CS181u Applied Logic & Automated Reasoning

Lecture 5

Binary Decision Diagrams
BDD operations
A Binary Decision Diagram (BDD) is a data structure for representing the truth values of formulas in propositional logic.

Example: consider the formula $\neg x \land \neg y$ and the truth table.

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>$\neg x \land \neg y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F$</td>
<td>$F$</td>
<td>$T$</td>
</tr>
<tr>
<td>$F$</td>
<td>$T$</td>
<td>$F$</td>
</tr>
<tr>
<td>$T$</td>
<td>$F$</td>
<td>$F$</td>
</tr>
<tr>
<td>$T$</td>
<td>$T$</td>
<td>$F$</td>
</tr>
</tbody>
</table>

The same information can be encoded in a decision tree.
A Binary Decision Diagram (BDD) is a data structure for representing the truth values of formulas in propositional logic.

Example: consider the formula $\neg x \land \neg y$ and the truth table.

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>$\neg x \land \neg y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F$</td>
<td>$F$</td>
<td>$T$</td>
</tr>
<tr>
<td>$F$</td>
<td>$T$</td>
<td>$F$</td>
</tr>
<tr>
<td>$T$</td>
<td>$F$</td>
<td>$F$</td>
</tr>
<tr>
<td>$T$</td>
<td>$T$</td>
<td>$F$</td>
</tr>
</tbody>
</table>

The same information can be encoded in a decision tree.
A **Binary Decision Diagram (BDD)** is a data structure for representing the truth values of formulas in propositional logic.

Example: consider the formula \( \neg x \land \neg y \) and the truth table.

<table>
<thead>
<tr>
<th>( x )</th>
<th>( y )</th>
<th>( \neg x \land \neg y )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( F )</td>
<td>( F )</td>
<td>( T )</td>
</tr>
<tr>
<td>( F )</td>
<td>( T )</td>
<td>( F )</td>
</tr>
<tr>
<td>( T )</td>
<td>( F )</td>
<td>( F )</td>
</tr>
<tr>
<td>( T )</td>
<td>( T )</td>
<td>( F )</td>
</tr>
</tbody>
</table>

The same information can be encoded in a decision tree.
A Binary Decision Diagram (BDD) is a data structure for representing the truth values of formulas in propositional logic.

Example: consider the formula $\neg x \land \neg y$ and the truth table.

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>$\neg x \land \neg y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F$</td>
<td>$F$</td>
<td>$T$</td>
</tr>
<tr>
<td>$F$</td>
<td>$T$</td>
<td>$F$</td>
</tr>
<tr>
<td>$T$</td>
<td>$F$</td>
<td>$F$</td>
</tr>
<tr>
<td>$T$</td>
<td>$T$</td>
<td>$T$</td>
</tr>
</tbody>
</table>

The same information can be encoded in a decision tree.
A Binary Decision Diagram (BDD) is a data structure for representing the truth values of formulas in propositional logic.

Example: consider the formula \( \neg x \land \neg y \) and the truth table.

<table>
<thead>
<tr>
<th>( x )</th>
<th>( y )</th>
<th>( \neg x \land \neg y )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T )</td>
<td>( T )</td>
<td>( T )</td>
</tr>
<tr>
<td>( T )</td>
<td>( T )</td>
<td>( F )</td>
</tr>
<tr>
<td>( F )</td>
<td>( T )</td>
<td>( F )</td>
</tr>
<tr>
<td>( F )</td>
<td>( F )</td>
<td>( T )</td>
</tr>
</tbody>
</table>

The same information can be encoded in a decision tree.
We can make the decision diagram more compact.
We can make the decision diagram more compact.

Redundant terminal nodes
We can make the decision diagram more compact.

Redundant terminal nodes

Merge redundant nodes
We can make the decision diagram more compact.

Redundant terminal nodes

Merge redundant nodes

Redundant branches
We can make the decision diagram more compact.

Redundant terminal nodes

Merge redundant nodes

Redundant branches

Remove redundant branching

A more compact representation of \( \neg x \land \neg y \)
Reduction Rule 1: Merge duplicated terminal nodes.
Reduction Rule 2: Remove redundant tests.
Reduction Rule 3: Remove duplicate sub-BDDs.

\[ BDD_1 \equiv BDD_2 \]
Reduction Rule 3: Remove duplicate sub-BDDs.

\[ BDD_1 \equiv BDD_2 \]

NOTE: They can be structurally identical, even if they overlap.
Example: Reduce the given BDD for the formula \((x \land y) \lor (\neg y \land z)\) to an ROBDD.
Example: Reduce the given BDD for the formula \((x \land y) \lor (\neg y \land z)\) to an ROBDD.

Merge duplicate terminals
Example: Reduce the given BDD for the formula \((x \land y) \lor (\neg y \land z)\) to an ROBDD.

Merge duplicate terminals
Example: Reduce the given BDD for the formula \((x \land y) \lor (\neg y \land z)\) to an ROBDD.

Delete redundant tests
Example: Reduce the given BDD for the formula \((x \land y) \lor (\neg y \land z)\) to an ROBDD.

Delete redundant tests
Example: Reduce the given BDD for the formula \((x \land y) \lor (\neg y \land z)\) to an ROBDD.

Merge more duplicate terminals
Example: Reduce the given BDD for the formula \((x \land y) \lor (\neg y \land z)\) to an ROBDD.

Merge more duplicate terminals
Example: Reduce the given BDD for the formula \((x \land y) \lor (\neg y \land z)\) to an ROBDD.

Merge more duplicate terminals
Example: Reduce the given BDD for the formula \((x \land y) \lor (\neg y \land z)\) to an ROBDD.

Redraw to “untangle”
Example: Reduce the given BDD for the formula \((x \land y) \lor (\neg y \land z)\) to an ROBDD.

Redraw to "untangle"
Example: Reduce the given BDD for the formula $(x \land y) \lor (\neg y \land z)$ to an ROBDD.

Redraw to “untangle”
Example: Reduce the given BDD for the formula \((x \land y) \lor (\neg y \land z)\) to an ROBDD.

Remove duplicated sub-BDD
Example: Reduce the given BDD for the formula \((x \land y) \lor (\neg y \land z)\) to an ROBDD.

Remove duplicated sub-BDD
Example: Reduce the given BDD for the formula \((x \land y) \lor (\neg y \land z)\) to an ROBDD.

No more reduction possible
Important properties of BDDs

Ordered BDD (OBDD): variables are checked in a given order. E.g $x > y > z$.

Reduced OBDD (ROBDD): Cannot be reduced any further.

Theorem: ROBDDs are unique for a given ordering.
Important properties of BDDs

**ROBDD size** is sensitive to variable ordering!

Two different ROBDDs for the formula

\[ x' \leftrightarrow x \land y' \leftrightarrow y \]

Variable order: \( x, x', y, y' \)  
Variable order: \( x, y, x', y' \)
BDDs via Shannon Expansion

\[ f \equiv (x = F) \land f[F/x] \lor (x = T) \land f[T/x] \]

For now, recursively compute

In class example: \( f \equiv \neg x \lor \neg y \)
BDDs via Shannon Expansion

\[ f \equiv (x = F) \land f[F/x] \lor (x = T) \land f[T/x] \]

For now, recursively compute

\[ f \equiv (x_1 \lor x_2) \land x_3 \]

\[ g \equiv (x_1 \land \neg x_2) \]
BDDs via Shannon Expansion

\[ f \equiv (x = F) \land f[F/x] \lor (x = T) \land f[T/x] \]

For now, recursively compute

\[ f \equiv (x_1 \lor x_2) \land x_3 \]

\[ g \equiv (x_1 \land \neg x_2) \]
Symbolic Model Checking: \(10^{20}\) States and Beyond

J. R. Burch    E. M. Clarke    K. L. McMillan*
School of Computer Science
Carnegie Mellon University

D. L. Dill    L. J. Hwang
Stanford University

Abstract

Many different methods have been devised for automatically verifying finite state systems by examining state-graph models of system behavior. These methods all depend on decision procedures that explicitly represent the state space using a list or a table that grows in proportion to the number of states. We describe a general method that represents the state space symbolically instead of explicitly. The generality of our method comes from using a dialect of the Mu-Calculus as the primary specification language. We describe a model checking algorithm for Mu-Calculus formulas that uses Bryant’s *Binary Decision Diagrams* (1986) to represent relations and formulas. We then show how our new Mu-Calculus model checking algorithm can be used to derive efficient decision procedures for CTL model checking, satisfiability of linear-time temporal logic formulas, strong and weak observational equivalence of finite transition systems, and language containment for finite \(\omega\)-automata. The fixed point computations for each decision procedure are sometimes complex, but can be concisely expressed in the Mu-Calculus. We illustrate the practicality of our approach to symbolic model checking by discussing how it can be used to verify a simple synchronous pipeline circuit.
Useful things to do with BDDs

Imagine that you have two Boolean logic formulas $f$ and $g$, and you also have BDDs for each of them, say $B_f$ and $B_g$. How would you accomplish the following?

- Test if $f$ is a tautology
- Test if $f$ is a satisfiable
- Test if $f \equiv g$
- Compute the BDD for $\neg f$
- Compute the BDD for $f \land g$
- Compute the BDD for $f \lor g$
Given $f$ and $g$, let’s compute $B_{f \star g}$ from $B_f$ and $B_g$, where $\star \in \{\land, \lor\}$.

Let $\text{apply}(\star, B_f, B_g)$ be the function that computes $B_{f \star g}$.

We will compute $\text{apply}(\star, B_f, B_g)$ recursively.

What are the base cases?

$$\text{apply}(\star, t_1, t_2) = t_1 \star t_2$$
Binary ops on $B_f$ and $B_g$: A bunch of recursive cases:
Binary ops on $B_f$ and $B_g$: A bunch of recursive cases:

$$\text{apply}(\star, \begin{array}{c} x \\ F \\ B_f[F/x] \\ T \\ B_f[T/x] \end{array}, \begin{array}{c} t \end{array}) =$$
Binary ops on $B_f$ and $B_g$: A bunch of recursive cases:

$$
\text{apply}(\star, \frac{B_f[F/x]}{F}, \frac{B_f[T/x]}{T}, t) = \\
$$

Rule 1a
Binary ops on $B_f$ and $B_g$: A bunch of recursive cases:

\[
\text{apply}(\star, [t], \x) = \begin{cases} 
B_g[F/x] & \text{if } x = F \\
B_g[T/x] & \text{if } x = T
\end{cases}
\]
Binary ops on $B_f$ and $B_g$: A bunch of recursive cases:

$$\text{apply}(\star, \boxed{t}, B_g[F/x], B_g[T/x])$$

Rule 1b
Binary ops on $B_f$ and $B_g$: A bunch of recursive cases:

$$\text{apply}(\star, \quad B_f[F/x], B_f[T/x], B_g[F/x], B_g[T/x] \quad ) =$$
Binary ops on $B_f$ and $B_g$: A bunch of recursive cases:

\[
\text{apply}(\star, B_f[F/x], B_f[T/x], B_g[F/x], B_g[T/x]) =
\]

Rule 2
Binary ops on $B_f$ and $B_g$: A bunch of recursive cases:

$$\text{apply}(\star, \begin{array}{c} x \\ B_f[F/x] \\ \text{different variables} \\ B_f[T/x] \end{array}, \begin{array}{c} y \\ B_g[F/y] \\ B_g[T/y] \end{array}) =$$
Binary ops on $B_f$ and $B_g$: A bunch of recursive cases:

apply($\star$, $B_f[F/x]$, $B_f[T/x]$, $B_g[F/y]$, $B_g[T/y]$) =

If $x > y$ in the BDD ordering:

apply($\star$, $B_f[F/x]$, $B_g[F/y]$, $B_g[T/y]$) apply($\star$, $B_f[T/x]$, $B_g[F/y]$, $B_g[T/y]$)

Rule 3a
Binary ops on $B_f$ and $B_g$: A bunch of recursive cases:

$\text{apply}(\star, \quad B_f[F/x], \quad B_f[T/x], \quad B_g[F/y], \quad B_g[T/y]) = \quad \text{different variables}$
Binary ops on $B_f$ and $B_g$: A bunch of recursive cases:

$$\text{apply}(\star, B_f[x/F], B_f[x/T], B_g[y/F], B_g[y/T]) =$$

If $y > x$ in the BDD ordering:

$$\text{apply}(\star, B_f[x/F], B_f[x/T], B_g[y/F]) \quad \text{apply}(\star, B_f[x/F], B_f[x/T], B_g[y/F])$$

Rule 3b
Binary ops on $B_f$ and $B_g$: 
Binary ops on $B_f$ and $B_g$:

Compute $B_{f \lor g}$ using $B_f$ and $B_g$

\[ f \equiv (x_1 \lor x_2) \land x_3 \]

\[ g \equiv (x_1 \land \neg x_2) \]
Binary ops on $B_f$ and $B_g$: 
Binary ops on $B_f$ and $B_g$:

Compute $B_{f \lor g}$ using $B_f$ and $B_g$

\[ f \equiv (x_1 \lor x_2) \land x_3 \quad g \equiv (x_1 \land \neg x_2) \]
Binary ops on $B_f$ and $B_g$:

Compute $B_{f\lor g}$ using $B_f$ and $B_g$

$f \equiv (x_1 \lor x_2) \land x_3$

$g \equiv (x_1 \land \neg x_2)$
Binary ops on $B_f$ and $B_g$:

$$f \equiv (x_1 \lor x_2) \land x_3$$

$$g \equiv (x_1 \land \neg x_2)$$

Compute $B_{f \lor g}$ using $B_f$ and $B_g$
Binary ops on $B_f$ and $B_g$:

Compute $B_{f\lor g}$ using $B_f$ and $B_g$

\[ f \equiv (x_1 \lor x_2) \land x_3 \]
\[ g \equiv (x_1 \land \neg x_2) \]
Binary ops on $B_f$ and $B_g$:

Compute $B_{f\lor g}$ using $B_f$ and $B_g$

$$f \equiv (x_1 \lor x_2) \land x_3$$

$$g \equiv (x_1 \land \neg x_2)$$
Binary ops on $B_f$ and $B_g$:

Compute $B_{f \lor g}$ using $B_f$ and $B_g$

\[ f \equiv (x_1 \lor x_2) \land x_3 \]
\[ g \equiv (x_1 \land \neg x_2) \]
Binary ops on $B_f$ and $B_g$:

Compute $B_{f \lor g}$ using $B_f$ and $B_g$

\[ f \equiv (x_1 \lor x_2) \land x_3 \]

\[ g \equiv (x_1 \land \neg x_2) \]
**Binary ops on $B_f$ and $B_g$:**

Compute $B_{f \lor g}$ using $B_f$ and $B_g$

- $f \equiv (x_1 \lor x_2) \land x_3$
- $g \equiv (x_1 \land \neg x_2)$

Rule 1a

Base Case

Rule 3b
Binary ops on $B_f$ and $B_g$:

Compute $B_{f \lor g}$ using $B_f$ and $B_g$

$f \equiv (x_1 \lor x_2) \land x_3$

g $\equiv (x_1 \land \neg x_2)$
Binary ops on $B_f$ and $B_g$:

Compute $B_{f \lor g}$ using $B_f$ and $B_g$

\[
f \equiv (x_1 \lor x_2) \land x_3
g \equiv (x_1 \land \neg x_2)
\]
Binary ops on $B_f$ and $B_g$:

Compute $B_{f \lor g}$ using $B_f$ and $B_g$

\[
f \equiv (x_1 \lor x_2) \land x_3
\]

\[
g \equiv (x_1 \land \neg x_2)
\]
Binary ops on $B_f$ and $B_g$:

Compute $B_{f \lor g}$ using $B_f$ and $B_g$

\[ f \equiv (x_1 \lor x_2) \land x_3 \]

\[ g \equiv (x_1 \land \neg x_2) \]
Binary ops on $B_f$ and $B_g$:

Compute $B_f \lor g$ using $B_f$ and $B_g$

\[ f \equiv (x_1 \lor x_2) \land x_3 \quad g \equiv (x_1 \land \neg x_2) \]
Binary ops on $B_f$ and $B_g$:

Compute $B_{f \lor g}$ using $B_f$ and $B_g$

$$f \equiv (x_1 \lor x_2) \land x_3$$

$$g \equiv (x_1 \land \neg x_2)$$
Binary ops on $B_f$ and $B_g$:

Compute $B_{f \lor g}$ using $B_f$ and $B_g$

\[ f \equiv (x_1 \lor x_2) \land x_3 \]
\[ g \equiv (x_1 \land \neg x_2) \]

```
Compute $B_{f \lor g}$ using $B_f$ and $B_g$

\[ f \equiv (x_1 \lor x_2) \land x_3 \]
\[ g \equiv (x_1 \land \neg x_2) \]
```
Binary ops on $B_f$ and $B_g$:

Compute $B_{f\lor g}$ using $B_f$ and $B_g$

$$f \equiv (x_1 \lor x_2) \land x_3$$

$$g \equiv (x_1 \land \neg x_2)$$
Binary ops on $B_f$ and $B_g$:

Compute $B_{f\lor g}$ using $B_f$ and $B_g$

\[ f \equiv (x_1 \lor x_2) \land x_3 \]

\[ g \equiv (x_1 \land \neg x_2) \]
Binary ops on $B_f$ and $B_g$:

Compute $B_{f \lor g}$ using $B_f$ and $B_g$

\[ f \equiv (x_1 \lor x_2) \land x_3 \]
\[ g \equiv (x_1 \land \neg x_2) \]
Binary ops on $B_f$ and $B_g$:

Compute $B_{f \lor g}$ using $B_f$ and $B_g$

$$f \equiv (x_1 \lor x_2) \land x_3$$

$$g \equiv (x_1 \land \neg x_2)$$
Binary ops on $B_f$ and $B_g$:

Compute $B_{f \lor g}$ using $B_f$ and $B_g$

\[ f \equiv (x_1 \lor x_2) \land x_3 \]
\[ g \equiv (x_1 \land \neg x_2) \]
Binary ops on $B_f$ and $B_g$:

Compute $B_{f \lor g}$ using $B_f$ and $B_g$

\[ f \equiv (x_1 \lor x_2) \land x_3 \]
\[ g \equiv (x_1 \land \neg x_2) \]
Binary ops on $B_f$ and $B_g$:

Compute $B_{f\lor g}$ using $B_f$ and $B_g$

\[ f \equiv (x_1 \lor x_2) \land x_3 \]

\[ g \equiv (x_1 \land \neg x_2) \]
A transition system $\mathcal{M}$ can be specified by listing out all of the pieces.

States: $S = \{0, 1, 2, 3\}$

Initial States: $I = \{0\}$

Transitions:

$R = \{(0, 1), (0, 2), (1, 3), (2, 3), (1, 0), (2, 0), (3, 1), (3, 2)\}$
Symbolic Representation

Represent $M$ using Boolean logic.
Symbolic Representation

Represent $M$ using Boolean logic.
Symbolic Representation

Represent $M$ using Boolean logic.

$$\begin{array}{c|c|c}
\text{States} & \text{binary} & \\
\hline
0 & 0 & 0 \\
1 & 0 & 1 \\
2 & 1 & 0 \\
3 & 1 & 1 \\
\end{array}$$
Symbolic Representation

Represent $M$ using Boolean logic.

States | binary | truth values |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$x$</td>
<td>$y$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Symbolic Representation

Represent $\mathcal{M}$ using Boolean logic.

### Boolean State Variables

$$V = \{x, y\}$$

### States

<table>
<thead>
<tr>
<th>States</th>
<th>binary</th>
<th>truth values</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$x$</td>
<td>$y$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
# Symbolic Representation

Represent $M$ using Boolean logic.

## Boolean state variables

$V = \{x, y\}$

## States

<table>
<thead>
<tr>
<th>States</th>
<th>binary</th>
<th>truth values</th>
<th>Boolean formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 0</td>
<td>F F</td>
<td>$\neg x \land \neg y$</td>
</tr>
<tr>
<td>1</td>
<td>0 1</td>
<td>F T</td>
<td>$\neg x \land y$</td>
</tr>
<tr>
<td>2</td>
<td>1 0</td>
<td>T F</td>
<td>$x \land \neg y$</td>
</tr>
<tr>
<td>3</td>
<td>1 1</td>
<td>T T</td>
<td>$x \land y$</td>
</tr>
</tbody>
</table>
Symbolic Representation

Represent $M$ using Boolean logic.
Symbolic Representation

Represent $\mathcal{M}$ using Boolean logic.

Transitions:
Let the “next” state variables be $V’ = \{x’, y’\}$

```
V’ = \{x’, y’\}
```
Represent $M$ using Boolean logic.

Transitions:
Let the “next” state variables be $V' = \{x', y'\}$

$$R \equiv (x' = x \land y' = \neg y) \lor (x' = \neg x \land y' = y)$$
Symbolic Representation

Represent $\mathcal{M}$ using Boolean logic.

Transitions:
Let the “next” state variables be $V' = \{x', y'\}$

$$R \equiv (x' = x \land y' = \neg y) \lor (x' = \neg x \land y' = y)$$

“we can get from one state to the next by keeping one variable the same and negating the other”
Represent $\mathcal{M}$ using Boolean logic.

**Transitions:**
Let the “next” state variables be $V' = \{x', y'\}$

\[ R \equiv (x' = x \land y' = \neg y) \lor (x' = \neg x \land y' = y) \]

Explicit transitions:
- $(0, 1)$
- $(2, 3)$
- $(1, 3)$
- $(0, 2)$
- $(1, 0)$
- $(3, 2)$
- $(3, 1)$
- $(2, 0)$

“we can get from one state to the next by keeping one variable the same and negating the other”
Symbolic Representation

\[ R \equiv (x' = x \land y' = \neg y) \lor (x' = \neg x \land y' = y) \]

BDD for R