Main Operations on OBDDs

Cesare Tinelli

tinelli@cs.uiowa.edu

The University of Iowa
BDDs as Inductive Data Types

Mathematically, we can see BDDs as terms generated by the following grammar:

\[
BDD \ ::= \ \text{leaf}(Bit) \mid \text{node}(Label, BDD, BDD)
\]

\[
Bit \ ::= \ 0 \mid 1
\]

\[
Label \ ::= \ x \mid x_1 \mid \cdots \mid y \mid y_1 \cdots \mid z \mid z_1 \mid \cdots
\]

In this term representation,

- \text{leaf}(0) (resp., \text{leaf}(1)) denotes the BDD consisting of just a terminal node with label 0 (resp., 1),
- a term of the form \text{node}(x, B_0, B_1) denotes any BDD with a root node whose
  - label is \(x\),
  - low child is the BDD \(B_0\)
  - high child is the BDD \(B_1\).
BDDs as Inductive Data Types

**Note:** Representing BDDs as terms is not completely accurate because any information on which subBDDs are shared is lost in the term representation.

For instance, the term

\[
\text{node}(x, \text{node}(y, \text{leaf}(0), \text{leaf}(1)), \text{leaf}(1))
\]

represents both the BDD with two terminal nodes with label 1 and its reduced version containing only one such node.

The terms representation is helpful in explaining how various composition operations on ordered BDDs are defined—if one ignores for simplicity the issue of subBDD sharing.

In actual implementations though, subgraph sharing and memoization are crucial for time and space efficiency.
The apply operation

The operation apply implements a function

\[
\text{apply} : (\text{Bit} \times \text{Bit} \rightarrow \text{Bit}) \times \text{BDD} \times \text{BDD} \rightarrow \text{BDD}
\]

that takes a binary bit-operator \( op \) and two BDDs ordered \( \preceq \) over labels, and returns a new BDD ordered \( \preceq \).

Similarly, we also have

\[
\text{restrict} : \text{Bit} \times \text{Label} \times \text{BDD} \rightarrow \text{BDD}
\]

\[ \text{exists} : \text{Label} \times \text{BDD} \rightarrow \text{BDD} \]

These functions are defined inductively and by cases as follows.
The apply operation

1. \( \text{apply}(\text{op, leaf}(b), \text{leaf}(c)) = \text{leaf}(\text{op}(b, c)) \)

2. \( \text{apply}(\text{op, node}(u, B_0, B_1), \text{node}(v, C_0, C_1)) \)
   \[
   \begin{cases}
   \text{node}(u, \text{apply}(\text{op, } B_0, C_0), \text{apply}(\text{op, } B_1, C_1)) & \text{if } u = v \\
   \text{node}(u, \text{apply}(\text{op, } B_0, C), \text{apply}(\text{op, } B_1, C)) & \text{if } u < v \\
   \text{node}(u, \text{apply}(\text{op, } B, C_0), \text{apply}(\text{op, } B, C_1)) & \text{if } v < u
   \end{cases}
   \]

3. \( \text{apply}(\text{op, node}(u, B_0, B_1), \text{leaf}(b)) \)
   \[
   = \text{node}(u, \text{apply}(\text{op, } B_0, C), \text{apply}(\text{op, } B_1, C))
   \]

4. \( \text{apply}(\text{op, leaf}(b), \text{node}(v, C_0, C_1)) \)
   \[
   = \text{node}(v, \text{apply}(\text{op, } B, C_0), \text{apply}(\text{op, } B, C_1))
   \]
The restrict operation

Let $i \in \{0, 1\}$

1. $\text{restrict}(i, v, \text{leaf}(b)) = \text{leaf}(b)$

2. $\text{restrict}(i, v, \text{node}(u, B_0, B_1))$

\[
= \begin{cases} 
\text{node}(u, \text{restrict}(i, v, B_0), \text{restrict}(i, v, B_1)) & \text{if } u \prec v \\
B_i & \text{if } u = v \\
\text{node}(u, B_0, B_1) & \text{otherwise}
\end{cases}
\]
The \texttt{exists} operation

Basic definition:

$$\text{exists}(v, B) = \text{apply}(+, \text{restrict}(0, v, B), \text{restrict}(1, v, B))$$

An equivalent, but more direct, definition:

1. \text{exists}(v, \texttt{leaf}(b)) = \texttt{leaf}(b)
2. \text{exists}(v, \texttt{node}(u, B_0, B_1))

$$= \begin{cases} 
\text{node}(u, \text{exists}(i, v, B_0), \text{exists}(i, v, B_1)) & \text{if } u < v \\
\text{apply}(+, B_1, B_2) & \text{if } u = v \\
\text{node}(u, B_0, B_1) & \text{otherwise}
\end{cases}$$