Binary Decision Diagrams

Binary Decision Diagrams (BDDs) are a class of graphs that can be used as data structure for compactly representing boolean functions. BDDs were introduced by R. Bryant in 1986. BDDs are used to solve equivalence problems between formulas of propositional logic. Very important in the areas of hardware design and hardware optimization.

Boolean Functions

A boolean function of arity \( n \geq 1 \) is a function \( \{0, 1\}^n \to \{0, 1\} \).

Examples:

\[
\text{or}(x_1, x_2) = \begin{cases} 
1 & \text{if } x_1 = 1 \text{ or } x_2 = 1 \\
0 & \text{if } x_1 = 0 \text{ or } x_2 = 0 
\end{cases}
\]

\[
\text{if}_\text{then}_\text{else}(x_1, x_2, x_3) = \begin{cases} 
 x_2 & \text{if } x_1 = 1 \\
 x_3 & \text{if } x_1 = 0 
\end{cases}
\]

Z.B.: \( \text{if}_\text{then}_\text{else}(1, 0, 1) = 0 \), \( \text{if}_\text{then}_\text{else}(0, 0, 1) = 1 \)

\[
\text{sum}(x_1, x_2, x_3, x_4) = \begin{cases} 
1 & \text{if } x_1 + x_2 = x_3x_4 \\
0 & \text{otherwise} 
\end{cases}
\]

Z.B.: \( \text{sum}(1, 1, 1, 0) = 1 \) (because \( 1 + 1 = 10 \)), \( \text{sum}(0, 0, 0, 1) = 0 \) (because \( 0 + 0 = 00 \)).

Graphs

Recall some basic graph-theoretical concepts:

directed graph node edge predecessor successor path cycle acyclic graph tree forest

Graphs recall some basic graph-theoretical concepts:
majority\(_n\)(x_1, \ldots, x_n) = \begin{cases} 
1 & \text{if the majority of } x_1, \ldots, x_n \text{ has value 1} \\
0 & \text{otherwise} 
\end{cases}

Z.B.: majority_4(1, 1, 0, 0) = 0, majority_3(1, 0, 1) = 1

parity\(_n\)(x_1, \ldots, x_n) = \begin{cases} 
1 & \text{if the number of inputs } x_1, \ldots, x_n \text{ equal to 1 is even} \\
0 & \text{otherwise} 
\end{cases}

Z.B.: parity_3(1, 0, 1) = 1, parity_2(1, 0) = 0

Let \( F \) be a formula, and let \( n \) be a number such that all atomic formulas occurring in \( F \) belong to \( \{A_1, \ldots, A_n\} \).

Example: \( F = A_1 \land A_2 \), \( n = 2 \), but also \( n = 3! \)

We define the boolean function \( f^n_F : \{0, 1\}^n \to \{0, 1\} : \)

\[
f^n_F(x_1, \ldots, x_n) = \text{truth value of } F \text{ under the assignment that sets } A_1, \ldots, A_n \text{ to } x_1, \ldots, x_n
\]

Example: For \( F = A_1 \land A_2 \) :

\[
f^2_F(0, 1) = \text{value of } 0 \land 1 = 0
\]

\[
f^3_F(0, 1, 1) = \text{value of } 0 \land 1 = 0
\]

Remark: If all of \( \{A_1, \ldots, A_n\} \) occur in \( F \), then \( f^n_F \) is essentially the truth table of \( F \).

Convention: We write e.g. \( f(x_1, x_2, x_3) = x_1 \lor (x_2 \land \neg x_1) \), meaning \( f = f^3_F \) for the formula \( F = A_1 \lor (A_2 \land \neg A_1) \).

Fact: Let \( F_1 \) and \( F_2 \) be two formulas, and let \( n \) be a number such that all atomic formulas occurring in \( F_1 \) or \( F_2 \) belong to \( \{A_1, \ldots, A_n\} \). Then \( f^n_{F_1} = f^n_{F_2} \) iff \( F_1 \equiv F_2 \).

Example: \( F_1 = A_1 \), \( F_2 = A_1 \land (A_2 \lor \neg A_2) \).

\[
\begin{align*}
f^2_{F_1}(0, 0) &= 0 = f^2_{F_2}(0, 0) \\
f^2_{F_1}(0, 1) &= 0 = f^2_{F_2}(0, 1) \\
f^2_{F_1}(1, 0) &= 1 = f^2_{F_2}(1, 0) \\
f^2_{F_1}(1, 1) &= 1 = f^2_{F_2}(1, 1)
\end{align*}
\]

Convention: The constants 0 and 1 represent the only two boolean functions of arity 0.
A boolean function can be represented by a decision tree.

A decision tree can use a variable order different from the order used in the function.

A variable order is a bijection

\[ b: \{1, \ldots, n\} \rightarrow \{x_1, \ldots, x_n\} \]

We say that \( b(1), b(2), b(3), \ldots, b(n) \) are the first, second, third, \ldots, \( n \)-th variable w.r.t. the order \( b \).

We denote the bijection \( b(1) = x_{i_1}, \ldots, b(n) = x_{i_n} \) by

\[ x_{i_1} < x_{i_2} < \ldots < x_{i_n} \]

A decision tree for the variable order \( x_{i_1} < \ldots < x_{i_n} \) is a tree satisfying the following conditions:

1. All leaves are labelled by 0 or by 1.
2. All other nodes are labelled by a variable and have exactly two children, the 0-child and the 1-child. The edges leading to these children are labelled by 0 resp. by 1.
3. If the root is not a leaf, then it is labelled by \( x_{i_1} \).
4. If a node is labelled by \( x_{i_n} \) then its two children are leaves.
5. If a node is labelled by \( x_{i_j} \) and \( j < n \), then its two children are labelled by \( x_{i_{j+1}} \).
Every path of a decision tree determines an assignment of the variables \( x_{i_1}, \ldots, x_{i_n} \) and vice versa.

The boolean function \( f_T \) represented by a decision tree \( T \) is defined as follows:

\[
f_T(x_1, \ldots, x_n) = \text{label of the leaf reached by the path corresponding to the assignment } x_{i_1} x_{i_2} \ldots x_{i_n}
\]

A binary decision forest is a forest of decision trees with the same variable order. A decision forest represents the set of functions represented by its elements.

A BDD (multiBDD) is a “compact representation” of a binary decision tree (decision forest).

A BDD (multiBDD) is obtained from a decision tree (forest) through repeated application of two compression rules (see example in the next slide):

- Rule 1: Sharing of identical subtrees.
- Rule 2: Elimination of nodes for which the 0-child and the 1-child coincide (redundant nodes).

The rules are applied until all subtrees are different and there are no redundant nodes.

Example: sharing of subtrees

All 0- and 1-leaves are merged.
Example: sharing of subtrees

Identical $x_4$-nodes are merged.

Example: sharing of subtrees

Identical $x_3$-nodes are merged.

Example: removing redundant nodes

Redundant $x_4$-node is removed
A BDD for a given variable order is an acyclic directed graph satisfying the following properties:

1. There is exactly one node without predecessors (the root).
2. There is one or two nodes without successors, labelled by 0 or 1 (if there are two then they carry different labels).
3. All other nodes are labelled by a variable and have exactly two distinct children, the 0-child and the 1-child. The edges leading to these children are labelled by 0 resp. by 1.
4. A child of a node is labelled by 0, by 1, or by a variable larger than the label of its parent w.r.t. the variable order.
5. All descendant-closed subgraphs of the graph are non-isomorphic.

A multiBDD is an acyclic graph satisfying (2)-(5) together a distinguished nonempty subset of nodes called the roots. Every node without predecessors is a root, but other nodes may also be roots. A multiBDD represents a set of boolean functions, one for each root.

Remarks

Remark: A "closed subgraph" of a BDD is again a BDD.

Remark: The function true_n(x_1, ..., x_n) given by

\[ \text{true}_n(x_1, \ldots, x_n) = 1 \text{ for every } x_1, x_n \in \{0, 1\}^n \]

is represented, for every \( n \geq 1 \) and for every variable order, by the BDD consisting of one single node labelled by 1.

Similarly for false_n(x_1, ..., x_n)

Relevance of variable orders

The variable order can have large impact on the size of the BDD.

Example:

\[ f(x_1, \ldots, x_{2n}) = (x_1 \leftrightarrow x_{n+1}) \land (x_2 \leftrightarrow x_{n+2}) \land \cdots \land (x_n \leftrightarrow x_{2n}) \]

Size grows exponentially in \( n \) for \( x_1 < \cdots < x_n < x_{n+1} < \cdots < x_{2n} \).
Size grows linearly in \( n \) for \( x_1 < x_{n+1} < x_2 < x_{n+2} < \cdots < x_n < x_{2n} \).

Problem in practice: finding a good order.
### Canonicity of BDDs

We show that for a given boolean function and a given variable order there is a **unique BDD** representing the function.

More generally (but simpler to prove!), we show that for every set of boolean functions of the same arity and for every variable order there is a **unique multiBDD** representing the set.

### The functions $f[0]$ und $f[1]$

**Lemma I**: Let $f$ be a boolean function of arity $n \geq 1$. There are exactly two boolean functions $f[0]$ und $f[1]$ of arity $(n - 1)$ satisfying

$$f(x_1, \ldots, x_n) = (\neg x_1 \land f[0](x_2, \ldots, x_n)) \lor (x_1 \land f[1](x_2, \ldots, x_n)) \quad (*)$$

**Proof**: The functions $f[0]$ and $f[1]$ defined by

$$f[0](x_2, \ldots, x_n) = f(0, x_2, \ldots, x_n)$$ and

$$f[1](x_2, \ldots, x_n) = f(1, x_2, \ldots, x_n)$$ satisfy (*).

Let $f_0$ and $f_1$ be arbitrary functions satisfying (*). Then

$$f(x_1, \ldots, x_n) = (\neg x_1 \land f_0(x_2, \ldots, x_n)) \lor (x_1 \land f_1(x_2, \ldots, x_n))$$

By the properties of $\lor$ and $\land$ we have

$$f(0, x_2, \ldots, x_n) = f_0(x_2, \ldots, x_n)$$ and together with

$$f(0, x_2, \ldots, x_n) = f[0](x_2, \ldots, x_n)$$ we get $f_0 = f[0]$.

The proof that $f_1 = f[1]$ holds is analogous.

**Theorem**: Let $\mathcal{F}$ be a nonempty set of boolean functions of arity $n$ and let $x_{i_1} < \ldots < x_{i_n}$ be a variable order. There is exactly one multiBDD that follows this order and represents $\mathcal{F}$.

**Proof**: We consider the order $x_1 < x_2 < \ldots < x_n$, for other orders the proof is similar. Proof by induction on the arity $n$.

**Basis**: $n = 0$. There are exactly two boolean functions with $n = 0$, namely the constants 0 and 1, and two BDDs $K_0, K_1$ consisting of one single node labelled by 0 or by 1. The set $\{0\}$ is represented by $K_0$, the set $\{1\}$ by $K_1$, and the set $\{0, 1\}$ by the multiBDD consisting of $K_0$ and $K_1$.

Let $f: \{0, 1\}^n \rightarrow \{0, 1\}$ be a boolean function, let $B$ be a BDD with variable order $x_1 < x_2 < \ldots < x_n$, and let $v$ be the root of $B$. Define the nodes $v[0]$ and $v[1]$ as follows:

1. If $v$ is labelled by $x_1$, then $v[0]$ and $v[1]$ are the 0-child and the 1-child of $v$.

**Lemma II**: $B$ represents the function $f$ iff $v[0]$ and $v[1]$ represent the functions $f[0]$ and $f[1]$, respectively.

**Proof**: Easy.
Let \( B \) be the multiBDD with roots \( v_1, \ldots v_n \) obtained from \( B' \) after executing the following steps for \( i = 1, 2, \ldots, k \):

- If \( v_{i0} = v_{i1} \) then set \( v_i := v_{i0} \).
  (In this case \( v_{i0} \) represents \( f_i \).)
- If \( v_{i0} \neq v_{i1} \) and \( B' \) has a node \( v \) such with \( v_{i0} \) as 0-child and \( v_{i1} \) as 1-child then set \( v_i := v \).
- If \( v_{i0} \neq v_{i1} \) and \( B' \) contains no such node then add a new node \( v_i \) having \( v_{i0} \) as 0-Kind and \( v_{i1} \) as 1-Kind.
  (So \( v_i \) represents \( f_i \), see Lemma II.)

Clearly, \( B \) represents \( F \). We now show that \( B \) is the only multiBDD representing \( F \).

### Computing BDDs from Formulas

**Goal**: Given a formula \( F \) over the atomic formulas \( A_1, \ldots, A_n \) and a variable order for \( \{x_1, \ldots, x_n\} \), compute a BDD representing \( f_F(x_1, \ldots, x_n) \).

**Naive procedure**: Compute the decision tree of \( f_F \) and reduce it using the compression rules.

**Problem**: The decision tree is too large!

**Better procedure (idea)**: Compute recursively the multiBDD representing \( \{f_F[A_i/0], f_F[A_i/1]\} \) for a suitable \( A_i \), and derive from it the BDD for \( f_F \), where \( F[A_i/0] \) bzw. \( F[A_i/0] \) are the formulas obtained by replacing every occurrence of \( A_i \) by 0 resp. by 1.

In the next slides we formalize this idea.
Let $S = \{F_1, \ldots, F_n\}$ be a nonempty set of formulas.

We define a procedure multiBDD($S$) that returns the roots of a multiBDD representing the set $\{f_{F_1}, \ldots, f_{F_n}\}$.

$K_0$ denotes the BDD with only one node labelled by 0. $K_1$ denotes the BDD with only one node labelled by 1.

A proper formula is a formula containing at least one occurrence of a variable (i.e., not only 0 and 1). An atomic formula $A_i$ is smaller than $A_j$ if $x_i$ appears before $x_j$ in the variable order.

An atomic formula $A_i$ is smaller than $A_j$ if $x_i$ appears before $x_j$ in the variable order.

The function multiBDD($S$) if $S$ contains no proper formulas then if all formulas of $S$ are equivalent to 0 then return $\{K_0\}$ else if all formulas in $S$ are equivalent to 1 then return $\{K_1\}$ else return $\{K_0, K_1\}$ else choose a proper formula $F \in S$.

Let $A_i$ be the smallest atomic formula occurring in $F$.

Let $B = $ multiBDD( ($S \setminus \{F\}$) $\cup \{F[A_i/0], F[A_i/1]\}$ ) .

Let $v_0, v_1$ be the roots of $B$ representing $F[A_i/0], F[A_i/1]$.

if $v_0 = v_1$ then return $B$ else add a new node $v$ with $v_0, v_1$ as 0- and 1-child (if such a node does not exist yet); return $(B \setminus \{v_0, v_1\}) \cup \{v\}$.

Equivalence problems

Given two formulas $F_1, F_2$, the following algorithm decides whether $F_1 \equiv F_2$ holds:

- Choose a suitable variable order $x_1 < \ldots < x_n$.
- Compute a multiBDD for $\{F_1, F_2\}$.
- Check whether the roots $v_{F_1}, v_{F_2}$ are equal.

For digital circuits: the BDDs are not derived from formulas, but directly from the circuits.

Operations on BDDs

Given:

- two formulas $F, G$ over the atomic formulas $A_1, \ldots, A_n$,
- a variable order for $\{x_1, \ldots, x_n\}$,
- a multiBDD with two roots $v_F, v_G$ representing the functions $f_F(x_1, \ldots, x_n)$ and $f_F(x_1, \ldots, x_n)$, and
- a binary boolean operation (e.g. $\lor, \land, \rightarrow, \leftrightarrow$)

Goal: compute a BDD for the function $f_{F \circ G}(x_1, \ldots, x_n)$.

With our convention we have $f_{F \circ G} = f_F \circ f_G$.
Idea

**Lemma:** \((f_F \circ f_G)[0] = f_F[0] \circ f_G[0]\) and \((f_F \circ f_G)[1] = f_F[1] \circ f_G[1]\).

**Proof:** Exercise.

**Algorithm:** (for the order \(x_1 < x_2 < \ldots < x_n\), similar for others)

- Compute a multivariable BDD for \(\{f_F[0] \circ f_G[0], f_F[1] \circ f_G[1]\}\).
  (Recursively.)
- Use the Lemma to build a BDD for \(f_F \circ f_G(x_1, \ldots, x_n)\).

---

**Implementing BDDs**

BDD-nodes coded as numbers 0, 1, 2, . . . with 0, 1 for the end nodes.

BDD-nodes are stored in a table

\[
T: u \mapsto (i, l, h)
\]

where \(i, l, h\) are the label, the 0-child and the 1-child of \(u\).

(Here \(l\) stands for “low” and \(h\) for “high”.)

We maintain a second table

\[
H: (i, l, h) \mapsto u
\]

so that following invariant holds:

\[
T(u) = (i, l, h) \quad \text{iff} \quad H(i, l, h) = u
\]

---

**The function \(\text{Or}(v_F, v_G)\)**

if \(v_F = K_1\) or \(v_F = K_1\) then return \(K_1\)
else if \(v_F = v_G = K_0\) then return \(K_0\)
else let \(v_{F0}, v_{G0}\) be the nodes for \(F[0], G[0]\) and
let \(v_{F1}, v_{G1}\) be the nodes for \(F[1], G[1]\)
\(v_0 := \text{Or}(v_{F0}, v_{G0}); v_1 := \text{Or}(v_{F1}, v_{G1})\)
if \(v_0 = v_1\) then return \(v_0\)
else add a new node \(v\) with \(v_0, v_1\) as 0- and 1-child
(if such a node does not exist yet);
return \(v\)

---

**Data structures**

Source: An introduction to Binary Decision Diagrams
Prof. H.R. Andersen
http://www.itu.dk/people/hra/notes-index.html
Basic operations on $T$:

- $\text{init}(T)$: Initializes $T$ with 0 and 1
- $\text{add}(T, i, l, h)$: Adds node with attributes $(i, l, h)$ to $T$ and returns it

- $\text{var}(u)$, $\text{low}(u)$, $\text{high}(u)$: Returns the variable, 0-child, 1-child of $u$

Basic operations on $H$:

- $\text{init}(H)$: Initializes $H$ as the empty table
- $\text{member}(H, i, l, h)$: Checks whether $(i, l, h)$ belongs to $H$
- $\text{lookup}(H, i, l, h)$: Returns the node $H(i, l, h)$
- $\text{insert}(H, i, l, h, u)$: Adds $(i, l, h) \mapsto u$ to $H$ (if not yet there)

Implementing $\text{Or}$

**Problem:** the function can be called many times with the same arguments.

**Solution:** dynamic programming. The results of all calls are stored. Each call checks first if the result has already been computed earlier.

\[
\text{Or}(u_1, u_2)
\]

1: init $G$
2: return $\text{Or}'(u_1, u_2)$

\[
\text{Or}'(u_1, u_2)
\]

1: if $G(u_1, u_2) \neq \text{empty}$ then return $G(u_1, u_2)$
2: else if $u_1 = 1$ or $u_2 = 1$ then return 1
3: else if $u_1 = 0$ and $u_2 = 0$ then return 0
4: else if $\text{var}(u_1) = \text{var}(u_2)$ then
5: \hspace{1em} $u := \text{Make}(\text{var}(u_1), \text{Or}'(\text{low}(u_1), \text{low}(u_2)), \text{Or}'(\text{high}(u_1), \text{high}(u_2)))$
6: else if $\text{var}(u_1) < \text{var}(u_2)$ then
7: \hspace{1em} $u := \text{Make}(\text{var}(u_1), \text{Or}'(\text{low}(u_1), u_2), \text{Or}'(\text{high}(u_1), u_2))$
8: else $u := \text{Make}(\text{var}(u_2), \text{Or}'(u_1, \text{low}(u_2)), \text{Or}'(u_1, \text{high}(u_2)))$
9: \hspace{1em} $G(u_1, u_2) = u$
10: return $u$