

Fundamentals of Computing

Michael Zakharyashev

Department of Computer Science and Information Systems

Birkbeck, University of London

- room 161, Main Building
- email: michael@dc.s.bbk.ac.uk
- homepage: <http://www.dcs.bbk.ac.uk/~michael>
- FoC module site: <http://www.dcs.bbk.ac.uk/~michael/foc/foc.html>

Reading list

- ✓ **These slides** at <http://www.dcs.bbk.ac.uk/~michael/foc/foc.html>
- **M.M. Mano and C.R. Kime.** *Logic and Computer Design Fundamentals*. 4th ed. Pearson, 2008.
 - **S.S. Epp.** *Discrete Mathematics with Applications*. 4th ed. Brooks/Cole, 2011.
 - **D.A. Patterson and J.L. Hennessy.** *Computer Organization and Design: The Hardware/Software Interface*. Fourth Edition. Morgan Kaufmann, 2008.
 - **E. Kinber and C. Smith.** *Theory of Computing*. A gentle introduction, Prentice Hall, 2001
 - **D. Cohen.** *Introduction to Computer Theory*. John Wiley & Sons, 1997.
- ✓ **Video tutorials** <http://www.bbk.ac.uk/business/about-us/film-unit/MathsTutorials>
- ✓ **Web resources** (see <http://www.dcs.bbk.ac.uk/~michael/foc/foc.html> for more links)
- http://en.wikipedia.org/wiki/Logic_gate
 - http://en.wikipedia.org/wiki/Naive_set_theory
 - http://en.wikipedia.org/wiki/Automata_theory
 - <http://plato.stanford.edu/entries/turing-machine>

About the module

- **Tutorials** with Dr Stanislav Kikot — every 2nd Tuesday, starting on 11 October, Main Building B18, 18:00–21:00
- **Coursework** to be issued on Friday, 2 December 2016
submission deadline: **Friday, 20 January 2017**
late submission ($\leq 50\%$): Friday, 3 February 2017
- **Examination** in May 2017
- **Exercises** to prepare for the coursework and exam will be available at

<http://www.dcs.bbk.ac.uk/~michael/foc/foc.html>

- **Final mark = CW (20%) + exam (80%)**

Syllabus: autumn term

Part I: Computer logic and arithmetic

- How are numbers represented in computers?
- How are negative numbers represented?
- What is the largest number that can be represented in a computer word?
- How does hardware really add, subtract, multiply, or divide numbers?

M.M. Mano and C.R. Kime. *Logic and Computer Design Fundamentals*. 4th Ed. Pearson, 2008
also: S.S. Epp. *Discrete Mathematics with Applications*, Sections 2.4–2.5

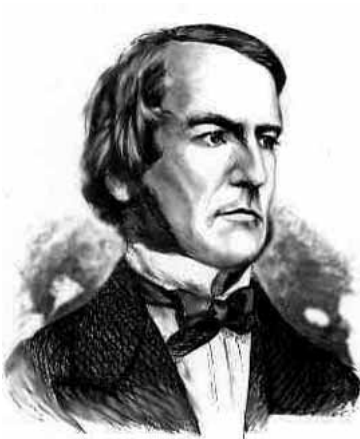
Part II: Models of computation

- What is a computation or an algorithm?
- What can and what cannot be computed?
- What can be computed with limited memory?
- What makes some problems computationally hard and others easy?

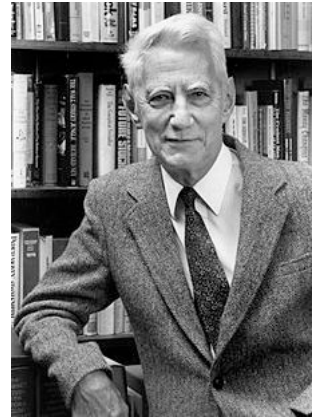
E. Kinber and C. Smith. *Theory of Computing. A gentle introduction*. Prentice Hall, 2001
also: S.S. Epp. *Discrete Mathematics with Applications*, Chapter 12

Part I:

Computer logic and arithmetic



George Boole
(1815–1864)



Claude Shannon
(1916–2001)

Why are computers binary?



- Simple; easy to build.
- Unambiguous signals (hence noise immunity).
- Flawless copies can be made.
- 0 and 1 are enough to encode anything we need.
- Binary arithmetic is **more efficient** than decimal.
- ... (there also exist ternary computers)
- **George Boole** invented Boolean algebra, an algebra of two values, which is the basis of all modern computer arithmetic
- **Claude Shannon** showed how electrical application of Boolean algebra could construct and resolve any logical, numerical relationship (also founded information theory)

Logic

Logic is the formal systematic study of the principles of **valid inference** and **correct reasoning**

Are the following inferences valid?

- If it is raining then I take an umbrella
- It is raining
- Therefore I take an umbrella

- If it is raining then I take an umbrella
- It is not raining
- Therefore I don't take an umbrella

What does it mean for one sentence to **follow logically** from certain others?

The sentences above are **declarative sentences**, or **propositions**,
which one can, in principle, argue as being **true** or **false**

Boolean algebra (or **Boolean logic**) is a logical calculus of truth values,
developed by George Boole in the 1840s

Elements of Boolean logic

Basic assumption: every **proposition** is either **true** or **false** (but not both)

Examples:

(A) George W. Bush is the current president of the United States of America.

(B) Barack Obama is the current president of the United States of America.

(C) Extraterrestrial life does not exist.

NB: Questions/exclamations, paradoxical statements like 'this proposition is false' are **not** propositions.

Propositional connectives:

- 'not' (negation) denoted by \neg (! in C++/Java) Is $\neg A$ true?
 - 'and' (conjunction) denoted by \wedge (&& in C++/Java) Is $A \wedge B$ true?
 - 'or' (disjunction) denoted by \vee (|| in C++/Java) Is $A \vee B$ true?
 - 'if ... then ...' (implication) denoted by \rightarrow Is $A \rightarrow C$ true?
- ? Are there any other propositional connectives?

Complex propositions (formulas): $(\neg A) \rightarrow (B \vee C)$, $((\neg B) \wedge (\neg \neg C)) \rightarrow \neg A$, etc.

Semantics: truth-tables

Notation: 1 for 'true', 0 for 'false'

A, B, C, A_1, B_1, \dots for atomic (in a given context) propositions
a.k.a. **propositional variables**

$A \vee B, (\neg A) \rightarrow (A_1 \wedge \neg B_2), \dots$ for complex propositions
a.k.a. **propositional** or **Boolean formulas**

Truth-tables for $\wedge, \vee, \rightarrow$ and \neg :

A	B	$A \wedge B$	$A \vee B$	$A \rightarrow B$	$\neg A$
0	0	0	0	1	1
0	1	0	1	1	1
1	0	0	1	0	0
1	1	1	1	1	0

so the proposition 'if the Moon is made of green cheese, then $2 \times 2 = 7$ ' is true

Explaining 'implication'

One possible 'explanation' of the truth-table for the 'implication' \rightarrow :

The following statement is true for **every** natural number n :

If n is divisible by 4, then n is divisible by 2.

So the following instances of this general statement must be true as well:

If 8 is divisible by 4, then 8 is divisible by 2 $(1 \rightarrow 1) = 1$

If 7 is divisible by 4, then 7 is divisible by 2 $(0 \rightarrow 0) = 1$

If 2 is divisible by 4, then 2 is divisible by 2 $(0 \rightarrow 1) = 1$

And of course, 'if 8 is divisible by 4, then 7 is divisible by 2' is false $(1 \rightarrow 0) = 0$

This interpretation of logical connectives is a **mathematical abstraction**. Under such abstractions, meaningless sentences may become sensible, and the other way round.

There are different interpretations of logic connectives,
e.g., with three or more truth-values.

Truth-tables for Boolean formulas

A	B	$(\neg A) \vee B$			
0	0	1	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	1	0	1	1	1

Note that this truth-table (the column under the main connective \vee)
is the same as the truth-table for $A \rightarrow B$

So we can say that $(\neg A) \vee B$ is **equivalent to** $A \rightarrow B$

Exercise Brown, Jones and Smith are suspected of income tax evasion.

They testify under oath as follows:

- Brown: Jones is guilty and Smith is innocent.
- Jones: If Brown is guilty, then so is Smith.
- Smith: I'm innocent, but at least one of the others is guilty.

Assuming everyone's testimony is true, who is innocent and who is guilty?

Solution

BG stands for 'Brown is guilty', JG for 'Jones is guilty' and SG for 'Smith is guilty'

– Brown says: $JG \wedge \neg SG$ Jones says: $BG \rightarrow SG$ Smith says: $\neg SG \wedge (BG \vee JG)$

BG	JG	SG	$JG \wedge \neg SG$	$BG \rightarrow SG$	$\neg SG \wedge (BG \vee JG)$
0	0	0	0	1	0
0	0	1	0	1	0
0	1	0	1	1	1
0	1	1	0	1	0
1	0	0	0	0	1
1	0	1	0	1	0
1	1	0	1	0	1
1	1	1	0	1	0

The only case where all the statements are true: $BG = 0$, $JG = 1$ and $SG = 0$

This problem can be solved in a more direct way. From the first statement we can infer that Jones is guilty and Smith is innocent. From the second statement, it follows that if Smith is not guilty then Brown is not guilty. Therefore we can infer that Brown is innocent.

Boolean logic in computers

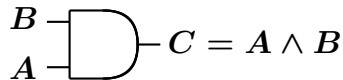
In the world of computers, 0 and 1 are known as **bits**

- 0 is represented by the lower voltage level (LOW), say, 0V – 0.1V
- 1 is represented by the higher voltage level (HIGH), say, 0.9V – 1.1V

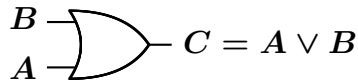
All computer circuits consist of hundreds of millions of interconnected primitive elements called **gates**, which correspond to the basic logic connectives:

Basic logic gates:

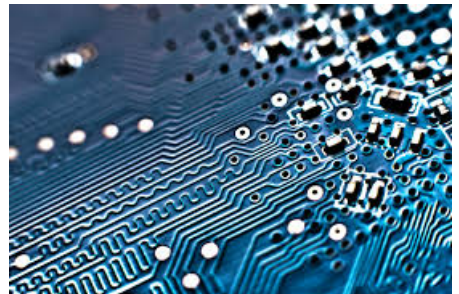
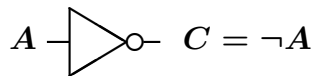
AND gate



OR gate

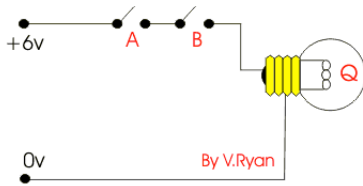


NOT gate



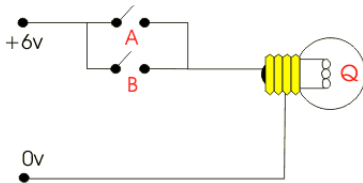
Examples

AND GATE



The AND gate has two inputs, switch *A* and switch *B*. The bulb *Q* will only light if both switches are closed. This will allow current to flow through the bulb, illuminating the filament

OR GATE



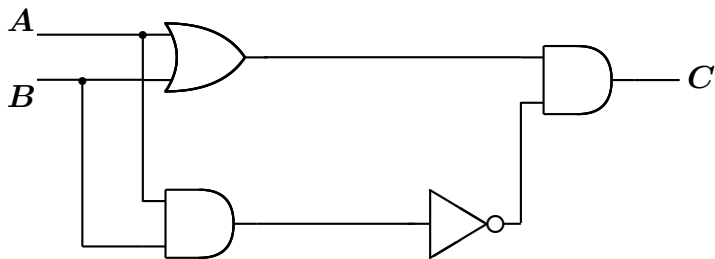
The OR gate has two inputs, switch *A* and switch *B*. The bulb *Q* will light if either switch *A* or *B* are closed. This will allow current to flow through the bulb, illuminating the filament

Since the 1990s, most logic gates are made of transistors

(semiconductor devices used to amplify and switch electronic signals)

Transistors are so small that hundreds of thousands fit on one processing chip on a computer motherboard

Example: Boolean circuit



What does this circuit do?

- Represent the circuit as a **Boolean equation**

$$C = (A \vee B) \wedge \neg(A \wedge B)$$

- Construct the truth-table

A	B	$(A \vee B) \wedge \neg(A \wedge B)$						
0	0	0	0	0	0	1	0	0
0	1	0	1	1	1	1	0	1
1	0	1	1	0	1	1	1	0
1	1	1	1	1	0	0	1	1

The circuit, the truth-table and the formula $(A \vee B) \wedge \neg(A \wedge B)$ represent the **Boolean function** known as **exclusive or** and denoted by **XOR**, or $A \oplus B$

Boolean functions of one argument

A	0
0	0
1	0

— **constant function 0** (always returns 0 and doesn't depend on A)

draw a Boolean circuit for this function

A	1
0	1
1	1

— **constant function 1** (always returns 1 and doesn't depend on A)

draw a Boolean circuit for this function

A	A
0	0
1	1

— **identical function 0** (always returns the input A)

draw a Boolean circuit for this function


A	$\neg A$
0	1
1	0

— **function NOT** or \neg (inverts the input A)


draw a Boolean circuit for this function


Boolean functions of two arguments

A	B	$A \wedge B$	$A \vee B$	$A \rightarrow B$	$A \oplus B$	$A \leftrightarrow B$	$A B$	$A \downarrow B$
0	0	0	0	1	0	1	1	1
0	1	0	1	1	1	0	1	0
1	0	0	1	0	1	0	1	0
1	1	1	1	1	0	1	0	0

XOR gate  $C = A \oplus B$

$A \leftrightarrow B$ — **equivalence** (if, and only if), equivalent to $(A \rightarrow B) \wedge (B \rightarrow A)$

NAND gate  $C = A | B = \neg(A \wedge B)$ — **Scheffer stroke**

NOR gate  $C = A \downarrow B = \neg(A \vee B)$ — **Pierce arrow**

- What functions are missing here?
- What is the number of Boolean functions of two arguments?

Important questions

There are very many Boolean functions: 2^{2^n} distinct functions of n variables
For example, there are $2^{2^5} = 4,294,967,296$ functions with 5 inputs

We don't know *a priori* which of them are required in computer architecture

- Is it possible to fix some, relatively simple set(s) of Boolean functions (gates), using which one can built **all** other Boolean functions?

We have already seen that the same Boolean functions can be realised in different ways using different gates.

Of course we need smallest possible circuits (formulas). . .

- How to build 'optimal' Boolean circuits (formulas)?
- How to simplify Boolean circuits (formulas)?
- What basic gates to choose?

We consider some aspects of these problems.

Example: the majority function

Suppose we want to realise, using only the AND, OR and NOT gates,
the **majority function** $\mu(A, B, C)$ whose output takes the same value
as the **majority** of inputs:

A	B	C	$\mu(A, B, C)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Each binary 3-tuple in the table can be represented
as a conjunction. E.g.,

0 1 1 is represented by $\neg A \wedge B \wedge C$

It equals 1 if, and only if, $A = 0$, $B = 1$, $C = 1$

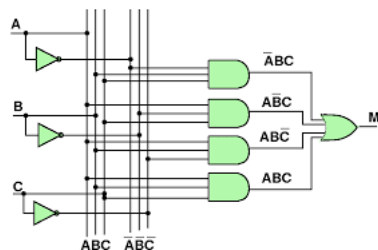
The function $\mu(A, B, C)$ can then be realised as a
disjunction of the conjunctions that correspond to the
rows where $\mu(A, B, C) = 1$ i.e.,

$$(\neg A \wedge B \wedge C) \vee (A \wedge \neg B \wedge C) \vee (A \wedge B \wedge \neg C) \vee (A \wedge B \wedge C)$$

Example: the majority function (cont.)

- Use the formula above to construct a Boolean circuit for $\mu(A, B, C)$

- Can you simplify it?



Consider, for instance, the formula

$$(A \wedge B) \vee (B \wedge C) \vee (A \wedge C)$$

- Does it define the same function? (Construct the truth-table)
- Is the corresponding circuit simpler?
- Can you simplify it?

what about the formula $(A \wedge (B \vee C)) \vee (B \wedge C)$?

Universal sets of Boolean functions

The method used on page 19 can be used to represent **any Boolean function** by means of a formula with the connectives \neg , \wedge and \vee

(if there is no 1 among the function values then this function is 0,

which can be represented as $A \wedge (\neg A)$)

We say that $\{\neg, \wedge, \vee\}$ is a **universal** set of Boolean connectives/functions

- Are there other universal sets of Boolean formulas?
- Can simplifications like those on page 20 be done in a systematic way?

Boolean formulas φ , ψ are called **equivalent** if their truth-tables are identical.
In this case we write $\varphi \equiv \psi$.

As equivalent formulas φ and ψ determine the same Boolean function,
we can use either of them to construct Boolean circuits

Useful equivalences

$$\neg(\varphi \wedge \psi) \equiv \neg\varphi \vee \neg\psi$$
$$\neg(\varphi \vee \psi) \equiv \neg\varphi \wedge \neg\psi$$

(De Morgan laws)

(φ and ψ are arbitrary
Boolean formulas)

$$\neg\neg\varphi \equiv \varphi \quad (\text{the law of double negation})$$

$$\neg\varphi \vee \varphi \equiv 1 \quad (\text{the law of the excluded middle, 'to be or not to be'})$$

$$\neg\varphi \wedge \varphi \equiv 0 \quad (\text{the law of contradiction})$$

$$\varphi \wedge (\psi \vee \chi) \equiv (\varphi \wedge \psi) \vee (\varphi \wedge \chi) \quad (\text{distributivity of } \wedge \text{ over } \vee)$$

$$\varphi \vee (\psi \wedge \chi) \equiv (\varphi \vee \psi) \wedge (\varphi \vee \chi) \quad (\text{distributivity of } \vee \text{ over } \wedge)$$

$$\varphi \wedge 1 \equiv \varphi, \quad \varphi \wedge 0 \equiv 0, \quad \varphi \vee 1 \equiv 1, \quad \varphi \vee 0 \equiv \varphi, \quad \varphi \wedge \varphi \equiv \varphi, \quad \varphi \vee \varphi \equiv \varphi$$

It follows, for instance, that $\varphi \vee \psi \equiv \neg((\neg\varphi) \wedge (\neg\psi))$

$$\varphi \wedge \psi \equiv \neg((\neg\varphi) \vee (\neg\psi))$$

Thus, we can express \vee by means of \neg and \wedge ;

likewise, \wedge can be expressed by means of \neg and \vee

So both $\{\neg, \wedge\}$ and $\{\neg, \vee\}$ are universal (e.g., $\varphi \rightarrow \psi \equiv \neg\varphi \vee \psi$)

Example

$$\begin{aligned} A \vee (A \wedge B) &\equiv (A \wedge 1) \vee (A \wedge B) \\ &\equiv (A \wedge (B \vee \neg B)) \vee (A \wedge B) \\ &\equiv (A \wedge B) \vee (A \wedge \neg B) \vee (A \wedge B) \\ &\equiv (A \wedge B) \vee (A \wedge B) \vee (A \wedge \neg B) \\ &\equiv (A \wedge B) \vee (A \wedge \neg B) \\ &\equiv A \wedge (B \vee \neg B) \\ &\equiv A \wedge 1 \\ &\equiv A \end{aligned}$$

Thus,

$$A \vee (A \wedge B) \equiv A$$

Universality of NAND

To prove that $|$ (or NAND) is universal, it is enough to show using NAND
we can express NOT and AND:

- $\neg A \equiv (A | A) \equiv \neg(A \wedge A) \equiv \neg A$, because $A \wedge A \equiv A$
- $A \wedge B \equiv (A | B) | (A | B)$ why?

Exercise 1: Show that NOR is also universal

Exercise 2: A **2-to-1 multiplexer** has three inputs, say A_0 , A_1 and S ;
the output is A_0 if $S = 0$ and A_1 if $S = 1$.
Design a Boolean circuit for the 2-to-1 multiplexer.

In general, a multiplexer selects one of many input signals
and outputs that into a single line