

CS2100 - L18 - Simplification

Week 10 + 11

18.1 - Function simplification (algebraic simplification)

18.2 - Karnaugh maps (K-maps)

- Terminology
- Algorithm
- 2 variables
- 3 variables
- 4 variables
- Simplified POS expression
- "Don't care" conditions

18.1 - Function simplification (algebraic simplification)

Function Simplification

- Why simplify?
 - Simpler expression uses fewer logic gates.
 - Thus cheaper, uses less power, (sometimes) faster
- Techniques
 - **Algebraic**
 - Using theorems
 - Open-ended; requires skills
 - **Karnaugh Maps (K-Map)**
 - Easy to use
 - Limited to no more than 6 variables
 - **Quine-McCluskey** (non-examinable)
 - Suitable for automation
 - Can handle many variables (but computationally intensive)

[L18 - AY2021S1]

Algebraic Simplification

- Aims to minimise
 - Number of literals, and
 - Number of terms
- Sometimes the two aims conflict, so let's aim at reducing the number of literals in this section
- **Difficult** – needs good algebraic manipulation skills

[L18 - AY2021S1]

18.2 - Karnaugh maps (K-maps)

Terminology

Terminology: I, PI and EPI

Implicant:

- A product term that could be used to cover several minterms of the function

Prime Implicant:

- The maximal (largest) possible implicant for a group of minterms

Essential Prime Implicant:

- Prime Implicant that contains 1 or more **unique minterm**

Algorithm

Simplification Algorithm Rephrased

1. Draw the **Prime Implicant(s)** for each minterm in the K-Map
2. Using (1), take all **Essential Prime Implicants**
3. Choose the smallest collection of **Prime Implicant** for the rest of minterms not covered in (2)

[L18 - AY2021S1]

EX: Generalize Observation

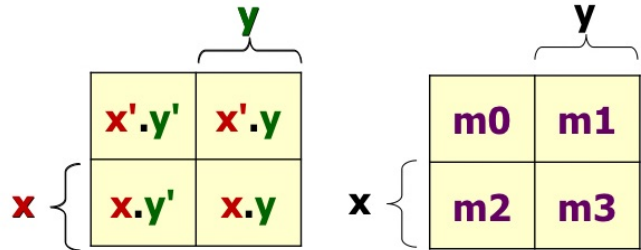
- A group of size 1 can remove 0 literals
- A group of size 2 can remove 1 literals
-<use your power of deduction>.....
- So, a group of size X can remove $\frac{\log_2 X}{K}$ literals

[L18 - AY2021S1]

2 variables

2-Variables K-Map

x	y	F
0	0	m0
0	1	m1
1	0	m2
1	1	m3



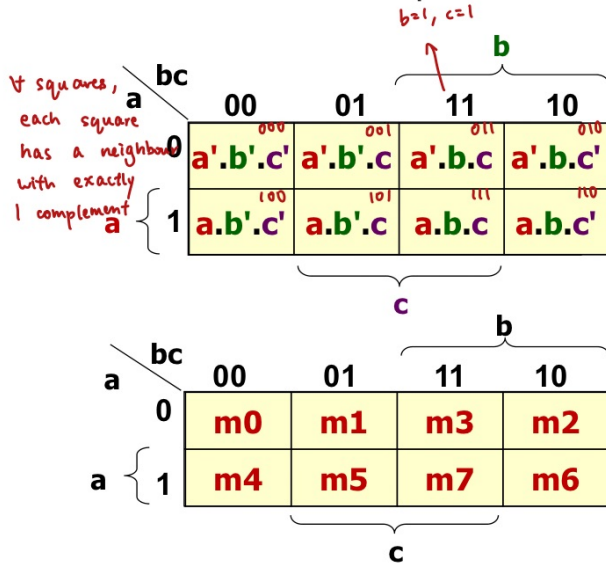
- Other layouts are possible:
 - e.g. swap X and Y, reverse the rows / columns etc
- We will use this one due to its easily remembered association to the truth table entries

3 variables

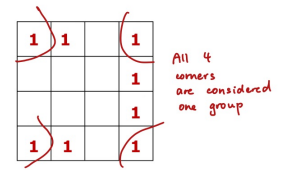
3-Variables K-Map

- There are 8 squares in a 3-variable K-map

a	b	c	F
0	0	0	m0
0	0	1	m1
0	1	0	m2
0	1	1	m3
1	0	0	m4
1	0	1	m5
1	1	0	m6
1	1	1	m7

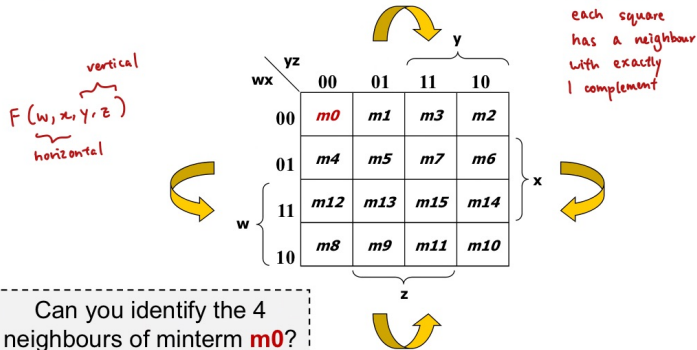


4 variables



4-Variables K-Map

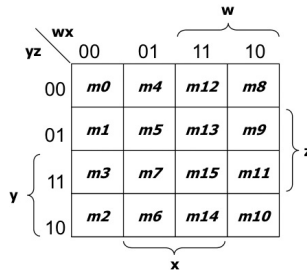
- 16 square cells in a 4-variable K-map:
 - Each cell has **4 neighbours**
 - Note the wraps around along the edges



[L18 - AY2021S1]

4-Var K-Map: Alternative Placement

- Beware that there is another popular 4 variable K-Map placements:



Can you spot the **differences**?

[L18 - AY2021S1]

Simplified POS Expression

- To find the **Simplified POS expression** of a function F:
 1. Find the Simplified **SOP expression** for the **F'**
 2. Negate (1) to get simplified **POS expression** of **F**
- Find the simplified POS for $F = \Sigma m(0,2,4,5,6,8,10,12,13,14)$

K-map of **F**

		C			
	CD	00	01	11	10
AB	00	1	0	0	1
	01	1	1	0	1
	11	1	1	0	1
	10	1	0	0	1
A		B			

Groupings: **B** (vertical group of 0s in column 11), **D** (horizontal group of 0s in row 10), **C** (horizontal group of 0s in row 00).

K-map of **F'**

		C			
	CD	00	01	11	10
AB	00	0	1	1	0
	01	0	0	1	0
	11	0	0	1	0
	10	0	1	1	0
A		B			

Groupings: **B** (vertical group of 1s in column 11), **D** (horizontal group of 1s in row 10), **C** (horizontal group of 1s in row 00).

SOP of **F' = $B' \cdot D + C \cdot D$** → POS of **F =** _____

"Don't care" conditions

Don't Care Condition (1/3)

- In certain problems, some outputs are not specified or are invalid
- Hence, these outputs can be either '1' or '0'
- They are called **don't-care conditions**, denoted by **X** (or sometimes, **d**)
- Example: **ABCD** represents a digit (0..9). Function **P(A,B,C,D)** counts whether there are **even** number of '1's in the binary representation
 - Input (1010...1111) are invalid and we don't care about the output for them

A	B	C	D	P
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

[L18 - AY2021S1]

Don't Care Condition (2/3)

- Don't-care conditions can be used to help simplify Boolean expression further in K-maps
 - Could be chosen to be **either** '1' or '0', depending on which choice **results in a simpler expression**
- We can use the notation Σd to denote the set of don't-care minterms.
 - e.g., the function **P** in the previous slide can be written as:

$$P = \Sigma m(0, 3, 5, 6, 9) + \Sigma d(10, 11, 12, 13, 14, 15)$$

[L18 - AY2021S1]