Ordered Binary Decision Diagrams

Altin Qeriqi, altinq@ifi.uio.no

University of Oslo

2015
Problem with CTL

In CTL, each state has an explicit representation by the predecessor and successor list.

- State space can be too big (may be exponential)
  - Not enough memory
  - Takes too much time to explore all states
Tackling the state explosion problem

Abstracting the model

CTL procedure can be reformulated in a symbolic way

▶ Sets of states and sets of transitions are manipulated, not single states and transitions
▶ Natural for CTL since it relies on satisfaction sets for subformulae
  ▶ \( sat(\phi) \): the set of states where \( \phi \) holds
▶ Much larger systems can be verified
Binary Decision Diagrams

An efficient way of representing sets of states and transitions

Boolean functions (switching functions) are represented using BDDs (Binary decision diagrams). They are a compact representation for many switching functions in practical applications. They can easily be manipulated.

BDDs are directed, acyclic graphs with

- Inner nodes where each inner node $v$
  - Is labeled with a boolean variable $\text{var}$, e.g., $\text{var}(v) = x$
  - Have two successor/child nodes:
    - $\text{high}(v)$: where $\text{var}(v) = x$ and $x = 1$
    - $\text{low}(v)$: where $\text{var}(v) = x$ and $x = 0$
- Leaf nodes, aka. terminals or drains, where each drain $v$ is
  - Labeled by a value $\text{val}$ which can be either 1 or 0.
Binary Decision Diagram example

Figure: BDD for $z_1 \land (\neg z_2 \lor z_3)$

The assignment $z_1 = 1$, $z_2 = 0$, $z_3 = 1$ makes the formulae true.
Binary Decision Diagrams

We begin at the root node and follow one of the lines from there till we reach a drain (leaf node)

- If the boolean variable at the current node is 1, follow the solid line. If it is 0, follow the dashed line. Repeat until we are at a drain.
- The truth value of the formula is the boolean value of the drain.
Ordered Binary Decision Diagrams

Ordered binary decision diagrams (OBDDs) are just like BDDs but with a defined variable ordering. That is, a tuple $\varphi = (z_1, \ldots, z_m)$.

- Let a node $u$ labeled $z_i$ have two two child nodes $v$ and $w$.
- The variable $v$ must be labeled by a variable $z_j$ such that $z_i < \varphi z_j$, that is, $z_j$ must come later in the ordered list of variables in $\varphi$. The same thing must be true for the other child node ($w$).

The variable ordering makes it possible to simplify (reduce) OBDDs.
Redundancy in OBDDs

Can remove these inner nodes and merge the leaf node. See next slide
Reduced OBBD
Reduced OBDDs (ROBDDs)

We can reduce the size of OBDDs by using two rules:

**Elimination rule** If $v$ is an inner node and both of its children points to the same node $w$, then remove $v$ and redirect all incoming edges to $w$.

**Isomorphism rule** If the nodes $v$ and $w$ are terminals and if they have the same value, then remove node $v$ and redirect all incoming edges to node $w$. If $v$ and $w$ are inner nodes, they are labeled by the same boolean variable and their children are the same, then remove node $v$ and redirect all incoming edges to $w$. 
Reduction examples 1

Figure: Reduction using the elimination rule

Figure: Reduction using the isomorphism rule for drains
Reduction examples 2

Figure: Reduction using the isomorphism rule for inner nodes
More about ROBDDs

The advantage of an ROBDD is that it is canonical (unique) for a particular function and variable order.

Depending on the algorithms used, boolean connective can be realized in time linear to the size of input OBDD and equivalence checking can even be performed in constant time.

The reduction rules are

▶ Sound. They do not affect the semantics.
▶ Complete, provided we apply the rules in a bottom-up fashion. We start from the drains and for each level upwards, we apply one of the reduction rules on the inner nodes.
The problem with variable ordering - 1

Varying the variable ordering can lead to different ROBDDs

Figure: ROBDD for the function $f = (z_1 \land y_1) \lor (z_2 \land y_2) \lor (z_3 \land y_3)$ for the variable ordering $\wp = (z_1, y_1, z_2, y_2, z_3, y_3)$
The problem with variable ordering - 2

Figure: ROBDD for the function $f = (z_1 \land y_1) \lor (z_2 \land y_2) \lor (z_3 \land y_3)$ for the variable ordering $\varphi = (z_1, z_2, z_3, y_1, y_2, y_3)$
The efficiency of OBDD-based model checking techniques relies on the use of algorithms that improves the ordering of variables.

Finding and optimal ordering is known to be NP-hard. There are several heuristics for improving the current ordering.
Logical operations on OBDDs

Logical negation (\(\neg\))

- Replace the value of each leaf node by its negation

All 16 logical operations can be applied on boolean functions using the Apply algorithm.

Restriction of the variable \(x_i\) to a constant \(b\):

- \(f|_{x_i \leftarrow b}(x_1, \ldots x_n) = f(x_1, x_2, \ldots, x_{i-1}, b, x_{i+1}, \ldots, x_n)\)
- \(f|_{x_i \leftarrow 1}\): positive Shannon cofactor of \(f\) for \(x_i\)
- \(f|_{x_i \leftarrow 0}\): negative Shannon cofactor of \(f\) for \(x_i\)
- To compute the new OBDD:
  - We traverse the tree in a depth-first manner
  - All incoming edges to \(v\), s.t. \(\text{var}(v) = x_i\) should be redirected to \(\text{low}(v)\) if \(b = 0\) or \(\text{high}(v)\) if \(b = 1\)
  - Reduce the OBDD
Apply algorithm

Shannon expansion:

\[ f = (\neg x \land f|_{x \leftarrow 0}) \lor (x \land f|_{x \leftarrow 1}) \]

- Allows us to split a problem into two subproblems

Using the *Apply* algorithm to solve all 16 logical operations. Let

- \( \bullet \) be a two-argument logical operation (and, or, xor etc.)
- \( f \) and \( f' \) be two boolean functions
- \( v \) and \( v' \) be the OBDDs roots for \( f \) and \( f' \)
- \( \text{var}(v) = x \) and \( \text{var}(v') = x' \)

If both \( v \) and \( v' \) are drains:

- \( f \bullet f' = \text{val}(v) \bullet \text{val}(v') \)
Apply algorithm (cont.)

If $x = x'$:

- Recursively solve the two subproblems:
  
  \[
  f \bullet f' = (\neg x \land (f|_{x \leftarrow 0} \bullet f'|_{x \leftarrow 0})) \lor (x \land (f|_{x \leftarrow 1} \bullet f'|_{x \leftarrow 1}))
  \]

- The root of this new OBDD will be a new node $w$ such that
  
  - $\text{var}(w) = x$
  - $\text{low}(w)$ will be OBDD for $(f|_{x \leftarrow 0} \bullet f'|_{x \leftarrow 0})$
  - $\text{high}(w)$ will be OBDD for $(f|_{x \leftarrow 1} \bullet f'|_{x \leftarrow 1})$

If $x < x'$ ($x = x_i$ and $x' = x_j$ where $i < j$):

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

- Similar for $x > x'$

Algorithm is polynomial with dynamic programming
Boolean quantification

If $f$ is a function, $x$ is a variable, then

- $\exists x. f = (f|_{x=0}) \lor (f|_{x=1})$
- $\forall x. f = (f|_{x=0}) \land (f|_{x=1})$

We need to compute the OBDD for both subproblems using the Restrict algorithm:

- $f|_{x=0}$: For each node $v$ where $\text{var}(v) = x$
  - Incoming edges are redirected to $\text{low}(v)$
  - Remove node $v$

- $f|_{x=1}$: For each node $v$ where $\text{var}(v) = x$
  - Incoming edges are redirected to $\text{high}(v)$
  - Remove node $v$
References I

