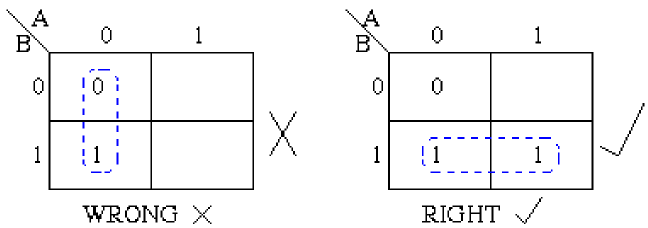
Karnaugh Maps - Rules of Simplification

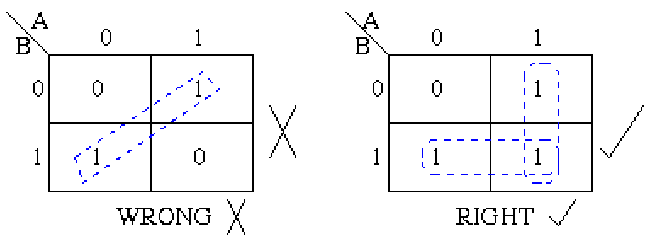
(from https://www.electro-tech-online.com/attachments/karnaugh_maps-doc.2477 but slightly modified by DK)

The Karnaugh map uses the following rules for the simplification of expressions by *grouping* together adjacent cells containing *ones*

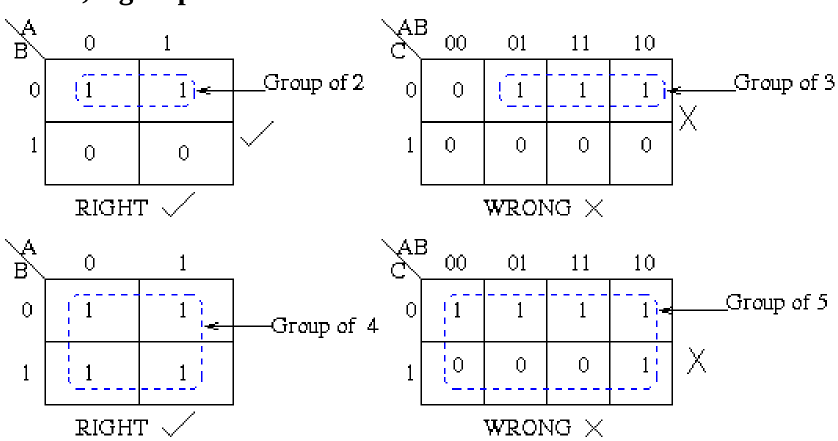
- **Groups may not include any cell containing a zero**



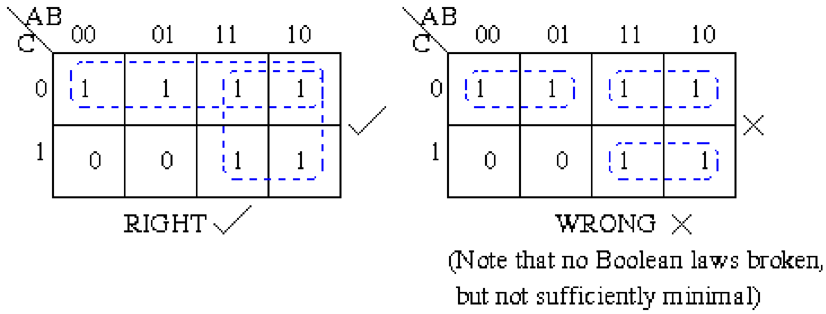
- **Groups may be horizontal or vertical, but not diagonal.**



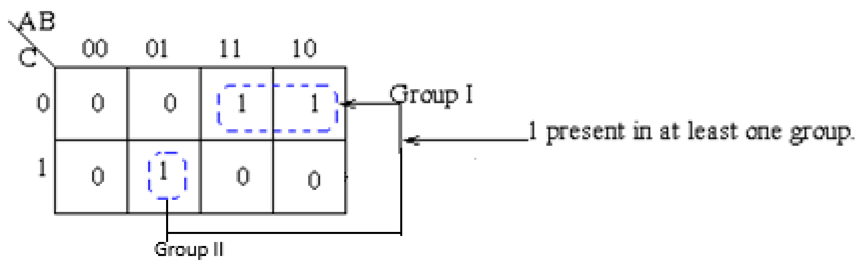
- **Groups must contain 1, 2, 4, 8, or in general 2^n cells.**
  That is if n = 1, a group will contain two 1's since $2^1 = 2$.
  If n = 2, a group will contain four 1's since $2^2 = 4$.



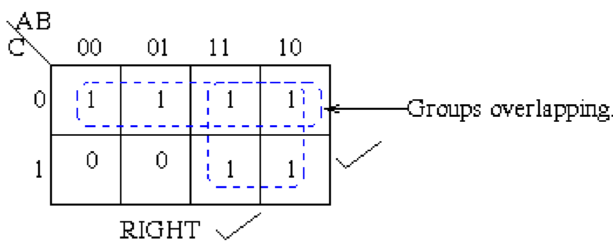
- **Each group should be as large as possible.**

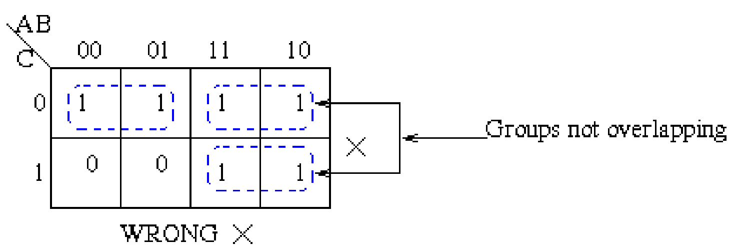


- **Each cell containing a *one* must be in at least one group.**

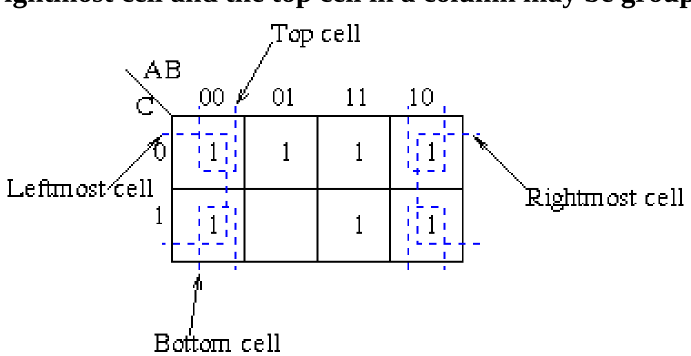


- **Groups may overlap.**

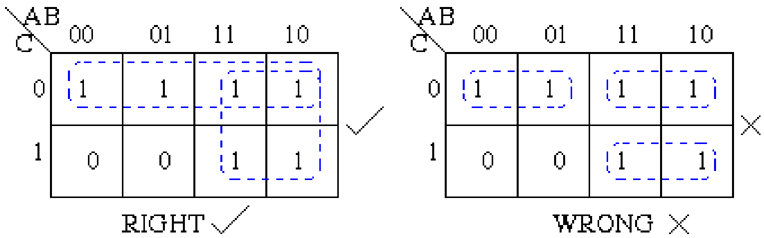




- **Groups may wrap around the table. The leftmost cell in a row may be grouped with the rightmost cell and the top cell in a column may be grouped with the bottom cell.**



- **There should be as few groups as possible, as long as this does not contradict any of the previous rules.**



Summary:

1. No zeros allowed.
2. No diagonals.
3. Only power of 2 number of cells in each group.
4. Groups should be as large as possible.
5. Every one must be in at least one group.
6. Overlapping allowed.
7. Wrap around allowed.
8. Fewest number of groups possible.

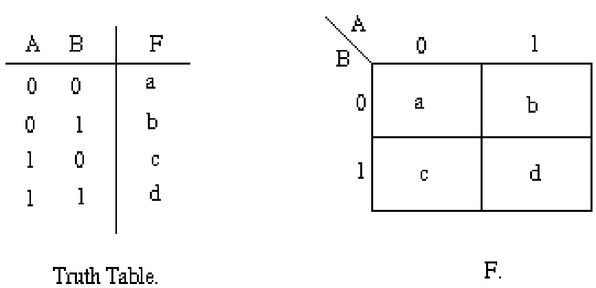
Composed by David Belton

The Karnaugh map provides a simple and straight-forward method of minimising boolean expressions. With the Karnaugh map Boolean expressions having up to four and even six variables can be simplified.

So what is a Karnaugh map?

A Karnaugh map provides a pictorial method of grouping together expressions with common factors and therefore eliminating unwanted variables. The Karnaugh map can also be described as a special arrangement of a truth table.

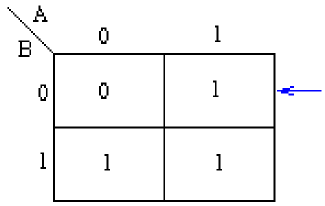
The diagram below illustrates the correspondence between the Karnaugh map and the truth table for the general case of a two variable problem.



The values inside the squares are copied from the output column of the truth table, therefore there is one square in the map for every row in the truth table. Around the edge of the Karnaugh map are the values of the two input variable. A is along the top and B is down the left hand side. The diagram below explains this:

A	B	F
0	0	0
0	1	1
1	0	1
1	1	1

Truth Table.

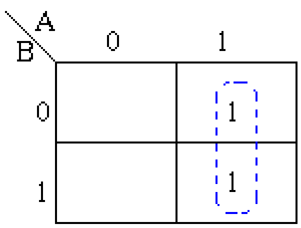


F.

The values around the edge of the map can be thought of as coordinates. So as an example, the square on the top right hand corner of the map in the above diagram has coordinates A=1 and B=0. This square corresponds to the row in the truth table where A=1 and B=0 and F=1. Note that the value in the F column represents a particular function to which the Karnaugh map corresponds.

Example 1:

Consider the following map. The function plotted is: $Z = f(A,B) = A\overline{B} + AB$



- Note that values of the input variables form the rows and columns. That is the logic values of the variables A and B (with one denoting true form and zero denoting false form) form the head of the rows and columns respectively.
- Bear in mind that the above map is a one dimensional type which can be used to simplify an expression in two variables.
- There is a two-dimensional map that can be used for up to four variables, and a three-dimensional map for up to six variables.

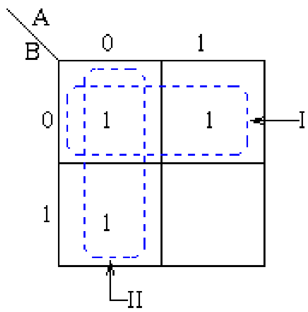
Using algebraic simplification,

$Z = A\overline{B} + AB$
$Z = A(\overline{B} + B)$
$Z = A$

Variable B becomes redundant due to Boolean Theorem T9a.

Referring to the map above, the two adjacent 1's are grouped together. Through inspection it can be seen that variable B has its true and false form within the group. This eliminates variable B leaving only variable A which only has its true form. The minimised answer therefore is Z = A.

Example 2: Consider the expression $Z = f(A,B) = \overline{A}\overline{B} + A\overline{B} + \overline{A}B$ plotted on the Karnaugh map:



Pairs of 1's are *grouped* as shown above, and the simplified answer is obtained by using the following steps:

Note that two groups can be formed for the example given above, bearing in mind that the largest rectangular clusters that can be made consist of two 1s. Notice that a 1 can belong to more than one group.
The first group labelled I, consists of two 1s which correspond to A = 0, B = 0 and A = 1, B = 0. Put in another way, all squares in this example that correspond to the area of the map where B = 0 contains 1s, independent of the value of A. So when B = 0 the output is 1. The expression of the output will contain the term $\overline{B}$

For group labelled II corresponds to the area of the map where A = 0. The group can therefore be defined as $\overline{A}$. This implies that when A = 0 the output is 1. The output is therefore 1 whenever B = 0 and A = 0
Hence the simplified answer is Z = $\overline{A} + \overline{B}$

Composed by David Belton

University of Southern Maine

Minimization of Boolean expressions using Karnaugh maps.

Given the following truth table for the majority function.

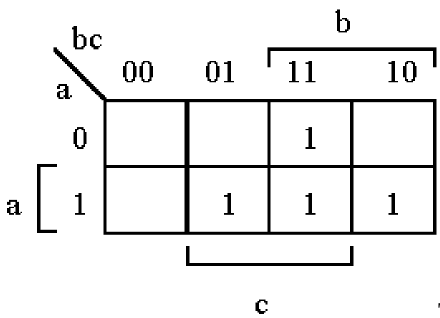
abc	m
000	0
001	0
010	0
011	1
100	0
101	1
110	1
111	1

The Boolean algebraic expression is m = a'bc + ab'c + abc' + abc.

We have seen that the minimization is done as follows. m = a'bc + abc + ab'c + abc + abc' + abc

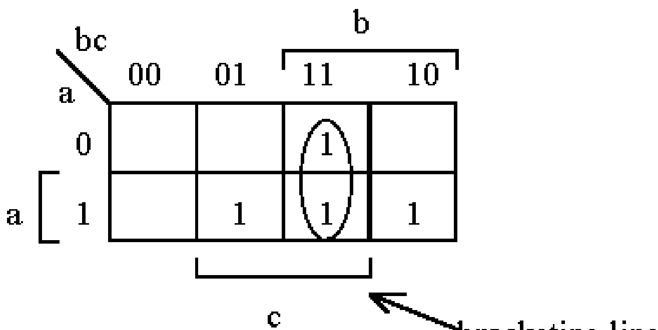
= (a' + a)bc + a(b' + b)c + ab(c' + c) = bc + ac + ab The **abc** term was replicated and combined with the other terms.

To use a Karnaugh map we draw the following map which has a position (square) corresponding to each of the 8 possible combinations of the 3 Boolean variables. The upper left position corresponds to the 000 row of the truth table, the lower right position corresponds to 110. Each square has two coordinates, the vertical coordinate corresponds to the value of variable **a** and the horizontal corresponds to the values of **b** and **c**.



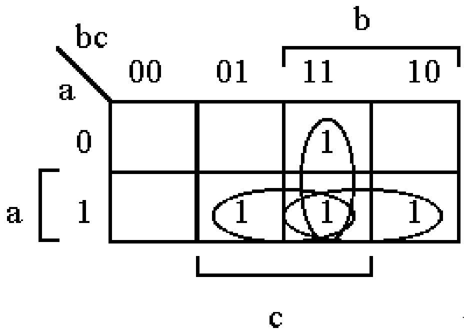
The 1s are in the same places as they were in the original truth table. The 1 in the first row is at position 011 (**a** = 0, **b** = 1, **c** = 1). The vertical coordinate, variable **a**, has the value 0. The horizontal coordinates, the variables **b** and **c**, have the values 1 and 1.

The minimization is done by drawing circles around sets of adjacent 1s. Adjacency is horizontal, vertical, or both. The circles must always contain 2^n 1s where n is an integer.



We have circled two 1s. The fact that the circle spans the two possible values of **a**

(0 and 1) means that the **a** term is eliminated from the Boolean expression corresponding to this circle. The bracketing lines shown above correspond to the positions on the map for which the given variable has the value 1. The bracket delimits the set of squares for which the variable has the value 1. We see that the two circled 1s are at the intersection of sets **b** and **c**, this means that the Boolean expression for this set is **bc**.



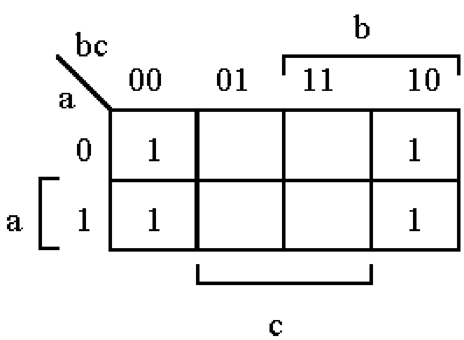
Now we have drawn circles around all the 1s. The left bottom circle is the term **ac**. Note that the circle spans the two possible values of **b**, thus eliminating the **b** term. Another way to think of it is that the set of squares in the circle contains the same squares as the set **a** intersected with the set **c**. The other circle (lower right) corresponds to the term **ab**. Thus the expression reduces to   bc + ac + ab   as we saw before.

What is happening? What does adjacency and grouping the 1s together have to do with minimization? Notice that the 1 at position 111 was used by all 3 circles. This 1 corresponds to the abc term that was replicated in the original algebraic minimization. Adjacency of 2 1s means that the terms corresponding to those 1s differ in one variable only. In one case that variable is negated and in the other it is not.

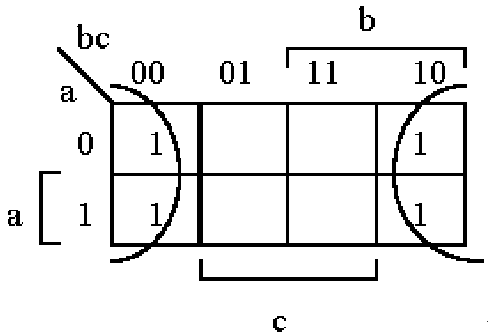
For example, in the first map above, the one with only 1 circle. The upper 1 is the term **a'bc** and the lower is **abc**. Obviously they combine to form **bc** (**a'bc + abc = (a' + a)bc = bc**). That is exactly what we got using the map.

The map is easier than algebraic minimization because we just have to recognize patterns of 1s in the map instead of using the algebraic manipulations. Adjacency also applies to the edges of the map.

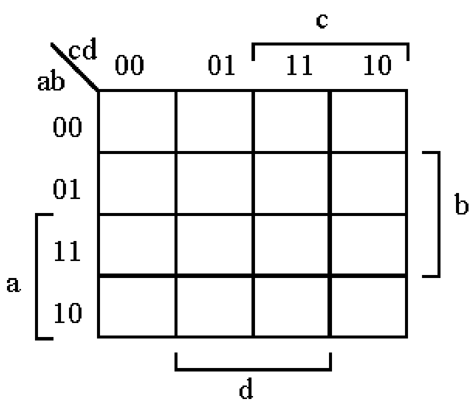
Let's try another 3 variable map.



At first it may seem that we have two sets, one on the left of the map and the other on the right. Actually there is only 1 set because the left and right are adjacent as are the top and bottom. The expression for all 4 1s is **c'**. Notice that the 4 1s span both values of **a** (0 and 1) and both values of **b** (0 and 1). Thus, only the **c** value is left. The variable **c** is 0 for all the 1s, thus we have **c'**. The other way to look at it is that the 1's overlap the horizontal **b** line and the short vertical **a** line, but they all lay outside the horizontal **c** line, so they correspond to **c'**. (The horizontal **c** line delimits the **c** set. The **c'** set consists of all squares outside the **c** set. Since the circle includes all the squares in **c'**, they are defined by **c'**. Again, notice that both values of **a** and **b** are spanned, thus eliminating those terms.)

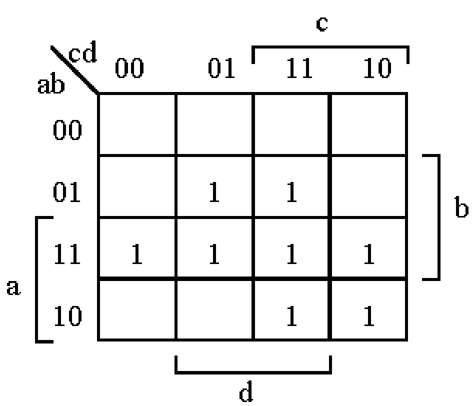


Now for 4 Boolean variables. The Karnaugh map is drawn as shown below.



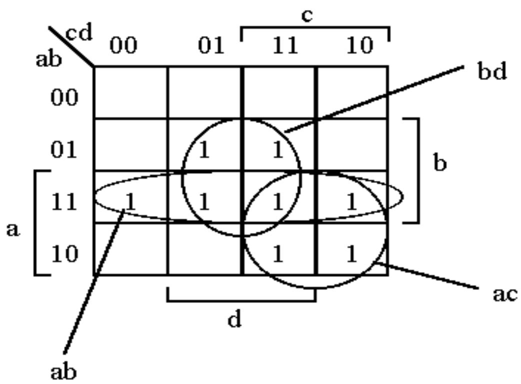
The following corresponds to the Boolean expression

q = a'bc'd + a'bcd + abc'd' + abc'd + abcd + abcd' + ab'cd + ab'cd'



RULE: Minimization is achieved by drawing the smallest possible number of circles, each containing the largest possible number of 1s.

Grouping the 1s together results in the following.

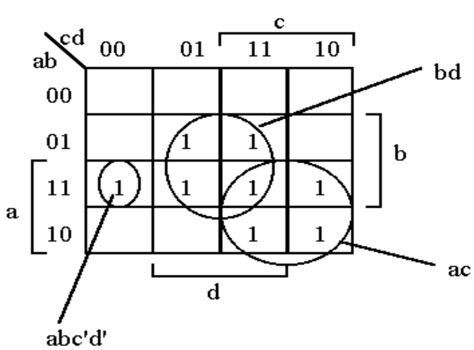


The expression for the groupings above is

q = bd + ac + ab

This expression requires 3 2-input **and** gates and 1 3-input **or** gate.

We could have accounted for all the 1s in the map as shown below, but that results in a more complex expression requiring a more complex gate.



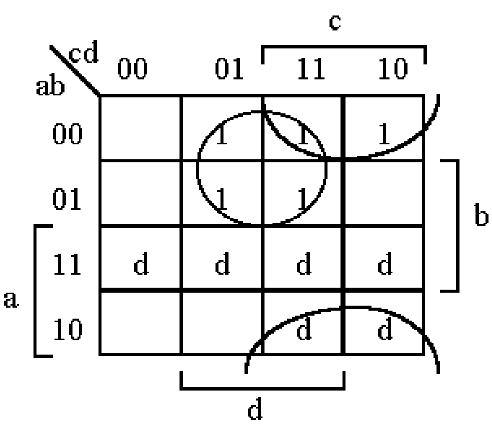
The expression for the above is **bd + ac + abc'd'**. This requires 2 2-input **and** gates, a 4-input **and** gate, and a 3 input **or** gate. Thus, one of the **and** gates is more complex (has two additional inputs) than required above. Two inverters are also needed.

Don't Cares

Sometimes we do not care whether a 1 or 0 occurs for a certain set of inputs. It may be that those inputs will never occur so it makes no difference what the output is. For example, we might have a bcd (binary coded decimal) code which consists of 4 bits to encode the digits 0 (0000) through 9 (1001). The remaining codes (1010 through 1111) are not used. If we had a truth table for the prime numbers 0 through 9, it would be

abcd	p
0000	0
0001	1
0010	1
0011	1
0100	0
0101	1
0110	0
0111	1
1000	0
1001	0
1010	d
1011	d
1100	d
1101	d
1110	d
1111	d

The ds in the above stand for "don't care", we don't care whether a 1 or 0 is the value for that combination of inputs because (in this case) the inputs will never occur.



The circle made entirely of 1s corresponds to the expression **a'd** and the combined 1 and d circle (actually a combination of arcs) is **b'c**. Thus, if the disallowed input 1011 did occur, the output would be 1 but if the disallowed input 1100 occurs, its output would be 0. The minimized expression is

p = a'd + b'c

Notice that if we had ignored the ds and only made a circle around the 2 1s, the resulting expression would have been more complex, **a'b'c** instead of **b'c**.