WEST UNIVERSITY OF TIMIŞOARA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
BACHELOR STUDY PROGRAM: Computer Science in English

BACHELOR THESIS

SUPERVISOR:
Conf. Dr. Adrian Craciun

GRADUATE:
Roberto Tarta

TIMIŞOARA
2022

WEST UNIVERSITY OF TIMIŞOARA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
BACHELOR STUDY PROGRAM: Computer Science in English

Logic Optimization

SUPERVISOR:
Conf. Dr. Adrian Craciun

GRADUATE:
Roberto Tarta

TIMIŞOARA
2022

Abstract

Logic circuits are crucial components in modern electronic systems, but their increasing complexity often leads to higher energy consumption, material requirements, and costs. To address these challenges, various methods have been developed to simplify complex circuits, ultimately enhancing efficiency and profitability. This thesis aims to compare three prominent logic circuit optimization techniques: Boolean Algebra, Karnaugh Maps, and the Quine McClusky Algorithm. By applying these methods to a common Running Example, their respective results will be evaluated and compared. This research contributes to the field of logic circuit design by providing a comprehensive analysis of different optimization approaches, aiding engineers in selecting the most suitable method for reducing circuit complexity, minimizing costs, and improving overall performance.

Contents

List of Figures

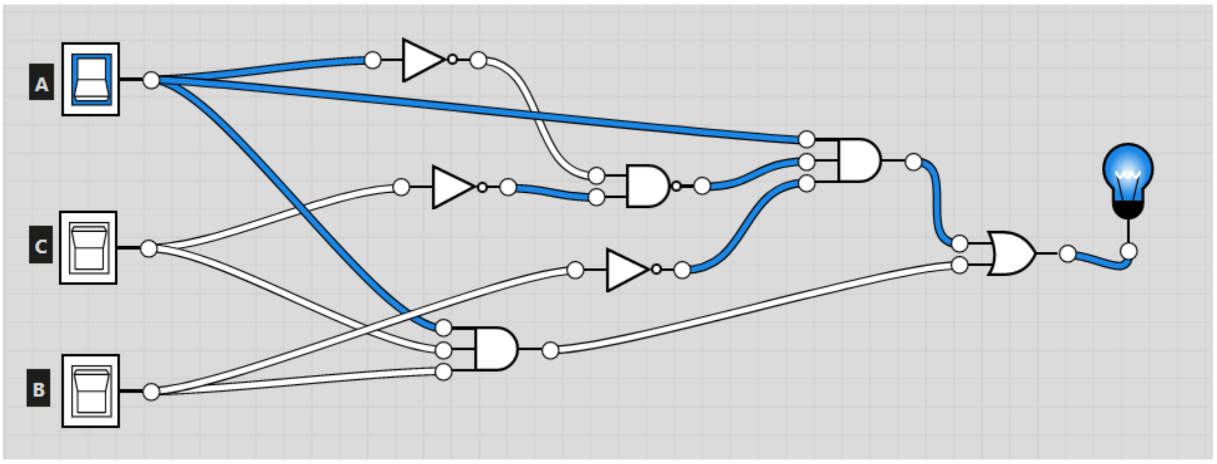
Listings

Chapter 1

Introduction

1.1 Problem Statement

Logic optimization at its core, consists in taking a more complex logic circuit and turning it into a simpler one without changing the output, in other words, without changing its functionality. Turning the more complex logic circuit into a more simple one helps in reducing material costs, electricity consumption and computation time.



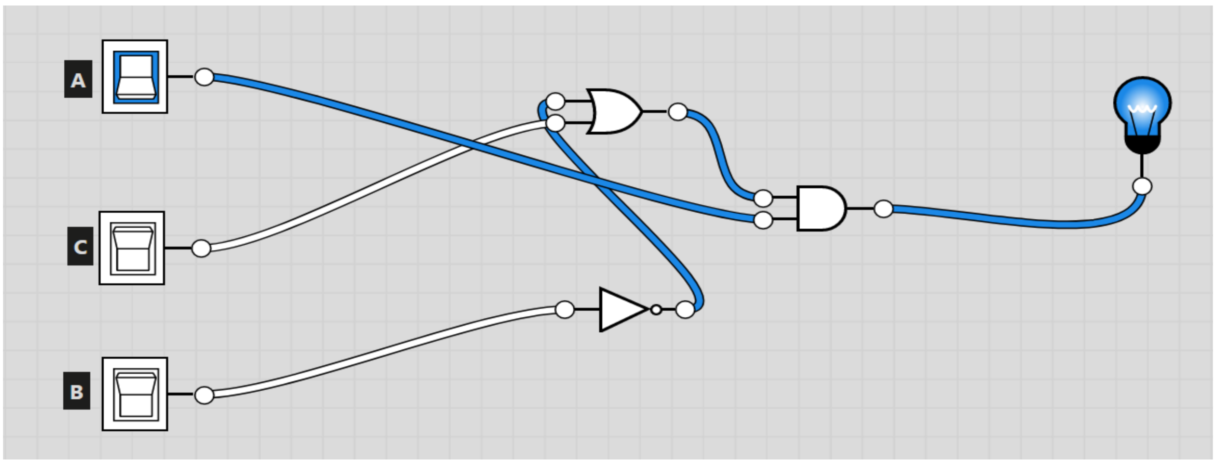


Figure 1.1: Example of a circuit's simplification

There are plenty of methods to simplify a logic circuit. In this thesis paper, I will be applying three such methods. One of the methods, for example, can be applied by taking a logic circuit and turning it into a Boolean expression. Afterwards, Boolean algebra is applied to the expression to turn it into a simpler form until the simplest form is achieved. This can be seen in **Figure 1.1**. Further ahead in the paper **different methods** will be used and a **more complex logic circuit** will be used in testing.

1.2 Motivation

When redesigning a circuit board for a certain electronic system due to the previous version of the circuit board being either too slow, it costed too much raw material or wasting too much electricity to run, either way, it being obsolete. There is the necessity to improve the obsolete version by simplifying it because, that way, the overall functionality can be kept the same while improving the results. The main goal is to design a circuit with the exact same functionality but that has either reduced material cost, electricity cost or better speed. Removing even a single logic gate can make a large difference in large scale production.

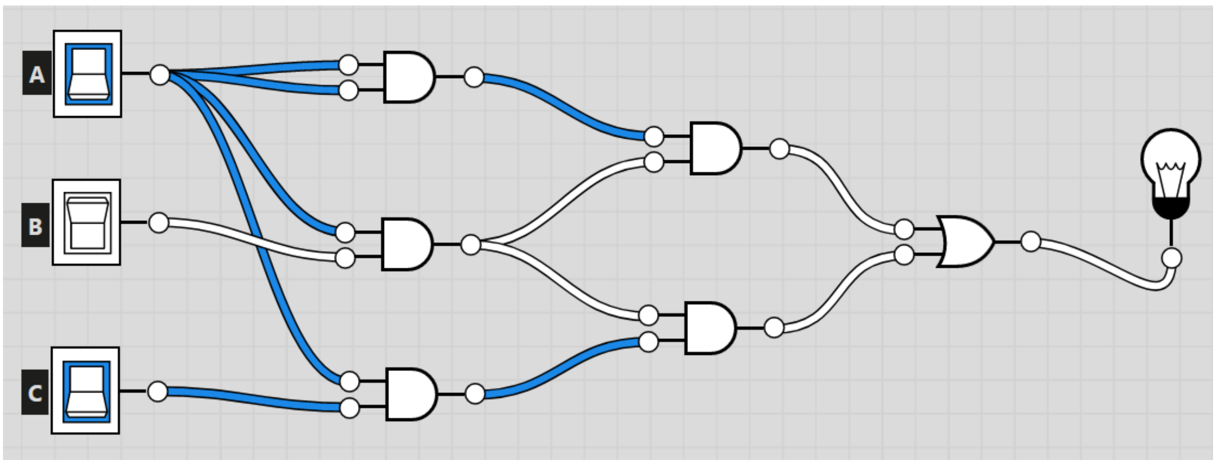


Figure 1.2: Example of a circuit

What is required to do so is to apply an algorithm that simplifies the circuit. Three different algorithms will be applied and then compared to find which of them brings better results.

Taking as an example the logic circuit in **Figure 1.2**. I will simplify it using one of the three methods. In this case, I will exemplify Boolean Algebra. By turning the circuit from **Fig1.2** into an algebraic expression we get the following expression:

$$A.A.(A.C) + A.B(A.C) \tag{1.1}$$

And when simplified we end up with:

$$A.C \tag{1.2}$$

Which is equivalent to the circuit in **Fig1.3**

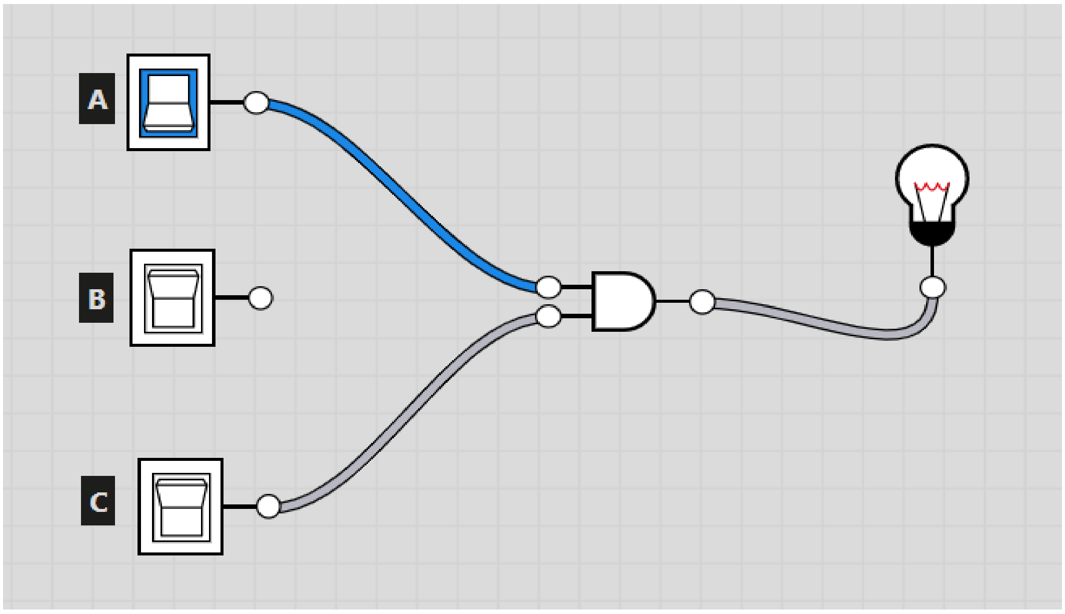


Figure 1.3: Example of simplification

Another slightly more complex example example would be:

$$AB\overline{C}D + AB\overline{C}\overline{D} + A\overline{B}\overline{C}D + ABCD + A\overline{B}CD + ABC\overline{D} + A\overline{B}C\overline{D} \tag{1.3}$$

Which turns simplifies into:

$$AB + AC + AD \tag{1.4}$$

And as a Running example i have chosen the following expression, due to its higher complexity. Its circuit equivalent can be found in **Fig1.4**.

$$\overline{A}B\overline{C}\overline{D} + \overline{A}B\overline{C}D + \overline{A}BCD + A\overline{B}\overline{C}\overline{D} + A\overline{B}C\overline{D} + A\overline{B}CD + AB\overline{C}D + ABCD \tag{1.5}$$

Which will be simplified by applying each of the three methods in detail in the following chapters

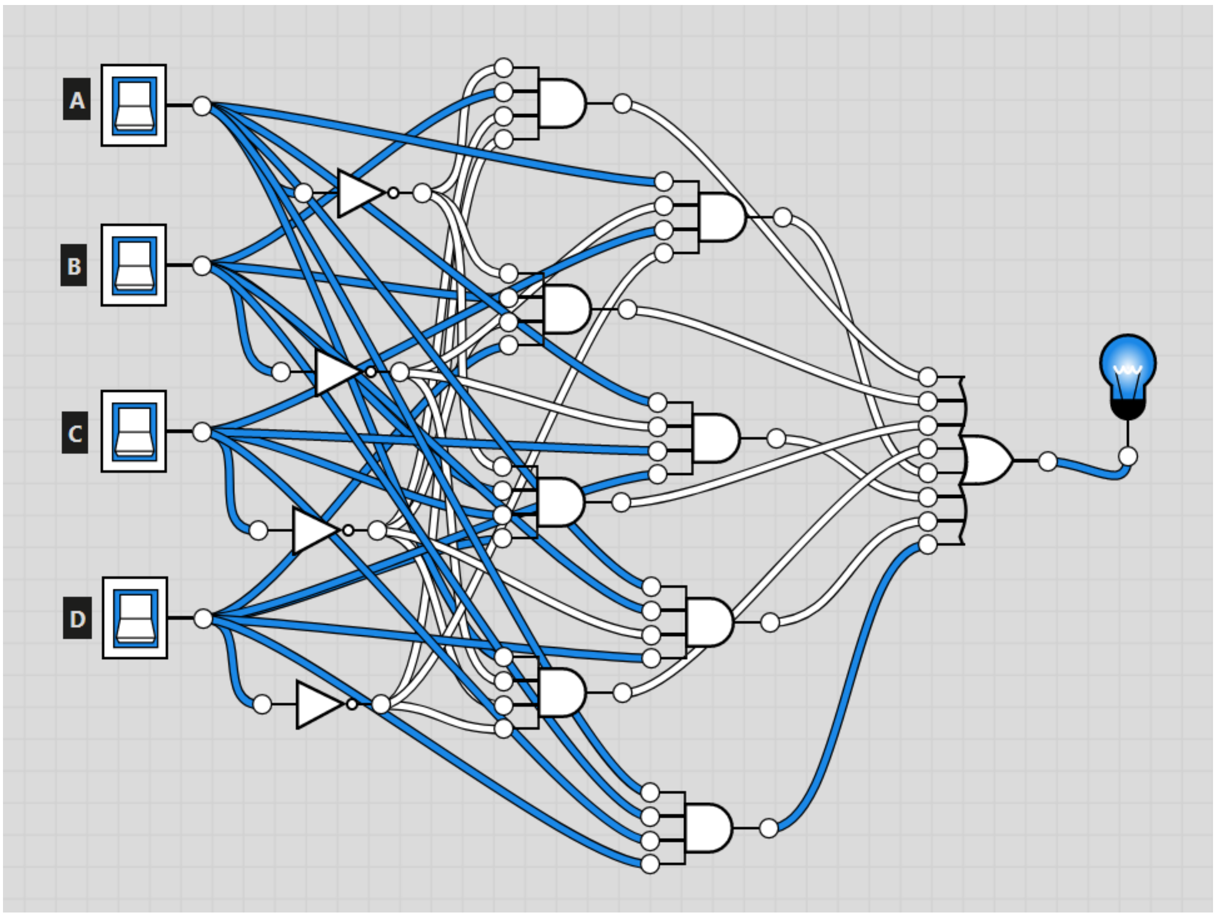


Figure 1.4: Circuit of the Running Example

1.3 Originality Declaration

While working on this thesis, I have achieved that, if the input method is a Boolean expression, for the Karnaugh map method to be applied, a negation cannot affect more than a single variable at a time. If it does, the input must be simplified previously either manually or using another method until every negation affects only a single variable.

I have also tested and confirmed that out of all three methods, the most efficient to use manually is the Quine-McClusky algorithm due to its consistency and better results.

1.4 Reading Instructions

The thesis' structure will be as follows:

- Chapter 2 starts by presenting the types of input that could be used when applying a simplification method, and how to transform from one type into another. Afterwards it presents all three simplification methods step by step on how to apply them.
- Chapter 3 introduces the implementation of the app, its requirements and features. I will also explain about the challenges that occurred and the optimization wise choices I made.
- Chapter 4 shows some related works and related apps. It explains the similarities between the works of other authors and their pros and cons in comparison to my thesis and prospective app.
- Chapter 5 shows my experiments where I test out each algorithm manually, using related apps showcased in Chapter 4 and my own app.

Chapter 2

Formal Presentation

2.1 Presentation Method

Before introducing the methods of simplifying a logic circuit it would be better to first explain the multiple ways of representing a certain logic circuit. Each method of simplification may need a different type of representation and so, it would be very helpful to have that knowledge. I will also be explaining how to change from one representation type to another and vice-versa.

2.1.1 Boolean Expression

A Boolean expression is a common way of representing a certain logic circuit where we have variables, whose values differ between 1 and 0. The number of variables is not limited but it is more common for the number of variables to be below 5. The general format is quite similar to algebraic expressions in math equations. We can have as an example, the Running example in **chapter 1.2**.

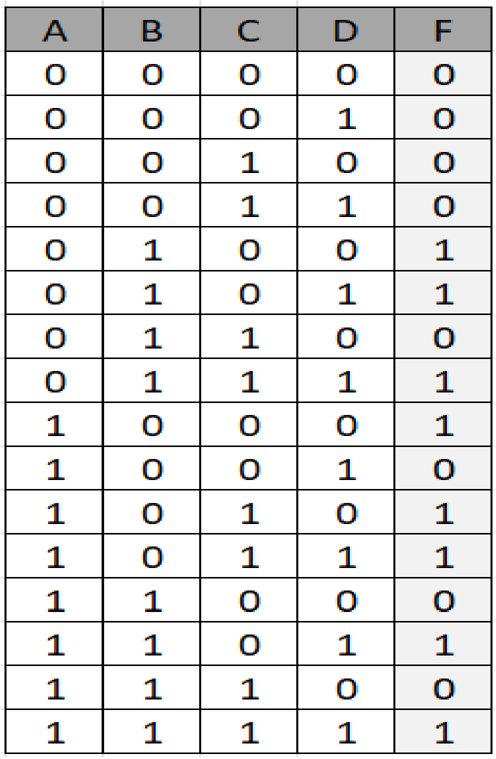
$$\overline{A}.BoverlineC\,\overline{D}+\overline{A}.BoverlineC.D+\overline{A}.B.C.D+AoverlineB\,\overline{C}\,\overline{D}+AoverlineB.CoverlineD+Aoverl \tag{2.1}$$

As it can be seen in this example, in respect to the variable A, it can be seen as A, representing the value 1, and $\overline{A}$ (also said as not A), representing the value 0.

2.1.2 Binary Table

A binary table is another common way of representing a logic circuit. Just like in the previous sub chapter, we have a certain number of values (usually below 5) and the values vary between 1 and 0. What differs is that instead of having an expression similar to algebraic expressions, there is a table. This table will have 1 column for each variable and 2^n rows, being n the number of variables. There will also be one extra column which corresponds to the solutions, which I will denominate as F.

For example see Figure 2.1:



A	B	C	D	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

Figure 2.1: Example of a Binary Table. Corresponds to the running example from **chapter 1.2**.

2.1.3 Graphical Representation

Graphical representation of a logic circuit corresponds to any physical or virtual representation using logic gates (NOT, OR, AND, etc...). A good example for this way of representation is Figure 1.1.

2.1.4 Changing the way of representation

Depending on the simplification method you want to apply, a different representation method may be needed.

From Graphical representation to Boolean expression

First of all, an example circuit is required, so I will be using the one in **Fig2.2**

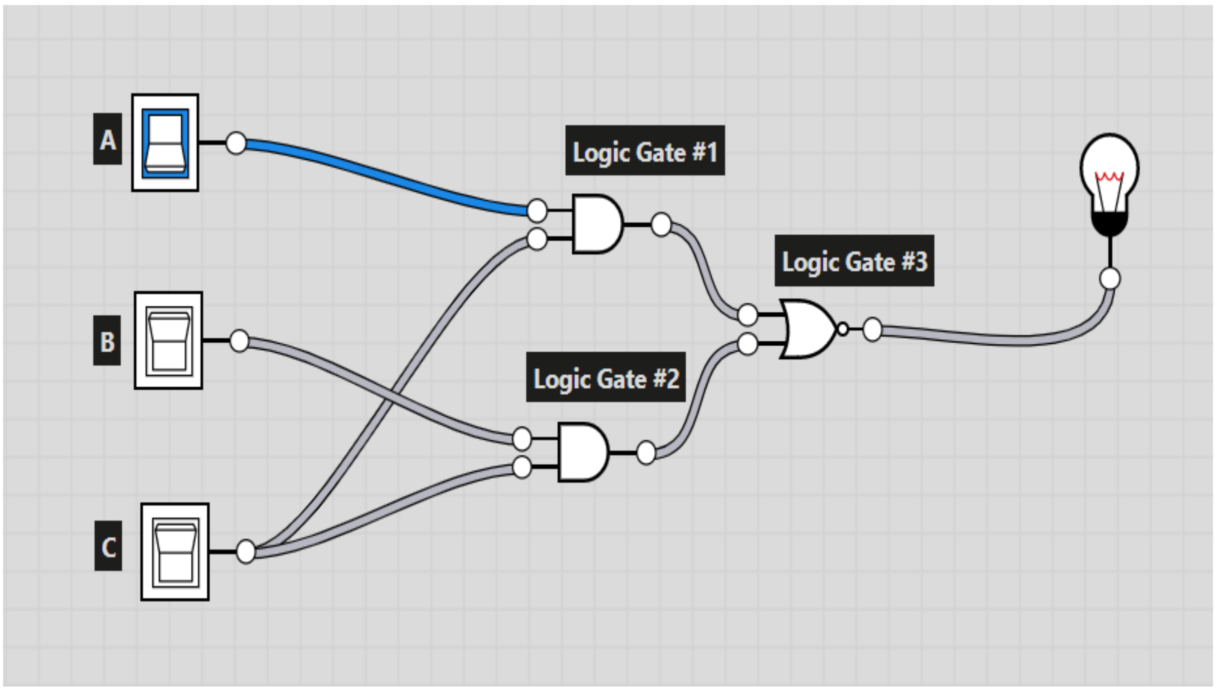


Figure 2.2: Circuit Example

To start, we need to select one of the logic gates that are directly connected to variables, then see which type of gate they are and which variables they are connected to.

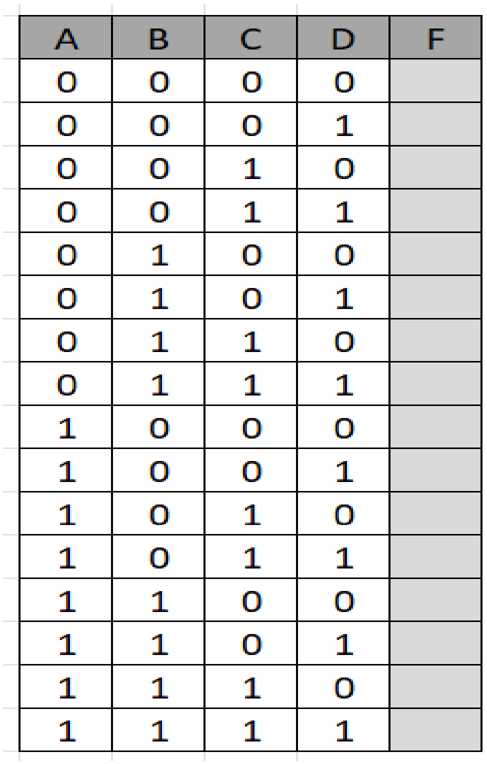
Lets choose Logic Gate number 1 from **Fig 2.2**. It is an "AND" type logic gate and it is connected to variables "A" and "C". In case you have no knowledge of logic gates, an "AND" type gate is equivalent to a "x" symbol in maths, meaning multiplication. So Logic Gate number 1 is translated into A x C which is equivalent to AC.

In the case of Logic gate number 2, it is also an "AND" type and is connected to variables "B" and "C", so it translates into BC.

For Logic Gate number 3, we have a "OR" type logic gate instead, which is equivalent to a "+" symbol in maths, also meaning sum. It is also connected to logic gates 1 and 2 instead of individual variables. In this case we sum the values of logic gates 1 and logic gates 2 which results in $AC + BC$.

From Boolean Expression to Binary Table

The way to do it is not that complicated. Firstly, the Boolean expression must be in the sum of products form, as it is in the example of **chapter 2.1.1**. Then a 4 variable binary table is needed, with a F column blank. As shown in the following Figure 2.3



A	B	C	D	F
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

Figure 2.3: Example of a 4 variable Binary Table.

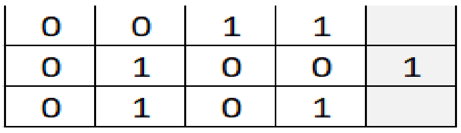
Now the F column must be filled. The Running example from chapter1.2 will be used to fill this table. Taking a look at the first term of the example:

$$\overline{A}.BoverlineC\,\overline{D} \tag{2.2}$$

We just need to turn negative variables into a 0 and positive variables into a 1 without changing the order of the variables. We get the following binary number:

$$0100 \tag{2.3}$$

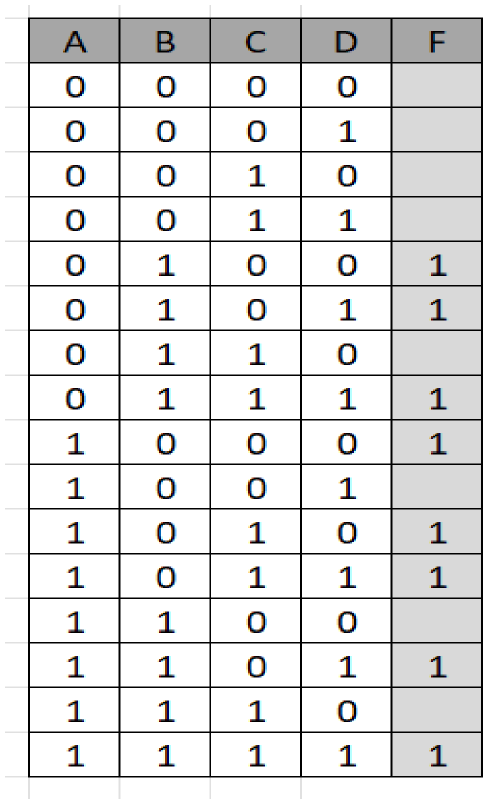
Now, taking a look at the binary table, it is needed to find the row that contain the values 0100 (in that exact order). The corresponding row is the fifth one, so now we fill, in the same row, the F column with a 1. See Figure 2.4



0	0	1	1	
0	1	0	0	1
0	1	0	1	

Figure 2.4: Filling the 5Th row

And we repeat the same procedure for all the products. Resulting in Figure 2.5



A	B	C	D	F
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	1
0	1	0	1	1
0	1	1	0	
0	1	1	1	1
1	0	0	0	1
1	0	0	1	
1	0	1	0	1
1	0	1	1	1
1	1	0	0	
1	1	0	1	1
1	1	1	0	
1	1	1	1	1

Figure 2.5: Filled Binary Table

The remaining empty spaces are filled with a 0 which ends up being the same as Figure 2.1.

From Binary Table to Boolean Expression

To turn a binary table into a Boolean expression is doing the opposite of what was done in **2.1.4.** By taking a look at each row that contains a solution (where F is 1), the binary number corresponding to each row will be turned into a product of variables, and each product of variables will then be summed to the other rows. For example, in row 5 of **figure 2.5**, the binary number of that row is 0100 which by turning it into a product of variables turns into $\overline{A}.BoverlineC\,\overline{D}$.

The same is done to all rows and then they are summed resulting in the Boolean expression of **chapter 2.1.1**.

2.2 Boolean Algebra

To simplify a Boolean expression it is required to follow a set of rules that are as shown in the following table:

Name	AND Form	OR Form
Identity Law	$1A = A$	$0 + A = A$
Null Law	$0A = A$	$1 + A = 1$
Idempotent Law	$AA = A$	$A + A = A$
Inverse Law	$A\overline{A} = 0$	$A + \overline{A} = 1$
Commutative Law	$AB = BA$	$A + B = B + A$
Associative Law	$(AB)C = A(BC)$	$(A + B) + C = A + (B + C)$
Distributive Law	$A + BC = (A + B)(B + C)$	$A(B + C) = AB + AC$
Absorption Law	$A(A + B) = A$	$A + AB = A$
DeMorgan's Law	$\overline{AB} = \overline{A} + \overline{B}$	$(\overline{A + B}) = \overline{A}\,\overline{B}$

By following this table of rules, we can use any of its rules to simplify any Boolean expression as long as the rule can be applied.

-Solving step-by-step of the running example

$$\overline{A}.BoverlineC\,\overline{D}+\overline{A}.BoverlineC.D+\overline{A}.B.C.D+AoverlineB\,\overline{C}\,\overline{D}+AoverlineB.CoverlineD+Aoverl \tag{2.4}$$

we start by applying the distributive law to three terms

$$\overline{A}.BoverlineC(D+\overline{D})+AoverlineB\,\overline{D}(C+\overline{C})+\overline{A}.B.C.D+AoverlineB.C.D+A.B.D(C+\overline{C}) \tag{2.5}$$

then we apply the Inverse law

$$\overline{A}.BoverlineC.1+AoverlineB\,\overline{D}.1+\overline{A}.B.C.D+AoverlineB.C.D+A.B.D.1 \tag{2.6}$$

then we apply the Identity law

$$\overline{A}.BoverlineC + AoverlineB\,\overline{D} + \overline{A}.B.C.D + AoverlineB.C.D + A.B.D \tag{2.7}$$

This is as far as the simplification can go since no other rule can be applied. The results can change depending on which rules are applied and where you apply them. In this case, the solution could not be further simplified using Boolean algebra, at most, it can be factorized. The resulting circuit is shown in Figure 2.6

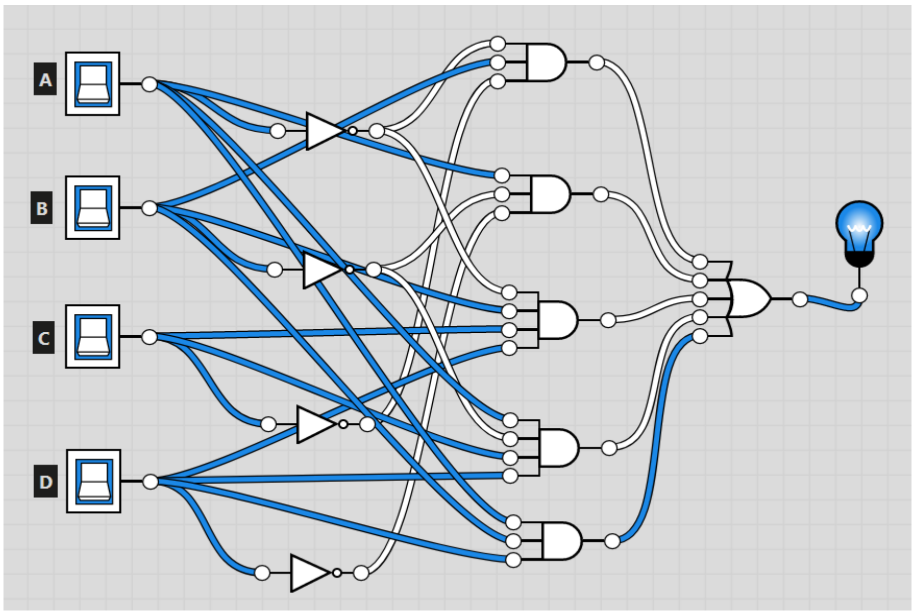


Figure 2.6: Running example's circuit after simplifying using Boolean Algebra

2.3 Karnaugh Maps

To apply the Karnaugh Map method it is recommended to have a Binary Table of the logic circuit that is needed to simplify, although it is also possible to apply to a Boolean expression. The basic Karnaugh map looks as in Figures 2.7 and 2.8

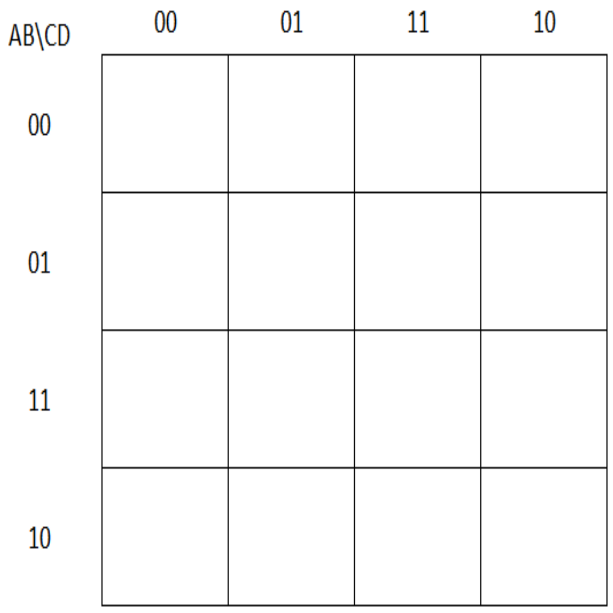


Figure 2.7: Basic 4 Variable Karnaugh Map

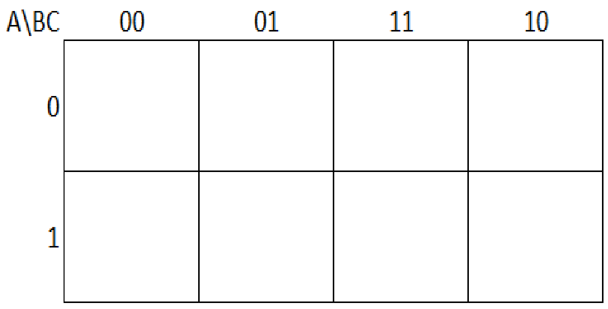
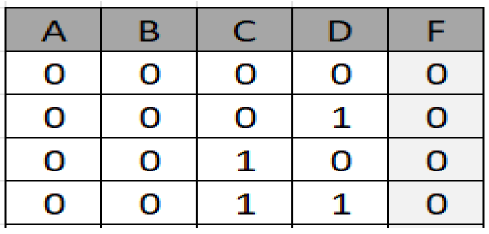


Figure 2.8: Basic 3 Variable Karnaugh Map

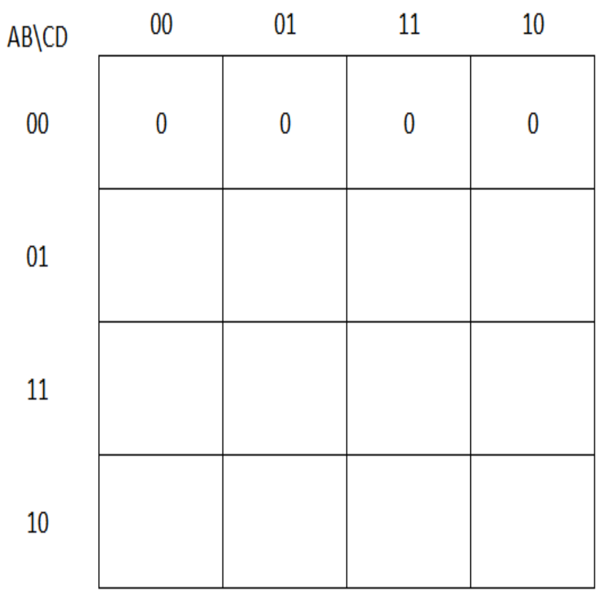
To apply the method, the Karnaugh map must be filled up. To do so, it is first required to understand how the map works. Behind each row and above each column there is a 2-digit binary number that work as ”coordinates”. Taking the Binary table from figure 2.1 as an example. In the first row, it can be seen that the variables A,B,C,D have the value 0, which corresponds to the first square of the Karnaugh map. See Figure 2.9



A	B	C	D	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0

Figure 2.9: First for rows of the binary table

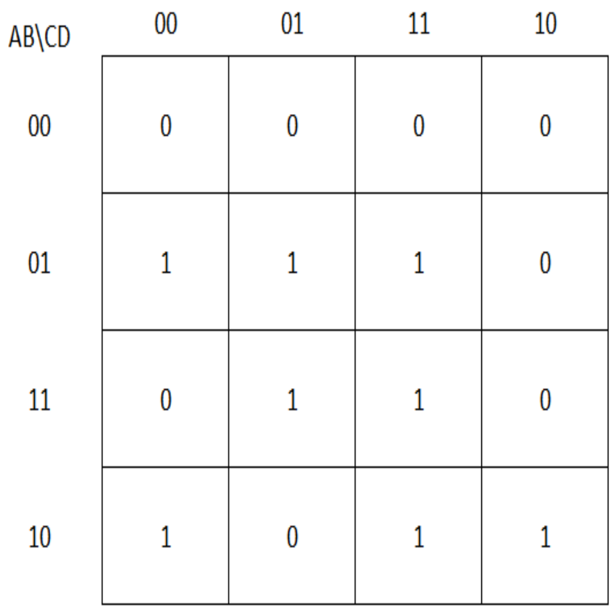
In the first row of the binary table, the value of F is 0. Therefore, when representing the function F using a Karnaugh map, the corresponding cell in the map for the first row should also be 0. This process is repeated for the remaining three rows of the binary table, resulting in the Karnaugh map shown in Figure 2.10.



AB\CD	00	01	11	10
00	0	0	0	0
01				
11				
10				

Figure 2.10: First row of the Karnaugh Map

By slowly filling up the remaining cells of the Karnaugh map based on the values in the binary table, we arrive at the following Karnaugh map. Please refer to Figure 2.11 for visual representation.



AB\CD	00	01	11	10
00	0	0	0	0
01	1	1	1	0
11	0	1	1	0
10	1	0	1	1

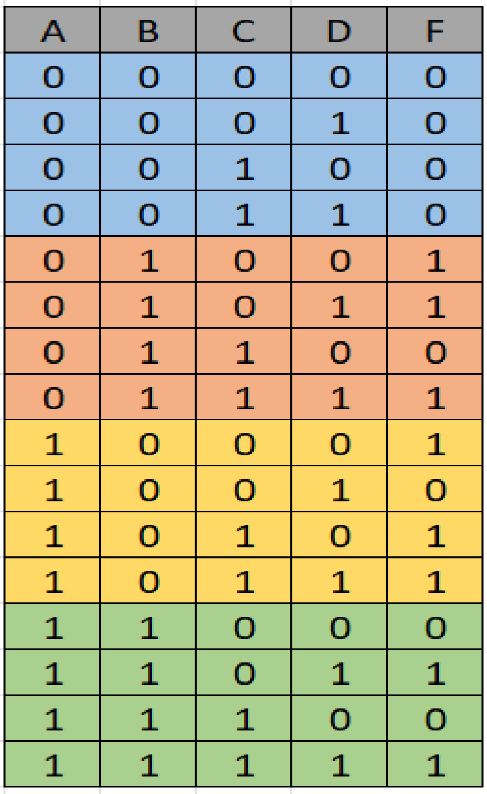
Figure 2.11: Completed Karnaugh Map

Filling the Karnaugh map using the traditional method can be quite time-consuming, requiring a significant amount of our time just to complete a single map.

However, in order to address this issue and expedite the process, I will now present an alternative method for filling a Karnaugh map that significantly reduces the time required.

Quick method to fill a Karnaugh Map

If a binary table is divided in groups of 4 rows just like in the following Figure 2.12

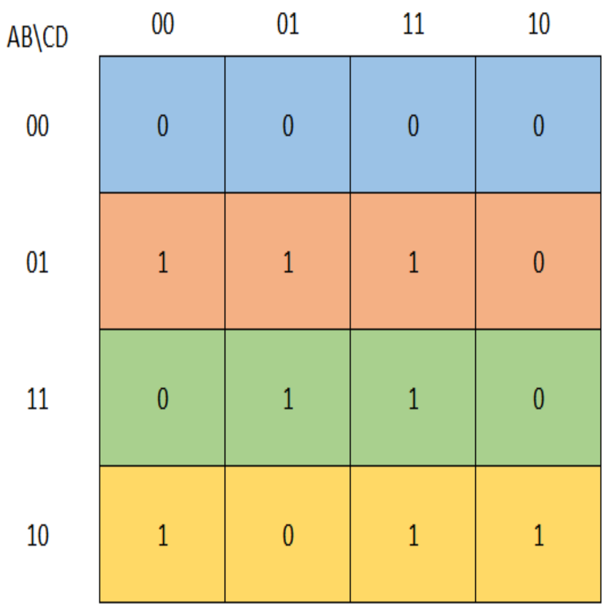


A	B	C	D	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

Figure 2.12: Binary table divided in groups of 4 rows

If you fill up the Karnaugh map using the solutions (F column) from the table from top to bottom while swapping the 3rd solution of each group with the 4th solution while also swapping the 3rd group with the 4th group, you can quickly fill up the Karnaugh map.

By following the same color pattern, the Karnaugh map will end up as in the following Figure 2.13



AB\CD	00	01	11	10
00	0	0	0	0
01	1	1	1	0
11	0	1	1	0
10	1	0	1	1

Figure 2.13: Completed Karnaugh Map

This way, the time it takes to fill up a Karnaugh map can be heavily reduced.

Now that the Karnaugh map is completed, it is required to observe the following rules:

- pair up the 1's in the squares in groups of 2^n (1,2,4,8 or 16)
- two 1's cannot pair diagonally unless there is a 1 horizontally and vertically connected
- The same 1 can be part of multiple pairs
- The number of pairs must be the least possible
- A left most 1 can pair with a right most 1 of the same row
- A bottom most 1 can pair with a top most 1 of the same column

By applying the rules the Karnaugh map becomes as shown in Figure 2.14

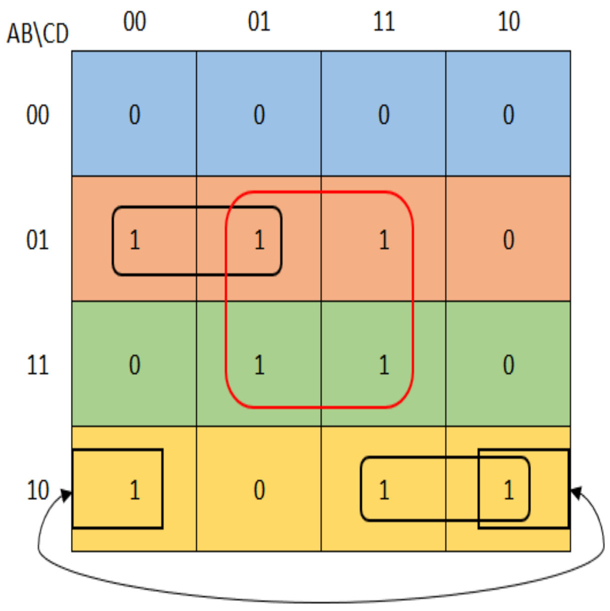


Figure 2.14: Completed Pairing

Now that the pairs are completed, it is necessary to observe the 2-digit binary numbers above and on the left. And for a better explanation, an example will be required. See Figure 2.15

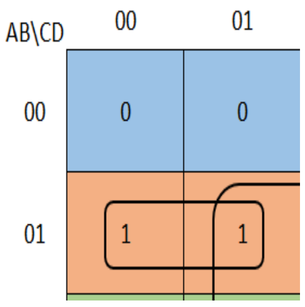


Figure 2.15: Example two 1 pair: $\overline{A}B\overline{C}$

Taking this two 1's pair as an example, if we observe the 2-digit numbers, it is needed to find the digits that **DO NOT** change, no matter the element of the pair we look at. The number 01 corresponding to AB encompasses the whole pair and does not change if we observe the left-most 1 or the right-most 1. When observing the top digits, only the 0 corresponding to C does not change while D changes between 1 and 0. Now we turn the digits that did not change into their corresponding variables and multiply one another resulting in the next literal product:

$$\overline{A}B\overline{C} \tag{2.8}$$

Now, the same is done to the other groups. See Figures 2.16, 2.17 and 2.18

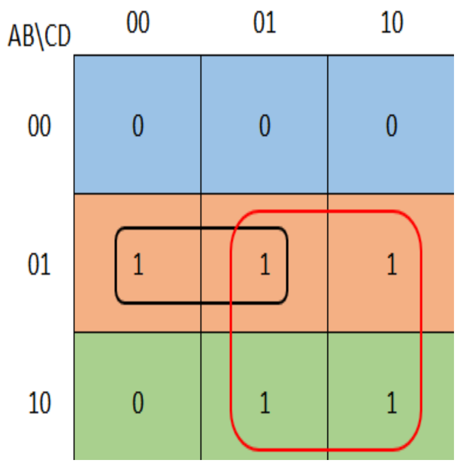


Figure 2.16: Four 1 pair: AC

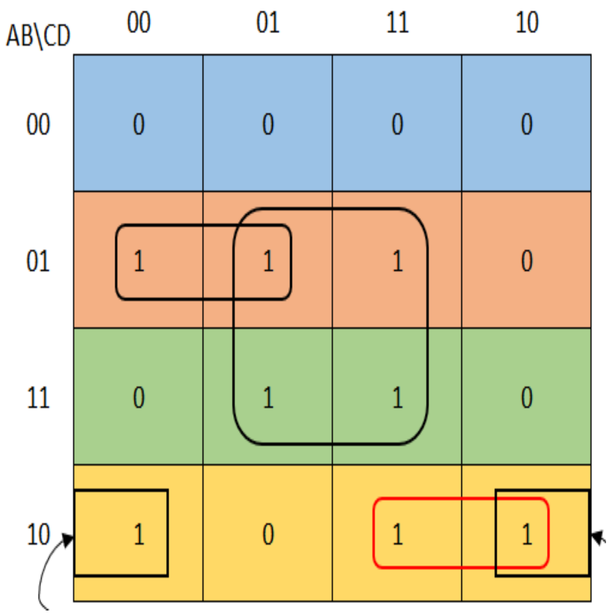


Figure 2.17: Two 1 bottom pair: $B\overline{C}D$

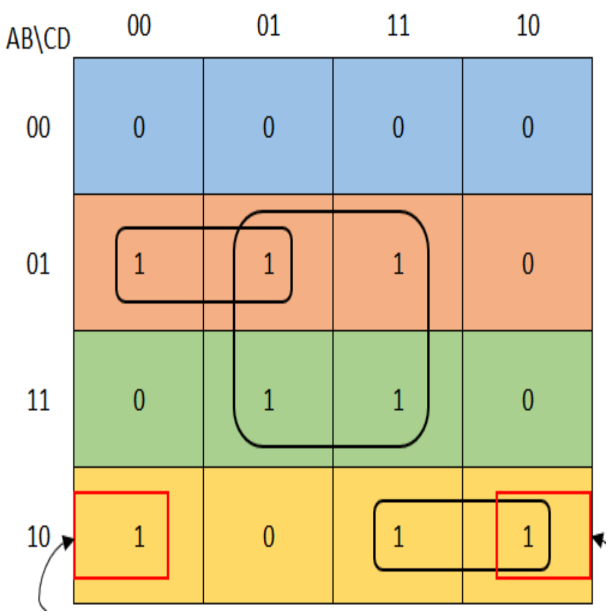


Figure 2.18: Two 1 corner pair: $\overline{A}\,\overline{C}D$

After turning all pairs into products, we retrieve the literal products from Figures 2.15, 2.16, 2.17, 2.18 and we sum all of them resulting in:

$$\overline{A}B\overline{C} + AC + B\overline{C}D + \overline{A}\,\overline{C}D$$

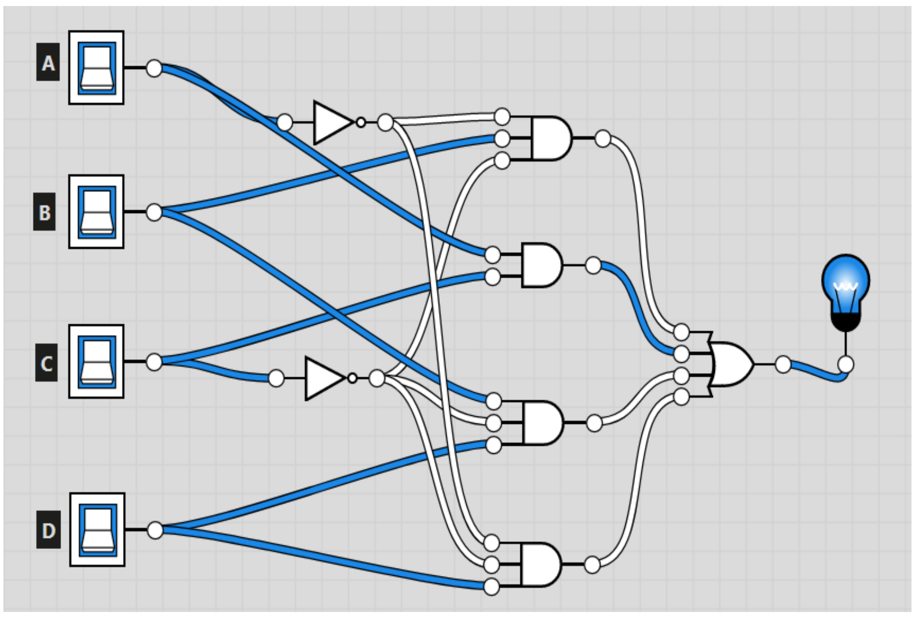


Figure 2.19: Running example's circuit after simplifying using the Karnaugh map method

As it can be seen, the resulting expression will have the sum of products form. It may happen in some cases that after applying the method, although simplified, the solution will not be minimal (meaning that it can still be simplified). To further simplify, it is recommended to use Boolean algebra. The resulting circuit can be seen in Figure 1.4

2.3.1 Don't Cares

In some cases, there are certain values in a logic circuit that can be considered irrelevant, meaning that they can take on any value, whether it's 1 or 0. In Karnaugh maps, these values are represented by an 'X'.

When dealing with don't care conditions in Karnaugh maps, if we encounter a 1 that has no adjacent 1's but has an adjacent 'X', we can pair them up. However, it's important to note that 'X's can only be used for pairing when there is at least one 1 present. The goal is to form groups with as many 1's and as few 'X's as possible. You can refer to **Figure 2.20** for an example demonstrating the pairing of don't care conditions.

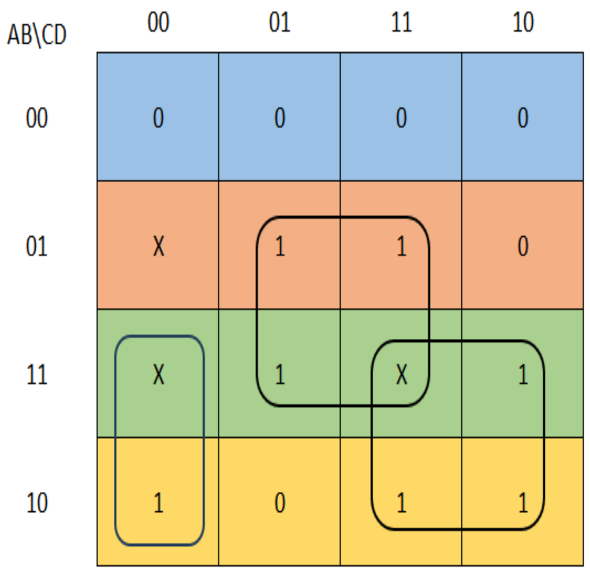


Figure 2.20: Example of pairing Don't cares

2.3.2 Alternate pairing method

There is an alternate method that can be used that is not as common, instead of pairing 1's, we pair 0's but that is not the only difference. When using this method, the resulting pairs do not generate an expression in the sum of products form, instead, it generates an expression in the product of sums form. Refer to **Figure 2.21**.

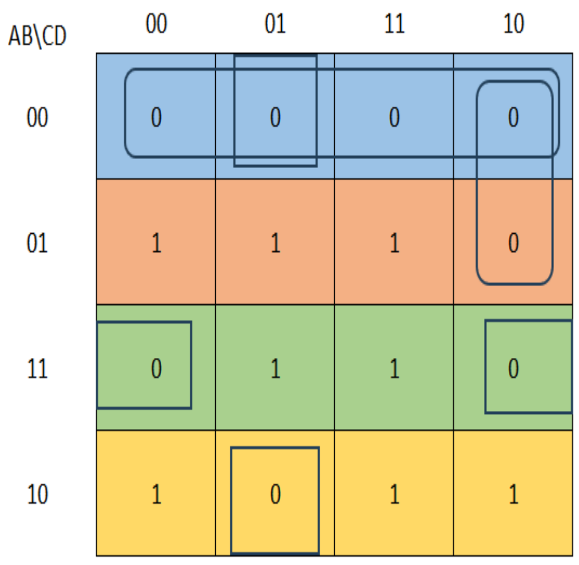


Figure 2.21: Alternate pairing method example

When using this method, instead of multiplying the top or left digits that do not change, we add them for each group and then multiply the groups. By applying this method to the map in **Figure 2.21** we get the following terms: $(\overline{A}+\overline{B})$,$(A+B+\overline{D})$, $(\overline{B}+\overline{C}+D)$,$(\overline{A}+C+\overline{D})$

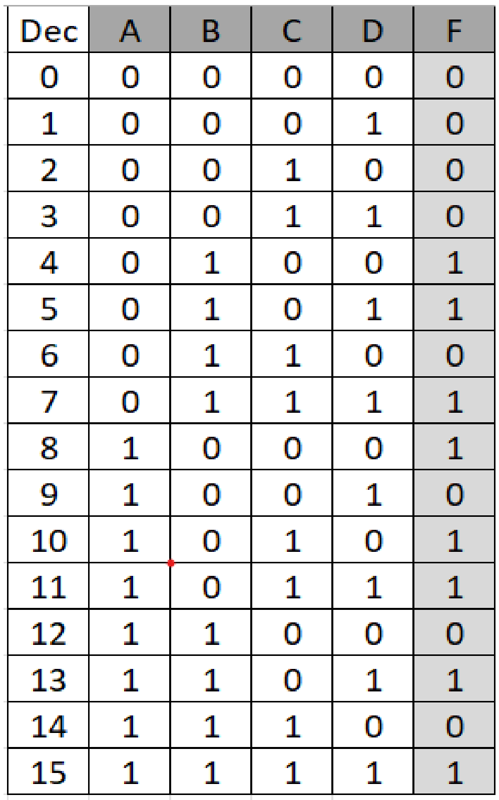
Now we simply multiply them by one another getting the following expression:

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

As you can see, the resulting expression does not look as simple as the one resulted from applying the regular method, as to why it is less commonly used. If you apply the distributive rule to the resulting expression, you will end up with the same expression as the running example shown in chapter 2.1.1.

2.4 Quine McCluskey

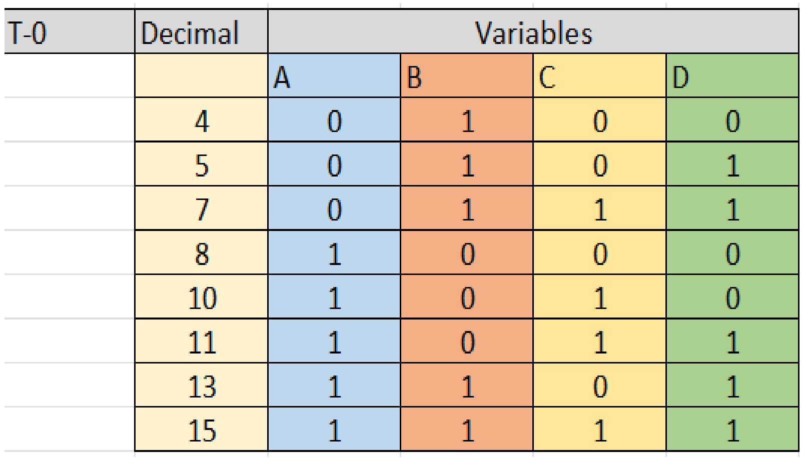
When applying this method, it is recommended to start from a Binary Table. For this method, the table used will be the same as in Figure 2.1. It is also recommended to have a column representing the decimal value of the binary number just like in the following Figure 2.22.Note: that in this chapter, the colors in the tables are irrelevant:



Dec	A	B	C	D	F
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	1
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	0
15	1	1	1	1	1

Figure 2.22: Binary Table

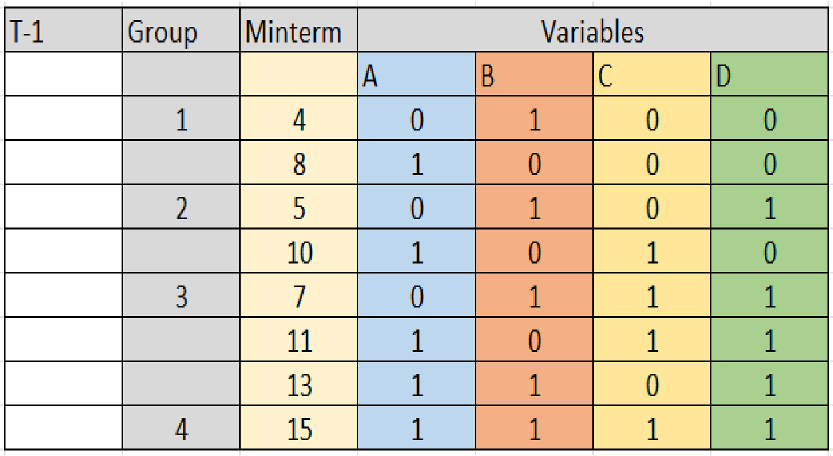
Then, to proceed, create a new table with every solution (every row where F is equal to 1) just like in the following Figure 2.23:



T-0	Decimal	Variables			
		A	B	C	D
	4	0	1	0	0
	5	0	1	0	1
	7	0	1	1	1
	8	1	0	0	0
	10	1	0	1	0
	11	1	0	1	1
	13	1	1	0	1
	15	1	1	1	1

Figure 2.23: Quine McClusky Step 1

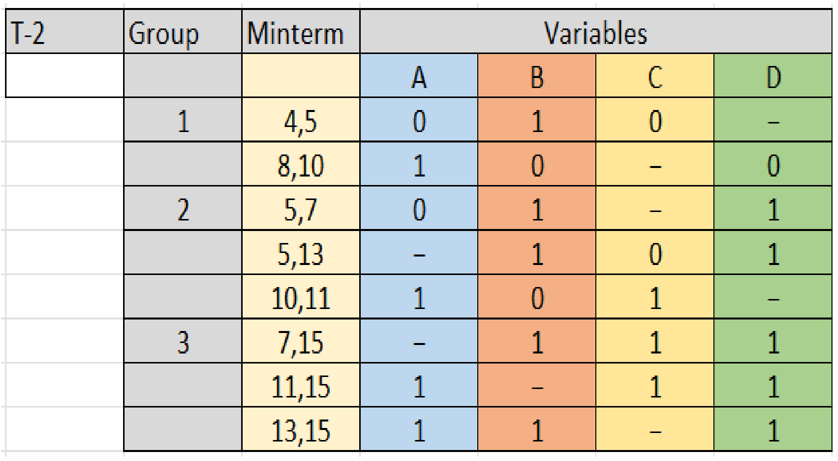
Now, it is needed to repeat the same as done in Figure 2.23 but instead of ordering the rows by decimal value, they are ordered by the quantity of 1's in each row. If two rows have the same quantity of 1's per row, they are ordered by decimal value, just like in the next Figure 2.24:



T-1	Group	Minterm	Variables			
			A	B	C	D
	1	4	0	1	0	0
		8	1	0	0	0
	2	5	0	1	0	1
		10	1	0	1	0
	3	7	0	1	1	1
		11	1	0	1	1
		13	1	1	0	1
	4	15	1	1	1	1

Figure 2.24: Quine McClusky Step 2

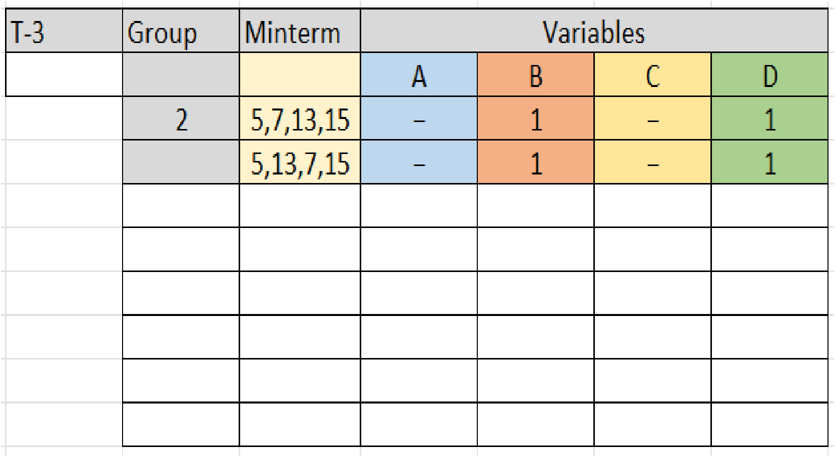
In Figure 2.24, it can be seen that a new column "Group" was added, it corresponds to the number of 1's per row. Now, to proceed, it is required to compare each row to another row from a one level higher group. This means that you have to compare each row from group 1 to a row each row from group 2, always 1 level higher, **NEVER** 2 levels higher. When comparing, check if, between both rows, in the variables columns, there is only one column with a different value, just like between the row with decimal value 4 and 5 from Figure 2.24. The only changes between are in column D. To proceed, both rows are merged into one and the values in the column that was different are changed with a line just like in the following Figure 2.25:



T-2	Group	Minterm	Variables			
			A	B	C	D
	1	4,5	0	1	0	–
		8,10	1	0	–	0
	2	5,7	0	1	–	1
		5,13	–	1	0	1
		10,11	1	0	1	–
	3	7,15	–	1	1	1
		11,15	1	–	1	1
		13,15	1	1	–	1

Figure 2.25: Quine McClusky Step 3

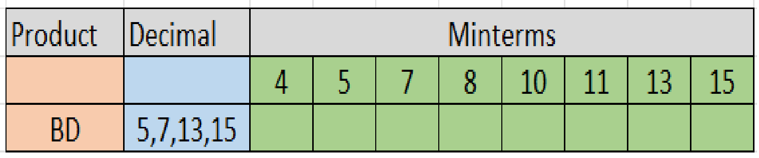
And we repeat the same procedure as in step 3 until no more comparisons can be made. See Figure 2.26



T-3	Group	Minterm	Variables			
			A	B	C	D
	2	5,7,13,15	–	1	–	1
		5,13,7,15	–	1	–	1

Figure 2.26: Quine McClusky Step 4

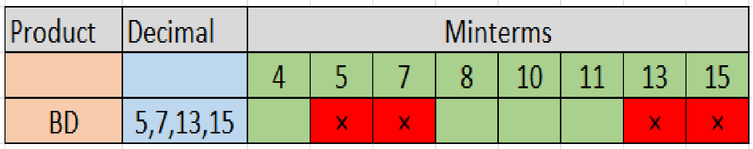
Now a new table must be made, with a column for every solution (also called minterm) from Figure 2.23. Now from the remaining rows in Figure 2.26, turn them into a literal product. Since both rows have the same minterms and same variables, we merge them into a single row and get the following Figure 2.27:



Product	Decimal	Minterms							
		4	5	7	8	10	11	13	15
BD	5,7,13,15								

Figure 2.27: Quine McClusky Step 5

And now, mark an ”X” on every column for every minterm that coincides with one of the decimal values, just as in Figure 2.28

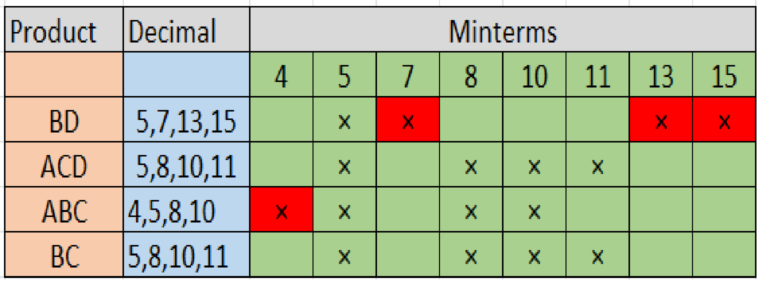


Product	Decimal	Minterms							
		4	5	7	8	10	11	13	15
BD	5,7,13,15		x	x				x	x

Figure 2.28: Quine McClusky Step 6

For every row that contains at least one column that has at most one "X", it is summed, since there is only one row, the final solution is BD.

Since in Figure 2.28 there is only one row, the step may not be visualized well so I will present a new made up table. See Figure 2.29



Product	Decimal	Minterms							
		4	5	7	8	10	11	13	15
BD	5,7,13,15		x	x				x	x
ACD	5,8,10,11		x		x	x	x		
ABC	4,5,8,10	x	x		x	x			
BC	5,8,10,11		x		x	x	x		

Figure 2.29: Quine McClusky Made Up Example

As seen in Figure 2.29, the columns with only one "X" are marked with red. By retrieving the rows that contain those "X"s we get the following Boolean expression :

$$BD + ABC$$

The solution gotten is in its minimal form, meaning that it can not be simplified more that it has been. See the resulting circuit in Figure 2.30

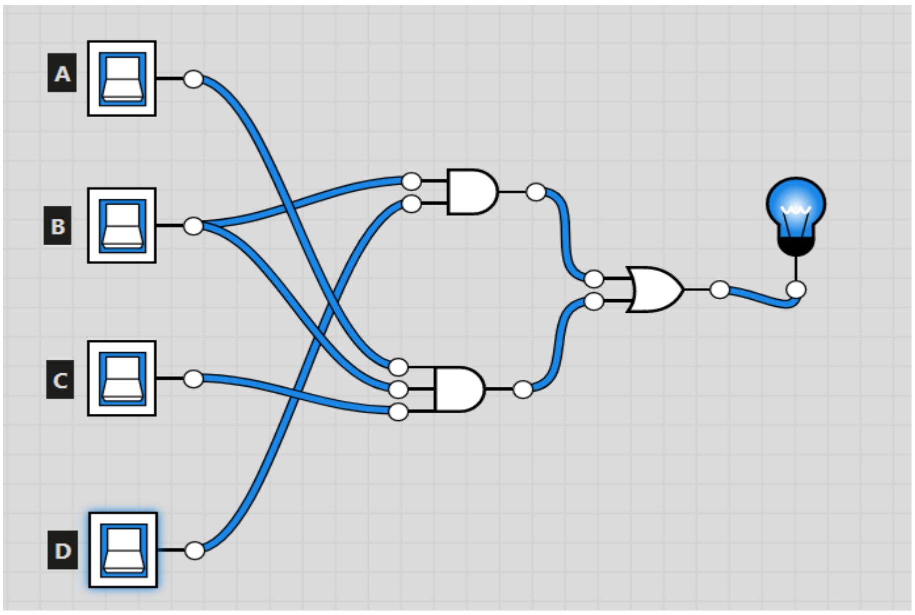


Figure 2.30: Running example's circuit after simplifying using the Quine McCluskey algorithm

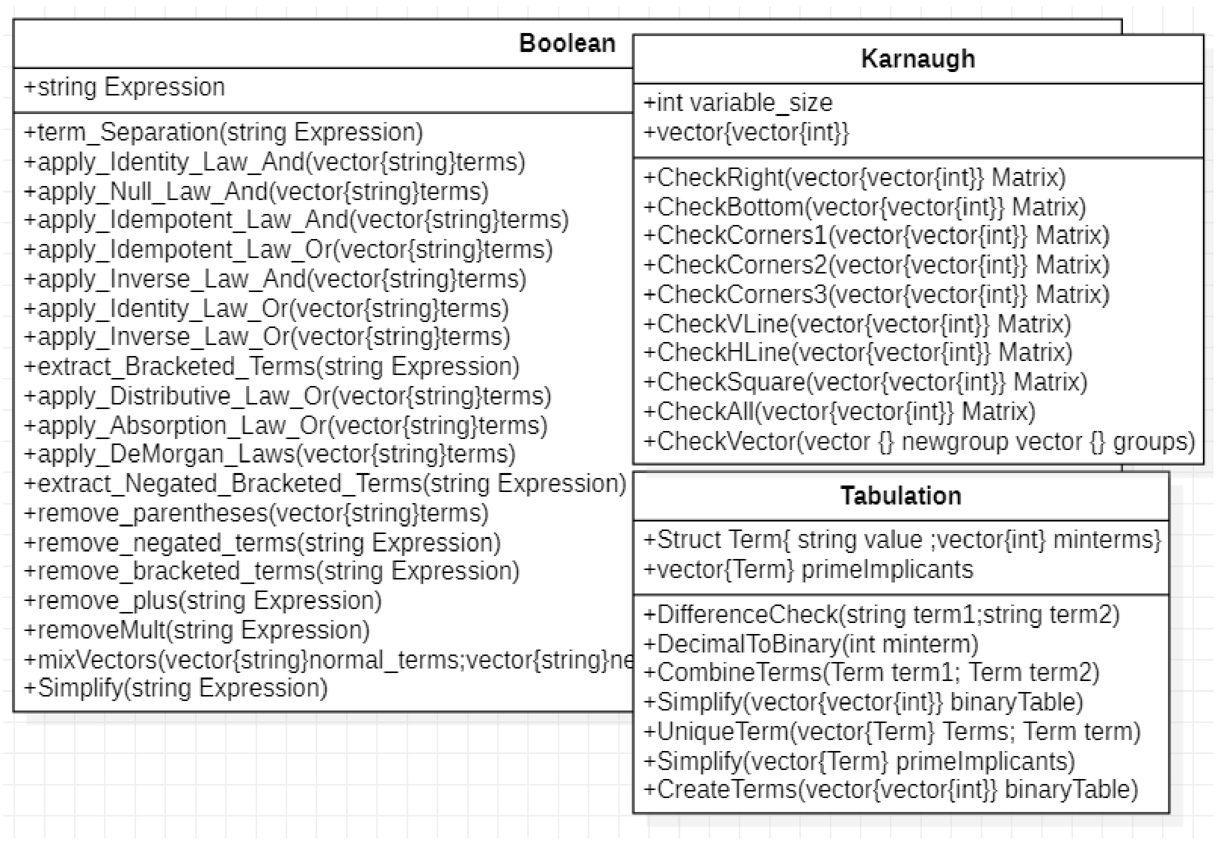
Method comparison

By comparing all three methods, it can be easily observed that Quine McClusky is a superior method in comparison to the other two since it consistently achieves the best simplification. Boolean Algebra and Karnaugh maps may achieve different solutions since the way you approach may differ and result in a better or a worse solution. Such can be seen by observing Figures 2.6, 2.19 and 2.30

Chapter 3

Implementation

In this chapter, I will provide a detailed account of the development process behind my application, offering insights into the coding and the algorithms implemented, as well as shedding light on the underlying thought processes that guided my decision-making throughout the project. My app is currently divided in three different classes, one for each algorithm I applied. The app itself is not easy to use so I will be explaining how to use the application further ahead in the chapter. The application is based on the terminal. In the following **Figure3.1** you can see them in more detail. In the top section of each class you have the stored variables and in the down section you have each function belonging to the class.



```
Boolean
+string Expression
+term_Separation(string Expression)
+apply_Identity_Law_And(vector{string}terms)
+apply_Null_Law_And(vector{string}terms)
+apply_Idempotent_Law_And(vector{string}terms)
+apply_Idempotent_Law_Or(vector{string}terms)
+apply_Inverse_Law_And(vector{string}terms)
+apply_Identity_Law_Or(vector{string}terms)
+apply_Inverse_Law_Or(vector{string}terms)
+extract_Bracketed_Terms(string Expression)
+apply_Distributive_Law_Or(vector{string}terms)
+apply_Absorption_Law_Or(vector{string}terms)
+apply_DeMorgan_Laws(vector{string}terms)
+extract_Negated_Bracketed_Terms(string Expression)
+remove_parentheses(vector{string}terms)
+remove_negated_terms(string Expression)
+remove_bracketed_terms(string Expression)
+remove_plus(string Expression)
+removeMult(string Expression)
+mixVectors(vector{string}normal_terms;vector{string}ne
+Simplify(string Expression)
```

```
Karnaugh
+int variable_size
+vector{vector{int}}
+CheckRight(vector{vector{int}} Matrix)
+CheckBottom(vector{vector{int}} Matrix)
+CheckCorners1(vector{vector{int}} Matrix)
+CheckCorners2(vector{vector{int}} Matrix)
+CheckCorners3(vector{vector{int}} Matrix)
+CheckVLine(vector{vector{int}} Matrix)
+CheckHLine(vector{vector{int}} Matrix)
+CheckSquare(vector{vector{int}} Matrix)
+CheckAll(vector{vector{int}} Matrix)
+CheckVector(vector {} newgroup vector {} groups)
```

```
Tabulation
+Struct Term{ string value ;vector{int} minterms}
+vector{Term} primeImplicants
+DifferenceCheck(string term1;string term2)
+DecimalToBinary(int minterm)
+CombineTerms(Term term1; Term term2)
+Simplify(vector{vector{int}} binaryTable)
+UniqueTerm(vector{Term} Terms; Term term)
+Simplify(vector{Term} primeImplicants)
+CreateTerms(vector{vector{int}} binaryTable)
```

Figure 3.1: Classes and Functions Used

3.1 Boolean

The Boolean class was the first one to be implemented and in my opinion, the hardest class to implement, simply due to the overwhelming amount of functions I had to implement. When implementing each Boolean law, in many cases I decided to split their OR form from their AND form. You can see it in **Figure3.1**, the functions that have an "OR" or an "AND" at the end of their name mean the form of the law that was implemented. Those that do not have anything at the end of their name apply both forms.

Negations are represented by a "!" before what is supposed to be negated. The starting Boolean expression was split into multiple terms using the "+" as a reference, in cases where the terms contained a "+" between parentheses, the whole expression between the parentheses is considered as a term.

The simplify function is the core function of the class where all of the Boolean rules are called.

I chose to split my starting Boolean expression into 3 vectors of terms according to their category. The categories were:

- Normal Term - terms like $ABCD$ or $\overline{A}A$
- Bracketed Term - terms with parentheses, like $AB(ABC)$ or $(AB(C(\overline{D}E))$
- Negated Bracketed Term - terms that contain a parentheses that is negated in its entirety, like $\overline{(AB)}$ or $\overline{(A+B)}$

The reason behind separating the terms into categories was because, some laws only apply to some categories of terms. For example, the Distributive Law only applies to terms that contain parentheses. Also, the DeMorgan's laws only apply to terms that contain negated parenthesis. For that reason, I chose to split the terms into three categories. After splitting and applying those specific rules to each category, I mix them back together and I apply the rules that are common to all categories.

The order in which the laws are applied is meaningful towards optimization, for example if you apply a rule before the distributive law, you may have to apply it again if you want to have better results since the distributive law gets rid of the parentheses and generates more terms.

```
string Boolean::Simplify(string expression){
//string expression = "A + !A ";
cout << "Starting expression:" << expression << endl;
vector<string> Normal_Terms;
vector<string> Bracketed_Terms;
vector<string> Negated_Bracketed_Terms;
Boolean A;
if(expression.empty())
    return 0;
Negated_Bracketed_Terms = A.
   extract_Negated_Bracketed_Terms(expression);
if(!Negated_Bracketed_Terms.empty())
    for (int i = 0; i < Negated_Bracketed_Terms.size(); i
       ++)
        cout << "Extracted negated terms:" <<
           Negated_Bracketed_Terms[i] << endl;
expression = A.remove_negated_terms(expression);
expression = A.removePlus(expression);
cout << "Expression after removing negated terms:" <<
   expression << endl;

Bracketed_Terms = A.extract_Bracketed_Terms(expression);
if(!Bracketed_Terms.empty())
    for (int i = 0; i < Bracketed_Terms.size(); i++)
        cout << "Extracted bracketed terms:" <<
           Bracketed_Terms[i] << endl;

expression = A.remove_bracketed_terms(expression);

cout << "Expression after removing bracketed terms:" <<
   expression << endl;

Normal_Terms = A.term_Separation(expression);

for (int i = 0; i < Normal_Terms.size(); i++)
    cout << "Separated normal terms:" << Normal_Terms[i]
       << endl;
if(!Negated_Bracketed_Terms.empty()) {
    Negated_Bracketed_Terms = A.apply_DeMorgan_Laws(
       Negated_Bracketed_Terms);

    for (int i = 0; i < Negated_Bracketed_Terms.size(); i
       ++)
        cout << "Negated terms after applying De Morgan's
           :" << Negated_Bracketed_Terms[i] << endl;
    (...)
```

Listing 3.1: Boolean Class Code Snippet

3.2 Karnaugh

The Karnaugh class was the second to be implemented and it gave me quite the conundrum. How would I implement an algorithm that can be applied to any Karnaugh map, because Karnaugh maps can vary in their sizes.

After discussing with my coordinating teacher I came to a decision. I chose to iterate through each element of the map and if the element's value is a 1, I will then check the square at its left and the square below it. If the square at its left is also an 1, I will then check the whole line, if the line is also composed of only 1's I will store the coordinates of its points in a vector, if not I will only store the small group composed of 2 points.

The same applies to the bottom square. In the case that both the left square and the bottom square are 1's I will check if a bigger 2x2 square can be formed by checking if the diagonal square is also an 1, if it is, I will store its points in a vector, if not I will store 2 different groups of two points.

Before iterating through each element of the karnaugh map, I chose to implement functions that deal with specific cases. For example, cases where every element is a 1, it will not go through the trouble of iterating through every element if all the element is a 1. Also cases where groups are created in the corners.

The implementation of the algorithm takes a greedy approach, aiming to optimize computation time and efficiency. By using this strategy, the algorithm prioritizes making locally optimal decisions at each step to achieve overall efficiency gains.

The groups are stored in a $vector < vector < pair < int, int >>>$ which translating into words would be a group(vector) of points(vector) with a pair(pair) of coordinates(int).

```
string Karnaugh::Simplify(vector<vector<int>> matrix) {
    string result;
    vector<vector<pair<int,int>>> groups; // Updated to
       vector<vector<int>>
    vector<pair<int,int>> new_group;

    // Check if any new pairs are present
    if (checkAll(matrix)) {
        result = "The whole matrix is composed of 1's so the
           resulting output is 1";
        return result;
    }
    cout<< "After checkALL"<<endl;
    // Check for specific patterns and conditions
    if (checkCorners1(matrix)) {
        cout << "Group found out of the 4 corners of the
           matrix"<<endl;
        new_group.emplace_back(0, 0);
        new_group.emplace_back(0, matrix[0].size());
        new_group.emplace_back(matrix.size(), 0);
        new_group.emplace_back(matrix.size(), matrix[0].size
           ());
        if(checkVector(new_group,groups))
            groups.push_back(new_group);
    }
    cout<< "After 4 corners"<<endl;
    if (checkCorners2(matrix)) {
        new_group.clear();
        cout << "Group found in top left corner and bottom
           right corner"<< endl;
        new_group.emplace_back(0, 0);
        new_group.emplace_back(matrix.size(), matrix[0].size
           ());
        if(checkVector(new_group,groups))
            groups.push_back(new_group);

    (...)
    }
```

Listing 3.2: Karnaugh Class Code Snippet

3.3 Tabulation

This was the last algorithm I implemented and there is not much I can say about it. In my point of view, little can be changed in the process of implementing this algorithm. What I mean is that, unlike the previous two algorithms, where you can let your imagination flow and implement them in plenty of creative or unique ways, this algorithm seems to be more consistent, more strict.

I started by storing the solutions of a binary table in a vector of structures (structs) just like in the following listing.

```
struct Term {
    string value;
    vector<int> minterms;

    Term(string val, vector<int> mints) : value(val),
       minterms(mints) {}
};
```

Listing 3.3: Term Struct

Then I started iterating through the vector and comparing pairs of terms. I would check if their ”value” (their binary representation) has a difference of 1 digit. If they did, I would turn that pair of terms into a single one and mix the minterms (their decimals) into a single vector and substitute the different digit by a ”-”.

This comparing process would be repeated a total of 4 times. The 4 was chosen because through manual testing I observed that when applying the Quine-McCluskey algorithm to a group of solutions, it will always reach the optimal solution in its 4th loop, if there is no optimal solution, the process will be complete in 3 or less loops.

After the four loops, I will check each Term to see which of them has at least 1 minterm that is unique among all other terms. If a term has an unique minterm, it is added to the final group of terms.

```
    string Tabulation::simplify(vector<vector<int>>
       Binary_Table){
    vector<Tabulation::Term> terms;
    vector<Tabulation::Term> new_terms;
    vector<Tabulation::Term> final_terms;
    Tabulation A;
    terms = A.createTerms(Binary_Table);
    if(terms.empty())
        return "No solutions in Binary Table";

    for(int k=0;k<4;k++) {
        for (int i = 0; i < terms.size(); i++) {
            for (int j = 0; j < terms.size(); j++)
                if (differenceCheck(terms[i].value, terms[j].
                   value) && i != j) {
                    new_terms.push_back(A.combineTerms(terms[
                       i], terms[j]));
                    cout<< "Terms combined" << endl;
                }
        }
        terms=new_terms;
    }
    for(int i=0;i<terms.size();i++){
        if(uniqueTerm(terms[i],terms))
            final_terms.push_back(terms[i]);
    }

    return "Done";
}
```

Listing 3.4: Tabulation Class Code Snippet

3.4 Usage of the application

As the application is not easy to use I will explain how you can use it for each method. In the "main.cpp" file you will have a choice variable, with it you can change which method you wish to apply. Simply equal it to 1 to apply the Boolean algorithm, equal it to 2 for the Karnaugh algorithm and equal it to 3 for the Tabulation algorithm. Then you will have 3 if clauses where you will input your expression/table.

Boolean

If you select the Boolean algorithm, in its corresponding if clause, you will have an expression variable where you can write the expression you want the simplify. After writing it, simply execute the code and it will show you the results in the terminal

Karnaugh

If you choose Karnaugh, it will be just like in the Boolean clause, but instead of an expression, you have a matrix. Simply change the values from 0 to 1 of the matrix. If you want to change the number or variables, in the file "Karnaugh.h" you have other matrices for up to 6 variables that you can copy.

Tabulation

In the case of Tabulation, the method is the same as before, but instead of an expression or a matrix, you have a table. In the table, simply change the value of the F column from "0" to "1" if you want to use that row as a solution. You can only change the F column or the code may break.

Requirements

- Software: The app software itself and Windows Operative System
- Hardware: A basic desktop computer (including screen,keyboard,mouse) or laptop
- Connectivity: No interned connection required

3.5 Other Data

App Architecture

- Classes: Boolean (will calculate using Boolean algebra; Karnaugh (will calculate using Karnaugh Map algorithm); Tabulation (will calculate using Quine McCluskey algorithm)

App User Interface

This is how I envisioned the app would look like after everything was done but due to a lack of time and a lack of overall knowledge in the User Interface department, I opted for a console user interface due to lack of time and knowledge.

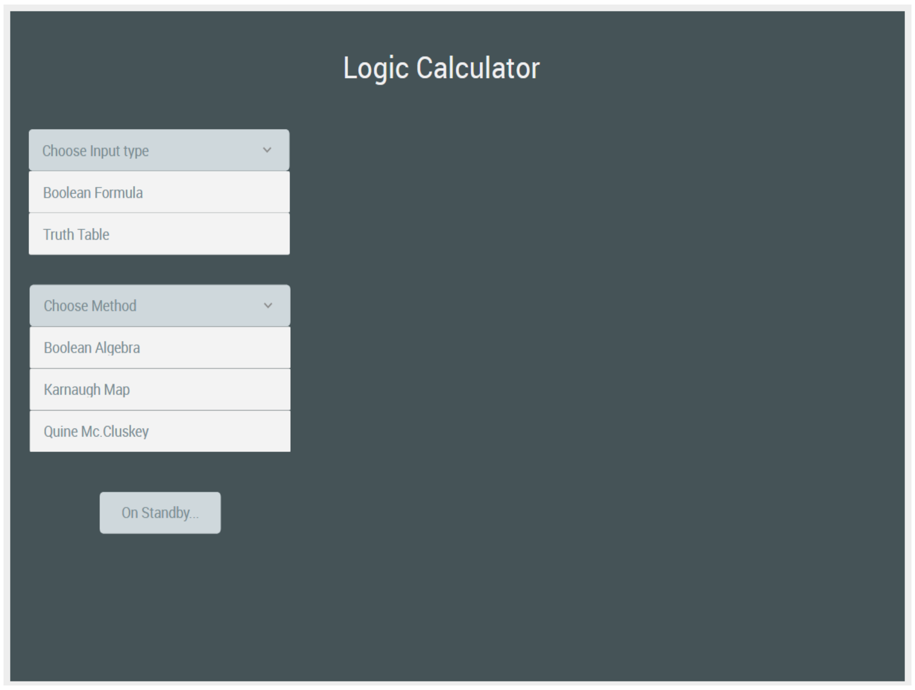
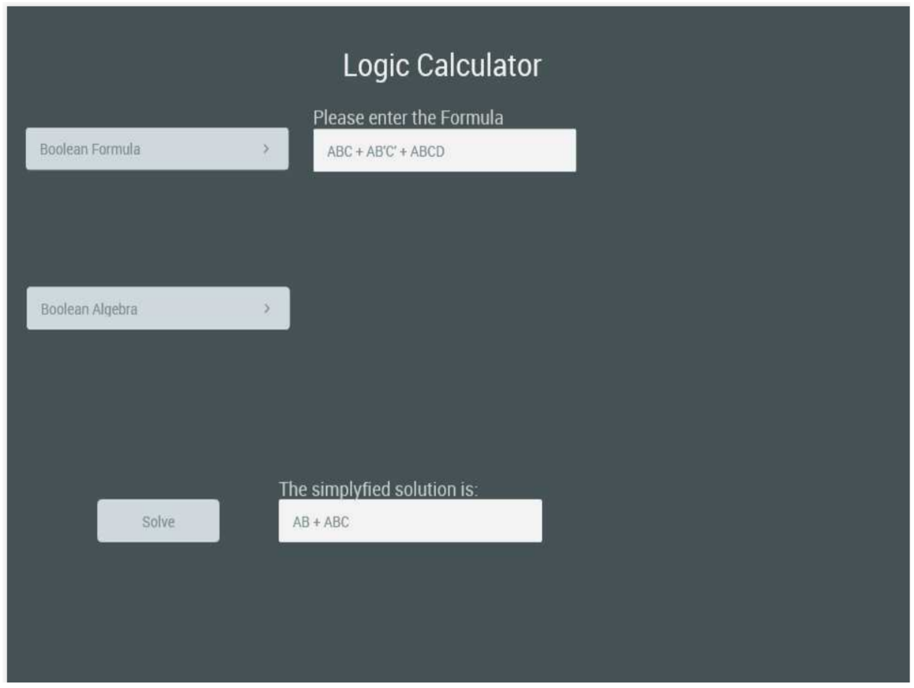


Figure 3.2: Interface page 1

For the app to solve a certain formula, the user must first select the input type and method. As it can be seen in Figure 3.2

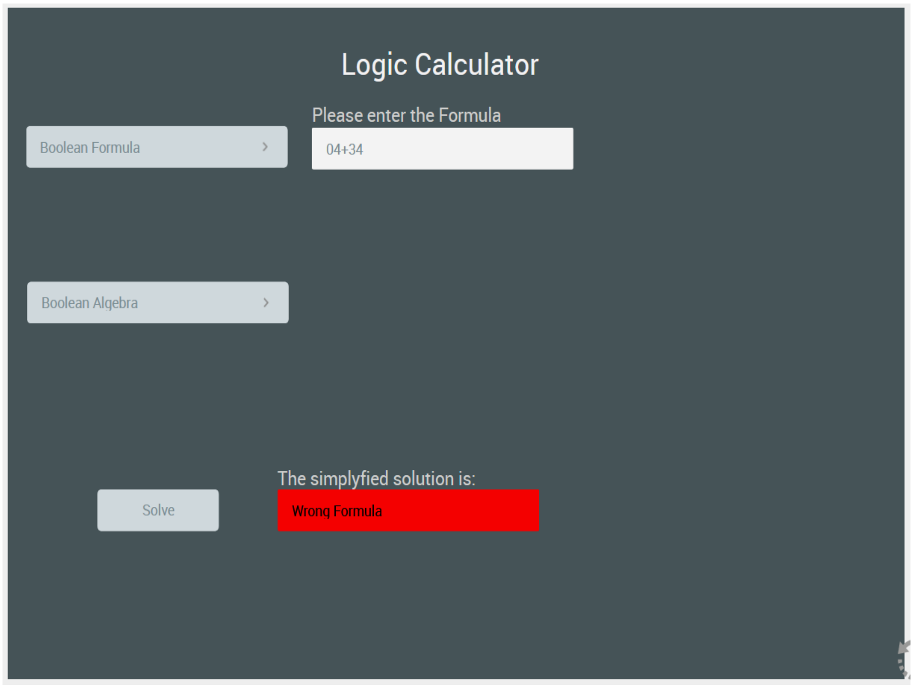


Please enter the Formula
ABC + AB'C' + ABCD

The simplyfied solution is:
AB + ABC

Figure 3.3: Interface page 2

In the case of Figure 3.3, the user has selected the Boolean Formula type of input and has selected the Boolean Algebra method. Then he proceeded to introduce an expression that was then simplified by the app.

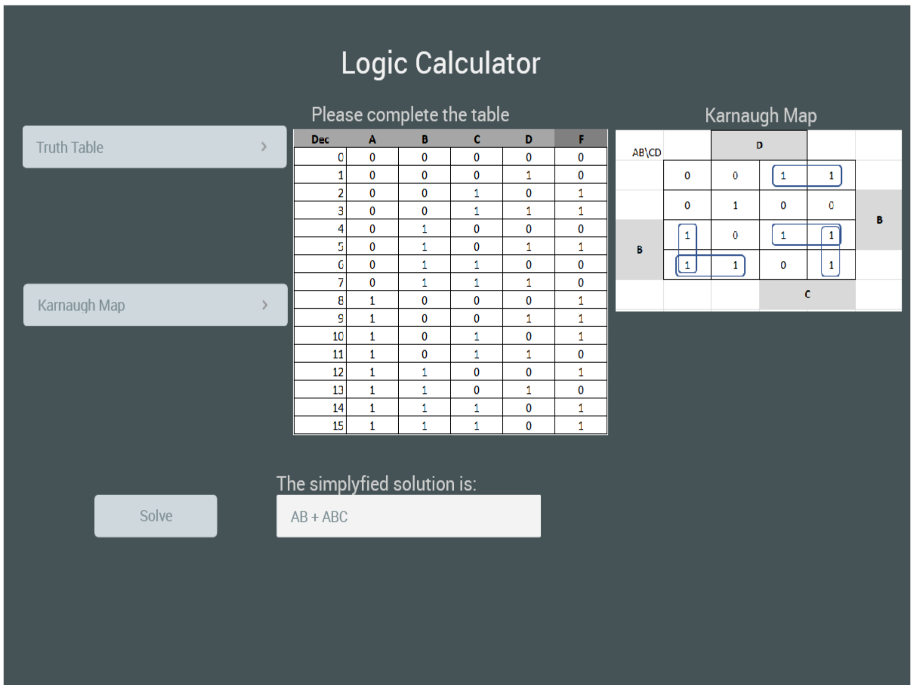


Please enter the Formula
04+34

The simplyfied solution is:
Wrong Formula

Figure 3.4: Interface page 3

In Figure 3.4, it can be seen what happens when the input introduced is of the wrong type.



Please complete the table

Dec	A	B	C	D	F
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	1
9	1	0	0	1	1
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	1
13	1	1	0	1	0
14	1	1	1	0	1
15	1	1	1	0	1

Karnaugh Map

AB\CD		D			
	0	0	1	1	
	0	1	0	0	B
B	1	0	1	1	
	1	1	0	1	
			C		

The simplyfied solution is:

AB + ABC

Figure 3.5: Interface page 4

In Figure 3.5, it can be seen the case of a user trying to simplify a binary table by using a Karnaugh map.

Other Diagrams

Theses are diagrams that were done at the start of development.

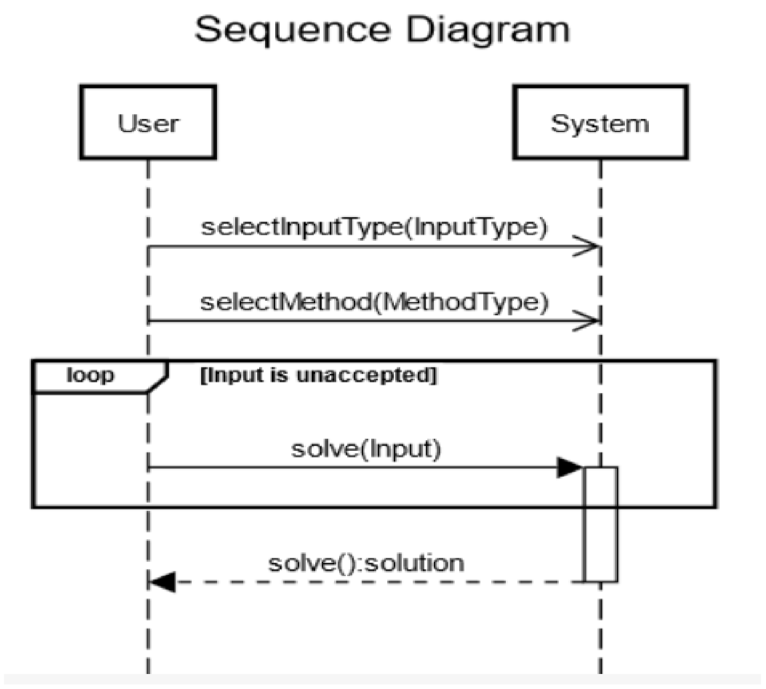


Figure 3.6: Sequence Diagram

In Figure 3.6, you can see the Sequence Diagram for the app, and in Figure 3.7, the use case for trying to simplify a certain input.

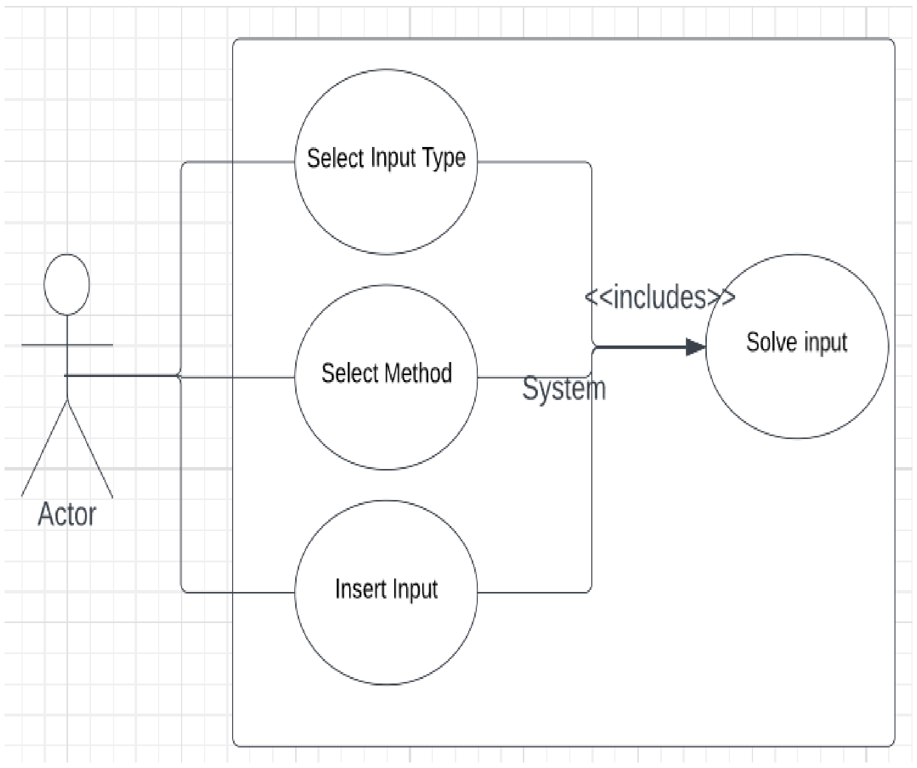


Figure 3.7: Use case

Chapter 4

Related Work

This chapter presents a comprehensive review of the various papers and applications that I studied in order to expand my knowledge on Logical Circuits. By exploring these resources, I aimed to gain a deeper understanding of the fundamental concepts and principles underlying logical circuit design. In this section, I will provide an overview of the different books and apps considered, highlighting their respective strengths and weaknesses.

4.1 Papers

The papers studied were the following:

"Digital Logic and Computer Design" by Morris Mano [6]. In chapter 2, he presents Boolean Algebra. He explains each and every rule that can be applied to Boolean expressions. He also explains the two possible types of notation of the solutions, those being sum of products and product of sums. Although he doesn't use common notation, he uses a more high-level notation but it is easily understandable for anyone with some understanding of Boolean Algebra. In chapter 3, he presents Karnaugh maps method and explains them quite well. He presents the 2 methods of solving a karnaugh map (by pairing 1's or by pairing 0's). He solves karnaugh maps up to 6 variable ones. In chapter 3-9, he presents the Quine-McCluskey method. He explains the Quine-McCluskey method in a simple and easy to understand manner. Overall, Morris Mano explains every method quite well and in a simple way that most people would be able to understand, the only problem being that he does not use common notation in his work.

Digital Logic and Microprocessor Design With VHDL by Enoch O.Hwang [7]. In chapter 3.4 he presents the Karnaugh Maps method and the Quine-McCluskey method. His method of explaining the methods is very similar to Morris Mano's in "Digital Logic and Computer Design" but he doesn't mention the two methods one can interpret the Karnaugh Maps, His way of explaining the Quine-McCluskey method is a bit harder to understand because his graphical representation is not very detailed. Overall, Enoch O.Hwang's work, although very similar to Morris Mano's, it is quite harder to understand if you have no prior knowledge.

4.2 Apps

Boolean Expression Calculator - This app solves any Boolean expression accepting multiple types of notation (usage of different symbols to represent the same operator) and represents and also allowing to choose the notation of the solution itself. It also allows to simplify using only NOR gates or NAND gates which is a more advanced type of simplification.It also explains the methods of simplification. The disadvantage it has would be that it is a quite slow to introduce the input (in comparison to other apps). It does not show the steps that it takes to simplify. Also, the interface representation is quite basic and it does not seem to be very user friendly. It also only solves Boolean expressions via Boolean algebra while the app I plan to develop will have two other methods. See [3]

Boolean Algebra Calculator - This app solves any Boolean expression as well but it does not allow multiple types of notation but the solution's notation is a bit better in comparison to "Boolean Expression Calculator" although it does not have alternative solution notations. The interface is also quite simple and user friendly. Also it shows every step in computing the solution. As in the previous app, it only solves Boolean expressions. See [2]

Boolean Algebra - This app can solve Boolean expressions and Karnaugh Maps while also showing each step to achieve the final solution. It accepts any size for the Karnaugh map while most accept only below 5 variables. It also show additional information such as a Truth Table, Logic Gates and Karnaugh Map. The method of introducing the inputs is also quite simplified. The interface is very attractive, simple and user friendly. It is quite a remarkable app. See [1]

Symbolab Boolean Algebra Calculator - Very similar to Boolean Algebra Calculator, just that it requires an paid account in order to access the steps and the interface is quite worse. With an account, it tells each rule that was applied in order to obtain the final solution. All tests resulted in a minimal solution, so the effectiveness is quite high. See [10]

Calculators.tech Boolean Algebra Calculator - Almost the same as Boolean Algebra Calculator, the only difference being the interface itself, but both are really similar. Also it accepts one different type of input notation, and contains a few examples of Boolean expressions the user could test. See [4]

Four variable Karnaugh Map Solver - This app can solve Karnaugh maps up until a size of 4 variables. It has a very simple and attractive interface. It allows the user to quickly insert the input. It also shows every step in the process of solving the Karnaugh map. The only disadvantage is that it only allows karnaugh maps with less or equal to 4 variables. See [5]

Quine McCluskey Solver - Solves the logic circuit using the Quince McCluskey algorithm. The way of introducing input is quite simplified. It also shows every step. While it has a simple interface it is not recommended for any user since it can be somewhat confusing for some users. Also it only allows 4 variables, neither more neither less. See [8]

Quine–McCluskey Algorithm - Quite similar to the previous "Quine McCluskey Solver" but it has a better interface allowing the user to quickly fill up the input table. It also allows to change the amount of variables up to 8, which is quite surprising and very versatile. A big disadvantage compared to "Quine McCluskey Solver" is that it does not use common notation (instead of using the variables

A,B,C,D,etc... is uses x1,x2,x3,x4,etc) See [9]

Chapter 5

Experimentation

In this chapter, We will explore the practical application of three algorithms by selecting three random examples and manually implementing each algorithm We will also use the related apps discussed in Chapter 4. By employing this approach, We aim to demonstrate the effectiveness and versatility of these algorithms.

5.1 Example 1

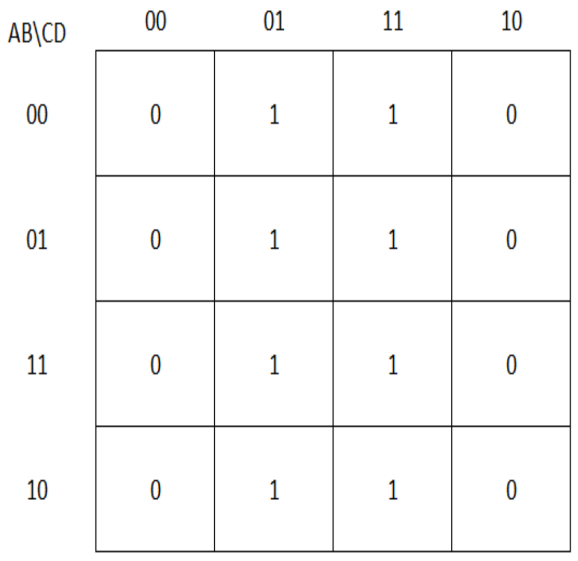
In this example the Boolean Expression is:

$$ABCD + A\overline{B}CD + \overline{A}BCD + \overline{A}\,\overline{B}CD + AB\overline{C}D + A\overline{B}\,\overline{C}D + \overline{A}B\overline{C}D + \overline{A}\,\overline{B}\,\overline{C}D$$

We will start by using it to fill a Binary Table

Dec	A	B	C	D	F
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	0
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	1
10	1	0	1	0	0
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	0
15	1	1	1	1	1

We will also fill the Karnaugh map



AB\CD	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	0	1	1	0
10	0	1	1	0

Figure 5.1: Karnaugh Map Example 1

Let us start with Boolean Algebra

$$ABCD + A\overline{B}CD + \overline{A}BCD + \overline{A}\,\overline{B}CD + AB\overline{C}D + A\overline{B}\,\overline{C}D + \overline{A}B\overline{C}D + \overline{A}\,\overline{B}\,\overline{C}D$$

Applying the Distributive Law

$$ACD(B+\overline{B}) + \overline{A}CD(B+\overline{B}) + A\overline{C}D(B+\overline{B}) + \overline{A}\,\overline{C}D(B+\overline{B})$$

Applying the Inverse Law

$$ACD1 + \overline{A}CD1 + A\overline{C}D1 + \overline{A}\,\overline{C}D1$$

Applying the Identity Law

$$ACD + \overline{A}CD + A\overline{C}D + \overline{A}\,\overline{C}D$$

Applying the Distributive Law

$$CD(A+\overline{A}) + \overline{C}D(\overline{A}+A)$$

Applying the Inverse Law

$$CD1 + \overline{C}D1$$

Applying the Identity Law

$$CD + \overline{C}D$$

Applying the Distributive Law

$$D(\overline{C}+C)$$

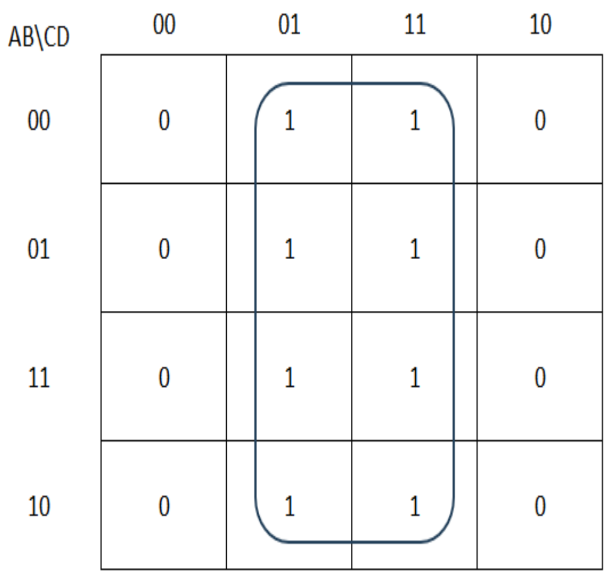
Applying the Inverse Law

$$D1$$

Applying the Identity Law

$$D$$

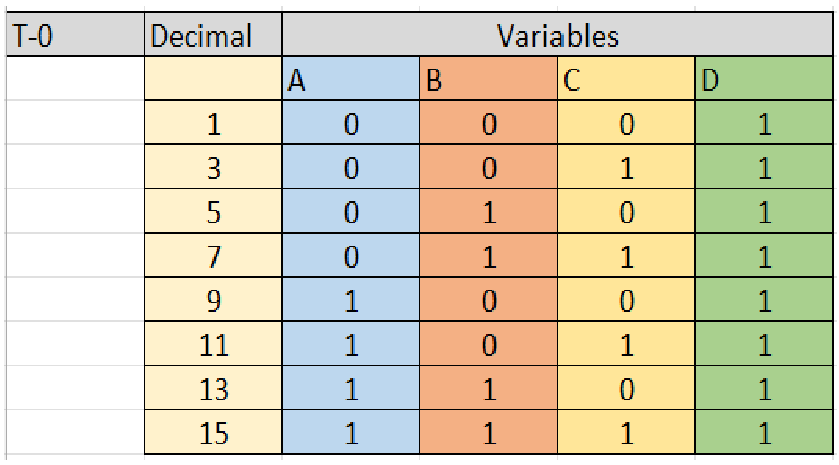
The result obtained from applying Boolean Algebra is D
Now lets proceed with the Karnaugh Map



AB\CD	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	0	1	1	0
10	0	1	1	0

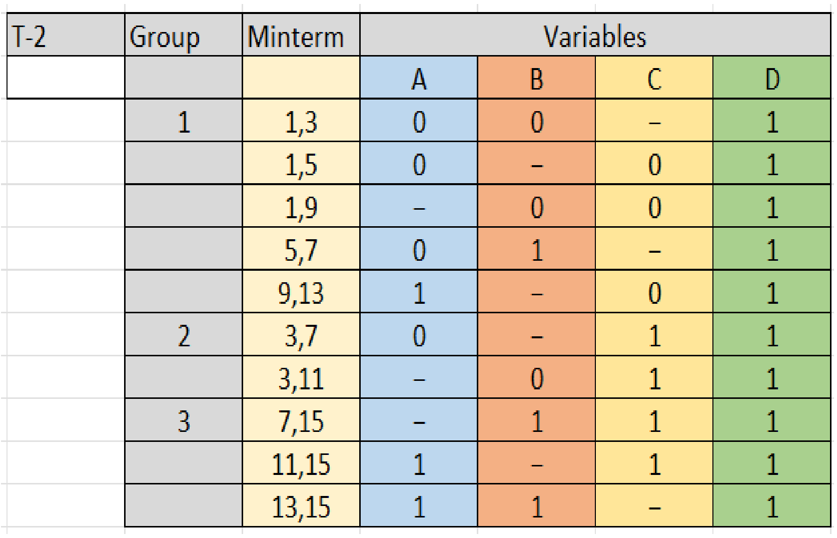
Figure 5.2: Karnaugh Map 2 Example 1

The result obtained from applying the Karnaugh Method is also D
Now unto the Quine-McCluskey Method



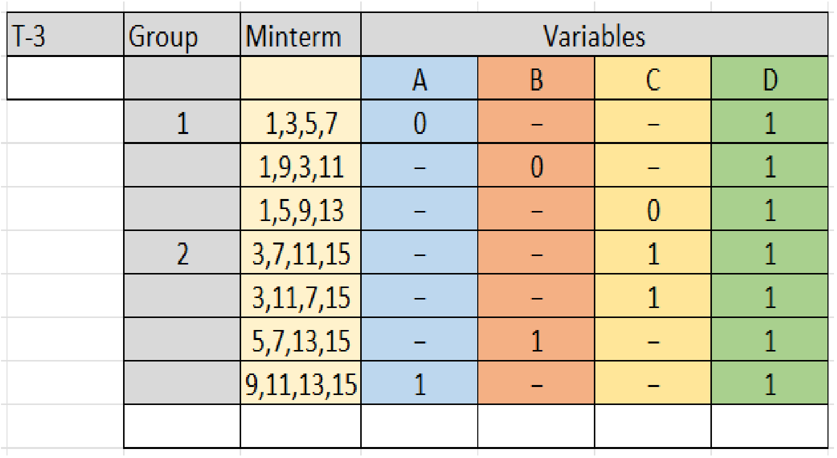
T-0	Decimal	Variables			
		A	B	C	D
	1	0	0	0	1
	3	0	0	1	1
	5	0	1	0	1
	7	0	1	1	1
	9	1	0	0	1
	11	1	0	1	1
	13	1	1	0	1
	15	1	1	1	1

Figure 5.3: Quine McClusky Example 1



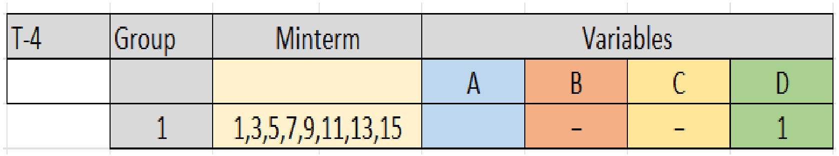
T-2	Group	Minterm	Variables			
			A	B	C	D
	1	1,3	0	0	–	1
		1,5	0	–	0	1
		1,9	–	0	0	1
		5,7	0	1	–	1
		9,13	1	–	0	1
	2	3,7	0	–	1	1
		3,11	–	0	1	1
	3	7,15	–	1	1	1
		11,15	1	–	1	1
		13,15	1	1	–	1

Figure 5.4: Quine McClusky 2 Example 1



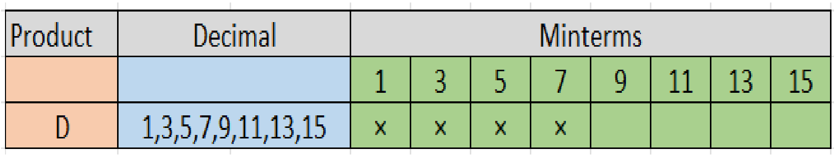
T-3	Group	Minterm	Variables			
			A	B	C	D
	1	1,3,5,7	0	–	–	1
		1,9,3,11	–	0	–	1
		1,5,9,13	–	–	0	1
	2	3,7,11,15	–	–	1	1
		3,11,7,15	–	–	1	1
		5,7,13,15	–	1	–	1
		9,11,13,15	1	–	–	1

Figure 5.5: Quine McClusky 3 Example 1



T-4	Group	Minterm	Variables			
			A	B	C	D
	1	1,3,5,7,9,11,13,15		–	–	1

Figure 5.6: Quine McClusky 4 Example 1



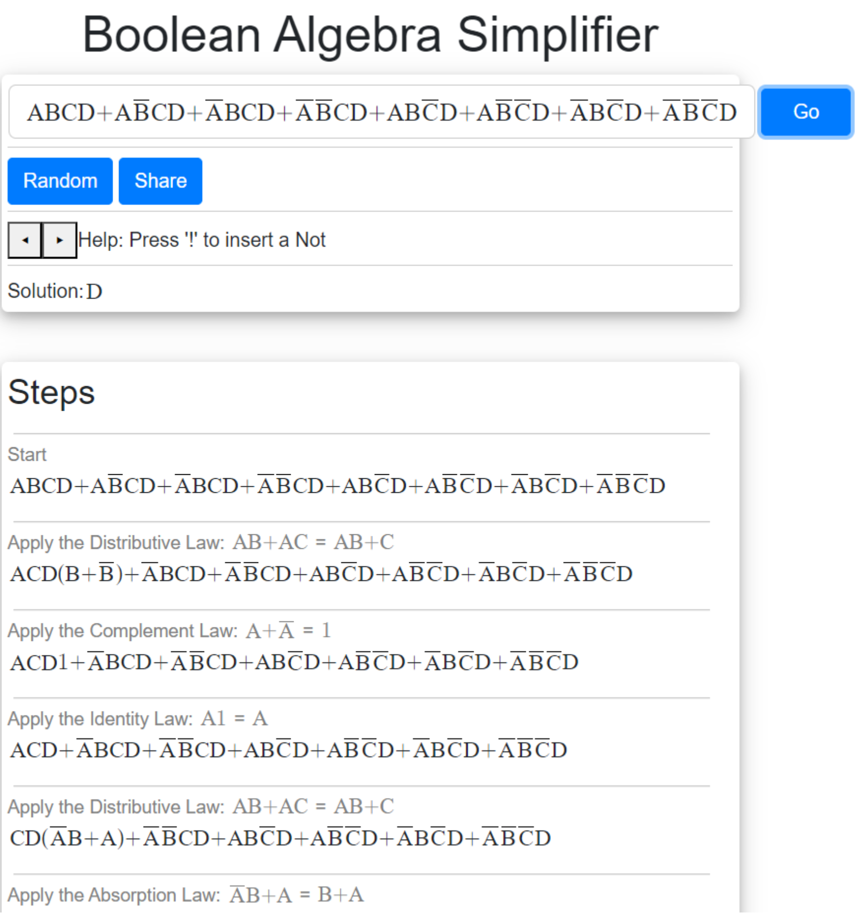
Product	Decimal	Minterms							
		1	3	5	7	9	11	13	15
D	1,3,5,7,9,11,13,15	x	x	x	x				

Figure 5.7: Quine McClusky 5 Example 1

The Quine-McCluskey also resulted into D For this first example, all three algorithms ended up with the same results.

Now We will test some related apps to see if their results differ.

For starters We will use the Boolean Algebra app[1] You can see in the following **Figure5.8**



Boolean Algebra Simplifier

$ABCD+A\overline{B}CD+\overline{A}BCD+\overline{A}\,\overline{B}CD+AB\overline{C}D+A\overline{B}\,\overline{C}D+\overline{A}B\overline{C}D+\overline{A}\,\overline{B}\,\overline{C}D$

Help: Press '!' to insert a Not

Solution: D

Steps

Start

$ABCD+A\overline{B}CD+\overline{A}BCD+\overline{A}\,\overline{B}CD+AB\overline{C}D+A\overline{B}\,\overline{C}D+\overline{A}B\overline{C}D+\overline{A}\,\overline{B}\,\overline{C}D$

Apply the Distributive Law: $AB+AC = AB+C$

$ACD(B+\overline{B})+\overline{A}BCD+\overline{A}\,\overline{B}CD+AB\overline{C}D+A\overline{B}\,\overline{C}D+\overline{A}B\overline{C}D+\overline{A}\,\overline{B}\,\overline{C}D$

Apply the Complement Law: $A+\overline{A} = 1$

$ACD1+\overline{A}BCD+\overline{A}\,\overline{B}CD+AB\overline{C}D+A\overline{B}\,\overline{C}D+\overline{A}B\overline{C}D+\overline{A}\,\overline{B}\,\overline{C}D$

Apply the Identity Law: $A1 = A$

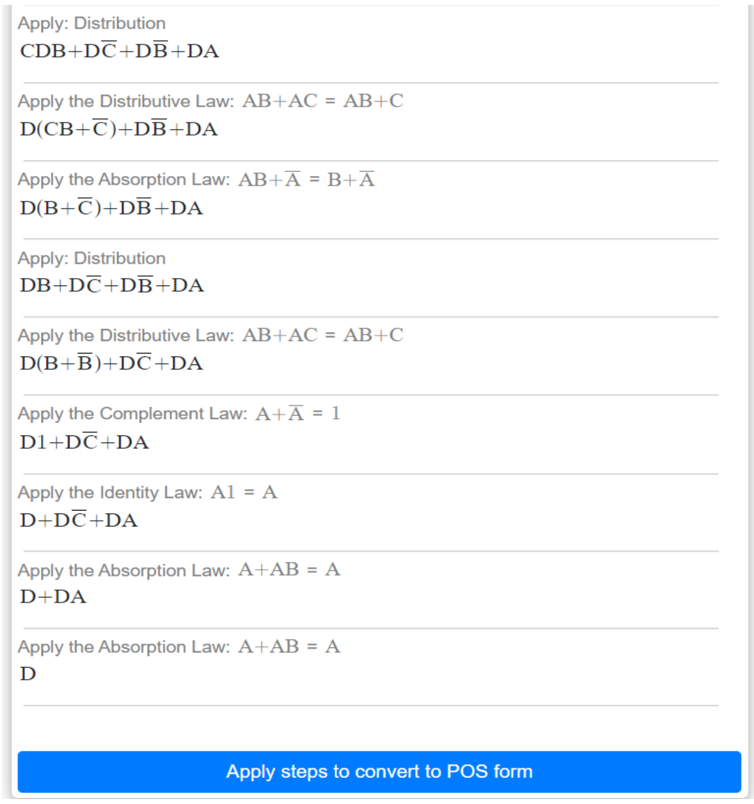
$ACD+\overline{A}BCD+\overline{A}\,\overline{B}CD+AB\overline{C}D+A\overline{B}\,\overline{C}D+\overline{A}B\overline{C}D+\overline{A}\,\overline{B}\,\overline{C}D$

Apply the Distributive Law: $AB+AC = AB+C$

$CD(\overline{A}B+A)+\overline{A}\,\overline{B}CD+AB\overline{C}D+A\overline{B}\,\overline{C}D+\overline{A}B\overline{C}D+\overline{A}\,\overline{B}\,\overline{C}D$

Apply the Absorption Law: $\overline{A}B+A = B+A$

Figure 5.8: Boolean Algebra App Example 1


Apply: Distribution
$CDB+D\overline{C}+D\overline{B}+DA$

Apply the Distributive Law: $AB+AC = AB+C$
$D(CB+\overline{C})+D\overline{B}+DA$

Apply the Absorption Law: $AB+\overline{A} = B+\overline{A}$
$D(B+\overline{C})+D\overline{B}+DA$

Apply: Distribution
$DB+D\overline{C}+D\overline{B}+DA$

Apply the Distributive Law: $AB+AC = AB+C$
$D(B+\overline{B})+D\overline{C}+DA$

Apply the Complement Law: $A+\overline{A} = 1$
$D1+D\overline{C}+DA$

Apply the Identity Law: $A1 = A$
$D+D\overline{C}+DA$

Apply the Absorption Law: $A+AB = A$
$D+DA$

Apply the Absorption Law: $A+AB = A$
D

Figure 5.9: Boolean Algebra App 2 Example 1

I did omit some of the laws applied because the app applies many rules redundantly. The app applied around 10 to 15 more rules than We did manually which is a bit of a waste of computation time.

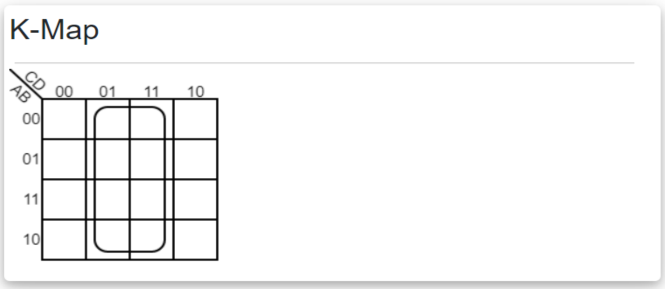
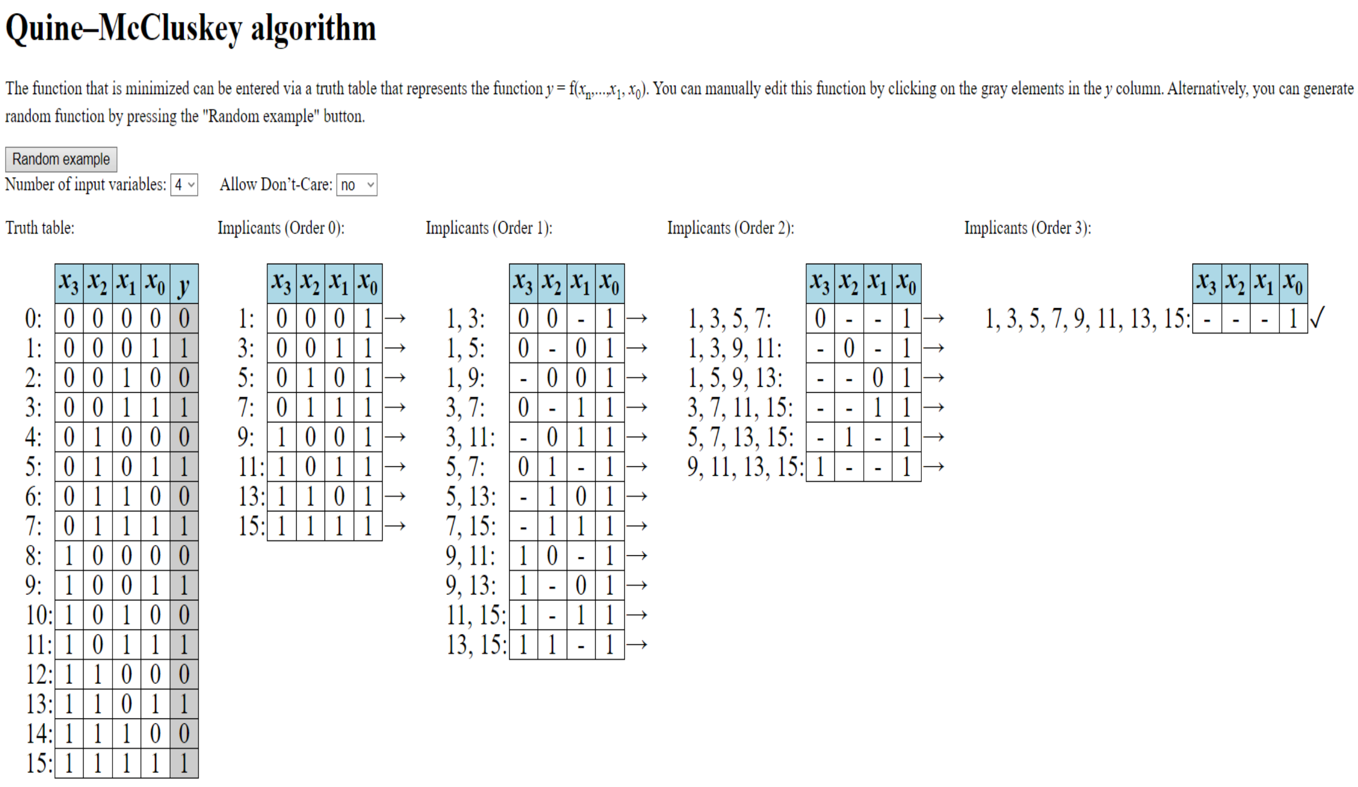


Figure 5.10: Boolean Algebra App/Karnaugh Example 1

The app also fills up karnaugh map. As you can see it is identical to the one shown in **Figure5.2**.

For the Quine-McCluskey Method, We chose to use the Quine-McCluskey algorithm app [8]



Quine–McCluskey algorithm

The function that is minimized can be entered via a truth table that represents the function $y = f(x_n,...,x_1, x_0)$. You can manually edit this function by clicking on the gray elements in the y column. Alternatively, you can generate random function by pressing the "Random example" button.

Number of input variables: 4 Allow Don't-Care: no

Truth table:

	x_3	x_2	x_1	x_0	y
0:	0	0	0	0	0
1:	0	0	0	1	1
2:	0	0	1	0	0
3:	0	0	1	1	1
4:	0	1	0	0	0
5:	0	1	0	1	1
6:	0	1	1	0	0
7:	0	1	1	1	1
8:	1	0	0	0	0
9:	1	0	0	1	1
10:	1	0	1	0	0
11:	1	0	1	1	1
12:	1	1	0	0	0
13:	1	1	0	1	1
14:	1	1	1	0	0
15:	1	1	1	1	1

Implicants (Order 0):

	x_3	x_2	x_1	x_0	
1:	0	0	0	1	→
3:	0	0	1	1	→
5:	0	1	0	1	→
7:	0	1	1	1	→
9:	1	0	0	1	→
11:	1	0	1	1	→
13:	1	1	0	1	→
15:	1	1	1	1	→

Implicants (Order 1):

	x_3	x_2	x_1	x_0	
1, 3:	0	0	-	1	→
1, 5:	0	-	0	1	→
1, 9:	-	0	0	1	→
3, 7:	0	-	1	1	→
3, 11:	-	0	1	1	→
5, 7:	0	1	-	1	→
5, 13:	-	1	0	1	→
7, 15:	-	1	1	1	→
9, 11:	1	0	-	1	→
9, 13:	1	-	0	1	→
11, 15:	1	-	1	1	→
13, 15:	1	1	-	1	→

Implicants (Order 2):

	x_3	x_2	x_1	x_0	
1, 3, 5, 7:	0	-	-	1	→
1, 3, 9, 11:	-	0	-	1	→
1, 5, 9, 13:	-	-	0	1	→
3, 7, 11, 15:	-	-	1	1	→
5, 7, 13, 15:	-	1	-	1	→
9, 11, 13, 15:	1	-	-	1	→

Implicants (Order 3):

	x_3	x_2	x_1	x_0	
1, 3, 5, 7, 9, 11, 13, 15:	-	-	-	1	✓

Figure 5.11: Quine McClusky Algorithm App Example 1

As it can be seen the result obtained from this app is also D.

All methods applied and related apps resulted in the same result, D, which is an optimal solution, meaning it can't be further simplified.

5.2 Example 2

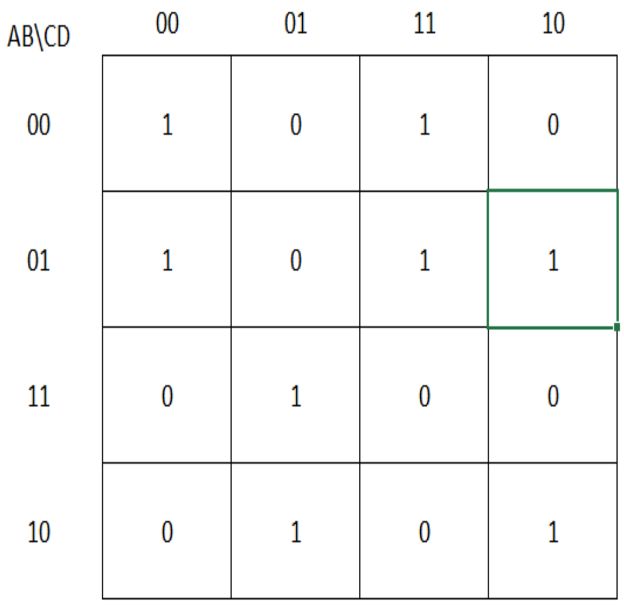
For this example, We will be using the following example:

$$AB\overline{C}D + \overline{A}BCD + \overline{A}\,\overline{B}CD + A\overline{B}\,\overline{C}D + A\overline{B}C\overline{D} + \overline{A}B\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + \overline{A}BC\overline{D}$$

Let us proceed to filling the Binary Table

Dec	A	B	C	D	F
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	1
4	0	1	0	0	1
5	0	1	0	1	0
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	1
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	0
15	1	1	1	1	0

Now, We will fill the Karnaugh map



AB\CD	00	01	11	10
00	1	0	1	0
01	1	0	1	1
11	0	1	0	0
10	0	1	0	1

Figure 5.12: Karnaugh Map Example 2

Proceeding with Boolean Algebra:

$$AB\overline{C}D + \overline{A}BCD + \overline{A}\,\overline{B}CD + A\overline{B}\,\overline{C}D + A\overline{B}C\overline{D} + \overline{A}B\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + \overline{A}BC\overline{D}$$

Applying Distributive Law

$$A\overline{C}D(B + \overline{B}) + \overline{A}CD(B + \overline{B}) + +A\overline{B}C\overline{D} + \overline{A}\,\overline{C}\,\overline{D}(\overline{B} + B) + \overline{A}BC\overline{D}$$

Applying Inverse Law

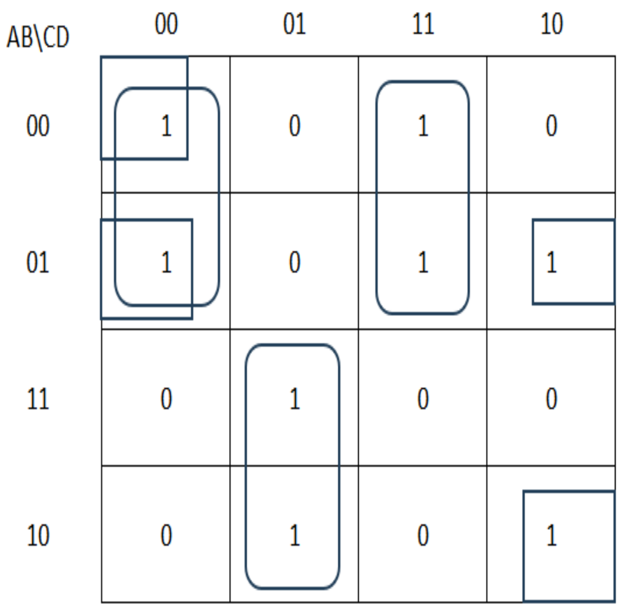
$$A\overline{C}D1 + \overline{A}CD1 + +A\overline{B}C\overline{D} + \overline{A}\,\overline{C}\,\overline{D}1 + \overline{A}BC\overline{D}$$

Applying Identity Law

$$A\overline{C}D + \overline{A}CD + A\overline{B}C\overline{D} + \overline{A}\,\overline{C}\,\overline{D} + \overline{A}BC\overline{D}$$

This is as far as we can apply Boolean rules. The resulting expression is $A\overline{C}D + \overline{A}CD + A\overline{B}C\overline{D} + \overline{A}\,\overline{C}\,\overline{D} + \overline{A}BC\overline{D}$

Moving on to the Karnaugh map method, refer to **Figure5.13**



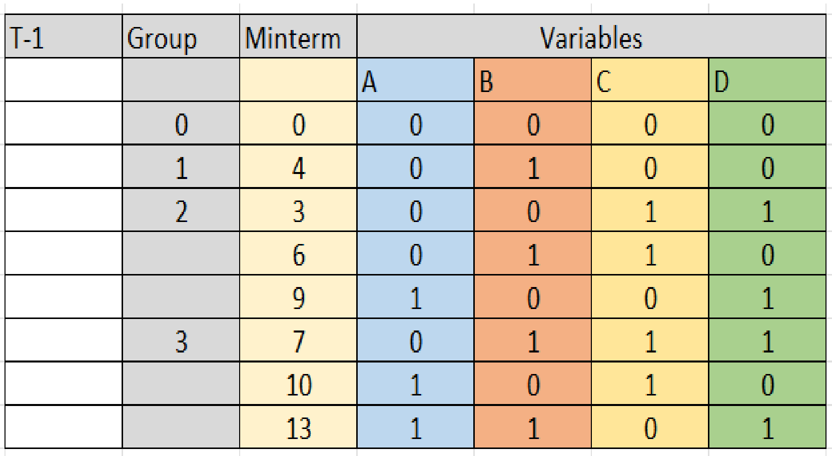
AB\CD	00	01	11	10
00	1	0	1	0
01	1	0	1	1
11	0	1	0	0
10	0	1	0	1

Figure 5.13: Karnaugh Map 2 Example 2

The Karnaugh Map results in the following Boolean Expression:

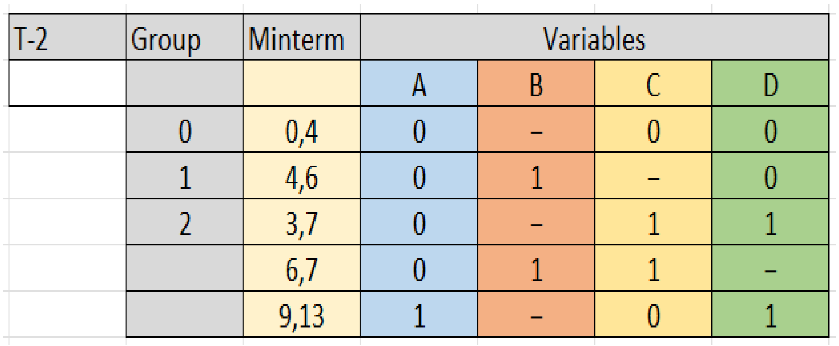
$$\overline{A}\,\overline{C}\,\overline{D} + A\overline{C}D + \overline{A}CD + \overline{A}B\overline{D} + \overline{B}\,\overline{D}$$

Proceeding to the Quine-McCluskey Method:



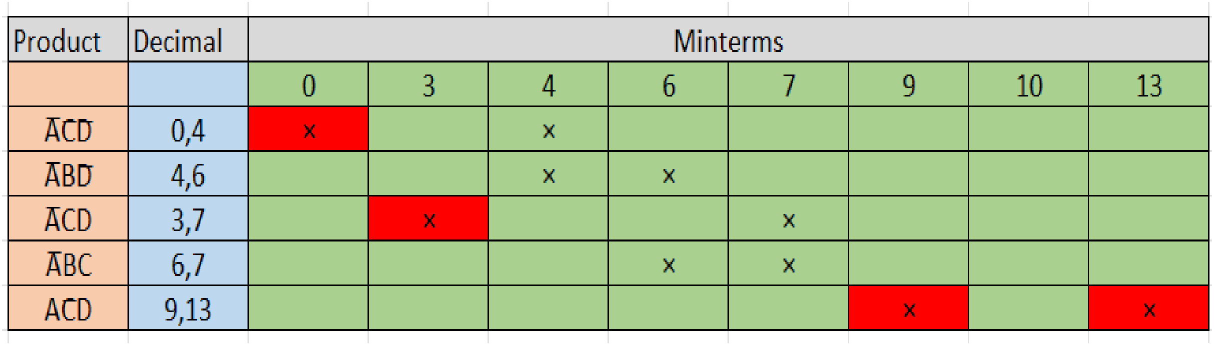
T-1	Group	Minterm	Variables			
			A	B	C	D
	0	0	0	0	0	0
	1	4	0	1	0	0
	2	3	0	0	1	1
		6	0	1	1	0
		9	1	0	0	1
	3	7	0	1	1	1
		10	1	0	1	0
		13	1	1	0	1

Figure 5.14: Quine McClusky Example 2



T-2	Group	Minterm	Variables			
			A	B	C	D
	0	0,4	0	–	0	0
	1	4,6	0	1	–	0
	2	3,7	0	–	1	1
		6,7	0	1	1	–
		9,13	1	–	0	1

Figure 5.15: Quine McClusky 2 Example 2



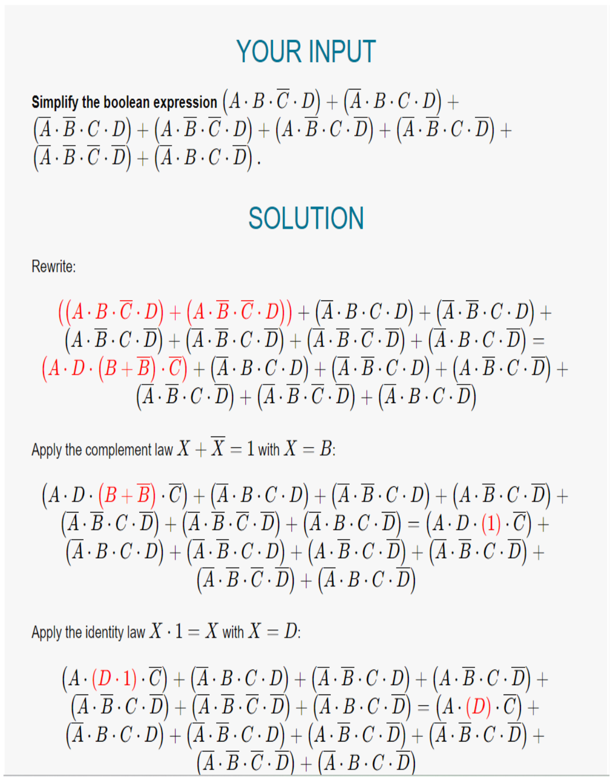
Product	Decimal	Minterms							
		0	3	4	6	7	9	10	13
$\overline{A}\overline{C}\overline{D}$	0,4	x		x					
$\overline{A}B\overline{D}$	4,6			x	x				
$\overline{A}CD$	3,7		x			x			
$\overline{A}BC$	6,7				x	x			
$A\overline{C}D$	9,13						x		x

Figure 5.16: Quine McClusky 3 Example 2

The resulting expression is:

$$\overline{A}\,\overline{C}\,\overline{D} + \overline{A}CD + A\overline{C}D$$

This time We will use the Boolean Algebra Calculator [2], refer to **Figure5.17**



YOUR INPUT

Simplify the boolean expression $\left(A \cdot B \cdot \overline{C} \cdot D\right) + \left(\overline{A} \cdot B \cdot C \cdot D\right) + \left(\overline{A} \cdot \overline{B} \cdot C \cdot D\right) + \left(A \cdot \overline{B} \cdot \overline{C} \cdot D\right) + \left(A \cdot \overline{B} \cdot C \cdot \overline{D}\right) + \left(\overline{A} \cdot \overline{B} \cdot C \cdot \overline{D}\right) + \left(\overline{A} \cdot \overline{B} \cdot \overline{C} \cdot \overline{D}\right) + \left(\overline{A} \cdot B \cdot C \cdot \overline{D}\right)$.

SOLUTION

Rewrite:

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

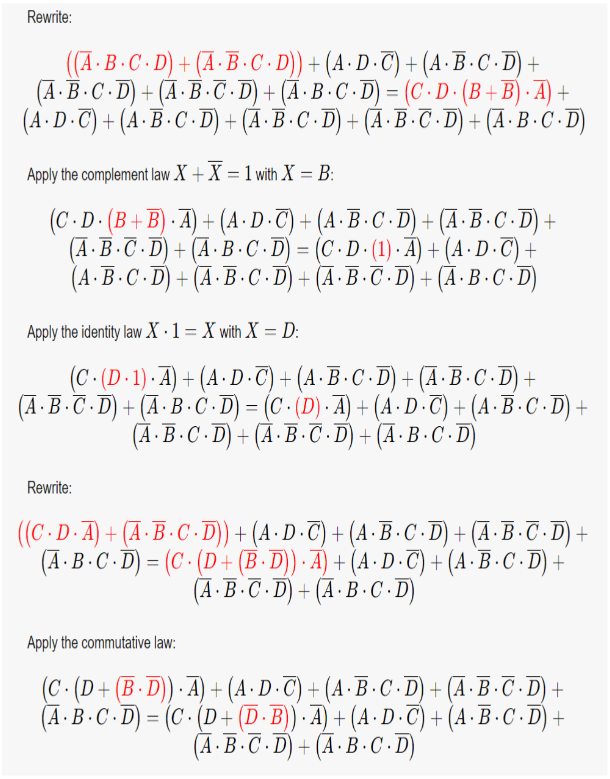
Apply the complement law $X + \overline{X} = 1$ with $X = B$:

$$\left(A \cdot D \cdot \left(B + \overline{B}\right) \cdot \overline{C}\right) + \left(\overline{A} \cdot B \cdot C \cdot D\right) + \left(\overline{A} \cdot \overline{B} \cdot C \cdot D\right) + \left(A \cdot \overline{B} \cdot C \cdot \overline{D}\right) + \left(\overline{A} \cdot \overline{B} \cdot C \cdot \overline{D}\right) + \left(\overline{A} \cdot \overline{B} \cdot \overline{C} \cdot \overline{D}\right) + \left(\overline{A} \cdot B \cdot C \cdot \overline{D}\right) = \left(A \cdot D \cdot (1) \cdot \overline{C}\right) + \left(\overline{A} \cdot B \cdot C \cdot D\right) + \left(\overline{A} \cdot \overline{B} \cdot C \cdot D\right) + \left(A \cdot \overline{B} \cdot C \cdot \overline{D}\right) + \left(\overline{A} \cdot \overline{B} \cdot C \cdot \overline{D}\right) + \left(\overline{A} \cdot \overline{B} \cdot \overline{C} \cdot \overline{D}\right) + \left(\overline{A} \cdot B \cdot C \cdot \overline{D}\right)$$

Apply the identity law $X \cdot 1 = X$ with $X = D$:

$$\left(A \cdot (D \cdot 1) \cdot \overline{C}\right) + \left(\overline{A} \cdot B \cdot C \cdot D\right) + \left(\overline{A} \cdot \overline{B} \cdot C \cdot D\right) + \left(A \cdot \overline{B} \cdot C \cdot \overline{D}\right) + \left(\overline{A} \cdot \overline{B} \cdot C \cdot \overline{D}\right) + \left(\overline{A} \cdot \overline{B} \cdot \overline{C} \cdot \overline{D}\right) + \left(\overline{A} \cdot B \cdot C \cdot \overline{D}\right) = \left(A \cdot (D) \cdot \overline{C}\right) + \left(\overline{A} \cdot B \cdot C \cdot D\right) + \left(\overline{A} \cdot \overline{B} \cdot C \cdot D\right) + \left(A \cdot \overline{B} \cdot C \cdot \overline{D}\right) + \left(\overline{A} \cdot \overline{B} \cdot C \cdot \overline{D}\right) + \left(\overline{A} \cdot \overline{B} \cdot \overline{C} \cdot \overline{D}\right) + \left(\overline{A} \cdot B \cdot C \cdot \overline{D}\right)$$

Figure 5.17: Boolean Algebra Calculator App Example 2



Rewrite:

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

Apply the complement law $X + \overline{X} = 1$ with $X = B$:

$$\left(C \cdot D \cdot \left(B + \overline{B}\right) \cdot \overline{A}\right) + \left(A \cdot D \cdot \overline{C}\right) + \left(A \cdot \overline{B} \cdot C \cdot \overline{D}\right) + \left(\overline{A} \cdot \overline{B} \cdot C \cdot \overline{D}\right) + \left(\overline{A} \cdot \overline{B} \cdot \overline{C} \cdot \overline{D}\right) + \left(\overline{A} \cdot B \cdot C \cdot \overline{D}\right) = \left(C \cdot D \cdot (1) \cdot \overline{A}\right) + \left(A \cdot D \cdot \overline{C}\right) + \left(A \cdot \overline{B} \cdot C \cdot \overline{D}\right) + \left(\overline{A} \cdot \overline{B} \cdot C \cdot \overline{D}\right) + \left(\overline{A} \cdot \overline{B} \cdot \overline{C} \cdot \overline{D}\right) + \left(\overline{A} \cdot B \cdot C \cdot \overline{D}\right)$$

Apply the identity law $X \cdot 1 = X$ with $X = D$:

$$\left(C \cdot (D \cdot 1) \cdot \overline{A}\right) + \left(A \cdot D \cdot \overline{C}\right) + \left(A \cdot \overline{B} \cdot C \cdot \overline{D}\right) + \left(\overline{A} \cdot \overline{B} \cdot C \cdot \overline{D}\right) + \left(\overline{A} \cdot \overline{B} \cdot \overline{C} \cdot \overline{D}\right) + \left(\overline{A} \cdot B \cdot C \cdot \overline{D}\right) = \left(C \cdot (D) \cdot \overline{A}\right) + \left(A \cdot D \cdot \overline{C}\right) + \left(A \cdot \overline{B} \cdot C \cdot \overline{D}\right) + \left(\overline{A} \cdot \overline{B} \cdot C \cdot \overline{D}\right) + \left(\overline{A} \cdot \overline{B} \cdot \overline{C} \cdot \overline{D}\right) + \left(\overline{A} \cdot B \cdot C \cdot \overline{D}\right)$$

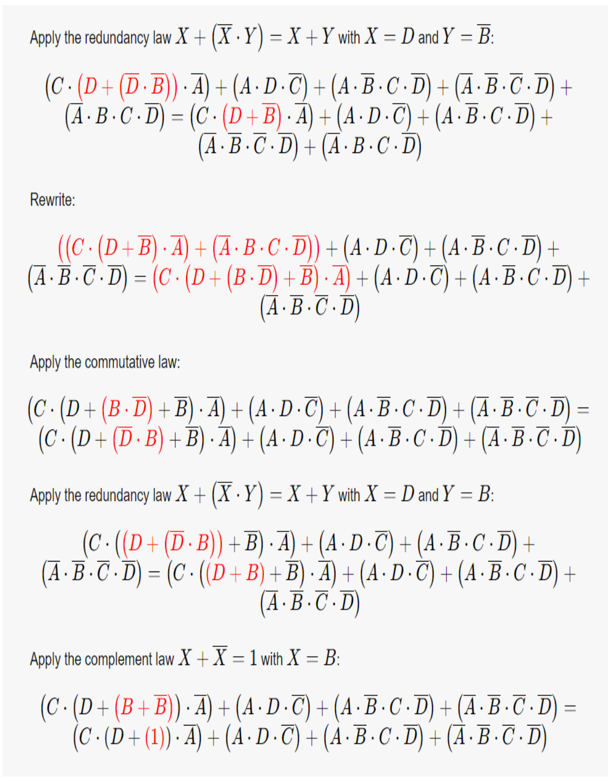
Rewrite:

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

Apply the commutative law:

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

Figure 5.18: Boolean Algebra Calculator App 2 Example 2



Apply the redundancy law $X+\left(\overline{X}\cdot Y\right)=X+Y$ with $X=D$ and $Y=\overline{B}$:

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

Rewrite:

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

Apply the commutative law:

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

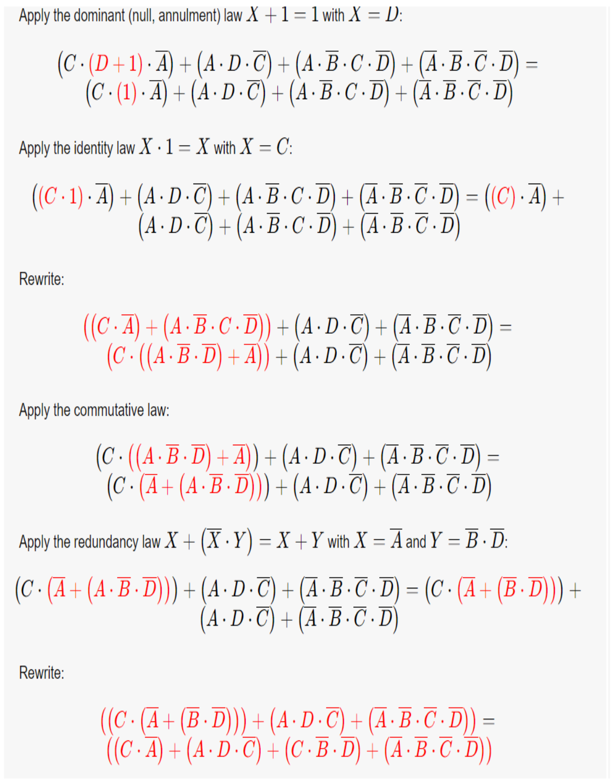
Apply the redundancy law $X+\left(\overline{X}\cdot Y\right)=X+Y$ with $X=D$ and $Y=B$:

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

Apply the complement law $X+\overline{X}=1$ with $X=B$:

$$\left(C\cdot\left(D+\left(B+\overline{B}\right)\right)\cdot\overline{A}\right)+\left(A\cdot D\cdot\overline{C}\right)+\left(A\cdot\overline{B}\cdot C\cdot\overline{D}\right)+\left(\overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}\right)=\left(C\cdot\left(D+\left(1\right)\right)\cdot\overline{A}\right)+\left(A\cdot D\cdot\overline{C}\right)+\left(A\cdot\overline{B}\cdot C\cdot\overline{D}\right)+\left(\overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}\right)$$

Figure 5.19: Boolean Algebra Calculator 3 Example 2



Apply the dominant (null, annulment) law $X+1=1$ with $X=D$:

$$\left(C\cdot\left(D+1\right)\cdot\overline{A}\right)+\left(A\cdot D\cdot\overline{C}\right)+\left(A\cdot\overline{B}\cdot C\cdot\overline{D}\right)+\left(\overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}\right)=\left(C\cdot\left(1\right)\cdot\overline{A}\right)+\left(A\cdot D\cdot\overline{C}\right)+\left(A\cdot\overline{B}\cdot C\cdot\overline{D}\right)+\left(\overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}\right)$$

Apply the identity law $X\cdot 1=X$ with $X=C$:

$$\left(\left(C\cdot 1\right)\cdot\overline{A}\right)+\left(A\cdot D\cdot\overline{C}\right)+\left(A\cdot\overline{B}\cdot C\cdot\overline{D}\right)+\left(\overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}\right)=\left(\left(C\right)\cdot\overline{A}\right)+\left(A\cdot D\cdot\overline{C}\right)+\left(A\cdot\overline{B}\cdot C\cdot\overline{D}\right)+\left(\overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}\right)$$

Rewrite:

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

Apply the commutative law:

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

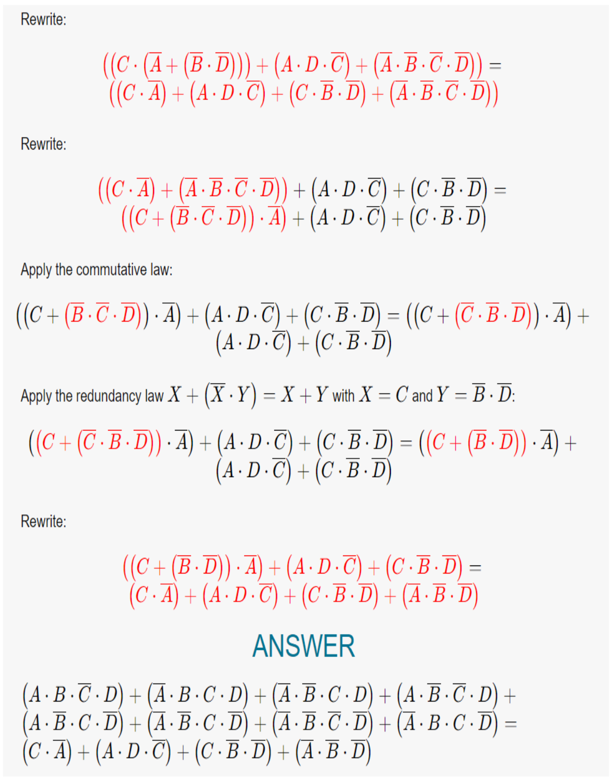
Apply the redundancy law $X+\left(\overline{X}\cdot Y\right)=X+Y$ with $X=\overline{A}$ and $Y=\overline{B}\cdot\overline{D}$:

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

Rewrite:

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

Figure 5.20: Boolean Algebra Calculator App 4 Example 2



Rewrite:

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

Rewrite:

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

Apply the commutative law:

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

Apply the redundancy law $X+\left(\overline{X}\cdot Y\right)=X+Y$ with $X=C$ and $Y=\overline{B}\cdot\overline{D}$:

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

Rewrite:

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

ANSWER

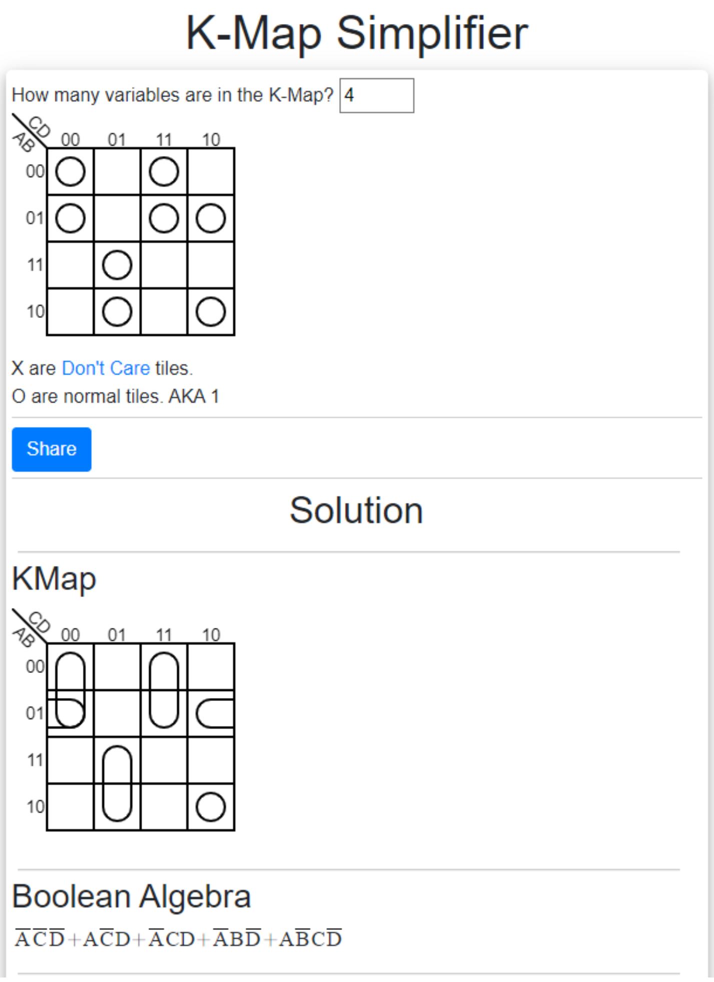
$$\left(A\cdot B\cdot\overline{C}\cdot D\right)+\left(\overline{A}\cdot B\cdot C\cdot D\right)+\left(\overline{A}\cdot\overline{B}\cdot C\cdot D\right)+\left(A\cdot\overline{B}\cdot\overline{C}\cdot D\right)+\left(A\cdot\overline{B}\cdot C\cdot\overline{D}\right)+\left(\overline{A}\cdot\overline{B}\cdot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

Figure 5.21: Boolean Algebra Calculator App 5 Example 2

The app results in the following expression:

$$\overline{A}C+AD\overline{C}+C\overline{B}\,\overline{D}+\overline{A}\,\overline{B}\,\overline{D}$$

For the Karnaugh Method We decided to use the Boolean Algebra app [1]. It contains a section for solving Karnaugh maps.


K-Map Simplifier

How many variables are in the K-Map? 4

X are Don't Care tiles.

O are normal tiles. AKA 1

Solution

KMap

Boolean Algebra

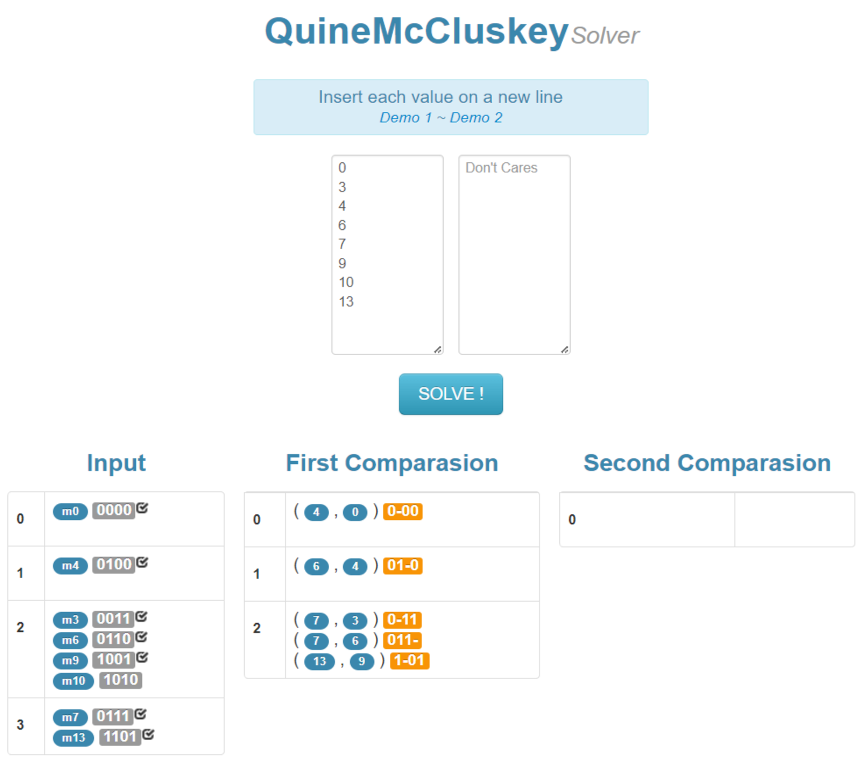
$\overline{A}\overline{C}\overline{D}+A\overline{C}D+\overline{A}CD+\overline{A}B\overline{D}+A\overline{B}C\overline{D}$

Figure 5.22: Boolean Algebra App/Karnaugh Example 2

The resulting solution is:

$$\overline{A}\,\overline{C}\,\overline{D} + A\overline{C}D + \overline{A}CD + \overline{A}B\overline{D} + A\overline{B}C\overline{D}$$

For Quine-McCluskey we will use the Quine McCluskey Solver [8] Refer to **Figure5.23**



```
0
3
4
6
7
9
10
13
```

Input

0	m0 0000 ✓
1	m4 0100 ✓
2	m3 0011 ✓ m6 0110 ✓ m9 1001 ✓ m10 1010
3	m7 0111 ✓ m13 1101 ✓

First Comparasion

0	(4 , 0) 0-00
1	(6 , 4) 01-0
2	(7 , 3) 0-11 (7 , 6) 011- (13 , 9) 1-01

Second Comparasion

0		

Figure 5.23: Quine-McCluskey Solver App Example 2

The result from this app is the following:

$$\overline{A}B\overline{D} + \overline{A}CD + \overline{A}BC + A\overline{C}D + A\overline{B}C\overline{D}$$

Now we will compare the results obtained:

- Manual application of Boolean Algebra: $A\overline{C}D + \overline{A}CD + A\overline{B}C\overline{D} + \overline{A}\,\overline{C}\,\overline{D} + \overline{A}BC\overline{D}$
- Manual application of Karnaugh Map: $\overline{A}\,\overline{C}\,\overline{D} + A\overline{C}D + \overline{A}CD + \overline{A}B\overline{D} + \overline{B}\,\overline{D}$
- Manual application of Quine-McCluskey:$\overline{A}\,\overline{C}\,\overline{D} + \overline{A}CD + A\overline{C}D$
- Boolean Algebra Calculator: $\overline{A}C + AD\overline{C} + C\overline{B}\,\overline{D} + \overline{A}\,\overline{B}\,\overline{D}$
- Boolean Algebra/Karnaugh:$\overline{A}\,\overline{C}\,\overline{D} + A\overline{C}D + \overline{A}CD + \overline{A}B\overline{D} + A\overline{B}C\overline{D}$
- Quine-McCluskey Solver:$\overline{A}B\overline{D} + \overline{A}CD + \overline{A}BC + A\overline{C}D + A\overline{B}C\overline{D}$

Although most results are fairly similar to each other, the Manual application stands out, showing the best results. Most likely due to there not existing redundant functions to be applied and the consistency of the Quine-McCluskey method itself.

If you refer to **Figure5.23**, you can see that the Second Comparison returns a 0 due to there not being a second loop to pair the terms, so the app cannot proceed.

5.3 Example 3

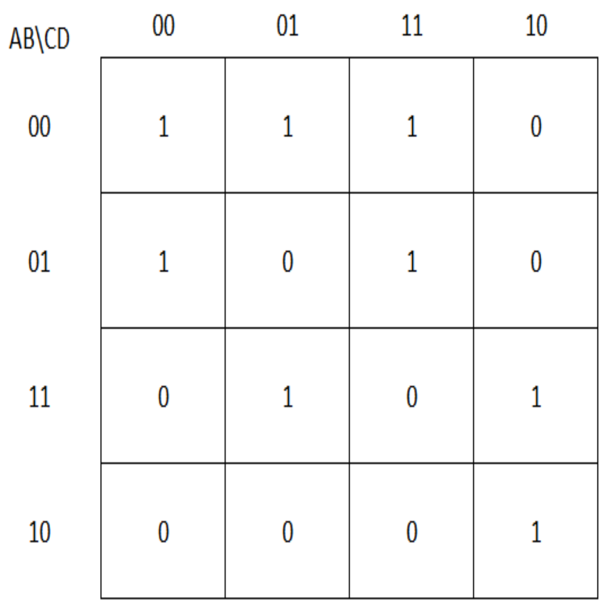
For this example, we will use the following expression:

$$AB\overline{C}D + \overline{A}BCD + \overline{A}\,\overline{B}CD + \overline{A}B\overline{C}\,\overline{D} + A\overline{B}C\overline{D} + \overline{A}\,\overline{B}\,\overline{C}D + ABC\overline{D} + \overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$$

Proceeding with the Binary Table:

Dec	A	B	C	D	F
0	0	0	0	0	1
1	0	0	0	1	1
2	0	0	1	0	0
3	0	0	1	1	1
4	0	1	0	0	1
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	1
15	1	1	1	1	0

Now, unto the Karnaugh Map:



AB\CD	00	01	11	10
00	1	1	1	0
01	1	0	1	0
11	0	1	0	1
10	0	0	0	1

Figure 5.24: Karnaugh Map Example 3

Now, we will apply Boolean Algebra to the expression:

$$AB\overline{C}D + \overline{A}BCD + \overline{A}\,\overline{B}CD + \overline{A}B\overline{C}\,\overline{D} + A\overline{B}C\overline{D} + \overline{A}\,\overline{B}\,\overline{C}D + ABC\overline{D} + \overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$$

Applying Distributive Rule

$$AB\overline{C}D + \overline{A}CD(\overline{B}+B) + \overline{A}\,\overline{C}\,\overline{D}(\overline{B}+B) + A\overline{B}C\overline{D} + \overline{A}\,\overline{B}\,\overline{C}D + ABC\overline{D}$$

Applying Inverse Rule

$$AB\overline{C}D + \overline{A}CD1 + \overline{A}\,\overline{C}\,\overline{D}1 + A\overline{B}C\overline{D} + \overline{A}\,\overline{B}\,\overline{C}D + ABC\overline{D}$$

Applying Identity Rule

$$AB\overline{C}D + \overline{A}CD + \overline{A}\,\overline{C}\,\overline{D} + A\overline{B}C\overline{D} + \overline{A}\,\overline{B}\,\overline{C}D + ABC\overline{D}$$

And that is as far as rules can simplify the expression. The resulting expression of manually applying Boolean Algebra is: $AB\overline{C}D + \overline{A}CD + \overline{A}\,\overline{C}\,\overline{D} + A\overline{B}C\overline{D} + \overline{A}\,\overline{B}\,\overline{C}D + ABC\overline{D}$.

Now we will apply the Karnaugh Map method to Example 3, refer to **Figure5.25**.

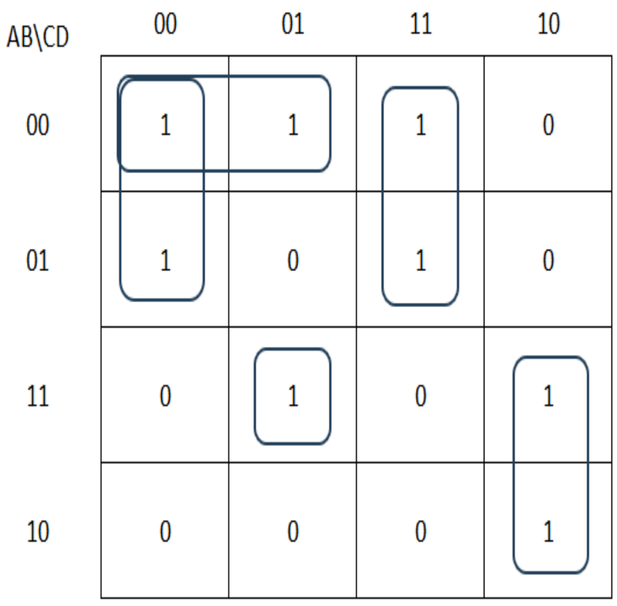
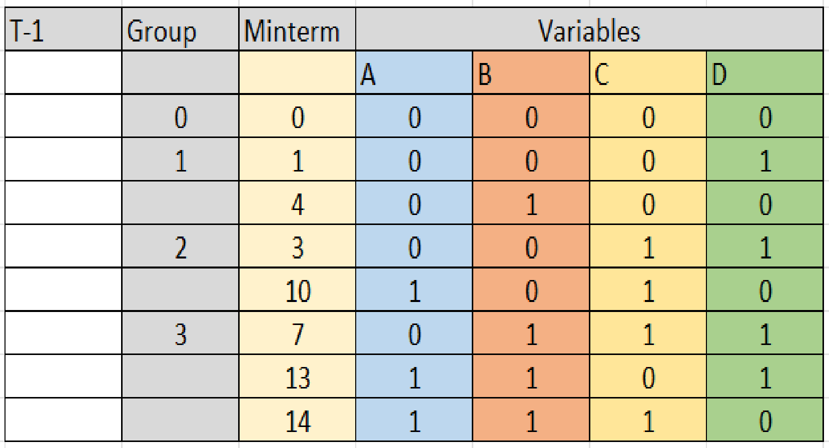


Figure 5.25: Karnaugh Map 2 Example 3

The resulting expression is:

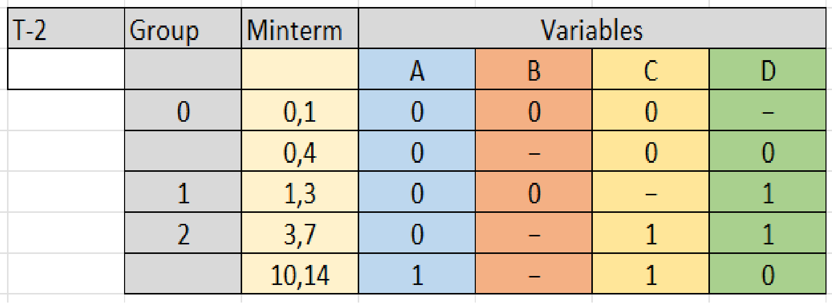
$$\overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,\overline{C}\,\overline{D} + \overline{A}CD + AB\overline{C}D + AC\overline{D}$$

Now we will apply the Quine-McCluskey algorithm:



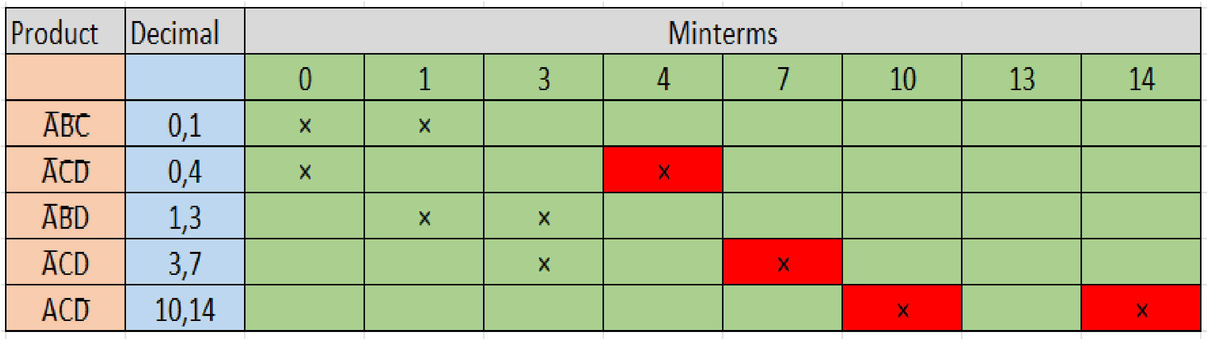
T-1	Group	Minterm	Variables			
			A	B	C	D
	0	0	0	0	0	0
	1	1	0	0	0	1
		4	0	1	0	0
	2	3	0	0	1	1
		10	1	0	1	0
	3	7	0	1	1	1
		13	1	1	0	1
		14	1	1	1	0

Figure 5.26: Quine McClusky Example 3



T-2	Group	Minterm	Variables			
			A	B	C	D
	0	0,1	0	0	0	–
		0,4	0	–	0	0
	1	1,3	0	0	–	1
	2	3,7	0	–	1	1
		10,14	1	–	1	0

Figure 5.27: Quine McClusky 2 Example 3



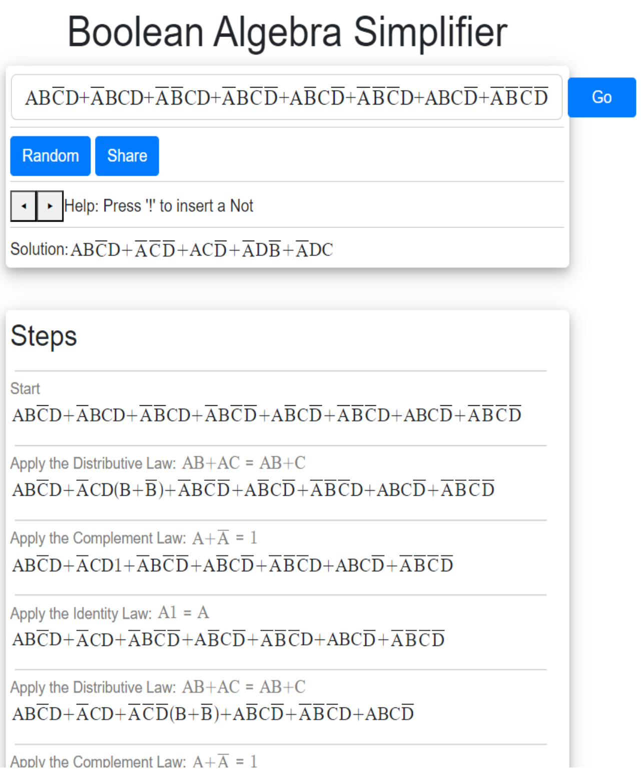
Product	Decimal	Minterms							
		0	1	3	4	7	10	13	14
$\overline{A}\,\overline{B}\,\overline{C}$	0,1	x	x						
$\overline{A}\,\overline{C}\,\overline{D}$	0,4	x			x				
$\overline{A}\,\overline{B}D$	1,3		x	x					
$\overline{A}CD$	3,7			x		x			
$AC\overline{D}$	10,14						x		x

Figure 5.28: Quine McClusky 3 Example 3

The resulting expression is:

$$\overline{A}\,\overline{C}\,\overline{D} + \overline{A}CD + AC\overline{D}$$

Now we will test the expression using some related apps. For applying Boolean Algebra, we will use Boolean Algebra App[1], as it has been giving good results.


Boolean Algebra Simplifier

$AB\overline{C}D+\overline{A}BCD+\overline{A}\overline{B}CD+\overline{A}B\overline{C}\overline{D}+A\overline{B}C\overline{D}+\overline{A}\overline{B}\overline{C}D+ABC\overline{D}+\overline{A}\overline{B}\overline{C}\overline{D}$

Help: Press '!' to insert a Not

Solution: $AB\overline{C}D+\overline{A}\overline{C}\overline{D}+AC\overline{D}+\overline{A}D\overline{B}+\overline{A}DC$

Steps

Start
$AB\overline{C}D+\overline{A}BCD+\overline{A}\overline{B}CD+\overline{A}B\overline{C}\overline{D}+A\overline{B}C\overline{D}+\overline{A}\overline{B}\overline{C}D+ABC\overline{D}+\overline{A}\overline{B}\overline{C}\overline{D}$

Apply the Distributive Law: $AB+AC$ = $AB+C$
$AB\overline{C}D+\overline{A}CD(B+\overline{B})+\overline{A}B\overline{C}\overline{D}+A\overline{B}C\overline{D}+\overline{A}\overline{B}\overline{C}D+ABC\overline{D}+\overline{A}\overline{B}\overline{C}\overline{D}$

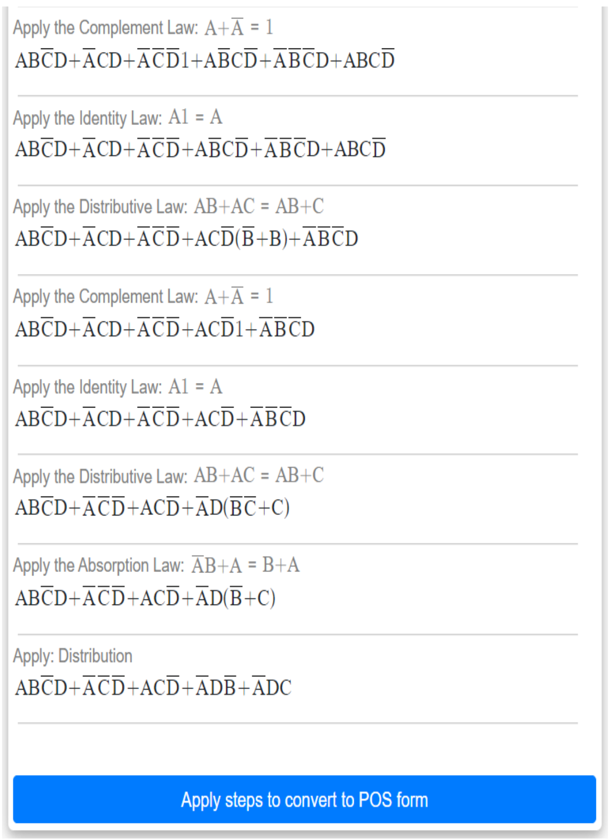
Apply the Complement Law: $A+\overline{A}$ = 1
$AB\overline{C}D+\overline{A}CD1+\overline{A}B\overline{C}\overline{D}+A\overline{B}C\overline{D}+\overline{A}\overline{B}\overline{C}D+ABC\overline{D}+\overline{A}\overline{B}\overline{C}\overline{D}$

Apply the Identity Law: $A1$ = A
$AB\overline{C}D+\overline{A}CD+\overline{A}B\overline{C}\overline{D}+A\overline{B}C\overline{D}+\overline{A}\overline{B}\overline{C}D+ABC\overline{D}+\overline{A}\overline{B}\overline{C}\overline{D}$

Apply the Distributive Law: $AB+AC$ = $AB+C$
$AB\overline{C}D+\overline{A}CD+\overline{A}\overline{C}\overline{D}(B+\overline{B})+A\overline{B}C\overline{D}+\overline{A}\overline{B}\overline{C}D+ABC\overline{D}$

Apply the Complement Law: $A+\overline{A}$ = 1

Figure 5.29: Boolean Algebra App Example 3


Apply the Complement Law: $A+\overline{A}$ = 1
$AB\overline{C}D+\overline{A}CD+\overline{A}\overline{C}\overline{D}1+A\overline{B}C\overline{D}+\overline{A}\overline{B}\overline{C}D+ABC\overline{D}$

Apply the Identity Law: $A1$ = A
$AB\overline{C}D+\overline{A}CD+\overline{A}\overline{C}\overline{D}+A\overline{B}C\overline{D}+\overline{A}\overline{B}\overline{C}D+ABC\overline{D}$

Apply the Distributive Law: $AB+AC$ = $AB+C$
$AB\overline{C}D+\overline{A}CD+\overline{A}\overline{C}\overline{D}+AC\overline{D}(\overline{B}+B)+\overline{A}\overline{B}\overline{C}D$

Apply the Complement Law: $A+\overline{A}$ = 1
$AB\overline{C}D+\overline{A}CD+\overline{A}\overline{C}\overline{D}+AC\overline{D}1+\overline{A}\overline{B}\overline{C}D$

Apply the Identity Law: $A1$ = A
$AB\overline{C}D+\overline{A}CD+\overline{A}\overline{C}\overline{D}+AC\overline{D}+\overline{A}\overline{B}\overline{C}D$

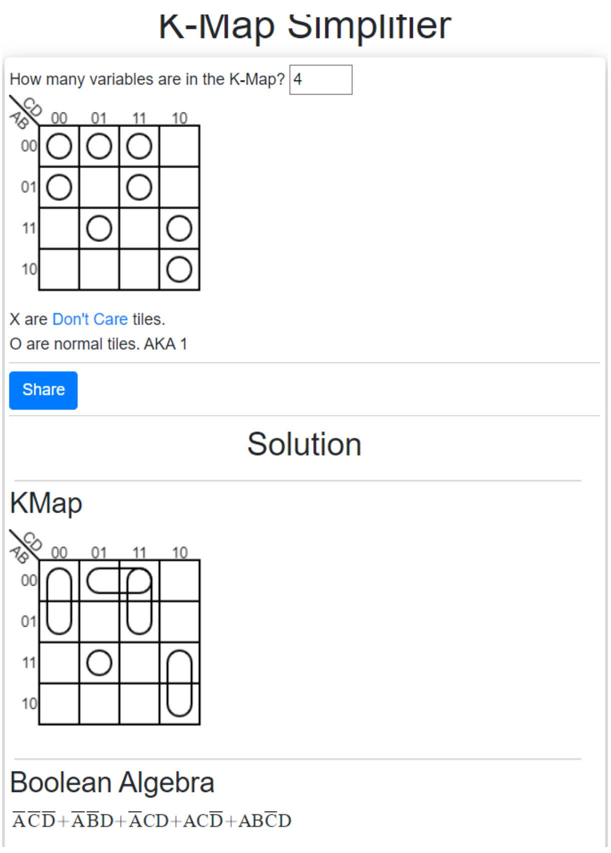
Apply the Distributive Law: $AB+AC$ = $AB+C$
$AB\overline{C}D+\overline{A}\overline{C}\overline{D}+AC\overline{D}+\overline{A}D(\overline{B}\overline{C}+C)$

Apply the Absorption Law: $\overline{A}B+A$ = $B+A$
$AB\overline{C}D+\overline{A}\overline{C}\overline{D}+AC\overline{D}+\overline{A}D(\overline{B}+C)$

Apply: Distribution
$AB\overline{C}D+\overline{A}\overline{C}\overline{D}+AC\overline{D}+\overline{A}D\overline{B}+\overline{A}DC$

Figure 5.30: Boolean Algebra App 2 Example 3

The resulting expression $AB\overline{C}D+\overline{A}\,\overline{C}\,\overline{C}+AC\overline{D}+\overline{A}D\overline{B}+\overline{A}DC$ For Karnaugh we will use Boolean Algebra App's Karnaugh Section.



K-Map Simplifier

How many variables are in the K-Map? 4

X are Don't Care tiles.
O are normal tiles. AKA 1

Share

Solution

KMap

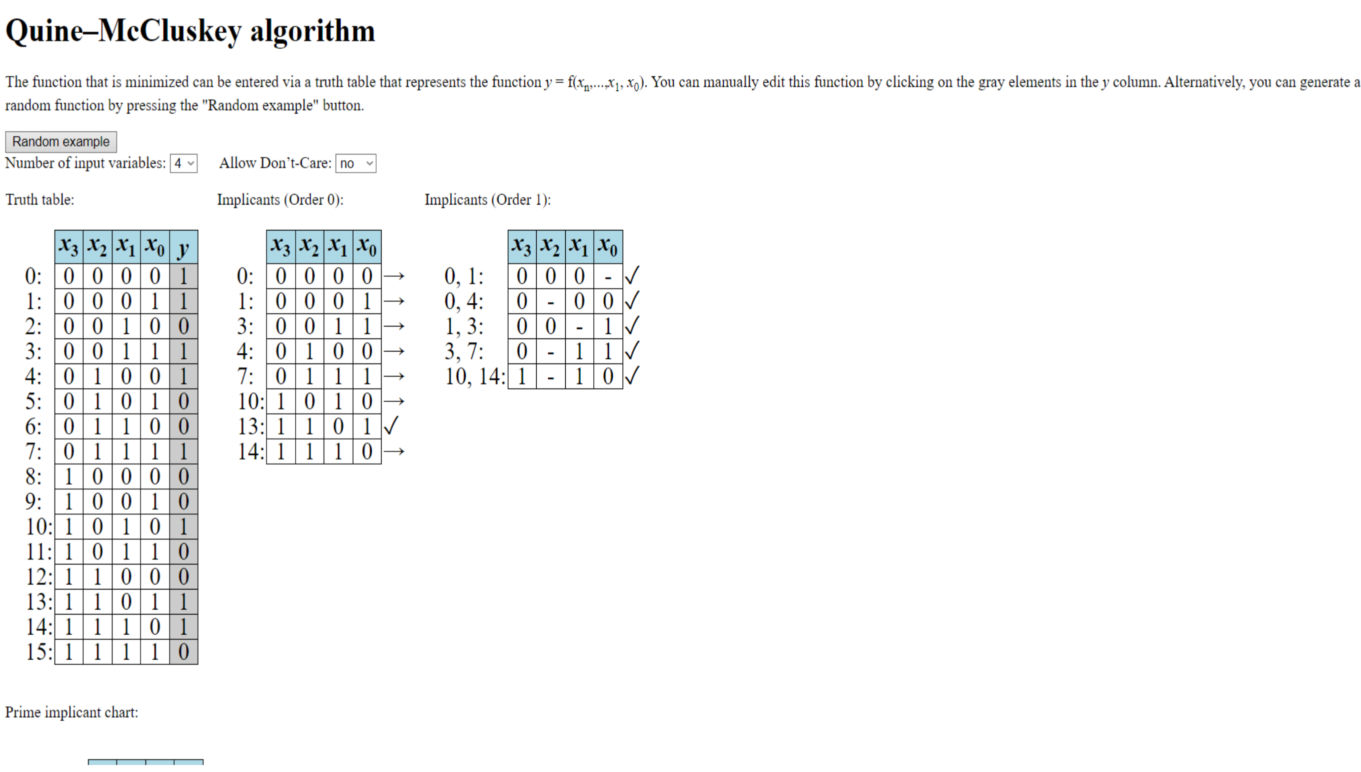
Boolean Algebra

$\overline{A}\,\overline{C}\,\overline{D}+\overline{A}\,\overline{B}D+\overline{A}CD+AC\overline{D}+AB\overline{C}D$

Figure 5.31: Boolean Algebra App/Karnaugh Example 3

The resulting expression $AB\overline{C}D+\overline{A}\,\overline{C}\,\overline{C}+AC\overline{D}+\overline{A}D\overline{B}+\overline{A}DC$

Now, in the case of Quine-McCluskey, we will use the Quine–McCluskey Algorithm app [9].



Quine–McCluskey algorithm

The function that is minimized can be entered via a truth table that represents the function y = f(x_n,...,x_1, x_0). You can manually edit this function by clicking on the gray elements in the y column. Alternatively, you can generate a random function by pressing the "Random example" button.

Random example
Number of input variables: 4   Allow Don't-Care: no

Truth table:

	x_3	x_2	x_1	x_0	y
0:	0	0	0	0	1
1:	0	0	0	1	1
2:	0	0	1	0	0
3:	0	0	1	1	1
4:	0	1	0	0	1
5:	0	1	0	1	0
6:	0	1	1	0	0
7:	0	1	1	1	1
8:	1	0	0	0	0
9:	1	0	0	1	0
10:	1	0	1	0	1
11:	1	0	1	1	0
12:	1	1	0	0	0
13:	1	1	0	1	1
14:	1	1	1	0	1
15:	1	1	1	1	0

Implicants (Order 0):

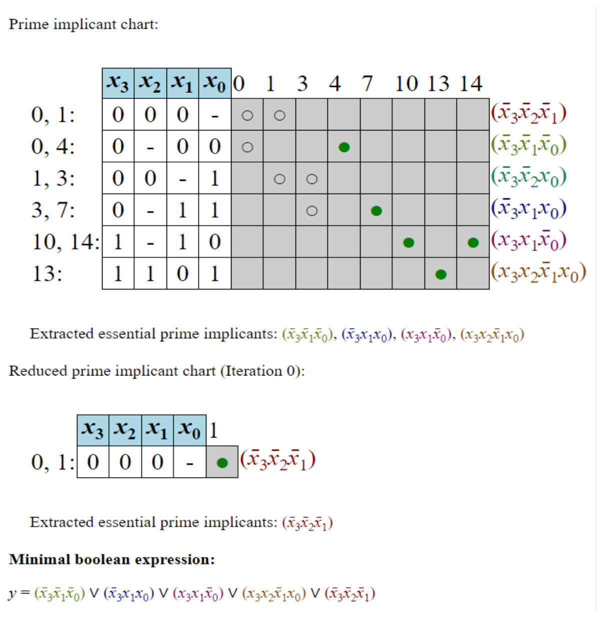
	x_3	x_2	x_1	x_0	
0:	0	0	0	0	→
1:	0	0	0	1	→
3:	0	0	1	1	→
4:	0	1	0	0	→
7:	0	1	1	1	→
10:	1	0	1	0	→
13:	1	1	0	1	✓
14:	1	1	1	0	→

Implicants (Order 1):

	x_3	x_2	x_1	x_0	
0, 1:	0	0	0	-	✓
0, 4:	0	-	0	0	✓
1, 3:	0	0	-	1	✓
3, 7:	0	-	1	1	✓
10, 14:	1	-	1	0	✓

Prime implicant chart:

Figure 5.32: Quine McClusky Algorithm App Example 3



Prime implicant chart:

	x_3	x_2	x_1	x_0	0	1	3	4	7	10	13	14	
0, 1:	0	0	0	-	○	○							$(\bar{x}_3\bar{x}_2\bar{x}_1)$
0, 4:	0	-	0	0	○			●					$(\bar{x}_3\bar{x}_1\bar{x}_0)$
1, 3:	0	0	-	1		○	○						$(\bar{x}_3\bar{x}_2x_0)$
3, 7:	0	-	1	1			○		●				$(\bar{x}_3x_1x_0)$
10, 14:	1	-	1	0						●		●	$(x_3x_1\bar{x}_0)$
13:	1	1	0	1							●		$(x_3x_2\bar{x}_1x_0)$

Extracted essential prime implicants: $(\bar{x}_3\bar{x}_1\bar{x}_0)$, $(\bar{x}_3x_1x_0)$, $(x_3x_1\bar{x}_0)$, $(x_3x_2\bar{x}_1x_0)$

Reduced prime implicant chart (Iteration 0):

	x_3	x_2	x_1	x_0	1	
0, 1:	0	0	0	-	●	$(\bar{x}_3\bar{x}_2\bar{x}_1)$

Extracted essential prime implicants: $(\bar{x}_3\bar{x}_2\bar{x}_1)$

Minimal boolean expression:

$y = (\bar{x}_3\bar{x}_1\bar{x}_0) \vee (\bar{x}_3x_1x_0) \vee (x_3x_1\bar{x}_0) \vee (x_3x_2\bar{x}_1x_0) \vee (\bar{x}_3\bar{x}_2\bar{x}_1)$

Figure 5.33: Quine McClusky Algorithm App 2 Example 3

The resulting expression is:

$$\overline{A}\,\overline{C}\,\overline{D} + \overline{A}CD + AC\overline{D} + AB\overline{C}D + \overline{A}\,\overline{B}\,\overline{C}$$

Now we will compare all of the obtained results to check which one stands out as the better result.

- Manual application of Boolean Algebra:$AB\overline{C}D + \overline{A}CD + \overline{A}\,\overline{C}\,\overline{D} + A\overline{B}C\overline{D} + \overline{A}\,\overline{B}\,\overline{C}D + ABC\overline{D}$
- Manual application of Karnaugh Map:$\overline{A}\,\overline{B}\,\overline{C}+\overline{A}\,\overline{C}\,\overline{D}+\overline{A}CD+AB\overline{C}D+AC\overline{D}$
- Manual application of Quine-McCluskey:$\overline{A}\,\overline{C}\,\overline{D} + \overline{A}CD + AC\overline{D}$
- Boolean Algebra App:$AB\overline{C}D + \overline{A}\,\overline{C}\,\overline{C} + AC\overline{D} + \overline{A}D\overline{B} + \overline{A}DC$
- Boolean Algebra App/Karnaugh:$AB\overline{C}D + \overline{A}\,\overline{C}\,\overline{C} + AC\overline{D} + \overline{A}D\overline{B} + \overline{A}DC$
- Quine McClusky Algorithm:$\overline{A}\,\overline{C}\,\overline{D} + \overline{A}CD + AC\overline{D} + AB\overline{C}D + \overline{A}\,\overline{B}\,\overline{C}$

The result that stands out once again is the Manual application of the Quine-McCluskey algorithm.

5.4 Conclusions

After comparing the results from all three examples, it can be seen that manually applying the Quine-McCluskey algorithm offers the best results.

Most likely because there are no redundant functions or exceptions that occur. When you apply the algorithm manually, you can adapt to any exceptions/issues that occur. we will not exclude the possibility of making mistakes when applying algorithms manually, that is the great issue with doing it manually in comparison to apps. Apps won't make mistakes, but they also may contain errors.

We could say that manually applying Karnaugh map and the remaining apps offer slightly worse results, but they still do the process of simplification well.

The Manual application of Boolean Algebra offers the worst results mainly due to the way we chose to apply the Boolean rules, had we applied the different rules to different terms, the results might have been better, or maybe might have been worse. This is the main issue with Boolean Algebra, depending on how the process is done, the results will be different

Chapter 6

Conclusions and Future Direction

Challenges

The only challenge I had on the writing of the thesis paper was the research on the Quine McClusky Algorithm since it was a completely new method for simplifying logic expressions and it proved to be quite simple to apply with enough practice.

On the coding of the application, I had multiple challenges.

Firstly, in the Boolean Class, was the order in which each function is called, depending on the order, some functions might have to be called multiple times. In order to optimize the process, I tested multiple ways in which I could call the functions.

Secondly, in the Karnaugh Class, I had no idea on how to adapt the simplification process to all possible karnaugh map sizes. Karnaugh maps can vary from 2x4 to 4x4, 4x8, 8x8 and beyond. My issue was how would I make my Class work on any Karnaugh map. I managed to find a way to iterate through each element of the map and instead of looking at the whole map, I look at each element individually.

Thirdly, the amount of issues in all three classes is affecting the functioning of the app itself. In its current state, only the Boolean Class works somewhat properly. The Karnaugh class has an issue when iterating through the elements, I had a confusion when coding with how coordinates of the elements worked. Instead of the coordinates being represented as X and Y they were representing as Y and X. And to fix this is not as simple as swapping them. In the case of Tabulation, I'm still in the middle of finding the issue. The output of the function is empty so I'm still trying to find where the problem is.

Conclusion

As seen in the previous chapters of this Thesis, for the simplification of a complex logic circuit, I have presented 3 methods that can be applied. Those being, Boolean Algebra, Karnaugh Maps and Quine-McClusky Algorithm. Out of those 3 methods, the Quine-McClusky Algorithm ended up being the far superior one due to its better consistency and better results, since they come in a more simplified manner. Also, once learned, the Quine McCluskey algorithm can be quite nimble to use.

What i have achieved

As mentioned in chapter 1.2, by trying to apply the Karnaugh Map method to a Boolean expression that includes a negation that contains more than one variable, that expression could not be applied to the Karnaugh map. Only by previously simplifying the expression could it be applied. Also any parenthesis should be simplified as well.

I also determined the better one of the three methods. By applying each method to the same problem, the Quine McClusky Algorithm showed the best results since, when applying the method, there is only one way to apply it. In the case of Karnaugh Maps and Boolean Algebra, there are many ways one can go to apply them.

I also coded an app that applies all three methods. There are still plenty of features that require fixing. The Karnaugh and Tabulation classes are not functional have still some issues. The logic behind the codes is good and makes sense but it still needs some it still needs some touches.

Future Direction

My future direction is completing the app, as it still has parts that need to be completed Once the app is complete, the app could be used as part of other apps that simplify logic circuits. I had an intention to further simplify the simplified expressions by using only NAND or NOR logic gates, but it was not possible due to my lack of time.

Bibliography

[1] *Boolean Algebra.* https : / / www . boolean - algebra . com/https://www.boolean-algebra.com/.

[2] *Boolean Algebra Calculator.* https : / / www . emathhelp . net / calculators / discrete - mathematics / boolean - algebra - calculator/https://www.emathhelp.net/calculators/discrete-mathematics/boolean-algebra-calculator/.

[3] *Boolean Expression Calculator.* https : / / www . dcode . fr / boolean - expressions-calculator.

[4] *Calculators.tech Boolean Algebra Calculator.* https : / / www . calculators . tech/boolean-algebra-calculatorhttps://www.calculators.tech/boolean-algebra-calculator.

[5] *Four variable Karnaugh Map Solver.* https://getcalc.com/karnaugh-map/4variable-kmap-solver.htmhttps://getcalc.com/karnaugh-map/4variable-kmap-solver.htm.

[6] Morris Mano. *Digital Logic and Computer Design.*

[7] Enoch O.Hwang. *Digital Logic and Microprocessor Design With VHDL.*

[8] *Quine McCluskey Solver.* http : / / quinemccluskey . com/http://quinemccluskey.com/.

[9] *Quine–McCluskey Algorithm.* https : / / www . mathematik . uni - marburg . de / ~thormae / lectures / ti1 / code / qmc/https://www.mathematik.uni-marburg.de/ thormae/lectures/ti1/code/qmc/.

[10] *Symbolab Boolean Algebra Calculator.* https://www.symbolab.com/solver/boolean-algebra-calculatorhttps://www.symbolab.com/solver/boolean-algebra-calculator.