COSC2473

Digital Logic

Logic Gates & Bit Masking

Boolean Algebra

Boolean Algebra

- Boolean algebra has many of the same rules as normal high school level algebra, and indeed the symbols used (+ for OR, * for AND, etc) reflect this.
  - And just like algebra, you can use various methods to simplify.
  - We will show here two methods:

- Algebraic Simplification
  - Can work for any number of variables

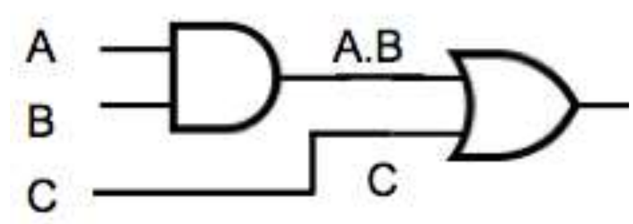
- Graphical Simplification using Karnaugh maps
  - This works for up to 4 variables, but beyond that, it gets too complicated.

Boolean Logic Rules

- Reminder of the Boolean Logic rules

		NOT	AND	OR	XOR
A	B	~A	•	+	⊕
0	0	1	0	0	0
0	1	1	0	1	1
1	0	0	0	1	1
1	1	0	1	1	0

- Typical Boolean expressions are:
  - C + AB = true
    - if C is true or both A and B are true and is equivalent to the circuit.at right



Boolean Algebra Rules

Commutative law	$A + B = B + A$
Associative law	$A + (B + C) = A + B + C = (A + B) + C$ $A(BC) = ABC = (AB)C$
Distributive law	$A(B + C) = AB + AC$
de Morgan's theorems	$\overline{A + B} = \overline{A}.\overline{B}$ $\overline{A.B} = \overline{A} + \overline{B}$
Identities	$1 + any = 1, \quad 0 + A = A$ $1 \, . \, A = A, \quad 0 \, . \, any = 0$ $A \, . \, A = A, \quad A + A = A$ $A + \overline{A} = 1, \quad A \, . \, \overline{A} = 0$ $AB + \overline{A}B = B$ $A + BC = (A + B)(A + C)$

Algebraic Simplification and Python

Example of simplification and how it is used in programming

$$A + /AC \quad = (A + /A)(A + C),$$

$$\text{using the pattern } A + BC = (A+B)(A+C)$$

$$= 1\ (A + C = A + C, \quad \text{since} \quad (A + /A) = 1$$

<u>Python Code</u>

Let's test the above. Suppose we have two sensors,S1, S2 = +/- 5 volts, and an activation decision Z

```
>>> S1=-5;  S2=-5;  A=S1>0;  C=S2>0;  Z = A or (not A and C); print(A,C,Z)
False False False
>>> S1=-5;  S2=+5;  A=S1>0;  C=S2>0;  Z = A or (not A and C); print(A,C, Z)
False True True
>>> S1=+5;  S2=-5;  A=S1>0;  C=S2>0;  Z = A or (not A and C); print(A,C, Z)
True False True
>>> S1=+5;  S2=+5;  A=S1>0;  C=S2>0;  Z = A or (not A and C); print(A,C, Z)
True True True
>>>
```

<u>Proof</u>

The above values of A,B,Z look like a truth table for Z=A + C

Algebraic Simplification

- Example of simplification

1. Eliminating common variables

$A + AB = A\underline{(1 + B)} = A,$ *since 1+anything = 1*

$AB + /AB = \underline{(A + /A)}B = B,$ *where /A = NOT A*

2. More Complex Factoring

$A + BC = (A + B)(A + C)$ *multiply out, so*

$= \underline{A.A} + A.C + \underline{B.A} + B.C,$ *but AA=A, BA=AB*

$= A + A.C + A.B + B.C,$ *but A = A.1, so*

$= \underline{A.1} + A.C + AB + B.C,$ *factor out A, so*

$= A.(\underline{1 + C + B}) + B.C,$ *but (1+any)=1, so*

$= A.(1) + B.C,$ *but A.1 = A, so*

$= A + B.C$ *…QED*

Using Truth Table as Proof

- We can also prove the previous equation by showing all combinations in a truth table. We show that Left hand side (LHS) = right hand side (RHS)

A	B	C	X B.C	LHS A + X	Y A + B	Z A +C	RHS Y . Z
0	0	0	*0*	*0*	*0*	*0*	*0*
0	0	1	*0*	*0*	*0*	*1*	*0*
0	1	0	*0*	*0*	*1*	*0*	*0*
0	1	1	*1*	*1*	*1*	*1*	*1*
1	0	0	*0*	*1*	*1*	*1*	*1*
1	0	1	*0*	*1*	*1*	*1*	*1*
1	1	0	*0*	*1*	*1*	*1*	*1*
1	1	1	*1*	*1*	*1*	*1*	*1*

De Morgan's Theorem

- One way in which Boolean algebra is different to other forms is the following
  - Most commonly used to change a term involving ORs into one involving ANDs, and vice versa
  - It works due to the symmetry of the OR and AND truthtables

$$\boxed{\begin{aligned}\overline{A+B} &= \overline{A}.\overline{B}\\ \overline{A.B} &= \overline{A}+\overline{B}\end{aligned}}$$

A	B	$\overline{A}$	$\overline{B}$	A.B	$\overline{A}+\overline{B}$
0	0	1	1	0	1
0	1	1	0	0	1
1	0	0	1	0	1
1	1	0	0	1	0

- One is basically the upside-down inverse of the other
  - You can get upside-down by complementing the operands A and B
  - and then simply map AND to OR

- de Morgan's Laws are also applied to Set Theory, where OR ⇔ ∪ (set union)
  AND ⇔ ∩ (set intersection)
  NOT ⇔ ' (set complement)

A Practical Example

- Consider a platform elevator on Flinders Street Station
  - Would you know how to program a 2 level elevator?
  - Where do you start?
  - Consider the cases
    - 2 levels: Concourse, Platform
    - Up and Down buttons on each level outside of the car
    - Up and Down buttons inside the car
    - Activate motor to travel up or down
    - Door open sensor to decide whether to move the car
    - Door Open/Close on arrival or based on inside/outside buttons

- Not so simple, right?

Flinders Street Platform Elevator 1

- Consider a platform elevator on Flinders Street Station
  - 2 levels: Concourse (C), Platform (/C) C
  - Up button on lower level outside car B_{OU}
  - Down Button on upper level outside car B_{OD}
  - Up button inside car B_{IU}
  - Down button inside car B_{ID}
  - Door open sensor (1 = closed) S

  - Travel Up for the car M_U
  - Travel Down for the car M_D
  - Door open / close Activator A

Flinders Street Platform Elevator 2

- We can simplify further
  - Position of car (1=Concourse, 0= Platform) C
  - Up button on lower level outside car U
  - Down Button on upper level outside car D
  - Lever inside car L = 1 (Up) or 0 (Down) L

  - Motor = 1 (travel Up) or -1 (down) or 0 (do nothing) M

  - Assume doors open automatically on arrival, or button push when already there.

Flinders Street Platform Elevator 3

- We can simplify further
  - Position of car (1=Concourse, 0= Platform) — C
  - Outside Button calls the car — B
  - Lever inside car L = 1 (Up) or 0 (Down) — L

  - Motor = 1 (travel Up) or -1 (down) or 0 (do nothing) — M

- What is needed for the motor to start?

```
If        (B /C L)          M = +1
else if   (B C /L)          M = -1
else                        M = 0

M = 0
If (B)
    if    (C and not L)          M = -1
    if    (L and not C)          M = +1
```

Graphical Simplification

- A Karnaugh Map is a pictorial representation of the bit patterns arising from a truth table

- Suppose we have 2 variables A,B

  X = A + /AB

  $= T_1 + T_2$

1. We populate the grid with our terms

	A	**/A**
B	1	2
/B	1	

– More: https://www.youtube.com/watch?v=3vkMgTmieZI

Karnaugh Maps – 2 Variables

- A Karnaugh Map is also ***"a special form of truth table that enables easier pattern recognition"*** by humans.

- Suppose we have 2 variables A,B

  X = A + /AB

  $= T_1 + T_2$

1. We populate the grid with our terms
2. We then draw the largest possible overlapping rectangles that can cover all terms

	A	**/A**
B	1	2
/B	1	

Karnaugh Maps – 2 Variables

- Suppose we have 2 variables A,B

  X = A + /AB

  $= T_1 + T_2$

1. We populate the grid with our terms
2. We then draw the largest possible overlapping rectangles that can cover all terms
3. We then describe these rectangles as:

   X = vert-rect + horiz-rect

   = A + B

	A	/A
B	1	2
/B	1	

Karnaugh Maps – 2 Variables

- Equivalent Boolean Logic simplification

  X = A + /AB
  = A(B + /B) + /AB
  = AB + A/B + /AB + AB
  = A(B + /B) + B(/A + A)
  = A + B
  = / (/A/B) …de Morgan's

Extra Term = Overlap

	A	**/A**
B	1	1
/B	1	

  – See: Karnaugh Maps Introduction
  https://www.youtube.com/watch?v=A0XupfXiKIo

Karnaugh Maps – Term 1

- Now we have 3 variables A, B, C on a 4 variable map
  - X = AB + AC + /AB/C + /B/C
  - X = $T_1 + T_2 + T_3 + T_4$ 'minterms'
  - AB

	AB	A/B	/A/B	/AB
CD	1			
C/D	1			
/C/D	1			
/CD	1			

Karnaugh Maps – Term 2

- Now we have 3 variables A, B, C on a 4 variable map
  - X = AB + AC + /AB/C + /B/C
  - X = $T_1 + T_2 + T_3 + T_4$ 'minterms'
  - AC

	AB	A/B	/A/B	/AB
CD	2	2		
C/D	2	2		
/C/D				
/CD				

Karnaugh Maps – Term 3

- Now we have 3 variables A, B, C on a 4 variable map
  - X = AB + AC + /AB/C + /B/C
  - X = $T_1 + T_2 + T_3 + T_4$ 'minterms'
  - /AB/C

	AB	A/B	/A/B	/AB
CD				
C/D				
/C/D				3
/CD				3

Karnaugh Maps – Term 4

- Now we have 3 variables A, B, C on a 4 variable map
  - X = AB + AC + /AB/C + /B/C
  - X = $T_1 + T_2 + T_3 + T_4$ 'minterms'
  - /B/C

	AB	A/B	/A/B	/AB
CD				
C/D				
/C/D		4	4	
/CD		4	4	

Karnaugh Maps – A

- Karnaugh map rects must be as large as possible

- Now we have 3 variables A, B, C on a 4 variable map
  - X = AB + AC + /AB/C + /B/C
  - X = $T_1 + T_2 + T_3 + T_4$ 'minterms'
  - Make largest overlapping rect
  - A

	AB	A/B	/A/B	/AB
CD	12	2		
C/D	12	2		
/C/D	1	4	4	3
/CD	1	4	4	3

Karnaugh Maps – /C

- Karnaugh map rects must contain 2^n elements <u>only</u>
- Now we have 3 variables A, B, C on a 4 variable map
  - X = AB + AC + /AB/C + /B/C
  - $X = T_1 + T_2 + T_3 + T_4$ 'minterms'
  - Make largest overlapping rect
  - A + /C

	AB	A/B	/A/B	/AB
CD	12	2		
C/D	12	2		
/C/D	1	4	4	3
/CD	1	4	4	3

K-Map – Confirm using deMorgan’s

- Confirm using deMorgan’s Theorem
- Now we have 3 variables A, B, C on a 4 variable map
  - X = AB + AC + /AB/C + /B/C
  - X = $T_1 + T_2 + T_3 + T_4$ ’minterms’
  - Make largest overlapping rect but for the unoccupied parts
  - Then invert this.
  - /(/AC)

	AB	**A/B**	**/A/B**	**/AB**
CD	12	2		
C/D	12	2		
/C/D	1	4	4	3
/CD	1	4	4	3

Karnaugh Maps – 4 Variables

- Karnaugh map rects must contain no gaps

- Now change a term to add a fourth variable, D
  - X = AB + ACD + /AB/C + /B/C
  - X = T_1 + T_2 + T_3 + T_4 'minterms'
  - So T_2 now is ACD
  - The 2's are changed
  - Gap in A/BC/D
  - So cannot use A rect
  - Now
    X = AB + ACD + /C

	AB	A/B	/A/B	/AB
CD	12	2		
C/D	1			
/C/D	1	4	4	3
/CD	1	4	4	3

Karnaugh Maps – Wrap-around Rects

- Karnaugh map rects can also wrap around.

- Now change a term to show wrap on 3 variable K-map
  - X = AB + <u>BC</u> + /AB/C + /B/C
  - X = T_1 + T_2 + T_3 + T_4 'minterms'
  - So T_2 now is BC
  - The rect for T_2 now wraps around

	AB	**A/B**	**/A/B**	**/AB**
C	12			2
/C	1	4	4	3

  - The biggest rect for T_2 is 1x2 that wraps around and corresponds to B
  - This entirely covers T_1, so we can remove it.
  - Finally
    X = <u>B</u> + /C

K-Maps – General Rules

- General comments about K-maps
  - Expressions must be organised into “minterms”, which are “sums of products” of the form
    $X = T_1 + T_2 + \ldots$ where T is a group of ANDed variables.
  - Area of rect R = $2^n / m$, where n is the total number of variables, and m is the number of variables in this particular term.
    - In previous slide, n=3 variables, m for T_2 (BC) = 2, so $R=2^3/2 = 4$

- For example:
  Y = A(B + C) is in the wrong form, multiply out
  Y = AB + AC = 2 rects of 4 elements each
  before populating the map.

Summary

- So we have three ways of simplifying Boolean expressions
  - First Principles
    - Good as a first choice for when you know the situation, such as the elevator example where we simplified the algorithm first.
  - Boolean Algebra
    - Good when you have a few Boolean condition variables, culminating into a small number of decision variables
  - Karnaugh Maps
    - As a graphical way of simplification, but it does not scale well beyond about 6 variables (if you can manage the 3D).
  - Truth Tables
    - The lowest level, but also the most robust, and the easiest to scale and automate.

Quiz

- Using K-maps, simplify the expression

  X = A.B + A./B .C + D(/C.+ /BC)

  using the following steps.

  1. Rewrite expression in terms of minterms
  2. Assign minterm numbers
  3. Draw each minterm on the K-Map
  4. Draw as few maximal overlapping rectangles as possible to cover all the non-blank entries
  5. List the terms corresponding to each rectangle.

	AB	**A/B**	**/A/B**	**/AB**
CD				
C/D				
/C/D				
/CD				