Section 7.3: Minimization

April 14, 2008

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 7.2, 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 simple 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 manageable Boolean expression.

2 Simplification and the Karnaugh Map

2.1 Simplification Rules

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

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

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

Example 17, p. 500/574. Consider the canonical form given by

$$E = x_1x_2x_3 + x_1'x_2x_3 + x_1'x_2x_3'$$

We demonstrate how rule (1) works as follows:

$$E = x_2(x_1x_3 + x_1'x_3) + x_1'x_2x_3' = x_2(x_1 + x_1')x_3 + x_1'x_2x_3' = x_2x_3 + x_1'x_2x_3'$$

Now we're set up for the use of rule (2):

$$E = x_2x_3 + x_1'x_2x_3' = x_2(x_3 + x_1'x_3')$$

To demonstrate rule (2) above, we get to use that wacky distributive rule:

$$E = x_2(x_3 + x_1'x_3') = x_2((x_3 + x_1')(x_3 + x_3')) = x_2(x_3 + x_1') = x_2x_3 + x_2x_1'$$

2.2 Karnaugh Map Examples

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

	x_1	x_1'
x_2		1
x_2'	1	

Example: Example 17: $x_1x_2x_3 + x_1'x_2x_3 + x_1'x_2x_3'$

	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_1'
x_2	1	1
x_2'	1	1

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

Check out this trick (idempotence):

	x_1	x_1'
x_2		1
x_2'	1	1

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

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 17: $x_1x_2x_3 + x_1'x_2x_3 + x_1'x_2x_3'$

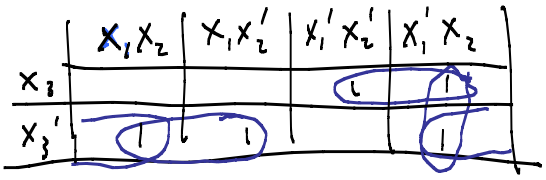
	x_1x_2	x_1x_2'	$x_1'x_2'$	$x_1'x_2$
x_3	1			1
x_3'				1

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. 512/586



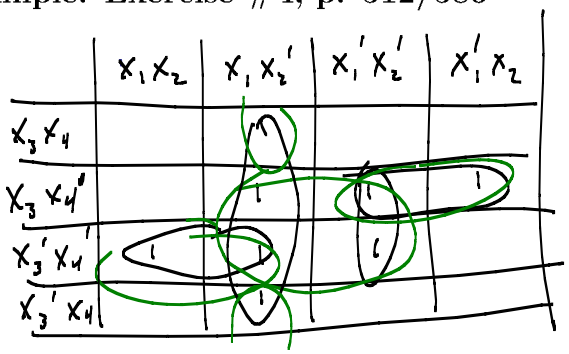
$$x_1'x_3 + x_1'x_2 + x_1x_2' + x_2x_3'$$

$$x_1'x_3 + x_1x_3' + x_2x_3'$$

$$x_1'x_3 + x_1'x_2 + x_1x_3'$$

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

Example: Exercise #4, p. 512/586



$$x_1x_2' + x_1'x_3x_4' + x_1'x_2'x_4' + x_1x_3'x_4'$$

$$x_1x_2' + x_2'x_4' + x_1x_3'x_4' + x_1'x_3x_4'$$

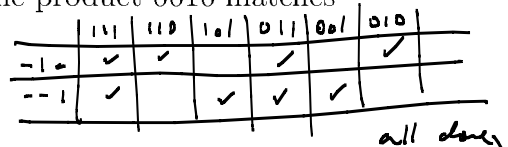
$$x_1x_2'x_4 + x_1'x_3x_4' + x_1x_3'x_4' + x_2'x_4'$$

3 Simplification and the Quine-McCluskey procedure

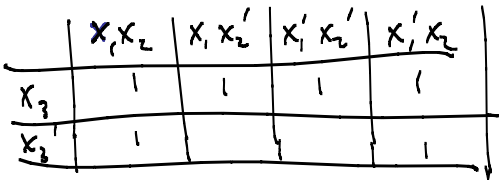
In this procedure, we do exactly the same thing as in Karnaugh, but we do it without the matrix (or table). We search for those elements of the truth table which differ by a single entry, and then reduce them.

We may have to do the reduction in several steps, as illustrated in Table 7.16, p. 509/583. Part (c) of that table 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 7.17, p. 510/584. This is essentially a pattern-matching table (we'll talk about these pattern-matches in our discussions of regular expressions in section 8.2): 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–).

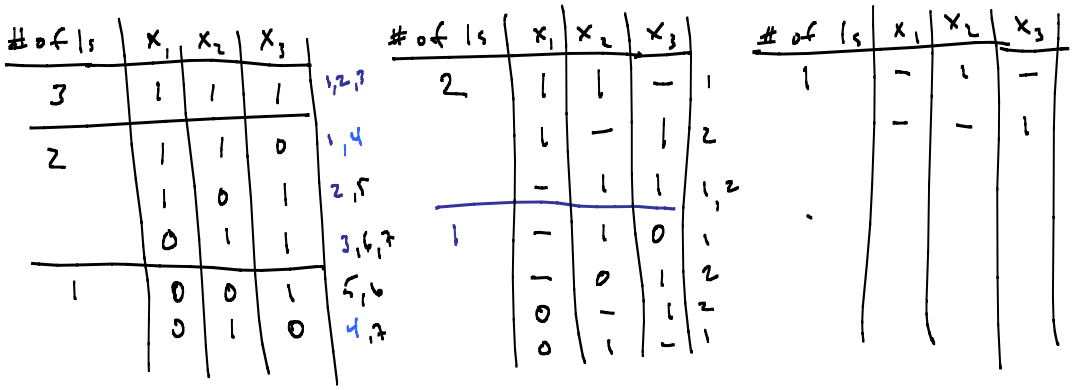


Example: Exercise 16/18, p. 514/588



$f(x_1, x_2, x_3, x_4) = x_2 + x_3$

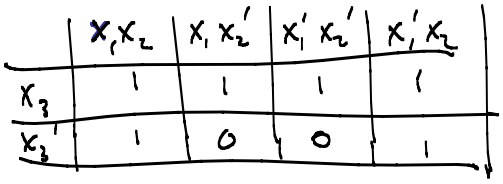
all done



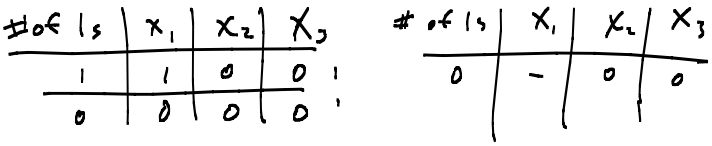
Furthermore, it is sometimes easier to use the Quine-McCluskey procedure on the complement of the truth function, if the complement has fewer entries. The downside is that we won't end up with a sum of products, but rather a product of sums - if that bothers you!

Example: Exercise 19/21, p. 515/589 (or Exercise 16/18 above, for a simpler example)

Work on the complement f'



$x_1x_2'x_3' + x_1'x_2'x_3'$



$$f'(x_1, x_2, x_3) = x_2' \cdot x_3'$$

$$f(x_1, x_2, x_3) = (x_2' \cdot x_3')' = x_2 + x_3$$