Section 8.3: Minimization

April 20, 2021

Abstract

The word "minimization" in the title of this section refers to our pursuit of simplified but equivalent Boolean expressions, which we think of as representing hardware (logic networks). Our objective is to start with the canonical form derived from a truth table, and reduce it to a simpler expression (generally also a sum of products), which is easier (cheaper, faster) to implement in hardware.

We examine two different techniques for accomplishing this: the Karnaugh map, and the Quine-McCluskey procedure. The first is aesthetically pleasing, but limited to a few Boolean variables; the second can be generalized to handle any number of variables, and can be coded up relatively easily.

1 Overview

In Section 8.2, we discovered that there is a relatively simple way of passing back and forth between representations of a truth function (table), Boolean expression, and logic network. In particular, we saw that, given a truth table, it is simple to construct a Boolean expression (the canonical form) which is a sum of products. Unfortunately, this expression is often much more complicated than necessary - it can be simplified, or minimized.

In this section, we consider two methods for dealing with truth tables, and turning them into simpler Boolean expressions. The **Karnaugh Map** turns a truth table into an equivalent "matrix", which we simplify using known visual patterns; and the Quine- McCluskey procedure plays a similar pattern-matching game, making selective deletions to trim down the canonical form to a simpler Boolean expression.

2 Simplification and the Karnaugh Map

The Karnaugh Map[**?**] is named for Maurice Karnaugh, who is still alive (as of 2021) – just 96 years old!

2.1 Simplification Rules

A couple of simple equivalence rules make our lives easier (and expressions smaller):

$$xy + x'y = y \quad (1)$$
$$x + x'y = x + y \quad (2)$$

Rule (1) follows by standard distributivity and the properties of complements and identity:

$$xy + x'y = (x + x') \cdot y = 1 \cdot y = y$$

Rule (2) follows by "the curious distributive rule" and the properties of complements and identity:

$$x + x'y = (x + x') \cdot (x + y) = 1 \cdot (x + y) = x + y$$

We demonstrate the utility of these two rules in the following simplification:

Example 19, p. 664. Consider the canonical form given by

$$\begin{aligned}
E &= x_1x_2x_3 \\
&+ x_1'x_2x_3 \\
&+ x_1'x_2x_3' \\
&= x_2(x_1x_3 + x_1'x_3) + x_1'x_2x_3' \\
&= x_2(x_1 + x_1')x_3 + x_1'x_2x_3' \\
&= x_2x_3 + x_1'x_2x_3' \\
&= x_2(x_3 + x_1'x_3') \\
&= x_2((x_3 + x_1')(x_3 + x_3')) \\
&= x_2(x_3 + x_1') \\
&= x_2x_3 + x_2x_1'
\end{aligned}$$

optimal

This expression is simplified as a sum of products.

2.2 Karnaugh Map Examples

We can illustrate our two equivalence rules with tables known as the Karnaugh maps associated with the expressions. Here's $x_1x_2' + x_1x_2 = x_1$:

	x_1	x_1'
x_2	1	
x_2'	1	

The terms from the table having a 1 in them indicate when the expression (given by the canonical sum of products) $x_1x_2 + x_1x_2'$ is true (or when the function is true). Giving this a little thought, we can interpret this table as telling us that the variable x_2 is irrelevant – i.e. $x_1x_2' + x_1x_2 = x_1$.

And here's the second identity: $x + x'y = x + y$ (notice that we've already simplified the canonical sum of products, using $xy + xy' = x$).

	x	x'
y	1	1
y'	1	

Note that the canonical sum of products of the 1-terms is equivalent to the negation of the 0-term: $(x' \cdot y')' = x + y$. In this case, it appears more efficient to work with the 0-term, and then just "demorgan it" to get the solution. However the 0-term requires three negations and a dot, whereas the "undemorganized term" requires one simple sum.

Here's a two-variable example (the XOR from the half-adder of section 8.2): $x_1x_2' + x_1'x_2$

	x_1	x_1'
x_2		1
x_2'	1	

Example: Example 19, p. 664: $x_1x_2x_3 + x_1'x_2x_3 + x_1'x_2x_3'$

	x_1x_2	x_1x_2'	$x_1'x_2'$	$x_1'x_2$
x_3	1			1
x_3'				1

While the position of the Boolean variables in the 2x2 example above is arbitrary, not so for the column labels of the example above: notice that there is a single change in the Boolean expressions as you read across the top. Note also that the far left and right expressions are also only different by one change. We could wrap this table and put it onto a cylinder.

A four-variable example.

	x_1x_2	x_1x_2'	$x_1'x_2'$	$x_1'x_2$
x_3x_4	1			1
x_3x_4'				1
$x_3'x_4'$				1
$x_3'x_4$				1

In this case, there is nothing arbitrary about either row- or column-labels: you could wrap top to bottom and right to left, which means that this table could be wrapped onto a torus (or donut shape).

In this section we study a method for simplification, not just representation, so how do we simplify?

	x_1	x_1'
x_2	1	1
x_2'		

$\Longrightarrow x_1x_2 + x_1'x_2 = x_2$

	x_1	x_1'
x_2		1
x_2'		1

$\Longrightarrow x_1'x_2 + x_1'x_2' = x_1'$

$x_1'(x_2 + x_2') = x_1' \cdot 1 = x_1'$

	x_1	x_1'
x_2	1	1
x_2'	1	1

$\Longrightarrow x_1x_2 + x_1x_2' + x_1'x_2 + x_1'x_2' = 1$

$x_1(x_2 + x_2') + x_1'(x_2 + x_2') = x_1 \cdot 1 + x_1' \cdot 1 = x_1 + x_1' = 1$

Check out this trick (idempotence):

	x_1	x_1'
x_2		1
x_2'	1	1

$\Longrightarrow x_1'x_2 + x_1x_2' + x_1'x_2' = x_1'x_2 + x_1x_2' + (x_1'x_2' + x_1'x_2') = x_1' + x_2'$

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

Notice, however, that this is really the same as rule (2) above:

$$x_1'x_2 + x_1x_2' + x_1'x_2' = x_1'(x_2 + x_2') + x_1x_2' = x_1' + x_1x_2' = x_1' + x_2'$$

Example: Example 19, p. 664 (Again! – now let's simplify):
$x_1x_2x_3 + x_1'x_2x_3 + x_1'x_2x_3'$

	x_1x_2	x_1x_2'	$x_1'x_2'$	$x_1'x_2$
x_3	1			1
x_3'				1

$x_1x_2x_3 + x_1'x_2$

$= x_2x_3 + x_1'x_2$

$x_2(x_1' + x_3)$

Note that we need to wrap to do this one; furthermore see how much more simply we simplify this expression than we did up top: we use idempotence, then the simplification rule (1) twice (not needing the second, its role being handled by the idempotence).

There may be multiple simplifications of a Boolean expression:

Example: Exercise #1, p. 678

	x_1x_2	x_1x_2'	$x_1'x_2'$	$x_1'x_2$
x_3			1	1
x_3'	1	1		1

$E = x_1'x_3 + x_1'x_2 + x_1x_3'$

$E = x_1'x_3 + x_2x_3' + x_1x_3'$

$= x_1'x_3 + (x_1 + x_2)x_3'$

We may need to look for quads, rather than pairs:

Example: Exercise #5, p. 678

	x_1x_2	x_1x_2'	$x_1'x_2'$	$x_1'x_2$
x_3x_4		1		
x_3x_4'		1	1	1
$x_3'x_4'$	1	1	1	
$x_3'x_4$		1		

$E = x_1x_2' + x_2'x_4' + x_1'x_3x_4' + x_1x_3x_4'$

Still a sum of products, since 2021!

3 Simplification and the Quine-McCluskey procedure

Typically we don't use Karnaugh Maps when we get beyond four variables; but Karnaugh suggested how we might proceed if we have up to six, using his "3-dimensional plastic framework for maps":

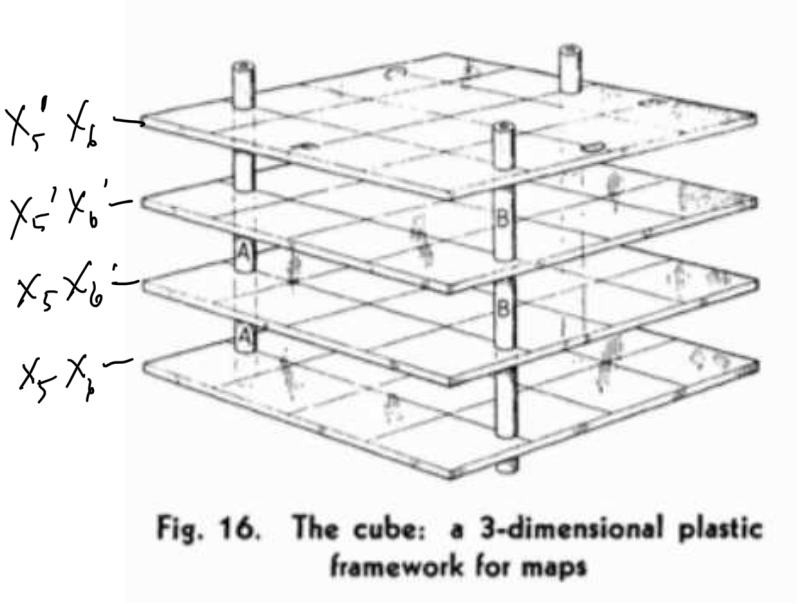


Fig. 16. The cube: a 3-dimensional plastic framework for maps

Figure 1: From Karnaugh's article *The Map Method for Synthesis of Combinational Logic Circuits*[**?**].

Very cool – but not very practical, frankly.

Edward McCluskey died recently (2016), at the age of 86. He was a bridge back in time to Bertrand Russell and Alfred North Whitehead, because of his work with Willard Van Orman Quine (who died in 2000, at the age of 92 – his doctorate was supervised by Whitehead). "Ed was born October 16, 1929,... and in 1956 earned his doctorate in electrical engineering from the Massachusetts Institute of Technology, where he developed what became known as the Quine-McCluskey algorithm for the design of minimum-cost digital logic circuits. This was the first systematic approach to logic circuit design and is still used and taught today."[**?**]

About the algorithm, Quine said "I do not do anything with computers, although one of my little results in mathematical logic has become a tool of the computer theory, the Quine McCluskey principle. And corresponds to terminals in series, or to those in parallel, so that if you simplify mathematical logical steps, you have simplified your wiring. I arrived at it not from an interest in computers, but as a pedagogical device, a slick way of introducing that way of teaching mathematical logic."[**?**]

TABLE 8.16

Number of 1s	x_1	x_2	x_3	x_4	
Three	1	0	1	1	✓
Two	0	1	1	0	✓
	0	1	0	1	3
	0	0	1	1	✓, 4, 5
One	1	0	0	0	6
	0	0	1	0	✓ 4, 7
	0	0	0	1	3, 5, 8
None	0	0	0	0	6, 7, 8

(a)

Number of 1s	x_1	x_2	x_3	x_4	
Two	–	0	1	1	
One	0	–	1	0	
	0	–	0	1	
	0	0	1	–	1
	0	0	–	1	1
None	–	0	0	0	
	0	0	–	0	1
	0	0	0	–	1

(b)

Number of 1s	x_1	x_2	x_3	x_4
None	0	0	–	–

(c)

For the second step of the process, we compare the original terms with the irreducible terms. We form a table with the original terms as column headers and the irreducible terms (the unnumbered terms in the reduction tables just constructed) as row labels. A check in the comparison table (Table 8.17) indicates that the original term in that column eventually led to the irreducible term in that row, which can be determined by following the pointers.

TABLE 8.17

	1011	0110	0101	0011	1000	0010	0001	0000
–011	✓			✓				
0–10		✓				✓		
0–01			✓				✓	
–000					✓			✓
00––				✓		✓	✓	✓

If a column in the comparison table has a check in only one row, the irreducible term for that row is the only one covering the original term, so it is an essential term and must appear in the final sum-of-products form. Thus, we see from Table 8.17 that the terms –011, 0–10, 0–01, and –000 are essential and must be in the final expression. We also note that all columns with a check in row 5 also have checks in another row and so are covered by an essential reduced term already in the expression. Thus, 00–– is redundant. As in Example 23, the minimal sum-of-products form is

$$x_2'x_3x_4 + x_1'x_3x_4' + x_1'x_3'x_4 + x_2'x_3'x_4'$$

Figure 2: Figures 8.16 and 8.17, illustrating the Quine-McCluskey minimization procedure.

In this procedure, we do exactly the same thing as we do in the method of Karnaugh maps, but we do it without the map (table). We search for those elements of the truth table which differ by a single entry, and then reduce them (essentially focusing on pairs, rather than quads, etc.).

We may have to do the reduction in several steps, as illustrated in Table 8.16, p. 675. Part (c) of Figure 8.17 represents a column of four 1s in the Karnaugh map.

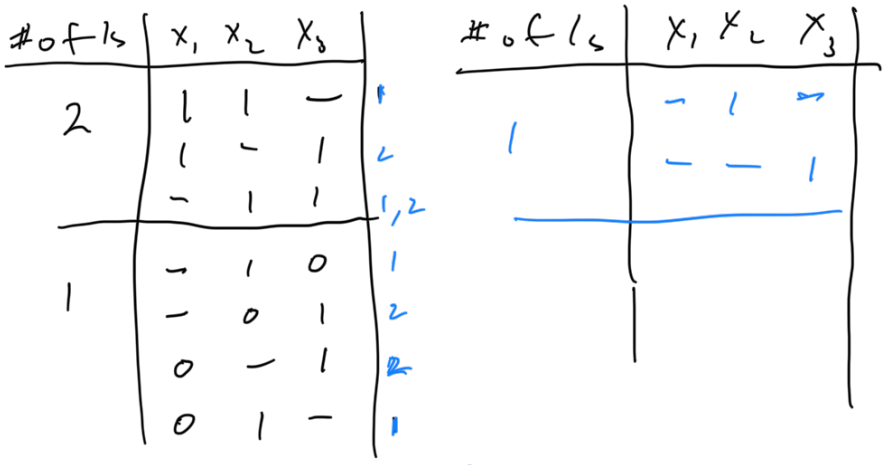
In the end, we have to determine which of the resultant products is necessary to recreate the initial truth table. We do this with a second type of table, as illustrated in Table 8.17, p. 675. This is essentially a pattern-matching table (we'll talk about these pattern-matches in our discussions of regular expressions in section 9.3): each of the column label expressions (the original product terms) is compared to the "deleted elements" of the result tables (e.g. the product 0010 matches both 0-10 and 00–).

Example: Exercise 21, p. 681

	x_1x_2	x_1x_2'	$x_1'x_2'$	$x_1'x_2$
x_3	1	1	1	1
x_3'	1	1	1	1

This is the basic starting table:

# of 1s	x_1	x_2	x_3	
3	1	1	1	1,2,3
2	1	1	0	1,4
	1	0	1	2,5
	0	1	1	3,6,7
1	0	0	1	5,6
	0	1	0	4,7



$E = x_2 + x_3$

$E' = x_1x_2'x_3' + x_1'x_2'x_3'$

$= (x_1 + x_1')x_2'x_3'$

$= x_2'x_3'$

$\therefore E = (x_2' \cdot x_3')'$

$= x_2 + x_3$ ✓

Furthermore, it is sometimes easier to use the Quine-McCluskey procedure on the complement of the truth function, if the complement has fewer entries. The downside is that we won't end up with a sum of products, but rather a product of sums - if that bothers you!

Example: Exercise 24, p. 682 (or Exercise 21 above, for a simpler example)

	x_1x_2	x_1x_2'	$x_1'x_2'$	$x_1'x_2$
x_3x_4	1	1	1	1
x_3x_4'	0	0	0	0
$x_3'x_4'$	0	1	1	1
$x_3'x_4$	1	1	1	1

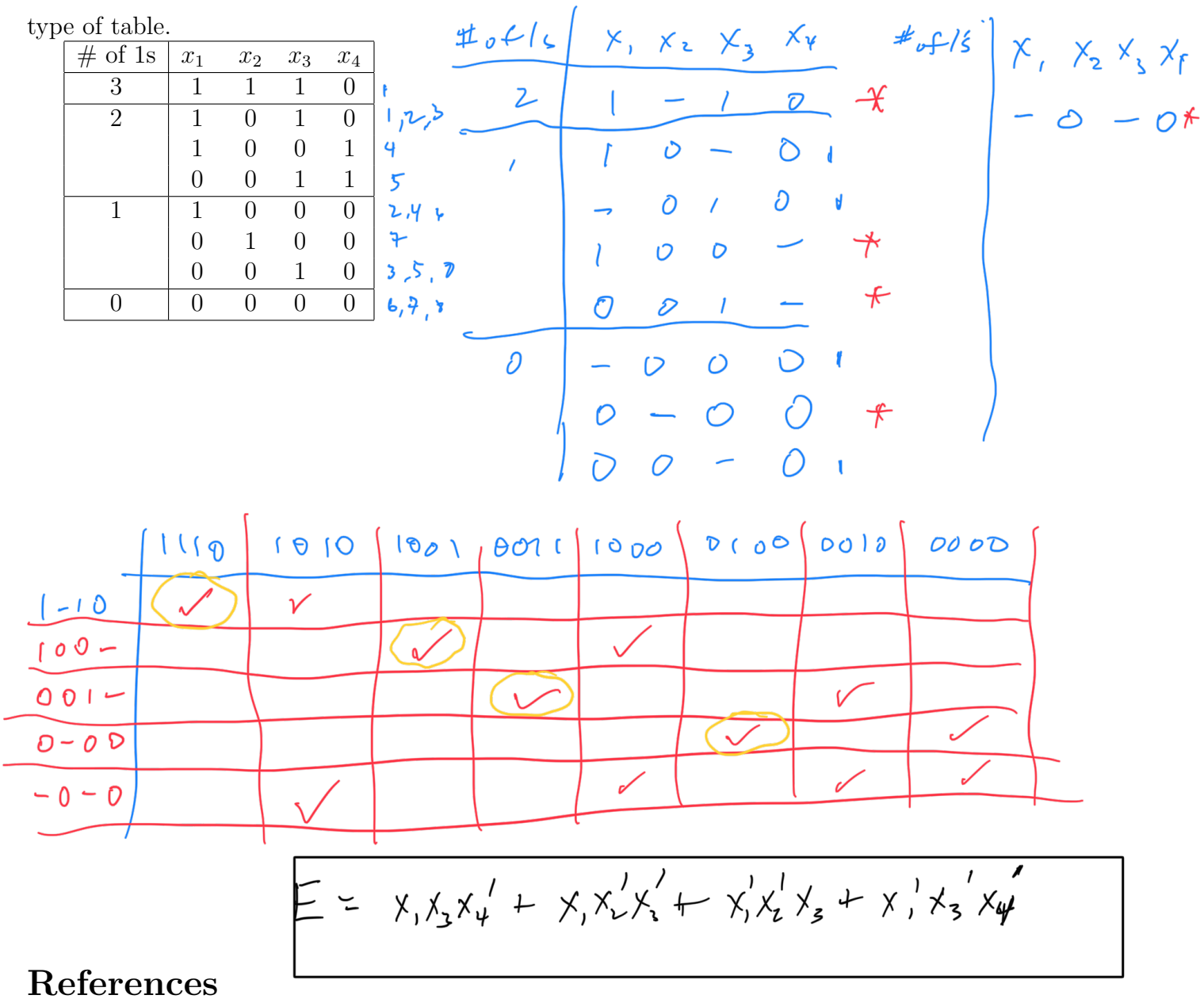
$E' = x_3x_4' + x_1x_2x_4'$

$E = (x_3x_4' + x_1x_2x_4')'$

$= ((x_3 + x_1x_2) \cdot x_4')'$ $= (x_3 + x_1x_2)' + x_4$

Example: Exercise 23, p. 682 illustrates the use of the second type of table.

# of 1s	x_1	x_2	x_3	x_4
3	1	1	1	0
2	1	0	1	0
	1	0	0	1
	0	0	1	1
1	1	0	0	0
	0	1	0	0
	0	0	1	0
0	0	0	0	0



References

[1] Maurice Karnaugh. The map method for synthesis of combinational logic circuits. *Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics*, 72(5), November 1953.

[2] Max Maxfield. Karnaugh maps 101. *EETimes*, 2011.

[3] J J O'Connor and E F Robertson. Willard Van Orman Quine. *MacTutor History of Mathematics archive*, 2003.

[4] National Academy of Engineering. *Memorial Tributes: Volume 21*. The National Academies Press, Washington, DC, 2017.