Foundations of Computer Science Comp109

University of Liverpool
Boris Konev
konev@liverpool.ac.uk
http://www.csc.liv.ac.uk/~konev/COMP109

Introduction

Comp109 Foundations of Computer Science

Information

Lecturer

- Prof Boris Konev
- Office: 1.15 Ashton building
- Email: konev@liverpool.ac.uk
- Course web page: `http://www.csc.liv.ac.uk:/~konev/COMP109`

$\sim$30 lectures + 2 class tests + 11 tutorials

Module aims

- To introduce the notation, terminology, and techniques underpinning the discipline of Theoretical Computer Science.
- To provide the mathematical foundation necessary for understanding datatypes as they arise in Computer Science and for understanding computation.
- To introduce the basic proof techniques which are used for reasoning about data and computation.
- To introduce the basic mathematical tools needed for specifying requirements and programs

Module outcomes

At the end of this module students should be able to:

- Understand how a computer represents simple numeric data types; reason about simple data types using basic proof techniques;
- Interpret set theory notation, perform operations on sets, and reason about sets;
- Understand, manipulate and reason about unary relations, binary relations, and functions;
- Apply logic to represent mathematical statement and digital circuit, and to recognise, understand, and reason about formulas in propositional and predicate logic;
- Apply basic counting and enumeration methods as these arise in analysing permutations and combinations.

Assessment

- Exam: 80%
  - Multiple-choice test
- Continuous Assessment: 20%
  - Assessment 1. Covers Parts 1-4
    - Class test
    - Tutorial contribution
  - Assessment 2. Covers Parts 5-7
    - Class test
    - Tutorial contribution

Lectures

We will have three lectures per week.
Your personal timetable is on *Liverpool Life*.

- Read the slides before (and after) the lecture.
- Take notes. (University is a lot different from school.)
- I will write on the slides.
- Notes often make no/little sense

  PDFs will appear on
  `http://cgi.csc.liv.ac.uk/~konev/COMP109`
  - These notes are not a replacement for your own notes!
- Please study as you go along.

Tutorials

- The class will be divided into tutorial groups. You will be able to find out which group you are in from your personal timetable.
- Each tutorial group meets once a week.
- Problem sheets will become available on the module web page (`https://intranet.csc.liv.ac.uk/~konev/COMP109`).
  Try to solve the problems before your tutorial. Part of your continuous assessment mark will be based on your contribution during tutorials, including
  1. making reasonable attempts to solve the problems, and bringing these (in writing) to tutorials, and
  2. your contribution to group discussions in the tutorial group.

  You will hand your work in at the end of each tutorial and get a feedback the following week.

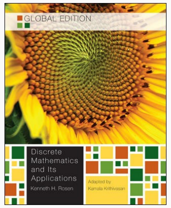
Extenuating circumstances

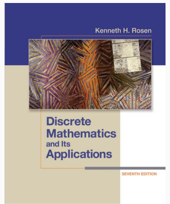
If you cannot attend a tutorial / test / exam for a good reason

- Notify the department (see the handbook)
- Missed tutorial: hand in your best attempt at your earliest opportunity.
- Missed class test: dept. decides either resit or module mark is based on other assessment.
- Missed exam: first attempt stutus in resits.

Core textbook

- K. Rosen. Discrete Mathematics and Its Applications, McGraw-Hill. 7th edition, 2012.

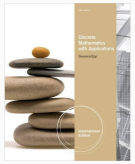
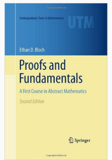


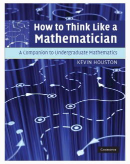


(any edition, including the US edition, is OK)

Recommended books

- S. Epp. Discrete Mathematics with Applications, Cengage Learning. 4th edition, 2011.
- E. Lehman, F. T. Leighton and A. R. Meyer Mathematics for Computer Science. **Free book**
- E. Bloch. Proofs and Fundamentals, Springer. 2nd edition, 2011
- K. Houston. How to Think Like a Mathematician, Cambridge University Press. 2009



Course contents

- Part 1. Number Systems and Proof Techniques
- Part 2. Set Theory
- Part 3. Functions
- Part 4. Relations
- Part 5. Propositional Logic & Digital Circuits
- Part 6. Combinatorics & Probability

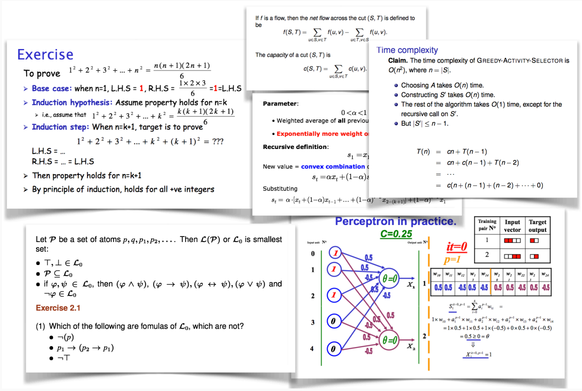
So, this is maths...

- The module does not depend upon A-level maths.
- You can get a first in this module even if you did badly at GCSE maths.
- To do well in this module, you have to work **hard**.

But Who Needs Maths?

You do!

Comp108, Comp 202, Comp226, Comp304, Comp305, Comp309,...



Datatypes

A datatype in a programming language is a set of values and the operations on those values. The datatype states

- the possible values for the datatype
- the operations that can be performed on the values
- the way that values are stored.

Number systems and datatypes

- The most basic datatypes
  - Natural Numbers
  - Integers
  - Rationals
  - Real Numbers
  - Prime Numbers

Number systems and proof techniques

- Proof Techniques
  - Finding a counter-example
  - Proof by contradiction
  - Proof by Induction

These are used, for example, to reason about data types and to reason about algorithms.

We use proof techniques, both to show that an algorithm is correct and to show that it is efficient.

Data collections

Most applications work with collections of data items

- Price list
- Phonebook
- Climate change data
- Stock exchange data
- ...

Sets

A set is a well-defined collection of objects. The objects in the set are called the elements or members of the set.

- The set containing the numbers 1, 2, 3, 4 and 5 is written $\{1, 2, 3, 4, 5\}$.
- The number 3 is an element of the set, that is, $3 \in \{1, 2, 3, 4, 5\}$.
- The number 6 is not an element of the set, that is, $6 \notin \{1, 2, 3, 4, 5\}$.
- The set {dog, cat, mouse} is a set with three elements: dog, cat and mouse.

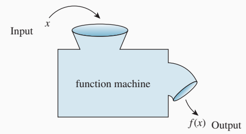
Young man, in mathematics you don't understand things. You just get used to them. (John von Neumann)

Some important sets

- $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$ (the **natural** numbers)
- $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$ (the **integers**)
- $\mathbb{Q} = \{p/q \mid p \text{ and } q \text{ are integers}, q \neq 0\}$ (the **rationals**)
- $\mathbb{R}$: (**real** numbers)

Functions

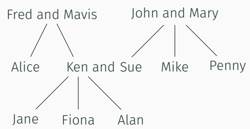
- A function is just a map from a set of inputs to a set of outputs.
  - This is exactly what an algorithm computes.
- Functions can also be used to determine how long algorithms take to run.



Examples:

- $y = x^2$
- $y = \sin(x)$
- first letter of your name

Family relations



Write down

- $R = \{(x, y) \mid x \text{ is a grandfather of } y\}$;

Relations and databases

Databases: Most databases store information as *relations* over *sets*. We need precise notation and terminology for sets and relations in order to talk about databases. Basic mathematical facts about relations and sets are required to understand how a database is designed and implemented.

Logic and specification languages

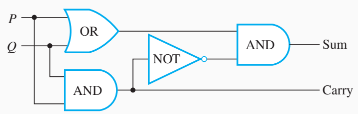
How can we specify what a program should do? Natural languages can be long-winded and ambiguous and are not appropriate for intricate problems.

A formal language without ambiguous statements is required.

Propositional and Predicate Logic are the most important formal languages for specifying programs.

Propositional logic and digital circuits

- Syntax: formulas and formal representations
- Semantics: interpretations and truth tables
- Logic and digital circuits
- Computer arithmetic
- Logical equivalence



Combinatorics

Combinatorics includes the study of counting and also the study of discrete structures such as graphs. It is essential for analysing the efficiency of algorithms.

Combinatorics

- Notation for sums and products, including the factorial function.
- Principles for counting permutations and combinations, for example, to enable you to solve the problem on the following slide.

Applications to discrete probability

The draw selects a set of six different numbers from $1, 2, \dots, 49$. Each choice is equally likely.

You choose a set of six numbers in advance. If your numbers come up, you win the jackpot. What is the probability of this event?

Reading mathematics[1]

- Read with a purpose
- Choose a book at the right level
- Read with pen and paper at hand
- Don't read it like a novel
- Identify what is important
- Stop periodically to review
- Read statements first—proofs later
- Do the exercises and problems
- Reflect
- Write a summary

[1] *How to think like a mathematician* by K. Houston.

Appendix: Greek letters

Alpha	α A	Iota	ι I	Sigma	σ Σ
Beta	β B	Kappa	κ K	Tau	τ T
Gamma	γ Γ	Lambda	λ Λ	Upsilon	υ Υ
Delta	δ Δ	Mu	μ M	Phi	ϕ Φ
Epsilon	ϵ E	Nu	ν N	Chi	χ X
Zeta	ζ Z	Omicron	o O	Psi	ψ Ψ
Eta	η E	Pi	π Π	Omega	ω Ω
Theta	θ Θ	Rho	ρ R		