Computing Minimal DNF of Boolean Functions for Digital Implementations

Reni Banov

Zagreb University of Applied Sciences

20th Central European Conference on Cryptology, June 2020
Table of Contents

1 Introduction

2 Binary Decision Diagrams

3 New ordering heuristic

4 Conclusion
Introduction

Modern communication systems impose strong constraints to digital circuit implementation of the Boolean function, such as

- reliability
- performance
- cost
- security

Notice

Constraints do not always appear in this order of importance.

A possible solution is to minimize the Boolean function expression form

\[ f : \mathbb{B}^n \rightarrow \mathbb{B}, \]

by which the number of gates is reduced.
Which form to choose for the minimization?

- **ANF** – Reed-Muller expansion

  $$f(x_1, \ldots, x_n) = \bigoplus_{k} \left( \bigwedge_{i_k=1}^{n_i} x_{i_k} \right), \quad n_i \leq n$$

- **CNF** – Product-Of-Sums

  $$f(x_1, \ldots, x_n) = \bigwedge_{k} \left( \bigvee_{i_k=1}^{n} \xi_{i_k} \right), \quad \xi_i \in \{x_i, \overline{x_i}\}$$

- **DNF** – Sum-Of-Products

  $$f(x_1, \ldots, x_n) = \bigvee_{k} \left( \bigwedge_{i_k=1}^{n} \xi_{i_k} \right), \quad \xi_i \in \{x_i, \overline{x_i}\}$$
What is the minimal form of DNF?

The conjunction term $p$ containing some literals

$$p = \wedge \xi_i, \ i = 1, \ldots, k \leq n$$

having no sub terms

$$q \subset p : q(x) = 1 \implies f(x) = 1$$

is called prime implicants.

**Minimal DNF**

The DNF with a minimum number of prime implicants.

Such a minimal DNF form, being implemented in the digital circuit, is optimal in the restricted class of two-level digital circuits (disjunctions of conjunctions of literals) gates [Weg91].
Why DNF is selected?

NAND implementation

It is easy to implement a Boolean function with NAND gates only if converted from a DNF form.

For example, \( f(x) = (x_1 \land x_2) \lor (x_3 \land x_4) = (\overline{x_1 \land x_2}) \land (\overline{x_3 \land x_4}) \)
# Table of Contents

1. Introduction

2. Binary Decision Diagrams

3. New ordering heuristic

4. Conclusion
How to find a minimal DNF of the Boolean function?

- Karnaugh maps – expression simplification; appropriate for functions defined on less than ≈ 6 variables
- Minterms (Quine-McClusky [Weg91]) – conjunction term simplification; appropriate for functions defined on less than ≈ 15 variables
- Heuristic (Espresso [MSBS93]) – prime implicants heuristic search; appropriate for functions with less than ≈ 50 variables
- BDDs (Coudert [CM94]) – set cover implicit computation; applicable for functions with ≈ 100 variables or more

**Monotonic Boolean functions**

Minimal DNF of monotonic functions can be computed with BDDs according to the Rauzy approach [Rau93].
What are BDDs?

One of the only really fundamental data structures that came out in the last twenty-five years (D.E. Knuth, 2008)

Bryant [Bry86]

The Boolean Decision Diagrams (BDDs) a variant of directed acyclic graphs (DAGs) used for Boolean functions representation.

Two-terminal DAG based on Shannon identity

\[ f(x) = (x_i \land f_{x_i}(x)) \lor (\overline{x_i} \land f_{\overline{x_i}}(x)) \]

built from vertices representing the If-Then-Else (ITE) construct.
Why and when BDDs?

Canonicity

The BDDs are a *canonical* representation of the Boolean function.

but, they are *sensitive* to the selection of the variable order

1Source: PyEDA documentation [Dra20]
Any strong results for BDDs? I

**Hardness of variable ordering**

- The problem of finding an optimal order is *NP-hard* [TY00], and even by improving a variable order, the problem is *NP-complete* [BW96],
- There are Boolean functions which have an exponential size BDD for every ordering [Bry86]

---

**Friedman-Supowit [FS90]**

Let \( I \subseteq \{1, \ldots, n\}, k = |I|, v \in I \), then there is a constant \( c \) such that for each \( \pi \in \Pi(I) \) satisfying \( \pi[k] = v \), we have

\[
\text{cost}_v(f, \pi) = c.
\]
Any strong results for BDDs? II
Table of Contents

1 Introduction

2 Binary Decision Diagrams

3 New ordering heuristic

4 Conclusion
Parse tree

First step: to build the parse tree of the Boolean function expression

Option: to build a Boolean Expression Diagram (BED) to remove redundant subexpressions [AH02]

\[ f(x) = x_0 \oplus x_1 \oplus (x_0 \land x_1) \oplus (x_2 \land x_3) \]
A new variable ordering heuristic

**Input:** $f$ top node in parse tree or BED  
**Output:** $\pi$ a new variable order for top node

1. Initialize an empty set of orders $\text{Orders} \leftarrow \emptyset$  
2. For each node $\in \text{DFSOrder}(f)$ do:
   - If node is Variable then:
     1. Set $x_i \leftarrow \text{Variable}(node)$
     2. Set $\pi_i \leftarrow \{(x_i, \text{count} : 1)\}$
     3. Insert $\text{Orders.insert}(\text{key:node, value:}\pi_i)$
   - Else:
     1. /* node is boolean operator */
     2. Set $\pi_l \leftarrow \text{Orders.findValue(\text{key:LeftChild(node)})}$
     3. Set $\pi_r \leftarrow \text{Orders.findValue(\text{key:RightChild(node)})}$
     4. Set $\pi \leftarrow \text{MergeOrders}(\pi_l, \pi_r)$  
     5. Insert $\text{Orders.insert}(\text{key:node, value:}\pi)$
3. $\pi \leftarrow \text{Orders.findValue(\text{key:f})}$
4. Return $\pi$  
   // final order of variables for $f$ node
MergeOrders($\pi_l, \pi_r$)

**Input:** $\pi_l, \pi_r$ two variable orders for merging

1. $\pi \leftarrow \emptyset$
2. $(x_l, c_l) \leftarrow \text{First}(\pi_l)$
3. $(x_r, c_r) \leftarrow \text{First}(\pi_r)$

while $(x_l \neq \emptyset) \& (x_r \neq \emptyset)$ do

    /* Heuristic criteria */

    if $(c_l + 2 \times c_r) \leq (c_r + 2 \times c_l)$ then
        Append_If_Not_Present($\pi, \{x_l, \text{count: } c_l + 2 \times c_r\}$)
        $(x_l, c_l) \leftarrow \text{Next}(\pi_l)$
    end
    else
        Append_If_Not_Present($\pi, \{x_r, \text{count: } c_r + 2 \times c_l\}$)
        $(x_r, c_r) \leftarrow \text{Next}(\pi_r)$
    end

end

if $x_l \neq \emptyset$ then
    Append($\pi, \text{from: } x_l, \pi_l$)  // Left finished?
end

if $x_r \neq \emptyset$ then
    Append($\pi, \text{from: } x_r, \pi_r$)  // Right finished?
end

return $\pi$  // Two orders merged
### Results

*Table: Preliminary results*

| example | variables | operators | \(|BDD|\)^1 | \(|DNF|\)^2 |
|---------|-----------|-----------|-------------|-------------|
| ex1     | 13        | 12        | 15          | 11          |
| ex2     | 70        | 53        | 254         | 36.292      |
| ex3     | 43        | 32        | 57          | 1.043       |
| ex4     | 66        | 50        | 281         | 32.369      |
| ex5     | 44        | 35        | 49          | 784         |
| ex6     | 98        | 141       | 264         | 960         |
| ex7     | 18        | 19        | 31          | 46          |
| ex8     | 61        | 84        | 2.481       | 46.188      |
| ex9     | 194       | 158       | 3.048       | 34.477.555  |

1. count of nodes in BDD built with heuristic order  
2. count of prime implicants for minimal DNF form
Table of Contents

1 Introduction

2 Binary Decision Diagrams

3 New ordering heuristic

4 Conclusion
Why minimal DNF and BDDs?

Some reasons for minimal DNF in digital implementations
- Reduced cost due to the minimal number of gates used
- Improved performance due to minimized total delay
- Design constraints fulfilled (die size, thermal, . . .)
- More reliable circuits

What else can be done with BDDs besides the DNF minimization?
- Expressing (and solving) of a 0/1 optimization problems
- Representing a structure function of complex systems
- Could properties of the Boolean functions be derived from topological properties of the BDD DAGs
- Other ideas?
Thank You!
References I


Bryant, *Graph-based algorithms for boolean function manipulation*, IEEE Transactions on Computers **C-35** (1986), no. 8, 677–691.


References II


References III
