Introduction to Software Verification

Orna Grumberg

Lectures Material
winter 2017-18
Lecture 9
BDD-based Symbolic Model Checking

A solution to the state explosion problem

• Binary Decision Diagrams (BDDs) are used to represent the model and sets of states.

• It can handle systems with hundreds of Boolean variables.
Operations on BDDs
Operations on BDDs - Reduce

Reduce

Given an unreduced BDD:

• Eliminate isomorphic sub-graphs:
  – Eliminate duplicated end nodes
  – Eliminate duplicated internal nodes

• Eliminate redundant nodes

Reduce works bottom-up in linear time in the BDD size
Important remark:

BDD for a complex function is built bottom-up starting from small sub-functions to larger ones

We do not build a full decision tree and then reduce
Operations on BDDs - Restrict

Restrict

Given a BDD for $f(x_1,\ldots,x_n)$, build a BDD for

$$f\big|_{x_i=b}(x_1,\ldots,x_n) = f(x_1,\ldots,x_{i-1},b,x_{i+1},\ldots,x_n) \quad b \in \{0,1\}$$

Example:

$f(x_1,x_2,x_3,x_4) = (x_1 \land x_2) \lor (x_3 \land x_4)$

$f\big|_{x_2=0}(x_1,x_2,x_3,x_4) = (x_1 \land 0) \lor (x_3 \land x_4) = (x_3 \land x_4)$
Operations on BDDs - Apply

• Gets two BDDs, representing functions \( f \) and \( f' \) and an operation \( * \)
  – Over the same variable ordering

• Returns the BDD representing \( f*f' \)

• \( * \) can be any of 16 binary operations on two Boolean functions
Operations on BDDs - Apply

• Shannon expansion
  for every Boolean function \( f \) and a variable \( x \):
  \[
  f = (\neg x \land f|_{x=0}) \lor (x \land f|_{x=1})
  \]

Notation:
• \( v, v' \) are the roots of \( f, f' \), respectively
• If \( v, v' \) are not end nodes then \( \text{var}(v) = x \), \( \text{var}(v') = x' \)
Operations on BDDs - Apply

Computing $f \cdot f'$:

• **Case 1**: $v$ and $v'$ are end nodes
  $$f \cdot f' = \text{value}(v) \times \text{value}(v')$$

• The BDD for $f \cdot f'$$
  consists of one leaf $v''$ with
  $$\text{value}(v'') = \text{value}(v) \times \text{value}(v')$$

This is the only case where $\cdot$ is taken into account
Operations on BDDs - Apply

Computing $f*f'$:
• **Case 2**: $x = x'$
• Use Shannon expansion:

$$f*f' = (\neg x \land ( f|_{x=0} * f'|_{x=0} ) ) \lor$$
$$\quad ( x \land ( f|_{x=1} * f'|_{x=1} ) )$$

• Two simpler sub-problems to solve
  – Each depends on one less variable
Operations on BDDs - Apply

Computing $f*f'$:

- **Case 2**: $x = x'$

- The BDD for $f*f'$
- **Root**: a new node $v''$
  - $\text{var}(v'') = x$
  - $\text{low}(v'')$ points to the root of the BDD for $(f|_{x=0} \cdot f'|_{x=0})$
  - $\text{high}(v'')$ points to the root of the BDD for $(f|_{x=1} \cdot f'|_{x=1})$
Example

- \( f(a) = a, \quad f'(a) = \neg a, \quad \ast \text{ is } \lor \)

- The BDD for \( f \lor f' \) is:

- The BDD for \( f \lor f' \) is:
Operations on BDDs - Apply

Computing $f \cdot f'$ :

- **Case 3:** $x < x'$

- $x$ does not appear in $f'$

  
  $f' \mid_{x=0} = f' \mid_{x=1} = f' $

- Use Shannon expansion as before:

  
  $f \cdot f' = (\neg x \land (f \mid_{x=0} \ast f')) \lor (x \land (f \mid_{x=1} \ast f')) $
Operations on BDDs - Apply

Computing $f \cdot f'$:

- **Case 4**: $x > x'$
  
  Similar to case 3
Example

- \( f(a,b) = a \rightarrow b, \quad f'(a,b) = \neg b, \quad * \text{ is } \leftrightarrow \)
- \( f \leftrightarrow f' \equiv (a \rightarrow b) \leftrightarrow (\neg b) \equiv (\neg a \lor b) \leftrightarrow (\neg b) \equiv (\neg a \land \neg b) \)
Example

- \( f(a,b) = a \rightarrow b, \quad f'(a,b) = \neg b, \quad * \text{ is } \leftrightarrow, \quad a < b \)
Example

• \( f(a,b) = a \rightarrow b, \quad f'(a,b) = \overline{b}, \quad \ast \text{ is } \leftrightarrow \)
Complexity of apply
Naive implementation

- two sub-problems for each variable
- exponential in the number of variables
Complexity of apply
Non-naive implementation

Notice:
• Every BDD node $u$ represents a function $f_u$
• $|f|$, $|f'|$ denote the number of nodes in the BDD for $f$, $f'$ respectively

Solution:
• Use hash table with entries:
  – Pointers to (the root node of) the BDDs for $g$, $g'$, and $\ast$
  – Pointer to the resulting BDD for $g * g'$
Consequences:
• Never redo an operation on the same BDDs
  – Never solve the same sub-problem twice
• Never insert into the BDD manager the same BDD twice

Complexity
• The number of different sub-problems is $O(|f|x|f'|)$
  – Polynomial in the BDD sizes
Symbolic (BDD-based) Model Checking for CTL
Symbolic (BDD-based) model checking

• Explicit-state model checking applies graph algorithms (for example: BFS, DFS, SCC)
• BDDs are not suitable for that
  - Highly inefficient

• BDD-based model checking manipulates set of states
  - BDD efficiently represents Boolean function which represents a set of states
Operations on sets

- Union of sets $\Rightarrow \lor$ (or) over their BDDs
- Intersection $\Rightarrow \land$ (and)
- Complementation $\Rightarrow \neg$ (not)
- Equality of sets $\Rightarrow \iff$ (iff)
Two additional operations

• $\exists x_i \ f(x_1,\ldots,x_n) = f|_{x_i=0} \lor f|_{x_i=1}$

• $\forall x_i \ f(x_1,\ldots,x_n) = f|_{x_i=0} \land f|_{x_i=1}$

• No additional expressive power
• Can be implemented with apply + restrict
  - Exponential in the number of quantified variables
• Heuristics can be more efficient, but not in the worst case
BDD-based Model Checking

- Accept: Kripke structure $M$, CTL formula $f$
- Returns: $S_f$ - the set of states satisfying $f$

$M$ is given by:
- BDD $R(V,V')$, representing the transition relation
- BDD $p(V)$, for every $p \in AP$, representing $S_p$
  - the set of states satisfying $p$
- $V = (v_1,\ldots,v_n)$
BDD-based Model Checking

• The algorithm works from simpler formulas to more complex ones.
• When a formula $g$ is handled, the BDD for $S_g$ is built.
• A formula is handled only after all its sub-formulas have been handled.
BDD-based Model Checking

- For $p \in \text{AP}$, return $p(V)$
- For $f = f_1 \land f_2$, return $f(V) = f_1(V) \land f_2((V)$ (using apply)
- For $f = \neg f_1$, return $f(V) = \neg f_1(V)$
BDD-based Model Checking

• For $f = \text{EX} \ f_1$ return
  
  $f(V) = \exists V' \ [ \ f_1(V') \land R(V,V') ]$

• This BDD represents all (encoding $V$ of) states that have a successor (with encoding $V'$) in $f_1$
• Defined as a new BDD operator:
  \[ \text{EX } f_1(V) = \exists V' [ f_1(V') \land R(V,V') ] \]

• This operation is also called pre-image

• Important:
  the formula defines a sequence of BDD operations and therefore is considered as a symbolic algorithm
Model Checking $f = EF \ g$

Given: BDDs $R(V, V')$ and $g(V)$:

procedure CheckEF ($g(V)$)

\[ Q(V) := \text{emptyset}; \quad Q'(V) := g(V) ; \]

while $Q(V) \neq Q'(V)$ do

\[ Q(V) := Q'(V); \]

\[ Q'(V) := Q(V) \lor \text{EX} ( Q(V) ) \]

end while

\[ f(V) := Q(V) ; \quad \text{return}(f(V)) \]
The algorithm applies
• BDD operations (or $\lor$), and EX
• comparison $Q(V) \neq Q'(V)$ (easy)
Therefore, this is a symbolic algorithm!

The algorithm is based on the equivalence:
$$EF \ g \equiv g \lor EX \ EF \ g$$
Example: $f = EF g$
Model Checking $f = E[g_1 U g_2]$  

Given: BDDs $R(V, V')$, $g_1(V)$ and $g_2(V)$:

procedure CheckEU ($g_1$, $g_2$)  
  $Q := \text{emptyset};$  $Q' := g_2$;  
  while $Q \neq Q'$ do  
    $Q := Q'$;  
    $Q' := Q \lor (\text{EX}(Q) \land g_1)$  
  end while  
  $f := Q; \quad \text{return}(f)$
Model Checking $f = EG\ g$

Given: BDDs $R(V, V')$, $g(V)$

procedure CheckEG (g)

\[
\begin{align*}
Q &:= S ; \quad Q' := g ; \\
\text{while } Q \neq Q' \text{ do} & \\
& \quad Q := Q' ; \\
& \quad Q' := Q \land EX (Q) \\
\text{end while} & \\
\text{return} (f) &
\end{align*}
\]
Example: $f = EG \ g$
Bounded (SAT-based) Model Checking
State explosion problem - revisited

- state of the art symbolic model checking can handle effectively designs with a few hundreds of Boolean variables

Other solutions for the state explosion problem are needed!
SAT-based model checking

- Translates the model and the specification to a propositional formula
- Uses efficient tools (SAT solvers) for solving the satisfiability problem

Since the satisfiability problem is NP-complete, SAT solvers are based on heuristics.
SAT tools

- Using heuristics, SAT tools can solve very large problems fast.
- They can handle systems with 1000 variables that create formulas with a few millions of variables.

GRASP (Silva, Sakallah)
Prover (Stalmark)
Chaff (Malik)
MiniSAT
Bounded model checking (BMC) for checking AGp

• Given
  - A finite system $M$
  - A safety property $AGp$
  - A bound $k$

• Determine
  - Does $M$ contain a counterexample to $AGp$ of $k$ transitions (or fewer)?
Bounded Model Checking (BMC) for checking AGp

- Unwind the model for \( k \) levels, i.e., construct all computations of length \( k \)

- If a state satisfying \( \neg p \) is encountered, produce a counterexample; Otherwise, increase \( k \)

[BCCZ 99]