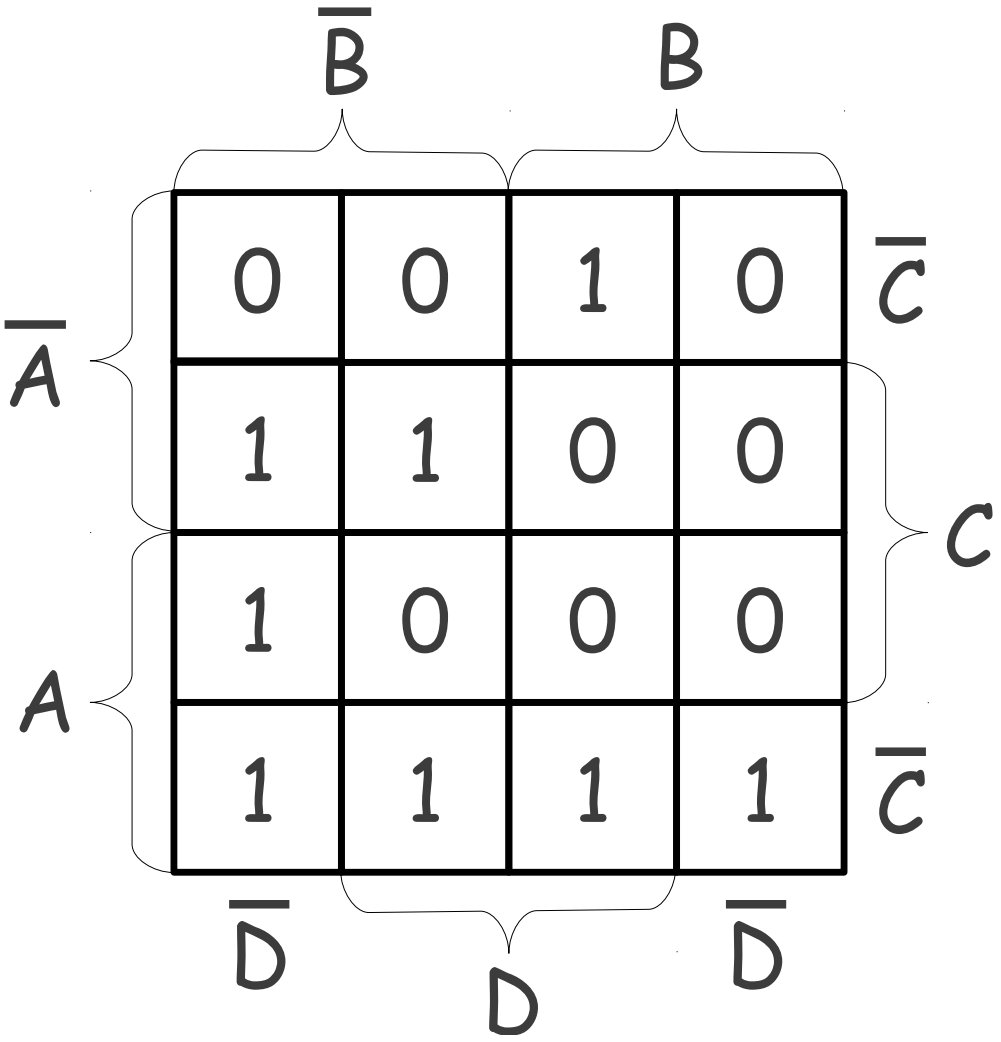
Designing Computer Systems

Simplification



Designing Computer Systems

Simplification

Using DeMorgan's theorem, any binary expression can be transformed into a multitude of equivalent forms that represent the same behavior. So why should we pick one over another? One form provides a canonical representation. Another provides a clear representation of the desired function. As engineers, we want more than functionality. We crave performance and efficiency!

So what improves an expression? … fewer logical functions. Every logical function requires computational resources that add delay and/or cost energy, switches, design time, and dollars. If we can capture the desired behavior with fewer logical operations, we are building the Ferrari of computation; or maybe the Prius?

Simplify Your Life: There are many techniques to simplify Boolean expressions. Expression reductions (e.g., $X + \overline{X} \cdot Y \rightarrow X + Y$) is good. But it's not obvious what to reduce first, and it's hard to know when you're finished. An intuitive method can be seen in a truth table … sometimes. Here's an example. Consider the expression:

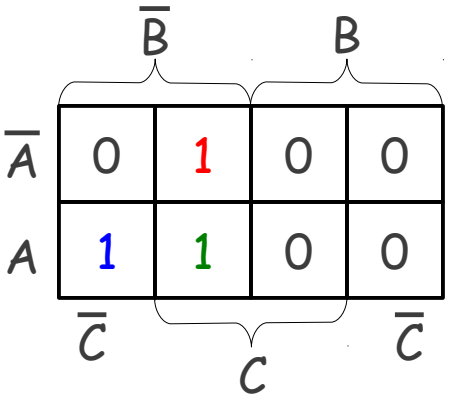
$$Out = A \cdot \overline{B} \cdot \overline{C} + \overline{A} \cdot \overline{B} \cdot C + A \cdot \overline{B} \cdot C$$

In truth table form, one might notice that the red and green terms suggest that when B is false (zero) and C is true (one), Out is true (one) no matter what state A is in. The blue and green terms express a similar simplification. If A is true and B is false, Out is true independent of C. A simplified expression $Out = \overline{B} \cdot C + A \cdot \overline{B}$ expresses the same behavior with three dyadic (two input) logical operations versus eight for the original expression.

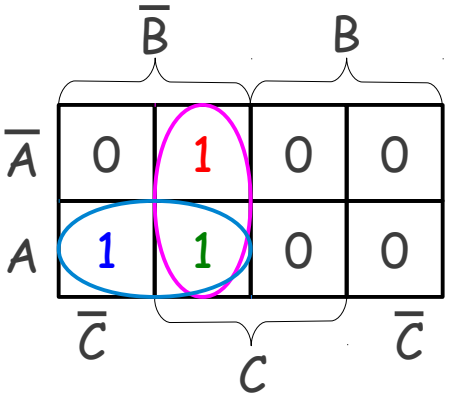
A	B	C	Out
0	0	0	0
1	0	0	1
0	1	0	0
1	1	0	0
0	0	1	1
1	0	1	1
0	1	1	0
1	1	1	0

This simplification is intuitive. In all cases when a subexpression (e.g., $\overline{B}\cdot C$) is true, the output is true, then including extra qualifying terms is unnecessary. Unfortunately, truth tables lack uniform adjacency of these simplifying groupings.

For a small number of variables, a Karnaugh Map (K-map) displays the same behavior information in a different way. A K-map are composed of a two-dimensional map displaying the output for every combination of input values. But these combinations are arranged so that horizontal or vertical movement results in exactly one variable changing. Here's the K-map for the function being considered.



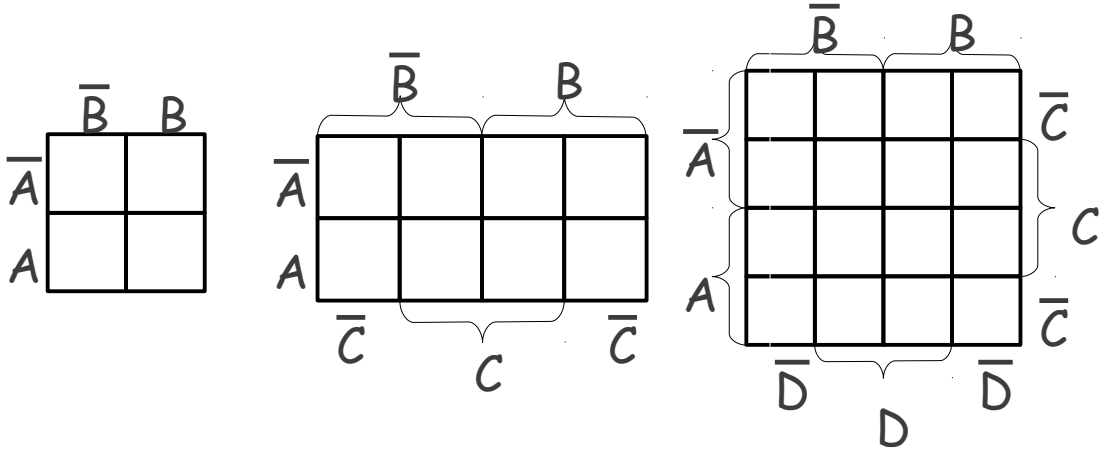
In this map, the top row includes all input combinations where A is false. The second row includes all combinations where A is true. The left two columns include input combinations where B is false. The right two columns cover when B is true. The outermost columns include input combinations where C is false. The middle two columns include cases where C is true.



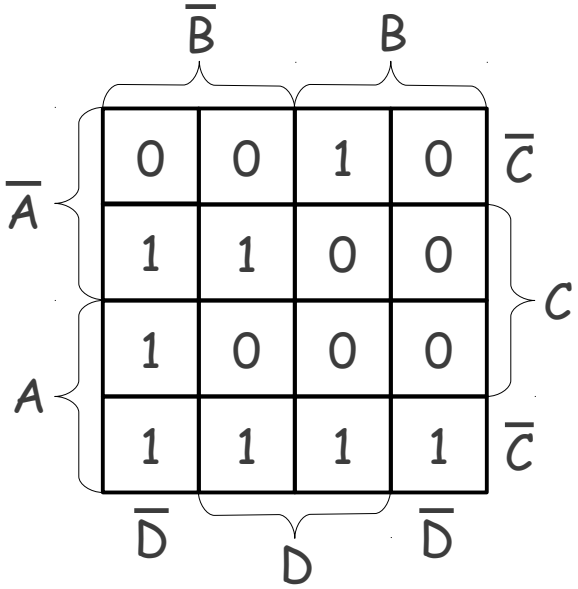
In this arrangement, adjacent ones (true outputs) suggests an opportunity for simplification. The red and green ones can be grouped into a single term covering all combinations where B is zero and C is one ($\overline{B}\cdot C$). The adjacent blue and green ones are grouped to cover where A is one and B is zero ($A\cdot\overline{B}$). Since a simplified expression must cover all cases when the output is one, these terms can be

combined to express the function's behavior: Out = $\overline{B}\cdot C + A\cdot\overline{B}$, a simplified sum of products expression.

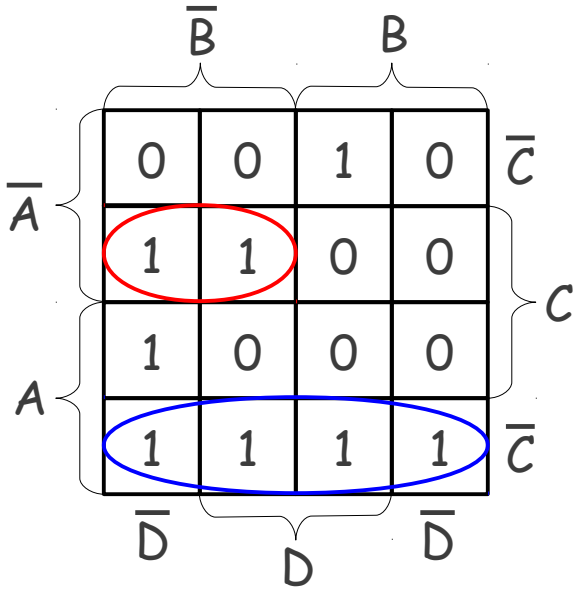
Two-dimensional K-maps accommodate two-, three-, and four-variable expressions. Larger K-maps (five- and six-variable) are possible in three dimensions. But they are error prone and better simplification techniques exist.



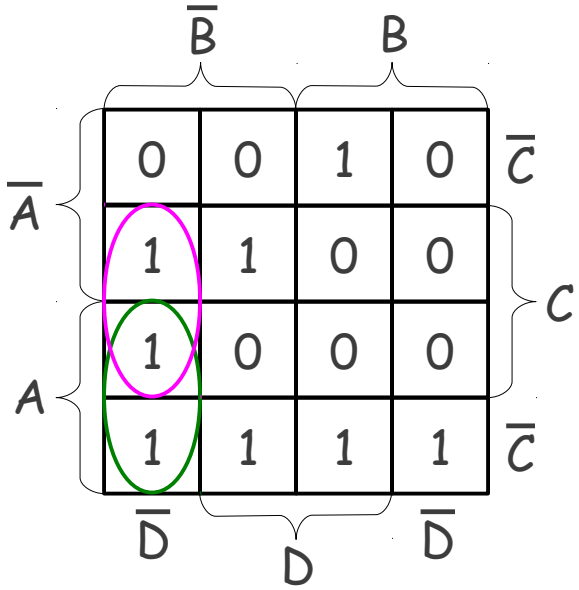
Just like a truth table, the K-map describes a function's behavior by giving the output for every combination of the inputs. But adjacency in a K-map also indicates opportunities for expression simplification. Here's a four-variable K-map.



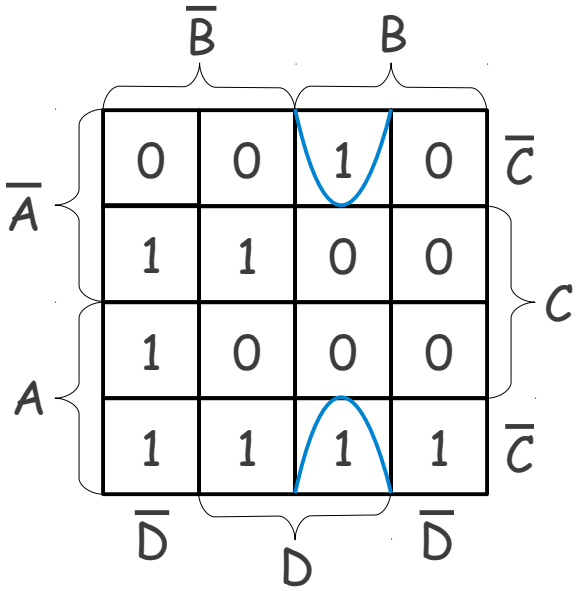
The behavior represented by this K-map could be represented as a truth table. Adjacent ones are opportunities for simplification. The size of groupings are given as (width x height). So a (2x1) grouping is two squares wide and one square high.



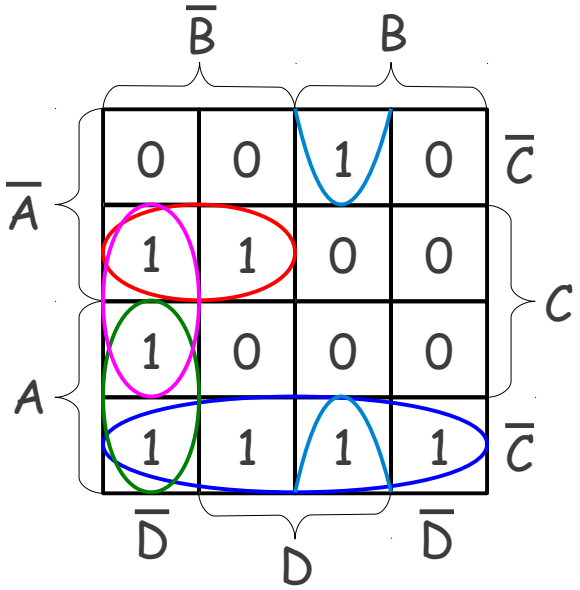
The second row grouping (2x1) represents all cases where A is false, B is false, and C is true ($\overline{A}\cdot\overline{B}\cdot C$). The bottom row (4x1) is all cases where A is true and C is false ($A\cdot\overline{C}$). Larger groupings lead to smaller terms. But the grouping has to be describable as *all cases where variables have a certain value*.



The first column contains three adjacent ones. In this candidate grouping, B and D are zero. But it is not *all cases* where B and D are zero. A (1x3) grouping is not a describable grouping. Instead these adjacent ones are cover by two overlapping (1x2) groupings: $C\cdot\overline{D}$ and $A\cdot\overline{D}$. Overlapping groups are okay, so long as one grouping is not subsumed by another grouping. Groupings of ones always have power of two dimensions (1, 2, 4).



Adjacency extends at the K-map edges. The one in the third column of the top row can be grouped with the corresponding one in the bottom row. This grouping represents all cases where B is true, C is false, and D is true ($B \cdot \overline{C} \cdot D$). Here are all legal groupings in this K-map.



Note that while groupings overlap, no grouping falls completely within another. The grouped terms: $\overline{A} \cdot \overline{B} \cdot C$, $A \cdot \overline{C}$, $C \cdot \overline{D}$, $A \cdot \overline{D}$, and $B \cdot \overline{C} \cdot D$, represent all candidate terms in the simplified expression. But are all terms necessary?

The objective is to correctly define the behavior by expressing all cases when the output is one. This means selecting groupings that cover all true outputs in the K-map. Sometimes this requires all groupings. Sometimes not. In this K-map, three groupings, $\overline{A} \cdot \overline{B} \cdot C$, $A \cdot \overline{C}$, and $B \cdot \overline{C} \cdot D$, include a true not covered by any other grouping.

This makes them *essential* for the simplified expression. If they are not included, the behavior is not accurately defined. But in the example, these essential groupings don't cover all the ones in the K-map. Additional groupings are needed.

Two groupings $C\cdot\overline{D}$ and $A\cdot\overline{D}$, are not essential (*non-essential*) but cover the missing one in the behavior ($A\cdot B\cdot C\cdot\overline{D}$). Since they have the same number of variables, and the same cost to implement, either will provide an equivalently simplified expression.

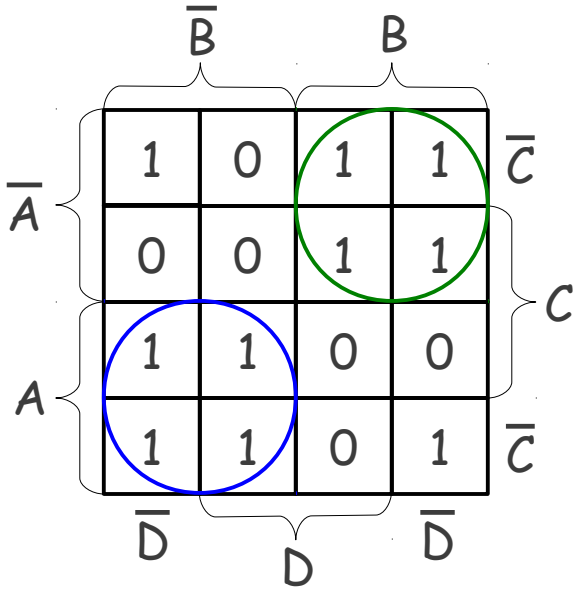
Out = $\overline{A}\cdot\overline{B}\cdot C + A\cdot\overline{C} + C\cdot\overline{D} + B\cdot\overline{C}\cdot D$ Out = $\overline{A}\cdot\overline{B}\cdot C + A\cdot\overline{C} + A\cdot\overline{D} + B\cdot\overline{C}\cdot D$

These two simplified expressions are significantly less expensive to implement than the canonical sum of products expression.

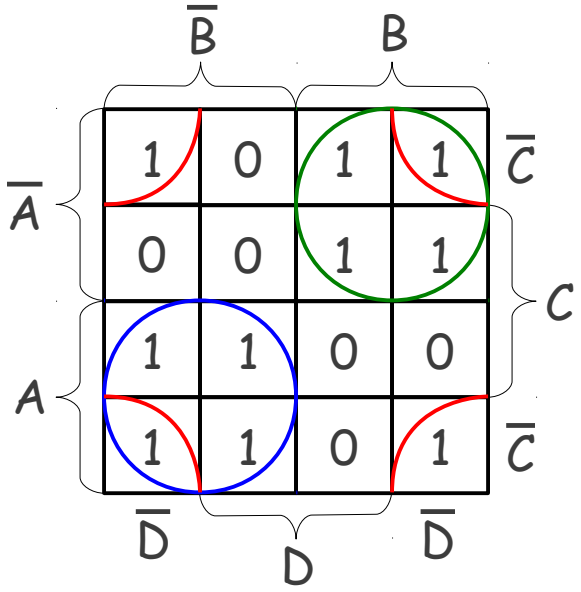
Out = $\overline{A}\cdot B\cdot\overline{C}\cdot D + \overline{A}\cdot\overline{B}\cdot C\cdot\overline{D} + \overline{A}\cdot\overline{B}\cdot C\cdot D + A\cdot\overline{B}\cdot C\cdot\overline{D} + A\cdot\overline{B}\cdot\overline{C}\cdot\overline{D} + A\cdot\overline{B}\cdot\overline{C}\cdot D + A\cdot B\cdot\overline{C}\cdot D + A\cdot B\cdot\overline{C}\cdot\overline{D}$

Parlance of the Trade: Due to its arithmetical origin, these groupings are named *Prime Implicants*. PIs for short. Essential prime implicants are always included in the simplified expression because they exclusively contain one of the grouped elements. Non-essential PIs may or may not be needed, depending on whether essential PIs cover the selected outputs. Formally, this simplification process is defined as *minimally spanning the selected outputs*.

But aren't the selected outputs always true? Not always. More on this later. Here's another example.



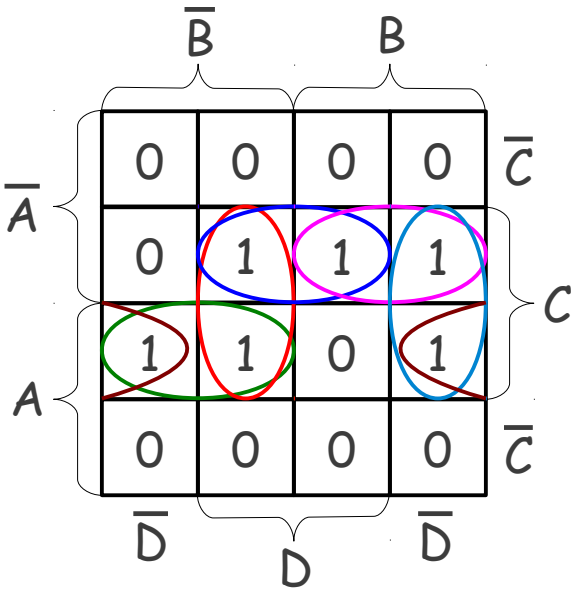
Two (2x2) PIs stand out: all cases where A is false and B is true ($\overline{A}\cdot B$), and all cases where A is true and B is false ($A\cdot\overline{B}$). But how to group the remaining true outputs?



The final (2x2) PI groups the four corners, covering all cases where C and D are zero ($\overline{C}\cdot\overline{D}$). The edges of a K-map connected; there's just no good way to draw it on a two-dimensional plane. If the vertical edges are joined, the map becomes a cylinder. If the ends of the cylinder are joined, it becomes a donut (torus). In two-dimensions, one must look for connections on the edges of K-maps.

Since all PIs are essential and necessary to span true outputs in the behavior, the simplified sum or products expression is Out = $\overline{A}\cdot B + A\cdot\overline{B} + \overline{C}\cdot\overline{D}$

Here's another example.



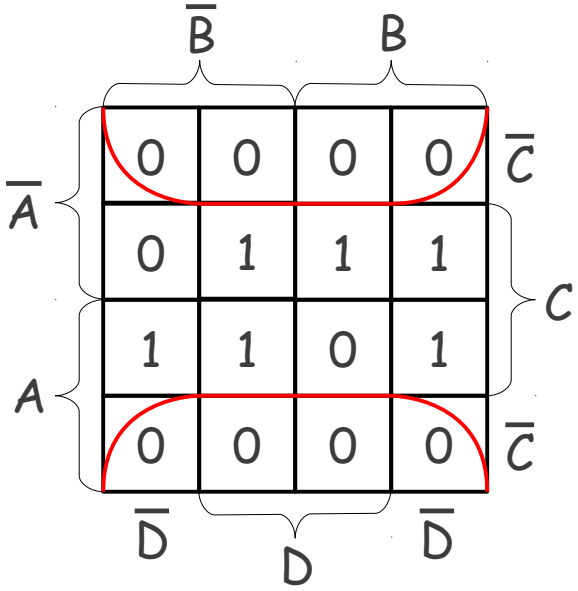
This example contains six overlapping PIs: $A\cdot\overline{B}\cdot C$, $\overline{B}\cdot C\cdot D$, $\overline{A}\cdot C\cdot D$, $\overline{A}\cdot B\cdot C$, $B\cdot C\cdot\overline{D}$, and $A\cdot C\cdot\overline{D}$. What makes them interesting is that *none are essential*. Sometimes there is an urge to ignore non-essential PIs when simplifying a K-map. This example

demonstrates the need to record all legal PIs before considering minimal spanning. There are two, equally simplified sum of products expressions for this behavior.

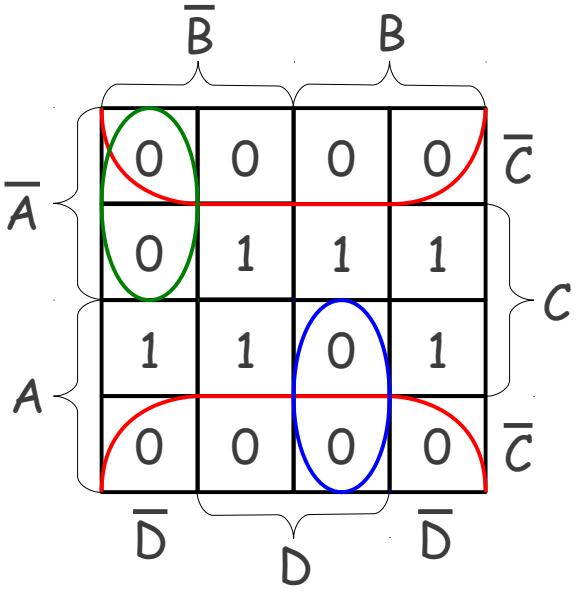
Out = $A\cdot\overline{B}\cdot C + \overline{A}\cdot C\cdot D + B\cdot C\cdot\overline{D}$ Out = $\overline{B}\cdot C\cdot D + \overline{A}\cdot B\cdot C + A\cdot C\cdot\overline{D}$

Incomplete listing of PIs might not expose both of these expressions due to false appeasement of essentialness. It might even yield a less simplified result.

SoP versus PoS: Boolean Algebra explains the duality where a function's behavior can be defined by stating where the output is true (sum of products) or by stating where the function is not false (product of sums). This applies in K-maps. In the first case (SoP), product terms define a group of true outputs (a PI). A spanning set of these groupings is ORed together to form the simplified expression. In the second case (PoS), groupings represent where the output is not true, but false. Since the simplified expression must still represent where the behavior is true, a grouping (PI) must express *states not in the grouping*. Here's the same example, targeting a simplified product of sums expression.



The largest grouping of zeros (false) covers the cases where C is false. As in the SoP process, a (4x2) grouping is drawn. But labeling this group $\overline{C}$ is not helpful, since the goal is expressing when the behavior is true. This grouping include many of the false outputs and none of the true outputs. So the PI should represent when it is *not in this grouping*, namely C. Being outside the red PI C is not enough; additional Pis are required to guarantee a true output.



Just as ones where grouped in SoP, here zeros are grouped. A (1x2) grouping of false outputs represents when A is true and B is true and D is true. But the PI for this grouping represents cases *not in this grouping*. That occurs when A is false OR B is false OR D is false ($\overline{A}+\overline{B}+\overline{D}$). It is ORed because the output is not in this grouping if any of the variables are false. Again, this doesn't mean the output is true, its just not in this grouping of zeros.

Another (1x2) grouping of false outputs occurs when A is false and B is false and D is false. This PI is expressed as A is true or B is true or D is true (A+B+D). These are all cases not in the green grouping.

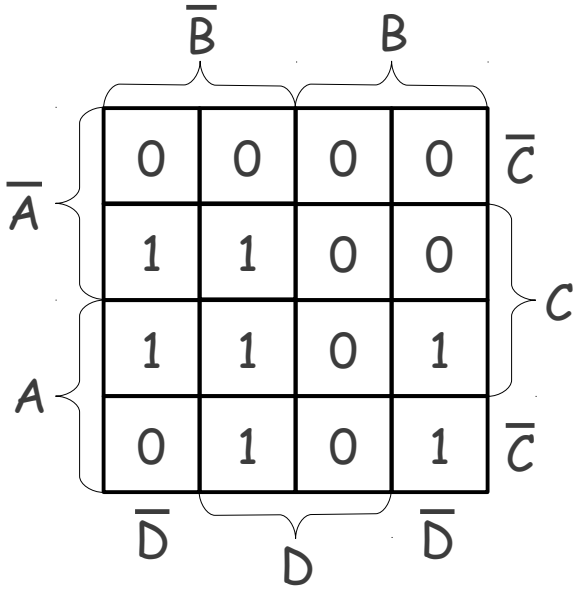
So how does the simplified expression show when the behavior is true? By showing when it it is not false! Each of the PIs represents cases that are not in one of the groups of zeros. If the PIs span all false outputs, and all terms are true, the output must be true. In this example, all PIs are essential (i.e., they contain a false output not included in any other PI). So the simplified product of sums expression is:

$$\text{Out} = C \cdot (\overline{A}+\overline{B}+\overline{D}) \cdot (A+B+D)$$

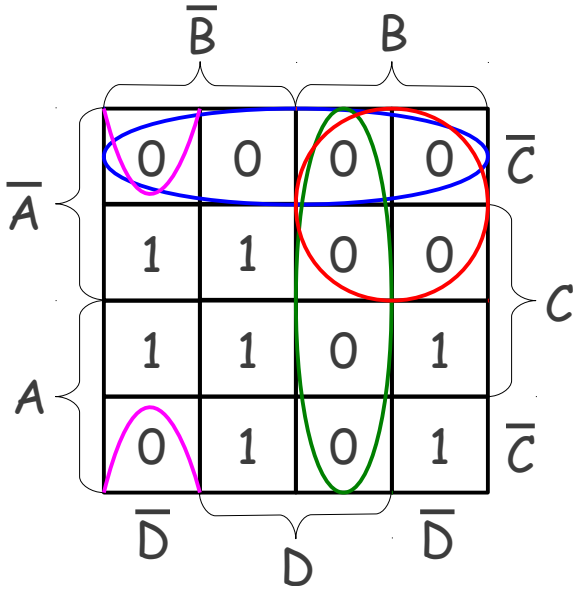
Note that this simplified PoS expression has no obvious relationship to the equivalent simplified SoP expressions:

$$\text{Out} = A\cdot\overline{B}\cdot C + \overline{A}\cdot C\cdot D + B\cdot C\cdot\overline{D} = \overline{B}\cdot C\cdot D + \overline{A}\cdot B\cdot C + A\cdot C\cdot\overline{D}$$

This PoS example has unusual symmetries in its PIs. Here's a different example.



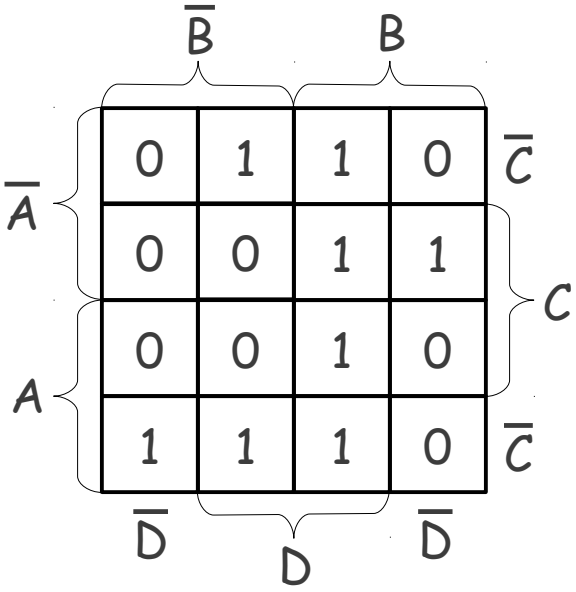
The groupings of false outputs mirror the SoP technique for true outputs. Only here the PI is labeled for cases outside its group. The (4x1) in the top row contains cases where A and C are false. The PI is labeled (A+C). The (1x4) in the third column includes cases where B and D are true. The PI is labeled ($\overline{B}$+$\overline{D}$). The upper right quadrant (2x2) is selected when A is false and B is true. The PI is (A+$\overline{B}$). The (1x2) in the first column is cases where B, C and D are all false. The PI is (B+C+D).



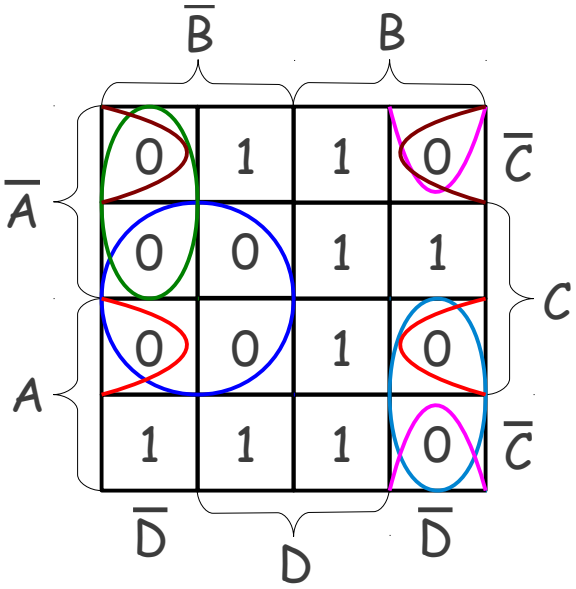
All PIs are essential. So the simplified PoS expression is:

$$\text{Out} = (A+C) \cdot \overline{B}+\overline{D} \cdot (B+C+D)$$

Here's another example.



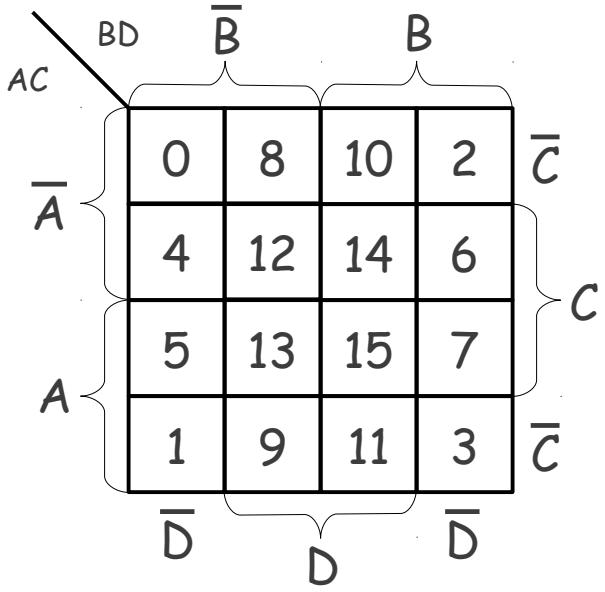
The PIs include the following: (B+$\overline{C}$), (A+B+D), (A+C+D), ($\overline{B}$+C+D), ($\overline{A}$+$\overline{B}$+D), ($\overline{A}$+$\overline{C}$+D).



Only (B+$\overline{C}$) is essential. No other PI exclusively contains a false output. But this simplified expression is not ambiguous. The minimal span of zeros yields:

Out = (B+$\overline{C}$) · (A+C+D) · ($\overline{A}$+$\overline{B}$+D)

Six of One; A Half Dozen of Another: Some folks find it advantageous to place the input variables in the upper left corner of a K-map and assign truth table row numbers to each square. While truth table/K-map correspondences are never row/column sequential, the numbering can have some semi-sequential ordering. In our examples, it looks like this:

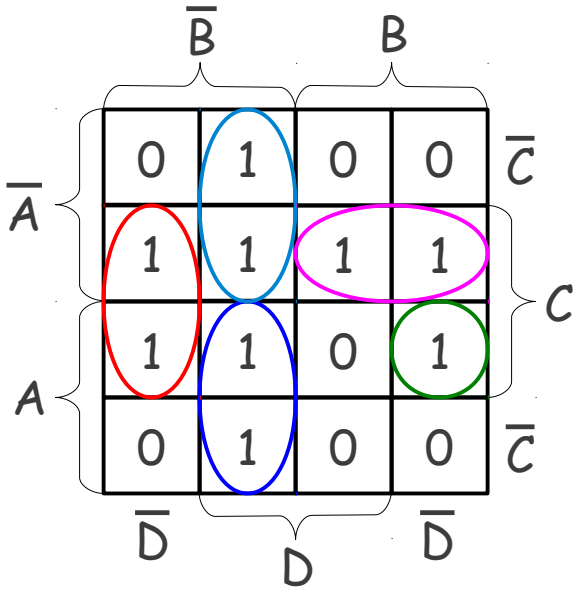


So what difference does it make? In terms of simplification, none. Reordering the input variables scrambles the K-map cells. But any proper ordering, where vertical and horizontal movements change exactly one variable, yields the same PIs and the same simplified expression.

Simplifying Boolean Expression: This ordering helps simplify Boolean expressions. Suppose a behavior, defined as a Boolean expression, is to be simplified.

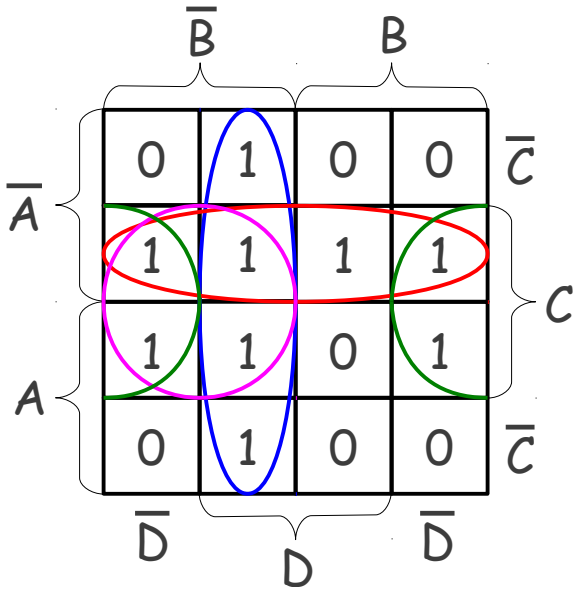
$$Out = A \cdot \overline{B} \cdot D + A \cdot B \cdot C \cdot \overline{D} + \overline{B} \cdot C \cdot \overline{D} + \overline{A} \cdot B \cdot C + \overline{A} \cdot \overline{B} \cdot D$$

In this SoP expression, each product term represents a grouping of true outputs. The ungrouped cases represent false outputs. Here's the mapping of each term.



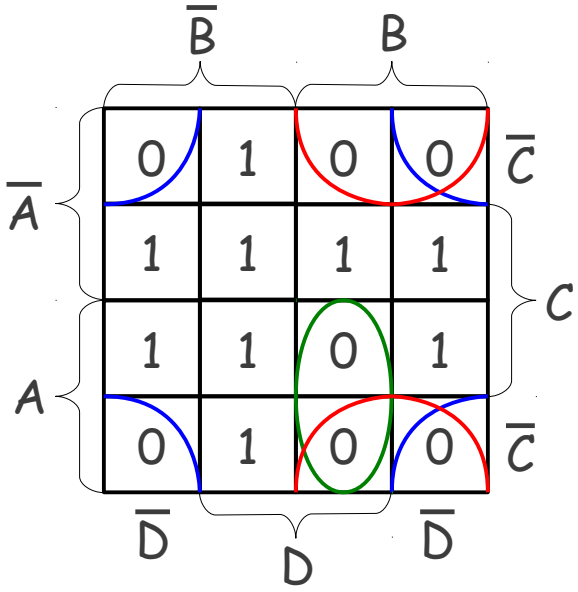
Once the expression's behavior is defined, it can be simplified as either a SoP or PoS expression. In SoP simplification, there are four PIs ($\overline{B}\cdot D$, $\overline{A}\cdot C$, $C\cdot\overline{D}$, $\overline{B}\cdot C$), three of which are essential and span the true outputs.

$$Out = \overline{B}\cdot D + \overline{A}\cdot C + C\cdot\overline{D}$$



In PoS, false outputs are grouped. Here three PIs are identified. All are essential.

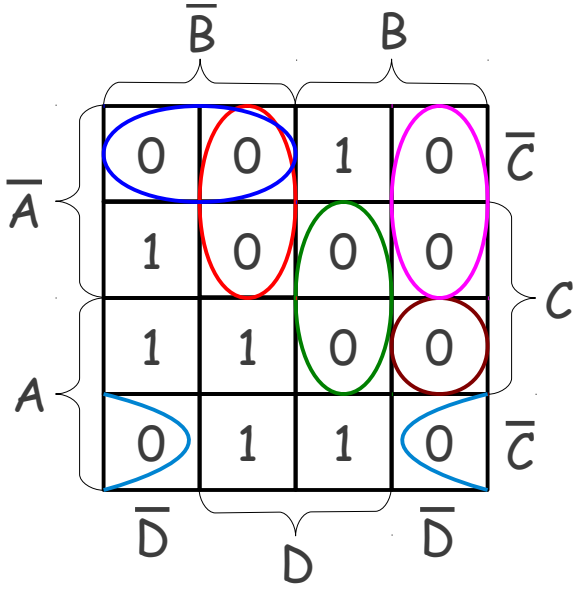
$$Out = (C+D)\cdot(\overline{B}+C)\cdot(\overline{A}+\overline{B}+\overline{D})$$



Both simplified expressions require fewer dyadic logical operations than the original expression (five for SoP, six for PoS versus 15 for original).

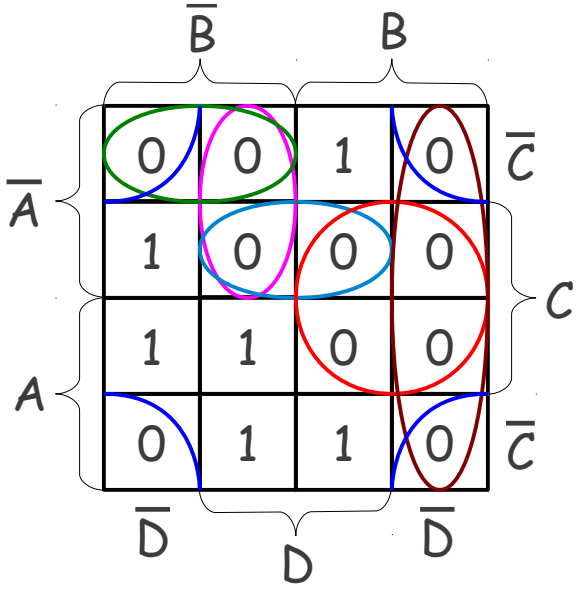
Here's a PoS expression to be simplified. Each term represents potentially true outputs *outside* a grouping of known false values. So zeros can be added for each term. Finally, the ungrouped cases are assigned a true value (one).

$$\text{Out} = (A+B+C) \cdot (A+B+\overline{D}) \cdot (\overline{B}+\overline{C}+\overline{D}) \cdot (A+\overline{B}+D) \cdot (\overline{A}+C+D) \cdot (\overline{A}+\overline{B}+\overline{C}+D)$$



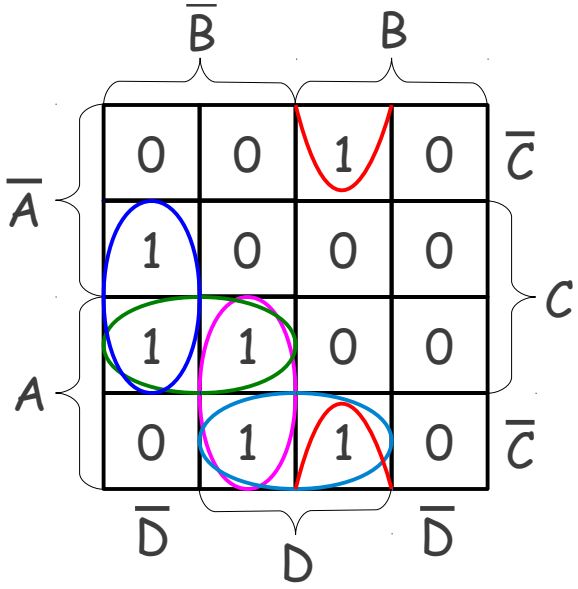
The PoS simplification produces six PIs: $(C+D)$, $(\overline{B}+\overline{C})$, $(A+B+C)$, $(A+B+\overline{D})$, $(A+\overline{C}+\overline{D})$, $(\overline{B}+D)$, two of which are essential: $(C+D)$, $(\overline{B}+\overline{C})$. One non-essential is require to span false outputs: $(A+B+\overline{D})$.

$$\text{Out} = (C+D) \cdot (\overline{B}+\overline{C}) \cdot (A+B+\overline{D})$$



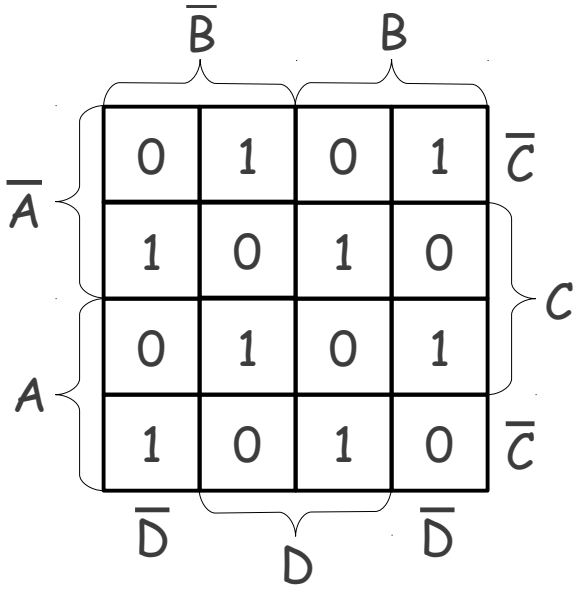
This behavior also can be simplified to a SoP expression. It produces five PIs: $\overline{B}\cdot C\cdot\overline{D}$, $A\cdot\overline{B}\cdot C$, $A\cdot\overline{B}\cdot D$, $A\cdot\overline{C}\cdot D$, $B\cdot\overline{C}\cdot D$, two are essential: $\overline{B}\cdot C\cdot\overline{D}$, $B\cdot\overline{C}\cdot D$. One non-essential PI is required to cover true outputs: $A\cdot\overline{B}\cdot D$.

$$\text{Out} = \overline{B}\cdot C\cdot\overline{D} + B\cdot\overline{C}\cdot D + A\cdot\overline{B}\cdot D$$



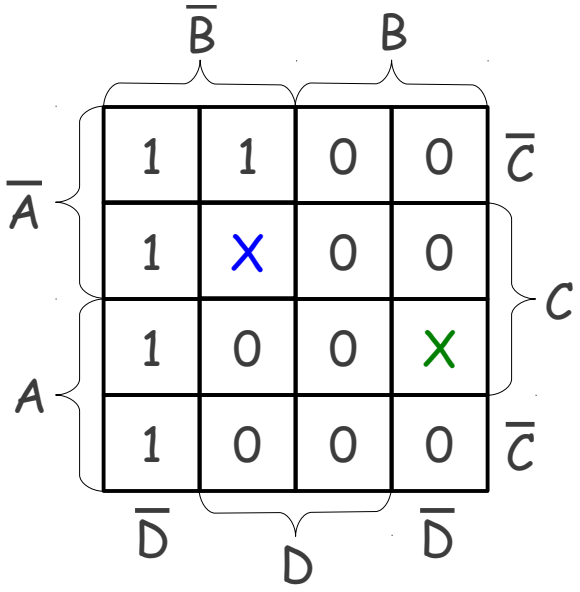
Again, simplification reduced implementation cost from 18 logical operations (original) to six (PoS) and eight (SoP). It's clear from these examples that there is no direct translation of a product term to a sum term, or vice versa.

Simplification Nightmare: Is there an *unsimplifiable behavior*? Yes. Here's odd parity (XOR).

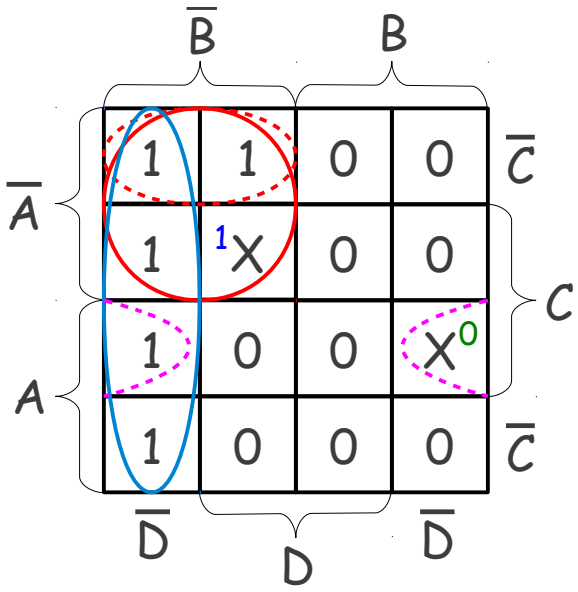


Odd and even parity (XOR and XNOR) toggle their outputs with every horizontal or vertical movement. Exactly one variable is changing. So the number of ones also toggles between odd and even. For this reason, there are no adjacent true or false outputs. SoP terms are always *minterms* and PoS terms are always *maxterms*. Parity is a useful function. But it is relatively costly.

What If You Don't Care?: Sometimes a function's behavior is unimportant for certain combinations of the inputs. Maybe it cannot occur. Or when a combination of the inputs occurs, the output is not used. This is recorded in the truth table as an "X" for the output. When an "X" occurs in a K-map, it can be defined as either a zero or one to improve the simplification process. Here's an example.



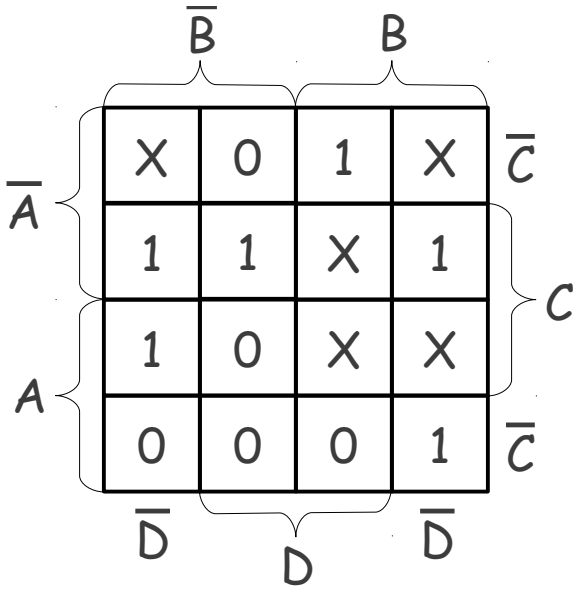
For two combinations of inputs: $\overline{A}\cdot\overline{B}\cdot C\cdot D$ and $A\cdot B\cdot C\cdot\overline{D}$, the output is unspecified and listed as don't care "X". This underspecification of the behavior permits the don't cares to be specified to reduce implementation cost.



To simplify this behavior to a SoP expression, the don't cares must be specified. For the first case, the choice of a true output doesn't add additional PIs. Rather it increases a PI from a (2x1) $\overline{A}\cdot\overline{B}\cdot\overline{C}$ to a (2x2) $\overline{A}\cdot\overline{B}$. This eliminates a logical operation from the simplified expression. In the second case, a false output is selected to eliminate the need for an additional PI: $A\cdot C\cdot\overline{D}$, to cover it. The simplified expression is found:

$$\text{Out} = \overline{A}\cdot\overline{B} + \overline{B}\cdot\overline{D}$$

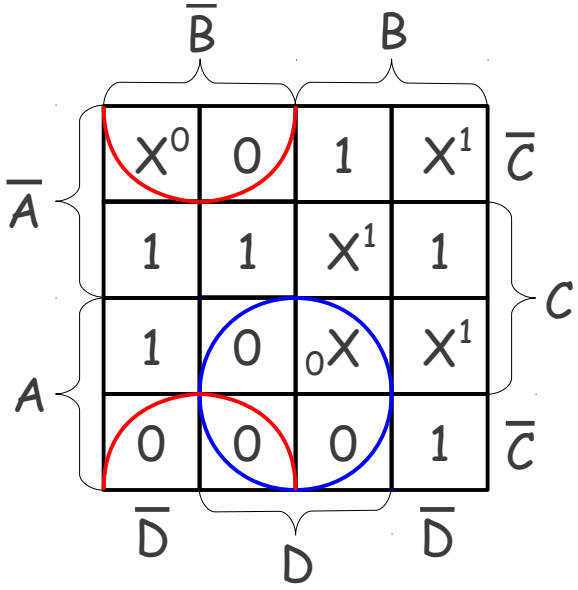
Sometimes specifying don't cares can take some thought. Here's another example.



Suppose a simplified PoS expression is required. Don't cares are specified to maximize the size of false output PIs while minimizing the number of PIs.

Here, the don't cares a specifies false to create two (2x2) PIs: (B+C), ($\overline{A}$+$\overline{D}$). The remaining don't cares are specified as true and don't contribute to groupings. The simplified expressing is:

Out = (B+C) · ($\overline{A}$+$\overline{D}$)



More than Four Variables?: How does handle more complex behaviors with over four variables. Other simplification techniques, like the Quine-McCluskey algorithm, are not limited in the number of variables. They are less intuitive to humans. But they are more amenable to computers and can be programmed into design tools like Espresso.

Summary: This chapter addresses methods for Boolean expression simplification using Karnaugh maps.

- Term reduction is possible when all combinations of a subexpression have a true (or false) output.
- A Karnaugh map specifies a behavior where vertical and horizontal movement changes exactly one variable.
- For a simplified sum of products expressions, true outputs are grouped. For product of sums expression, false outputs are grouped.
- A prime implicant is a group of adjacent true or false outputs that are of power of two (1, 2, 4) dimensions, and not enclosed in a larger grouping.
- A product PI lists where the output is true. A sum PI lists where the output is not in a specified grouping of false outputs.
- Boolean expressions and behaviors with don't cares can be simplified.

