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: BDD-based model checking

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

- It can handle systems with hundreds of Boolean variables.
BDD for \( f(a,b,c) = (a \land b) \lor c \)

Decision tree

BDD
Advantage of BDDs (revisited)

• Often (but not always) concise in size

• **Canonical** representation for a given variable ordering
  – Easy to check equivalence between two functions

• A function depends exactly on all variables that appear in its BDD

• Most **Boolean operations** can be performed on BDDs in **polynomial time** in the BDD size
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

\[
f \mid _{x_i = b} (x_1, \ldots, x_n) = f(x_1, \ldots, x_{i-1}, b, x_{i+1}, \ldots, x_n) \quad b \in \{0, 1\}
\]

Example:

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

\( f \mid _{x_2 = 0} (x_1, x_2, x_3, x_4) = (x_1 \land 0) \lor (x_3 \land x_4) = (x_3 \land x_4) \)
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
  \[ f \cdot f' = \text{value}(v) \cdot \text{value}(v') \]

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

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:

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

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

Computing $f \cdot f'$ :

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

- The BDD for $f \cdot f'$

- 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

- $f(a) = a$, $f'(a) = \neg a$, * is $\lor$

\begin{itemize}
  \item The BDD for $f \lor f'$ is:
\end{itemize}

- $f(a) = a$, $f'(a) = \neg a$, * is $\lor$
Operations on BDDs - Apply

Computing $f*f'$:

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

  - $x$ does not appear in $f'$
  
  \[ f'|_{x=0} = f'|_{x=1} = f' \]

  - Use Shannon expansion as before:

    \[
    f*f' = (\neg x \land (f|_{x=0} \ast f')) \lor \\
    (x \land (f|_{x=1} \ast f'))
    \]
Operations on BDDs - Apply

Computing $f \cdot f'$:

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

• $f(a,b) = a \rightarrow b$, $f'(a,b) = \lnot b$, * is $\iff$

• $f \iff f' \equiv (a \rightarrow b) \iff (\lnot b) \equiv (\lnot a \lor b) \iff (\lnot b)$

$\equiv (\lnot a \land \lnot b)$
Example

- $f(a, b) = a \rightarrow b$, $f'(a, b) = \neg b$, * is $\leftrightarrow$, $a < b$
Example

- $f(a,b) = a \rightarrow b$,  $f'(a,b) = \neg b$,  * 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 \(*\)
  – 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 $O(|f| \times |f'|)$
  – 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 $\Rightarrow \leftrightarrow$ (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 $p$
• $V = (v_1, ... v_n)$
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 \( p \in AP \), 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 \text{pre-image}.

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

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

procedure $\text{CheckEF} \ (g(V))$

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

done
Model Checking $f = \text{E}[g_1 \lor g_2]$ 

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

procedure $\text{CheckEU} (g_1, g_2)$

$Q := \text{emptyset}$; $Q' := g_2$

while $Q \neq Q'$ do

$Q := Q'$;
$Q' := Q \lor (\text{EX}(Q) \land g_1)$

end while

$f := Q$; return($f$)
Model Checking \( f = EG \ g \)

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

procedure \texttt{CheckEG} (\( g \))
\[
Q := S ; \quad Q' := g ;
\]
while \( Q \neq Q' \) do
\[
Q := Q' ;
\]
\[
Q' := Q \land EX (Q) ;
\]
end while
\[
f := Q ; \quad \text{return}( f )
\]
Example: \( f = EG \ g \)