Boolean Logic I *

Jens Lechtenbörger

IT Systems, Summer Term 2024

We start our journey towards a computer with Boolean logic. This topic is covered in two presentations, of which this is the first one.

1 Introduction

Let us begin with a brief look at the core of our topic and its learning objectives, followed by a recap.

1.1 Today's Core Question

- How to implement Boolean functions systematically?

Our goal is to implement Boolean functions as building blocks for our computer. Thus, we explore how to implement them systematically.

1.2 Learning Objectives

- Generate DNF from truth table
- Simplify Boolean expressions
- Transform DNF graphically to All-Nand circuit
- Build chips of project 1

You will learn to implement Boolean functions starting from truth tables. In particular, you will see how to derive the Disjunctive Normal Form of a Boolean function from its truth table.

Based on laws of Boolean logic, Boolean expressions can be simplified before they are implemented as circuits, and Boolean expressions can be transformed into ones that contain only Nand operations.

Finally, you implement the chips of Project 1 as building blocks for the computer of Nand to Tetris.

1.3 Retrieval Practice

- Recall
  - 0 and 1 represent False and True
  - Rows of truth tables ordered by binary counting

*This PDF document is an inferior version of an OER HTML page; free/libre Org mode source repository.

- Write down the truth table for `Nand` now

Recall that you saw truth tables as specifications for Boolean functions, in particular for Nand.

Write down its truth table now. In what order do you write down the rows?

Agenda

- Part 1
  - Introduction
  - Boolean Logic
- Break for self-study
- Part 2
  - Boolean Circuits
  - Sample Transformations
  - Multiplexors and De-Multiplexors, Project 1
  - Conclusions

The material for Boolean Logic is as follows.

After this introduction, we delve into Boolean Logic, starting from a definition of Boolean algebra. In particular, you see how to derive a canonical representation from a function's truth table.

Then, you might want to take a break and apply covered techniques.

Afterwards, in part 2, we look at circuits, sample transformations, and chips of Project 1 of Nand to Tetris.

2 Boolean Logic

Boolean logic is grounded is a specific mathematical structure, namely in an algebra whose definition we see next.

2.1 Boolean Algebra

- Mathematical view on logic, going back to (Boole 1847)

  Boolean algebra formalizes a branch of logic in mathematical terms. It is named after George Boole, who wrote two seminal books on this topic in the middle of the nineteenth century.

  - **Algebra** $B = \{0, 1\}$ with **three operations** for $x, y \in B$:

    As you see here, Boolean algebra is defined with three operations over a set with two elements, namely 0 and 1, which represent truth values.

    * **Or**: $x \vee y = \max(x, y)$ $(= x + y = \mathrm{Or}(x, y))$

      The Or operation can be defined as maximum of two values: If at least one value is 1, the result is 1. You see typical symbols that are used to denote this operation.

      Subsequently, we usually use the plus sign. Consequently, we may refer to the result of the operation as **sum** (with the special interpretation that the "sum" of one and one equals one).

* **And**: $x \wedge y = \min(x, y)$ $(= x \cdot y = xy = \text{And}(x, y))$
  The And operation can be defined as minimum of two values: If at least one value is 0, the result is 0.
  Again, you see typical symbols that are used to denote this operation. Subsequently, we usually use multiplication, which does not require any symbol. Consequently, we may refer to the result of the operation as **product**.
* **Not**: $\neg x = 1 - x$ $(= \bar{x} = \text{Not}(x))$
  The Not operation flips, or inverts, or negates a value.
  Subsequently, we usually use an overline to indicate a negation.

– Precedence: Not binds strongest, then And, then Or
  * E.g.: $\neg x_1 x_2 + \neg x_3 = ((\neg x_1)x_2) + (\neg x_3)$
  * Use parentheses for more complex cases or if in doubt

  As usual, parentheses can be used to structure more complex expressions. Without parentheses, the precedence rules shown here apply.

– Lots of laws/equalities can be proven, see later slide
  Based on this definition, lots of laws or equalities among Boolean expressions can be proven. This will be revisited on later slides.

2.2 Boolean Functions and Truth Tables

- **Boolean functions** have arguments and result in B

  – k-ary function $f : B^k \to B$ for $k \geq 0$
    Boolean functions can take any number of Boolean arguments and return a Boolean value. Although the special case of 0 arguments is included here, we do not consider such degenerate functions subsequently. In case you are curious, a function without arguments denotes a constant.

    * E.g., $k = 3$: $f_0(x_1, x_2, x_3) = x_1 x_2 + \bar{x_3}$
      Consider the Boolean function f_0, which takes 3 arguments. Clearly, this function will produce a uniquely defined output value for each of the possible combinations of input values.
    * Can use **truth table** with 2^k rows and $k+1$ columns to represent f

x_1	x_2	x_3	$f_0(x_1, x_2, x_3)$
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

      · See table to right for f_0
      Please convince yourself that the values in the final column of the truth table here are those produced by the expression defining f_0.
    * Interpret argument in row as binary number, called **index**: $(x_1 x_2 \ldots k_k)_2$
      · For $k = 3$, count from $0 = (000)_2$ to $2^k - 1 = 7 = (111)_2$
      Note that the truth table lists **all** possible input combinations for 3 variables. Moreover, each input combination can be understood as binary number in the range from 0 to 7. Concerning terminology, this binary number is called **index**.
    * Index i with $f(i) = 1$ is called **positive index**

· 0, 2, 4, 6, 7 are positive indices of f_0

If the output value produced for an index is 1, the index is a **positive** one. (Otherwise, the index is negative.)
Please verify that the indices listed here are the positive ones for f_0.

* **Minterm**: Product/And-operation, in which each variable occurs once, potentially negated
  · E.g., $x_1x_2x_3$, $x_1x_2\bar{x_3}$

Again concerning terminology, a **minterm** is a product that combines **all** input variables, where each variable may be negated or not. Examples are shown here.

2.3 Boolean Function as Sum of Products

- Representation **Theorem**
  - Every Boolean function f can be uniquely represented as sum of all minterms for its positive indices I, i.e.: $f = \sum_{i \in I} m_i$

    An important theorem of Boolean logic states that every Boolean function can be uniquely represented as **sum of products**. More precisely, we sum up minterms for the positive indices.

    * **Minterm for index** i, denoted m_i: Variable occurs negated if its value in row i is 0; otherwise, variable without negation

      In that sum, the binary representation of an index number specifies which variables to negate in a minterm: If the bit for a variable is 0, it is negated. If the bit is 1, the variable is not negated.

      · E.g., $k = 3$: $m_0 = \bar{x_1}\bar{x_2}\bar{x_3}$, $m_6 = x_1x_2\bar{x_3}$

x_1	x_2	x_3	$f_0(x_1, x_2, x_3)$
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

      For example, in minterm 0 the binary representation contains three zeros. Hence, all three variables are negated.
      As another examples, as 6 is 1 1 0 in binary, in minterm 6 the first two variables are not negated, while the third one is negated.

    * Theorem applied to f_0 with $I = \{0, 2, 4, 6, 7\}$
      · $f_0 = m_0 + m_2 + m_4 + m_6 + m_7$
      · $f_0 = \bar{x_1}\bar{x_2}\bar{x_3} + \bar{x_1}x_2\bar{x_3} + x_1\bar{x_2}\bar{x_3} + x_1x_2\bar{x_3} + x_1x_2x_3$
      Here you see the result of applying the theorem to f_0.
    * Such a sum (logical Or) of products (logical And) is also called **Disjunctive Normal Form** (DNF)
      · Disjunction is another word for logical Or
      · (Conjunction is another word for logical And; conjunctive normal form exists as well...)

    As we sum up minterms, which are products of variables, the result is called **sum of products**. As disjunction is another word for logical Or, the result is also called **disjunctive normal form**, DNF for short.
    Besides, conjunctive normal forms exists as well, but we do not consider them.

2.3.1 Observations

- Previous theorem implies that **every** Boolean function can be expressed using only And, Or, Not
  - DNF is a **canonical** representation
- We say that {And, Or, Not} is **functionally complete**
- In Nand To Tetris, along the way (by construction) we prove that {Nand} is functionally complete
  - Recall: `Not(x) = Nand(x, x)`, `And(x, y) = ...`

The theorem on the previous slide implies that **every** Boolean function can be expressed using only the set of operations And, Or, Not. As these operations are applied in a special order, DNF is also called a **canonical** representation for Boolean functions.

Moreover, we call sets of operations which are sufficient to represent any Boolean function **functionally complete**.

One may wonder whether smaller sets are also functionally complete, and it turns out that the answer is "yes". For example, in Nand to Tetris, we start from Nand alone to build chips for other Boolean operations. This process can be understood as constructive proof for the fact that Nand alone is functionally complete.

2.4 Laws of Boolean Algebra

- Sample laws
  - Commutativity: $xy = yx$, $x + y = y + x$
  - Associativity: $(xy)z = x(yz)$, $(x + y) + z = x + (y + z)$
  - Fusion: $(x + y)x = x$, $(xy) + x = x$
  - Distributivity: $x(y + z) = xy + xz$, $x + yz = (x + y)(x + z)$
  - Complements: $x + \bar{x} = 1$, $x\bar{x} = 0$
  - De Morgan: $\overline{x + y} = \bar{x}\bar{y}, \overline{xy} = \bar{x} + \bar{y}$
  - $x + 0 = x$, $x + 1 = 1$, $x \cdot 0 = 0$, $x \cdot 1 = x$
  - $x = x + x = xx = \bar{\bar{x}}$
  - Use above laws for simplifications, proofs of equality

A variety of rules or laws allows transforming Boolean expressions in equality preserving ways. We can use such rules to simplify Boolean expressions before we implement them as circuits.

Here, you see selected rules, all of which come with strict mathematical proofs. Some rules, such as Commutativity and Associativity, may appear to be obvious, while others may require more thought. In any case, one can use truth tables to prove all of these equations. In fact, subsequently, you will see a sample proof for one of the De Morgan rules.

2.5 Sample Proof with Truth Table

- Proof for one of the De Morgan rules: $\overline{xy} = \bar{x} + \bar{y}$

  In general, We can use truth tables to prove the correctness of laws and rules of Boolean logic. Let us see an example for one of the De Morgan rules. It says that the Negation of x And y is the same as Not x Or Not y.
  - Sub-expressions in truth table

x	y	xy	$\overline{\boldsymbol{xy}}$	$\bar{x}$	$\bar{y}$	$\boldsymbol{\bar{x}+\bar{y}}$
0	0	0	**1**	1	1	**1**
0	1	0	**1**	1	0	**1**
1	0	0	**1**	0	1	**1**
1	1	1	**0**	0	0	**0**

We create one row for each possible input combination, resulting in 4 rows. We add columns for the left and right hand side of the expression as well as for intermediate expressions. Then, we fill out the table and check whether the values in the columns for the left and right hand side are equal.

* Note that column values corresponding to left- and right-hand side of rule are identical (highlighted)
* Thus, expressions are equal (as Boolean functions)
  In this case, the values in the columns for the left and right hand side are equal. Thus, the functions represented by both columns are equal.

3 Self-Study Tasks

Maybe take a break.

3.1 DNF

- What do the gates And, Or, Nand do?
- Consider $f_1(x_1, x_2, x_3) = \bar{x_1}\overline{(\bar{x_2} + \bar{x_3})} + x_1x_3$.
  - Write down the truth table for f_1. Then determine its positive indices and its DNF as sum of minterms.
  - Simplify the DNF of f_1 using laws of Boolean algebra. (The next presentation contains examples.)

You can verify your answers in Learnweb.
To support your learning, pause here to work on the given task.

3.2 Sample proofs

- Prove laws of Boolean algebra that seem surprising
  - E.g., second De Morgan rule

Revisit the laws of Boolean algebra and prove those that seem surprising.
Maybe start with the De Morgan rule whose proof is not covered in this presentation.

Bibliography

Boole, George. 1847. *The Mathematical Analysis of Logic, Being an Essay towards a Calculus of Deductive Reasoning*. https://www.gutenberg.org/files/36884/36884-pdf.pdf.
The bibliography contains references used in this presentation.

License Information