Binary Decision Diagrams
BDDs
Introduction

❖ BDDs were introduced by Randal E. Bryant:


❖ BDDs provide a (usually) very compact and canonical representation of Boolean functions (i.e., functions of the form \(\{0, 1\}^k \rightarrow \{0, 1\}, k \geq 0\)), corresponding to propositional formulae (possibly representing finite sets or relations).

❖ BDDs have a form of rooted, directed, connected, acyclic graph, which consists of internal Boolean decision nodes and terminal Boolean result nodes.

❖ BDDs may be viewed to arise from Boolean decision trees by removing redundancies from them (merging isomorphic sub-trees, removing useless nodes with isomorphic children).

❖ Operations on BDDs are done without uncompressing the represented objects.

❖ Applications: synthesis of circuits, symbolic verification, fault tree analysis, decision procedures, ...
From Formulae to BDDs

❖ The propositional formula $\varphi = (a \land b \land c) \lor (a \land b \land \neg c)$ may be represented by:

(a) its truth table

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>$\varphi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
From Formulae to BDDs

The propositional formula \( \varphi = (a \land b \land c) \lor (a \land b \land \neg c) \) may be represented by:

(a) its truth table

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>( \varphi )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

(b) a decision tree
(a not reduced BDD)
The propositional formula $\varphi = (a \land b \land c) \lor (a \land b \land \neg c)$ may be represented by:

(a) its truth table

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>$\varphi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

(b) a decision tree (a not reduced BDD)

(c) a (reduced) BDD
A Formal Definition of BDDs

A BDD $G$ over a set of Boolean variables $Var$ is defined as a 7-tuple $G = (N, T, \text{var}, \text{low}, \text{high}, \text{root}, \text{val})$ where:

- $N$ is a finite set of non-terminal (internal) nodes,
- $T$ is a finite set of terminal nodes (leaves), $N \cap T = \emptyset$. 
A Formal Definition of BDDs

A BDD $G$ over a set of Boolean variables $Var$ is defined as a 7-tuple $G = (N, T, \text{var}, \text{low}, \text{high}, \text{root}, \text{val})$ where:

- $N$ is a finite set of non-terminal (internal) nodes,
  $T$ is a finite set of terminal nodes (leaves), $N \cap T = \emptyset$.
- $\text{var} : N \rightarrow Var$ labels the internal nodes by variables.
A Formal Definition of BDDs

A BDD $G$ over a set of Boolean variables $Var$ is defined as a 7-tuple $G = (N, T, var, low, high, root, val)$ where:

- $N$ is a finite set of non-terminal (internal) nodes,
  $T$ is a finite set of terminal nodes (leaves), $N \cap T = \emptyset$.
- $var : N \rightarrow Var$ labels the internal nodes by variables.
- $low, high : N \rightarrow N \cup T$ define the low and high successors of internal nodes $n \in N$, for the value of $var(n)$ being 0 or 1, respectively.
  - It is required that $G$ is acyclic, i.e., $\neg \exists n \in N. n(low \cup high)^+ n$. 

Introduction – p.5/21
A Formal Definition of BDDs

A BDD $G$ over a set of Boolean variables $Var$ is defined as a 7-tuple $G = (N, T, var, low, high, root, val)$ where:

- $N$ is a finite set of non-terminal (internal) nodes,
  $T$ is a finite set of terminal nodes (leaves), $N \cap T = \emptyset$.
- $var : N \rightarrow Var$ labels the internal nodes by variables.
- $low, high : N \rightarrow N \cup T$ define the low and high successors of internal nodes $n \in N$, for the value of $var(n)$ being 0 or 1, respectively.
  - It is required that $G$ is acyclic, i.e., $\neg \exists n \in N. n(low \cup high)^+ n$.
- $root \in N \cup T$ is the root node such that $\forall n \in (N \cup T) \setminus \{root\}. root(low \cup high)^+ n$
  (note that $root$ cannot have any incoming edges without breaking the acyclicity requirement).
A Formal Definition of BDDs

A BDD $G$ over a set of Boolean variables $Var$ is defined as a 7-tuple $G = (N, T, \text{var}, \text{low}, \text{high}, \text{root}, \text{val})$ where:

- $N$ is a finite set of non-terminal (internal) nodes, $T$ is a finite set of terminal nodes (leaves), $N \cap T = \emptyset$.
- $\text{var} : N \rightarrow Var$ labels the internal nodes by variables.
- $\text{low}, \text{high} : N \rightarrow N \cup T$ define the low and high successors of internal nodes $n \in N$, for the value of $\text{var}(n)$ being 0 or 1, respectively.
  - It is required that $G$ is acyclic, i.e., $\neg \exists n \in N. n(\text{low} \cup \text{high})^+ n$.
- $\text{root} \in N \cup T$ is the root node such that $\forall n \in (N \cup T) \setminus \{\text{root}\}. \text{root}(\text{low} \cup \text{high})^+ n$ (note that $\text{root}$ cannot have any incoming edges without breaking the acyclicity requirement).
- $\text{val} : T \rightarrow \{0, 1\}$ labels the leaves by their values.
A Formal Definition of BDDs

❖ A BDD $G$ over a set of Boolean variables $Var$ is defined as a 7-tuple $G = (N, T, \text{var}, \text{low}, \text{high}, \text{root}, \text{val})$ where:

- $N$ is a finite set of non-terminal (internal) nodes,
  $T$ is a finite set of terminal nodes (leaves), $N \cap T = \emptyset$.
- $\text{var} : N \rightarrow Var$ labels the internal nodes by variables.
- $\text{low}, \text{high} : N \rightarrow N \cup T$ define the low and high successors of internal nodes $n \in N$, for the value of $\text{var}(n)$ being 0 or 1, respectively.
  - It is required that $G$ is acyclic, i.e., $\neg \exists n \in N. n(\text{low} \cup \text{high})^+ n$.
- $\text{root} \in N \cup T$ is the root node such that $\forall n \in (N \cup T) \setminus \{\text{root}\}. \text{root}(\text{low} \cup \text{high})^+ n$ (note that $\text{root}$ cannot have any incoming edges without breaking the acyclicity requirement).
- $\text{val} : T \rightarrow \{0, 1\}$ labels the leaves by their values.

❖ For convenience, we often assume that $Var$ is indexed by some bijection $f : I \leftrightarrow Var$ over the set of indices $I = \{1, \ldots, n\}$, yielding an indexed family of variables denoted $\{v_i\}_{i \in I}$. 

Introduction – p.5/21
Functions Represented by BDDs

❖ A node $x \in N \cup T$ of a BDD $G = (N, T, \text{var}, \text{low}, \text{high}, \text{root}, \text{val})$ over an indexed family of variables $\{v_i\}_{i \in I}, I = \{1, \ldots, k\}, k \geq 0$, represents the Boolean function $f_x : \{0, 1\}^k \rightarrow \{0, 1\}$ defined as follows:

1. If $x \in T$, then $f_x(v_1, \ldots, v_k) = \text{val}(x)$.
2. If $x \in N$ and $\text{var}(x) = v_i$ for some $i \in I$, then
   $f_x(v_1, \ldots, v_k) = (\neg v_i \land f_{\text{low}(x)}(v_1, \ldots, v_k)) \lor (v_i \land f_{\text{high}(x)}(v_1, \ldots, v_k))$.

❖ $G$ itself represents the function $f_{\text{root}}(v_1, \ldots, v_k)$. 

Functions Represented by BDDs

❖ A node \( x \in N \cup T \) of a BDD \( G = (N, T, \var, \text{low}, \text{high}, \text{root}, \text{val}) \) over an indexed family of variables \( \{v_i\}_{i \in I}, \ I = \{1, \ldots, k\}, \ k \geq 0 \), represents the Boolean function \( f_x : \{0, 1\}^k \rightarrow \{0, 1\} \) defined as follows:

1. If \( x \in T \), then \( f_x(v_1, \ldots, v_k) = \text{val}(x) \).
2. If \( x \in N \) and \( \var(x) = v_i \) for some \( i \in I \), then
   \[
   f_x(v_1, \ldots, v_k) = (\neg v_i \land f_{\text{low}(x)}(v_1, \ldots, v_k)) \lor (v_i \land f_{\text{high}(x)}(v_1, \ldots, v_k)).
   \]

❖ \( G \) itself represents the function \( f_{\text{root}}(v_1, \ldots, v_k) \).

❖ An example:

\[
\begin{align*}
    f_{t_1}(v_1, v_2) &= 0, \quad f_{t_2}(v_1, v_2) = 1, \\
    f_{n_2}(v_1, v_2) &= (\neg v_2 \land f_{t_1}(v_1, v_2)) \lor (v_2 \land f_{t_2}(v_1, v_2)) = v_2, \\
    f_{n_3}(v_1, v_2) &= (\neg v_2 \land f_{t_2}(v_1, v_2)) \lor (v_2 \land f_{t_1}(v_1, v_2)) = \neg v_2, \\
    f_{n_1}(v_1, v_2) &= (\neg v_1 \land v_2) \lor (v_1 \land \neg v_2).
\end{align*}
\]
Reduced BDDs

Two BDDs $G_1 = (N_1, T_1, var_1, low_1, high_1, root_1, val_1)$ and $G_2 = (N_2, T_2, var_2, low_2, high_2, root_2, val_2)$ over the same set of variables are isomorphic iff there exists a bijection $h : N_1 \cup T_1 \leftrightarrow N_2 \cup T_2$ such that:

1. $H(N_1) = N_2$ and $H(T_1) = T_2$ for the pointwise extension $H$ of $h$ to sets of elements.
2. $\forall n \in N_1.\ \ h(low_1(n)) = low_2(h(n)) \land h(high_1(n)) = high_2(h(n)) \land var_1(n) = var_2(h(n))$.
3. $h(root_1) = root_2$.
4. $\forall t \in T_1.\ val_1(t) = val_2(h(t))$. 

Introduction – p.7/21
Reduced BDDs

❖ Two BDDs $G_1 = (N_1, T_1, \text{var}_1, \text{low}_1, \text{high}_1, \text{root}_1, \text{val}_1)$ and $G_2 = (N_2, T_2, \text{var}_2, \text{low}_2, \text{high}_2, \text{root}_2, \text{val}_2)$ over the same set of variables are isomorphic iff there exists a bijection $h : N_1 \cup T_1 \leftrightarrow N_2 \cup T_2$ such that:

1. $H(N_1) = N_2$ and $H(T_1) = T_2$ for the pointwise extension $H$ of $h$ to sets of elements.
2. $\forall n \in N_1.$
   
   \begin{align*}
   h(\text{low}_1(n)) &= \text{low}_2(h(n)) \\
   h(\text{high}_1(n)) &= \text{high}_2(h(n)) \\
   \text{var}_1(n) &= \text{var}_2(h(n)).
   \end{align*}
3. $h(\text{root}_1) = \text{root}_2.$
4. $\forall t \in T_1. \text{val}_1(t) = \text{val}_2(h(t)).$

❖ A BDD $G$ is reduced iff

1. there is no node $n \in N$ such that $\text{low}(n) = \text{high}(n)$ and
2. there are no two nodes $x_1, x_2 \in N \cup T$ such that the BDDs obtained from $G$ by making $x_1$ and $x_2$ the roots and removing their predecessors are isomorphic.
Ordered BDDs

❖ Given some (strict, total) ordering $\prec$ on $Var$, a BDD $G$ is ordered wrt. $\prec$ iff $\forall n \in N$.

1. $\text{low}(n) \in N \implies \text{var}(n) \prec \text{var}(\text{low}(n))$ and
2. $\text{high}(n) \in N \implies \text{var}(n) \prec \text{var}(\text{high}(n))$.

❖ Intuitively, in an ordered BDD, the variables encountered in any path from the root are ordered in an ascending way wrt. $\prec$.

❖ We abbreviate ordered BDDs as OBDDs and reduced OBDDs as ROBDDs.
Ordered BDDs

- Given some (strict, total) ordering \(<\) on \(Var\), a BDD \(G\) is ordered wrt. \(<\) iff \(\forall n \in N\).
  1. \(low(n) \in N \implies var(n) < var(low(n))\) and
  2. \(high(n) \in N \implies var(n) < var(high(n))\).

- Intuitively, in an ordered BDD, the variables encountered in any path from the root are ordered in an ascending way wrt. \(<\).

- We abbreviate ordered BDDs as **OBDDs** and reduced OBDDs as **ROBDDs**.

- **Theorem (canonical representation of Boolean functions by BDDs).** For every Boolean function \(f\) over some set of variables \(Var\) and every variable ordering \(<\) on \(Var\), there is a unique (up to isomorphism) ROBDD (wrt. \(<\) \(G_f\) which represents \(f\).

- **Corollary.** Checking equivalence of the functions represented by two ROBDDs \(G_1\) and \(G_2\) wrt. the same ordering \(<\) amounts to checking isomorphism of \(G_1\) and \(G_2\).
Ordered BDDs

❖ Given some (strict, total) ordering $\prec$ on $\text{Var}$, a BDD $G$ is ordered wrt. $\prec$ iff $\forall n \in N$. 

1. $\text{low}(n) \in N \implies \text{var}(n) \prec \text{var}(\text{low}(n))$ and 
2. $\text{high}(n) \in N \implies \text{var}(n) \prec \text{var}(\text{high}(n))$.

❖ Intuitively, in an ordered BDD, the variables encountered in any path from the root are ordered in an ascending way wrt. $\prec$.

❖ We abbreviate ordered BDDs as OBDDs and reduced OBDDs as ROBDDs.

❖ Theorem (canonical representation of Boolean functions by BDDs). For every Boolean function $f$ over some set of variables $\text{Var}$ and every variable ordering $\prec$ on $\text{Var}$, there is a unique (up to isomorphism) ROBDD (wrt. $\prec$) $G_f$ which represents $f$.

❖ Corollary. Checking equivalence of the functions represented by two ROBDDs $G_1$ and $G_2$ wrt. the same ordering $\prec$ amounts to checking isomorphism of $G_1$ and $G_2$.

❖ Moreover, if several Boolean functions are represented by a generalised BDD with multiple roots, the equivalence checking amounts to checking identity of the roots.
Obtaining ROBDDs from OBDDs

❖ For a fixed ordering \(<\), the ROBDD can be obtained from an OBDD by a procedure denoted \textit{Reduce} which applies the following \textbf{three transformation rules} until no rule is applicable anymore:

- **Rule 1**—remove duplicate leaves: merge all equivalued leaves into a single node, which becomes the target of all the edges leading to the merged nodes.

- **Rule 2**—remove duplicate nonterminals: if there are inner nodes \(n_1, n_2 \in N\) such that \(n_1 \neq n_2\), but \(\text{var}(n_1) = \text{var}(n_2), \text{low}(n_1) = \text{low}(n_2),\) and \(\text{high}(n_1) = \text{high}(n_2)\), then merge \(n_1\) and \(n_2\) into a single node being the target of all the edges coming originally into \(n_1\) and \(n_2\).
Obtaining ROBDDs from OBDDs

- Rule 3—remove redundant nodes: remove inner nodes $n \in N$ with $low(n) = high(n)$ and redirect all edges coming into $n$ to $low(n)$.

An example: the decision tree from Slide 4 (which is an OBDD but not reduced) can be transformed into the BDD from Slide 4 (which is in fact the appropriate ROBDD).
Obtaining ROBDDs from OBDDs

- Rule 3—remove redundant nodes: remove inner nodes $n \in N$ with $\text{low}(n) = \text{high}(n)$ and redirect all edges coming into $n$ to $\text{low}(n)$.

- An example: the decision tree from Slide 4 (which is an OBDD but not reduced) can be transformed into the BDD from Slide 4 (which is in fact the appropriate ROBDD).

- Constant ROBDDs:
  - A propositional formula is not satisfiable iff its ROBDD is isomorphic to the “0” ROBDD (a ROBDD consisting of a single 0-valued leaf only).
  - A propositional formula is a tautology iff its ROBDD is isomorphic to the “1” ROBDD (a ROBDD consisting of a single 1-valued leaf only).
**Variable Ordering**

- The size of the ROBDD depends **very significantly** on the chosen variable ordering.
- For example, for the function \( f(x_1, \ldots, x_{2n}) = (x_1 \land x_2) \lor (x_3 \land x_4) \lor \cdots \lor (x_{2n-1} \land x_{2n}) \),
  - \( 2^{n+1} \) ROBDD nodes are needed when using the variable ordering \( x_1 < x_3 < \cdots < x_{2n-1} < x_2 < x_4 < \cdots < x_{2n} \), but
  - \( 2n + 2 \) nodes suffice when using the ordering \( x_1 < x_2 < x_3 < x_4 < \cdots < x_{2n-1} < x_{2n} \).
Variable Ordering

- Variable ordering is usually fixed at the beginning and maintained throughout all operations with BDDs.

- Finding an optimal ordering is \textbf{NP}-hard.

- Various heuristics may be used, e.g., based on putting close to each other the variables which are in some sense closely related (the value of one is computed from the other one or they are together used as an input of some function, etc.).

- Another possibility is the so-called dynamic reordering:
  - It is started when the size of the ROBDD starts to grow.
  - It is based on moving (one-by-one: the so-called sifting) the individual variables to different positions in the ordering by iteratively re-ordering two successive variables \( v_i \) and \( v_{i+1} \) via swapping the “0-1” and “1-0” successors of nodes labelled with \( v_i \).

Operations on ROBDDs

- **equivalence checking**: isomorphism checking (in $O(\min(|N_1|, |N_2|))$) or just root (pointer) comparison (in $O(1)$),
- **negation**: simply invert the value of leaves (in $O(1)$),
- **binary Boolean operations** (16 in total)—via a single function $\text{Apply}$:
  - uses restriction, Shannon expansion, and dynamic programming,
  - works in $O(|N_1| \cdot |N_2|)$ as we shall see.
Operations on ROBDDs

- Operations on ROBDDs:
  - equivalence checking: isomorphism checking (in $\mathcal{O}(\min(|N_1|, |N_2|))$) or just root (pointer) comparison (in $\mathcal{O}(1)$),
  - negation: simply invert the value of leaves (in $\mathcal{O}(1)$),
  - binary Boolean operations (16 in total)—via a single function $\text{Apply}$:
    - uses restriction, Shannon expansion, and dynamic programming,
    - works in $\mathcal{O}(|N_1| \cdot |N_2|)$ as we shall see.

- Restriction of a Boolean function $f$ is a Boolean function obtained by fixing some parameter of $f$ to a given value: $f|_{v_i \leftarrow b}(v_1, \ldots, v_n) = f(v_1, \ldots, v_{i-1}, b, v_{i+1}, \ldots, v_n)$.
  - On ROBDDs:
    1. for each node $n \in N$ such that $\text{var}(n) = v_i$, redirect all edges leading to $n$ to $\text{low}(n)$ if $b = 0$ and to $\text{high}(n)$ if $b = 1$, respectively, and remove $n$,
    2. apply $\text{Reduce}$ (to obtain a canonical form again).
Shannon Expansion and Apply

- The Shannon expansion of a Boolean function \( f(v_1, \ldots, v_i, \ldots, v_n) \) wrt. a variable \( v_i \):
  \[
  f(v_1, \ldots, v_n) = (\neg v_i \land f|_{v_i \leftarrow 0}(v_1, \ldots, v_n)) \lor (v_i \land f|_{v_i \leftarrow 1}(v_1, \ldots, v_n))
  \]

- Using the Shannon expansion as a basis of the `Apply` function:
  - \( f \ op g = (\neg v \land (f|_{v \leftarrow 0} \ op \ g|_{v \leftarrow 0})) \lor (v \land (f|_{v \leftarrow 1} \ op \ g|_{v \leftarrow 1})). \)
  - For example:
    - \( f \land g = (\neg v \land (f|_{v \leftarrow 0} \land g|_{v \leftarrow 0})) \lor (v \land (f|_{v \leftarrow 1} \land g|_{v \leftarrow 1})). \)
    - \( f \lor g = (\neg v \land (f|_{v \leftarrow 0} \lor g|_{v \leftarrow 0})) \lor (v \land (f|_{v \leftarrow 1} \lor g|_{v \leftarrow 1})). \)

- Intuitively, the functions are unfolded into their decision trees on whose leaves the appropriate operation is done.
The Apply Function

Function Apply

Input: a binary Boolean operator $op$, ROBDDs $G_1, G_2$ representing Boolean functions $f_1, f_2$, respectively, over the same indexed family of variables $\{v_i\}_{i \in I}$ ordered wrt. $\prec$.

Output: a ROBDD $G$ representing the Boolean function $f_1 \ op \ f_2$ over $\{v_i\}_{i \in I}$.

Method:
1. Call ApplyFrom($op, G_1, G_2, root_1, root_2$).
2. Apply Reduce on the result of step 1 and return the result.
ApplyFrom (part 1/2)

Function ApplyFrom

Input: a binary Boolean operator $op$, ROBDDs $G_1, G_2$ over the same indexed family of variables $\{v_i\}_{i \in I}$ ordered wrt. $\prec$, and nodes $x_1 \in N_1 \cup T_1$, $x_2 \in N_2 \cup T_2$.

Output: an OBDD $G$ representing the Boolean function $f_1 op f_2$ over $\{v_i\}_{i \in I}$ where $f_1, f_2$ are Boolean functions represented by $G_1$ and $G_2$, respectively, when $x_1$ and $x_2$ are considered as the roots (and their predecessors are ignored).

Method:
ApplyFrom (part 1/2)

Function ApplyFrom

Input: a binary Boolean operator $\text{op}$, ROBDDs $G_1$, $G_2$ over the same indexed family of variables $\{v_i\}_{i \in I}$ ordered wrt. $\prec$, and nodes $x_1 \in N_1 \cup T_1$, $x_2 \in N_2 \cup T_2$.

Output: an OBDD $G$ representing the Boolean function $f_1 \text{ op } f_2$ over $\{v_i\}_{i \in I}$ where $f_1$, $f_2$ are Boolean functions represented by $G_1$ and $G_2$, respectively, when $x_1$ and $x_2$ are considered as the roots (and their predecessors are ignored).

Method:
1. If $x_1 \in T_1$ and $x_2 \in T_2$ (i.e., both $x_1$ and $x_2$ are leaves), return the ROBDD consisting of a single leaf with the value $\text{val}(x_1) \text{ op } \text{val}(x_2)$. 


**Function** ApplyFrom

**Input:** a binary Boolean operator $op$, ROBDDs $G_1, G_2$ over the same indexed family of variables $\{v_i\}_{i \in I}$ ordered wrt. $\prec$, and nodes $x_1 \in N_1 \cup T_1, x_2 \in N_2 \cup T_2$.

**Output:** an OBDD $G$ representing the Boolean function $f_1 \ op \ f_2$ over $\{v_i\}_{i \in I}$ where $f_1, f_2$ are Boolean functions represented by $G_1$ and $G_2$, respectively, when $x_1$ and $x_2$ are considered as the roots (and their predecessors are ignored).

**Method:**
1. If $x_1 \in T_1$ and $x_2 \in T_2$ (i.e., both $x_1$ and $x_2$ are leaves), return the ROBDD consisting of a single leaf with the value $\text{val}(x_1) \ op \ \text{val}(x_2)$.
2. Otherwise (at least one of $x_1$ and $x_2$ is an inner node):
ApplyFrom (part 1/2)

Function ApplyFrom

Input: a binary Boolean operator $op$, ROBDDs $G_1, G_2$ over the same indexed family of variables $\{v_i\}_{i \in I}$ ordered wrt. $\prec$, and nodes $x_1 \in N_1 \cup T_1$, $x_2 \in N_2 \cup T_2$.

Output: an OBDD $G$ representing the Boolean function $f_1 \ op \ f_2$ over $\{v_i\}_{i \in I}$ where $f_1, f_2$ are Boolean functions represented by $G_1$ and $G_2$, respectively, when $x_1$ and $x_2$ are considered as the roots (and their predecessors are ignored).

Method:

1. If $x_1 \in T_1$ and $x_2 \in T_2$ (i.e., both $x_1$ and $x_2$ are leaves), return the ROBDD consisting of a single leaf with the value $val(x_1) \ op \ val(x_2)$.
2. Otherwise (at least one of $x_1$ and $x_2$ is an inner node):
   
   (a) If $var(x_1) = var(x_2) = v$ for some variable $v$,
   
   - let $G'_1 = \text{ApplyFrom}(op, G_1, G_2, \text{low}_1(x_1), \text{low}_2(x_2))$, i.e., compute $f_1|_{v \leftarrow 0} \ op \ f_2|_{v \leftarrow 0}$ using the fact that $f_i|_{v \leftarrow 0} = \text{low}_i(x_i)$ for $i \in \{1, 2\}$,
**ApplyFrom (part 1/2)**

**Function** ApplyFrom

**Input:** a binary Boolean operator $op$, ROBDDs $G_1$, $G_2$ over the same indexed family of variables $\{v_i\}_{i \in I}$ ordered wrt. $\prec$, and nodes $x_1 \in N_1 \cup T_1$, $x_2 \in N_2 \cup T_2$.

**Output:** an OBDD $G$ representing the Boolean function $f_1 \ op \ f_2$ over $\{v_i\}_{i \in I}$ where $f_1$, $f_2$ are Boolean functions represented by $G_1$ and $G_2$, respectively, when $x_1$ and $x_2$ are considered as the roots (and their predecessors are ignored).

**Method:**

1. If $x_1 \in T_1$ and $x_2 \in T_2$ (i.e., both $x_1$ and $x_2$ are leaves), return the ROBDD consisting of a single leaf with the value $val(x_1) \ op \ val(x_2)$.
2. Otherwise (at least one of $x_1$ and $x_2$ is an inner node):
   
   (a) If $\var(x_1) = \var(x_2) = v$ for some variable $v$,
   
   - let $G'_1 = \text{ApplyFrom}(op, G_1, G_2, \text{low}_1(x_1), \text{low}_2(x_2))$, i.e., compute $f_1|_{v \leftarrow 0} \ op \ f_2|_{v \leftarrow 0}$ using the fact that $f_i|_{v \leftarrow 0} = \text{low}_i(x_i)$ for $i \in \{1, 2\}$,
   - let $G'_2 = \text{ApplyFrom}(op, G_1, G_2, \text{high}_1(x_1), \text{high}_2(x_2))$,
   - return the OBDD constructed from $G'_1$ and $G'_2$ having roots $\text{root}'_1$ and $\text{root}'_2$, resp., by uniting their sets of terminals and non-terminals (assumed to be disjoint), the $\var$, $\text{low}$, $\text{high}$, and $val$ functions, and by adding a new root node $n$ such that $\var(n) = v$, $\text{low}(n) = \text{root}'_1$, and $\text{high}(n) = \text{root}'_2$. 

Introduction – p.16/21
ApplyFrom (part 2/2)

Continuation of step 2:

(b) Otherwise, if \( var(x_1) = v \) for some variable \( v \) and either \( x_2 \in T_2 \) or \( x_2 \in N_2 \) and \( v \prec var(x_2) \) (meaning that \( f_2 \) is independent of \( v \), i.e., \( f_2|_{v\leftarrow 0} = f_2|_{v\leftarrow 1} = f_2 \)).
ApplyFrom (part 2/2)

Continuation of step 2:

(b) Otherwise, if $\text{var}(x_1) = v$ for some variable $v$ and either $x_2 \in T_2$ or $x_2 \in N_2$ and $v < \text{var}(x_2)$ (meaning that $f_2$ is independent of $v$, i.e., $f_2|_{v \leftarrow 0} = f_2|_{v \leftarrow 1} = f_2$),

- let $G'_1 = \text{ApplyFrom}(\text{op}, G_1, G_2, \text{low}_1(x_1), x_2)$,
- let $G'_2 = \text{ApplyFrom}(\text{op}, G_1, G_2, \text{high}_1(x_1), x_2)$,
- return the OBDD constructed from $G'_1$ and $G'_2$ having roots $\text{root}'_1$ and $\text{root}'_2$, respectively, by uniting their sets of terminals and non-terminals (assumed to be disjoint), the $\text{var}$, $\text{low}$, $\text{high}$, and $\text{val}$ functions, and by adding a new root node $n$ such that $\text{var}(n) = v$, $\text{low}(n) = \text{root}'_1$, and $\text{high}(n) = \text{root}'_2$. 
Continuation of step 2:

(b) Otherwise, if \(\text{var}(x_1) = v\) for some variable \(v\) and either \(x_2 \in T_2\) or \(x_2 \in N_2\) and \(v \prec \text{var}(x_2)\) (meaning that \(f_2\) is independent of \(v\), i.e., \(f_2|_{v \leftarrow 0} = f_2|_{v \leftarrow 1} = f_2\)),

- let \(G_1' = \text{ApplyFrom}(\text{op}, G_1, G_2, \text{low}_1(x_1), x_2)\),
- let \(G_2' = \text{ApplyFrom}(\text{op}, G_1, G_2, \text{high}_1(x_1), x_2)\),
- return the OBDD constructed from \(G_1'\) and \(G_2'\) having roots \(\text{root}_1'\) and \(\text{root}_2'\), respectively, by unifying their sets of terminals and non-terminals (assumed to be disjoint), the \(\text{var}\), \(\text{low}\), \(\text{high}\), and \(\text{val}\) functions, and by adding a new root node \(n\) such that \(\text{var}(n) = v\), \(\text{low}(n) = \text{root}_1'\), and \(\text{high}(n) = \text{root}_2'\).

(c) Otherwise \(\text{var}(x_2) = v\) for some variable \(v\) and either \(x_1 \in T_1\) or \(x_1 \in N_1\) and \(v \prec \text{var}(x_1)\) and a symmetric step to step 2(b) is taken:

- let \(G_1' = \text{ApplyFrom}(\text{op}, G_1, G_2, x_1, \text{low}_2(x_2))\),
- let \(G_2' = \text{ApplyFrom}(\text{op}, G_1, G_2, x_1, \text{high}_2(x_2))\),
- return the OBDD constructed from \(G_1'\) and \(G_2'\) having roots \(\text{root}_1'\) and \(\text{root}_2'\), respectively, by unifying their sets of terminals and non-terminals (assumed to be disjoint), the \(\text{var}\), \(\text{low}\), \(\text{high}\), and \(\text{val}\) functions, and by adding a new root node \(n\) such that \(\text{var}(n) = v\), \(\text{low}(n) = \text{root}_1'\), and \(\text{high}(n) = \text{root}_2'\).
ApplyFrom: Further Remarks

- An example: use Apply over the ROBDDs representing $v_1 \land \neg v_2$ and $\neg v_1 \land v_2$, respectively, with $op$ being either $\land$ or $\lor$. 
**ApplyFrom: Further Remarks**

- An example: use `Apply` over the ROBDDs representing $v_1 \land \neg v_2$ and $\neg v_1 \land v_2$, respectively, with `op` being either $\land$ or $\lor$.

- Every call to `ApplyFrom` can result in two new calls to `ApplyFrom`: hence *exponential complexity*!
ApplyFrom: Further Remarks

❖ An example: use Apply over the ROBDDs representing $v_1 \land \neg v_2$ and $\neg v_1 \land v_2$, respectively, with $op$ being either $\land$ or $\lor$.

❖ Every call to ApplyFrom can result in two new calls to ApplyFrom: hence exponential complexity!

❖ Use dynamic programming:
  • store results of finished invocations of ApplyFrom in a hash table together with the appropriate arguments,
  • use the hash table to avoid re-computation of ApplyFrom over arguments on which it has already been applied.
**ApplyFrom: Further Remarks**

- An example: use `Apply` over the ROBDDs representing $v_1 \land \neg v_2$ and $\neg v_1 \land v_2$, respectively, with $op$ being either $\land$ or $\lor$.

- Every call to `ApplyFrom` can result in two new calls to `ApplyFrom`: hence exponential complexity!

- Use dynamic programming:
  - store results of finished invocations of `ApplyFrom` in a hash table together with the appropriate arguments,
  - use the hash table to avoid re-computation of `ApplyFrom` over arguments on which it has already been applied.

- The number of subgraphs in ROBDDs depends on the number of vertices $V = N \cup T$,
  - hence we have $O(|V_1| \cdot |V_2|)$ ways how to call `ApplyFrom`,
  - so the complexity becomes $O(|V_1| \cdot |V_2|)$. 

BDDs in Symbolic Verification
Encoding Kripke Structures by BDDs

- For symbolic CTL model checking, we need to represent Kripke structures and sets of their states satisfying some formulae using BDDs.
For symbolic CTL model checking, we need to represent Kripke structures and sets of their states satisfying some formulae using BDDs.

Hence, we need to use BDDs to encode sets of states and relations on states:
- the labelling function $L$ of Kripke structures can be encoded by encoding separately, for each atomic proposition $p \in AP$, the set of states in which $p$ holds.
For symbolic CTL model checking, we need to represent Kripke structures and sets of their states satisfying some formulae using BDDs.

Hence, we need to use BDDs to encode sets of states and relations on states:

- the labelling function $L$ of Kripke structures can be encoded by encoding separately, for each atomic proposition $p \in AP$, the set of states in which $p$ holds.

Having a finite set $S$ of states:

- We may code each state using a binary vector with $\lceil \log_2 |S| \rceil$ bits.
- An $i$-th bit may be assigned a Boolean variable $v_i$ and sets of the states may be coded as propositional formulae and hence BDDs:
For symbolic CTL model checking, we need to represent Kripke structures and sets of their states satisfying some formulae using BDDs.

Hence, we need to use BDDs to encode sets of states and relations on states:

- the labelling function $L$ of Kripke structures can be encoded by encoding separately, for each atomic proposition $p \in AP$, the set of states in which $p$ holds.

Having a finite set $S$ of states:

- We may code each state using a binary vector with $\lceil \log_2 |S| \rceil$ bits.
- An $i$-th bit may be assigned a Boolean variable $v_i$ and sets of the states may be coded as propositional formulae and hence BDDs:
  - For example, for $S = \{s_1, s_2, s_3\}$,
    - we may use 2 bits;
    - encode $s_1$ as 00, $s_2$ as 01, $s_3$ as 10;
    - associate the most-significant bit with $v_1$, the least-significant bit with $v_2$;
    - code $S$ as $\neg v_1 \lor (v_1 \land \neg v_2)$; and use the corresponding ROBDD.
Encoding Kripke Structures by BDDs

❖ For symbolic CTL model checking, we need to represent Kripke structures and sets of their states satisfying some formulae using BDDs.

❖ Hence, we need to use BDDs to encode sets of states and relations on states:

• the labelling function \( L \) of Kripke structures can be encoded by encoding separately, for each atomic proposition \( p \in AP \), the set of states in which \( p \) holds.

❖ Having a finite set \( S \) of states:

• We may code each state using a binary vector with \( \lceil \log_2 |S| \rceil \) bits.

• An \( i \)-th bit may be assigned a Boolean variable \( v_i \) and sets of the states may be coded as propositional formulae and hence BDDs:

  – For example, for \( S = \{s_1, s_2, s_3\} \),
    ○ we may use 2 bits;
    ○ encode \( s_1 \) as 00, \( s_2 \) as 01, \( s_3 \) as 10;
    ○ associate the most-significant bit with \( v_1 \), the least-significant bit with \( v_2 \);
    ○ code \( S \) as \( \neg v_1 \lor (v_1 \land \neg v_2) \); and use the corresponding ROBDD.

  – In practice, the encoding schema may reflect the internal structure of states (e.g., if states contain one 8-bit integer encoding a line number, two 8-bit integer variables, and 2 Boolean flags, we may use 26 bits by concatenating the bit representations of all the mentioned state variables).
A transition relation \( R \subseteq S \times S \) for \( S \) coded on \( n \) bits, associated with Boolean variables \( v_1, ..., v_n \), may be coded using \( 2n \) bits, associated with the Boolean variables \( v_1, ..., v_n \) and also Boolean variables \( v'_1, ..., v'_n \) constraining future values of the state variables.

For example, for the set \( S = \{s_1, s_2, s_3\} \) and the encoding of \( s_1 \) as 00, \( s_2 \) as 01, and \( s_3 \) as 10 from the previous slide,

- the relation \( R = \{(s_1, s_2), (s_1, s_3), (s_2, s_3), (s_3, s_3)\} \) may be encoded as
  - \((-v_1 \land -v_2 \land ((-v'_1 \land v'_2) \lor (v'_1 \land -v'_2))) \lor (-v_1 \land v_2 \land v'_1 \land -v'_2) \lor (v_1 \land -v_2 \land v'_1 \land -v'_2),\)
- which can in turn be represented as a ROBDD over 4 variables.

The encoding of the transition relation may again reflect the internal structure of the states and the bitwise implementation of the transitions on the components of states.