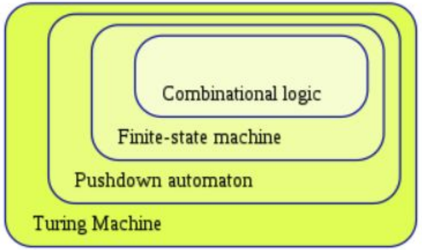
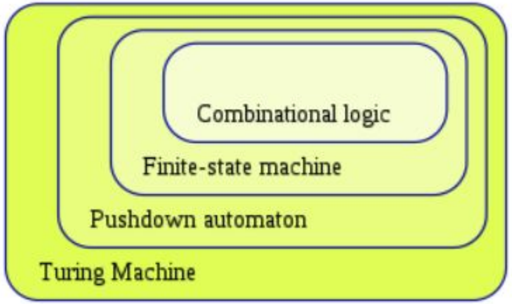
Lecture



- Section 2: Digital Logic: Combinational and Sequential Circuits
    - 2 weeks followed by Quiz #2

1. Boolean Algebra ⇔ Digital Circuits
2. Combinational Circuits: half adder, full adder, decoders, multiplexers, BCD-7 segment
3. Sequential Circuits: Latches, Registers, and Memory
4. Pipeline Architecture: MIPS

Models of Computation

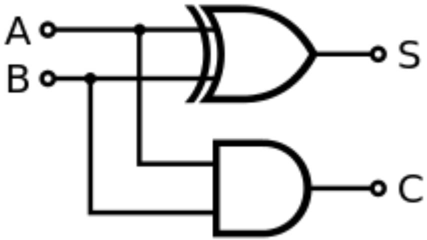


Turning Machines	Recursively Enumerable	TM(Q, Σ, Γ, q0, δ)
Linear Bounded Automata	Context Sensitive Languages	LBA(Q, Σ, Γ, q0, δ)
Pushdown Automata	Context Free Languages	PDA(Q, Σ, Γ, δ, q0, z0, F)
Finite State Automata	Regular Expressions	FA(Q, Σ, δ, q0, F)
Sequential Circuits		
Combinational Logic	Boolean Algebra	

Combinational Logic

- Based upon Boolean Algebra
  - all inputs and outputs restricted to True (1) and False (0)
- Operations are restricted to: AND (*), OR (+), NOT (')
- Equivalent to Digital Logic, with gates:

- Can be used as a building blocks: 
  - XOR: $A \oplus B$ is equivalent to $(A + B) * (A' + B')$
- Example: Half-Adder



Boolean Algebra ⇔ Digital Circuits

- Circuit: a digital realization of a function: y = f(), y = f(x), y =f(a,b), …
- Values:
  - True (T), 1, +5v
  - False (F), 0, ⏚
- Functions:
  - zero inputs: clear (0), set (1)
  - one input: clear, invert, id, set

Input A	Possible Outputs			
0	0	1	0	1
1	0	0	1	1

0 0

0 1

1 0

1 1

#Inputs	Input Combinations		Output Combinations	
0	2^0	1	2^2^0	2
1	2^1	2	2^2^1	4
2	2^2	4	2^2^2	16

Two Input Functions

0010
0011
0100
1000
1001
…
1111

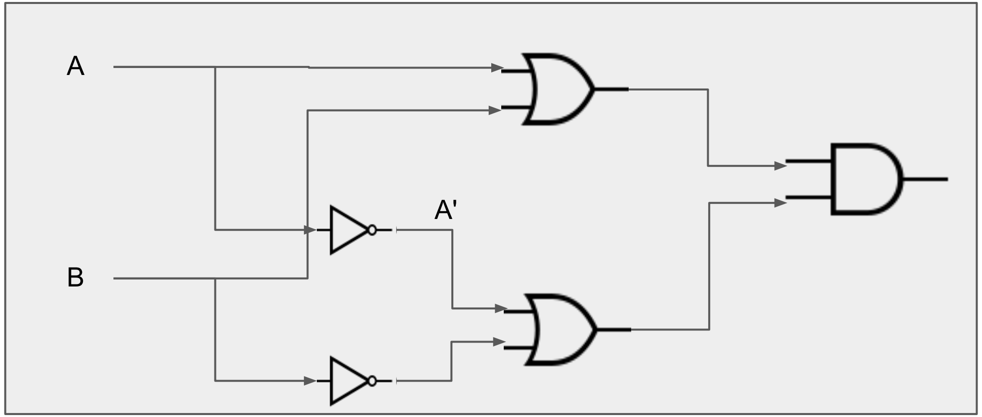
- 16 different functions can be created: $2\text{^ } 2^2$
- List of functions:
- Definitions: Truth Table:

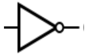
clear	**set**
nor	**or**
↚	←
A'	A
↛	→
B'	B
neq	equiv
nand	**and**

fold line

A	B	0	nor	↚	A'	↛	B'	'equiv	nand	and	equiv	B	→	A	←	or	1
0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Boolean Algebra to Circuits



- Clear: 0 →
- Set: 1 →
- A: A →
- A': A 

- A + B: 

- A * B: 

- A ⊕ B: 

- A ⊕ B: (A + B) * (A' + B')

A	B	xor	and	or
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	0	1	1

Truth Table Reduced to a Circuit

- Generate a circuit as a sum of products
- Values: 0, 1
- Functions: Not ('), And (*), Or (+)

- Example: Implication: A ← B
    - ~~If A is true then B must also be true!~~
- Consider the Truth Table
- Evaluate each row:
- Combine all rows that are true
    - Output = ~~A'B' + A'B + AB~~
    - Output = A'B' + AB' + AB

A	B	Output	
0	0	1	A'B'
0	1	0	A'B
1	0	1	AB'
1	1	1	AB

output: A'B'C' + A'BC' + AB'C + ABC' + ABC

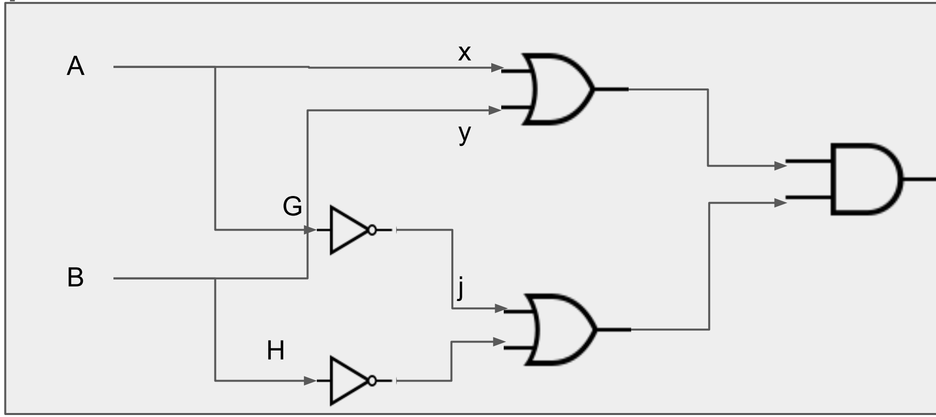
More Complex Reduction

A	B	C	Output
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

A'B'C'
A'B'C
A'BC'
A'BC
AB'C'
AB'C
ABC'
ABC

Circuit to Boolean Algebra

- One Formula for each output
- Work Backwards, Divide and Conquer!



Output = (x + y) * (j + k)

Output = (A + B) * ('G + 'H)

Output = (A + B) * ('A + 'B)

$3 * (5 + 6) == 3 * 5 + 3 * 6$

Simplification ⇔ Minimization

- The above method can yield large circuits
  - Unnecessary large circuits, like programs, are bad:
    - require larger chips
    - require more time to evaluate
- Two approaches:
  - Use algebraic properties to rewrite the formulas
  - Use Karnaugh maps to visualize patterns of simplification
- Algebraic Properties:
  - associative, commutative, distributive
  - complement ($A*A' \Leftrightarrow 0$), De Morgan's law : $A' * B' \Leftrightarrow (A + B)'$
  - $A' + A \Leftrightarrow 1$
- Example
  - $R = A'*B + A*B$

$R = A'*B + A*B$

$R = B*A' + B*A$

$R = B*(A' + A)$

$R = B * 1$

$R = B$

Karnaugh Map

- Truth Table: R = A'*B + A*B

A	B	Output
0	0	0
0	1	1
1	0	0
1	1	1

'(A'B')
A'B
'(AB')
AB

- Karnaugh Map: for two variables

		B	
		0	1
A	0	0	1
	1	0	1

- Find groups that contain only 1s
- Group size must be in power of two (1, 2, 4, 8)
- R = B'

00
01
10
11

3-Variable Karnaugh Map

A	B	C	Output
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

		BC			
		00	01	11	10
A	0	0	1	0	1
	1	0	0	1	1

output = A'B'C + AB + A'BC'

Rules for K-maps

- Groups only contains 1s
- Group size must be a power of two
- Groups must be rectangular, no diagonals
- Groups can overlap
- Groups can wrap
- For Minimization:
  - Use the largest group
  - Use the fewest number of groups
  - But all 1s must be contained in at least one group.

		BC			
		00	01	11	10
A	0	0	1	1	1
	1	1	1	0	0

output = A'C + AB' + A'BC'

		BC				BC			
		00	01	11	10	00	01	11	10
A	0	1			1	1			1
	1	1			1	1			1