

Section 8.2: Logic Networks

April 26, 2022

Abstract

We examine the relationship between the abstract structure of a Boolean algebra and the practical problem of creating (optimal!) logic networks for solving problems¹. There is a fundamental equivalence between Truth Functions, Boolean Expressions, and Logic Networks which allows us to pass from one to the other. While a problem might be easiest formulated in terms of a truth function, we might then recast it as a Boolean expression to then feed into a logic network. Then Boolean algebra provides us with a simple mechanism by which to simplify the expressions, and hence to simplify the underlying logic network.

We'll examine the binary adder (and half-adder) as a particular example, which will later be implemented as a Finite State Machine.

1 An Example Application, and Fundamental Parallels

Example: Two light switches, one light!

The problem is as follows: A light at the bottom of some stairs is controlled by two light switches, one at each end of the stairs. The two switches should be able to control the light **independently**. How do we wire the light?

- A Truth Function: $f(s_1, s_2) = L$

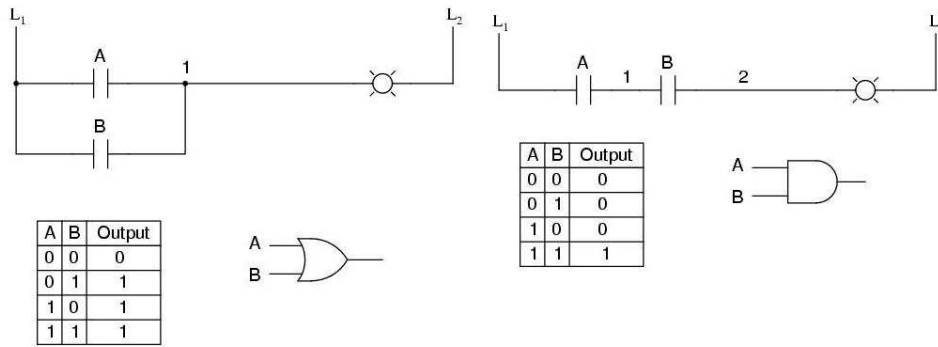
¹From our text: "In 1938 the American mathematician Claude Shannon perceived the parallel between propositional logic and circuit logic and realized that Boolean algebra could play a part in systematizing this new realm of electronics."

- A Boolean Expression (find two equivalent Boolean expressions)

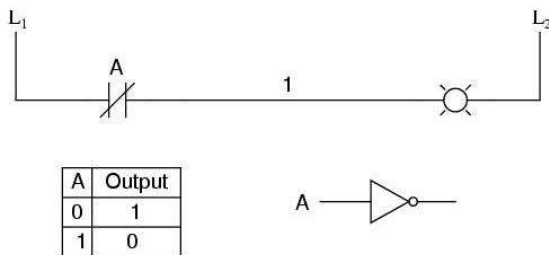
- A Logic Network (Basic Components, Mechanics, and Conventions)

- Input or output lines are not tied together except by passing through gates:

- OR gate
- AND gate



- NOT gate



- Lines can be split to serve as input to more than one device.

- There are no loops, with output of a gate serving as input to the same gate. (feedback).
- There are no delay elements.

Figure 8.6, p. 638, shows how to wire an “or” – we do it in parallel (“and” is wired in series).

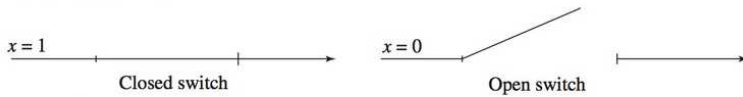


Figure 8.5

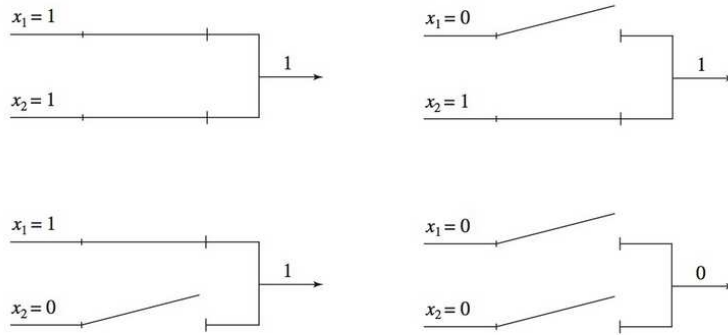


Figure 8.6

2 Applications

2.1 Converting Truth Tables to Boolean Expressions (Canonical Sum-of-Products Form)

Example: Practice 11, p. 645

PRACTICE 11

- Find the canonical sum-of-products form for the truth function of Table 8.5.
- Draw the network for the expression of part (a).

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

Example: Exercise 15, p. 657 Find the canonical sum-of-products

form for the truth function:

15.

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

(notice that you can easily simplify that canonical sum-of-products, using some Boolean algebra.)

2.2 Converting Boolean Expressions to Logic Networks

Example: Practice 11, p. 645 (reprise)

PRACTICE 11

- Find the canonical sum-of-products form for the truth function of Table 8.5.
- Draw the network for the expression of part (a).

TABLE 8.5

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

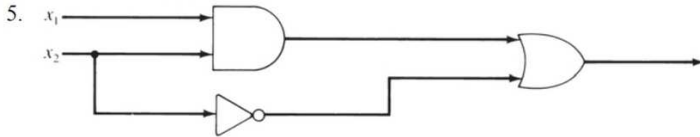
Example: Exercise 2, p. 655 Write a truth function and construct a

logic network using AND gates, OR gates, and inverters for the Boolean expression $(x_1 + x_2) + x_1'x_3$

2.3 Converting Logic Networks to Truth Functions or Boolean Expressions

Example: Exercise 5, p. 655

For Exercises 5–8, write a Boolean expression and a truth function for each of the logic networks shown.



2.4 Simplifying Canonical Form

We can use properties of Boolean algebra to simplify the canonical form, creating a much simpler logic network as a result.

Example: Practice 11, p. 645 (reprise)

PRACTICE 11

- Find the canonical sum-of-products form for the truth function of Table 8.5.
- Draw the network for the expression of part (a).

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

Wouldn't it be nice if there were some systematic way of doing this? That's the subject matter of the next section! We'll see two different ways to simplify a canonical sum of products.

2.5 An example: Adding Binary numbers

2.5.1 Half-Adders

Half-Adder: Adds two binary digits.

$$s = x_1'x_2 + x_1x_2'$$

$$c = x_1x_2$$

s is the result of an “XOR” operation (exclusive or) of the two inputs, whereas c is the product of the two inputs. Note, however, that the half-adder doesn’t implement s in this way: instead,

$$s = (x_1 + x_2) \cdot (x_1 x_2)'$$

Questions:

- a. How?
- b. Why?

2.5.2 Full-Adders

Full-Adder: Adds two digits plus the carry digit from the preceding step (which we can create out of two half-adders!).

- Given the preceding carry digit c_{i-1} , and binary digits x_i and y_i .
- We’ll use a half-adder to add x_i to y_i , obtaining write digit σ and carry digit γ .
- Then use a half-adder to add the carry digit c_{i-1} to σ ; the write digit is s_i , and call the carry digit c .
- To get the carry digit c_i , compare the carry digits c and γ : if either gives a 1, then $c_i = 1$ (so it’s an “or”).

Let’s derive all that from the truth functions, representing the sum from the full-adder:

c_{i-1}	x_i	y_i	c_i	s_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

So the canonical sum of products forms of each function are

$$\begin{aligned}
 s_i(c_{i-1}, x_i, y_i) &= && c'_{i-1}x'_iy_i \\
 &+ && c'_{i-1}x_iy'_i \\
 &+ && c_{i-1}x'_iy'_i \\
 &+ && c_{i-1}x_iy_i \\
 &= && c'_{i-1}(x'_iy_i + x_iy'_i) + c_{i-1}(x'_iy_i + x_iy'_i)'
 \end{aligned}$$

and

$$\begin{aligned}
 c_i(c_{i-1}, x_i, y_i) &= && c'_{i-1}x_iy_i \\
 &+ && c_{i-1}x'_iy_i \\
 &+ && c_{i-1}x_iy'_i \\
 &+ && c_{i-1}x_iy_i \\
 &= && x_iy_i + c_{i-1}(x'_iy_i + x_iy'_i)
 \end{aligned}$$

We recognize these quantities in terms of half-adders:

- We recognize the write digit σ and the carry digit γ of the half-adder of x_i and y_i .
- Then s_i is just the write digit s of the half-adder of c_{i-1} and σ ;
- Meanwhile, c_i is the sum of γ and the carry digit c of the half-adder of c_{i-1} and σ .
- That is illustrated in this sad figure I once drew:

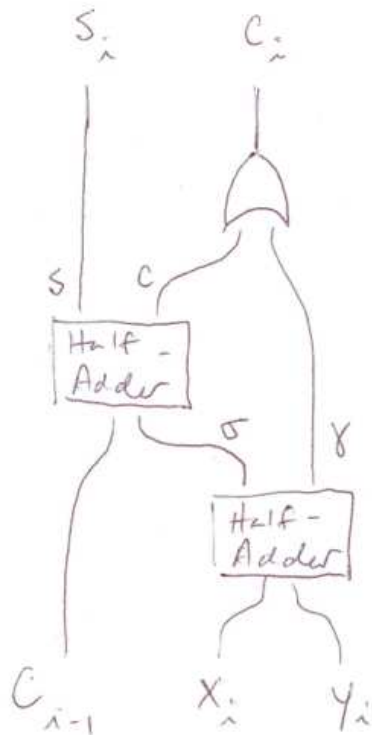


Figure 1: The full-adder takes input digits x_i and y_i , as well as the carry digit c_{i-1} from the previous step and computes write digit s_i and carry digit c_i . Then do it again!

Example: Practice 12, p. 650 Trace the operation of the circuit as it adds 101 and 111.