BDD operations

Paul Jackson\textsuperscript{1}
Paul.Jackson@ed.ac.uk

University of Edinburgh

Formal Verification
Autumn 2023

\textsuperscript{1}Diagrams from Huth & Ryan, LiCS, 2nd Ed.
reduce algorithm

Aim is to construct a ROBDD from an OBDD.

- Adds integer labels \( \text{id}(n) \) to each node \( n \) of a BDD in a single bottom-up pass.
- Key property: if nodes \( m \) and \( n \) are labelled, then \( \text{id}(m) = \text{id}(n) \) iff \( m \) and \( n \) represent the same Boolean function.
- Rules for adding label to node \( n \):
  - remove duplicate terminals: if \( n \) terminal, set \( \text{id}(n) \) to \( \text{val}(n) \)
  - remove redundant tests: if \( \text{id(lo}(n)) = \text{id(hi}(n)) \), set \( \text{id}(n) \) to \( \text{id(lo}(n)) \)
  - remove duplicate nodes: if there exists a labelled node \( m \) such that
    \[
    \begin{cases} 
    \text{var}(m) = \text{var}(n) \\
    \text{id(lo}(m)) = \text{id(lo}(n)) \\
    \text{id(hi}(m)) = \text{id(hi}(n))
    \end{cases}
    \]
    set \( \text{id}(n) \) to \( \text{id}(m) \)

Use hash table with \( \langle \text{var}(n), \text{id(lo}(n)), \text{id(hi}(n)) \rangle \) keys for O(1) search time
- otherwise, set \( \text{id}(n) \) to unused number
- ROBDD generated by using 1 node from each class of nodes with the same label
- Node sharing between ROBDDs possible if hash table shared
reduce example

Reduce $3/15$
apply algorithm I - specification

Given

- Boolean formulas $f$ and $g$,
- ROBDDs $B_f$ and $B_g$ for $f$ and $g$,
- a binary operation $\text{op}$ on boolean formulas (e.g. $\land$, $\lor$, $\oplus$)

$$\text{apply}(\text{op}, B_f, B_g)$$

computes a ROBDD for $f \text{ op } g$.

Can also use $\text{apply}$ for negation: compute the ROBDD for $\neg f$ using

$$\text{apply}(\oplus, B_f, [1])$$

In essence, this just swaps terminal nodes 0 and 1.
apply algorithm II - the Shannon expansion

Consider a Boolean formula \( f \) represented by a BDD with top-level structure

\[
\begin{array}{c}
\times \\
\downarrow \\
B_0 \quad B_1
\end{array}
\]

Sub-BDDs \( B_0 \) and \( B_1 \) also correspond to formulas, say \( f_0 \) and \( f_1 \)

What are the relationships between \( f \), \( f_0 \) and \( f_1 \)?

\[
\begin{align*}
f_0 & \equiv f[0/x] \\
f_1 & \equiv f[1/x] \\
f & \equiv \overline{x}.f_0 + x.f_1
\end{align*}
\]

The implied formula

\[
f \equiv \overline{x}.f[0/x] + x.f[1/x]
\]

is called the Shannon expansion of Boolean formula \( f \) with respect to the variable \( x \).
apply algorithm III - the key idea

Consider the Shannon expansion of \( f \) \( \op \) \( g \) and pushing substitutions through \( \op \):

\[
f \op g \equiv \bar{x} \cdot (f \op g)[0/x] + x \cdot (f \op g)[1/x]
\]

\[
\equiv \bar{x} \cdot (f[0/x] \op g[0/x]) + x \cdot (f[1/x] \op g[1/x])
\]

This recursive characterisation of \( \op \) suggests a recursive algorithm for computing \( \op \) on BDDs.
apply algorithm IV - the definition

\[
\text{apply}(\text{op}, x, x) = \text{apply(\text{op}, B, C)} \times \text{apply(\text{op}, B', C')}
\]

\[
\text{apply}(\text{op}, x, C) = \text{apply(\text{op}, B, C)} \times \text{apply(\text{op}, B', C')}
\]

where \( C \) is 1) a terminal node or 2) a non-terminal with \( \text{var(root}(C)) > x \)

\[
\text{apply}(\text{op}, B, x) = \text{apply(\text{op}, B, C)} \times \text{apply(\text{op}, B, C')}
\]

where \( B \) is 1) a terminal node or 2) a non-terminal with \( \text{var(root}(B)) > x \)

\[
\text{apply}(\text{op}, u, v) = w \quad \text{where} \quad w = u \times v
\]
apply example

Compute \( \text{apply}(+, B_f, B_g) \) where \( B_f \) and \( B_g \) are:

\[
\begin{align*}
R_1 & \rightarrow x_1 \\
R_2 & \rightarrow x_2 \\
R_3 & \rightarrow x_3 \\
R_4 & \rightarrow x_4 \\
R_5 & \rightarrow 0 \\
R_6 & \rightarrow 1 \\
S_1 & \rightarrow x_1 \\
S_2 & \rightarrow x_3 \\
S_3 & \rightarrow x_4 \\
S_4 & \rightarrow 0 \\
S_5 & \rightarrow 1
\end{align*}
\]
Recursive calls of apply
Final result from apply execution
In general, result will not be an ROBDD, so need to use `reduce` afterwards.

- Or can incorporate aspects of `reduce` into `apply` so result is always reduced.

Naive implementation has run-time exponential in number of variables.

- Each `apply` call in 3 of 4 cases results in two recursive calls.

However, only $|B_f| \cdot |B_g|$ distinct calls.

- If calls `memoized`, $O(|B_f| \cdot |B_g|)$ time complexity is possible.
Other operations

- **restrict**\((0, x, B_f)\) computes ROBDD for \(f[0/x]\)
  1. For each node \(n\) labelled with an \(x\), incoming edges are redirected to \(\text{lo}(n)\) and \(n\) is removed.
  2. Resulting BDD is reduced.

- **exists**\((x, B_f)\) computes ROBDD for \(\exists x. f\)
  - Uses identity \((\exists x. f) \equiv f[0/x] + f[1/x]\) and restrict and apply functions
## Time complexities

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Input OBDD(s)</th>
<th>Output OBDD</th>
<th>Time-complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>reduce</td>
<td>$B$</td>
<td>reduced $B$</td>
<td>$O(</td>
</tr>
<tr>
<td>apply</td>
<td>$B_f, B_g$ (reduced)</td>
<td>$B_f \circ g$ (reduced)</td>
<td>$O(</td>
</tr>
<tr>
<td>restrict</td>
<td>$B_f$ (reduced)</td>
<td>$B_f[0/x]$ or $B_f[1/x]$ (reduced)</td>
<td>$O(</td>
</tr>
<tr>
<td>$\exists$</td>
<td>$B_f$ (reduced)</td>
<td>$B_{\exists x_1 \exists x_2 \ldots \exists x_n} f$ (reduced)</td>
<td>NP-complete</td>
</tr>
</tbody>
</table>

H&R Figure 6.23
States represented using Boolean vectors $\langle v_1, \ldots, v_n \rangle$, where $v_i \in \{0, 1\}$.

Sets of states represented using BDDs on $n$ variables $x_1, \ldots x_n$ describing characteristic functions of sets.

Set operations $\cup, \cap, \overline{\cdot}$ computed using the apply algorithm and the Boolean operations $+, \cdot, \overline{\cdot}$.

Transition relations described using BDDs on $2n$ variables.

If Boolean variables $x_1, \ldots x_n$ describe initial state and Boolean variables $x'_1, \ldots x'_n$ describe next state, then good ordering is $x_1, x'_1, x_2, x'_2, \ldots x_n, x'_n$.

Translations of Boolean formulas describing state sets and transition relations into BDDs make use of apply algorithm, following structure of formulas

This avoids the intractable exponential blow-up if instead one tried to first construct a binary decision tree.
The existential pre-image function

\[
\text{pre}_\exists(Y) = \{ s \in S \mid \exists s' \in S. s \to s' \land s' \in Y \}
\]

is computed using

\[
\text{exists}(x'_1, \text{exists}(x'_2, \ldots \text{exists}(x'_n, \text{apply}(\cdot, B\to, B'_Y)) \ldots))
\]

where

- \( B\to \) is the ROBDD representing the transition relation \( \to \)
- \( B'_Y \) is the ROBDD representing set \( Y \) with the variables \( x_1, \ldots x_n \) renamed to \( x'_1, \ldots x'_n \)

To compute the universal pre-image function

\[
\text{pre}_\forall(Y) = \{ s \in S \mid \forall s' \in S. s \to s' \Rightarrow s' \in Y \}
\]

we observe that

\[
\text{pre}_\forall(Y) = S - \text{pre}_\exists(S - Y)
\]

and note that the computation for \(- (\text{set complement})\) is the same as the computation for logical negation.