Symbolic Model Checking

Next, we consider the **symbolic** approach to model checking. Note that every assertion over a finite-domain FDS can be represented as a boolean formula over boolean variables. We assume that a finite-state FDS is represented by such formulas, including the **initial condition** $\Theta$ and the **bi-assertion** $\rho$ representing the transition relation.

A key development for symbolic model checking was the development of **binary decision diagrams** (BDD) as an efficient representation of boolean assertions.
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 (leaves) labeled 0 or 1, and

- A set of variable nodes $u$ of out-degree 2. The two outgoing edges are given by the functions $\text{low}(u)$ and $\text{high}(u)$. A variable $\text{var}(u)$ 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 – $\text{low}(u) \neq \text{high}(u)$ for all nodes $u$ in the graph.

For simplicity, we will refer to ROBDD simply as BDDs.
Claim 12. For every function $f : \text{Bool}^n \rightarrow \text{Bool}$ and variable ordering $x_1 < x_2 < \cdots < x_n$, there exists exactly one BDD representing this function.
Sensitivity to Variable Ordering

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]

1. **BDD Diagram**: The diagram shows the BDD representation of the given Boolean formula under the specified variable ordering. The nodes represent variables, and the paths represent the decision process. The terminal nodes are labeled with 1 or 0, indicating the truth value of the formula.

2. **Variable Ordering**: The ordering \(x_1 < x_2 < y_1 < y_2\) affects the structure of the BDD, influencing its size and complexity.
Implementation of BDD Packages

Types and Variables:

\[
\begin{align*}
\text{node} & = \text{naturals} \\
\text{var\_num} & = \text{naturals} \\
\text{node\_rec} & = \begin{array}{c}
\text{record of} \\
\text{var : var\_num;}
\text{low, high : node}
\end{array} \\
T & : \text{node} \rightarrow \text{node\_rec} \\
H & : \text{node\_rec} \rightarrow \text{node} \cup \{\perp\}
\end{align*}
\]

Operations:

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

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

**Table Representation**

<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>

**Function**

\[ T : u \rightarrow \langle i, \ell, h \rangle \]
Making or Retrieving a node_id

Function \text{\texttt{MK}}(i: \texttt{var_num}; \ell, h: \texttt{node}) : \texttt{node}

\begin{itemize}
  \item \textbf{1:} \textbf{if} \ell = h \textbf{ then return } \ell
  \item \textbf{2:} \textbf{if} \ H(i, \ell, h) \neq \bot \textbf{ then return } H(i, \ell, h)
  \item \textbf{3:} \ u := \texttt{new}(i, \ell, h)
  \item \textbf{4:} \ H(i, \ell, h) := u
  \item \textbf{5:} \textbf{ return } u
\end{itemize}
Applying a Binary Boolean Operation to Two BDD’s

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

Function $\text{Apply}(op; u_1, u_2 : \text{node}) : \text{node}$ \hspace{1em} -- Apply $op$ to BDD’s $u_1$ and $u_2$

$G := \bot$

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

if $G[u_1, u_2] \neq \bot$ then $u := G[u_1, u_2]$

else if $u_1 \in \{0, 1\} \land u_2 \in \{0, 1\}$ then $u := op(u_1, u_2)$

else if $\text{var}(u_1) = \text{var}(u_2)$ then

$u := \text{MK}(\text{var}(u_1), \text{App}(\text{low}(u_1), \text{low}(u_2)), \text{App}(\text{high}(u_1), \text{high}(u_2)))$

else if $\text{var}(u_1) < \text{var}(u_2)$ then

$u := \text{MK}(\text{var}(u_1), \text{App}(\text{low}(u_1), u_2), \text{App}(\text{high}(u_1), u_2))$

else (*$\text{var}(u_1) > \text{var}(u_2)$*)

$u := \text{MK}(\text{var}(u_2), \text{App}(u_1, \text{low}(u_2)), \text{App}(u_1, \text{high}(u_2)))$

$G[u_1, u_2] := u$

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}

\begin{align*}
\text{function } \text{res}(u : \text{node}) : \text{node} & = \\
\text{if } \text{var}(u) > j \text{ then return } u \\
\text{if } \text{var}(u) < j \text{ then } \\
\quad \text{return } \text{Mk}(\text{var}(u), \text{res}(\text{low}(u)), \text{res}(\text{high}(u))) \\
\quad (\star \text{var}(u) = j \star) \text{ if } b = 0 \text{ then return } \text{low}(u) \\
\quad \text{else return } \text{high}(u) \\
\text{end res}
\end{align*}

\text{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 of BDD’s to Symbolic Model Checking

Let $V$ be the state variables for the FDS $D$. 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 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 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 $t := (old \leftrightarrow new)$ and then verify that the result is the singleton BDD 1.

The existential pre-condition transformer is computed by

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

Priming an assertion $\psi$ is performed by

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