Fundamentals of Computing

Michael Zakharyaschev

Department of Computer Science and Information Systems
Birkbeck University of London

- room 161, Main Building, Malet Street
- email: michael@dcs.bbk.ac.uk
- homepage: http://www.dcs.bbk.ac.uk/~michael
- FoC module site: http://www.dcs.bbk.ac.uk/~michael/foc/foc.html
Module structure

- **Lectures** on Wednesdays in UCL Bedford Way G03, 18:00 – 19:30, part time
  Thursdays in Malet Street 403, 15:30 – 17:00, full time

- **Tutorials** with Dr Stanislav Kikot 18:00–21:00:
  - for **MSc DS**: every 2nd Thursday in UCL Christopher Ingold XLG1 Chemistry LT
  - for **all others**: every 2nd Tuesday in UCL Medawar G01 Lankester LT

- **Coursework** to be issued on Friday, 8 December 2017
  submission deadline: **Friday, 19 January 2018**
  late submission (≤ 50%): Friday, 2 February 2018

- **Examination** in May 2018

- **Exercises** to prepare for the coursework and exam will be available at
  [http://www.dcs.bbk.ac.uk/~michael/foc/foc.html](http://www.dcs.bbk.ac.uk/~michael/foc/foc.html)

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

✓ These slides at http://www.dcs.bbk.ac.uk/~michael/foc/foc.html


✓ 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
Learning the material

- You are not supposed to understand everything during the lectures. The lecture material is prepared to guide you through the chosen topics.

- The only way to master maths material is by doing it yourself.

- In the tutorials, you are expected to work through the provided exercises yourself. The exercises are carefully chosen to help you understand the introduced concepts and techniques. It is often helpful to go through the lecture slides once more in your attempt of solving the exercises. The tutor will help you if you have any difficulties, and will also discuss your solutions (but he won’t solve the exercises instead of you). Model solutions will be published right after the tutorial.

- More exercises are provided on the module site. If you think you need even more, please consult the recommended textbooks. And finally: please use the web (there is an extensive amount of online material, including videos, on the covered topics).
What is this module for?

- This module aims to bring the mathematical skills of the class to the same level, and to provide a common mathematical background for many of your other modules.

- The module also aims to develop your problem solving skills and your ability to express yourself in a more precise manner.

- If you had Mathematics A-level, you may find that some parts of the course cover material you already know to some extent (and you might be bored:-)
  But watch out: there will be something new any time

- If you did not have Mathematics A-level: \textbf{don’t panic}
  You are not expected to know topics that you have never seen before, everything is developed from scratch.
  And much of Maths A-level is not relevant to this module anyway (for example, we won’t use differentiation or trigonometry).
What is used?

- integer numbers
  - negative: \( \ldots, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, \ldots \)
  - positive: \( \ldots, 1, 2, 3, 4, 5, \ldots \)

- odd numbers, even numbers, divisibility
- operations on integers: +, −, ×, /
- exponentiation: \( 5^6, (-2)^0, 6^3, \ldots \)
- comparing integers with =, >, <, ≥, ≤
- denoting integers with letters (\( n, m, x, y, \ldots \))

\[ \sim \quad \text{understanding statements like } \quad \text{"}2^n > n^2 \text{ for every integer } n\text{".} \]

- a bit of algebra: \( (a + b)^2 = a^2 + 2ab + b^2, \quad a^2 - b^2 = (a - b)(a + b), \ldots \)
- working with fractions

Please refresh your knowledge of these!
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?

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?

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, ‘The Laws of Thought’ 1854 which is the basis for 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?

1. If it is raining then I take an umbrella
2. It is raining
3. Therefore I take an umbrella

1. If it is raining then I take an umbrella
2. It is not raining
3. 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) Donald Trump 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 \( \land \) (\(\&\&\) in C++/Java)  
  Is \( A \land B \) true?
- ‘or’ (disjunction) denoted by \( \lor \) (\(\|\|\) in C++/Java)  
  Is \( A \lor B \) true?
- ‘if . . . then . . . ’ (implication) denoted by \( \rightarrow \)  
  Is \( A \rightarrow C \) true?

? Are there any other propositional connectives?

Complex propositions (formulas):  
\( (\neg A) \rightarrow (B \lor C) \),  
\( ((\neg B) \land (\neg C)) \rightarrow \neg A \), etc.
Semantics: truth-tables

Notation: 1 for ‘true’, 0 for ‘false’

for atomic (in a given context) propositions a.k.a. propositional variables

for complex propositions a.k.a. propositional or Boolean formulas

Truth-tables for $\land$, $\lor$, $\rightarrow$ and $\neg$:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A $\land$ B</th>
<th>A $\lor$ B</th>
<th>A $\rightarrow$ B</th>
<th>$\neg$A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

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’ →:

The following statement is true for every natural number \( n \):

\[
\text{If } n \text{ is divisible by } 4, \text{ then } n \text{ is divisible by } 2.
\]

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

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
<th>$(\neg A) \lor B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1 0 1 0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1 0 1 1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0 1 0 0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0 1 1 1</td>
</tr>
</tbody>
</table>

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

So we can say that $(\neg A) \lor B$ 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: \(JG \land \neg SG\)
- Jones says: \(BG \rightarrow SG\)
- Smith says: \(\neg SG \land (BG \lor JG)\)

<table>
<thead>
<tr>
<th>(BG)</th>
<th>(JG)</th>
<th>(SG)</th>
<th>(JG \land \neg SG)</th>
<th>(BG \rightarrow SG)</th>
<th>(\neg SG \land (BG \lor JG))</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

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.
Formalising English sentences

Exercise: Translate the following English sentences to propositional logic:

(1) If I am not playing tennis, I am watching tennis.
(2) If I am not watching tennis, I am reading about tennis.
(3) I cannot do more than one of these activities at the same time.

Solution: First we choose our propositional variables.

(NB: they must denote propositions! say $T$: ‘playing tennis’ is no good)

$T$: ‘I am playing tennis’
$W$: ‘I am watching tennis’
$R$: ‘I am reading about tennis’

Then we can formalise the above English sentences as follows:

(1) $\neg T \rightarrow W$
(2) $\neg W \rightarrow R$
(3) $\neg(T \land W) \land \neg(T \land R) \land \neg(W \land R)$

or $\neg T \land \neg W \land \neg R \lor \neg T \land W \land \neg R \lor \neg T \land \neg W \land R \lor \neg T \land \neg W \land \neg R$
Logically correct arguments in propositional logic

An argument is a sequence of propositions:

\[
\begin{align*}
  p_1 & \\
  p_2 & \\
  \vdots & \\
  p_n & \\
\end{align*}
\]

\text{n premises (aka assumptions)}

Therefore \text{one conclusion}

An argument is \text{logically correct} if

in every ‘situation’ that makes all the premises true, the conclusion is true as well

a situation = a row in the truth table for \(p_1, \ldots, p_n, q\)

= a possible assignment for the propositional variables in \(p_1, \ldots, p_n, q\)

\[
\begin{array}{c}
p_1 \\
p_2 \\
\vdots \\
p_n \\
q
\end{array}
\]

is logically correct iff the formula \((p_1 \land p_2 \land \cdots \land p_n) \rightarrow q\) is always true

in which case it is called a \text{tautology}
Logically correct argument: an example

If I am not playing tennis,
  I am watching tennis \( \neg T \rightarrow W \)
If I am not watching tennis,
  I am reading about tennis \( \neg W \rightarrow R \)
I cannot do more than one of
these activities at the same time \( \neg(T \land W) \land \neg(T \land R) \land \neg(W \land R) \)

Therefore, \( W \) conclusion

In every situation that makes all the premises true, the conclusion is true as well:

<table>
<thead>
<tr>
<th>( T )</th>
<th>( W )</th>
<th>( R )</th>
<th>( \neg T \rightarrow W )</th>
<th>( \neg W \rightarrow R )</th>
<th>( \neg(T \land W) \land \neg(T \land R) \land \neg(W \land R) )</th>
<th>( W )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Incorrect argument: a simple example

If you solve every exercise in the textbook, then you will get an A. \( S \rightarrow A \)

You did not solve every exercise in the textbook. \( \neg S \)

Therefore \( \neg A \)

You won’t get an A.

Incorrect argument: It is not the case that ‘in every situation that makes all the premises true, the conclusion is true as well.’

So, to show that an argument is incorrect, it is enough to find one situation where

- all premises are true,
- but the conclusion is false.

\[
\begin{array}{c|c|c|c|c}
S & A & S \rightarrow A & \neg S & \neg A \\
1 & 1 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 \\
0 & 1 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 \\
\end{array}
\]
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**

\[
B \quad C = A \land B
\]

**OR gate**

\[
B \quad C = A \lor B
\]

**NOT gate**

\[
A \quad C = \neg A
\]
Examples

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.

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**

  \[ C = (A \lor B) \land \neg (A \land B) \]

- Construct the truth-table

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>((A \lor B) \land \neg (A \land B))</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0 0 0 0 1 0 0 0 0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0 1 1 1 1 0 0 1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1 1 0 1 1 1 0 0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1 1 1 0 0 1 1 1</td>
</tr>
</tbody>
</table>

The circuit, the truth-table and the formula \((A \lor B) \land \neg (A \land B)\) represent the **Boolean function** known as **exclusive or** and denoted by **XOR**, or \(A \oplus B\).
**Boolean functions of one argument**

<table>
<thead>
<tr>
<th>$A$</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

- **constant function** 0 (always returns 0 and doesn’t depend on $A$)

  draw a Boolean circuit for this function

<table>
<thead>
<tr>
<th>$A$</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- **constant function** 1 (always returns 1 and doesn’t depend on $A$)

  draw a Boolean circuit for this function

<table>
<thead>
<tr>
<th>$A$</th>
<th>$A$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- **identical function** 0 (always returns the input $A$)

  draw a Boolean circuit for this function

<table>
<thead>
<tr>
<th>$A$</th>
<th>$\neg A$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

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

  draw a Boolean circuit for this function
### Boolean functions of two arguments

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>( A \land B )</th>
<th>( A \lor B )</th>
<th>( A \rightarrow B )</th>
<th>( A \oplus B )</th>
<th>( A \leftrightarrow B )</th>
<th>( A \mid B )</th>
<th>( A \downarrow B )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**XOR gate**

\[
\begin{array}{ccc}
B & \downarrow & C = A \oplus B \\
A & \rightarrow & \\
\end{array}
\]

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

**NAND gate**

\[
\begin{array}{ccc}
B & \neg & C = A \mid B = \neg (A \land B) \\
A & \neg & \\
\end{array}
\]

**Scheffer stroke**

**NOR gate**

\[
\begin{array}{ccc}
B & \neg & C = A \downarrow B = \neg (A \lor B) \\
A & \neg & \\
\end{array}
\]

**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 \textit{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 \textbf{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:

<table>
<thead>
<tr>
<th>( A )</th>
<th>( B )</th>
<th>( C )</th>
<th>( \mu(A, B, C) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

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

\[
\neg A \land B \land C
\]

is represented by

\[
\neg A \land B \land C
\]

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.,

\[
(\neg A \land B \land C) \lor (A \land \neg B \land C) \lor (A \land B \land \neg C) \lor (A \land B \land C)
\]
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

$$ (A \land B) \lor (B \land C) \lor (A \land C) $$

- Does it define the same function? (Construct the truth-table)

- Is the corresponding circuit simpler?

- Can you simplify it?

  what about the formula $$(A \land (B \lor C)) \lor (B \land C)$$?
Universal sets of Boolean functions

The method shown on page 26 can be used to represent any Boolean function by means of a formula with the connectives $\neg$, $\land$ and $\lor$.

If there is no 1 among the function values then this function is 0, which can be represented as $A \land (\neg A)$.

We say that $\{\neg, \land, \lor\}$ is a universal set of Boolean connectives/functions.

- Are there other universal sets of Boolean formulas?
- Can simplifications like those on page 27 be done in a systematic way?

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

As equivalent formulas $\varphi$ and $\psi$ determine the same Boolean function, we can use either of them to construct Boolean circuits.
Useful equivalences

\[ \neg(\varphi \land \psi) \equiv \neg\varphi \lor \neg\psi \quad \text{(De Morgan laws)} \]
\[ \neg(\varphi \lor \psi) \equiv \neg\varphi \land \neg\psi \]
\[ \neg\neg\varphi \equiv \varphi \quad \text{(the law of double negation)} \]
\[ \neg\varphi \lor \varphi \equiv 1 \quad \text{(the law of the excluded middle, ‘to be or not to be’)} \]
\[ \neg\varphi \land \varphi \equiv 0 \quad \text{(the law of contradiction)} \]
\[ \varphi \land (\psi \lor \chi) \equiv (\varphi \land \psi) \lor (\varphi \land \chi) \quad \text{(distributivity of \( \land \) over \( \lor \))} \]
\[ \varphi \lor (\psi \land \chi) \equiv (\varphi \lor \psi) \land (\varphi \lor \chi) \quad \text{(distributivity of \( \lor \) over \( \land \))} \]

\[ \varphi \land 1 \equiv \varphi, \quad \varphi \land 0 \equiv 0, \quad \varphi \lor 1 \equiv 1, \quad \varphi \lor 0 \equiv \varphi, \quad \varphi \land \varphi \equiv \varphi, \quad \varphi \lor \varphi \equiv \varphi \]

It follows, for instance, that
\[ \varphi \lor \psi \equiv \neg((\neg\varphi) \land (\neg\psi)) \]
\[ \varphi \land \psi \equiv \neg((\neg\varphi) \lor (\neg\psi)) \]

Thus, we can express \( \lor \) by means of \( \neg \) and \( \land \);
likewise, \( \land \) can be expressed by means of \( \neg \) and \( \lor \)

So both \{\neg, \land\} and \{\neg, \lor\} are universal (e.g., \( \varphi \to \psi \equiv \neg\varphi \lor \psi \))
How to show equivalence: method 1

\[ A \lor (A \land B) \equiv (A \land 1) \lor (A \land B) \]
\[ \equiv (A \land (B \lor \neg B)) \lor (A \land B) \]
\[ \equiv (A \land B) \lor (A \land \neg B) \lor (A \land B) \]
\[ \equiv (A \land B) \lor (A \land B) \lor (A \land \neg B) \]
\[ \equiv (A \land B) \lor (A \land \neg B) \]
\[ \equiv A \land (B \lor \neg B) \]
\[ \equiv A \land 1 \]
\[ \equiv A \]

Thus,

\[ A \lor (A \land B) \equiv A \]
How to show equivalence: method 2

Exercise 1: Show that \( P \oplus Q \equiv (P \lor Q) \land \neg(P \land Q) \)

Solution:

\[
\begin{array}{c|c|c}
P & Q & P \oplus Q \\
\hline
1 & 1 & 0 \\
1 & 0 & 1 \\
0 & 1 & 1 \\
0 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{c|c|c|c|c|c|c}
P & Q & P \lor Q & P \land Q & \neg(P \land Q) & (P \lor Q) \land \neg(P \land Q) \\
\hline
1 & 1 & 1 & 1 & 0 & 0 \\
1 & 0 & 1 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 \\
\end{array}
\]

Exercise 2: Show that \( P \leftrightarrow Q \equiv (P \land Q) \lor (\neg P \land \neg Q) \)

Solution:

\[
\begin{array}{c|c|c|c}
P & Q & P \leftrightarrow Q \\
\hline
1 & 1 & 1 \\
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1 \\
\end{array}
\]

\[
\begin{array}{c|c|c|c|c|c|c|c|c}
P & Q & P \land Q & \neg P & \neg Q & \neg P \land \neg Q & (P \land Q) \lor (\neg P \land \neg Q) \\
\hline
1 & 1 & 1 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 \\
\end{array}
\]
Universality of NAND

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

- \(\neg A \equiv (A \mid A) \equiv \neg(A \land A) \equiv \neg A\), because \(A \land A \equiv A\)
- \(A \land B \equiv (A \mid B) \mid (A \mid B)\) 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