2. Binary Decision Diagrams

Content

2.1 BDD concepts
2.2 Variable orderings
2.3 OBDD algorithms
2.4 FDD´s and OKFDD´s
2.5 Integer valued decision diagrams
The problem of logic verification: show that two circuits implement the same boolean function
2. Binary Decision Diagrams

2.1 BDD concepts

- **Problem:** efficient representation of Boolean functions
  - DNF: linear for OR of n variables, exponential for XOR
  - Reed-Muller: linear for XOR of n variables, exponential for OR

- **Problem:** efficient application of Boolean operations
  - example:
    - DNF ➔ Negation ➔ DNF, e.g.:
      
      \[ ab + cd + ef + gh \Rightarrow (ab + cd + ef + gh) \Rightarrow \]

- Possible solution in many cases: binary decision diagrams (BDD´s)
Investigated systematically first by R. Bryant (CMU)
- Seminal paper by Bryant in ´86
- Early work by Shannon ~ 1940 (relais-networks)
- Revolutionary impact on logic synthesis, logic verification, etc.
- Many modern CAD-tools employ BDD´s
### 2. Binary Decision Diagrams

#### 2.1 BDD concepts

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- **Idea:** decompose a function into two sub-functions which do not depend on a certain variable, e.g., $x_1$
2. Binary Decision Diagrams

2.1 BDD concepts

- Idea: Decompose a function into two sub-functions which do not depend on a certain variable, e.g., $x_1$.
- Apply Boole’s expansion theorem in a systematic way to all variables.
- Represent result graphically.

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- $f(0, b, c, d)$
- $f(1, b, c, d)$
2. Binary Decision Diagrams

2.1 BDD concepts

\[ f = \overline{a} \cdot f(0, b, c, d) + a \cdot f(1, b, c, d) \]

\[
\begin{array}{cccc|c}
 a & b & c & d & f \\
 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 1 & 0 \\
 0 & 0 & 1 & 0 & 0 \\
 0 & 0 & 1 & 1 & 0 \\
 0 & 1 & 0 & 0 & 0 \\
 0 & 1 & 0 & 1 & 0 \\
 0 & 1 & 1 & 0 & 0 \\
 0 & 1 & 1 & 1 & 0 \\
 1 & 0 & 0 & 0 & 0 \\
 1 & 0 & 0 & 1 & 0 \\
 1 & 0 & 1 & 0 & 0 \\
 1 & 0 & 1 & 1 & 0 \\
 1 & 1 & 0 & 0 & 0 \\
 1 & 1 & 0 & 1 & 0 \\
 1 & 1 & 1 & 0 & 0 \\
 1 & 1 & 1 & 1 & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
 f(0, b, c, d) \\
 0 & 0 & 0 & 0 \\
 0 & 0 & 1 & 0 \\
 0 & 1 & 0 & 0 \\
 0 & 1 & 1 & 0 \\
 1 & 0 & 0 & 0 \\
 1 & 0 & 1 & 0 \\
 1 & 1 & 0 & 0 \\
 1 & 1 & 1 & 0 \\
\end{array}
\]

\[
\begin{array}{cccc}
 f(1, b, c, d) \\
 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 \\
 0 & 0 & 1 & 0 \\
 0 & 1 & 0 & 0 \\
 0 & 1 & 1 & 0 \\
 1 & 0 & 0 & 0 \\
 1 & 0 & 1 & 0 \\
 1 & 1 & 0 & 0 \\
 1 & 1 & 1 & 0 \\
\end{array}
\]
The application of Boole’s expansion theorem to all variables leads to a decision tree. Example: XOR in 3 variables
The application of Boole´s expansion theorem to all variables leads to a decision tree. Example: XOR in 3 variables
• **Variable ordering**: order in which Boole's expansion theorem is applied
2. Binary Decision Diagrams

2.1 BDD concepts

- Concepts:
  - Nodes
  - Directed edges
  - Edge labellings
  - (Direct) successors of node a
2. Binary Decision Diagrams

2.1 BDD concepts

- Concepts:
  - Root node
  - Paths
  - Leafs or Terminal nodes
Decision trees are ordered (identical variable ordering on all paths) or free

— Example of a free decision tree:
- A **fully expanded** decision tree has $2^n$ leaf nodes
- Example of a (free) decision tree which is not fully expanded:

```
  a
 /\
<table>
<thead>
<tr>
<th>0 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>b  c</td>
</tr>
</tbody>
</table>
  |\
<table>
<thead>
<tr>
<th>0 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>0 1</td>
</tr>
</tbody>
</table>
```
Observation: there are identical sub-trees
Observation: there are identical sub-trees
Merging identical sub-trees results in a decision-graph.
2. Binary Decision Diagrams

2.1 BDD concepts

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>(a \oplus b \oplus c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

"0-part" "1-part"
2. Binary Decision Diagrams

2.1 BDD concepts

- Shannon: *A symbolic analysis of relay and switching circuits* (1938)

  - Size of the networks grows linearly in the number of variables

![Diagram of BDD](image)

**Figure 28.** $\prod_{k=1}^{n} X_k$ for $n$ odd, $(\prod_{k=1}^{n} X_k)'$ for $n$ even
Some simple examples of BDD's:
2. Binary Decision Diagrams

2.1 BDD concepts

- AND-, OR-, XOR-operation in n variables

- #nodes grows linearly in #variables
2. Binary Decision Diagrams

2.1 BDD concepts

— Example:
SN 74181 ALU:
2. Binary Decision Diagrams

2.1 BDD concepts

— SN 74181 BDD ("shared BDD"): [Diagram of a binary decision diagram with variables and nodes labeled]
One path to the 1 leaf-node corresponds to a product – an implicant of the function. Example:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>c</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ a \oplus b \oplus c \]
2. Binary Decision Diagrams

2.1 BDD concepts

- 2 Problems:
  - Given a binary decision diagram.
    How to derive the Boolean function represented by the BDD?
  - Given a Boolean function.
    How to derive the BDD for it?
  - First: BDD  Boolean function
2. Binary Decision Diagrams

2.1 BDD concepts

- A node $v$ of a BDD is characterized by a triple $(x, v_0, v_1)$, where $v_0, v_1$ are the successors of $v$

  ![Diagram](image)

- The leaf nodes 0 and 1 represent the Boolean functions 0 and 1

- According to Boole's expansion theorem, the Boolean function $bf(v)$ is associated with node $v$ as follows (where $var(v)$ is the variable of node $v$):

$$bf(v) = \overline{var(v)} \cdot bf(v_0) + var(v) \cdot bf(v_1)$$
— Example: which Boolean function is represented by the following BDD?

The function associated with a node can be determined only if the functions associated with the successor nodes are known

\[ bf(v) = \overline{\text{var}(v)} \cdot bf(v_0) + \text{var}(v) \cdot bf(v_1) \]
"Bottom-up procedure": 1. Step
"Bottom-up procedure": 2. Step

\[ bf(v) = \overline{\text{var}(v)} \cdot bf(v_0) + \text{var}(v) \cdot bf(v_1) \]

\[ = \overline{b} \cdot 0 + b \cdot 1 = b \]
"Bottom-up procedure": 3. Step

\[ bf(v) = \overline{\text{var}(v)} \cdot bf(v_0) + \text{var}(v) \cdot bf(v_1) \]

\[ = a \cdot 0 + a \cdot b = a \cdot b \]

\[ bf(v) = \overline{\text{var}(v)} \cdot bf(v_0) + \text{var}(v) \cdot bf(v_1) \]

\[ = \overline{b} \cdot 0 + b \cdot 1 = b \]

Functions 0 and 1
There are many variants of binary decision diagrams

Most useful and common: OBDD's (Ordered Binary Decision Diagrams, Bryant 1986)

- **Ordered**: The variables appear in a fixed ordering on all paths
  - Technically, an index (a positive integer) is associated with each variable $\text{index}(\text{var}(v))$
  - For each node $v$ with successors $v_0$ and $v_1$ we have:
    - $\text{index}(\text{var}(v)) < \text{index}(\text{var}(v_0))$ and
    - $\text{index}(\text{var}(v)) < \text{index}(\text{var}(v_1))$
2. Binary Decision Diagrams

2.1 BDD concepts

variable order a, b, c

index(a) = 1

index(b) = 2

index(c) = 3
OBDD properties (cont'd.):

- Reduced:
  - The function represented by one node is different from the functions of all other nodes
  - The two successors of each node are distinct
2. Binary Decision Diagrams

2.1 BDD concepts

— Reduction example:

Several representations of 1

Identical successors
2. Binary Decision Diagrams

2.1 BDD concepts

- Simplified representations exist, e.g.,
  - 1-edges to the right, 0-edges to the left
  - Edges to 0 omitted
  - etc.
    - Example: \((a \oplus b) \cdot (c \oplus d) \cdot (e \oplus f)\)
    - or: 0 edges are dashed lines
Now: Boolean function \( \Rightarrow \) OBDD

- Example above: \((a \oplus b) \cdot (c \oplus d) \cdot (e \oplus f)\)
- Let
  \[ F = (a \oplus b) \cdot (c \oplus d) \cdot (e \oplus f) = (a \oplus b) \cdot r \]

- Variable ordering \(a, b, c, d, e, f\)
- Following Boole's expansion theorem, we have the following cofactors of \(F\) w.r.t. \(a\)

\[
\begin{align*}
F_a = (0 \oplus b) \cdot r &= b \cdot r \\
F_a = (1 \oplus b) \cdot r &= \overline{b} \cdot r
\end{align*}
\]
More expansions:

\[
F_a = (0 \oplus b) \cdot r = b \cdot r \\
F_a = (1 \oplus b) \cdot r = \overline{b} \cdot r
\]

\[
F_{\overline{a}b} = 0 \\
F_{ab} = F_{\overline{a}b} = r \\
F_{ab} = 0
\]

etc.
2. Binary Decision Diagrams

2.1 BDD concepts

- The problem of reduction:
  - In the example above it was easy to detect \( F_{\overline{a}b} = F_{\overline{a}b} = r \)
    and to merge the nodes for \( F_{\overline{a}b} \) and \( F_{\overline{a}b} \)
  - Redundant nodes have to be removed
  - Redundant nodes
    - Either represent the same function
    - Or have identical successors (easy to detect)

- How to know that two nodes represent the same function?
Two functions

\[ f = a f_a + \overline{a} f_{\overline{a}} \]

\[ g = a g_a + \overline{a} g_{\overline{a}} \]

are equal iff they have identical cofactors.
2. Binary Decision Diagrams

2.1 BDD concepts

- If we presume that all successor nodes of two nodes represent distinct functions then the two nodes represent identical functions iff the direct successor nodes are pairwise identical.

This results in a simple *bottom-up*-procedure: redundant nodes are eliminated in the bottom-level first, the in the second level, etc.
2. Binary Decision Diagrams

2.1 BDD concepts

1. Step:

0/1 leafs

all others

0
2. Binary Decision Diagrams

2.1 BDD concepts

2. Step: c nodes

all others

0

all others

0
We can decide that the two b-nodes do not represent the same function by means of the c-nodes.
Example: derive the OBDD for the following function, variable order r,e,g

\[ p = eg + rg + \bar{r}\bar{e}g \]

\[ p_r = eg + \bar{g} = \bar{g} \]

\[ p_{\bar{r}} = e\bar{g} + e\bar{g} \]

\[ 2^3 = 8 \text{ cases} \]
2. Binary Decision Diagrams

2.1 BDD concepts

\[ p = e\overline{g} + r\overline{g} + \overline{r}\overline{e}\overline{g} \]

\[ p_r = e\overline{g} + \overline{g} = \overline{g} \]

\[ p_{\overline{r}} = e\overline{g} + e\overline{g} \]
- Reduction was necessary in the original concept by R. Bryant (1986), but can be avoided completely (s. Sect. 2.3)
OBDD’s can be implemented easily by means of 2:1-Multiplexors.
Example:
Given a certain variable ordering, OBDD's are **canonical representations** of Boolean functions, i.e., there exists exactly one OBDD-representation for each Boolean function.

Two circuits implementing the same function have identical OBDD's.
2. Binary Decision Diagrams

2.1 BDD concepts

Circuit 1:

Circuit 2:

OBDD´s
2.2 Variable Orderings

- The variable ordering has a critical impact on the size of the OBDD (= #nodes)
- There are static and dynamic procedures to determine "good" orderings
The number of nodes of a OBDD depends critically on the variable ordering.

Classical example (Bryant 1986):

\[ f = x_1x_2 + x_3x_4 + x_5x_6 \]
2. Binary Decision Diagrams

2.2 Variable orderings

Example: n-bit adder:
- Order $R_1$: $a_n, b_n, a_{n-1}, b_{n-1}, \ldots, a_0, b_0$
- Order $R_2$: $a_n, a_{n-1}, \ldots, a_0, b_n, b_{n-1}, \ldots, b_0$

<table>
<thead>
<tr>
<th>$n=$</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_1$:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>time</td>
<td>0.02</td>
<td>0.03</td>
<td>0.11</td>
<td>0.19</td>
</tr>
<tr>
<td>#nodes</td>
<td>35</td>
<td>75</td>
<td>155</td>
<td>315</td>
</tr>
<tr>
<td>$R_2$:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>time</td>
<td>0.39</td>
<td>16.34</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#nodes</td>
<td>750</td>
<td>196574</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2. Binary Decision Diagrams

2.2 Variable orderings

- Calculating the best order may result in exponential run time
- For a given circuit, "good" orderings can be heuristically determined
  
  — Example: Distribution of a "weight"

![Binary Decision Diagram Example]

Sum of weights: $x = 1/2$, $y = 1/4$, $z = 1/4$,  
⇒ first use $x$ for expansion
2. Binary Decision Diagrams

2.2 Variable orderings

— Delete selected variable and distribute weight again

Sum of weights: \( y = \frac{3}{4}, \ z = \frac{1}{4}, \)  
\[ \Rightarrow \] next use \( y \) for expansion

— Order: \( x, \ y, \ z \)
2. Binary Decision Diagrams

2.2 Variable orderings

- Sifting: dynamic ordering procedure (Rudell ICCAD´93)
  - Basic step: exchange two adjacent variables (Fujita et al. EDAC´91)
Principle: exchange 0-1 and 1-0 path
2. Binary Decision Diagrams

2.2 Variable orderings

Diagram showing the variable orderings with nodes labeled a, b, and c, and edges connecting them with values 0 and 1.
2. Binary Decision Diagrams

2.2 Variable orderings
2. Binary Decision Diagrams

2.2 Variable orderings

• Sifting-procedure:
  - Calculate variable with max. #nodes (the "thickest" part of the OBDD)
  - Shift variable over OBDD by pairwise exchange of adjacent variables
  - Shift variable to a position where #nodes is minimal
Movie "Sifting" by Stefan Höreth:
2. Binary Decision Diagrams

2.2 Variable orderings
2. Binary Decision Diagrams

2.2 Variable orderings
2. Binary Decision Diagrams

2.2 Variable orderings
2. Binary Decision Diagrams

2.2 Variable orderings
2. Binary Decision Diagrams

2.2 Variable orderings
2. Binary Decision Diagrams

2.2 Variable orderings
2. Binary Decision Diagrams

2.2 Variable orderings

![Binary Decision Diagrams]

- Diagram on the left:
  - Variable orderings V3, V4, V5
  - Paths from root to leaves:
    - V3 → V4 → V5
    - V3 → V4 → V5

- Diagram on the right:
  - Variable orderings V3, V4, V5
  - Paths from root to leaves:
    - V4 → V3 → V5
    - V4 → V3 → V5
2. Binary Decision Diagrams

2.2 Variable orderings

[Diagram of a binary decision diagram with nodes labeled V0, V1, V2, V3, V4, V5, V6, V7, 1, and 0.]
2. Binary Decision Diagrams

2.2 Variable orderings
2. Binary Decision Diagrams

2.2 Variable orderings
2. Binary Decision Diagrams

2.2 Variable orderings
2. Binary Decision Diagrams

2.2 Variable orderings
2. Binary Decision Diagrams

2.2 Variable orderings
2. Binary Decision Diagrams

2.2 Variable orderings
2. Binary Decision Diagrams

2.2 Variable orderings
2. Binary Decision Diagrams

2.2 Variable orderings
2. Binary Decision Diagrams

2.2 Variable orderings
2. Binary Decision Diagrams

2.2 Variable orderings
2. Binary Decision Diagrams

2.2 Variable orderings
2. Binary Decision Diagrams

2.2 Variable orderings
2. Binary Decision Diagrams

2.3 OBDD Construction
- Principle: build OBDD while traversing the circuit from inputs to outputs
2. Binary Decision Diagrams

2.3 OBDD construction

![OBDD Diagram]

OBDD-Package

C-Program Traverser

C-Program

C-Program $\geq 1$
2. Binary Decision Diagrams

2.3 OBDD construction

[Diagram of OBDD construction with nodes labeled a, b, c, and edges showing 0 and 1 transitions]
2. Binary Decision Diagrams

2.3 OBDD construction

- **Orthogonality** of Boole's expansion

\[
\begin{align*}
  f + g &= x*(f_x + g_x) + \bar{x}*(f_{\bar{x}} + g_{\bar{x}}), \\
  f \cdot g &= x*(f_x \cdot g_x) + \bar{x}*(f_{\bar{x}} \cdot g_{\bar{x}}), \\
  \bar{f} &= x*f_{\bar{x}} + \bar{x}f_x
\end{align*}
\]
2. Binary Decision Diagrams

2.3 OBDD construction

- **AND-operation of two OBDD´s**
  - Assumption: nodes are represented as triples

\[(x, v_0, v_1)\]

- var, low, high

access-functions
function \text{AND}(\text{bdd1}, \text{bdd2}):

\text{IF} \ \text{bdd1}=0 \ \text{OR} \ \text{bdd2}=0 \ \text{THEN return} \ 0;

\text{ELSEIF} \ \text{bdd1}=1 \ \text{THEN return} \ \text{bdd2};

\text{ELSEIF} \ \text{bdd2}=1 \ \text{THEN return} \ \text{bdd1};

\text{ELSE} \ \text{var1}:=\text{var(}\text{bdd1})\text{;} \ \text{var2}:=\text{var(}\text{bdd2})\text{;}

\text{IF} \ \text{var1}=\text{var2} \ \text{THEN} \ x:=\text{var1}; \ v0:=\text{AND}(\text{low(}\text{bdd1})\text{, low(}\text{bdd2}))\text{,}

\quad v1:=\text{AND}(\text{high(}\text{bdd1}),\text{high(}\text{bdd2}))\text{;}

\text{ELSEIF} \ \text{index(}\text{var1}) < \text{index(}\text{var2}) \ \text{THEN} \ x:=\text{var1};

\quad v0:=\text{AND}(\text{low(}\text{bdd1}), \ \text{bdd2})\text{,}

\quad v1:=\text{AND}(\text{high(}\text{bdd1}), \ \text{bdd2})\text{;}

\text{ELSEIF} \ ...

\text{IF} \ v0 = v1 \ \text{THEN return} \ v0 \ \text{ELSE return} \ (x, v0, v1); \ ...

2. Binary Decision Diagrams

2.3 OBDD construction

$\text{bdd1} \quad \text{bdd2}$

$\text{var1}=a \quad \text{var2}=c \quad \Rightarrow \quad \text{index(var1)} < \text{index(var2)}$
2. Binary Decision Diagrams

2.3 OBDD construction

\[\text{bdd1} \quad \text{bdd2}\]
\[\text{var1}=a \quad \text{var2}=c \quad \Rightarrow \quad \text{index(var1)} < \text{index(var2)}\]

\[x:=\text{var1} := a\]
\[v0:= \text{and(low(bdd1), bdd2)}, \quad v1:= \text{and(high(bdd1), bdd2)}\]
2. Binary Decision Diagrams

2.3 OBDD construction

\[
\begin{align*}
\text{bdd1} & \quad \text{bdd2} \\
\text{var1=b} & \quad \text{var2=c} \quad \Rightarrow \quad \text{index(var1)} < \text{index(var2)}
\end{align*}
\]
2. Binary Decision Diagrams

2.3 OBDD construction

bdd1 bdd2

var1=b var2=c => index(var1) < index(var2)

x:= var1 := b
v0:= and(low(bdd1), bdd2), v1:= and(high(bdd1), bdd2)
2. Binary Decision Diagrams

2.3 OBDD construction

```
var2 = c    =>    index(var1) < index(var2)

x := var1 := b
v0 := and(low(bdd1), bdd2),
v1 := and(high(bdd1), bdd2)
```
2. Binary Decision Diagrams

2.3 OBDD construction

\[ \text{bdd1} \quad \text{bdd2} \]

\[ \text{var1}=b \quad \text{var2}=c \quad \Rightarrow \quad \text{index(var1)} < \text{index(var2)} \]

\[ \text{x:=var1 := b} \]

\[ \text{v0:= and(low(bdd1),bdd2),} \quad \checkmark \]

\[ \text{v1:= and(high(bdd1),bdd2)} \quad \checkmark \]
2. Binary Decision Diagrams

2.3 OBDD construction

\[
\begin{align*}
\text{bdd1} & \quad \text{bdd2} \\
\text{var1}=a & \quad \text{var2}=c \quad \Rightarrow \quad \text{index(var1)} < \text{index(var2)} \\
\text{x:=var1 := a} & \\
\text{v0:= and(low(bdd1),bdd2)} & \quad \checkmark \\
\text{v1:= and(high(bdd1),bdd2)} &
\end{align*}
\]
2. Binary Decision Diagrams

2.3 OBDD construction

\[ \text{bdd1} \quad \text{bdd2} \]
\[ \text{var1} = a \quad \text{var2} = c \quad \Rightarrow \quad \text{index(var1)} < \text{index(var2)} \]
\[ x := \text{var1} := a \]
\[ v_0 := \text{and(low(bdd1), bdd2)}, \quad v_1 := \text{and(high(bdd1), bdd2)} \]
2. Binary Decision Diagrams

2.3 OBDD construction

\[
\begin{align*}
&\text{bdd1} \\
&\text{bdd2} \\
&\text{var1=a} \\
&\text{var2=c} \\
&\Rightarrow \quad \text{index(var1) < index(var2)} \\
&x := \text{var1} := a \\
&v0 := \text{and(low(bdd1), bdd2)}, \quad \checkmark \\
&v1 := \text{and(high(bdd1), bdd2)} \quad \checkmark
\end{align*}
\]
"OBDD-Packages" manage two tables:

- The unique table (ut) with entries:

  | x | v0 | v1 |
  |

- For uniqueness of OBDD's
The computed table (ct) with entries

<table>
<thead>
<tr>
<th>Operation</th>
<th>bdd1</th>
<th>bdd2</th>
<th>Result bdd</th>
</tr>
</thead>
</table>

Stores previously calculated results
Reduction was needed in the original OBDD procedures.

OBDD uniqueness is guaranteed by:
- Checking in the unique-table (ut) if the OBDD was calculated before.
- Testing for identical successor nodes.

In addition, it is checked in the computed table (ct) if the result was calculated before.

Many steps of recursion may be saved.
2. Binary Decision Diagrams

2.3 OBDD construction

function AND(bdd1, bdd2):

IF (AND,bdd1,bdd2,x) ∈ ct THEN return x;

IF bdd1=0 OR bdd2=0 THEN return 0;

ELSEIF bdd1=1 THEN return bdd2;

ELSEIF bdd2=1 THEN return bdd1;

ELSE var1:=var(bdd1); var2:=var(bdd2);

IF var1=var2 THEN x:=var1; v0:= AND(low(bdd1), low(bdd2)), v1:= AND(high(bdd1),high(bdd2));

ELSEIF index(var1) < index(var2) THEN x:=var1; v0:= AND(low(bdd1), bdd2), v1:= AND(high(bdd1), bdd2);

ELSEIF ... IF v0 = v1 THEN return v0
ELSEIF (x,v0,v1) ∉ ut THEN put in ut; ELSE return (x,v0,v1); ...
2. Binary Decision Diagrams

2.3 OBDD construction

The computed table is essential for the efficiency of the algorithms:

- In principle, two additional steps of recursion may result at each step
  - the number of steps may grow exponentially in the number of variables

- Using the computed table with entries

<table>
<thead>
<tr>
<th>Operation</th>
<th>bdd1</th>
<th>bdd2</th>
<th>Result bdd</th>
</tr>
</thead>
</table>

the number of recursions is reduced to $|n1|*|n2|$ where $|n1|$ and $|n2|$ are the number of nodes of bdd1 and bdd2, respectively
2. Binary Decision Diagrams

2.3 OBDD construction

\[ a \oplus b \oplus c \oplus d \oplus e \oplus f \oplus g \]

\[ a \oplus b \oplus c \oplus d \oplus e \oplus f \oplus g \]
- * exponential in the number of variables?

- $O(n_1 \times n_2)$ using the computed table!
General result:

- If two OBDD’s with m and n nodes are logically combined then the resulting OBDD has \( \leq m*n \) nodes

This is due to the fact that not more than \( m*n \) distinct functions are generated!
Negated edges:

- The OBDD of a function \( f \) and the OBDD of the negated function are very similar: exchange the terminal nodes 0 and 1!
- Orthogonality of negation: negate a function by negating its cofactors

Problem: non-canonical representation!
Solution:

— Only the 0-edge can be a negated edge
— 1 terminal leaf only (or the dual version ...)

\[
\begin{align*}
\text{X} & \quad 0 \quad 1 \\
\text{X} & \quad 0 \quad 1 \\
\text{X} & \quad 0 \quad 1 \\
\text{X} & \quad 0 \quad 1
\end{align*}
\]
Examples: variable and negated variable

\[\text{g} \quad \text{g} \quad \text{g} \quad \text{g} \quad \text{g} \quad \text{g} \quad \text{g} \quad \text{g} \quad \text{g} \]

\[0 \quad 1 \quad 0 \quad 1 \quad 1 \quad 1 \quad 0 \quad 1 \quad 1 \]
2. Binary Decision Diagrams

2.3 OBDD construction

Example: XOR function
- **Cofactor calculation** using OBDD's
- `cof(x, pol, OBDD)`: `x` variable, `pol` polarity 1 or 0
- Easy if variable = top-variable, e.g., `cof(a, 0, OBDD)`: 

```
 a
 0 1
 b
 0 1
 c
 0 1
```

```
 b
 0 1
 c
 0 1
```
Generally: Replace pointers to the variable by the pointer to the 1-(0-)successor:

\[ f = ac + \bar{a}bc \]

\[ f_b = c \]

\[ f_{\bar{b}} = ac \]
Example: determine the 0-cofactor for variable d:
**Functional substitution:** substitute function $g$ vor variable $x$

- The paper-and-pencil method is: replace all occurrences of $x$ textually by $g$
- How to do that with a OBDD-representation?

$$f[x \leftarrow g] = \overline{g} \cdot f_{\overline{x}} + g \cdot f_x$$

**Rationale:**

$$f = x \cdot f_x + \overline{x} \cdot f_{\overline{x}}$$

$$f = \overline{g} \cdot f_{\overline{x}} + g \cdot f_x$$
Functional substitution: substitute function $g$ vor variable $x$

$$f[x \leftarrow g] = \bar{g} \cdot f_{\bar{x}} + g \cdot f_x$$

Note: $\exists x : (f \cdot (x \equiv g)) = [f_{\bar{x}} \cdot (0 \equiv g)] + [f_x \cdot (1 \equiv g)]$

$= f_{\bar{x}} \cdot \bar{g} + f_x \cdot g$

$= f[x \leftarrow g]$

Functional substitution can be reduced to the application of the $\exists$-operator
Using Boolean operations plus cofactor-calculation more advanced Boolean operations like the \( \exists \) - and \( \forall \) -quantifier and functional substitutions can be implemented
2. Binary Decision Diagrams

2.3 OBDD construction

- OBDD's are used in many CAD-tools for synthesis, verification and simulation
- Many public domain OBDD-packages
- Many are based on the ite(p, f, g)-operator (if p then f else g)
- CUDD package (Boulder Univ.)

☑️ OBDD are very efficient decision procedures for propositional calculus, and are integrated into many theorem provers like PVS and ACL2
2.4 FDD's and OKFDD's

- OBDD's are based on Boole's expansion theorem
- OBDD's represent the systematic decomposition in all variables
- Q: Are there other types of "decomposition"? How many?
Boole' expansion:

$$ f = x \cdot f_x + \overline{x} \cdot f_{\overline{x}} $$

There are more types of expansion (exactly two more):

- **Positive Davio-expansion**
  $$ f = f_{\overline{x}} \oplus x \cdot (f_{\overline{x}} \oplus f_x) $$

- **Negative Davio-expansion**
  $$ f = f_x \oplus \overline{x} \cdot (f_{\overline{x}} \oplus f_x) $$
FDD’s \textit{(Functional Decision Diagrams, Kebschull et al. 92)}

\[ f = f_{\bar{x}} \oplus x^*(f_{\bar{x}} \oplus f_x) \]

- for \( x = 0 \) \( \rightarrow f_{\bar{x}} \)
- for \( x = 1 \) \( \rightarrow f_{\bar{x}} \oplus f_{\bar{x}} \oplus f_x = f_x \)

\[ f \]

\[ f_{\bar{x}} \]

\[ (f_{\bar{x}} \oplus f_x) \]

Same graph structure, but different interpretation:

\textbf{Boolean difference of }f\textbf{ w.r.t. }x
FDD’s (*Functional Decision Diagrams*, Kebschull et al. 92)

- \( f = f_x \oplus x^* (f_x \oplus f) \)
  - for \( x = 0 \) \( \rightarrow f_x \)
  - for \( x = 1 \) \( \rightarrow f_x \oplus f_x \oplus f_x = f_x \)

- Same graph structure, but different interpretation:
  - Rule: variable = 1 \( \rightarrow \) XOR both branches to get the value of \( f \)
FDD’s are canonical representations

FDD's obey a different rule of reduction:

\[(f_{\bar{x}} \oplus f_x) : \text{if the Boolean difference is 0 then } f \text{ does not depend on } x\]
● Orthogonality of XOR and AND for FDD’s:

\[ f \oplus g = f_\overline{x} \oplus x^*(f_\overline{x} \oplus f_x) \oplus g_\overline{x} \oplus x^*(g_\overline{x} \oplus g_x) = (f_\overline{x} \oplus g_\overline{x}) \oplus x^*((f_\overline{x} \oplus f_x) \oplus (g_\overline{x} \oplus g_x)) \]

Yes!

\[ f \ast g = (f_\overline{x} \oplus x^*(f_\overline{x} \oplus f_x)) \ast (g_\overline{x} \oplus x^*(g_\overline{x} \oplus g_x)) = (f_\overline{x} \ast g_\overline{x}) \oplus x^*[f_\overline{x} \ast (g_\overline{x} \oplus g_x) \oplus (f_\overline{x} \oplus f_x) \ast g_\overline{x} \oplus (f_\overline{x} \oplus f_x)^*(g_\overline{x} \oplus g_x)] \]

No!

All 4 combinations have to be considered for the AND of 2 FDD's
OBDD and FDD for 4-bit adder
2.4 FDD's and OKFDD's

- **OKFDD's** *(Ordered Kronecker FDD's, Drechsler et al. 94)*
  - Allows any of the three types of decomposition for each variable
  - The type of decomposition is stored in a *decomposition type list* (DTL)

\[
f = a^*[(0 \oplus c^*(0 \oplus 1)) \oplus b^*((0 \oplus c^*(0 \oplus 1)) \oplus 1)] + \overline{a}^*[0 \oplus c^*(0 \oplus 1)]
\]
\[= a^*(c \oplus b\overline{c}) + \overline{a}^*c\]

![Diagrams showing Boole, p. Davio, and DTL for decomposition types](image)
2. Binary Decision Diagrams

2.4 FDD's and OKFDD's

- OBDD's/FDD's/OKFDD'S in comparison
  - OKFDD´s have OBDD´s and FDD´s as subclasses
  - OBDD´s:
    - AND, OR, XOR of two OBDD´s of size n and m requires max. n*m operations
  - FDD´s/OKFDD´s:
    - XOR requires max. n*m, but AND and OR may need exponentially many operations!
      - However: #nodes of FDD/OKFDD may be exponentially smaller than #nodes of the OBDD (and vice versa)
      - Important for logic synthesis
  - OKFDD´s: determining the decomposition-type list (DTL) is an additional problem
The OBDD size grows only linearly in #variables for many circuits (AND, OR, XOR, adders, ALU's, etc.)

Can all circuits be represented with linear (or polynomial) effort?

The theoretical answer is that there will never be any representation with this nice property for all circuits.

While the OBDD's are very compact representations for many classes of circuits they fail for others ...
Example: multiplier circuits

Word length: 4 6 8
#OBDD nodes: 150 2.183 10.766

Interest in other types of decision diagrams
2.5 Integer-Valued Decision Diagrams

- So far type $B^n \rightarrow B^m$, now: type $B^n \rightarrow \mathbb{Z}$:
  - MTBDD´s (Multi Terminal Binary Decision Diagrams, Clarke et al. DAC ´93)
  - BMD´s (Binary Moment Diagrams, Bryant/Chen DAC ´95)

$$
\begin{align*}
4a + b \\
\text{MTBDD}
\end{align*}
$$

$$
\begin{align*}
4a + b \\
\text{BMD}
\end{align*}
$$

Rule: variable = 1 $\rightarrow$ sum both branches
MTBDD:

\[ f = (1 - x) \cdot f_x^- + x \cdot f_x^+ \]

where +, -, * are the addition, subtraction and multiplication, respectively

BMD:

\[ f = f_x^- + x \cdot (f_x^+ - f_x^-) \]

HDD’s (Clarke/Zhao): combination of MTBDD/BMD, one decomposition types for each variable (~ OKFDD’s)
### Example of application (Fujita et al. ’96):

- **Vector-matrix operations employing MTBDD’s**

**Idea:**
- encode rows and columns by means of boolean variables
- elements ~ leafs

**Example 2*2 Matrix:**

<table>
<thead>
<tr>
<th></th>
<th>$f_{xy}$</th>
<th>$f_{xy}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_{xy}$</td>
<td>$f_{xy}$</td>
<td></td>
</tr>
<tr>
<td>$f_{xy}$</td>
<td>$f_{xy}$</td>
<td></td>
</tr>
</tbody>
</table>

![Binary Decision Diagram](image-url)
2. Binary Decision Diagrams

2.5 Integer-valued decision diagrams

- Type $B^n \rightarrow \mathbb{Z}$ and attributed edges
  - EVBDD's (Edge Valued Binary Decision Diagrams, Lai et al. ICCAD '93)
  - *BMD's (Multiplicative Binary Moment Diagrams, Bryant/Chen DAC '95)

```
0 1
\downarrow \quad \downarrow
0 1
\quad \downarrow
0

4a + 2b
EVBDD
```

```
0 1
\downarrow \quad \downarrow
0 1
\quad \downarrow
0 1 4

4a + 2b
*BMD
```

(rule: variable = 1 => add both branches, multiply by weight)

(rule: add weight)
2.5 Integer-valued decision diagrams

- **EVBDD:**
  \[ f = a + (1 - x)\bar{f}_x + x f_x \]
  
  where +, -, * are the addition, subtraction and multiplication, respectively

- **BMD:**
  \[ f = m^*(f_x + x(f_x - f_{\bar{x}})) \]

- **K*BMD´s (Drechsler EDTC´96):** one decomposition type for each variable (~ OKFDD´s, HDD´s), additive + multiplicative weights

- **PHDD (Chen/Bryant ICCAD´97):** multiplicative power hybrid decision diagrams for floating-point circuits
2. Binary Decision Diagrams

2.5 Integer-valued decision diagrams

- For *BMD´s we have for an edge without weight:

\[ f = f_x + x(f_x - f_{\overline{x}}) = f_{\overline{x}} + x f_x \]

- *BMD´s are canonical representations provided that:
  
  1. Rule:
2. Rule: the weight on an edge equals the gcd of the weights of the successor edges

A leave "n" is a 1 node with weight n
Example: BMD for $f = 4x + 2$

$$f = f_x + x(f_x - f_{\bar{x}})$$

$$= 2 + x(6 - 2),$$

$$\gcd(2, 4) = 2$$
Next example: BMD for $f = 3y + 4x + 2$

\[ f = f_y + y*(f_y - f_{\overline{y}}) \]

\[ = (4x + 2) + y*((4x + 5) - (4x + 2)) \]

\[ = (4x + 2) + y*3 \]
3. Rule: sign of $t$ is sign of left branch
For *BMD’s with range \{0, 1\}, the boolean operations can be reduced to integer addition, subtraction and multiplication:

- \( f \) \rightarrow 1 - f
- \( f \text{ and } g \) \rightarrow f\text{ xor } g
- \( f \text{ or } g \) \rightarrow f + g - f\text{ xor } g
- \( f \text{ xor } g \) \rightarrow f + g - 2f\text{ xor } g
Some example *BMD´s:

- 4 variable AND (a boolean function)
4 variable OR (a boolean function)
4 variable OR (a boolean function)

\[ x_4 = 0, x_3 = 0, \quad \begin{cases} 1 & x_2 = 1, x_1 = 0 \\ \end{cases} \]
Example: 2-Bit multiplication

\[ x_1, x_0 \quad y_1, y_0 \]

Result:
\[
(x_1 \cdot 2^1 + x_0 \cdot 2^0) \cdot (y_1 \cdot 2^1 + y_0 \cdot 2^0) = \\
2^0 \cdot x_0 \cdot (y_1 \cdot 2^1 + y_0 \cdot 2^0) + 2^1 \cdot x_1 \cdot (y_1 \cdot 2^1 + y_0 \cdot 2^0) + 2 \cdot x_1 \cdot (y_1 \cdot 2^1 + y_0 \cdot 2^0)
\]
Classification schema of decision diagrams (based on Minato ´96):

- decomposition type:
  - Shannon
  - p. Davio
  - mixed

<table>
<thead>
<tr>
<th></th>
<th>OBDD</th>
<th>FDD</th>
<th>OKFDD</th>
</tr>
</thead>
<tbody>
<tr>
<td>B^n → B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MTBDD</td>
<td>BMD</td>
<td>HDD</td>
<td></td>
</tr>
<tr>
<td>B^n → Z</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EVBDD</td>
<td>*BMD</td>
<td>K*BMD</td>
<td></td>
</tr>
<tr>
<td>B^n → Z, attributed edges</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2.6 Bit-Vector Expressions

- **Bit-vectors:**
  - Used for the compact representation of complex digital hardware
  - More adequate than single bits in many cases
    - Examples: data-paths, arithmetic circuits, register-transfer-level (rtl) descriptions, storage elements, ...
  - Provided by many hardware description languages (HDL's) as a basic data-type
### Example: specification of 74181 ALU

**Generic bit-vector function "A PLUS B"**

<table>
<thead>
<tr>
<th>S3</th>
<th>S2</th>
<th>S1</th>
<th>S0</th>
<th>M = H</th>
<th>M = L</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>F = not(A)</td>
<td>F = A</td>
</tr>
<tr>
<td>L</td>
<td>L</td>
<td>L</td>
<td>H</td>
<td>F = not(A+B)</td>
<td>F = A + B</td>
</tr>
<tr>
<td>L</td>
<td>L</td>
<td>H</td>
<td>L</td>
<td>F = not(A) B</td>
<td>F = A + not(B)</td>
</tr>
<tr>
<td>L</td>
<td>L</td>
<td>H</td>
<td>H</td>
<td>F = 0</td>
<td>F = MINUS 1</td>
</tr>
<tr>
<td>L</td>
<td>H</td>
<td>L</td>
<td>L</td>
<td>F = not(A B)</td>
<td>F = A PLUS A not(B)</td>
</tr>
<tr>
<td>L</td>
<td>H</td>
<td>L</td>
<td>H</td>
<td>F = not(B)</td>
<td>F = (A + B) PLUS A not(B)</td>
</tr>
<tr>
<td>L</td>
<td>H</td>
<td>H</td>
<td>L</td>
<td>F = A ⊕ B</td>
<td>F = A MINUS B MINUS 1</td>
</tr>
<tr>
<td>L</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>F = A not(B)</td>
<td>F = A not(B) MINUS 1</td>
</tr>
<tr>
<td>H</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>F = not(A)+B</td>
<td>F = A PLUS AB</td>
</tr>
<tr>
<td>H</td>
<td>L</td>
<td>L</td>
<td>H</td>
<td>F = not(A⊕B)</td>
<td>F = A PLUS B</td>
</tr>
<tr>
<td>H</td>
<td>L</td>
<td>H</td>
<td>L</td>
<td>F = B</td>
<td>F = (A + not(B)) PLUS AB</td>
</tr>
<tr>
<td>H</td>
<td>L</td>
<td>H</td>
<td>H</td>
<td>F = A B</td>
<td>F = A B MINUS 1</td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>L</td>
<td>L</td>
<td>F = 1</td>
<td>F = A PLUS A</td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>L</td>
<td>H</td>
<td>F = A + not(B)</td>
<td>F = (A + B) PLUS A</td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>H</td>
<td>L</td>
<td>F = A + B</td>
<td>F = (A + not(B)) PLUS A</td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>F = A</td>
<td>F = A MINUS 1</td>
</tr>
</tbody>
</table>

**M = L**

<table>
<thead>
<tr>
<th>Cn = H</th>
<th>Cn = L</th>
</tr>
</thead>
<tbody>
<tr>
<td>F = A PLUS 1</td>
<td>F = A PLUS 1</td>
</tr>
<tr>
<td>F = (A + B) PLUS 1</td>
<td>F = (A + not(B)) PLUS 1</td>
</tr>
<tr>
<td>F = (A + B) PLUS AB PLUS 1</td>
<td>F = (A + B) PLUS AB PLUS 1</td>
</tr>
<tr>
<td>F = A MINUS B</td>
<td>F = A MINUS B</td>
</tr>
<tr>
<td>F = A PLUS AB PLUS 1</td>
<td>F = A PLUS AB PLUS 1</td>
</tr>
<tr>
<td>F = A PLUS B PLUS 1</td>
<td>F = A PLUS B PLUS 1</td>
</tr>
<tr>
<td>F = (A + not(B)) PLUS AB PLUS 1</td>
<td>F = (A + not(B)) PLUS AB PLUS 1</td>
</tr>
<tr>
<td>F = AB</td>
<td>F = A PLUS A PLUS</td>
</tr>
<tr>
<td>F = (A + B) PLUS A PLUS 1</td>
<td>F = (A + B) PLUS A PLUS 1</td>
</tr>
<tr>
<td>F = (A + not(B)) PLUS A PLUS 1</td>
<td>F = (A + not(B)) PLUS A PLUS 1</td>
</tr>
<tr>
<td>F = A</td>
<td>F = A</td>
</tr>
</tbody>
</table>
Bit-vector functions are necessary for input/output specifications, i.e., for the abstraction from internal details.
### Typical bit-vector functions:

<table>
<thead>
<tr>
<th>Function</th>
<th>Meaning</th>
<th>Example:</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAE(A,N)</td>
<td>fan-out</td>
<td>FAE(1,3)=&quot;111&quot;</td>
</tr>
<tr>
<td>ADC(A,B,C)</td>
<td>addition</td>
<td>ADC(&quot;11&quot;,&quot;01&quot;,´1´)=&quot;101&quot;</td>
</tr>
<tr>
<td>ADD(A,B)</td>
<td>addition modulo</td>
<td>ADD(&quot;11&quot;,&quot;01&quot;)=&quot;00&quot;</td>
</tr>
<tr>
<td>INC(A)</td>
<td>increment</td>
<td>INC(&quot;111&quot;)=&quot;000&quot;</td>
</tr>
<tr>
<td>DCR(A)</td>
<td>decrement</td>
<td>DCR(&quot;111&quot;)=&quot;110&quot;</td>
</tr>
<tr>
<td>RSH(C,V)</td>
<td>right-shift</td>
<td>RSH(´0´,&quot;111&quot;)=&quot;011&quot;</td>
</tr>
<tr>
<td>LSH(V,C)</td>
<td>left-shift</td>
<td>LSH(&quot;111&quot;,´0´)=&quot;110&quot;</td>
</tr>
<tr>
<td>ROL(A)</td>
<td>rotate left</td>
<td>ROL(&quot;001&quot;)=&quot;010&quot;</td>
</tr>
<tr>
<td>ROR(A)</td>
<td>rotate right</td>
<td>ROR(&quot;010&quot;)=&quot;001&quot;</td>
</tr>
<tr>
<td>MPX1(A,S)</td>
<td>multiplexor 1. Dim.</td>
<td>MPX1(&quot;0010&quot;,&quot;10&quot;)=´1´</td>
</tr>
<tr>
<td>MINT(A,N)</td>
<td>minterm</td>
<td>MINT(A(1:2),0)=(not A(1)) and (not A(2))</td>
</tr>
<tr>
<td>GT(A,B)</td>
<td>A greater B</td>
<td>GT(&quot;100&quot;,&quot;010&quot;)=1</td>
</tr>
<tr>
<td>LESS(A,B)</td>
<td>A less B</td>
<td>LESS(&quot;100&quot;,&quot;010&quot;)=0</td>
</tr>
</tbody>
</table>
2.6 Bit-vector expressions

- Bit-vectors in VHDL
  - Type `bit_vector` predefined
    - signal `X: bit_vector (1 to 16)` or: `(16 downto 1)`
  - Selection of single elements `X(4)` or slices `X(2 to 4)`
  - Constant-denotation (B)"1001"
  - Assignments `X(2 to 4) <= X(8 to 10)`
  - Overloaded boolean primitives, e.g.,
    - "0101" AND "0011" = "0001"
  - Concatenation &, e.g., `X(2 to 4)&X(5 to 7) = X(2 to 7)`
  - ...
Verification problems:

- How to demonstrate the equality of arbitrary bit-vector expressions?
- Do we have to reason formally about tuples, etc.?
2. Binary Decision Diagrams

2.6 Bit-vector expressions

- Decision procedure: procedure to decide the truth of a statement in some domain
- For generic expressions (may contain expressions of arbitrary length), inductive reasoning is typically used
  - Example: prove $\text{ADD}(A, B) = \text{ADD}(B, A)$ for arbitrary vectors $A$ and $B$ of the same length
  - Typically a theorem prover is employed
    - You have to derive the proof in large part by yourself
    - The theorem prover checks if the proof is correct
      - Not automated, needs user interaction
For fixed-length expressions, the problem becomes much simpler

Example: prove \(\text{ADD}(A, B) = \text{ADD}(B, A)\) for 32-bit vectors \(A\) and \(B\)

Several decision procedures exist:

- The problem can be reduced to OBDD's
- The problem can be reduced to an integer-linear programming (ILP) problem
- A specific decision procedure was given by Cyrluk et al. (CAV´97) for a restricted repertoire of bit-vector functions
2.6 Bit-vector expressions

- Reduction to OBDD's ("bit-blasting"):  
  - Translate expression using bit-vector-functions into multi-level gate-networks  
    - e.g., A PLUS B, where A and B are two 4-bit vectors, is transformed into the gate-network of a four-bit adder  
  - Then as before!

```
vector expression 1  ->  gate network 1  ->  OBDD 1  = ?  
vector expression 2  ->  gate network 2  ->  OBDD 2
```
Technique offers the general possibility to carry out proofs involving complex bit-vector expressions

Examples:

\[ ADD(A, B) = ADD(B, A) \]

\[ (ADD(A, B) > ADD(B, C)) \rightarrow (A > C) \]

\[ (A > B) = NOT(ADD(0&A, 1&NOT(B))(1)) \]

where A, B, C have fixed length by declaration

Typically takes < 1 sec. for 64-bit vectors
A > B = NOT (ADD(0&A, 1&NOT(B)))(1)
2. Binary Decision Diagrams

2.6 Bit-vector expressions

- Example: verification of ALU´s
  - Verification of VHDL-specification using bit-vector-operations vs. network of standard-cells

<table>
<thead>
<tr>
<th>Wordlength</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU-time</td>
<td>1.0</td>
<td>1.5</td>
<td>2.8</td>
<td>6.6</td>
</tr>
</tbody>
</table>

- 32-Bit ALU: 2 * 32 boolean functions in up to 77 variables
Some references:
- Hassoun/Sasao (Eds.): Logic Synthesis and Verification, Springer
  — Book-Chapters on BDD's, SAT, Equivalence checking
● Written exam in the summer
  ↗ between 18. Juli and 7. October 2011
  ↗ please follow Doodle link
  http://www.doodle.com/y4igrfy6wcrmx24h