COMSM1302
Overview of Computer Architecture

Lecture 2

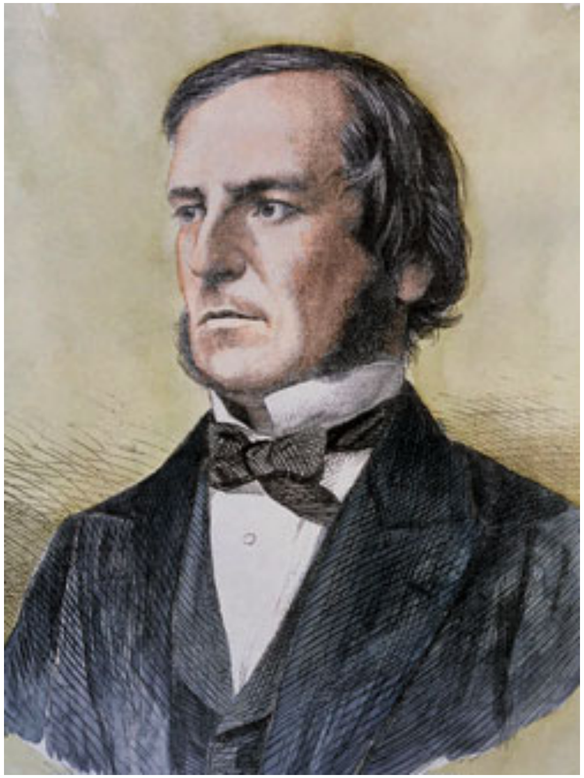
Propositional logic and Boolean algebra

In this lecture

Foundations	• Data representation, **logic, Boolean algebra.**
Building blocks	• Transistors, transistor based logic, simple devices, storage.
Modules	• Memory, simple controllers, FSMs, processors and execution.
Programming	• Machine code, assembly, high-level languages, compilers.
Wrap-up	• Operating systems, energy aware computing.

The origins of logic

- Boole, **1840s**
  - Work on algebraic logic, Boolean algebra.
  - This later enabled the work of Shannon (1948) and others in digital logic circuits, for **communication and computation**.



George Bool (1815-1864)

Image from By Unknown - http://schools.keldysh.ru/sch444/museum/1_17-19.htm, Public Domain, https://commons.wikimedia.org/w/index.php?c...=196...97

Propositional logic

- A *proposition* is a statement that meets the following criteria
  - We can determine a truth value, true or false, for it.
    - Giving a 1 bit result, i.e.

      either a proposition is true or it is false.
  - It is unambiguous.
- To determine whether any given statement is a proposition, prefix it with ***"It is true that . . ."*** and check whether the result makes sense.

Propositional logic

- Which of the below statements are propositions?
  - We can tell whether or not the statement is true or false.
  - The statement is unambiguous.

1. Good morning!
2. The temperature is 20 degrees C.
3. It is warm.
4. Who is speaking?
5. 1+1
6. 1+1=3
7. Grass is blue.
8. There is more than 5 ml of milk in this jug.
9. Love has five letters.

Propositional logic

- Which of the below statements are propositions?
  - Can give a truth value (true or false), when evaluated
  - Is unambiguous

1. ~~Good morning!~~
2. The temperature is 20 degrees C.
3. It is warm.
4. ~~Who is speaking?~~
5. ~~1+1~~
6. 1+1=3
7. Grass is blue.
8. There is more than 5 ml of milk in this jug.
9. Love has five letters.

More examples?

What about "This statement is false." (a paradox)

Propositional logic

- Propositions can be represented as short-hand using propositional variables.

Propositional variable	**Assigned meaning**
• f:	The temperature is 20 degrees C.
• w:	It is warm.
• s:	It is sunny.
• light_on:	The light is on.

Compound propositions

- Propositions can be combined with connectives, using brackets to clarify precedence, as necessary.
- The resulting statements are compound propositions, which can be combined again with connectives, etc.

Examples:

- f: The temperature is 20 degrees C.
- ¬f: The temperature is **not** 20 degrees C.

  not (The temperature is 20 degrees C.)
- w ∧ s: It is warm and it is sunny.

  (It is warm *and* sunny.)

Connectives

- We have seen how, by using logic connectives, compound propositions can be assembled.

Natural	**Formal**	**C (logical operators)**
not x	$\neg x$	! x
x and y	$x \wedge y$	x && y
x or y	$x \vee y$	x || y
x or y but not x and y Exclusively x or y	$x \oplus y$	

Can you have your cake and eat it?

- Let x be "You can have your cake"
- Let y be "You can eat your cake"
- What does the following statement mean?

$$x \oplus y$$

Implication and equivalence

- x ➡ y — x implies y
  if x then y

  "If you work hard, then you will pass the exam."

- x ⬌ y — x is equivalent to y
  x if and only if y
  x iff y

  "My cat comes in if and only if it is hungry."

Reviewing the connectives

- Natural language is flexible. Logic requires formal notation and well defined semantics (meaning).
- Symbols vary between subjects, but they are usually very similar.
- The symbols shown below are commonly recognisable/used.

Symbol	Description	Formal name	
$\neg$	not	Complement	($\overline{A}$)
$\wedge$	and	Conjunction	(.)
$\vee$	or	(Inclusive) disjunction	(+)
$\oplus$	exclusive-or (xor)	(Exclusive) disjunction	
$\Rightarrow$	if then	Implication	
$\Leftrightarrow$	if and only if	Equivalence	

Truth tables

- The meaning of connectives can be represented as a **truth table**, where the input(s) produce a particular output.

Truth tables

- The meaning of connectives can be represented as a **truth table**, where the input(s) produce a particular output.

A	¬A
False	
True	

Truth tables

- The meaning of connectives can be represented as a **truth table**, where the input(s) produce a particular output.

A	¬A
False	
True	

A	B	A ∧ B
False	False	
False	True	
True	False	
True	True	

Truth tables - I

- The meaning of connectives can be represented as a **truth table**, where the input(s) produce a particular output.

A	¬A
False	**True**
True	**False**

A	B	A ∧ B
False	False	**False**
False	True	**False**
True	False	**False**
True	True	**True**

A	B	???
False	False	**False**
False	True	**True**
True	False	**True**
True	True	**True**

Truth tables - I

- The meaning of connectives can be represented as a **truth table**, where the input(s) produce a particular output.

A	¬A
False	**True**
True	**False**

A	B	A ∧ B
False	False	**False**
False	True	**False**
True	False	**False**
True	True	**True**

A	B	A ∨ B
False	False	**False**
False	True	**True**
True	False	**True**
True	True	**True**

Truth tables - II

- The meaning of connectives can be represented as a **truth table**, where the input(s) produce a particular output.

A	B	A ⊕ B
False	False	
False	True	
True	False	
True	True	

Truth tables - II

- The meaning of connectives can be represented as a **truth table**, where the input(s) produce a particular output.

A	B	A ⊕ B
False	False	**False**
False	True	**True**
True	False	**True**
True	True	**False**

A	B	A ⇒ B
False	False	
False	True	
True	False	
True	True	

Truth tables - II

- The meaning of connectives can be represented as a **truth table**, where the input(s) produce a particular output.

A	B	A ⊕ B
False	False	**False**
False	True	**True**
True	False	**True**
True	True	**False**

A	B	A ➡ B
False	False	**True**
False	True	**True**
True	False	**False**
True	True	**True**

A	B	A ⬌ B
False	False	
False	True	
True	False	
True	True	

Truth tables - II

- The meaning of connectives can be represented as a **truth table**, where the input(s) produce a particular output.

A	B	$A \oplus B$
False	False	**False**
False	True	**True**
True	False	**True**
True	True	**False**

A	B	$A \Rightarrow B$
False	False	**True**
False	True	**True**
True	False	**False**
True	True	**True**

A	B	$A \Leftrightarrow B$
False	False	**True**
False	True	**False**
True	False	**False**
True	True	**True**

Truth tables

- The revision friendly version ☺

A	B	¬A	A ∧ B	A ∨ B	A ⊕ B	A ➡ B	A ⬌ B
False	False	True	False	False	False	True	True
False	True	True	False	True	True	True	False
True	False	False	False	True	True	False	False
True	True	False	True	True	False	True	True

Some intuition on implication

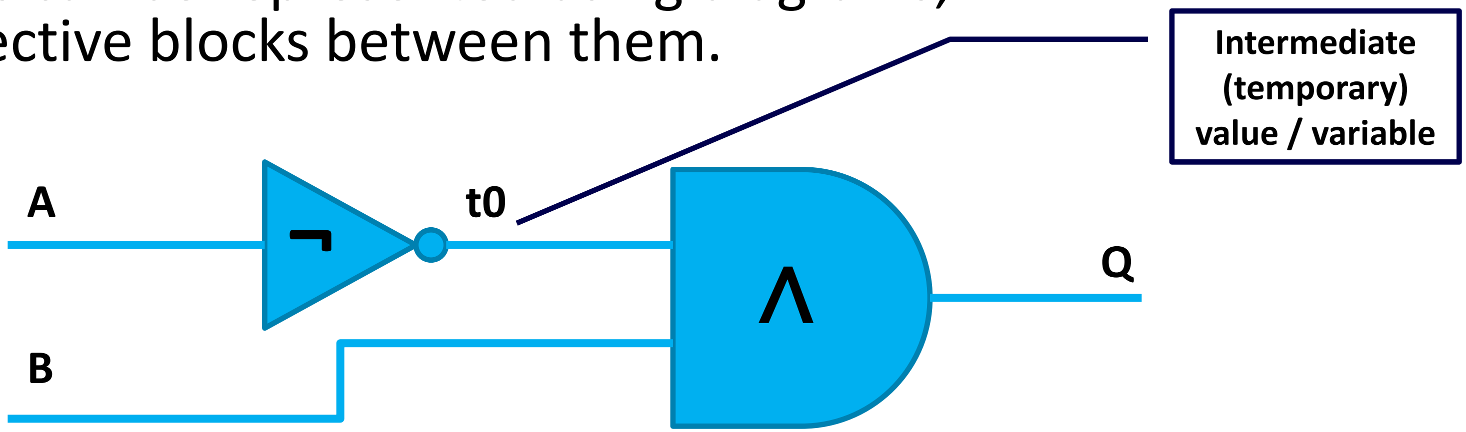
A	B	A ➡ B
False	False	**True**
False	True	**True**
True	False	**False**
True	True	**True**

- Let A be “You pass the exam.”
- Let B be “I invite you for a pizza.”
- Then *A implies B* means “If you pass the exam then I invite you for a pizza.”

- A ➡ B is true in three cases:
  - You pass the exam and I take you out for a pizza.
  - You don’t pass the exam and I don’t invite you for a pizza.
  - You don’t pass the exam and I do invite you for a pizza.
- But, if you’ve passed the exam but I don’t invite you for pizza, then the implication is clearly false.

Diagrammatically

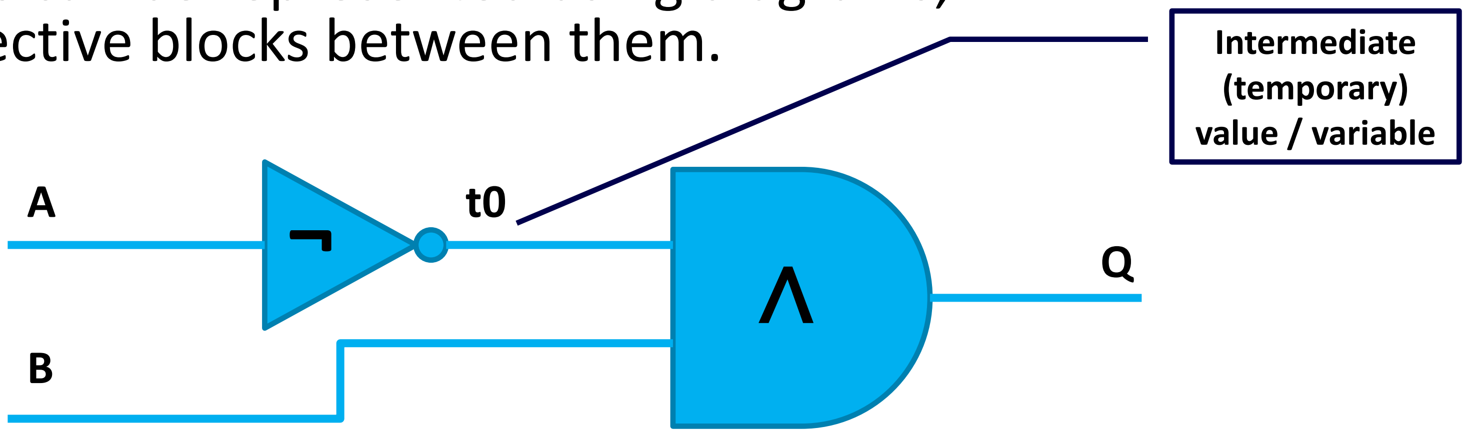
- Statements can be represented using diagrams, with connective blocks between them.



A	B	t0	Result (Q)
False	False	True	?
False	True	True	?
True	False	False	?
True	True	False	?

Diagrammatically

- Statements can be represented using diagrams, with connective blocks between them.



A	B	t0	Result (Q)
False	False	True	False
False	True	True	True
True	False	False	False
True	True	False	False

Boolean Algebra

- Represent **false as 0** and **true as 1**.
  - Binary digits
- Basic operators **¬, ∧, ∨** to connect statements, forming larger expressions.
- Secondary operators: **⊕ , ➡ , ⬌**
  - can be built from the basic operators (Try this out!)
- Observe a set of **axioms** (rules).
  - These help us manipulate expressions while preserving their truth value.

Axioms in Boolean algebra

Is logically equivalent to

Rule	Axioms
Commutativity	$x \land y \equiv y \land x$ $x \lor y \equiv y \lor x$
Associativity	$(X \lor y) \lor z \equiv x \lor (y \lor z)$ $(X \land y) \land z \equiv x \land (y \land z)$
Distributivity	$X \land (y \lor z) \equiv (x \land y) \lor (x \land z)$ $X \lor (y \land z) \equiv (x \lor y) \land (x \lor z)$

Axioms in Boolean algebra

- Some rules help us avoid evaluating parts, because we can know the answer regardless of the values of the variables.

Rule	Axioms
Identity	$x \wedge 1 \equiv x$
Null	$x \wedge 0 \equiv 0$
Idempotence	$x \wedge x \equiv x$
Inverse	$x \wedge \neg x \equiv 0$

- **Duality**: Swap 0s and 1s, conjunction and disjunction. Equivalence is preserved.

Axioms in Boolean algebra

- Some rules help us avoid evaluating parts, because we can know the answer regardless of the values of the variables.

Rule	Axioms
Identity	$x \wedge 1 \equiv x$ and $x \vee 0 \equiv x$
Null	$x \wedge 0 \equiv 0$ and $x \vee 1 \equiv 1$
Idempotence	$x \wedge x \equiv x$ and $x \vee x \equiv x$
Inverse	$x \wedge \neg x \equiv 0$ and $x \vee \neg x \equiv 1$

- **Duality**: Swap 0s and 1s, conjunction and disjunction. Equivalence is preserved.
  - (It is sufficient to remember only one version of axioms.)

Axioms in Boolean algebra

- Some help us avoid evaluating parts, because we can know the answer regardless of the values of the variables, others help simplify/transform expressions.

Rule	Axioms
Absorption	$x \wedge (x \vee y) \equiv x$ $x \vee (x \wedge y) \equiv x$
De Morgan	$\neg(x \wedge y) \equiv \neg x \vee \neg y$ $\neg(x \vee y) \equiv \neg x \wedge \neg y$
Equivalence	$(x \Leftrightarrow y) \equiv (x \Rightarrow y) \wedge (y \Rightarrow x)$
Implication	$x \Rightarrow y \equiv \neg x \vee y$
Double Negation	$\neg\neg x \equiv x$

Axioms in Boolean algebra

- Some help us avoid evaluating parts, because we can know the answer regardless of the values of the variables, others help simplify/transform expressions.

$x \oplus y \equiv$???

Rule	Axioms
Absorption	$x \wedge (x \vee y) \equiv x$ $x \vee (x \wedge y) \equiv x$
De Morgan	$\neg(x \wedge y) \equiv \neg x \vee \neg y$ $\neg(x \vee y) \equiv \neg x \wedge \neg y$
Equivalence	$(x \Leftrightarrow y) \equiv (x \Rightarrow y) \wedge (y \Rightarrow x)$
Implication	$x \Rightarrow y \equiv \neg x \vee y$
Double Negation	$\neg\neg x \equiv x$

Axioms in Boolean algebra

- Some help us avoid evaluating parts, because we can know the answer regardless of the values of the variables, others help simplify/transform expressions.

$x \oplus y \equiv (x \vee y) \wedge \neg(x \wedge y)$

Rule	Axioms
Absorption	$x \wedge (x \vee y) \equiv x$ $x \vee (x \wedge y) \equiv x$
De Morgan	$\neg(x \wedge y) \equiv \neg x \vee \neg y$ $\neg(x \vee y) \equiv \neg x \wedge \neg y$
Equivalence	$(x \Leftrightarrow y) \equiv (x \Rightarrow y) \wedge (y \Rightarrow x)$
Implication	$x \Rightarrow y \equiv \neg x \vee y$
Double Negation	$\neg\neg x \equiv x$

Complements

- For an expression, e, its **complement (or negation)**, ¬e, can be formed by:
  - Complementing all variables, e.g. from x to ¬x
  - Complementing all constants, e.g. from 0 to ¬ 0 = 1
  - Interchanging conjunction and disjunction
- Let's try it!

$$e = x \land y \land z$$
$$\neg e = ?$$

- Note that complement and dual are two different concepts.

Exercise

- Is $((p \wedge q) \vee r)$ logically equivalent to $(p \wedge (q \vee r))$?

Exercise

- Is ((p∧q)∨r) logically equivalent to (p∧(q∨r))?

p	q	r	p∧q	(p∧q) ∨r	q∨r	p∧(q ∨r)
0	0	0				
0	0	1				
0	1	0				
0	1	1				
1	0	0				
1	0	1				
1	1	0				
1	1	1				

Exercise

- Is ((p∧q)∨r) logically equivalent to (p∧(q∨r))?

p	q	r	p∧q	(p∧q) ∨r	q∨r	p∧(q ∨r)
0	0	0	0	0	0	0
0	0	1	0	1	1	0
0	1	0	0	0	1	0
0	1	1	0	1	1	0
1	0	0	0	0	0	0
1	0	1	0	1	1	1
1	1	0	1	1	1	1
1	1	1	1	1	1	1

Exercise

- Is ((p∧q)∨r) logically equivalent to (p∧(q∨r))?

p	q	r	p∧q	(p∧q) ∨r	q∨r	p∧(q ∨r)
0	0	0	0	0	0	0
0	0	1	0	1	1	0
0	1	0	0	0	1	0
0	1	1	0	1	1	0
1	0	0	0	0	0	0
1	0	1	0	1	1	1
1	1	0	1	1	1	1
1	1	1	1	1	1	1

Exercise

- Is $((p \wedge q) \vee r)$ logically equivalent to $(p \wedge (q \vee r))$?

p	q	r	p∧q	(p∧q) ∨r	q∨r	p∧(q ∨r)
0	0	0	0	0	0	0
0	0	1	0	1	1	0
0	1	0	0	0	1	0
0	1	1	0	1	1	0
1	0	0	0	0	0	0
1	0	1	0	1	1	1
1	1	0	1	1	1	1
1	1	1	1	1	1	1

Normal forms, DNF and CNF

- Disjunctive Normal Form (Sum of Products)
  - Groups of conjunctions (products) connected together with disjunctions (sums)

$$\underbrace{(a \wedge \neg b \wedge c)}_{\text{Minterm}} \vee \underbrace{(\neg d \wedge e)}_{\text{Minterm}}$$

- Conjunctive Normal Form (Product of Sums)
  - Groups of disjunctions connected with conjunctions

$$\underbrace{(a \vee \neg b \vee c)}_{\text{Maxterm}} \wedge \underbrace{(\neg d \vee e)}_{\text{Maxterm}}$$

Normal forms, DNF and CNF

- Disjunctive Normal Form (Sum of Products)
  - Groups of conjunctions (products) connected together with disjunctions (sums)

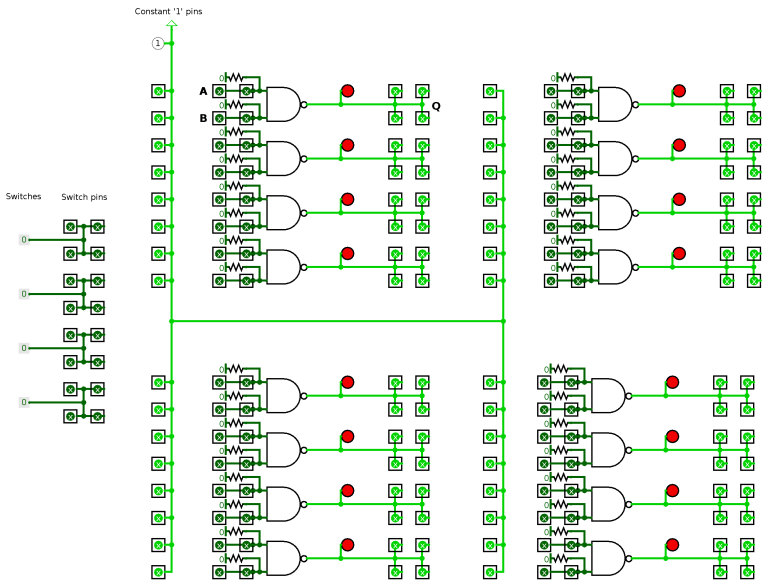
$$\underbrace{(a\ .\ \neg b\ .\ c)}_{\text{Minterm}}\ +\ \underbrace{(\neg d\ .\ e)}_{\text{Minterm}}$$

- Conjunctive Normal Form (Product of Sums)
  - Groups of disjunctions connected with conjunctions

$$\underbrace{(a + \neg b + c)}_{\text{Maxterm}}\ .\ \underbrace{(\neg d + e)}_{\text{Maxterm}}$$

Summary

- Propositional logic
  - Conjunction, disjunction, negation, implication, equivalence, etc…
  - Truth tables
  - Circuit diagrams
- Boolean algebra
  - Based on propositional logic
  - Axioms
  - Normal forms
- Lots of maths
  - But now we can start to build digital systems!

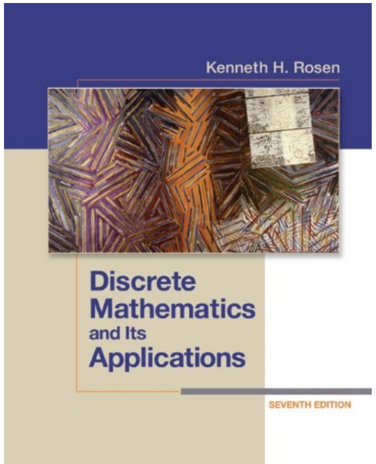


Further reading

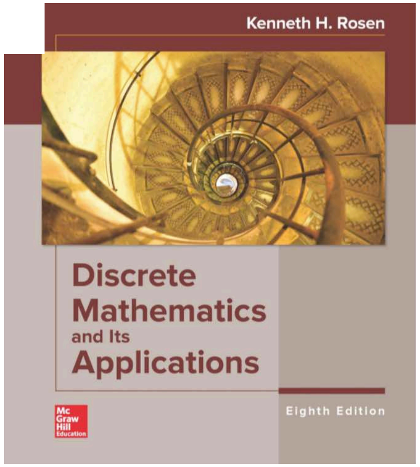
A good textbook on Discrete Mathematics is the one by Rosen:

Kenneth H. Rosen

Discrete Mathematics and Its Applications (7th or 8th Edition)



- Read the parts on **Logic and Boolean Algebra** to advance your understanding of the material covered in this lecture.
- Solve the exercises in the book to practice problem solving.



Note: There are many books on Logic and Boolean Algebra. The best level for you would be an Introduction to Logic or an Introduction to Boolean Algebra.

In this lecture

Foundations	• Data representation, **logic, Boolean algebra.**
Building blocks	• Transistors, transistor based logic, simple devices, storage.
Modules	• Memory, simple controllers, FSMs, processors and execution.
Programming	• Machine code, assembly, high-level languages, compilers.
Wrap-up	• Operating systems, energy aware computing.

In the next lecture

Foundations	• Data representation, logic, Boolean algebra.
Building blocks	• **Transistors, transistor based logic**, simple devices, storage.
Modules	• Memory, simple controllers, FSMs, processors and execution.
Programming	• Machine code, assembly, high-level languages, compilers.
Wrap-up	• Operating systems, energy aware computing.