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.
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:

```
x_1
y_1
x_2
y_1
y_2
y_2
0
1
```
 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 $low(u)$ 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 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](image-url)
Implementation of BDD Packages

Types and Variables:

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

Operations:

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

\(H\) 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

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

<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 \texttt{MK}(i:\texttt{var\_num}; \ell, h:\texttt{node}) : \texttt{node}

\begin{align*}
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 := \texttt{new}(i, \ell, h) \\
4: & \quad H(i, \ell, h) := u \\
5: & \quad \text{return } u
\end{align*}
Applying a Binary Boolean Operation to Two BDD’s

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

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

$G := \bot$

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

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 **App**

**return** Apply$(u_1, u_2)$
Restriction (Substitution)

Function \textbf{REST} (u : \textit{node}; \ j : \textit{var}_\text{num}; \ b : \textit{Bool}) : \textit{node} \\
\hspace{1cm} \text{-- Substitute } b \text{ for } x_j \text{ in } \textit{BDD} u \\

\begin{verbatim}
function res(u : \textit{node}) : \textit{node} = 
    if \textit{var}(u) > j \text{ then return } u 
    \text{if } \textit{var}(u) < j \text{ then} 
        \hspace{1cm} \text{return Mk(} \textit{var}(u), \text{res(}\textit{low}(u)), \text{res(}\textit{high}(u))) 
        \hspace{1cm}(\ast \textit{var}(u) = j \ast) \text{ if } b = 0 \text{ then return } \textit{low}(u) 
        \text{else return } \textit{high}(u) 
end res

return res(u)
\end{verbatim}

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:

$$\forall x : t \sim t[x \mapsto 0] \land t[x \mapsto 1]$$