No duplicate keys to be stored in the hash-table

\{(\text{high}(\text{key}), \text{low}(\text{key})\}\}

Then, compute its key: \text{key}.

Every time you create a new node, reduce it

Use of a symbol table as a unique table

\begin{align*}
\eta \cdot f + b \cdot f &= (\eta, b, f) \in \text{ITE} \\
(x b \land x f) x + (\overline{x b} \land \overline{x f}) x &= b \land f
\end{align*}

\text{In the past lectures we learnt:}

\text{Direct Construction of ROBDDs}
The ITE Algorithm

\[
\text{ITE}(f, g, h) \{ \\
\text{if (terminal case)} \\
\quad x = \text{top variable of } f, g, h; \\
\quad e = \text{ITE}(f_x, g_x, h_x); \\
\quad t = \text{ITE}(f_x, g_x, h_x); \\
\text{if } (t = e) / \text{ redundant node} */ \\
\text{return } t; */ \text{ or return } e */ \\
\text{/* Look-up the unique table for isomorphic subtrees */} \\
\quad r = \text{find_or_add_unique_table}(e, x, t); \\
\text{Update unique table, if required;} \\
\text{return } r; \\
\text{end if}
\]
Fill-up the Unigue Table (symbol/hash table)

Assume var order a, q, c

First construct trivial ROBDDs for a, q, c

\[ c + q = q \cdot f; c' + q = q; c + a = f \]

ROBDD Construction Example
When the $f, g$ have not have same top-vars

... When $f, g$ have same top variable

... Few things to keep in mind

Construcitng ROBDDs using ITE
\[ Z = \text{ITE}(a, 1, c) = \text{ITE}(\text{Dd}_0, \text{Dd}_1, \text{Ddc}_1) \]

\[ \text{TOP-VAR} = a \]

\[ \text{ITE(Dd}_0\text{.Dd}_1\text{.Ddc}_1) = \text{ITE(Dd}_0\rightarrow \text{child.} \text{Dd}_1\text{.Ddc}_1) \]

### Table:

<table>
<thead>
<tr>
<th>Value</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Dc1</td>
</tr>
<tr>
<td>Dc1</td>
<td>Dd0, c, Dd1</td>
</tr>
<tr>
<td>Dc1</td>
<td>Dd0, b, Dd1</td>
</tr>
<tr>
<td>Da1</td>
<td>Dd0, a, Dd1</td>
</tr>
<tr>
<td>Da1</td>
<td>NULL, 1, NULL</td>
</tr>
<tr>
<td>Da0</td>
<td>NULL, 0, NULL</td>
</tr>
<tr>
<td>Da0</td>
<td>Dd0</td>
</tr>
</tbody>
</table>

\[ + a + c = f \]
Group a recursion level and compute the right sub-tree.

\[
\begin{align*}
\{ & = \text{ITE}(Dd_0, Dd_1, Dd_0, \text{left child}) \\
\} & = \text{ITE}(Dd_0, Dd_1, Dd_1, \text{right child}) \\
\text{Top-var} = a & \\
\text{Key} = \{ & = \text{ITE}(Dd_0, Dd_1, Dd_1) \\
\\text{TOP-VAR} = a & \\
Z = \text{ITE}(a, 1, b) = \text{ITE}(Dd_0, Dd_1, Dd_1) \\
\end{align*}
\]
Z = ITE(a, 1, c) = ITE(Dda1, DD1, Ddc1)

Top-var = a = ITE(Dda1, DD1, Ddc1)

Top-var = c = ITE(Dd1, Dd1, Ddc1)
\[ Z = \text{ITE}(\text{ITE}(\text{Dd}_1, \text{Dd}_1), \text{Dd}_1) = \text{ITE}(\text{Dd}_1, \text{Dd}_1) \]

\[ \text{TOP-VAR} = a \]

\[ \text{ITE}(\text{Dd}_1, \text{Dd}_1) = \text{ITE}(\text{Dd}_0, \text{Dd}_1) \]

\[ = \text{ITE}(\text{Dd}_0, \text{Dd}_1) \]

\[ = \text{ITE}(\text{Dd}_0, \text{Dd}_1) \]

\[ = \text{ITE}(1, \text{Dd}_1) \]

<table>
<thead>
<tr>
<th>Value</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>Dd0</td>
<td>{Dd0, c, Dd1}</td>
</tr>
<tr>
<td>Dd1</td>
<td>{Dd0, b, Dd1}</td>
</tr>
<tr>
<td>Dd0</td>
<td>{Dd0, a, Dd1}</td>
</tr>
<tr>
<td>Dd1</td>
<td>NULL</td>
</tr>
<tr>
<td>Dd0</td>
<td>NULL</td>
</tr>
</tbody>
</table>

\[ a + c = f \]
Final ROBDD for $f = a + c$
how the graph reduces to $D_0$.

Homework: Compute $g \cdot f$ yourself.

Homework: Compute $g$ yourself.

$$(\mathcal{C} + q) \cdot (\mathcal{C} + q) = g \cdot f$$
\[ g = f \] and prove to yourself that \[ b = f \] Do the same with \( b \) and prove to yourself that 

- Compute \( f \), compute keys of every node, update symbol table 
- \( c + q \cdot c = b; (c + q)(c + a) = f \) 

\textbf{Equivalence Verification}
BDD and SAT

How to pick a solution?

For the circuit shown, SAT

Application to SAT
Least one output would have worst-case scenario.
Multiple: There exists NO good ordering. Take any order, at
least through re-ordering.

Careful: Var ordering of ALL the BDDs in the manager has to
change in size of ROBDD.

Dynamic Var Ordering: Do this while constructing ROBDDs

Variable Ordering: Interchange order of variables and see

Var $x \in$ many cubes $\rightarrow$ keep it up in the tree

Intersectional Problem: Though some heuristics exist...

Given a Boolean function, How do we forecast a good var order?

Worst-case ROBDD size $\rightarrow$ no reduction $\rightarrow$ full-blown tree.

Change Var order $\rightarrow$ ROBDD structure and size change!!