Introduction to Software Verification

Orna Grumberg

Lectures Material
winter 2015-16
Lecture 11

- Bounded Model Checking (continued)
- CBMC
- Efficient SAT solvers
Bounded (SAT-based) Model Checking
Bounded model checking (BMC) for checking $\text{AGp}$

- **Given**
  - A finite system $M$
  - A safety property $\text{AGp}$
  - A bound $k$

- **Determine**
  - Does $M$ contain a counterexample to $p$ of $k$ transitions (or fewer)?
BMC for checking $\varphi = \text{EF} \neg p$

1. $k=1$

2. Build a propositional formula $f_M^k$ describing all prefixes of length $k$ of paths of $M$ from an initial state

3. Build a propositional formula $f_\varphi^k$ describing all prefixes of length $k$ of paths satisfying $\varphi$

4. If $(f_M^k \land f_\varphi^k)$ is satisfiable, return the satisfying assignment as a counterexample

5. Otherwise, increase $k$ and return to 2.
BMC for checking $\varphi = EF \neg p$

- $f_M^k (V_0, \ldots, V_k) = \text{INIT}(V_0) \land \bigwedge_{i=0,\ldots,k-1} R(V_i, V_{i+1})$

- $f_{\varphi}^k (V_0, \ldots, V_k) = V_{i=0\ldots k} \neg p(V_i)$
BMC for checking $\text{AFp} \ (\varphi = \text{EG} \neg p)$

- Is there an infinite path in $M$
  - From an initial state
  - all of its states satisfying $\neg p$
  - Over $k+1$ states?

- Must be a lasso
BMC for checking $\text{AFp (} \varphi = \text{EG} \neg \varphi)$

An infinite path in $M$, from an initial state, over $k+1$ states, all satisfying $\neg \varphi$:

- $f_M^k (V_0, \ldots, V_k) =$
  \[ \text{INIT}(V_0) \land \bigwedge_{i=0}^{k-1} R(V_i, V_{i+1}) \land V_i=0, \ldots, k-1 (V_k=V_i) \]

- $V_k=V_i$ means bitwise equality: $\bigwedge_{j=0}^{n} (v_{kj} \Leftrightarrow v_{ij})$

- $f_\varphi^k (V_0, \ldots, V_k) = \bigwedge_{i=0}^{k} \neg \varphi (V_i)$
CBMC: C Bounded Model Checker
Bounded Model Checking of C programs

Based on slides by
Arie Gurfinkel
A (very) simple example (1)

Program

```c
int x;
int y=8,z=0,w=0;
if (x)
    z = y - 1;
else
    w = y + 1;
assert (z == 7 || w == 9)
```

Constraints

```c
y = 8,
z = x ? y - 1 : 0,
w = x ? 0 : y + 1,
z != 7,
w != 9
```

UNSAT
no counterexample
assertion always holds!
int x;
int y=8,z=0,w=0;
if (x)
    z = y - 1;
else
    w = y + 1;
assert (z == 5 || w == 9)
How does CBMC work

Transform a program 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
Example: Sufficient Loop Unwinding

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

unwind = 3
```

```c
void f(...) {
  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;
}

unwind = 3

void f(...) {
    j = 1
    if(j <= 10) {
        j = j + 1;
        if(j <= 10) {
            j = j + 1;
            assert(!(j <= 10));
        }
    }
    Remainder;
}
Transforming Loop-Free Programs Into Equations (2)

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

Program

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

SSA Program

\[ x_1 = x_0 + y_0; \]
\[ x_2 = x_1 * 2; \]
\[ a_1[i_0] = 100; \]

Single Static Assignment (SSA)
What about conditionals?

For each join point, add new variables with selectors
Example

```c
int main() {
    int x, y;
    y=8;
    if(x)
        y--;
    else
        y++;
    assert
        (y==7 ||
            y==9);
}
```

( $y_1 = 8$
$\land y_2 = y_1 - 1$
$\land y_3 = y_1 + 1$
$\land y_4 = x_0 \ ? y_2 : y_3$
$\Rightarrow (y_4=7 \lor y_4=9)$
valid?)
Example

```c
int main() {
    int x, y;
    y=8;
    if(x)
        y--;
    else
        y++;
    assert
        (y==7 || y==9);
}
```

```c
int main() {
    int x, y;
    y1=8;
    if(x0)
        y2=y1-1;
    else
        y3=y1+1;
    y4= x0 ? y2 : y3;
    assert
        (y4==7 || y4==9);
}
```

```c
( y1 = 8
\land y2 = y1 - 1
\land y3 = y1 + 1
\land y4 = x0 \ ? y2 : y3 )
\land \neg(y4=7 \lor y4=9)

Unsat?
```
Efficient SAT solvers
The SAT Problem

• Given a propositional formula $\varphi(\overline{v})$, is there a satisfying assignment $A$ for $\overline{v}$

• An assignment is a function from $\overline{v}$ to $\{true, false\}$

• $A$ is a satisfying assignment if $\varphi(A(\overline{v})) = true$

• $A$ is called a solution for $\varphi(\overline{v})$

• A partial assignment assigns a subset of $\overline{v}$
CNF representation of $\varphi(\overline{v})$

- $\varphi(\overline{v})$ is a conjunction of clauses: $\varphi(\overline{v}) = cl_1 \land cl_2 \land \ldots \land cl_n$
- A clause is a disjunction of literals: $cl_i = (lit_1 \lor \ldots \lor lit_l)$
- A literal is an atomic proposition or its negation

- Example:
  $(a \lor c) \land (b \lor c) \land (\neg a \lor \neg b \lor d)$

- $A$ satisfies $\varphi(\overline{v})$ iff $A$ satisfies all its clauses
• Example:

$$(a \lor c) \land (b \lor c) \land (\neg a \lor \neg b \lor d)$$

• Satisfying assignments:
  - $A_1 = (a=true, b=true, d=true, c=false)$
  - $A_2 = (c=true, a=false, b=true, d=false)$

• CNF formulas
  - Clause does not contain $a$, $\neg a$
  - No repetition of literals in clause

• CNF formulas can be represented as a set of sets of literals
Searching for a satisfying assignment

• **Inefficient way:**
  check each one of the $2^n$ assignments
  where $|\overline{v}| = n$

• **The basis for efficient SAT solving:**
  Davis, Putnam, Logeman, Loveland (DPLL)
  1960, 1962
First idea: **Unit Clause**

Given
- a propositional formula $\phi(\overline{v})$, and
- a partial assignment $A$

A Unit Clause is a clause with
- exactly one unassigned literal, while
- all other literals are false

- **Asserts** the value of the unassigned variable

\[
\begin{align*}
    a &= \,? \\
    b &= 1 \\
    d &= 0 \\
    cl &= (a \lor \neg b \lor d) \\
\end{align*}
\]

$\implies a = 1$

- $a=1$ is *implied* by $b=1$ and $d=0$
Boolean Constraint Propagation (BCP)

- **BCP**: For $\varphi(\overline{v})$ and a partial assignment $A$, computes all possible implications
  - Based on unit clauses
- **Conflict**: a variable gets both 0 and 1 under $A$
DPLL algorithm

1. Start with an empty partial assignment $A$
2. If $A$ is complete (no new decision to make), return ($SAT$, $A$)
3. Otherwise, extend $A$ with a decision: $D=$(variable, value)
4. BCP: extend $A$ with all implications of $D$
5. If (no conflict) go to 2
6. If (conflict), apply backtracking:
   - Let $D=$(v,b) be the last decision s.t. (v, !b) has not been checked yet
   - Flip decision $D$: Remove (v,b), extend $A$ with (v, !b)
   - Undo all implications from $D=$(v,b) and from flips made after $D$
   - Return to 4
7. If no decision to flip, return UNSAT
DPLL algorithm

• Termination
  - No unassigned variable – SAT
  - No decision variable to flip – UNSAT
\[(\neg b \lor c) \land (\neg a \lor \neg d) \land (a \lor b \lor \neg c) \land (\neg a \lor d) \land (a \lor \neg c \lor \neg e)\]

Decision 1: \(b=1\)
BCP: \(c=1\)

\[\square \land (\neg a \lor \neg d) \land \square \land (\neg a \lor d) \land (a \lor \neg c \lor \neg e)\]

Decision 2: \(a=1\)
BCP: \(d=0\)
\(d=1\)

Conflict! -Backtrack
\[ (-b \lor c) \land (-a \lor \neg d) \land (a \lor b \lor \neg c) \land (-a \lor d) \land (a \lor \neg c \lor \neg e) \]

Decision 1: b=1

BCP: c=1

\[ [ \quad ] \land (-a \lor \neg d) \land [ \quad ] \land (-a \lor d) \land (a \lor \neg c \lor \neg e) \]

Decision 2: a=0
\[(\neg b \lor c) \land (\neg a \lor \neg d) \land (a \lor b \lor \neg c) \land (\neg a \lor d) \land (a \lor \neg c \lor \neg e)\]

Decision 1: \(b = 1\)
BCP: \(c = 1\)

\[
[\quad] \land [\quad] \land [\quad] \land [\quad] \land (a \lor \neg c \lor \neg e)
\]

Decision 2: \(a = 0\)
BCP: \(e = 0\)

Partial satisfying assignment:
\(b = 1, c = 1, a = 0, e = 0\)
Other solutions to the state-explosion problem

Small models replace the full, concrete model:

- Abstraction
- Compositional verification
- Partial order reduction
- Symmetry