BDD’s

We start with a binary decision diagram. For example, following is a decision diagram (tree) for the formula $(x_1 = y_1) \land (x_2 = y_2)$:

In general, it requires an exponential number of nodes.
Optimize

- Identify identical subgraphs.
- Remove redundant tests.

Yielding:
Definitions

A binary decision diagram BDD is a rooted, directed acyclic graph with

- One or two nodes of out-degree zero labeled 0 or 1, and
- A set of variable nodes $u$ of out-degree 2. The two outgoing edges are given by the functions $low(u)$ and $high(u)$. A variable $var(u)$ is associated with each node.

A BDD is ordered (OBDD) if the variables respect a given linear order $x_1 < x_2 < \cdots < x_n$ on all paths through the graph. An OBDD is reduced (ROBDD) if it satisfies:

- Uniqueness – no two distinct nodes are the roots of isomorphic subgraphs.
- No redundant tests – $low(u) \neq high(u)$ for all nodes $u$ in the graph.

For simplicity, we will refer to ROBDD simply as BDDs.
**Claim 4.** For every function $f : \text{Bool}^n \to \text{Bool}$ and variable ordering $x_1 < x_2 < \cdots < x_n$, there exists exactly one BDD representing this function.

The complexity of BDD representation is very sensitive to the variable ordering. For example, the BDD representation of $(x_1 = y_1) \land (x_2 = y_2)$ under the variable ordering $x_1 < x_2 < y_1 < y_2$ is:

![BDD Diagram]
Implementation of BDD Packages

Types and Variables:

\[
\begin{align*}
\text{node} & \quad = \quad \text{naturals} \\
\text{var}_\text{num} & \quad = \quad \text{naturals} \\
\text{node}_\text{rec} & \quad = \quad \text{record of} \quad \begin{cases}
\text{var} & : \quad \text{var}_\text{num} \\
\text{low}, \text{high} & : \quad \text{node}
\end{cases} \\
\text{T} & \quad : \quad \text{node} \rightarrow \text{node}_\text{rec} \\
\text{H} & \quad : \quad \text{node}_\text{rec} \rightarrow \text{node} \cup \{\perp\}
\end{align*}
\]

Operations:

\[init(T)\] Initialize \( T \) to contain only 0 and 1
\[u := \text{new}(T, i, l, h)\] allocate a new node \( u \), such that \( T(u) = \langle i, l, h \rangle \)
\[init(H)\] initialize \( H \) to \( \perp \)

\( H \) is the inverse of \( T \). That is, \( H(T(u)) = u \), for every \( u \in \text{dom}(T) \).

We will write \( \text{var}(u), \text{low}(u), \text{high}(u), \) and \( H(i, l, h) \) as abbreviations for \( T(u).\text{var}, T(u).\text{low}, T(u).\text{high}, \) and \( H(\langle i, l, h \rangle) \).
Internal Representation

\[ T : u \rightarrow \langle i, \ell, h \rangle \]

<table>
<thead>
<tr>
<th>u</th>
<th>var</th>
<th>low</th>
<th>high</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>
Making or Retrieving a node_id

Function \text{Mk} (i : \text{var\_num}; \ell, h : \text{node}) : \text{node}

1: \quad \text{if } \ell = h \text{ then return } \ell \\
2: \quad \text{if } H(i, \ell, h) \neq \bot \text{ then return } H(i, \ell, h) \\
3: \quad u := \text{new}(i, \ell, h) \\
4: \quad H(i, \ell, h) := u \\
5: \quad \text{return } u
Applying a Binary Boolean Operation to two BDD’s

Let \( \text{op} : \text{Bool} \times \text{Bool} \to \text{Bool} \) be a binary boolean operation. The following function uses the auxiliary dynamic array \( G : \text{node} \times \text{node} \to \text{node} \).

**Function** \( \text{Apply}(\text{op} ; u_1, u_2 : \text{node}) : \text{node} \)

\[
G := \perp
\]

**function** \( \text{App}(u_1, u_2 : \text{node}) : \text{node} \)

\[
\begin{align*}
\text{if } & G[u_1, u_2] \neq \perp \text{ then return } G[u_1, u_2] \\
\text{if } & u_1 \in \{0, 1\} \land u_2 \in \{0, 1\} \text{ then } u := \text{op}(u_1, u_2) \\
\text{else if } & \text{var}(u_1) = \text{var}(u_2) \text{ then} \\
& u := \text{Mk}(\text{var}(u_1), \text{App}(\text{low}(u_1), \text{low}(u_2)), \\
& \quad \text{App}(\text{high}(u_1), \text{high}(u_2)))
\end{align*}
\]

\[
\begin{align*}
\text{else if } & \text{var}(u_1) < \text{var}(u_2) \text{ then} \\
& u := \text{Mk}(\text{var}(u_1), \text{App}(\text{low}(u_1), u_2), \text{App}(\text{high}(u_1), u_2))
\end{align*}
\]

\[
\begin{align*}
\text{else } (\ast \text{var}(u_1) > \text{var}(u_2)\ast) \\
& u := \text{Mk}(\text{var}(u_2), \text{App}(u_1, \text{low}(u_2)), \text{App}(u_1, \text{high}(u_2)))
\end{align*}
\]

\[
G[u_1, u_2] := u
\]

**return** \( u \)

**end** \( \text{App} \)

**return** \( \text{App}(u_1, u_2) \)
**Restriction (Substitution)**

**Function** \( \text{REST}(u : \text{node}; \ j : \text{var\_num}; \ b : \text{Bool}) : \text{node} \)

\[
G := \bot
\]

\[
\text{function } \text{res}(u : \text{node}) : \text{node} = \\
\quad \text{if } G[u] \neq \bot \text{ then return } G[u] \\
\quad \text{if } \text{var}(u) > j \text{ then } r := u \\
\quad \text{else if } \text{var}(u) < j \text{ then} \\
\qquad r := \text{Mk}(\text{var}(u), \text{res}(\text{low}(u)), \text{res}(\text{high}(u))) \\
\quad \text{else } (*\text{var}(u) = j*) \text{ if } b = 0 \text{ then } r := \text{low}(u) \\
\qquad \text{else } r := \text{high}(u) \\
G[u] := r \\
\quad \text{return } r \\
\text{end res}
\]

**return** \( \text{res}(u) \)

Restriction is the same as substitution. We denote by \( t[x \mapsto b] \) the result of substituting \( b \) for \( x \) in assertion \( t \).
Quantification

Existential quantification can be computed, using the equivalence

$$\exists x : t \sim t[x \mapsto 0] \lor t[x \mapsto 1]$$

Universal quantification can be computed dually:

$$\forall x : t \sim t[x \mapsto 0] \land t[x \mapsto 1]$$
Application to Symbolic Model Checking

Let $V$ be the state variables for the FDS $\mathcal{D}$. Taking a vocabulary $U = V \cup V'$, we represent the state formulas $\Theta$, $J$ for each $J \in \mathcal{J}$, $p_i$, $q_i$, for each $\langle p_i, q_i \rangle \in \mathcal{C}$, and the SMC-INV symbolic working variables $new$ and $old$ as BDD’s over $U$ which are independent of $V'$.

The transition relation $\rho$ is represented as a BDD over $U$ which may be fully dependent on both $V$ and $V'$.

All the boolean operations used in the SMC-INV algorithm can be implemented by the Apply function. Negation can be computed by $\neg t = t \oplus 1$, where $\oplus$ is sum modulo 2.

To check for equivalence such as $old = new$ we compute $t := \langle old \iff new \rangle$ and then verify that the result is the singleton BDD 1.

The existential pre-condition transformer is computed by

$$\rho \odot \psi = \exists V' : \rho(V, V') \land \psi(V')$$

Priming an assertion $\psi$ is performed by

$$prime(\psi) = \exists V : \psi(V) \land V' = V$$