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	

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

$xy + xy' + x'y$

$= x + y$

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 (from the half adder of section 7.2): $x_1x_2' + x_1'x_2$

	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 (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

$x_1'x_2 + x_1x_2x_3$

$x_1'x_2 + x_2x_3$

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).

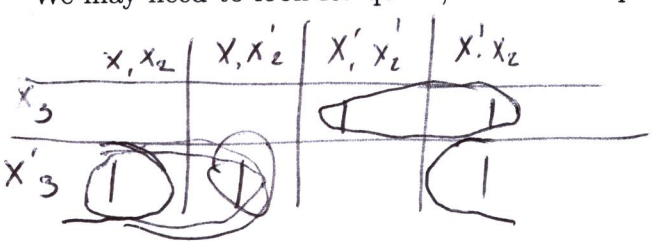
There may be multiple simplifications of a Boolean expression:

Example: Exercise #1, p. 586

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

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

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



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

Example: Exercise #4, p. 586

	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		

$x_1x_2' + x_2'x_4 + x_1x_3'x_4' + x_1x_3'x_4$

	x_2x_4	x_2x_4'	$x_2'x_4'$	$x_2'x_4$
x_1x_3			1	1
x_1x_3'		1	1	1
$x_1'x_3'$			1	
$x_1'x_3$		1	1	

3 Simplification and the Quine-McCluskey procedure

Edward McCluskey died recently (2016), at the age of 86. He was a bridge to Bertrand Russell and Alfred North Whitehead, because of his work with Willard Van Quine (who died in 2000, at the age of 92).

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. 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. 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 18, p. 588

	x_1x_2	x_1x_2'	$x_1'x_2'$	$x_1'x_2$
x_3	1	1	1	1
x_3'	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	5,7
	0	1	0	4,6

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

# of 1s	x_1	x_2	x_3
1	–	1	–
	–	–	1

$x_2 + x_3$

Furthermore, it is sometimes easier to use the Quine-McCluskey procedure on the complement of the truth function, if the complement has

	111	110	101	011	001	010
–1–	✓	✓		✓		
––1	✓		✓	✓	✓	✓

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 21, p. 589 (or Exercise 18 above, for a simpler example)

# of 1	x_1	x_2	x_3	x_4	
3	1	1	1	0	'
2	1	1	0	0	'
	1	0	1	0	✓ ✓
	0	1	1	0	✓ ✓
1	0	0	1	0	✓

# of 1	x_1	x_2	x_3	x_4	
2	1	1	–	0	
	–	1	1	0	'
	1	–	1	0	'
	–	0	1	0	'
1	0	–	1	0	'

Example: Exercise 20, p. 589 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

# of 1	x_1	x_2	x_3	x_4
1	–	–	1	0

	1110	1100	1010	0110	0010
– – 1 0	✓		✓	✓	✓
1 1 – 0	✓	✓			

$$f(x_1,x_2,x_3,x_4) = x_3x_4' + x_1x_2x_4'$$
$$= (x_3 + x_1x_2)\cdot x_4'.$$
$$f(x_1,x_2,x_3,x_4) = \left[(x_3 + x_1x_2)\cdot x_4'\right]'$$

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

$$= x_3' \cdot (x_1 \cdot x_2)' + x_4$$

$$= x_3' x_1' + x_3' x_2' + x_4$$