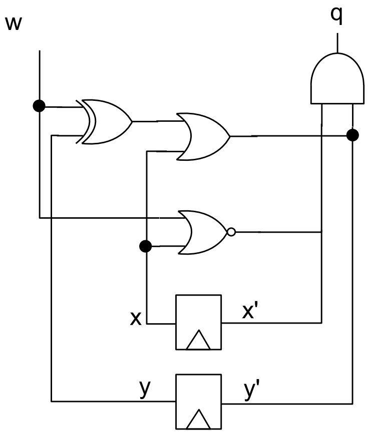
# SAT Solving and Model Checking

- 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

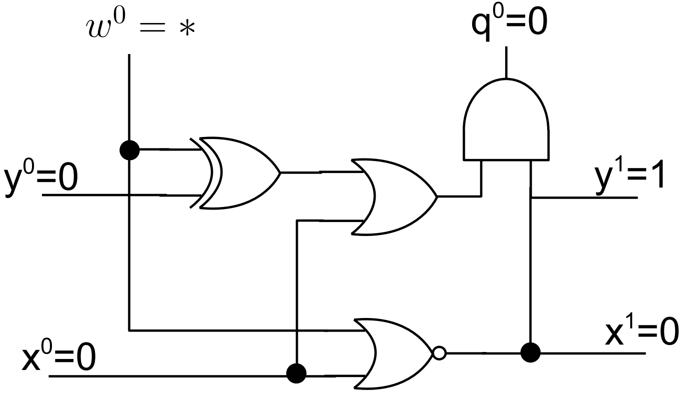
- Check if the circuit satisfies $\forall\Box\neg q$.

Initial state: x=0, y=0.



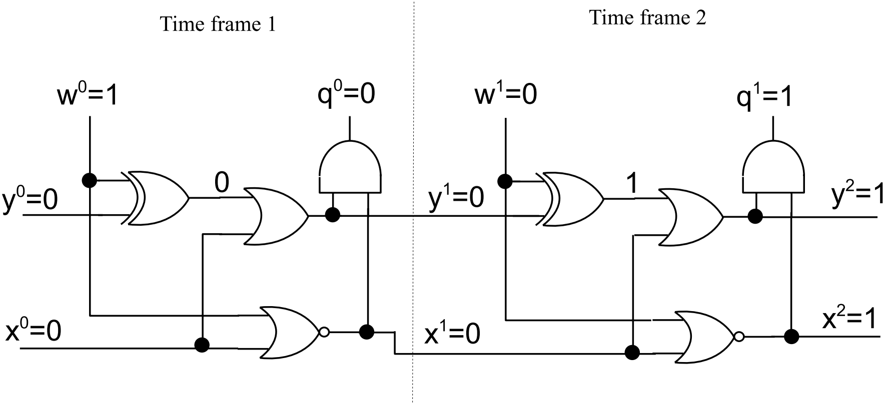

$$q = (w \oplus y \vee x) \wedge \neg(x \vee w)$$

# Circuit Initial State



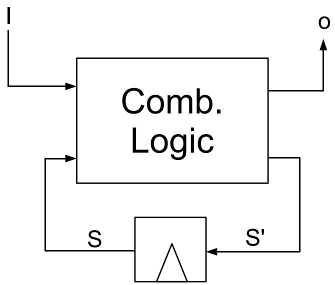

$$q^0 = (w^0 \oplus y^0 \vee x^0) \wedge \neg(x^0 \vee w^0) = 0$$

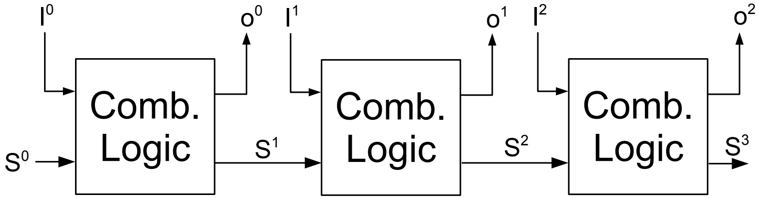
# Circuit State after Cycle $1$



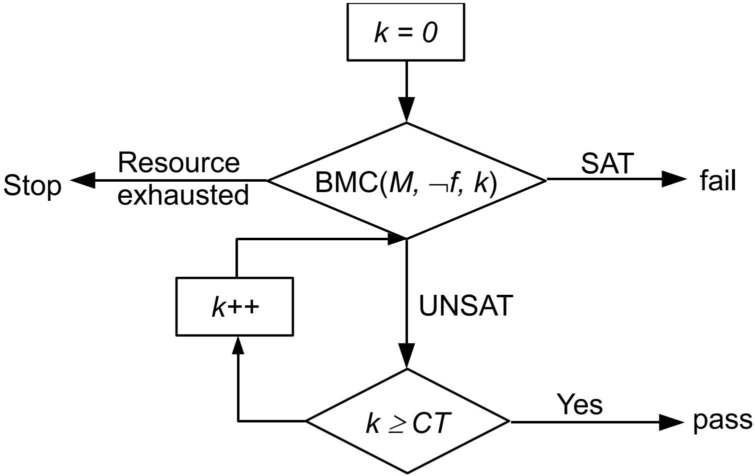

- $q^1 = 1$ if $w^0 = 1$ in the initial state and $w^1 = 0$ in cycle $1$.
- A counter-example to $\forall\Box\neg q$ is a $2$-state sequence.

# Big Picture of Bounded Model Checking



# How BMC Works



# Boolean Encoding of Bounded Model Checking

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

$[M, \neg f]_k$ is satisfiable $\Leftrightarrow$ A count-example of length $k$ exists

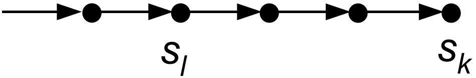
- $[M]_k$: all $k-$paths in $M(I, \Delta)$.

$$[M]_k = \underbrace{I(\vec{x}_0) \wedge \Delta(\vec{x}_0, \vec{x}_1)}_{step\ 1} \wedge \ldots \wedge \underbrace{\Delta(\vec{x}_{k-1}, \vec{x}_k)}_{step\ k} \wedge \underbrace{\Delta(\vec{x}_k, \vec{x}_l)}_{backedge\ k\ to\ l}$$

- 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.

# $k-$Bounded Paths

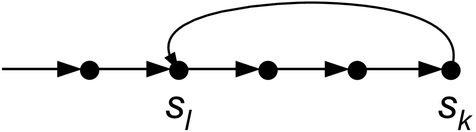
- A $k-$bounded path is a sequence of $k$ state transitions.




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

# $k-$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.

$$[M]_k = \underbrace{I(\vec{x}_0) \wedge \Delta(\vec{x}_0, \vec{x}_1)}_{step\ 1} \wedge \ldots \wedge \underbrace{\Delta(\vec{x}_{k-1}, \vec{x}_k)}_{step\ k} \wedge \underbrace{\Delta(\vec{x}_k, \vec{x}_l)}_{backedge\ k\ to\ l}$$

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

$$[M]_k = \underbrace{I(\vec{x}_0) \wedge \Delta(\vec{x}_0, \vec{x}_1)}_{step\ 1} \wedge \ldots \wedge \underbrace{\Delta(\vec{x}_{k-1}, \vec{x}_k)}_{step\ k} \wedge \underbrace{\forall 0 \leq l \leq k,\ \Delta(\vec{x}_k, \vec{x}_l)}_{backedge\ k\ to\ l}$$

# 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 $\rho \models_k^i f$ denote the truth of $f$ over the path from state $\rho(i)$ to $\rho(k)$.
- If a path $\rho$ is a $k-$loop,

$$\rho \models_k f \quad \Leftrightarrow \quad \rho \models f$$

# Bounded Semantics of LTL Formulas (2)

- $\rho \models_k f \quad \Leftrightarrow \quad \rho \models_k^0 f$ where

$$
\begin{array}{lcl}
\rho \models_k^i p & \Leftrightarrow & p \in L(\rho(i)) \\
\rho \models_k^i \neg p & \Leftrightarrow & p \notin L(\rho(i)) \\
\rho \models_k^i f \wedge g & \Leftrightarrow & \rho \models_k^i f \text{ and } \rho \models_k^i g \\
\rho \models_k^i f \vee g & \Leftrightarrow & \rho \models_k^i f \text{ or } \rho \models_k^i g \\
\rho \models_k^i \Box f & \Leftrightarrow & \text{false} \\
\rho \models_k^i \Diamond f & \Leftrightarrow & \exists i \leq j \leq k,\ \rho \models_k^j f \\
\rho \models_k^i \bigcirc f & \Leftrightarrow & i < k \text{ and } \rho \models_k^{i+1} f \\
\rho \models_k^i f \,\mathsf{U}\, g & \Leftrightarrow & \exists i \leq j \leq k,\ \rho \models_k^j f \text{ and } \forall i \leq n \leq j . \rho \models_k^n f
\end{array}
$$

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.

$$\rho \models_k \neg f \quad \Rightarrow \quad \rho \models \neg f$$

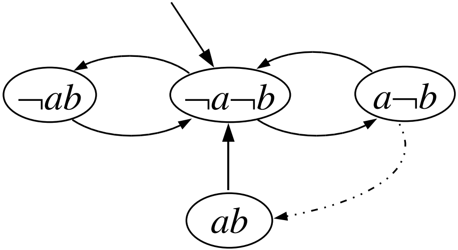
  - If there is a $\rho$ in $M$ such that $\rho \models_k \neg f$, then $M \models f$ does not hold.

    Search for $k$-bounded counter-example.
- $M \models f \quad \Leftrightarrow \quad \exists k \geq 0,\ M \models_k f$.
  - 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

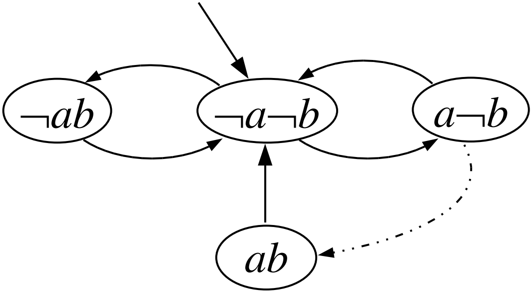
- $M \models \Box\neg(a \wedge b)$ for $k = 2$.
- $M = (I, \Delta)$ where

$$\begin{array}{rcl} I & = & \neg a \wedge \neg b \\ \Delta & = & (\neg a \wedge \neg b \wedge a' \wedge \neg b') \vee (\neg a \wedge \neg b \wedge \neg a' \wedge b') \vee \\ & & (\neg a \wedge b \wedge \neg a' \wedge \neg b') \vee (a \wedge \neg b \wedge \neg a' \wedge \neg b') \vee \\ & & (a \wedge \neg b \wedge a' \wedge b') \vee (a \wedge b \wedge \neg a' \wedge \neg b') \end{array}$$



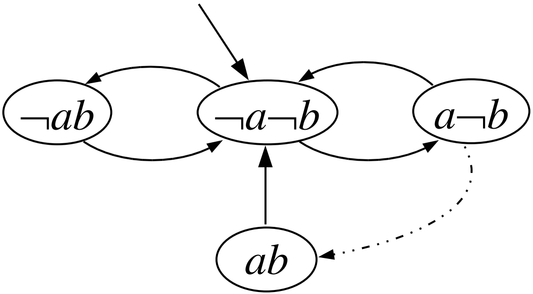
# An BMC Example

- $M \models \Box\neg(a \wedge b)$.
- BMC checks if there is a bounded path on which $\Diamond(a \wedge b)$ holds.




Check if $I(a_0, b_0) \wedge (a_0 \wedge b_0)$ is satisfiable?

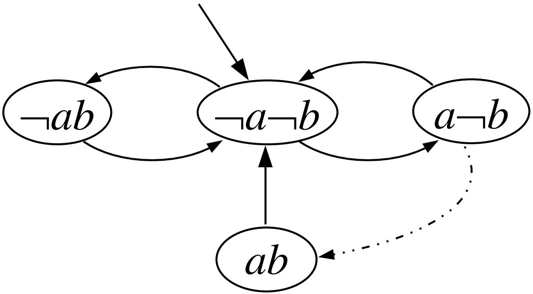
# An BMC Example – Cont'd




- $M \models_{k=1} \Box\neg(a \wedge b)$.
  - Check if the following formula is satisfiable?

$$I(a_0, b_0) \wedge \Delta(a_0, b_0, a_1, b_1) \wedge (a_1 \wedge b_1)$$

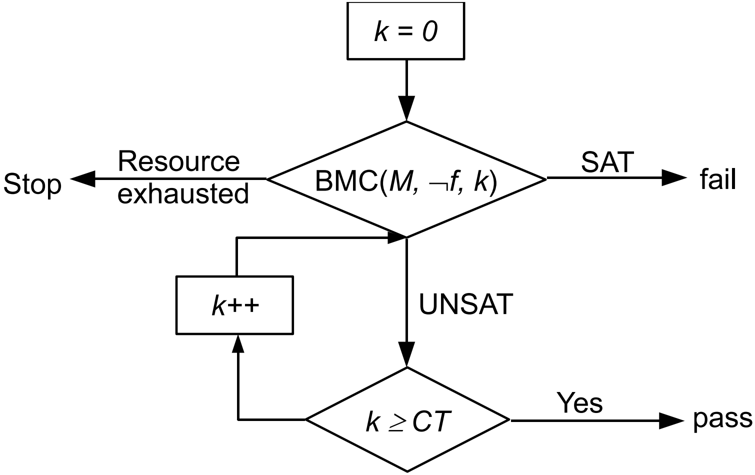
# An BMC Example – Cont'd




- $M \models_{k=2} \Box\neg(a \wedge b)$.
  - Check if the following formula is satisfiable?

$$I(a_0, b_0) \wedge \Delta(a_0, b_0, a_1, b_1) \wedge \Delta(a_1, b_1, a_2, b_2) \wedge (a_2 \wedge b_2)$$

# Bounded Model Checking: Overview



# 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

# Bounded Model Checking for 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);
```

# Encoding the C Programs

- 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

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

Code in SSA format

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

- Generated Constraints:

$$\begin{array}{ll} \textit{Program } C & x_1 = x_0 + y_0 \ \wedge \ (x_1 \neq 1 \rightarrow x_2 = 2) \\ & \wedge \ (x_1 = 1 \rightarrow x_3 = x_1 + 1) \ \wedge \\ & (x_1 \neq 1 \wedge x_4 = x_2 \ \vee \ x_1 = 1 \wedge x_4 = x_3) \\ \textit{Assertion } P & x_4 \leq 3 \end{array}$$

# Encoding: A Simple Example

Original Code

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

Code in SSA format

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

- BMC checks $C \wedge \neg P$. Assertion $P$ is violated if $C \wedge \neg P$ 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.