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

Aims of the Module

- 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.

Module Outcomes

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

- 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 basic counting and enumeration methods as these arise in analysing permutations and combinations;
- perform simple calculation about discrete probability.

Assessment

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

Lectures

We will have three lectures per week this term.

Your timetable is on *Liverpool Life*.

- Get copies of the slides from the help desk or the module web site. Read the slides before (and after) the lecture.
- Take notes. (University is a lot different from school.)
- I will write on the slides.
- Notes can 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.

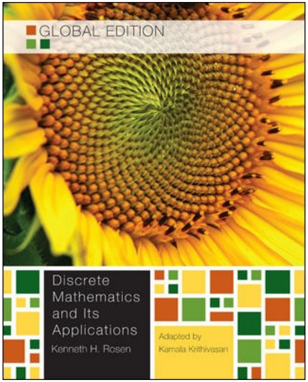
Tutorials

- 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 are available on the web page and at the help desk. 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.

  Written solutions will be distributed during tutorials.

Core Textbook

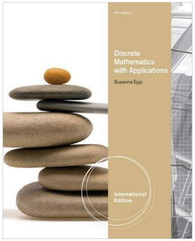
- K. Rosen. Discrete Mathematics and Its Applications, McGraw-Hill. 7th edition, 2012.

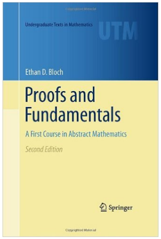
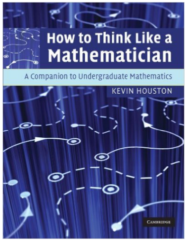

GLOBAL EDITION
Discrete Mathematics and Its Applications
Kenneth H. Rosen
Adapted by Kamala Krithivasan

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

Recommended Books

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




Undergraduate Texts in Mathematics
UTM
Ethan D. Bloch
Proofs and Fundamentals
A First Course in Abstract Mathematics
Second Edition
Springer

How to Think Like a Mathematician
A Companion to Undergraduate Mathematics
KEVIN HOUSTON
CAMBRIDGE

Course Contents

- Part 1. Number Systems and Proof Techniques
- Part 2. Set Theory
- Part 3. Functions
- Part 4. Relations
- Part 5. Propositional Logic
- Part 6. Combinatorics
- Part 7. Probability

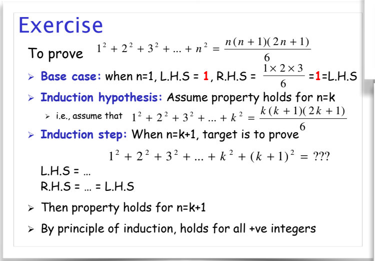
So, This Is Maths…

- 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,...



Exercise

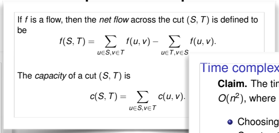
To prove $1^2 + 2^2 + 3^2 + \ldots + n^2 = \frac{n(n+1)(2n+1)}{6}$

- **Base case:** when n=1, L.H.S = **1**, R.H.S = $\frac{1 \times 2 \times 3}{6}$ =**1**=L.H.S
- **Induction hypothesis:** Assume property holds for n=k
  - i.e., assume that $1^2 + 2^2 + 3^2 + \ldots + k^2 = \frac{k(k+1)(2k+1)}{6}$
- **Induction step:** When n=k+1, target is to prove

$$1^2 + 2^2 + 3^2 + \ldots + k^2 + (k+1)^2 = ???$$

L.H.S = ...
R.H.S = ... = L.H.S

- Then property holds for n=k+1
- By principle of induction, holds for all +ve integers

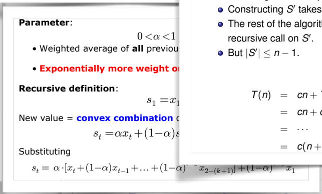


If f is a flow, then the *net flow* across the cut (S, T) is defined to be

$$f(S,T) = \sum_{u \in S, v \in T} f(u,v) - \sum_{u \in T, v \in S} f(u,v).$$

The *capacity* of a cut (S, T) is

$$c(S,T) = \sum_{u \in S, v \in T} c(u,v).$$



Parameter:

$$0 < \alpha < 1$$

- Weighted average of **all** previou
- **Exponentially more weight o**

Recursive definition:

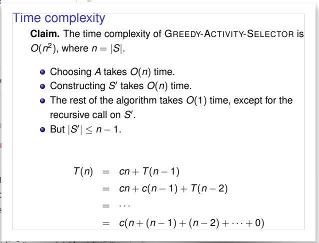
$$s_1 = x_1$$

New value = **convex combination** c

$$s_t = \alpha x_t + (1-\alpha)s$$

Substituting

$$s_t = \alpha \cdot [x_t + (1-\alpha)x_{t-1} + \ldots + (1-\alpha) \quad x_{2-(k+1)}] + (1-\alpha) \quad x_1$$

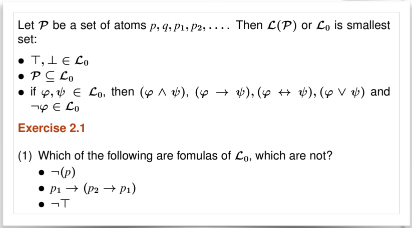


Time complexity

Claim. The time complexity of GREEDY-ACTIVITY-SELECTOR is $O(n^2)$, where $n = |S|$.

- Choosing A takes $O(n)$ time.
- Constructing S' takes $O(n)$ time.
- The rest of the algorithm takes $O(1)$ time, except for the recursive call on S'.
- But $|S'| \le n - 1$.

$$\begin{aligned} T(n) &= cn + T(n-1) \\ &= cn + c(n-1) + T(n-2) \\ &= \cdots \\ &= c(n + (n-1) + (n-2) + \cdots + 0) \end{aligned}$$

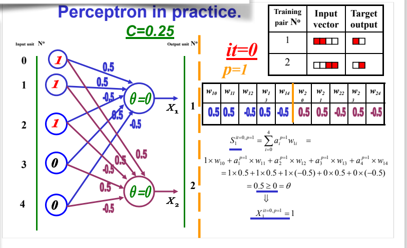


Let $\mathcal{P}$ be a set of atoms $p, q, p_1, p_2, \ldots$. Then $\mathcal{L}(\mathcal{P})$ or $\mathcal{L}_0$ is smallest set:

- $\top, \bot \in \mathcal{L}_0$
- $\mathcal{P} \subseteq \mathcal{L}_0$
- if $\varphi, \psi \in \mathcal{L}_0$, then $(\varphi \wedge \psi)$, $(\varphi \rightarrow \psi)$, $(\varphi \leftrightarrow \psi)$, $(\varphi \vee \psi)$ and $\neg\varphi \in \mathcal{L}_0$

Exercise 2.1

(1) Which of the following are fomulas of $\mathcal{L}_0$, which are not?
  - $\neg(p)$
  - $p_1 \rightarrow (p_2 \rightarrow p_1)$
  - $\neg\top$



Perceptron in practice.

Training pair N°	Input vector	Target output
1	🟥🟥⬜⬜	🟥⬜
2	⬜⬜🟥🟥	⬜🟥

w_{10}	w_{11}	w_{12}	w_{13}	w_{14}	w_{20}	w_{21}	w_{22}	w_{23}	w_{24}
0.5	0.5	-0.5	0.5	-0.5	0.5	0.5	-0.5	0.5	-0.5

$$\underline{S_1^{it=0,p=1}} = \sum_{i=0}^{4} a_i^{p=1} w_{1i} =$$

$$1 \times w_{10} + a_1^{p=1} \times w_{11} + a_2^{p=1} \times w_{12} + a_3^{p=1} \times w_{13} + a_4^{p=1} \times w_{14}$$

$$= 1 \times 0.5 + 1 \times 0.5 + 1 \times (-0.5) + 0 \times 0.5 + 0 \times (-0.5)$$

$$= \underline{0.5 \ge 0} = \theta$$

$$\Downarrow$$

$$\underline{X_1^{it=0,p=1} = 1}$$

Number Systems and Proof Techniques

- Number Systems
  - Natural Numbers
  - Integers
  - Rationals
  - Real Numbers
  - Prime Numbers
- Proof Techniques
  - Finding a counter-example
  - Proof by contradiction
  - Proof by induction

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.

Class Problem

Prove that the sum of any two odd integers is even.

Set Theory

The collections available in programming languages are built from simple datatypes.

- Notation for *sets*.
- What is a *subset* of a set?
- When are two sets *equal*?
- *Operations* on sets (how to construct new sets from old sets)
- *Algebra* of sets.
- *Cardinality* of sets.
- The *Cartesian product* of sets.
- Bit strings.
- Russell's paradox.

Class Problem

Suppose

$$A = \{4, 7, 8\}$$

and

$$B = \{4, 9, 10\}.$$

Compute

$$A \cup B.$$

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.

Functional Programming

A functional programming language manipulates symbols using basic primitive *functions*.

The power of such languages lies in their ability to build complicated processes from simple ones by composition of the basic primitive functions.

The Pigeonhole Principle

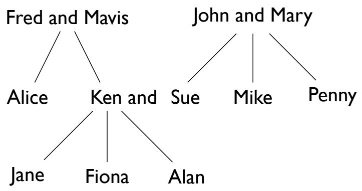
If you put n items into m pigeonholes with $n > m$ then at least one pigeonhole must contain more than one item.

Lossless compression on a computer: Any compression algorithm that makes at least one input file smaller must make some other input file larger.

Relation

- Definition
- How to represent a binary relation as a directed graph or a matrix
- Unary relations
- Properties of binary relations
- Transitive closure
- Equivalence relations and partitions
- Partial orders and total orders.

Class Problem: Family tree



Write down

- $R = \{(x, y) \mid$ x 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

- Syntax: formulas and formal representations
- Semantics: interpretations and truth tables
- Tautologies
- Contradictions
- Semantic consequence
- Logical equivalence

Next term, you will have have a module which extends your knowledge of propositional logic, and teaches you predicate logic, which is a more sophisticated logic in which formulas contain variables that can be quantified.

Class Problem: Logical Puzzle

- An island has two kinds of inhabitants, knights, who always tell the truth, and knaves, who always lie.
- You go to the island and meet A and B.
  - A says "B is a knight."
  - B says "The two of us are of opposite types."
- What are A and B?

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.

Class Problem

A small company employs eight people in the manufacturing department, five in the marketing department and three in the accounting department. A project team of six is to be formed to discuss the launch of a new product. In how many ways can the team be formed if:

1. there are exactly two representatives from each department?
2. there are at least two members from the manufacturing department?

Discrete probability

- Sample spaces and events
- Conditional probability and independence
- Random variables, expectation and linearity of expectation
- The principle of inclusion/exclusion.

In practice, many algorithms use randomisation. Also, randomisation is used in the analysis of algorithms.

Class Problem: The National Lottery lotto

The draw selects a set of six different numbers from $1, 2, \ldots, 49$. 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		