Introduction to Formal Methods

Lecture 3

Bounded Model Checking

Hossein Hojjat & Fatemeh Ghassemi

September 30, 2018

Bounded Model Checking

- Idea: only look for bugs up to specific depth
- One of the most successful techniques for hardware analysis
- Enabled by advances in SAT solving
- Mostly incomplete in practice
    - Can find bugs, cannot prove a system satisfies a specification

Bounded Model Checking Steps

- Represent the system and the specification symbolically
  - By using e.g. propositional logic formulas (p, $\overline{p}$, $p \vee q$, $p \wedge q$, $p \rightarrow q$,...)
- Reduce the bounded reachability problem to satisfiability of a Boolean formula
- Use efficient theorem provers (SAT solvers) for solving the satisfiability problem

Symbolic Representation of State Spaces

Key idea:

Instead of reasoning about individual states, reason about sets of states

How do we represent a set of states?

Symbolic Representation:

- Set = predicate
  Predicate $P(x)$ represents set S : $\qquad S = \{x \mid P(x)\}$
- Set of states = predicate on state variables

Symbolic Representation of Sets of States

Examples:

- Assume 3 state variables, p, q, r of type Boolean.
  $S_1 : \qquad p \vee q = \{p\overline{q}r, p\overline{qr}, \overline{p}qr, \overline{p}q\overline{r}, pqr, pq\overline{r}\}$
- Assume 3 state variables, x, i, b, of types Real, Integer, Boolean.
  $S_2 : \qquad (x \leq 0) \wedge (b \rightarrow i \geq 0)$

How many states are in S_2?

Symbolic Representation of Transition Relations

Key idea:

Use a predicate on two copies of the state variables:

unprimed (current state) + primed (next state)

- If $\vec{x}$ is the vector of state variables, then the transition relation R is a predicate on $\vec{x}$ and $\vec{x'}$
$$R(\vec{x}, \vec{x'})$$
- e.g., for three state variables, x, i, b:
$$R(x, i, b, x', i', b')$$

Symbolic Representation of Transition Relations

Examples:

- Assume one state variable, p, of type Boolean

$$R_1 : \qquad (p \rightarrow \overline{p'}) \wedge (\bar{p} \rightarrow p')$$

  Which transition relation does this represent? Is it a relation or a function (deterministic)?

- Assume one state variable, n, of type Integer.

$$R_2 : \qquad n' = n + 1 \vee n' = n$$

  Which transition relation does this represent? Is it a relation or a function (deterministic)?

Symbolic Representation of Transitions Systems

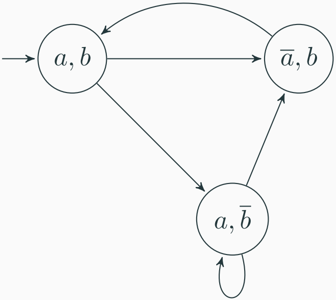
$$(V, I, R)$$

where

- $V = \{x_1, x_2, \cdots, x_n\}$: finite set of (Boolean) state variables
- Predicate $I(\vec{x})$ on vector $\vec{x} = (x_1, \cdots, x_n)$ represents the set S_0 of initial states
- Predicate $R(\vec{x}, \vec{x'})$ represents the transition relation R

Exercise

Represent the transition system symbolically $V = \{a, b\}$



Bounded Reachability

Question: Can a "bad" state be reached in up to n transitions?

Given a transition system (V, I, R) and a set of bad states P, does there exist a path

$$s_0 \longrightarrow s_1 \longrightarrow \cdots \longrightarrow s_k$$

in the transition system such that $s_0 \in I$ and $s_k \in P$, and $k \leq n$

Key idea: Reduce the above question to a SAT (satisfiability) problem.

Bounded Reachability

Transition system (V, I, R) and a set of bad states P

- Bad state reachable in 0 steps iff

$$\mathit{SAT}\big(I(\vec{x}) \wedge P(\vec{x})\big)$$

- Bad state reachable in 1 step iff

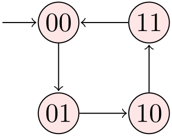
$$\mathit{SAT}\big(I(\vec{x_0}) \wedge R(\vec{x_0}, \vec{x_1}) \wedge P(\vec{x_1})\big)$$

- ...
- Bad state reachable in n steps iff

$$\mathit{SAT}\big(I(\vec{x_0}) \wedge R(\vec{x_0}, \vec{x_1}) \wedge \cdots \wedge R(\vec{x_{n-1}}, \vec{x_n}) \wedge P(\vec{x_n})\big)$$

Example: Two-bit Counter

Is the state $(x \wedge y)$ reachable from the initial state?

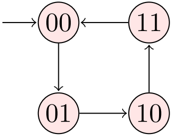


Boolean Variables: x , y

- Represent initial states and the transition relation as Boolean formulas

Example: Two-bit Counter

Is the state $(x \wedge y)$ reachable from the initial state?



Boolean Variables: x , y
Initial State: $I(x, y) = \overline{x} \wedge \overline{y}$

- Represent initial states and the transition relation as Boolean formulas

Example: Two-bit Counter

Is the state $(x \wedge y)$ reachable from the initial state?

$$
\begin{tikzcd}
\arrow[r] & 00 \arrow[d] & 11 \arrow[l] \\
& 01 \arrow[r] & 10 \arrow[u]
\end{tikzcd}
$$

Boolean Variables: x , y

Initial State: $I(x, y) = \overline{x} \wedge \overline{y}$

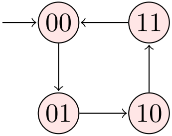
Transition Relation:

$R(x, y, x', y') = \Big(x' = (x \neq y) \wedge y' = \overline{y}\Big)$

- Represent initial states and the transition relation as Boolean formulas

Example: Two-bit Counter

Is the state $(x \wedge y)$ reachable from the initial state?



Boolean Variables: x , y

Initial State: $I(x, y) = \overline{x} \wedge \overline{y}$

Transition Relation:

$R(x, y, x', y') = \Big(x' = (x \neq y) \wedge y' = \overline{y}\Big)$

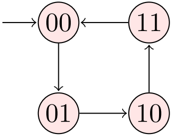
- Represent initial states and the transition relation as Boolean formulas
- Unroll the transition relation up to a bound k starting from the initial states

$$(\overline{x_0} \wedge \overline{y_0}) \quad \wedge \quad \begin{pmatrix} x_1 = (x_0 \neq y_0) \wedge y_1 = \overline{y_0} \\ \wedge \\ x_2 = (x_1 \neq y_1) \wedge y_2 = \overline{y_1} \\ \wedge \\ x_3 = (x_2 \neq y_2) \wedge y_3 = \overline{y_2} \end{pmatrix} \quad \wedge \quad \begin{pmatrix} (x_0 \wedge y_0) \\ \vee \\ (x_1 \wedge y_1) \\ \vee \\ (x_2 \wedge y_2) \\ \vee \\ (x_3 \wedge y_3) \end{pmatrix}$$

$\boxed{\text{UNSAT for } k = 0}$

Example: Two-bit Counter

Is the state $(x \wedge y)$ reachable from the initial state?



Boolean Variables: x , y

Initial State: $I(x, y) = \overline{x} \wedge \overline{y}$

Transition Relation:

$R(x, y, x', y') = \Big(x' = (x \neq y) \wedge y' = \overline{y}\Big)$

- Represent initial states and the transition relation as Boolean formulas
- Unroll the transition relation up to a bound k starting from the initial states

$$(\overline{x_0} \wedge \overline{y_0}) \quad \wedge \quad \begin{pmatrix} x_1 = (x_0 \neq y_0) \wedge y_1 = \overline{y_0} \\ \wedge \\ x_2 = (x_1 \neq y_1) \wedge y_2 = \overline{y_1} \\ \wedge \\ x_3 = (x_2 \neq y_2) \wedge y_3 = \overline{y_2} \end{pmatrix} \wedge \begin{pmatrix} (x_0 \wedge y_0) \\ \vee \\ (x_1 \wedge y_1) \\ \vee \\ (x_2 \wedge y_2) \\ \vee \\ (x_3 \wedge y_3) \end{pmatrix}$$

UNSAT for $k = 1$

Example: Two-bit Counter

Is the state $(x \wedge y)$ reachable from the initial state?

$$\begin{tikzcd} \arrow[r] & 00 \arrow[d] & 11 \arrow[l] \\ & 01 \arrow[r] & 10 \arrow[u] \end{tikzcd}$$

Boolean Variables: x , y

Initial State: $I(x, y) = \overline{x} \wedge \overline{y}$

Transition Relation:

$R(x, y, x', y') = \Big(x' = (x \neq y) \wedge y' = \overline{y}\Big)$

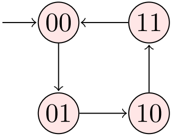
- Represent initial states and the transition relation as Boolean formulas
- Unroll the transition relation up to a bound k starting from the initial states

$$(\overline{x_0} \wedge \overline{y_0}) \quad \wedge \quad \begin{pmatrix} x_1 = (x_0 \neq y_0) \wedge y_1 = \overline{y_0} \\ \wedge \\ x_2 = (x_1 \neq y_1) \wedge y_2 = \overline{y_1} \\ \wedge \\ x_3 = (x_2 \neq y_2) \wedge y_3 = \overline{y_2} \end{pmatrix} \wedge \begin{pmatrix} (x_0 \wedge y_0) \\ \vee \\ (x_1 \wedge y_1) \\ \vee \\ (x_2 \wedge y_2) \\ \vee \\ (x_3 \wedge y_3) \end{pmatrix}$$

$\boxed{\text{UNSAT for } k = 2}$

Example: Two-bit Counter

Is the state $(x \wedge y)$ reachable from the initial state?



Boolean Variables: x , y

Initial State: $I(x, y) = \overline{x} \wedge \overline{y}$

Transition Relation:

$R(x, y, x', y') = \Big(x' = (x \neq y) \wedge y' = \overline{y}\Big)$

- Represent initial states and the transition relation as Boolean formulas
- Unroll the transition relation up to a bound k starting from the initial states

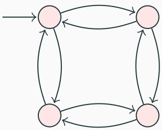
$$
(\overline{x_0} \wedge \overline{y_0}) \quad \wedge \quad \left(\begin{array}{c} x_1 = (x_0 \neq y_0) \wedge y_1 = \overline{y_0} \\ \wedge \\ x_2 = (x_1 \neq y_1) \wedge y_2 = \overline{y_1} \\ \wedge \\ x_3 = (x_2 \neq y_2) \wedge y_3 = \overline{y_2} \end{array} \right) \wedge \left(\begin{array}{c} (x_0 \wedge y_0) \\ \vee \\ (x_1 \wedge y_1) \\ \vee \\ (x_2 \wedge y_2) \\ \vee \\ (x_3 \wedge y_3) \end{array} \right)
$$

$\boxed{\text{SAT for } k = 3}$

Completeness

- Typical application of Bounded Model Checking: increment depth until counter-example found
- Incomplete BMC good for falsification not verification
- Can be used for verification by choosing depth which is large enough
- For every **finite** system and a property, there exists a number such that the absence of errors up to that number proves correctness
- Diameter d = longest shortest path from an initial state to any other reachable state

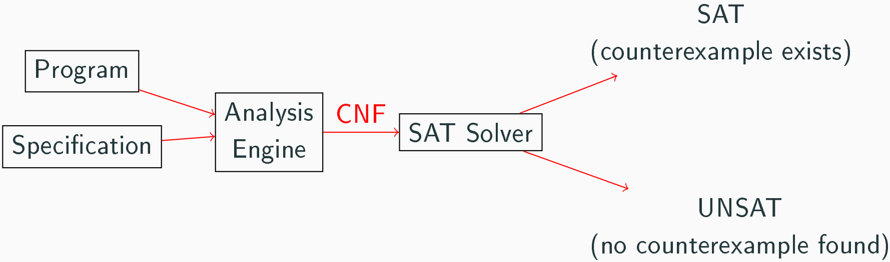
$d = 2$



- Using diameter bound is often not practical
- Worst case diameter is exponential. Obtaining better bounds is sometimes possible, but generally intractable

Bug Catching with SAT-Solvers

Main Idea:
Given a program and a specification use a SAT-solver to find whether there exists an execution that violates the specification



CBMC

- Bounded model checker by University of Oxford
  `http://www.cprover.org/cbmc/`
- Processes C code and subset of C++
- Simple Safety Claims
  - Array bound checks (i.e., buffer overflow)
  - Division by zero
  - Pointer checks (i.e., NULL pointer dereference)
  - Arithmetic overflow
  - User supplied assertions (i.e., assert (i > j))
  - etc

How does it work

Transform a programs into a set of equations

1. Simplify control flow
2. Unwind all of the loops
3. Convert into Single Static Assignment (SSA)
4. Convert into equations
5. Bit-blast
6. Solve with a SAT Solver
7. Convert SAT assignment into a counterexample

Control Flow Simplifications

- All side effect are removed
  - e.g., `j=i++` becomes `j=i;i=i+1`
- Control Flow is made explicit
  - `continue`, `break` replaced by `goto`
- All loops are simplified into one form
  - `for`, `do while` replaced by `while`

Loop Unwinding

- All loops are unwound
    - can use different unwinding bounds for different loops
    - to check whether unwinding is sufficient special "unwinding assertion" claims are added
- If a program satisfies all of its claims and all unwinding assertions then it is correct!
- Same for backward `goto` jumps and recursive functions

Loop Unwinding

```
void f(...) {
  ...
  while(cond) {
    Body;
  }
  Remainder;
}
```

- `while` loops are unwound iteratively
- `break` / `continue` replaced by `goto`

Loop Unwinding

```
void f(...) {
  ...
  if(cond) {
    Body;
    while(cond) {
      Body;
    }
  }
  Remainder;
}
```

- `while` loops are unwound iteratively
- `break` / `continue` replaced by `goto`

Loop Unwinding

```
void f(...) {
  ...
  if(cond) {
    Body;
    if(cond) {
      Body;
      while(cond) {
        Body;
      }
    }
  }
  Remainder;
}
```

- `while` loops are unwound iteratively
- `break` / `continue` replaced by `goto`

Loop Unwinding

```
void f(...) {
  ...
  if(cond) {
    Body;
    if(cond) {
      Body;

      assert(!cond)

    }
  }
  Remainder;
}
```

- `while` loops are unwound iteratively
- `break` / `continue` replaced by `goto`
- Assertion inserted after last iteration: violated if program runs longer than bound permits

Example: Sufficient Loop Unwinding

```
void f(...) {
  j = 1
  while (j <= 2)
    j = j + 1;
  Remainder;
}
```

```
void f(...) {
  j = 1
  if(j <= 2) {
    j = j + 1;
    if(j <= 2) {
      j = j + 1;
      if(j <= 2) {
        j = j + 1;
        assert(!(j <= 2));
      }
    }
  }
  Remainder;
}
```

Example: Insufficient Loop Unwinding

```
void f(...) {
  j = 1
  while (j <= 10)
    j = j + 1;
  Remainder;
}
```

```
void f(...) {
  j = 1
  if(j <= 10) {
    j = j + 1;
    if(j <= 10) {
      j = j + 1;
      if(j <= 10) {
        j = j + 1;
        assert(!(j <= 10));
      }
    }
  }
  Remainder;
}
```

Transforming Loop-Free Programs Into Equations

Easy to transform when every variable is only assigned once!

```
x = a;
y = x + 1;
z = y - 1;
```

$$\longrightarrow$$

$$x = a \land$$
$$y = x + 1 \land$$
$$z = y - 1$$

Transforming Loop-Free Programs Into Equations

When a variable is assigned multiple times, use a new variable for the RHS of each assignment

```
x = x + y;
x = x * 2;
a[i] = 100;
```

$\longrightarrow$

$$x_1 = x_0 + y_0 \wedge$$
$$x_2 = x_1 \times 2 \wedge$$
$$a_1[i_0] = 100$$

Bit-blasting

- Machine arithmetic: Bounded integer e.g., 8bit, 32bit, 64bit
- Numbers are represented by vector of Boolean variables $\langle b_{n-1}, b_{n-2}, ..., b_0 \rangle$
- Encoding of overflow/rounding-behaviour derived from hardware implementation

Reference

- E. Clarke, A. Biere, R. Raimi, and Y. Zhu.
  "Bounded model checking using satisfiability solving",
  Formal Methods in System Design, 19(1):7-34, 2001