Introduction to Software Verification

Orna Grumberg

Lectures Material
winter 2015-16
Lecture 9
Symbolic (BDD-based) Model Checking for CTL
BDDs as a data structure

A BDD for a Boolean function $f(x_1, \ldots, x_k)$ is an directed acyclic graph (DAG) with a root and two types of nodes:

- **Internal nodes $v$** with fields
  - $\text{var}(v)$ containing a variable name
  - Pointers $\text{low}(v), \text{high}(v)$ to other nodes

- **End nodes (leaves) $v$** with field
  - $\text{value}(v) \in \{0, 1\}$
Reduced, Ordered BDD (ROBDD)

To obtain a canonical representation, the following conditions are added:

• A variable appears at most once along every path from root to leaf
• The variables appear in the same order along every path from root to leaf
• The graph does not contain
  - isomorphic sub-graphs
  - Redundant nodes
Two graphs with root nodes u and v are isomorphic iff

- If u and v are leaves then value(u)=value(v)
- If u and v are internal nodes then
  - var(u) = var(v)
  - low(u) and low(v) are isomorphic
  - high(u) and high(v) are isomorphic

A node v is redundant if low(v)=high(v)
Remark: From now on we will use BDD to denote ROBDD
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 \( f(x_1,...x_n) \), build a BDD for

\[
f|_{x_i=b} (x_1,...x_n) = f(x_1,...,x_{i-1},b,x_{i+1},...x_n)
\]

Example:

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

\[
f|_{x_2=0} (x_1,x_2,x_3,x_4) = (x_1 \land 0) \lor (x_3 \land x_4) = (x_3 \land x_4)
\]
Operations on BDDs - Restrict

Given a BDD A for \( f(x_1, \ldots, x_n) \), build a BDD B for
\[
f|_{x_i=b} (x_1, \ldots, x_n) = f(x_1, \ldots, x_{i-1}, b, x_{i+1}, \ldots, x_n)
\]

- Traverse A from root to leaves
- For every node \( v \) with \( \text{var}(v) = x_i \)
  - Eliminate \( v \) from B
  - Replace edges to \( v \) by edges to low(v), if \( b=0 \) and to high(v), if \( b=1 \)
- Run Reduce
Example: Restrict

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

\[ f|_{x_2=0} (x_1, x_2, x_3, x_4) = (x_1 \land 0) \lor (x_3 \land x_4) = (x_3 \land x_4) \]
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
16 binary operations

<table>
<thead>
<tr>
<th>$f$</th>
<th>$f'$</th>
<th>$f_1$</th>
<th>$f_2$</th>
<th>$f_3$</th>
<th>...</th>
<th>$f_{15}$</th>
<th>$f_{16}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>...</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>...</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

const 0 AND  ?  OR  const 1
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 $\lor$

• The BDD for $f \lor f'$ is:

• The BDD for $f \lor f'$ is:
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:

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

Computing $f \ast f'$:

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

• $f(a,b) = a \rightarrow b$, $f'(a,b) = \neg b$, $*$ 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

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

• $f(a, b) = a \rightarrow b$, $f'(a, b) = \neg b$, * is $\leftrightarrow$

```
Example

• $f(a, b) = a \rightarrow b$, $f'(a, b) = \neg b$, * is $\leftrightarrow$

```

```
Example

• $f(a, b) = a \rightarrow b$, $f'(a, b) = \neg b$, * is $\leftrightarrow$

```

```
Example

• $f(a, b) = a \rightarrow b$, $f'(a, b) = \neg b$, * is $\leftrightarrow$

```

```
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 \( g, g', * \)
  - 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 \textit{(BDD-based)} model checking
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 $\phi$
• Returns: $S_\phi$ - the set of states satisfying $\phi$

$M$ 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 $p \in AP$, 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:
\[ \text{EX } f_1(V) = \exists V' \left[ f_1(V') \land R(V, V') \right] \]

• This operation is also called \textbf{pre-image}

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

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

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

\begin{align*}
Q(V) &:= \text{emptyset}; \quad Q'(V) := g(V) ; \\
\text{while } Q(V) \neq Q'(V) \text{ do} & \\
& \quad Q(V) := Q'(V) ; \\
& \quad Q'(V) := Q(V) \lor \text{EX} (Q(V)) \\
\text{end while} & \\
\end{align*}

\begin{align*}
f(V) &:= Q(V) ; \quad \text{return}(f(V))
\end{align*}
The algorithm applies
• BDD operations (or $\lor$), and
• comparison $Q(V) \neq Q'(V)$ (easy)

Therefore, this is a symbolic algorithm!
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 $\text{CheckEF} (g_1, g_2)$

\[ Q := \emptyset; \quad Q' := g_2; \]

\begin{algorithmic}
  \While {$Q \neq Q'$}
    \State $Q := Q'$;
    \State $Q' := Q \lor (EX(Q) \land g_1)$
  \EndWhile
  \State $f := Q$; \quad \text{return}(f)
\end{algorithmic}

Least fixpoint
Model Checking \( f = EG \) \( g \)

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

procedure CheckEG \((g)\)

\[
\begin{align*}
Q & := S \; ; \; Q' := g \; ; \\
\text{while } Q \neq Q' \text{ do} \\
& \quad \; Q := Q' ; \\
& \quad \; Q' := Q \wedge \text{EX} (Q) \\
\text{end while} \\
f & := Q ; \; \text{return} (f )
\end{align*}
\]
Example: \( f = EG g \)