Computer Science Tripos
Syllabus and Booklist 2010–2011

Contents

Introduction to Part IA

Entry to the Computer Science Tripos

The only essential GCE A level for admission to Cambridge to read for the Computer Science Tripos is Mathematics. Also desirable are Further Mathematics and a physical science (Physics, Chemistry or Geology) at A level, or at AS level if not taken at A level. Some colleges may ask candidates to take the Advanced Extension Award or STEP papers in Mathematics.

Computer Science Tripos Part IA

Part IA students accepted to read **Computer Science with Mathematics** will attend, besides the courses listed in this document, lectures for Papers 1 and 2 of Part IA of the Mathematical Tripos.

All other Part IA students are required to attend, besides the lectures listed in this document, the Mathematics course offered for Part IA of the Natural Sciences Tripos, together with **either** Paper 3 of Part IA of the Politics, Psychology and Sociology (PPS) Tripos **or** *one* other Natural Science subject selected from the following list: Chemistry, Evolution and Behaviour, Earth Sciences, Physics, and Physiology of Organisms.

Physics is recommended for those with an A-level in the subject; potential applicants may note that there is no A-level prerequisite for Evolution and Behaviour, Earth Sciences or Physiology of Organisms, although an AS-level science would be desirable. Laboratory work forms an integral part of the Natural Sciences Part IA course, and students reading the Computer Science Tripos will be required to undertake practical work on the same basis as for the Natural Sciences Tripos. There is no A-level requirement for those taking Paper 3 of the PPS Tripos.

Natural Sciences Part IA students

There is a Computer Science option in the first year of the Natural Sciences Tripos, counting as one quarter of the year's work. Students taking this option attend all the lectures and practicals listed in this document, with the exception of those indicated as being Paper 2 courses.

Politics, Psychology and Sociology Part I students

There is an "Introduction to Computer Science" option in Part I of the Politics, Psychology and Sociology Tripos. Students taking this option attend all the lectures and practicals listed in this document, with the exception of those indicated as being Paper 2 courses.

The curriculum

This document lists the courses offered by the Computer Laboratory for Papers 1 and 2 of Part IA of the Computer Science Tripos. Separate booklets give details of the syllabus for the second- and third-year courses in Computer Science.

The syllabus information given here is for guidance only and should not be considered definitive. Current timetables can be found at
`http://www.cl.cam.ac.uk/teaching/lectlist/`

For most of the courses listed below, a list of recommended books is given. These are roughly in order of usefulness, and lecturers have indicated by means of an asterisk those books which are most recommended for purchase by College libraries.

The Computer Laboratory Library aims to keep at least one copy of each of the course texts in "The Booklocker" (see `http://www.cl.cam.ac.uk/library/`).

For further copies of this booklet and for answers to general enquiries about Computer Science courses, please get in touch with:

Student Administrator
University of Cambridge
Computer Laboratory
William Gates Building
J J Thomson Avenue
Cambridge
CB3 0FD

telephone: 01223 334656
fax: 01223 334678
e-mail: `undergraduate.admissions@cl.cam.ac.uk`

Michaelmas Term 2010: Part IA lectures

Paper 1: Computer Fundamentals

Lecturer: Dr S.M. Hand

No. of lectures: 6

This course is a prerequisite for Operating Systems.

Aims

The overall aim of this course is to provide a general understanding of how a computer works. This includes aspects of the underlying hardware (CPU, memory, devices), as well as how to program a computer at a low level using assembly language.

Lectures

- History: from vacuum tubes to VLSI. The Von Neumann architecture. Hardware/software layers and languages. Operation of a simple computer (processors, memory, buses, devices). Memory: concepts, structures, hierarchy. [1 lecture]
- Processor: control and execution units. ALU and computer arithmetic. Logical and conditional operations. Branches. Memory access. Data representation: (integers), text, reals, compound structures, instructions. Fetch-Execute cycle revisited. [1 lecture]
- General I/O architecture. Example devices. Buses: general operation, hierarchy, synchronous *versus* asynchronous. Interrupts. Direct Memory Access. [1 lecture]
- Programming in Assembly. MIPS instruction formats. Assembler directives and pseudo-instructions. Memory access and control flow instructions. The SPIM simulator. Example programs. Procedures and the stack. Recursive procedures. [2 lectures]
- Course Review. [1 lecture]

Objectives

At the end of the course students should be able to

- describe the fetch–execute cycle of a simple computer with reference to the control and execution units;
- understand the different types of information which may be stored within a computer memory;
- understand a simple assembly language program.

Recommended reading

Tanenbaum, A.S. (1990). *Structured computer organisation*. Prentice Hall (3rd ed).
Patterson, D. & Hennessy, J. (1998). *Computer organisation and design*. Morgan Kaufmann (2nd ed.).

Paper 1: Foundations of Computer Science

Lecturer: Dr M.O. Myreen

No. of lectures and practicals: 15 + 6

This course is a prerequisite for Programming in Java and Prolog (Part IB).

Aims

The main aim of this course is to present the basic principles of programming. As the introductory course of the Computer Science Tripos, it caters to students from all backgrounds. To those who have had no programming experience, it will be comprehensible; to those experienced in languages such as C, it will attempt to correct any bad habits that they have learnt.

A further aim is to introduce the principles of data structures and algorithms. The course will emphasise the algorithmic side of programming, focusing on problem-solving rather than on hardware-level bits and bytes. Accordingly it will present basic algorithms for sorting, searching, etc., and discuss their efficiency using O-notation. Worked examples (such as polynomial arithmetic) will demonstrate how algorithmic ideas can be used to build efficient applications.

The course will use a functional language (ML). ML is particularly appropriate for inexperienced programmers, since a faulty program cannot crash. The course will present the elements of functional programming, such as curried and higher-order functions. But it will also discuss traditional (procedural) programming, such as assignments, arrays, pointers and mutable data structures.

Lectures

- **Introduction.** Levels of abstraction. Floating-point numbers, and why von Neumann was wrong. Why ML? Integer arithmetic. Giving names to values. Declaring functions. Static binding, or declaration *versus* assignment.

- **Recursive functions.** Examples: Exponentiation and summing integers. Overloading. Decisions and booleans. Iteration *versus* recursion.

- **O Notation.** Examples of growth rates. Dominance. O, Omega and Theta. The costs of some sample functions. Solving recurrence equations.

- **Lists.** Basic list operations. Append. Naïve *versus* efficient functions for length and reverse. Strings.

- **More on lists.** The utilities `take` and `drop`. Pattern-matching: zip, unzip. A word on polymorphism. The “making change” example.
- **Sorting.** A random number generator. Insertion sort, mergesort, quicksort. Their efficiency.
- **Datatypes and trees.** Pattern-matching and case expressions. Exceptions. Binary tree traversal (conversion to lists): preorder, inorder, postorder.
- **Dictionaries and functional arrays.** Functional arrays. Dictionaries: association lists (slow) *versus* binary search trees. Problems with unbalanced trees.
- **Queues and search strategies.** Depth-first search and its limitations. Breadth-first search (BFS). Implementing BFS using lists. An efficient representation of queues. Importance of efficient data representation.
- **Functions as values.** Nameless functions. Currying.
- **List functionals.** The “apply to all” functional, map. Examples: matrix transpose and product. The “fold” functionals. Predicate functionals “filter” and “exists”.
- **Polynomial arithmetic.** Addition, multiplication of polynomials using ideas from sorting, etc.
- **Sequences, or lazy lists.** Non-strict functions such as *IF*. Call-by-need *versus* call-by-name. Lazy lists. Their implementation in ML. Applications, for example Newton-Raphson square roots.
- **Elements of procedural programming.** Address *versus* contents. Assignment *versus* binding. Own variables. Arrays, mutable or not.
- **Linked data structures.** Linked lists. Surgical concatenation, reverse, etc.

Objectives

At the end of the course, students should

- be able to write simple ML programs;
- understand the importance of abstraction in computing;
- be able to estimate the efficiency of simple algorithms, using the notions of average-case, worse-case and amortised costs;
- know the comparative advantages of insertion sort, quick sort and merge sort;
- understand binary search and binary search trees;
- know how to use currying and higher-order functions.

Recommended reading

* Paulson, L.C. (1996). *ML for the working programmer*. Cambridge University Press (2nd ed.).
Okasaki, C. (1998). *Purely functional data structures*. Cambridge University Press.

Gentler alternative to the main text:
Hansen, M. & Rischel, H. (1999). *Introduction to programming using SML*. Addison-Wesley.

For reference only:
Gansner, E.R. & Reppy, J.H. (2004). *The Standard ML Basis Library*. Cambridge University Press. ISBN: 0521794781

Paper 1: Discrete Mathematics I

Lecturer: Dr S. Staton

No. of lectures: 9 (Continued into Lent Term)

This course is a prerequisite for all theory courses as well as Discrete Mathematics II, Algorithms I, Security (Parts IB and II), Artificial Intelligence (Parts IB and II), Information Theory and Coding (Part II).

Aims

This course will develop the intuition for discrete mathematics reasoning involving numbers and sets.

Lectures

- **Logic.** Propositional and predicate logic formulas and their relationship to informal reasoning, truth tables, validity, proof. [4 lectures]
- **Sets.** Basic set constructions. [2 lectures]
- **Induction.** Proof by induction, including proofs about total functional programs over natural numbers and lists. [2 lectures]

Objectives

On completing the course, students should be able to

- write a clear statement of a problem as a theorem in mathematical notation;
- prove and disprove assertions using a variety of techniques.

Recommended reading

* Velleman, D.J. (1994). *How to prove it (a structured approach).* Cambridge University Press.

* Rosen, K.H. (1999). *Discrete mathematics and its applications*. McGraw-Hill (6th ed.).
Biggs, N.L. (1989). *Discrete mathematics.* Oxford University Press.
Bornat, R. (2005). *Proof and disproof in formal logic.* Oxford University Press.
Devlin, K. (2003). *Sets, functions, and logic: an introduction to abstract mathematics.* Chapman and Hall/CRC Mathematics (3rd ed.).
Mattson, H.F. Jr (1993). *Discrete mathematics.* Wiley.
Nissanke, N. (1999). *Introductory logic and sets for computer scientists.* Addison-Wesley.
Pólya, G. (1980). *How to solve it.* Penguin.

Paper 2: Digital Electronics

This course is not taken by NST or PPST students.

Lecturer: Dr I.J. Wassell

No. of lectures and practical classes: 11 + 7

This course is a prerequisite for Operating Systems and Computer Design (Part IB).

Aims

The aims of this course are to present the principles of combinational and sequential digital logic design and optimisation at a gate level. The use of transistors for building gates is also introduced.

Lectures

- **Introduction.** Semiconductors to computers. Logic variables. Examples of simple logic. Logic gates. Boolean algebra. De Morgan's theorem.
- **Logic minimisation.** Truth tables and normal forms. Karnaugh maps.
- **Number representation.** Unsigned binary numbers. Octal and hexadecimal numbers. Negative numbers and 2's complement. BCD and character codes. Binary adders.
- **Combinational logic design: further considerations.** Multilevel logic. Gate propagation delay. An introduction to timing diagrams. Hazards and hazard elimination. Fast carry generation. Other ways to implement combinational logic.
- **Introduction to practical classes.** Prototyping box. Breadboard and Dual in line (DIL) packages. Wiring. Use of oscilloscope.
- **Sequential logic.** Memory elements. RS latch. Transparent D latch. Master–slave D flip-flop. T and JK flip-flops. Setup and hold times.
- **Sequential logic.** Counters: Ripple and synchronous. Shift registers.
- **Synchronous State Machines.** Moore and Mealy finite state machines (FSMs). Reset and self starting. State transition diagrams.

- **Further state machines.** State assignment: sequential, sliding, shift register, one hot. Implementation of FSMs.
- **Circuits.** Solving non-linear circuits. Potential divider. N-channel MOSFET. N-MOS inverter. N-MOS logic. CMOS logic. Logic families. Noise margin. [2 lectures]

Objectives

At the end of the course students should

- understand the relationships between combination logic and boolean algebra, and between sequential logic and finite state machines;
- be able to design and minimise combinational logic;
- appreciate tradeoffs in complexity and speed of combinational designs;
- understand how state can be stored in a digital logic circuit;
- know how to design a simple finite state machine from a specification and be able to implement this in gates and edge triggered flip-flops;
- understand how to use MOS transistors.

Recommended reading

* Harris, D.M. & Harris, S.L. (2007). Digital design and computer architecture. Morgan Kaufmann.
Katz, R.H. (2004). *Contemporary logic design*. Benjamin/Cummings. The 1994 edition is more than sufficient.
Hayes, J.P. (1993). *Introduction to digital logic design*. Addison-Wesley.

Books for reference:

Horowitz, P. & Hill, W. (1989). *The art of electronics*. Cambridge University Press (2nd ed.) (more analog).
Weste, N.H.E. & Harris, D. (2005). *CMOS VLSI Design – a circuits and systems perspective*. Addison-Wesley (3rd ed.).
Mead, C. & Conway, L. (1980). *Introduction to VLSI systems*. Addison-Wesley.
Crowe, J. & Hayes-Gill, B. (1998). *Introduction to digital electronics*. Butterworth-Heinemann.
Gibson, J.R. (1992). *Electronic logic circuits*. Butterworth-Heinemann.

Paper 2: Operating Systems

This course is not taken by NST or PPST students.

Lecturer: Dr S.M. Hand

No. of lectures: 13

Prerequisite courses: Computer Fundamentals, Digital Electronics

This course is a prerequisite for Concurrent & Distributed Systems (Part IB), Security (Parts IB and II) and Mobile and Sensor Systems (Part II).

Aims

The overall aim of this course is to provide a general understanding of the structure and key functions of the operating system. Case studies will be used to illustrate and reinforce fundamental concepts.

Lectures

- **Introduction to operating systems.** Abstract view of an operating system. OS evolution: multi-programming, time-sharing. Dual-mode operation. Protecting I/O, memory, CPU. Kernels and micro-kernels. [1 lecture]
- **Processes and scheduling.** Job/process concepts. Scheduling basics: CPU-I/O interleaving, (non-)preemption, context switching. Scheduling algorithms: FCFS, SJF, SRTF, priority scheduling, round robin. Combined schemes. [2 lectures]
- **Memory management.** Processes in memory. Logical addresses. Partitions: static *versus* dynamic, free space management, external fragmentation. Segmented memory. Paged memory: concepts, internal fragmentation, page tables. Demand paging/segmentation. Replacement strategies: OPT, FIFO, LRU (and approximations), NRU, LFU/MFU, MRU. Working set schemes. [3 lectures]
- **I/O subsystem.** General structure. Polled mode *versus* interrupt-driven I/O. Application I/O interface: block and character devices, buffering, blocking *versus* non-blocking I/O. Other issues: caching, scheduling, spooling, performance. [1 lecture]
- **File management.** File concept. Directory and storage services. File names and meta-data. Directory name-space: hierarchies, DAGs, hard and soft links. File operations. Access control. Existence and concurrency control. [1 lecture]
- **Protection.** Requirements. Subjects and objects. Design principles. Authentication schemes. Access matrix: ACLs and capabilities. Combined scheme. Covert channels. [1 lecture]
- **Unix case study.** History. General structure. Unix file system: file abstraction, directories, mount points, implementation details. Processes: memory image, life cycle, start of day. The shell: basic operation, commands, standard I/O, redirection, pipes, signals. Character and block I/O. Process scheduling. [2 lectures]

- **Windows NT case study.** History. Design principles. Overall architecture. HAL. Kernel: objects, processes, threads, scheduling. Executive: object manager and object namespace, process manager, VM manager, I/O manager. File-System. Security System. [2 lectures]

Objectives

At the end of the course students should be able to

- describe the general structure and purpose of an operating system;
- explain the concepts of process, address space, and file;
- compare and contrast various CPU scheduling algorithms;
- understand the differences between segmented and paged memories, and be able to describe the advantages and disadvantages of each;
- compare and contrast polled, interrupt-driven and DMA-based access to I/O devices.

Recommended reading

* Bacon, J. & Harris, T. (2003). *Operating systems*. Addison-Wesley (3rd ed.).
Silberschatz, A., Peterson, J.L. & Galvin, P.C. (2005). *Operating systems concepts*. Addison-Wesley (7th ed.).
Leffler, S. (1989). *The design and implementation of the 4.3BSD Unix operating system*. Addison-Wesley.
Solomon, D. & Russinovich, M. (2000). *Inside Windows 2000*. Microsoft Press (3rd ed.).

Lent Term 2011: Part IA lectures

Paper 1: Programming in Java

Lecturers: Dr A.R. Beresford and Dr A.C. Rice

No. of practical classes: 8 $\times$ 2-hour sessions

Prerequisite course: Foundations of Computer Science

Companion courses: Object-Oriented Programming, Floating-Point Computation

This course is a prerequisite for Algorithms I and II, and for the Group Project and Part IB course Concurrent and Distributed Systems.

Aims

The goal of this course is to provide students with the ability to write programs in Java and apply concepts described in the Object-Oriented Programming course. The course is designed to accommodate students with diverse programming backgrounds; consequently Java is taught from first principles in a practical class setting where students can work at their own pace from a course handbook. Each practical class will culminate in an assessed exercise.

Practical classes

- **Introduction.** This class will introduce the students to PWF Linux, the Java compiler and tool chain. Students will design, implement and test their first Java application.
- **Methods, operators and types.** This class will concentrate on the fundamentals of imperative programming. Students will learn about Java primitive types, variable declaration, operators and method calls.
- **Control structures.** Students will explore the control structures found in Java.
- **Arrays, references and classes.** This week the students will explore arrays and references in Java and learn how to define and instantiate their own class.
- **Input/Output and Exceptions.** This class will examine streams and Exceptions. Students will read and write data to and from the filesystem and network and learn to handle errors using Java Exceptions.
- **Inheritance and interfaces.** This class will explore object-oriented programming as expressed in Java. Students will learn how to extend classes, as well as specify and provide implementations for Java interfaces.
- **Abstraction and graphical interfaces.** Students will examine code-reuse through inheritance and the use of inner classes for encapsulation. Students will begin to construct a graphical interface using Swing.
- **Swing and event handling.** Students will complete their graphical interface by writing event handlers to control the execution of a graphical application.

Objectives

At the end of the course students should

- be familiar with the main features of the Java language;
- be able to write a Java program to solve a well specified problem;
- understand a Java program written by someone else;
- be able to debug and test Java programs;
- be familiar with major parts of Java 6 SE libraries;
- understand how to read Javadoc library documentation and reuse library code.

Recommended reading

* Eckel, B. (2006). *Thinking in Java*. Prentice Hall (4th ed.).

Paper 1: Object-Oriented Programming

Lecturer: Dr R.K. Harle

No. of lectures: 9

Companion course: Programming in Java

Aims

This course runs in parallel with the *Programming in Java* practical course. It is intended to provide both an introduction to Java that complements the practical workbooks and to highlight the abstract notion of object-oriented programming. Examples and discussions will use Java primarily, but other languages may be used to illustrate specific points where appropriate.

Syllabus

- **Hardware Refresher.** Fetch-execute cycle. Registers. System Architectures. Imperative languages as closer to the hardware. The JVM idea. [1/2 lecture]
- **Programmer's Model of Memory.** Primitive Types. Pointers. References. Pass-by-value and pass-by-reference. Reference Types. [1/2 lecture]
- **OOP Concepts.** Objects vs classes. Identifying objects. Distinguishing state and behaviour. UML class diagrams. Modularity. Encapsulation. Inheritance. Casting. Polymorphism. Abstract Classes. Multiple inheritance. Java interfaces. Representing class-level data. Exceptions in brief. [3 lecture]
- **Lifecycle of an Object.** Constructors. Destructors. Garbage Collection. [1/2 lecture]

- **Copying Objects.** Copy constructors. Cloning in Java. Cloneable as a marker interface in Java. [1/2 lecture]
- **Comparing Objects.** Comparing primitive types. Comparing reference types. Comparable and Comparator in Java. [1 lecture]
- **Templates and Generics.** Java Collections framework as motivation. Examples of generic programming. [1 lecture]
- **Design patterns and design examples.** Introduction to design patterns. Applying design patterns to example problems. Design patterns in the Java class library. Examples of building a Java program from problem statement to testing. [2 lectures]
- **Common Java errors.** The need for care with syntax. Numerical overflow and other common problems. [if time allows]

Objectives

At the end of the course students should

- understand the principles of OOP;
- be able to demonstrate good object-oriented programming skills in Java;
- understand the capabilities and limitations of Java;
- be able to describe, recognise, apply and implement selected design patterns in Java;
- be familiar with common errors in Java and its associated libraries.

Recommended reading

No single text book covers all of the topics in this course. For those new to OOP, the best introductions are usually found in the introductory programming texts for OOP languages (such as Java, python or C++). Look for those that are for people new to programming rather than those that are designed for programmers transitioning between languages (the Deitel book is highlighted for this reason). The web is also a very useful resource — look for Java tutorials.

* Deitel, H.M. & Deitel, P.J. (2003). *Java: How to Program*. Prentice Hall.
Flanagan, D. (2005). *Java in a nutshell : a desktop quick reference*. O'Reilly (5th ed.).
Flanagan, D. (2004). *Java examples in a nutshell : a tutorial companion to Java in a nutshell*. O'Reilly (3rd ed.).
Gamma, E., Helm, R., Johnson, R. & Vlissides, A. (1995). *Design patterns: elements of reusable object-oriented software*. Addison-Wesley.
Bloch, J. & Gafter, N. (2005). *Java puzzlers*. Addison-Wesley.

Paper 1: Floating-Point Computation

Lecturer: Dr D.J. Greaves

No. of lectures: 6

This course is useful for the Part II courses Advanced Graphics and Digital Signal Processing.

Aims

This course has two aims: firstly to provide an introduction to (IEEE) floating-point data representation and arithmetic; and secondly to show, how naïve implementations of obvious mathematics can go badly wrong. An overall implicit aim is to encourage caution when using any floating-point value produced by a computer program.

Lectures

- **Integer and floating-point representation and arithmetic.** Signed and unsigned integers and fixed-point; arithmetic, saturating arithmetic. IEEE 754/854 floating point (32 and 64 bit); zeros, infinities, NaN. Brief mention of IEEE 754r. What numbers are exactly representable in bases 2 and 10. Accuracy in terms of significant figures. Floating point arithmetic is non-associative, and mathematical equivalences fail. Nonsensical results, e.g. `sin(1e40)`, counting in floating point.
- **IEEE Floating-point arithmetic.** Floating point arithmetic, and the IEEE requirements. Why the IEEE standard has endured. Overflow, underflow, progressive loss of significance. Rounding modes. Difficulty in obtaining IEEE-quality in libraries. The java.lang.Math trigonometric library promises.
- **How floating-point computations diverge from real-number calculations.** Absolute Error, Relative Error, Machine epsilon, Unit in Last Place (ulp). Finite computation: solving a quadratic. Summing a finite series. Rounding (round-off) and truncation (discretisation) error. Numerical differentiation; determining a good step size.
- **Iteration and when to stop.** Unbounded computation may produce unbounded errors. Solving equations by iteration and comparison to terminate it. Newton's method. Idea of order of convergence. Why summing a Taylor series is problematic (loss of all precision, range reduction, non-examinable hint at economisation).
- **Ill-conditioned or chaotic problems.** Effect of changes of a few ulp in the inputs. Conditioning number when amenable to mathematical analysis; Monte-Carlo exploration when not.
- **Other approaches and their problems** Adaptive methods. Arbitrary precision floating point, adaptive floating point, interval arithmetic. Discussion on the problems of exact real arithmetic. Remark on the x86 implementations of IEEE arithmetic, and compiler "optimisations".

Objectives

At the end of the course students should

- be able to convert simple decimal numbers to and from IEEE floating-point format, and to perform IEEE arithmetic on them;
- be able to identify problems with floating-point implementations of simple mathematical problems;
- know when a problem is likely to yield incorrect solutions no matter how it is processed numerically;
- know to use a professionally-written package whenever possible (and still to treat claims of accuracy with suspicion).

Recommended reading

Overton, M.L. (2001). *Numerical computing with IEEE floating point arithmetic*. SIAM.

Further reading – goes far beyond the course

Goldberg, D. (1991). *What every computer scientist should know about floating-point arithmetic*. ACM Computing Surveys, vol. 23, pp. 5–48.

Paper 2: Probability

This course is not taken by NST or PPST students.

Lecturer: Dr R.J. Gibbens

No. of lectures: 6

This course is a prerequisite for the Part IB course Mathematical Methods for Computer Science, and the following Part II courses: Artificial Intelligence II, Computer Systems Modelling, Information Theory and Coding, Computer Vision, Digital Signal Processing, Natural Language Processing and Information Retrieval.

Aims

The main aim of this course is to provide a foundation in Probability with emphasis on areas that are particularly applicable to Computer Science.

Lectures

- **Review of elementary probability theory.** Random variables. Discrete and continuous distributions. Means and variances, moments, independence, conditional probabilities. Bayes's theorem. [2 lectures]
- **Probability generating functions.** Definitions and properties. Use in calculating moments of random variables and for finding the distribution of sums of independent random variables. [1 lecture]
- **Multivariate distributions and independence.** Random vectors and independence. Joint and marginal density functions. Variance, covariance and correlation. Conditional density functions. [1 lecture]

- **Elementary stochastic processes.** Random walks. Recurrence and transience. The Gambler's Ruin problem. Solution using difference equations. [2 lectures]

Objectives

At the end of the course students should

- have a thorough understanding of concepts in probability theory and a practical knowledge of associated calculations;
- be aware of applications of probability across the field of computer science.

Recommended reading

* Grimmett, G. & Welsh, D. (1986). *Probability: an introduction*. Oxford University Press.

Paper 2: Discrete Mathematics II

This course is not taken by NST or PPST students.

Lecturer: Professor G. Winskel

No. of lectures: 12

Prerequisite course: Discrete Mathematics I

This course is a prerequisite for all theory courses as well as Security (Part IB and Part II), Artificial Intelligence (Part IB and Part II), Information Theory and Coding (Part II).

Aims

This course will develop the theory of sets and their uses in Computer Science.

Lectures

- **Sets and logic.** The basic set operations (union, intersection and complement) on subsets of a fixed set. The Boolean laws. Propositional logic and its models. Validity, entailment, and equivalence of propositions revisited. Structural induction illustrated on propositions. [2 lectures]
- **Relations and functions.** Product of sets. Relations, functions and partial functions. Composition and identity relations. Injective, surjective and bijective functions. Direct and inverse image of a set under a relation. Equivalence relations and partitions; modular arithmetic as an example. Directed graphs and partial orders. Size of sets (cardinality), especially countability. Cantor's diagonal argument to show the reals are uncountable. [3 lectures]
- **Constructions on sets.** Russell's paradox. Basic sets, comprehension, indexed sets, unions, intersections, products, disjoint unions, powersets. Characteristic functions. Sets of functions. Lambda notation for functions. Cantor's diagonal argument to show power set strictly increases size. [2 lectures]

- **Introduction to inductive definitions.** Using rules to define sets; examples. Reasoning principles: rule induction and its instances; induction on derivations briefly. Simple applications, including transitive closure of a relation. [3 lectures]

- **Well-founded induction.** Well-founded relations and well-founded induction. Other induction principles as instances of well-founded induction. Product and lexicographic product of well-founded relations. Examples and applications, including to Euclid's algorithm for HCF/GCD. Informal understanding of definition by well-founded recursion. [2 lectures]

Objectives

On completing this part of the course, students should be able to

- understand and use the language of set theory; prove and disprove assertions using a variety of techniques;

- understand Boolean operations as operations on sets and formulate statements using Boolean logic;

- apply the principle of well-founded induction;

- define sets inductively using rules, and prove properties about them.

Recommended reading

Comprehensive notes will be provided.

Devlin, K. (2003). *Sets, functions, and logic: an introduction to abstract mathematics*. Chapman and Hall/CRC Mathematics (3rd ed.).
Biggs, N.L. (1989). *Discrete mathematics*. Oxford University Press.
Mattson, H.F. Jr (1993). *Discrete mathematics*. Wiley.
Nissanke, N. (1999). *Introductory logic and sets for computer scientists*. Addison-Wesley.
Pólya, G. (1980). *How to solve it*. Penguin.

Paper 2: Software Design

This course is not taken by NST or PPST students.

Lecturer: Dr A.F. Blackwell

No. of lectures: 6

Companion courses: Object-Oriented Programming, Programming in Java

This course is a prerequisite for the Group Project (Part IB).

Aims

The aim of this course is to present a range of effective methods for the design and implementation of software, especially where that software must meet professional quality standards. This will include a brief introduction to current commercial methods, but the main motivation is to understand the reasons why such methods have developed, how they differ from the concerns of academic computer science, and what are the technical foundations of good software engineering.

Lectures

- **Overview of the design process.** Building models suitable for the stages of a software development project. Introduction to UML.
- **Inception phase.** Requirements, use cases, scenarios and structured analysis.
- **Elaboration phase.** Object modelling. Interfaces and abstraction. Information hiding.
- **Construction phase.** Coupling and object interaction. Responsibilities, defensive programming and exceptions. Functional decomposition, module and code layout. Variable roles, object state, verification and assertions.
- **Transition phase.** Inspections, walkthroughs, testing, debugging. Iterative development, prototyping and refactoring. Optimisation.

Objectives

At the end of the course, students should be able to undertake system design in a methodical manner, starting from a statement of system requirements, developing a modular design model, refining it into an implementation that clearly identifies and minimises risk, coding in a manner that can be integrated with the work of a team, and using appropriate methods to identify and prevent faults.

Recommended reading

McConnell, S. (2004). *Code complete: a practical handbook of software construction*. Microsoft Press (2nd ed.).
Fowler, M. (2003). *UML distilled*. Addison-Wesley (3rd ed.).

Revision and reinforcement of object-oriented concepts for those needing this:

Barnes, D.J. & Kölling, M. (2006). *Objects first with Java: a practical introduction using BlueJ*. Pearson Education (3rd ed.).

Further reading

Broy, M. & Denert, E. (ed.) (2002). *Software pioneers: contributions to software engineering*. Springer-Verlag.
Collins, H. & Pinch, T. (1998). *The Golem at large: what you should know about technology*. Cambridge University Press.
Petroski, H. (1985). *To engineer is human: the role of failure in successful design*. Macmillan.

Vincenti, W.G. (1990). *What engineers know and how they know it: analytical studies from aeronautical history*. Johns Hopkins University Press.
Simon, H.A. (1996). *The sciences of the artificial*. MIT Press.
Schon, D.A. (1990). *Educating the reflective practitioner*. Jossey-Bass.
Pressman, R.S. (2001). *Software engineering*. McGraw-Hill (European ed.).

Easter Term 2011: Part IA lectures

Paper 1: Algorithms I

Lecturer: Dr F.M. Stajano

No. of lectures: 12

Prerequisite course: Discrete Mathematics I

This course is a prerequisite for Algorithms II, Artificial Intelligence and Prolog.

Aims

The aim of this course is to provide an introduction to computer algorithms and data structures, with an emphasis on foundational material.

Lectures

- **Sorting.** Review of insertion sort, merge sort and quicksort. Understanding their memory usage with arrays. Heapsort. Other sorting methods. Finding the minimum and maximum. [Ref: Ch 1, 2, 3, 4, 15, 16] [4 lectures]
- **Algorithm design.** Ideas for algorithm design: dynamic programming, divide and conquer, greedy algorithms and other useful paradigms. [Ref: Ch 2, 15] [2 lectures]
- **Data structures.** Abstract data types. Pointers, stacks, queues, lists, trees. Hash tables. Binary search trees. Red–black trees. B-trees. Priority queues and heaps. [Ref: Ch 10, 11, 12, 13, 18, 19, 20, 21] [6 lectures]

Objectives

At the end of the course students should

- have a good understanding of how several fundamental algorithms work, particularly those concerned with sorting and searching;
- have a good understanding of the fundamental data structures used in computer science;
- be able to analyse the space and time efficiency of most algorithms;
- be able to design new algorithms or modify existing ones for new applications and reason about the efficiency of the result.

Recommended reading

* Cormen, T.H., Leiserson, C.D., Rivest, R.L. & Stein, C. (2009). *Introduction to Algorithms*. MIT Press (3rd ed.). ISBN 0-262-53196-8
Sedgewick, R. (2004). *Algorithms in Java* vol. 1 (note that C and C++ editions are also available and are equally good for this course). Addison-Wesley. ISBN 0-201-36120-5. New edition forthcoming in 2008.

Kleinberg, J. & Tardos, É. (2006). *Algorithm design*. Addison-Wesley. ISBN 0-321-29535-8.
Knuth, D.E. (1997). *The art of computer programming* (three volumes so far; a boxed set is also available). Addison-Wesley (3rd ed.). ISBN 0-201-89683-4, 0-201-89684-2 and 0-201-89685-0.

Students are expected to buy, make extensive use of, and keep as reference for their future career, one of the above textbooks: those not doing so will be severely disadvantaged. The recommended choice is Cormen *et al.* which, in spite of its superb quality, is the cheapest (about 35 GBP new for over 1300 pages). The pointers in the syllabus are to chapters in the second edition of that book. The other textbooks are all excellent alternatives and their relative merits are discussed in the course handout.

Paper 2: Regular Languages and Finite Automata

This course is not taken by NST or PPST students.

Lecturer: Dr M.P. Fiore

No. of lectures: 6

This course is useful for Compiler Construction (Part IB) and Natural Language Processing (Part II).

Aims

The aim of this short course will be to introduce the mathematical formalisms of finite state machines, regular expressions and grammars, and to explain their applications to computer languages.

Lectures

- **Regular expressions.** Specifying sets of strings by pattern-matching.
- **Finite state machines.** Deterministic and non-deterministic finite automata and the languages they accept.
- **Regular languages I.** The language determined by a regular expression is regular.
- **Regular languages II.** Every regular language is determined by some regular expression.
- **The Pumping Lemma.** Proof and applications.
- **Grammars.** Context-free and regular grammars. The class of regular languages coincides with the class of languages generated by a regular grammar.

Objectives

At the end of the course students should

- be able to explain how to convert between the three ways of representing regular sets of strings introduced in the course; and be able to carry out such conversions by hand for simple cases;
- be able to prove whether or not a given set of strings is regular.

Recommended reading

* Hopcroft, J.E., Motwani, R. & Ullman, J.D. (2001). *Introduction to automata theory, languages, and computation*. Addison-Wesley (2nd ed.).
Kozen, D.C. (1997). *Automata and computability*. Springer-Verlag.
Sudkamp, T.A. (2005). *Languages and machines*. Addison-Wesley (3rd ed.).

Further Java Briefing

Lecturer: Dr A.R. Beresford

No. of lectures: 1

Prerequisite course: Programming in Java

This course is a prerequisite for Further Java.

Aims

To reinforce concepts introduced in Programming in Java, provide further practical experience with algorithms and data structures, and prepare students for the Part IB Further Java course.

Lecture

The lecture describes the requirements for the first assessed exercise of the Part IB Further Java course.

Objectives

On completing the exercise students should

- be prepared for the Part IB Further Java course;
- have developed their practical Java programming skills further.

Preparing to Study Computer Science

For general advice about preparing for the Computer Science course at Cambridge, please see `http://www.cl.cam.ac.uk/admissions/undergraduate/preparation/`

Introduction to Part IB

This document lists the courses offered by the Computer Laboratory for Part IB of the Computer Science Tripos. Separate booklets give details of the syllabus for the other Parts of the Computer Science Tripos.

The syllabus information given here is for guidance only and should not be considered definitive. Current timetables can be found at
`http://www.cl.cam.ac.uk/teaching/lectlist/`

For most of the courses listed below, a list of recommended books is given. These are roughly in order of usefulness, and lecturers have indicated by means of an asterisk those books which are most recommended for purchase by College libraries.

The Computer Laboratory Library aims to keep at least one copy of each of the course texts in “The Booklocker” (see `http://www.cl.cam.ac.uk/library/`).

For copies of the other syllabus booklets and for answers to general enquiries about Computer Science courses, please get in touch with:

Student Administrator
University of Cambridge
Computer Laboratory
William Gates Building
J J Thomson Avenue
Cambridge
CB3 0FD

telephone: 01223 334656
fax: 01223 334678
e-mail: `undergraduate.admissions@cl.cam.ac.uk`

Michaelmas Term 2010: Part IB lectures

Algorithms II

Lecturer: Dr F.M. Stajano

No. of lectures: 10

Prerequisite courses: Algorithms I

This course is a prerequisite for Computer Graphics and Image Processing, Complexity Theory, Artificial Intelligence I and II.

Aims

The aim of this course is to give further insights into the design and analysis of non-trivial algorithms through the discussion of several complex algorithms in the fields of graphs and computer graphics, which are increasingly critical for a wide range of applications.

Lectures

- **Advanced data structures.** Fibonacci heaps. Disjoint sets. [Ref: Ch 20, 21] [2–3 lectures]
- **Graph algorithms.** Graph representations. Breadth-first and depth-first search. Topological sort. Minimum spanning tree. Kruskal and Prim algorithms. Shortest paths. Bellman–Ford and Dijkstra algorithms. Maximum flow. Ford–Fulkerson method. Matchings in bipartite graphs. [Ref: Ch 22, 23, 24, 25, 26] [5–7 lectures]
- **Geometric algorithms.** Intersection of segments. Convex hull: Graham's scan, Jarvis's march. [Ref: Ch 33] [1–2 lectures]

Objectives

At the end of the course students should

- have a good understanding of how several elaborate algorithms work;
- have a good understanding of how a smart choice of data structures may be used to increase the efficiency of particular algorithms;
- be able to analyse the space and time efficiency of complex algorithms;
- be able to design new algorithms or modify existing ones for new applications and reason about the efficiency of the result.

Recommended reading

* Cormen, T.H., Leiserson, C.D., Rivest, R.L. & Stein, C. (2009). *Introduction to Algorithms*. MIT Press (3rd ed.). ISBN 0-262-53196-8

Sedgewick, R. (2004). *Algorithms in Java* vol. 2 (note that C and C++ editions are also available and are equally good for this course). Addison-Wesley. ISBN 0-201-36121-3. New edition forthcoming in 2008.
Kleinberg, J. & Tardos, É. (2006). *Algorithm design*. Addison-Wesley. ISBN 0-321-29535-8.

Students are expected to buy, make extensive use of, and keep as reference for their future career, one of the above textbooks: those not doing so will be severely disadvantaged. The recommended choice is Cormen *et al.* which, in spite of its superb quality, is the cheapest (about 35 GBP new for over 1300 pages). The pointers in the syllabus are to chapters in the second edition of that book. The other textbooks are all excellent alternatives and their relative merits are discussed in the course handout.

Artificial Intelligence I

Lecturer: Dr S.B. Holden

No. of lectures: 12

Prerequisite courses: Algorithms I. In addition the course requires some mathematics, in particular some use of vectors and some calculus. Part IA Natural Sciences Mathematics or equivalent, and Discrete Mathematics I + II, are likely to be helpful although not essential. Similarly, elements of Algorithms II are likely to be useful.

This course is a prerequisite for the Part II courses Artificial Intelligence II and Natural Language Processing.

Aims

The aim of this course is to provide an introduction to some basic issues and algorithms in artificial intelligence (AI). The course approaches AI from an algorithmic, computer science-centric perspective; relatively little reference is made to the complementary perspectives developed within psychology, neuroscience or elsewhere. The course aims to provide some basic tools and algorithms required to produce AI systems able to exhibit limited human-like abilities, particularly in the form of problem solving by search, representing and reasoning with knowledge, planning, and learning. Historically this corresponds roughly to the era prior to when probability became the standard method for dealing with the crucial concept of *uncertainty*. More recent material on uncertain reasoning is covered in Artificial Intelligence II.

Lectures

- **Introduction.** What is it that we're studying? Why is something that looks so easy to do actually so difficult to compute? Theories and methods: what approaches have been tried? *Agents* as a unifying view of AI systems. The basic structure of an agent. Interaction of an agent with the environment. Assessment of agents. What does this course cover, and what is left out? [1 lecture]
- **Search I.** How can search serve as a fundamental paradigm for intelligent problem-solving? Simple, *uninformed search* algorithms. Tree search and graph

search. More sophisticated *heuristic search* algorithms. The A* algorithm and its properties. Improving memory efficiency: the IDA* and recursive best first search algorithms. [2 lectures]

- **Search II.** Search in an adversarial environment. Computer game playing. The minimax algorithm and its shortcomings. Improving minimax using α-β pruning. [1 lecture]
- **Constraint satisfaction problems (CSPs).** Standardising search problems to a common format. The backtracking algorithm for CSPs. Heuristics for improving the search for a solution. Forward checking, constraint propagation and arc consistency. Backtracking, backjumping using Gaschnig's algorithm, graph-based backjumping. [2 lectures]
- **Knowledge representation and reasoning.** How can we represent and deal with commonsense knowledge and other forms of knowledge? Semantic networks, frames and rules. How can we use inference in conjunction with a knowledge representation scheme to perform reasoning about the world and thereby to solve problems? Inheritance, forward and backward chaining. Knowledge representation using first order logic. The frame, qualification and ramification problems. The situation calculus. [3 lectures]
- **Planning.** Methods for planning in advance how to solve a problem. The STRIPS language. Achieving preconditions, backtracking and fixing threats by promotion or demotion: the partial-order planning algorithm. [1 lecture]
- **Learning.** A brief introduction to supervised learning from examples. Learning as fitting a curve to data. The perceptron. Learning by gradient descent. Multilayer perceptrons and the backpropagation algorithm. [2 lectures]

Objectives

At the end of the course students should:

- Appreciate the distinction between the popular view of the field and the actual research results.
- Appreciate different perspectives on what the problems of artificial intelligence are and how different approaches are justified.
- Be able to design basic problem solving methods based on AI-based search, reasoning, planning, and learning algorithms.

Recommended reading

* Russell, S. & Norvig, P. (2003). *Artificial intelligence: a modern approach*. Prentice Hall (3rd ed.).
Cawsey, A. (1998). *The essence of artificial intelligence*. Prentice Hall.
Poole, D. L. & Mackworth, A. K. (2010). *Artificial intelligence: foundations of computational agents*. Cambridge University Press.

Programming in C and C++

Lecturer: Dr A.W. Moore

No. of lectures: 8

Prerequisite courses: None, though Operating Systems would be helpful.

Aims

The aims of this course are to provide a solid introduction to programming in C and C++ and to provide an overview of the principles and constraints that affect the way in which the C and C++ programming languages have been designed and are used.

Lectures

- **Introduction to the C language.** Background and goals of C. Types and variables. Expressions and statements. Functions. Multiple compilation units. [1 lecture]
- **Further C concepts.** Preprocessor. Pointers and pointer arithmetic. Data structures. Dynamic memory management. Examples. [2 lectures]
- **Introduction to C++.** Goals of C++. Differences between C and C++. References *versus* pointers. Overloading functions. [1 lecture]
- **Objects in C++.** Classes and structs. Operator overloading. Virtual functions. Multiple inheritance. Virtual base classes. Examples. [2 lectures]
- **Further C++ concepts.** Exceptions. Templates and meta-programming. Java Native Interface (JNI). Examples. [2 lectures]

Objectives

At the end of the course students should

- be able to read and write C and C++ programs;
- understand the interaction between C and C++ programs and the host operating system;
- be familiar with the structure of C and C++ program execution in machine memory;
- understand the object-oriented paradigm presented by C++;
- be able to make effective use of templates and meta-programming techniques as used in the STL;
- understand the potential dangers of writing programs in C and C++.

Recommended reading

* Eckel, B. (2000). *Thinking in C++, Vol. 1: Introduction to Standard C++*. Prentice Hall (2nd ed.). Also available at

`http://www.mindview.net/Books/TICPP/ThinkingInCPP2e.html`
Kernighan, B.W. & Ritchie, D.M. (1988). *The C programming language*. Prentice Hall (2nd ed.).
Stroustrup, B. (2008). *Programming — principles and practice using C++*. Addison-Wesley.
Stroustrup, B. (1994). *The design and evolution of C++*. Addison-Wesley.
Lippman, S.B. (1996). *Inside the C++ object model*. Addison-Wesley.

Computer Design

Lecturer: Dr S.W. Moore

No. of lectures: 22 (including 4 via a web-based tutor)

Prerequisite course: Digital Electronics

This course is a prerequisite for the Part II courses Comparative Architectures and System-on-Chip Design.

Aims

The aims of this course are to introduce a hardware description language (Verilog) and computer architecture concepts in order to design computer systems. This is an amalgam of the former ECAD and Computer Design courses.

There are 18 lectures which cover design with hardware description languages, computer architecture and then computer implementation. A web based tutor (equivalent of 4 lectures) is used to teach much of the Verilog hardware description language.

Lectures

- **Introduction and motivation.** Current technology, technology trends, ECAD trends, challenges.
- **Logic modelling, simulation and synthesis.** Logic value and delay modelling. Discrete event and device simulation. Automatic logic minimization.
- **Verilog systems design.** Practicalities of mapping Verilog descriptions of hardware (including a MIPS processor) onto an FPGA board. Introduction of SystemVerilog constructs not covered by IVC. Tips and pitfalls when generating larger modular designs.
- **Chip, board and system testing.** Production testing, fault models, testability, fault coverage, scan path testing, simulation models.
- **Historical perspective on computer architecture.**
- **Early instruction set architecture.** EDSAC *versus* Manchester Mark I.
- **Build your first computer.** Implement a Manchester Baby machine in Java and Verilog.

- **RISC machines.** Introduction to RISC processor design and the MIPS instruction set.
- **MIPS tools and code examples.**
- **CISC machines and the Intel x86 instruction set.**
- **Java Virtual Machine.**
- **Memory hierarchy.** Virtual memory and caching.
- **Simulating a MIPS processor.**
- **Pipelining and data paths.**
- **Implementation of a MIPS processor.** Verilog implementation of a MIPS processor subset.
- **Internal and external communication.**
- **Introduction to many-core processors.**
- **Data-flow machines. Future directions.**

On-Line Learning Component: Interactive Verilog Compiler

- The interactive Verilog compiler (IVC) teaches the synthesizable subset of Verilog which is required to complete the laboratory sessions.

Objectives

At the end of the course students should

- be able to read assembler given a guide to the instruction set and be able to write short pieces of assembler if given an instruction set or asked to invent an instruction set;
- understand the differences between RISC and CISC assembler;
- understand what facilities a processor provides to support operating systems, from memory management to software interrupts;
- understand memory hierarchy including different cache structures;
- appreciate the use of pipelining in processor design;
- understand the communications structures, from buses close to the processor, to peripheral interfaces;
- have an appreciation of control structures used in processor design;
- have an appreciation of how to implement a processor in Verilog.

Recommended reading

* Harris, D.M. & Harris, S.L. (2007). *Digital design and computer architecture: from gates to processors*. Morgan Kaufmann.

Recommended further reading:

Hennessy, J. & Patterson, D. (2006). *Computer architecture: a quantitative approach*. Elsevier (4th ed.). ISBN 978-0-12-370490-0. (Older versions of the book are also still generally relevant.)
Patterson, D.A. & Hennessy, J.L. (2004). *Computer organization and design*. Morgan Kaufmann (3rd ed., as an alternative to the above). (2nd ed., 1998, is also good.)
Pointers to sources of more specialist information are included in the lecture notes and on the associated course web page.

Concurrent and Distributed Systems I

Lecturer: Dr B.N. Shand

No. of lectures: 8

Prerequisite courses: Operating Systems, Programming in Java

This course is a pre-requisite for Mobile and Sensor Systems (Part II).

These eight lectures, together with the eight in the Lent Term, form a single course of 16 lectures.

Aims

The aim of the Concurrent and Distributed Systems course is to introduce concurrency control and distribution concepts and their implications for system design and implementation.

Lectures on Concurrency (Michaelmas Term)

- **Introduction; thread models.** Overview of properties of distributed and concurrent systems. Software system structure. Occurrence of concurrency in systems. Recap of scheduling and preemption. Thread models.

- **Classical concurrency control.** Shared data and critical regions. Mutual exclusion and condition synchronisation. Semaphores. Implementation of concurrency control.

- **Classical problems using semaphores.** Bounded cyclic buffer (producer(s) and consumer(s)), multiple readers and writers. Problems arising in semaphore programming.

- **Concurrency support in programming languages.** Shared data: monitors, pthreads, Java. No shared data: occam, Ada active objects, Erlang, Kilim, tuple spaces. Lock-free programming.

- **Concurrent composite operations.** Composite operations in main memory and persistent memory. Dynamic resources allocation and deadlock. Dining philosophers program. Deadlock detection and avoidance.
- **Transactions.** ACID properties. Concurrency control and crash recovery. Definition of conflicting operations. Serialisation. Cascading aborts.
- **Database concurrency control.** Pessimistic concurrency control: two-phase locking, timestamp ordering. Optimistic concurrency control.
- **Database recovery and summary of "Concurrency".** Write ahead log, undo/redo. Points to take forward.

Objectives

At the end of the course students should

- understand the need for concurrency control in operating systems and applications, both mutual exclusion and condition synchronisation;
- understand how multi-threading can be supported and the implications of different approaches;
- be familiar with the support offered by various programming languages for concurrency control and be able to judge the scope, performance implications and possible applications of the various approaches;
- be aware that dynamic resource allocation can lead to deadlock
- understand the concept of transaction; the properties of transactions, how concurrency control can be assured and how transactions can be distributed;
- understand the fundamental properties of distributed systems and their implications for system design;
- understand the effects of large scale on the provision of fundamental services and the tradeoffs arising from scale;
- be familiar with a range of distributed algorithms.

Recommended reading

* Bacon, J. & Harris, T. (2003). *Operating systems: distributed and concurrent software design*. Addison-Wesley.
Bacon, J. (1997). *Concurrent Systems*. Addison-Wesley.
Tanenbaum, A.S. & van Steen, M. (2002). *Distributed systems*. Prentice Hall.
Coulouris, G.F., Dollimore, J.B. & Kindberg, T. (2005, 2001). *Distributed systems, concepts and design*. Addison-Wesley (4th, 3rd eds.).

Further Java

Lecturers: Dr A.R. Beresford and Dr A.C. Rice

No. of practical classes: 5 $\times$ 2*-hour sessions*

Prerequisite course: Programming in Java, Further Java Briefing

Companion courses: Concurrent and Distributed Systems

This course is a prerequisite for the Group Project.

Aims

The goal of this course is to provide students with the ability to understand the advanced programming features available in the Java programming language, completing the coverage of the language started in the Programming in Java course. The course is designed to accommodate students with diverse programming backgrounds; consequently Java is taught from first principles in a practical class setting where students can work at their own pace from a course handbook. Each practical class will culminate in an assessed exercise.

Practical classes

- **Communication and client applications.** This class will introduce the Eclipse development environment. Students will write a simple client to send and receive data to a server via TCP.
- **Serialisation, reflection and class loaders.** This class will introduce object serialisation. Students will use a class loader and reflection to inspect an object which is only available at run-time.
- **Concurrency and synchronisation.** This class introduces the concurrency and synchronisation primitives found in Java. Students will implement a thread-safe first-in-first-out queue and learn about Java generics.
- **Server applications**. Students implement a server in Java which is capable of communicating concurrently with mulitple clients.
- **Databases**. This week students will use Java annotations and a relational database to build a persistent store.

Objectives

At the end of the course students should

- understand different mechanisms for communication between distributed applications and be able to evaluate their trade-offs;
- be able to use Java generics and annotations to improve software usability, readability and safety;
- understand and be able to exploit the Java class-loading mechansim;

- understand and be able to use concurrency control correctly;
- understand the concept of transactions and their application in a range of systems.

Recommended reading

* Lea, D. (1999). *Concurrent programming in Java*. Addison-Wesley (2nd ed.).
Bracha, G., Gosling, J., Joy, B. & Steele, G. (2000). *The Java language specification*. Addison-Wesley (2nd ed.).
`http://java.sun.com/docs/books/jls/`
Bacon, J. & Harris, T. (2003). *Operating systems* or Bacon, J. (1997) *Concurrent systems* (2nd ed.). Addison-Wesley.

Group Project

Lecturer: Dr A.F. Blackwell

No. of lectures: 3

Prerequisite courses: Software Design, Software Engineering, Further Java

Aims

The aim of this course is to give students a realistic introduction to software development as practised in industry. This means working to rigid deadlines, with a team of colleagues not of one's own choosing, having to satisfy an external client that a design brief has been properly interpreted and implemented, all within the constraints of limited effort and technical resources.

Lectures

- **Initial project briefing.** Software engineering: design, quality and management, application of course material. Introduction to possible design briefs. Formation of groups, selection of tools, review meetings.
- **Administrative arrangements.** Announcement of group members. Deliverables: functional specification and module design, module implementation and testing, system integration, testing and documentation. Timetable. Advice on specific tools. First project meeting.
- **Presentation techniques.** Public speaking techniques and the effective use of audio-visual aids. Planning a talk; designing a presentation; common mistakes to avoid.

Objectives

At the end of the course students should

- have a good understanding of how software is developed;

- have consolidated the theoretical understanding of software development acquired in the Software Design course;
- appreciate the importance of planning and controlling a project, and of documentation and presentation;
- have gained confidence in their ability to develop significant software projects and Part IB students should be prepared for the personal project they will undertake in Part II.

Logic and Proof

Lecturer: Dr A.R. Coble

No. of lectures: 12

This course is a prerequisite for the Part II courses Artificial Intelligence II, Hoare Logic, Temporal Logic and Natural Language Processing.

Aims

This course will teach logic, especially the predicate calculus. It will present the basic principles and definitions, then describe a variety of different formalisms and algorithms that can be used to solve problems in logic. Putting logic into the context of Computer Science, the course will show how the programming language Prolog arises from the automatic proof method known as resolution. It will introduce topics that are important in mechanical verification, such as binary decision diagrams (BDDs), SAT solvers and modal logic.

Lectures

- **Introduction to logic.** Schematic statements. Interpretations and validity. Logical consequence. Inference.
- **Propositional logic.** Basic syntax and semantics. Equivalences. Normal forms. Tautology checking using CNF.
- **The sequent calculus.** A simple (Hilbert-style) proof system. Natural deduction systems. Sequent calculus rules. Sample proofs.
- **First order logic.** Basic syntax. Quantifiers. Semantics (truth definition).
- **Formal reasoning in FOL.** Free *versus* bound variables. Substitution. Equivalences for quantifiers. Sequent calculus rules. Examples.
- **Clausal proof methods.** Clause form. A SAT-solving procedure. The resolution rule. Examples. Refinements.
- **Skolem functions and Herbrand's theorem.** Prenex normal form. Skolemisation. Herbrand models and their properties.

- **Unification.** Composition of substitutions. Most general unifiers. A unification algorithm. Applications and variations.
- **Prolog.** Binary resolution. Factorisation. Example of Prolog execution. Proof by model elimination.
- **Binary decision diagrams.** General concepts. Fast canonical form algorithm. Optimisations. Applications.
- **Modal logics.** Possible worlds semantics. Truth and validity. A Hilbert-style proof system. Sequent calculus rules.
- **Tableaux methods.** Simplifying the sequent calculus. Examples. Adding unification. Skolemisation. The world's smallest theorem prover?

Objectives

At the end of the course students should

- be able to manipulate logical formulas accurately;
- be able to perform proofs using the presented formal calculi;
- be able to construct a small BDD;
- understand the relationships among the various calculi, e.g. SAT solving, resolution and Prolog;
- be able to apply the unification algorithm and to describe its uses.

Recommended reading

* Huth, M. & Ryan, M. (2004). *Logic in computer science: modelling and reasoning about systems*. Cambridge University Press (2nd ed.).
Ben-Ari, M. (2001). *Mathematical logic for computer science*. Springer (2nd ed.).

Mathematical Methods for Computer Science

Lecturer: Dr R.J. Gibbens

No. of lectures: 12

Prerequisite course: Probability

This course is a prerequisite for Computer Graphics and Image Processing (Part IB) and the following Part II courses: Artificial Intelligence II, Bioinformatics, Computer Systems Modelling, Computer Vision, Digital Signal Processing, Information Theory and Coding, Quantum Computing.

Aims

The aim of this course is to introduce and develop mathematical methods that are key to many modern applications in Computer Science. The course proceeds on two fronts: (*i*) Fourier methods and their generalizations that lie at the heart of modern digital signal processing, coding and information theory and (*ii*) probability modelling techniques that allow stochastic systems and algorithms to be described and better understood. The style of the course is necessarily concise but will attempt to blend a mix of theory with examples that glimpse ahead at applications developed in Part II courses.

Lectures

- **Fourier methods.** Inner product spaces and orthonormal systems. Periodic functions and Fourier series. Results and applications. The Fourier transform and its properties. [3 lectures]
- **Discrete Fourier methods.** The Discrete Fourier transform and related algorithms and applications. [2 lectures]
- **Wavelets.** Introduction to wavelets with computer science applications. [1 lecture]
- **Inequalities and limit theorems.** Bounds on tail probabilities, moment generating functions, notions of convergence, weak and strong laws of large numbers, the central limit theorem, statistical applications, Monte Carlo simulation. [3 lectures]
- **Markov chains.** Discrete-time Markov chains, Chapman–Kolmogorov equations, classifications of states, limiting and stationary behaviour, time-reversible Markov chains. Examples and applications. [3 lectures]

Objectives

At the end of the course students should

- understand the fundamental properties of inner product spaces and orthonormal systems;
- grasp key properties and uses of Fourier series and transforms, and wavelets;
- understand discrete transform techniques and their applications;
- understand basic probabilistic inequalities and limit results and be able to apply them to commonly arising models;
- be familiar with the fundamental properties and uses of discrete-time Markov chains.

Reference books

* Pinkus, A. & Zafrany, S. (1997). *Fourier series and integral transforms*. Cambridge University Press.
* Ross, S.M. (2002). *Probability models for computer science*. Harcourt/Academic Press.

Mitzenmacher, M. & Upfal, E. (2005). *Probability and computing: randomized algorithms and probabilistic analysis*. Cambridge University Press.
Oppenheim, A.V. & Willsky, A.S. (1997). *Signals and systems*. Prentice Hall.

Prolog

Lecturer: Dr D.M. Eyers

No. of lectures: 6

Prerequisite courses: Foundations of Computer Science, Algorithms I and Logic & Proof

Aims

The aim of this course is to introduce programming in the Prolog language. Prolog encourages a different programming style to Java or ML and particular focus is placed on programming to solve real problems that are suited to this style. Practical experimentation with the language is strongly encouraged.

Lectures

- **Introduction to Prolog.** The structure of a Prolog program and how to use the Prolog interpreter. Unification revisited. Some simple programs.
- **Arithmetic and lists.** Prolog's support for evaluating arithmetic expressions and lists. The space complexity of program evaluation discussed with reference to last-call optimisation.
- **Backtracking, cut, and negation.** The `cut` operator for controlling backtracking. *Negation as failure* and its uses.
- **Search and cut.** Prolog's search method for solving problems. Graph searching exploiting Prolog's built-in search mechanisms.
- **Difference structures.** Difference lists: introduction and application to example programs.
- **Building on Prolog.** How particular limitations of Prolog programs can be addressed by techniques such as Constraint Logic Programming (CLP) and tabled resolution.

Objectives

At the end of the course students should

- be able to write programs in Prolog using techniques such as accumulators and difference structures;
- know how to model the backtracking behaviour of program execution;

- appreciate the unique perspective Prolog gives to problem solving and algorithm design;
- understand how larger programs can be created using the basic programming techniques used in this course.

Recommended reading

* Bratko, I. (2001). *PROLOG programming for artificial intelligence*. Addison-Wesley (3rd ed).

Software Engineering

Lecturer: Professor R.J. Anderson

No. of lectures: 6

This course is a prerequisite for the Group Project.

Aims

This course aims to introduce students to software engineering, and in particular to the problems of building large systems, safety-critical systems and real-time systems. Case histories of software failure are used to illustrate what can go wrong, and current software engineering practice is studied as a guide to how failures can be avoided.

Lectures

- **The software crisis.** Examples of large-scale project failure, such as the London Ambulance Service system and the NHS National Programme for IT. Intrinsic difficulties with software.
- **The software life cycle.** Getting the requirements right; requirements analysis methods; modular design; the role of prototyping; the waterfall, spiral and evolutionary models.
- **Critical systems.** Examples of catastrophic failure; particular problems with real-time systems; usability and human error; verification and validation.
- **Quality assurance.** The contribution of reviews and testing; reliability growth models; software maintenance and configuration management; life-cycle costs.
- **Tools.** The effect of high-level languages; object-oriented systems and object reuse; an overview of formal methods with some application examples; project planning tools; automated testing tools.
- **Large software systems.** The role of application domain knowledge; changing requirements; risk reduction *versus* due diligence; communications failure; organizational factors.

Objectives

At the end of the course students should know how writing programs with tough assurance targets, in large teams, or both, differs from the programming exercises they have engaged in so far. They should appreciate the waterfall, spiral and evolutionary models of software development and be able to explain which kinds of software project might profitably use them. They should appreciate the value of other tools and the difference between incidental and intrinsic complexity. They should understand the software development life cycle and its basic economics. They should be prepared for the organizational aspects of their Part IB group project.

Recommended reading

* Pressman, R.S. (1994). *Software engineering*. McGraw-Hill.
Leveson, N. (1994). *Safeware*. Addison-Wesley.
Maguire, S. (1993). *Writing solid code*. Microsoft Press.

Further reading:

Brooks, F.P. (1975). *The mythical man month*. Addison-Wesley.
Reason, J. (2008). *The human contribution*. Ashgate Publishing.
Leveson, N. (2008). *System safety engineering: back to the future*, available at `http://sunnyday.mit.edu/book2.pdf`
Neumann, P. (1994). *Computer-related risks*. ACM Press.
Report of the inquiry into the London Ambulance Service (SW Thames RHA, 40 Eastbourne Terrace, London W2 3QR, February 1993).
`http://www.cs.ucl.ac.uk/staff/A.Finkelstein/las.html`
Anderson, R. (2008). *Security engineering* (Chapters 25 and 26). Wiley. Alternatively see 2001 edition, Chapters 22 and 23, available at
`http://www.cl.cam.ac.uk/users/rja14/book.html`

Unix Tools

Lecturer: Dr M.G. Kuhn

No. of lectures: 10

Operating Systems provides a useful foundation for this course.

Aims

This non-examinable course provides students with basic Unix/Linux experience some important practical skills in using the Unix shell as an efficient working environment. It also introduces some popular software-engineering tools for working in teams, as well as formatting and data-analysis tools for preparing dissertations and scientific publications. These skills are essential not only for future practical CST projects, but for participating effectively in most real-world software projects.

Lectures

- **Unix concepts.** Brief review of Unix history and design philosophy, documentation, terminals, inter-process communication mechanisms and conventions, shell, command-line arguments, environment variables, file descriptors.
- **Shell basics.** Program invocation, redirecting standard I/O, pipes, file-system navigation, argument expansion, quoting, job control, signals, process groups, variables, locale.
- **Shell script programming and configuration.** Efficient command entry with history and alias functions. Scripts, plain-text file formats, control structures, functions. Customizing user environments. Basics of *X Window System* configuration. Some notes on PWF Linux.
- **Common tools.** Overview of common text, shell, and network utilities and their most frequently used options, including sed, grep, chmod, find, ssh, rsync, packaging and compression tools.
- **Revision control systems.** diff, patch, RCS, Subversion, git.
- **Software development tools.** C compiler, linker and debugger, makefiles.
- **Perl.** Introduction to a powerful scripting and text manipulation language. [2 lectures]
- **LaTeX.** Typesetting basics, introduction to the most popular tool for scientific document formatting.
- **Number crunching and data visualization.** Use of MATLAB on PWF machines.

Objectives

At the end of the course students should

- be confident in performing routine user tasks on a POSIX system, understand command-line user-interface conventions and know how to find more detailed documentation;
- appreciate how a range of simple tools can be combined with little effort in pipes and scripts to perform a large variety of tasks;
- be familiar with the most common tools, file formats and configuration practices;
- be able to understand, write, and maintain shell scripts and makefiles;
- appreciate how using revision control systems and fully automated build processes help to maintain reproducibility and audit trails during software development;
- know enough about basic development tools to be able to install and modify C source code;

- have gained experience in using Perl, LaTeX and MATLAB.

Recommended reading

* Lamport, L. (1994). *LaTeX – a documentation preparation system user's guide and reference manual*. Addison-Wesley (2nd ed.).
Robbins, A. (2005). *Unix in a nutshell*. O'Reilly (4th ed.).
Schwartz, R.L. & Phoenix, T. (2005). *Learning Perl*. O'Reilly (4th ed.).

Lent Term 2011: Part IB lectures

Compiler Construction

Lecturer: Dr T.G. Griffin

No. of lectures: 16

Prerequisite: (the last lecture of) Regular Languages and Finite Automata (Part IA)

This course is a prerequisite for Optimising Compilers (Part II).

Aims

This course aims to cover the main technologies associated with implementing programming languages, viz. lexical analysis, syntax analysis, type checking, run-time data organisation and code-generation.

Lectures

- **Survey of execution mechanisms.** The spectrum of interpreters and compilers; compile-time and run-time. Structure of a simple compiler. Java virtual machine (JVM), JIT. Simple run-time structures (stacks). Structure of interpreters for result of each stage of compilation (tokens, tree, bytecode). [3 lectures]
- **Lexical analysis and syntax analysis.** Recall regular expressions and finite state machine acceptors. Lexical analysis: hand-written and machine-generated. Recall context-free grammars. Ambiguity, left- and right-associativity and operator precedence. Parsing algorithms: recursive descent and machine-generated. Abstract syntax tree; expressions, declarations and commands. [2 lectures]
- **Simple type-checking.** Type of an expression determined by type of subexpressions; inserting coercions. [1 lecture]
- **Translation phase.** Translation of expressions, commands and declarations. [1 lecture]
- **Code generation.** Typical machine codes. Code generation from intermediate code. Simple peephole optimisation. [1 lecture]
- **Object Modules and Linkers.** Resolving external references. Static and dynamic linking. [1 lecture]
- **Non-local variable references.** Lambda-calculus as prototype, Landin's principle of correspondence. Problems with `rec` and class variables. Environments, function values are closures. Static and Dynamic Binding (Scoping). [1 lecture]
- **Machine implementation of a selection of interesting things.** Free variable treatment, static and dynamic chains, ML free variables. Compilation as source-to-source simplification, e.g. closure conversion. Argument passing mechanisms. Objects and inheritance; implementation of methods. Labels, `goto`

and exceptions. Dynamic and static typing, polymorphism. Storage allocation, garbage collection. [3 lectures]

- **Parser Generators.** A user-level view of Lex and Yacc. [1 lecture]
- **Parsing theory and practice.** Phrase Structured Grammars. Chomsky classification. LL(k) and LR(k) parsing. How tools like Yacc generate parsers, and their error messages. [2 lectures]

Objectives

At the end of the course students should understand the overall structure of a compiler, and will know significant details of a number of important techniques commonly used. They will be aware of the way in which language features raise challenges for compiler builders.

Recommended reading

* Appel, A. (1997). *Modern compiler implementation in Java/C/ML* (3 editions). Cambridge University Press.
Aho, A.V., Sethi, R. & Ullman, J.D. (2007). *Compilers: principles, techniques and tools*. Addison-Wesley (2nd ed.).
Bennett, J.P. (1990). *Introduction to compiling techniques: a first course using ANSI C, LEX and YACC*. McGraw-Hill.
Bornat, R. (1979). *Understanding and writing compilers*. Macmillan.
Fischer, C.N. & LeBlanc, J. Jr (1988). *Crafting a compiler*. Benjamin/Cummings.
Watson, D. (1989). *High-level languages and their compilers*. Addison-Wesley.

Computation Theory

Lecturer: Professor A. Dawar

No. of lectures: 12

Prerequisite course: Discrete Mathematics

This course is a prerequisite for Complexity Theory (Part IB), Quantum Computing (Part II).

Aims

The aim of this course is to introduce several apparently different formalisations of the informal notion of algorithm; to show that they are equivalent; and to use them to demonstrate that there are uncomputable functions and algorithmically undecidable problems.

Lectures

- **Introduction: algorithmically undecidable problems.** Decision problems. The informal notion of algorithm, or effective procedure. Examples of algorithmically undecidable problems. [1 lecture]

- **Register machines.** Definition and examples; graphical notation. Register machine computable functions. Doing arithmetic with register machines. [1 lecture]
- **Universal register machine.** Natural number encoding of pairs and lists. Coding register machine programs as numbers. Specification and implementation of a universal register machine. [2 lectures]
- **Undecidability of the halting problem.** Statement and proof. Example of an uncomputable partial function. Decidable sets of numbers; examples of undecidable sets of numbers. [1 lecture]
- **Turing machines.** Informal description. Definition and examples. Turing computable functions. Equivalence of register machine computability and Turing computability. The Church-Turing Thesis. [2 lectures]
- **Primitive and partial recursive functions.** Definition and examples. Existence of a recursive, but not primitive recursive function. A partial function is partial recursive if and only if it is computable. [2 lectures]
- **λ-Calculus.** Alpha and beta conversion. Normalization. Encoding data. Writing recursive functions in the λ-calculus. The relationship between computable functions and λ-definable functions. [3 lectures]

Objectives

At the end of the course students should

- be familiar with the register machine, Turing machine and λ-calculus models of computability;
- understand the notion of coding programs as data, and of a universal machine;
- be able to use diagonalisation to prove the undecidability of the Halting Problem;
- understand the mathematical notion of partial recursive function and its relationship to computability.

Recommended reading

* Hopcroft, J.E., Motwani, R. & Ullman, J.D. (2001). *Introduction to automata theory, languages, and computation*. Addison-Wesley (2nd ed.).
* Hindley, J.R. & Seldin, J.P. (2008). *Lambda-calculus and combinators, an introduction*. Cambridge University Press (2nd ed.).
Cutland, N.J. (1980). *Computability: an introduction to recursive function theory*. Cambridge University Press.
Davis, M.D., Sigal, R. & Weyuker, E.J. (1994). *Computability, complexity and languages*. Academic Press (2nd ed.).
Sudkamp, T.A. (2005). *Languages and machines*. Addison-Wesley (3rd ed.).

Computer Graphics and Image Processing

Lecturers: Professor N.A. Dodgson and Professor P. Robinson

No. of lectures: 16

Prerequisite courses: Algorithms, Mathematical Methods for Computer Science (for one lecture of Image Processing part of the course)

This course is a prerequisite for Advanced Graphics (Part II).

Aims

To introduce the necessary background, the basic algorithms, and the applications of computer graphics and image processing. A large proportion of the course considers the design and optimisation of algorithms, so can be considered a practical application of the lessons learnt in the *Algorithms* course.

Lectures

- **Background.** What is an image? What are computer graphics, image processing, and computer vision? How do they relate to one another? Image capture. Image display. Human vision. Resolution and quantisation. Colour and colour spaces. Storage of images in memory, and double buffering. Display devices: the inner workings of CRTs, LCDs, and printers. [3 lectures]
- **2D Computer graphics.** Drawing a straight line. Drawing circles and ellipses. Cubic curves: specification and drawing. Clipping lines. Filling polygons. Clipping polygons. 2D transformations, vectors and matrices, homogeneous co-ordinates. Uses of 2D graphics: HCI, typesetting, graphic design. [5 lectures]
- **3D Computer graphics.** Projection: orthographic and perspective. 3D transforms and matrices. 3D clipping. 3D curves. 3D scan conversion. *z*-buffer. A-buffer. Ray tracing. Lighting: theory, flat shading, Gouraud, Phong. Texture mapping. [5 lectures]
- **Image processing.** Operations on images: filtering, point processing, compositing. Halftoning and dithering, error diffusion. Encoding and compression: difference encoding, predictive, run length, transform encoding (including JPEG). [3 lectures]

Objectives

At the end of the course students should be able to

- explain the basic function of the human eye and how this impinges on resolution, quantisation, and colour representation for digital images; describe a number of colour spaces and their relative merits; explain the workings of cathode ray tubes, liquid crystal displays, and laser printers;
- describe and explain the following algorithms: Bresenham's line drawing, mid-point line drawing, mid-point circle drawing, Bezier cubic drawing, Douglas and Pucker's line chain simplification, Cohen–Sutherland line clipping, scanline polygon fill, Sutherland–Hodgman polygon clipping, depth sort, binary space partition tree, *z*-buffer, A-buffer, ray tracing, error diffusion;

- use matrices and homogeneous coordinates to represent and perform 2D and 3D transformations; understand and use 3D to 2D projection, the viewing volume, and 3D clipping;
- understand Bezier curves and patches; understand sampling and super-sampling issues; understand lighting techniques and how they are applied to both polygon scan conversion and ray tracing; understand texture mapping;
- explain how to use filters, point processing, and arithmetic operations in image processing and describe a number of examples of the use of each; explain how halftoning, ordered dither, and error diffusion work; understand and be able to explain image compression and the workings of a number of compression techniques.

Recommended reading

* Foley, J.D., van Dam, A., Feiner, S.K. & Hughes, J.F. (1990). *Computer graphics: principles and practice*. Addison-Wesley (2nd ed.).
Gonzalez, R.C. & Woods, R.E. (1992). *Digital image processing*. Addison-Wesley.
[Gonzalez, R.C. & Wintz, P. (1977). *Digital image processing* is the earlier edition and is almost as useful.]
* Slater, M., Steed, A. & Chrysanthou, Y. (2002). *Computer graphics and virtual environments: from realism to real-time*. Addison-Wesley.

Computer Networking

Lecturer: Dr A.W. Moore

No. of lectures: 24

This course is a prerequisite for the Part II courses Principles of Communication and Security II.

Aims

The aim of this course is to introduce key concepts and principles of computer networks. The course will use a top-down approach to study of the Internet and its protocol stack. Instances of architecture, protocol, application-examples will include email, web and media-streaming. We will cover communications services (e.g., TCP/IP) required to support such network applications. The implementation and deployment of communications services in practical networks: including wired and wireless LAN environments, will be followed by a discussion of issues of network-security and network-management, Throughout the course, the Internet's architecture and protocols will be used as the primary examples to illustrate the fundamental principles of computer networking.

Lectures

- **Introduction.** Overview of networking using the Internet as an example. LANs and WANs. OSI reference model, Internet TCP/IP Protocol Stack. Client/server

paradigm, circuit-switching, packet-switching, Internet structure, networking delays and packet loss. [3 lectures]

- **Application layer.** Service requirements, WWW, HTTP, electronic mail, Domain Name System, P2P, socket programming API. [3 lectures]
- **Transport layer.** Service models, multiplexing/demultiplexing, connection-less transport (UDP), principles of reliable data transfer, connection-oriented transport (TCP), TCP congestion control, TCP variants. [3 lectures]
- **Network layer addressing.** Network layer services, IP, IP addressing, IPv4, DHCP, NAT, ICMP, IPv6. [3 lectures]
- **Network layer routing.** Routing and forwarding, routing algorithms, routing in the Internet, RIP, OSPF, BGP, multicast. [3 lectures]
- **Link layer and local area networks.** Link layer services, error detection and correction, Multiple Access Protocols, link layer addressing, Ethernet, hubs and switches, Point-to-Point Protocol. [3 lectures]
- **Wireless and mobile networks.** Wireless links and network characteristics, Wi-Fi: IEEE 802.11 wireless LANs, mobility management and mobile IP. [2 lectures]
- **Multimedia networking.** Networked multimedia applications, best-effort service and multimedia delivery requirements, multimedia protocols (RTSP, RTP, RTCP, SIP), content distribution networks. [3 lectures]
- **Network security and network management.** Cryptography, integrity, securing email, securing TCP (SSL), firewalls and IDS, network management components, Internet management framework, presentation services. [1 lecture]

Objectives

At the end of the course students should

- be able to analyse a communication system by separating out the different functions provided by the network;
- understand that there are fundamental limits to any communications system;
- understand the general principles behind multiplexing, addressing, routing, reliable transmission and other stateful protocols as well as specific examples of each;
- understand what FEC is and how CRCs work;
- be able to compare communications systems in how they solve similar problems;
- have an informed view of both the internal workings of the Internet and of a number of common Internet applications and protocols.

Recommended reading

* Kurose, J.F. & Ross, K.W. (2009). *Computer networking: a top-down approach*. Addison-Wesley (5th ed.).
Peterson, L.L. & Davie, B.S. (2007). *Computer networks: a systems approach*. Morgan Kaufmann (4th ed.).
Comer, D. & Stevens, D. (2005). *Internetworking with TCP-IP, vol.* 1 *and* 2. Prentice Hall (5th ed.).
Stevens, W.R., Fenner, B. & Rudoff, A.M. (2003). *UNIX network programming, Vol.I: The sockets networking API*. Prentice Hall (3rd ed.).

Concurrent and Distributed Systems II

Lecturer: Dr D. Evans

No. of lectures: 8

Prerequisite courses: Concurrent and Distributed Systems I, Operating Systems

These eight lectures, together with the eight in the Michaelmas Term, form a single course of 16 lectures.

Aims

The aims of this course are to study the fundamental characteristics of distributed systems, including their models and architectures; the implications for software design; some of the techniques that have been used to build them; and the resulting details of good distributed algorithms and applications.

Lectures on Distributed Systems (Lent Term)

- **Introduction, Evolution, Architecture.** Fundamental properties. Evolution from LANs. Introduction to the need for naming, authentication, policy specification and enforcement. Examples of multi-domain systems. Why things can get difficult quickly. Enough Erlang to understand subsequent examples.

- **Time and event ordering.** Time, clocks and event ordering. Earth time, computer clocks, clock drift, clock synchronisation. Order imposed by inter-process communication. Timestamps point/interval. Event composition; uncertainty of ordering, failure and delay.
  Process groups: open/closed, structured/unstructured. Message delivery ordering: arrival order; causal order (vector clocks); total order. Physical causality from real-world examples.

- **Consistency and commitment.** Strong and weak consistency. Replica management. Quorum assembly. Distributed transactions. Distributed concurrency control: two-phase locking, timestamp ordering. Atomic commitment; two-phase commit protocol. Distributed optimistic concurrency control and commitment. Some algorithm outlines: Election of a leader. Distributed mutual exclusion.

- **Middleware.** Synchronous: RPC, object-orientated. Asynchronous: message orientated, publish/subscribe, peer-to-peer. Event-based systems. Examples of some simple distributed programs in Java and Erlang.
- **Naming and name services.** Unique identifiers, pure and impure names. Name spaces, naming domains, name resolution. Large scale name services: DNS, X.500/LDAP, GNS. Use of replication. Consistency-availability tradeoffs. Design assumptions and future issues.
- **Access control for multi-domain distributed systems.** Requirements from healthcare, police, emergency services, globally distributed companies. ACLs, capabilities, Role-Based Access Control (RBAC). Context aware access control. Examples: OASIS, CBCL OASIS, Microsoft Healthvault, . . . Authentication and authorisation: Raven, Shibboleth, OpenID.
- **Distributed storage services. Summary and roundup.** Network-based storage services. Naming and access control. Peer-to-peer protocols. Content distribution. Summary and roundup. Open problems for future years: transactional main memory; multicore concurrency control; untrusted components. Byzantine failure.

Objectives

At the end of the course students should

- understand the need for concurrency control in operating systems and applications, both mutual exclusion and condition synchronisation;
- understand how multi-threading can be supported and the implications of different approaches;
- be familiar with the support offered by various programming languages for concurrency control and be able to judge the scope, performance implications and possible applications of the various approaches;
- be aware that dynamic resource allocation can lead to deadlock;
- understand the concept of transaction; the properties of transactions, how concurrency control can be assured and how transactions can be distributed;
- understand the fundamental properties of distributed systems and their implications for system design;
- understand the effects of large scale on the provision of fundamental services and the tradeoffs arising from scale;
- be familiar with a range of distributed algorithms.

Recommended reading

* Bacon, J. & Harris, T. (2003). *Operating systems: distributed and concurrent software design*. Addison-Wesley.

Bacon, J. (1997). *Concurrent Systems*. Addison-Wesley.
Tanenbaum, A.S. & van Steen, M. (2002). *Distributed systems*. Prentice Hall.
Coulouris, G.F., Dollimore, J.B. & Kindberg, T. (2005, 2001). *Distributed systems, concepts and design*. Addison-Wesley (4th, 3rd eds.).

Security I

Lecturer: Dr M.G. Kuhn

No. of lectures: 12

Prerequisite courses: Discrete Mathematics II, Operating Systems

This course is a prerequisite for Security II.

Aims

This course covers essential concepts of computer security and cryptography.

Lectures

- **Cryptography.** Introduction, terminology, finite rings and fields, modular arithmetic, GF(2^n), pseudo-random functions and permutations.
- **Classic ciphers.** Vigenére, perfect secrecy, Vernam, computational security, Kerckhoffs' principle, random bit sources.
- **Stream ciphers.** Attacking linear-congruential RNGs and LFSRs, Trivium, RC4.
- **Block ciphers.** SP networks, Feistel/Luby–Rackoff structure, DES, AES, modes of operation, message authentication codes.
- **Secure hash functions.** One-way functions, collision resistance, Merkle–Damgård construction, padding, birthday problem, MD5, SHA, HMAC, stream authentication, Merkle tree, Lamport one-time signatures.
- **Asymmetric cryptography.** Key-management problem, signatures, certificates, PKI, discrete-logarithm problem, Diffie–Hellman key exchange, ElGamal encryption and signature, hybrid cryptography.
- **Entity authentication.** Passwords, trusted path, phishing, CAPTCHA. Authentication protocols: one-way and challenge–response protocols, Needham–Schroeder, protocol failure examples, hardware tokens.
- **Access control.** Discretionary access control matrix, DAC in POSIX and Windows, elevated rights and setuid bits, capabilities, mandatory access control, covert channels, Clark–Wilson integrity.
- **Operating system security.** Trusted computing base, domain separation, reference mediation, residual information protection.

- **Software security.** Malicious software, viruses. Common implementation vulnerabilities: buffer overflows, integer overflows, meta characters, syntax incompatibilities, race conditions, unchecked values, side channels.
- **Network security.** Vulnerabilities of TCP/IP, DNS. HTTP authentication, cookies, cross-site scripting, browser sandboxes. Firewalls, VPNs.
- **Security policies and management.** Application-specific security requirements, targets and policies, security management, BS 7799.

Objectives

By the end of the course students should

- be familiar with core security terms and concepts;
- have a basic understanding of some commonly used attack techniques and protection mechanisms;
- have gained basic insight into aspects of modern cryptography and its applications;
- appreciate the range of meanings that “security” has across different applications.

Recommended reading

* Paar, Ch. & Pelzl, J. (2010). *Understanding cryptography*. Springer.
Gollmann, D. (2006). *Computer security*. Wiley (2nd ed.).

Further reading:

Anderson, R. (2008). *Security engineering*. Wiley (2nd ed.).
Stinson, D. (2005). *Cryptography: theory and practice*. Chapman & Hall/CRC (3rd ed.).
Cheswick, W.R., Bellovin, S.M. & Rubin, A.D. (2003). *Firewalls and Internet security: repelling the wily hacker*. Addison-Wesley (2nd ed.).
Garfinkel, S., Spafford, G. & Schwartz, A. (2003). *Practical Unix and Internet security*. O’Reilly (3nd ed.).

Semantics of Programming Languages

Lecturer: Dr S. Staton

No. of lectures: 12

This course is a prerequisite for the Part II courses Topics in Concurrency, and Types.

Aims

The aim of this course is to introduce the structural, operational approach to programming language semantics. It will show how to specify the meaning of typical programming language constructs, in the context of language design, and how to reason formally about semantic properties of programs.

Lectures

- **Introduction.** Transition systems. The idea of structural operational semantics. Transition semantics of a simple imperative language. Language design options.
- **Types.** Introduction to formal type systems. Typing for the simple imperative language. Statements of desirable properties.
- **Induction.** Review of mathematical induction. Abstract syntax trees and structural induction. Rule-based inductive definitions and proofs. Proofs of type safety properties.
- **Functions.** Call-by-name and call-by-value function application, semantics and typing. Local recursive definitions.
- **Data.** Semantics and typing for products, sums, records, references.
- **Subtyping.** Record subtyping and simple object encoding.
- **Semantic equivalence.** Semantic equivalence of phrases in a simple imperative language, including the congruence property. Examples of equivalence and non-equivalence.
- **Concurrency.** Shared variable interleaving. Semantics for simple mutexes; a serializability property.
- **Low-level semantics.** Monomorphic typed assembly language.

Objectives

At the end of the course students should

- be familiar with rule-based presentations of the operational semantics and type systems for some simple imperative, functional and interactive program constructs;
- be able to prove properties of an operational semantics using various forms of induction (mathematical, structural, and rule-based);
- be familiar with some operationally-based notions of semantic equivalence of program phrases and their basic properties.

Recommended reading

Hennessy, M. (1990). *The semantics of programming languages*. Wiley. Out of print, but available on the web at
`http://www.cogs.susx.ac.uk/users/matthewh/semnotes.ps.gz`
* Pierce, B.C. (2002). *Types and programming languages*. MIT Press.
Winskel, G. (1993). *The formal semantics of programming languages*. MIT Press.

Easter Term 2011: Part IB lectures

Complexity Theory

Lecturer: Professor A. Dawar

No. of lectures: 12

Prerequisite courses: Algorithms, Computation Theory

Aims

The aim of the course is to introduce the theory of computational complexity. The course will explain measures of the complexity of problems and of algorithms, based on time and space used on abstract models. Important complexity classes will be defined, and the notion of completeness established through a thorough study of NP-completeness. Applications to cryptography will be considered.

Lectures

- **Algorithms and problems.** Complexity of algorithms and of problems. Lower and upper bounds. Examples: sorting and travelling salesman.
- **Time and space.** Models of computation and measures of complexity. Time and space complexity on a Turing machine. Decidability and complexity.
- **Time complexity.** Time complexity classes. Polynomial time problems and algorithms. P and NP.
- **Non-determinism.** Non-deterministic machines. The class NP redefined. Non-deterministic algorithms for reachability and satisfiability.
- **NP-completeness.** Reductions and completeness. NP-completeness of satisfiability.
- **More NP-complete problems.** Graph-theoretic problems. Hamiltonian cycle and clique.
- **More NP-complete problems.** Sets, numbers and scheduling. Matching, set covering and bin packing.
- **coNP.** Validity of boolean formulae and its completeness. NP∩coNP. Primality and factorisation.
- **Cryptographic complexity.** One-way functions. The class UP.
- **Space complexity.** Deterministic and non-deterministic space complexity classes. The reachability method. Savitch's theorem.
- **Hierarchy.** The time and space hierarchy theorems and complete problems.
- **Descriptive complexity.** Logics capturing complexity classes. Fagin's theorem.

Objectives

At the end of the course students should

- be able to analyse practical problems and classify them according to their complexity;
- be familiar with the phenomenon of NP-completeness, and be able to identify problems that are NP-complete;
- be aware of a variety of complexity classes and their interrelationships;
- understand the role of complexity analysis in cryptography.

Recommended reading

* Papadimitriou, Ch.H. (1994). *Computational complexity*. Addison-Wesley. Goldreich, O. (2008). *Computational complexity: a conceptual perspective*. Cambridge University Press. Sipser, M. (1997). *Introduction to the theory of computation*. PWS.

Concepts in Programming Languages

Lecturer: Dr M.P. Fiore

No. of lectures: 8

Prerequisite courses: None.

Aims

The general aim of this course is to provide an overview of the basic concepts that appear in modern programming languages, the principles that underlie the design of programming languages, and their interaction.

Lectures

- **Introduction, motivation, and overview.** What is a programming language? Application domains in language design. Program execution models. Theoretical foundations. Language standardization. History.
- **The first procedural language: FORTRAN (1954–58).** Execution model. Data types. Control structures. Storage. Subroutines and functions. Parameter passing.
- **The first declarative language: LISP (1958–62).** Expressions, statements, and declarations. S-expressions and lists. Recursion. Static and dynamic scope. Abstract machine. Garbage collection. Programs as data. Parameter passing. Strict and lazy evaluation.
- **Block-structured procedural languages: Algol (1958–68) and Pascal (1970).** Block structure. Parameters and parameter passing. Stack and heap storage. Data types. Arrays and pointers.

- **Object-oriented languages — Concepts and origins: Simula (1964–67) and Smalltalk (1971–80).** Dynamic lookup. Abstraction. Subtyping. Inheritance. Object models.
- **Types.** Types in programming languages. Type systems. Type safety. Type checking and type inference. Polymorphism. Overloading. Type equivalence.
- **Data abstraction and modularity: SML Modules (1984–97).** Information hiding. Modularity. Signatures, structures, and functors. Sharing.
- **The state of the art: Scala (2004–06).** Procedural and declarative aspects. Blocks and functions. Classes and objects. Generic types and methods. Variance annotations. Mixin-class composition.

Objectives

At the end of the course students should

- be familiar with several language paradigms and how they relate to different application domains;
- understand the design space of programming languages, including concepts and constructs from past languages as well as those that may be used in the future;
- develop a critical understanding of the programming languages that we use by being able to identify and compare the same concept as it appears in different languages.

Recommended reading

Books:

* Mitchell, J.C. (2003). *Concepts in programming languages*. Cambridge University Press.
* Scott, M.L. (2009). *Programming language pragmatics*. Morgan Kaufmann. Odersky, M. (2008). *Scala by example*. Programming Methods Laboratory, EPFL.
Pratt, T.W. & Zelkowitz, M.V. (2001). *Programming languages: design and implementation*. Prentice Hall.

Papers:

Kay, A.C. (1993). The early history of Smalltalk. *ACM SIGPLAN Notices*, Vol. 28, No. 3.
Kernighan, B. (1981). Why Pascal is not my favorite programming language. AT&T Bell Laboratories. *Computing Science Technical Report* No. 100.
Koenig, A. (1994). An anecdote about ML type inference. *USENIX Symposium on Very High Level Languages*.
Landin, P.J. (1966). The next 700 programming languages. *Communications of the ACM*, Vol. 9, Issue 3.
Odersky, M. *et al.* (2006). An overview of the Scala programming language. *Technical Report LAMP-REPORT-2006-001*, Second Edition.
McCarthy, J. (1960). Recursive functions of symbolic expressions and their computation by machine. *Communications of the ACM*, 3(4):184–195.

Stroustrup, B. (1991). What is Object-Oriented Programming? (1991 revised version). *Proceedings 1st European Software Festival*.

Databases

Lecturer: Dr T.G. Griffin

No. of lectures: 12

Aims

The overall aim of the course is to cover the fundamentals of database management systems (DBMSs), paying particular attention to relational database systems. The course covers modelling techniques, transferring designs to actual database implementations, SQL, models of query languages, transactions as well as more recent developments, including data warehouses and On-line Analytical Processing (OLAP), and use of XML as a data exchange language. The lectures will make use of the open source DBMS, MySQL.

Lectures

- **Introduction.** What is a database system? Database systems are more than just a collection of data. Three level architecture. OnLine Transaction Processing (OLTP) *versus* OnLine Analytic Processing (OLAP).
- **Entity-Relationship (E/R) modelling.** A bit of set theory. Entities have attributes. Relations have *arity*. Database design and data modelling.
- **The relational data model.** Relations are sets of records. Representing entities and relationships as relations. Queries as derived relations. Relations are the basis of SQL.
- **Relational algebra.** Relational algebra as an abstract query language. Core operations – selection, projection, product, renaming, and joins.
- **Relational calculus.** Relational calculus as an abstract query language that uses notation from set theory. Equivalence with relational algebra.
- **SQL and integrity constraints.** An overview of the core of SQL. SQL has constructs taken from both the relational algebra and the relational calculus. Integrity constraints as special queries.
- **Schema refinement I.** The evils of redundancy. The benefits of redundancy. Functional dependencies as a formal means of investigating redundancy. Relational decomposition. Armstrong's axioms.
- **Schema refinement II.** Schema normalisation. Lossless-join decomposition. Dependency preservation. Boyce–Codd normal form. Third normal form.
- **Further relational algebra, SQL.** SQL is really based on multi-sets (bags). Extending the relational algebra to bags. NULL values as an SQL design error?

- **Transaction management overview.** ACID properties – Atomicity, Consistency, Isolation, and Durability. Serialisability in the database context.
- **On-line Analytical Processing (OLAP).** When to forget about data normalisation. Beware of buzz-words and the Data Warehouse Death March. More on OLTP *versus* OLAP. What is a *data cube*? Data modelling for data warehouses: *star schema*.
- **XML as a data exchange format.** What is XML? XML can be used to share data between proprietary relational databases. XML-based databases?

Objectives

At the end of the course students should

- be able to design entity-relationship diagrams to represent simple database application scenarios;
- know how to convert entity-relationship diagrams to relational database schemas in the standard Normal Forms;
- be able to program simple database applications in SQL;
- understand the basic theory of the relational model and both its strengths and weaknesses;
- be familiar with various recent trends in the database area.

Recommended reading

* Date, C.J. (2004). *An introduction to database systems*. Addison-Wesley (8th ed.).
Elmasri, R. & Navathe, S.B. (2000). *Fundamentals of database systems*. Addison-Wesley (3rd ed.).
Silberschatz, A., Korth, H.F. & Sudarshan, S. (2002). *Database system concepts*. McGraw-Hill (4th ed.).
Ullman, J. & Widom, J. (1997). *A first course in database systems*. Prentice Hall.
Miszczyk, J. and others (1998). *Mastering data warehousing functions*. (IBM Redbook DB2/400) Chapters 1 & 2 only.
`http://www.redbooks.ibm.com/abstracts/sg245184.html`
Garcia-Molina, H. *Data warehousing and OLAP*. Stanford University.
`http://www.cs.uh.edu/ ceick/6340/dw-olap.ppt`
London Metropolitan University, Department of Computing. *Data warehousing and OLAP technology for data mining*. `http://learning.unl.ac.uk/csp002n/CSP002N_wk2.ppt`

Economics and Law

Lecturer: M J.A. Lang and Mr N.D.F. Bohm

No. of lectures: 8

Professional Practice and Ethics (Part IA) provides a useful foundation for this course.

This course is a prerequisite for the Part II courses Security II, Business Studies and E-Commerce.

Aims

This course aims to give students an introduction to some basic concepts in economics and law.

Lectures

- **Game theory.** The choice between cooperation and conflict. Prisoners' Dilemma; Nash equilibrium; hawk–dove; iterated games; evolution of strategies; application to biology and computer science.
- **Classical economics.** Brief history of economics. Definitions: preference, utility, choice and budget. Pareto efficiency; the discriminating monopolist. Welfare and the Arrow theorem.
- **Classical economics continued.** Supply and demand; elasticity; utility; the marginalist revolution; competitive equilibrium. Trade; monopoly rents; public goods; oligopoly. The business cycle.
- **Market failure.** Asymmetric information: the market for lemons; adverse selection; moral hazard; signalling; and brands. Transaction costs and the theory of the firm. Behavioural economics: bounded rationality, heuristics and biases.
- **Auctions.** English auctions; Dutch auctions; all-pay auctions; Vickrey auctions. The winner's curse. The revenue equivalence theorem. Mechanism design and the combinatorial auction. Problems with real auctions. Applicability of auction mechanisms in computer science.
- **Principles of law.** Contract and tort; copyright and patent; binding actions; liabilities and remedies; competition law; choice of law and jurisdiction.
- **Law and the Internet.** EU directives including distance selling, electronic commerce, data protection, electronic signatures and copyright; their UK implementation. UK laws that specifically affect the Internet, including RIP.
- **Network economics.** Real and virtual networks, supply-side *versus* demand-side scale economies, Metcalfe's law, the dominant firm model, price discrimination. Regulatory and other public policy issues of information goods and services markets.

Objectives

At the end of the course students should have a basic appreciation of economic and legal terminology and arguments. They should understand some of the applications of economic models to systems engineering and their interest to theoretical computer science. They should also understand the main constraints that markets and legislation place on firms dealing in information goods and services.

Recommended reading

* Shapiro, C. & Varian, H. (1998). *Information rules*. Harvard Business School Press.
Varian, H. (1999). *Intermediate microeconomics – a modern approach*. Norton.

Further reading:

Smith, A. (1776). *An inquiry into the nature and causes of the wealth of nations*, available at `http://www.econlib.org/LIBRARY/Smith/smWN.html`
Poundstone, W. (1992). *Prisoner's dilemma*. Anchor Books.
Levitt, S.D. & Dubner, S.J. (2005). *Freakonomics*. Morrow.
Seabright, P. (2005). *The company of strangers*. Princeton.
Anderson, R. (2008). *Security engineering* (Chapter 7). Wiley.
Galbraith, J.K. (1991). *A history of economics*. Penguin.
Lessig L. (2005). *Code and other laws of cyberspace v2*, available at `http://www.lessig.org/`

Introduction to Part II

This document lists the courses offered by the Computer Laboratory for Part II of the Computer Science Tripos. Separate booklets give details of the syllabus for the other Parts of the Computer Science Tripos.

The syllabus information given here is for guidance only and should not be considered definitive. Current timetables can be found at
`http://www.cl.cam.ac.uk/teaching/lectlist/`

For most of the courses listed below, a list of recommended books is given. These are roughly in order of usefulness, and lecturers have indicated by means of an asterisk those books which are most recommended for purchase by College libraries.

The Computer Laboratory Library aims to keep at least one copy of each of the course texts in "The Booklocker" (see `http://www.cl.cam.ac.uk/library/`).

For copies of the other syllabus booklets and for answers to general enquiries about Computer Science courses, please get in touch with:

Student Administrator
University of Cambridge
Computer Laboratory
William Gates Building
J J Thomson Avenue
Cambridge
CB3 0FD

telephone: 01223 334656
fax: 01223 334678
e-mail: `undergraduate.admissions@cl.cam.ac.uk`

Michaelmas Term 2010: Part II lectures

Denotational Semantics

Lecturer: Dr M.P. Fiore

No. of lectures + exercise classes: 10 + 2

Aims

The aims of this course are to introduce domain theory and denotational semantics, and to show how they provide a mathematical basis for reasoning about the behaviour of programming languages.

Lectures

- **Introduction.** The denotational approach to the semantics of programming languages. Recursively defined objects as limits of successive approximations.
- **Least fixed points.** Complete partial orders (cpos) and least elements. Continuous functions and least fixed points.
- **Constructions on domains.** Flat domains. Product domains. Function domains.
- **Scott induction.** Chain-closed and admissible subsets of cpos and domains. Scott's fixed-point induction principle.
- **PCF.** The Scott-Plotkin language PCF. Evaluation. Contextual equivalence.
- **Denotational semantics of PCF.** Denotation of types and terms. Compositionality. Soundness with respect to evaluation. [2 lectures].
- **Relating denotational and operational semantics.** Formal approximation relation and its fundamental property. Computational adequacy of the PCF denotational semantics with respect to evaluation. Extensionality properties of contextual equivalence. [2 lectures].
- **Full abstraction.** Failure of full abstraction for the domain model. PCF with parallel or.

Objectives

At the end of the course students should

- be familiar with basic domain theory: cpos, continuous functions, admissible subsets, least fixed points, basic constructions on domains;
- be able to give denotational semantics to simple programming languages with simple types;

- be able to apply denotational semantics; in particular, to understand the use of least fixed points to model recursive programs and be able to reason about least fixed points and simple recursive programs using fixed point induction;
- understand the issues concerning the relation between denotational and operational semantics, adequacy and full abstraction, especially with respect to the language PCF.

Recommended reading

Winskel, G. (1993). *The formal semantics of programming languages: an introduction*. MIT Press.
Gunther, C. (1992). *Semantics of programming languages: structures and techniques*. MIT Press.
Tennent, R. (1991). *Semantics of programming languages*. Prentice Hall.

Human-Computer Interaction

Lecturer: Dr A.F. Blackwell

No. of lectures: 8

Aims

This course will introduce systematic approaches to the design and analysis of user interfaces.

Lectures

- **The scope and challenges of HCI and Interaction Design.**
- **Visual representation.** Segmentation and variables of the display plane. Modes of correspondence.
- **Text and gesture interaction.** Evolution of interaction hardware. Measurement and assessment of novel methods.
- **Inference-based approaches.** Bayesian strategies for data entry, and programming by example.
- **Augmented reality and tangible user interfaces.** Machine vision, fiducial markers, paper interfaces, mixed reality.
- **Usability of programming languages.** End-user programming, programming for children, cognitive dimensions of notations.
- **User-centred design research.** Contextual observation, prototyping, think-aloud protocols, qualitative data in the design cycle.

- **Usability evaluation methods.** Formative and summative methods. Empirical measures. Evaluation of Part II projects.

Objectives

On completing the course, students should be able to

- propose design approaches that are suitable to different classes of user and application;
- identify appropriate techniques for analysis and critique of user interfaces;
- be able to design and undertake quantitative and qualitative studies in order to improve the design of interactive systems;
- understand the history and purpose of the features of contemporary user interfaces.

Recommended reading

* Sharp, H., Rogers, Y. & Preece, J. (2007). *Interaction design: beyond human–computer interaction*. Wiley (2nd ed.).

Further reading:

Carroll, J.M. (ed.) (2003). *HCI models, theories and frameworks: toward a multi-disciplinary science*. Morgan Kaufmann.
Cairns, P. & Cox, A. (eds.) (2008). *Research methods for human-computer interaction*. Cambridge University Press.

Information Retrieval

Lecturer: Dr S.H. Teufel

No. of lectures: 8

Prerequisite courses: a basic encounter with Probability is assumed

Aims

The course is aimed to characterise information retrieval in terms of the data, problems and concepts involved. The main formal retrieval model and the main evaluation methods are described. The course then covers problems and standard solutions in information extraction, and in question answering.

Lectures

- **Information retrieval introduction.** Key problems and concepts. Information need. Indexing model. Examples.
- **Retrieval models.** Boolean model. Vector Space model. Stemming.

- **Evaluation methodology.** TREC. User experiments. Evaluation metrics.
- **Search engines and linkage algorithms.** PageRank and Kleinberg's Hubs and Authorities.
- **Information extraction.** Task and evaluation. Lexico-semantic patterns.
- **Advanced information extraction methods.** Bootstrapping. Learning.
- **Question answering.** Performance criteria and effectiveness measures, test methodology, established results.
- **Overview of summarisation technology.** Extractive *versus* abstractive summarisation. Evaluation.

Objectives

At the end of this course, students should be able to

- define the tasks of information retrieval, question answering and information extraction and differences between them;
- understand the main concepts and strategies used in IR, QA, and IE;
- appreciate the challenges in these three areas;
- develop strategies suited for specific retrieval, extraction or question situations, and recognise the limits of these strategies;
- understand (the reasons for) the evaluation strategies developed for these three areas.

Recommended reading

* Manning, C.D., Raghavan, P. & Schütze, H. (2008). *Introduction to information retrieval*. Cambridge University Press. Available at `http://www-csli.stanford.edu/~hinrich/information-retrieval-book.html`.

Natural Language Processing

Lecturer: Dr A.A. Copestake

No. of lectures: 8

Prerequisite courses: Regular Languages and Finite Automata, Probability, Logic and Proof, and Artificial Intelligence

Aims

This course aims to introduce the fundamental techniques of natural language processing and to develop an understanding of the limits of those techniques. It aims to introduce some current research issues, and to evaluate some current and potential applications.

Lectures

- **Introduction.** Brief history of NLP research, current applications, generic NLP system architecture.
- **Finite-state techniques.** Inflectional and derivational morphology, finite-state automata in NLP, finite-state transducers.
- **Prediction and part-of-speech tagging.** Corpora, simple N-grams, word prediction, stochastic tagging, evaluating system performance.
- **Parsing and generation.** Generative grammar, context-free grammars, parsing and generation with context-free grammars, weights and probabilities.
- **Parsing with constraint-based grammars.** Constraint-based grammar, unification.
- **Compositional and lexical semantics.** Simple compositional semantics in constraint-based grammar. Semantic relations, WordNet, word senses, word sense disambiguation.
- **Discourse and dialogue.** Anaphora resolution, discourse relations.
- **Applications.** Combination of components into applications.

Objectives

At the end of the course students should

- be able to discuss the current and likely future performance of several NLP applications;
- be able to describe briefly a fundamental technique for processing language for several subtasks, such as morphological processing, parsing, word sense disambiguation etc.;
- understand how these techniques draw on and relate to other areas of computer science.

Recommended reading

* Jurafsky, D. & Martin, J. (2008). *Speech and language processing*. Prentice Hall.

For background reading, one of:
Pinker, S. (1994). *The language instinct*. Penguin.
Matthews, P. (2003). *Linguistics: a very short introduction*. OUP.

Although the NLP lectures don't assume any exposure to linguistics, the course will be easier to follow if students have some understanding of basic linguistic concepts.

For reference purposes:
The Internet Grammar of English,
`http://www.ucl.ac.uk/internet-grammar/home.htm`

Optimising Compilers

Lecturer: Professor A. Mycroft

No. of lectures: 16

Prerequisite course: Compiler Construction

Aims

The aims of this course are to introduce the principles of program optimisation and related issues in decompilation. The course will cover optimisations of programs at the abstract syntax, flowgraph and target-code level. It will also examine how related techniques can be used in the process of decompilation.

Lectures

- **Introduction and motivation.** Outline of an optimising compiler. Optimisation partitioned: *analysis* shows a property holds which enables a *transformation*. The flow graph; representation of programming concepts including argument and result passing. The phase-order problem.
- **Kinds of optimisation.** Local optimisation: peephole optimisation, instruction scheduling. Global optimisation: common sub-expressions, code motion. Interprocedural optimisation. The call graph.
- **Classical dataflow analysis.** Graph algorithms, *live* and *avail* sets. Register allocation by register colouring. Common sub-expression elimination. Spilling to memory; treatment of CSE-introduced temporaries. Data flow anomalies. Static Single Assignment (SSA) form.
- **Higher-level optimisations.** Abstract interpretation, Strictness analysis. Constraint-based analysis, Control flow analysis for lambda-calculus. Rule-based inference of program properties, Types and effect systems. Points-to and alias analysis.
- **Target-dependent optimisations.** Instruction selection. Instruction scheduling and its phase-order problem.
- **De-compilation.** Legal/ethical issues. Some basic ideas, control flow and type reconstruction.

Objectives

At the end of the course students should

- be able to explain program analyses as dataflow equations on a flowgraph;
- know various techniques for high-level optimisation of programs at the abstract syntax level;
- understand how code may be re-scheduled to improve execution speed;

- know the basic ideas of decompilation.

Recommended reading

* Nielson, F., Nielson, H.R. & Hankin, C.L. (1999). *Principles of program analysis*. Springer. Good on part A and part B.
Appel, A. (1997). *Modern compiler implementation in Java/C/ML* (3 editions).
Muchnick, S. (1997). *Advanced compiler design and implementation*. Morgan Kaufmann.
Wilhelm, R. (1995). *Compiler design*. Addison-Wesley.
Aho, A.V., Sethi, R. & Ullman, J.D. (2007). *Compilers: principles, techniques and tools*. Addison-Wesley (2nd ed.).

Principles of Communication

Lecturer: Professor J.A. Crowcroft

No. of lectures and examples classes: 20 + 4

Prerequisite course: Digital Communication I

This course is a prerequisite for Security II and Mobile & Sensor Systems.

Aims

This course aims to provide a detailed understanding of the underlying principles for how communications systems operate. Practical examples (from wired and wireless communications, the Internet, Telephony and Broadcast Networks) are used to illustrate the principles

Lectures

- **Introduction.** Course overview. Abstraction, layering. The structure of real networks, Telephony, Internet, ATM.
- **Modular functionality for communications.** Some systems design paradigms, often orthogonal to layers.
- **Information, Noise, Interference, Capacity** We review relevant information theory and how the limit for the capacity of a channel can be calculated.
- **Topology and Routing.** How many ways can we work out how to get from A to B? We review relevant graph theory, including recent advances in understanding the topology of the Internet and similar networks. [2 lectures]
- **Error control.** What do we do when things go wrong? Information can be coded and transmitted in a number of ways to survive interference. Retransmit, or pre-transmit?
- **Flow control.** Control theory is a branch of engineering familiar to people building dynamic machines. It can be applied to network traffic. Stemming the flood, at source, sink, or in between?

- **Shared media networks.** Ethernet and Radio networks: some special problems for media access and so forth. We revisit the problem of capacity of a channel in the context of a radio network. [2 lectures]
- **Switched networks.** What does a switch do and how? [2 lectures]
- **Integrated Service Packet Networks for IP.** Traffic may be adaptive to feedback control, or it may be a given. Characteristics may be quite complex in terms of time series. This has an impact on the design choices for scheduling and queue management algorithms for packet forwarding, including APIs to Quality of Service and routing with QoS. We recapitulate some basic queuing theory. [2 lectures]
- **The big picture for managing traffic.** Economics and policy are relevant to networks in many ways. [2 lectures]
- **Naming and addressing.** Reviewing Who is where?

Objectives

At the end of the course students should be able to explain the concepts such as addressing, buffer management, congestion control, differential services, estimation, feedback, gateways, hierarchy, IP, jitter, k-ary resilience, layering, multiplexing, networking, OSI, priority, queueing, routing, switching, transmission control, user plane, virtualisation, wireless, yield management, and capacity *versus* bandwidth.

Recommended reading

* Keshav, S. (1997). *An engineering approach to computer networking*. Addison-Wesley (1st ed.). ISBN 0201634422
Alternatives to Keshav:
Peterson, L.L. & Davie, B.S. (2007). *Computer networks: a systems approach*. Morgan Kaufmann (4th ed.).
Stevens, W.R. (1994). *TCP/IP illustrated, vol. 1: the protocols*. Addison-Wesley (1st ed.). ISBN 0201633469

Security II

Lecturer: Professor R.J. Anderson

No. of lectures: 16

Prerequisite courses: Introduction to Security, Discrete Mathematics, Economics and Law, Operating Systems, Digital Communication I, Principles of Communication

This course is a prerequisite for E-Commerce.

Aims

This course aims to give students a thorough understanding of computer security technology. This includes high-level issues such as security policy (modelling what ought to be protected) and engineering (how we can obtain assurance that the protection

provided is adequate). It also involves the protection mechanisms supported by modern processors and operating systems; cryptography and its underlying mathematics; electrical engineering issues such as emission security and tamper resistance; and a wide variety of attacks ranging from network exploits through malicious code to protocol failure.

Lectures

- **What is security?** Introduction and definitions: different meanings of principal, system, policy, trust. Diversity of applications. Relationship with distributed system issues such as fault-tolerance and naming.
- **Multilevel security.** The Bell–LaPadula policy model; similar formulations such as the lattice model, non-interference and non-deducibility. Composability. Real MLS systems and their problems: covert channels, the cascade problem, polyinstantiation, dynamic and non-monotonic labelling. Flexibility, usability and compatibility.
- **Multilateral security policy models.** Compartmented systems, Chinese Wall, the BMA policy. Inference security: query controls, trackers, cell suppression, randomization, stateful controls, and active attacks.
- **Banking and bookkeeping systems.** Double-entry bookkeeping, the Clark-Wilson policy model. Separation of duties, and its implementation problems. Payment systems and how they fail: SWIFT, ATMs.
- **Monitoring systems.** Alarms. Sensor defeats; feature interactions; attacks on communications; attacks on trust. Examples: antivirus software, tachographs, prepayment electricity meters. Seals; electronic postal indicia.
- **Telecommunications security.** Attacks on metering, signalling, switching and configuration. Attacks on end systems. Feature interactions. Mobile phone issues: protection issues in GSM, GPRS, 3g. Surveillance technology and practice. Models of attacks on communications systems.
- **Anonymity and peer-to-peer systems.** Dining cryptographers; mix-nets. Models of opponents. Surveillance *versus* service denial. Peer-to-peer systems; resilience and censorship resistance.
- **Hardware engineering issues.** Tamper resistance: smartcards, cryptoprocessors. Mechanical and optical probing, fault induction, power analysis, emission security, timing attacks.
- **Software engineering issues.** Classes of software vulnerabilities: stack overflows, buffer overflows, namespace and protocol issues, concurrency vulnerabilties. History, examples, exploits, and prevention.
- **Stream ciphers.** Historical systems: Caesar, Vigenère, Playfair. Revision of information theory: unicity distance, the one-time-pad, attacks in depth. Shift register based systems: the multiplexer generator, RC4, A5. Attacks on these systems: divide and conquer, fast correlation.

- **Block ciphers.** Design of block ciphers: SP-networks and Feistel ciphers. Differential and linear cryptanalysis. AES; Serpent; DES. Revision of the random oracle model: modes of operation. Splicing and collision attacks. Message authentication codes and hash functions.

- **Symmetric cryptographic protocols.** Needham–Schroder, Otway–Rees, Kerberos, the wide-mouthed frog. The BAN logic. Applying BAN to verify a payment protocol. API security.

- **Asymmetric cryptosystems.** Revision of public-key mathematics: RSA, ElGamal, Diffie–Hellman. Elliptic curve systems, factoring algorithms. Advanced primitives: identity-based schemes; threshold schemes; zero knowledge; blind signatures.

- **Asymmetric cryptographic protocols.** Needham–Schroder, Denning–Sacco, TMN. Applications including SSL/TLS, SSH and PGP. The BAN logic applied to public key systems.

- **Rights management and competition.** Copyright management systems; accessory control systems; the Trusted Computing architecture. Tensions between security and competition.

- **Security engineering.** Why is security management hard? Security economics: the effects of market races, externalities, coordination problems, correlated risks, the patching cycle, and supply chain effects. Problems with certification including the Common Criteria. Behavioural and organisational effects. Interaction with the regulatory environment.

Objectives

At the end of the course students should be able to tackle an information protection problem by drawing up a threat model, formulating a security policy, and designing specific protection mechanisms to implement the policy.

Recommended reading

* Anderson, R. (2008). *Security engineering*. Wiley (2nd ed.). First edition (2001) available at `http://www.cl.cam.ac.uk/users/rja14/book.html`
Stinson, D.R. (2002). *Cryptography: theory and practice*. Chapman & Hall (2nd ed.).
Schneier, B. (1995). *Applied cryptography: protocols, algorithms, and source code in C*. Wiley (2nd ed.).

Further reading:

Kahn, D. (1966). *The codebreakers: the story of secret writing*. Weidenfeld and Nicolson.
Cheswick, W.R., Bellovin, S.M. & Rubin, A.D. (2003). *Firewalls and Internet security: repelling the wily hacker*. Addison-Wesley (2nd ed.)
Howard, M. & leBlanc, D. (2003). *Writing secure code*. Microsoft Press (2nd ed.)
Gollmann, D. (2006). *Computer security*. Wiley (2nd ed.).
Koblitz, N. (1994). *A course in number theory and cryptography*. Springer-Verlag (2nd ed.).
Neumann, P. (1994). *Computer related risks*. Addison-Wesley.

Biham, E. & Shamir, A. (1993). *Differential cryptanalysis of the data encryption standard*. Springer-Verlag.
Leveson, N.G. (1995). *Safeware: system safety and computers*. Addison-Wesley.
Konheim, A.G. (2007). *Computer security and cryptography*. Wiley.
de Leeuw, K. & Bergstra, J. (2007). *The history of information security*. Elsevier.

Types

Lecturer: Dr C. Urban

No. of lectures: 8

Prerequisite course: Semantics of Programming Languages

Aims

The aim of this course is to show by example how type systems for programming languages can be defined and their properties developed, using techniques that were introduced in the Part IB course on *Semantics of Programming Languages*.

Lectures

- **Introduction.** The role of type systems in programming languages. Formalizing type systems. [1 lecture]
- **ML polymorphism.** ML-style polymorphism. Principal type schemes and type inference. [2 lectures]
- **Polymorphic reference types.** The pitfalls of combining ML polymorphism with reference types. [1 lecture]
- **Polymorphic lambda calculus.** Syntax and reduction semantics. Examples of datatypes definable in the polymorphic lambda calculus. Applications. [2 lectures]
- **Further topics.** The Curry–Howard correspondence as a source of type systems. Dependent types. [2 lectures]

Objectives

At the end of the course students should

- appreciate how type systems can be used to constrain or describe the dynamic behaviour of programs;
- be able to use a rule-based specification of a type system to infer typings and to establish type soundness results;
- appreciate the expressive power of the polymorphic lambda calculus.

Recommended reading

* Pierce, B.C. (2002). *Types and programming languages*. MIT Press.
Cardelli, L. (1997). Type systems. In *CRC handbook of computer science and engineering*. CRC Press.
Cardelli, L. (1987). Basic polymorphic typechecking. *Science of computer programming*, vol. 8, pp. 147–172.
Girard, J-Y. (tr. Taylor, P. & Lafont, Y.) (1989). *Proofs and types*. Cambridge University Press.

Lent Term 2011: Part II lectures

Advanced Graphics

Lecturers: Professor N.A. Dodgson and Dr P.A. Benton

No. of lectures: 12

Prerequisite course: Computer Graphics and Image Processing

Aims

This course provides students with a solid grounding in a variety of three-dimensional modelling mechanisms. It also provides an introduction to radiosity, animation, hardware technologies, and current commercial uses of computer graphics.

Lectures

- **Revision, polygons, hardware.** Revision of the ray tracing and polygon scan conversion methods of making images from 3D models; the pros and cons of each approach. Current uses of computer graphics in animation, special effects, Computer-Aided Design and marketing. Drawing polygons. Graphics cards. [NAD, 1 lecture]
- **Splines for modelling arbitrary 3D geometry.** (splines are the standard 3D modelling mechanism for Computer-Aided Design). Features required of surface models in a Computer-Aided Design package. Bezier curves and surfaces. B-splines, from uniform, non-rational B-splines through to non-uniform, rational B-splines (NURBS). [NAD, 3 lectures]
- **Subdivision surfaces.** (an alternative mechanism for representing arbitrary 3D geometry, now widely used in the animation industry). Introduction to subdivision. Pros and cons when compared to NURBS. [NAD, 2 lectures]
- **Geometric methods for ray tracing.** The fundamentals of raycasting and constructive solid geometry (CSG). [PAB, 1 lecture]
- **Illumination: Ray tracing effects and global lighting.** Visual effects, radiosity and photon mapping. [PAB, 1 lecture]
- **Computational geometry.** The mathematics of discrete geometry: what can you know, and how well can you know it? [PAB, 1 lecture]
- **Implicit surfaces, voxels and particle systems.** A sampler of special effects techniques. [PAB, 1 lecture]
- **OpenGL and shaders.** Tools and technologies available today; previews of what's coming tomorrow. [PAB, 2 lectures]

Objectives

On completing the course, students should be able to

- compare and contrast ray tracing with polygon scan conversion;
- define NURBS basis functions, and explain how NURBS curves and surfaces are used in 2D and 3D modelling;
- describe the underlying theory of subdivision and define the Catmull-Clark and Doo-Sabin subdivision methods;
- understand the core technologies of ray tracing, constructive solid geometry, computational geometry, implicit surfaces, voxel rendering and particle systems;
- understand several global illumination technologies such as radiosity and photon mapping, and be able to discuss each in detail;
- be able to describe current graphics technology and discuss future possibilities.

Recommended reading

Students should expect to refer to one or more of these books, but should not find it necessary to purchase any of them.

* Slater, M., Steed, A. & Chrysanthou, Y. (2002). *Computer graphics and virtual environments: from realism to real-time*. Addison-Wesley.
Watt, A. (1999). *3D Computer graphics*. Addison-Wesley (3rd ed).
de Berg, M., Cheong, O., van Kreveld, M. & Overmars, M. (2008). *Computational geometry: algorithms and applications*. Springer (3rd ed.).
Rogers, D.F. & Adams, J.A. (1990). *Mathematical elements for computer graphics*. McGraw-Hill (2nd ed.).
Warren, J. & Weimer, H. (2002). *Subdivision methods for geometric design*. Morgan Kaufmann.

Artificial Intelligence II

Lecturer: Dr S.B. Holden

No. of lectures: 16

Prerequisite courses: Artificial Intelligence I, Logic and Proof, Algorithms I + II, Mathematical Methods for Computer Science, Discrete Mathematics I + II, Probability/Probability from the NST Mathematics course.

Aims

The aim of this course is to build on Artificial Intelligence I, first by introducing more elaborate methods for planning within the symbolic tradition, but then by moving beyond the purely symbolic view of AI and presenting methods developed for dealing with the critical concept of uncertainty. The central tool used to achieve the latter is probability theory. The course continues to exploit the primarily algorithmic and computer science-centric perspective that informed Artificial Intelligence I.

The course aims to provide further tools and algorithms required to produce AI systems able to exhibit limited human-like abilities, with an emphasis on the need to obtain better planning algorithms, and systems able to deal with the uncertainty inherent in the environments that most real agents might be expected to perform within.

Lectures

- **Further planning.** Incorporating heuristics into partial-order planning. Planning graphs. The GRAPHPLAN algorithm. Planning using propositional logic. Planning as a constraint satisfaction problem. [3 lectures]
- **Uncertainty and Bayesian networks.** Review of probability as applied to AI. Representing uncertain knowledge using Bayesian networks. Inference in Bayesian networks using both exact and approximate techniques. Other ways of dealing with uncertainty. [3 lectures]
- **Utility and decision-making.** The concept of *utility*. Utility and preferences. Deciding how to act by maximizing expected utility. Decision networks. The value of information, and reasoning about when to gather more. [2 lectures]
- **Uncertain reasoning over time.** Markov processes, transition and sensor models. Inference in temporal models: filtering, prediction, smoothing and finding the most likely explanation. The Viterbi algorithm. Hidden Markov models. [2 lectures]
- **Further supervised learning I.** Bayes theorem as applied to supervised learning. The maximum likelihood and maximum *a posteriori* hypotheses. What does this teach us about the backpropagation algorithm? [1 lecture]
- **. How to classify optimally.** Bayesian decision theory and Bayes optimal classification. What does this tell us about how best to do supervised machine learning? [1 lecture]
- **Further supervised learning II.** Applying the Bayes optimal classification approach to neural networks. [3 lectures]
- **Reinforcement learning.** Learning from rewards and punishments. Markov decision processes. The problems of temporal credit assignment and exploration versus exploitation. Q-learning and its convergence. How to choose actions. [1 lecture]

Objectives

At the end of this course students should:

- Have gained a deeper appreciation of the way in which computer science has been applied to the problem of AI, and in particular for more recent techniques concerning knowledge representation, inference, planning and uncertainty.
- Know how to model situations using a variety of knowledge representation techniques.

- Be able to design problem solving methods based on knowledge representation, inference, planning, and learning techniques.
- Know how probability theory can be applied in practice as a means of handling uncertainty in AI systems.

Recommended reading

* Russell, S. & Norvig, P. (2003). *Artificial intelligence: a modern approach*. Prentice Hall (3rd ed.).
Bishop, C.M. (2006). *Pattern recognition and machine learning*. Springer.
Ghallab, M., Nau, D. & Traverso, P. (2004). *Automated planning: theory and practice*. Morgan Kaufmann.

Business Studies

Lecturer: Mr J.A. Lang

No. of lectures: 8

Or “How to Start and Run a Computer Company”

Prerequisite course: Economics and Law

This course is a prerequisite for E-Commerce.

Aims

The aims of this course are to introduce students to all the things that go to making a successful project or product other than just the programming. The course will survey some of the issues that students are likely to encounter in the world of commerce and that need to be considered when setting up a new computer company.

See also Business Seminars in the Easter Term.

Lectures

- **So you’ve got an idea?** Introduction. Why are you doing it and what is it? Types of company. Market analysis. The business plan.
- **Money and tools for its management.** Introduction to accounting: profit and loss, cash flow, balance sheet, budgets. Sources of finance. Stocks and shares. Options and futures.
- **Setting up: legal aspects.** Company formation. Brief introduction to business law; duties of directors. Shares, stock options, profit share schemes and the like. Intellectual Property Rights, patents, trademarks and copyright. Company culture and management theory.
- **People.** Motivating factors. Groups and teams. Ego. Hiring and firing: employment law. Interviews. Meeting techniques.

- **Project planning and management.** Role of a manager. PERT and GANTT charts, and critical path analysis. Estimation techniques. Monitoring.

- **Quality, maintenance and documentation.** Development cycle. Productization. Plan for quality. Plan for maintenance. Plan for documentation.

- **Marketing and selling.** Sales and marketing are different. Marketing; channels; marketing communications. Stages in selling. Control and commissions.

- **Growth and exit routes.** New markets: horizontal and vertical expansion. Problems of growth; second system effects. Management structures. Communication. Exit routes: acquisition, floatation, MBO or liquidation. Futures: some emerging ideas for new computer businesses. Summary. Conclusion: now you do it!

Objectives

At the end of the course students should

- be able to write and analyse a business plan;

- know how to construct PERT and GANTT diagrams and perform critical path analysis;

- appreciate the differences between profitability and cash flow, and have some notion of budget estimation;

- have an outline view of company formation, share structure, capital raising, growth and exit routes;

- have been introduced to concepts of team formation and management;

- know about quality documentation and productization processes;

- understand the rudiments of marketing and the sales process.

Recommended reading

Lang, J. (2001). *The high-tech entrepreneur's handbook: how to start and run a high-tech company*. FT.COM/Prentice Hall.

Students will be expected to able to use Microsoft Excel and Microsoft Project.

For additional reading on a lecture-by-lecture basis, please see the course website.

Students are strongly recommended to enter the CU Entrepreneurs Business Ideas Competition `http://www.cue.org.uk/`

Comparative Architectures

Lecturer: Dr R.D. Mullins

No. of lectures: 16

Prerequisite course: Computer Design

Aims

This course examines the techniques and underlying principles that are used to design high-performance computers and processors. Particular emphasis is placed on understanding the trade-offs involved when making design decisions at the architectural level. A range of processor architectures are explored and contrasted. In each case we examine their merits and limitations and how ultimately the ability to scale performance is restricted.

Lectures

- **Introduction**. The impact of technology scaling and market trends.
- **Fundamentals of Computer Design**. Amdahl's law, energy/performance trade-offs, ISA design.
- **Advanced pipelining**. Pipeline hazards; exceptions; optimal pipeline depth; branch prediction; the branch target buffer [2 lectures]
- **Superscalar techniques**. Instruction-Level Parallelism (ILP); superscalar processor architecture [2 lectures]
- **Software approaches to exploiting ILP**. VLIW architectures; local and global instruction scheduling techniques; predicated instructions and support for speculative compiler optimisations.
- **Multithreaded processors**. Coarse-grained, fine-grained, simultaneous multithreading
- **The memory hierarchy**. Caches; programming for caches; prefetching [2 lectures]
- **Vector processors**. Vector machines; short vector/SIMD instruction set extensions; stream processing
- **Chip multiprocessors**. The communication model; memory consistency models; false sharing; multiprocessor memory hierarchies; cache coherence protocols; synchronization
- **On-chip interconnection networks**. Bus-based interconnects; on-chip packet switched networks
- **Special-purpose architectures**. Converging approaches to computer design

Objectives

At the end of the course students should

- understand what determines processor design goals;
- appreciate what constrains the design process and how architectural trade-offs are made within these constraints;
- be able to describe the architecture and operation of pipelined and superscalar processors, including techniques such as branch prediction, register renaming and out-of-order execution;
- have an understanding of vector, multithreaded and multi-core processor architectures;
- for the architectures discussed, understand what ultimately limits their performance and application domain.

Recommended reading

* Hennessy, J. & Patterson, D. (2006). *Computer architecture: a quantitative approach*. Elsevier (4th ed.) ISBN 978-0-12-370490-0. (3rd edition is also good)

Computer Systems Modelling

Lecturer: Dr R.J. Gibbens and Dr C-K. Chau

No. of lectures: 12

Prerequisite courses: Probability, Mathematical Methods for Computer Science

Aims

The aims of this course are to introduce the concepts and principles of analytic modelling and simulation, with particular emphasis on understanding the behaviour of computer and communications systems.

Lectures

- **Introduction to modelling.** Overview of analytic techniques and simulation. Little's law.
- **Introduction to discrete event simulation.** Applicability to computer system modelling and other problems. Advantages and limitations of simulation approaches.
- **Random number generation methods and simulation techniques.** Review of statistical distributions. Statistical measures for simulations, confidence intervals and stopping criteria. Variance reduction techniques. [2 lectures]
- **Simple queueing theory.** Stochastic processes: introduction and examples. The Poisson process. Advantages and limitations of analytic approaches.

- **Birth–death processes, flow balance equations.** Birth–death processes and their relation to queueing systems. The M/M/1 queue in detail: existence and when possible solution for equilibrium distribution, mean occupancy and mean residence time.

- **Queue classifications, variants on the M/M/1 queue and applications to queueing networks.** Extensions to variants of the M/M/1 queue. Queueing networks. [2 lectures]

- **The M/G/1 queue and its application.** The Pollaczek–Khintchine formula and related performance measures. [2 lectures]

- **Randomized algorithms I.** Hashing, bloom filters and streaming algorithms.

- **Randomized algorithms II.** Random access protocols, random routing, random load balancing.

Objectives

At the end of the course students should

- be able to build simple Markov models and understand the critical modelling assumptions;

- be able to solve simple birth–death processes;

- understand that in general as the utilization of a system increases towards unity then the response time will tend to increase — often dramatically so;

- understand the tradeoffs between different types of modelling techniques;

- be aware of the issues in building a simulation of a computer system and analysing the results obtained;

- be familiar with different types of randomized algorithm.

Reference books

* Ross, S.M. (2002). *Probability models for computer science*. Academic Press.
Mitzenmacher, M. & Upfal, E. (2005). *Probability and computing: randomized algorithms and probabilistic analysis*. Cambridge University Press.
Jain, A.R. (1991). *The art of computer systems performance analysis*. Wiley.
Kleinrock, L. (1975). *Queueing systems, vol.* 1*. Theory*. Wiley.

Computer Vision

Lecturer: Dr C.P. Town

No. of lectures: 12

Prerequisite courses: Probability, Mathematical Methods for Computer Science. Artificial Intelligence I (recommended)

Aims

The aims of this course are to introduce the principles, models and applications of computer vision. The course will cover: image formation, structure, and coding; edge and feature detection; texture, colour, stereo, and motion; wavelet methods for visual coding and analysis; interpretation of surfaces, solids, and shapes; appearance modelling; pattern recognition and classification; visual inference and learning. Several of these issues will be illustrated using the examples of optical character recognition, image retrieval, and face recognition.

Lectures

- **Goals of computer vision; why they are so difficult.** How images are formed, and the ill-posed problem of making 3D inferences from them about objects and their properties.
- **Image sensing, pixel arrays, cameras.** Elementary operations on image arrays; coding and information measures. Sampling and aliasing.
- **Mathematical operators for extracting image structure.** Finite differences and directional derivatives. Filters; convolution; correlation. Fourier and wavelet transforms.
- **Edge detection operators; the information revealed by edges.** The Laplacian operator and its zero-crossings. Logan's theorem.
- **Multi-scale feature detection and matching.** Gaussian pyramids and SIFT (scale-invariant feature transform). Active contours; energy-minimising snakes. 2D wavelets as visual primitives.
- **Texture, colour, stereo, and motion descriptors.** Disambiguation and the achievement of invariances. Image and motion segmentation.
- **Lambertian and specular surfaces.** Reflectance maps. Image formation geometry. Discounting the illuminant when inferring 3D structure and surface properties.
- **Shape representation.** Inferring 3D shape from shading; surface geometry. Boundary descriptors; active appearance models; codons; superquadrics and the "2.5-Dimensional" sketch.
- **Perceptual psychology and visual cognition.** Vision as model-building and graphics in the brain. Learning to see. Visual illusions, and what they may imply about how vision works.

- **Bayesian inference in vision; knowledge-driven interpretations.** Classifiers and pattern recognition. Probabilistic methods in vision.
- **Applications of machine learning in computer vision.** Appearance and model based representations. Discriminative and generative methods. Optical character recognition. Content based image retrieval.
- **Approaches to face detection, face recognition, and facial interpretation.**

Objectives

At the end of the course students should

- understand visual processing from both “bottom-up” (data oriented) and “top-down” (goals oriented) perspectives;
- be able to decompose visual tasks into sequences of image analysis operations, representations, specific algorithms, and inference principles;
- understand the roles of image transformations and their invariances in pattern recognition and classification;
- be able to describe and contrast techniques for extracting and representing features, edges, shapes, and textures
- be able to analyse the robustness, brittleness, generalizability, and performance of different approaches in computer vision;
- understand some of the major practical application problems, such as face interpretation, character recognition, and image retrieval.

Recommended reading

* Forsyth, D.A. & Ponce, J. (2003). *Computer vision: a modern approach*. Prentice Hall.
Shapiro, L. & Stockman, G. (2001). *Computer vision*. Prentice Hall.

Digital Signal Processing

Lecturer: Dr M.G. Kuhn

No. of lectures: 12

Prerequisite courses: Probability, Mathematical Methods for Computer Science
The last lecture of Unix Tools (MATLAB introduction) is a prerequisite for the practical exercises. Some of the material covered in Floating-Point Computation will also help in this course.

Aims

This course teaches the basic signal-processing principles necessary to understand many modern high-tech systems, with digital-communications examples. Students will gain

practical experience from numerical experiments in MATLAB-based programming assignments.

Lectures

- **Signals and systems.** Discrete sequences and systems, their types and properties. Linear time-invariant systems, convolution.
- **Phasors.** Eigen functions of linear time-invariant systems. Review of complex arithmetic. Some examples from electronics, optics and acoustics.
- **Fourier transform.** Phasors as orthogonal base functions. Forms of the Fourier transform. Convolution theorem, Dirac's delta function, impulse combs in the time and frequency domain.
- **Discrete sequences and spectra.** Periodic sampling of continuous signals, periodic signals, aliasing, sampling and reconstruction of low-pass and band-pass signals, spectral inversion.
- **Discrete Fourier transform.** Continuous *versus* discrete Fourier transform, symmetry, linearity, review of the FFT, real-valued FFT.
- **Spectral estimation.** Leakage and scalloping phenomena, windowing, zero padding.
- **Finite and infinite impulse-response filters.** Properties of filters, implementation forms, window-based FIR design, use of frequency-inversion to obtain high-pass filters, use of modulation to obtain band-pass filters, FFT-based convolution, polynomial representation, *z*-transform, zeros and poles, use of analog IIR design techniques (Butterworth, Chebyshev I/II, elliptic filters).
- **Digital modulation.** IQ representation of band-pass signals, in particular AM, FM, MSK, QAM, and OFDM signals. Clock recovery, symbol detection, matched filter, software-defined radio.
- **Random sequences and noise.** Random variables, stationary processes, autocorrelation, crosscorrelation, deterministic crosscorrelation sequences, filtered random sequences, white noise, exponential averaging.
- **Correlation coding.** Random vectors, dependence *versus* correlation, covariance, decorrelation, matrix diagonalization, eigen decomposition, Karhunen–Loève transform, principal component analysis. Relation to orthogonal transform coding using fixed basis vectors, such as DCT.
- **Lossy versus lossless compression.** What information is discarded by human senses and can be eliminated by encoders? Perceptual scales, masking, spatial resolution, colour coordinates, some demonstration experiments.
- **Quantization, image coding standards.** A/μ-law coding, delta coding, JPEG.

Objectives

By the end of the course students should be able to

- apply basic properties of time-invariant linear systems;
- understand sampling, aliasing, convolution, filtering, the pitfalls of spectral estimation;
- explain the above in time and frequency domain representations;
- use filter-design software;
- visualize and discuss digital filters in the *z*-domain;
- use the FFT for convolution, deconvolution, filtering;
- implement, apply and evaluate simple DSP applications in MATLAB;
- apply transforms that reduce correlation between several signal sources;
- understand the basic principles of several widely-used modulation and image coding techniques.

Recommended reading

* Lyons, R.G. (2004). *Understanding digital signal processing.* Prentice Hall (2nd ed.).
Oppenheim, A.V. & Schafer, R.W. (1999). *Discrete-time digital signal processing.* Prentice Hall (2nd ed.).
Stein, J. (2000). *Digital signal processing – a computer science perspective.* Wiley.
Salomon, D. (2002). *A guide to data compression methods.* Springer.

E-Commerce

Lecturers: Mr J.A. Lang and others

No. of lectures and examples classes: 8 + 1

Prerequisite courses: Business Studies, Security, Economics and Law

Aims

This course aims to give students an outline of the issues involved in setting up an e-commerce site.

Lectures

- **The history of electronic commerce.** Mail order; EDI; web-based businesses, credit card processing, PKI, identity and other hot topics.

- **Network economics.** Real and virtual networks, supply-side *versus* demand-side scale economies, Metcalfe's law, the dominant firm model, the differentiated pricing model Data Protection Act, Distance Selling regulations, business models.
- **Web site design.** Stock and price control; domain names, common mistakes, dynamic pages, transition diagrams, content management systems, multiple targets.
- **Web site implementation.** Merchant systems, system design and sizing, enterprise integration, payment mechanisms, CRM and help desks. Personalisation and internationalisation.
- **The law and electronic commerce.** Contract and tort; copyright; binding actions; liabilities and remedies. Legislation: RIP; Data Protection; EU Directives on Distance Selling and Electronic Signatures.
- **Putting it into practice.** Search engine interaction, driving and analysing traffic; dynamic pricing models. Integration with traditional media. Logs and audit, data mining modelling the user. collaborative filtering and affinity marketing brand value, building communities, typical behaviour.
- **Finance.** How business plans are put together. Funding Internet ventures; the recent hysteria; maximising shareholder value. Future trends.
- **UK and International Internet Regulation.** Data Protection Act and US Privacy laws; HIPAA, Sarbanes-Oxley, Security Breach Disclosure, RIP Act 2000, Electronic Communications Act 2000, Patriot Act, Privacy Directives, data retention; specific issues: deep linking, Inlining, brand misuse, phishing.

Objectives

At the end of the course students should know how to apply their computer science skills to the conduct of e-commerce with some understanding of the legal, security, commercial, economic, marketing and infrastructure issues involved.

Recommended reading

Shapiro, C. & Varian, H. (1998). *Information rules*. Harvard Business School Press.

Additional reading:

Standage, T. (1999). *The Victorian Internet*. Phoenix Press. Klemperer, P. (2004). *Auctions: theory and practice*. Princeton Paperback ISBN 0-691-11925-2.

Hoare Logic

Lecturer: Professor M.J.C. Gordon

No. of lectures: 8

Prerequisite courses: Logic and Proof

Aims

The aim of the course is to show how Hoare logic provides a basis for the formal specification and verification of imperative programs. A simple language will be used to illustrate core ideas. Some current research activities and challenges will be outlined.

Lectures

- **Formal specification of imperative programs.** Formal versus informal methods. Specification using preconditions and postconditions.
- **Axioms and rules of inference.** Hoare logic for a simple language with assignments, sequences, conditionals and while-loops.
- **Loops and invariants.** Various examples illustrating loop invariants and how they can be found.
- **Partial and total correctness.** Hoare logic for proving termination. Variants.
- **Semantics.** Mathematical interpretation of Hoare logic. Soundness and relative completeness.
- **Mechanising program verification.** Weakest preconditions and strongest postconditions. Verification conditions.
- **Automated theorem proving.** Property checking versus proof of correctness. Interactive versus automatic methods.
- **Current reseach.** Recent developments in Hoare logic such as separation logic.

Objectives

At the end of the course students should

- be able to prove simple programs correct by hand and implement a simple program verifier;
- be familiar with the theory and use of Hoare logic and its mechanisation;
- understand the core concepts underlying modern formal program verification.

Recommended reading

Huth, M. & Ryan M. (2004). *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press (2nd ed.).

Mobile and Sensor Systems

Lecturer: Dr C. Mascolo

No. of lectures: 8

Prerequisite courses: Operating Systems, Principles of Communication, Concurrent and Distributed Systems

Aims

This course will cover topics in the area of mobile systems and communications as well as sensor systems and networking and the mixture of the two. It aims to help students develop and understand the additional complexity introduced by mobility and by energy constraints of modern systems.

Lectures

- **Wireless propagation and MAC Layer.** Differences in transmission in wired and wireless medium. Introduction to MAC layer protocols of wireless and mobile systems.
- **Mobile phones architectures and communication.** Introduction to existing mobile phones and operating systems for mobiles.
- **Mobile Infrastructure communication and opportunistic networking protocol.** Description of common communication architectures and protocols for mobile phones and introduction to models of opportunistic networking.
- **Introduction to sensor systems architecture.** sensor systems challenges and applications.
- **Sensor systems MAC layer protocols.** Introduction to concepts related to duty cycling and energy preservation protocols.
- **Sensor systems routing protocols.** Communication protocols, data aggregation and dissemination in sensor networks.
- **Sensor systems programming and reprogramming.** Motivation of sensor reprogramming and approaches to sensor network management and update.
- **Mobile sensing and participatory sensing.** Mobile sensor networks and use of mobile phones as sensors.

Objectives

On completing the course, students should be able to

- describe similarities and differences between standard distributed systems and mobile and sensor systems;

- explain the fundamental tradeoffs related to energy limitations and communication needs in these systems;
- argue for and against different mobile and sensor systems architectures and protocols.

Recommended reading

* Schiller, J. (2003). *Mobile communications*. Pearson (2nd ed.).
* Karl, H. & Willig, A. (2005). *Protocols and architectures for wireless sensor networks*. Wiley.
Agrawal, D. & Zheng, Q. (2006). *Introduction to wireless and mobile systems*. Thomson.

Easter Term 2011: Part II lectures

Business Studies Seminars

Lecturer: Mr J.A. Lang and others

No. of seminars: 8

Aims

This course is a series of seminars by former members and friends of the Laboratory about their real-world experiences of starting and running high technology companies. It is a follow on to the Business Studies course in the Michaelmas Term. It provides practical examples and case studies, and the opportunity to network with and learn from actual entrepreneurs.

Lectures

Eight lectures by eight different entrepreneurs.

Objectives

At the end of the course students should have a better knowledge of the pleasures and pitfalls of starting a high tech company.

Recommended reading

Lang, J. (2001). *The high-tech entrepreneur's handbook: how to start and run a high-tech company*. FT.COM/Prentice Hall.

See also the additional reading list on the Business Studies web page.

System-on-Chip Design

Lecturer: Dr D.J. Greaves

No. of lectures: 12

Prerequisite courses: Computer Design, C and C++, Computer Systems Modelling

Aims

A current-day system on a chip (SoC) consists of several different processor subsystems together with memories and I/O interfaces. This course covers SoC design and modelling techniques with emphasis on architectural exploration, assertion-driven design and the concurrent development of hardware and embedded software. This is the "front end" of the design automation tool chain. (Back end material, such as design of individual gates, layout, routing and fabrication of silicon chips is not covered.)

A percentage of each lecture is used to develop a running example. Over the course of the lectures, the example evolves into a System On Chip demonstrator with CPU and bus models, device models and device drivers. All code and tools are available online so the examples can be reproduced and exercises undertaken. The main languages used are

Verilog and C++ using the SystemC library.

Lectures

- **Verilog RTL design with examples.** Event-driven simulation with and without delta cycles, basic gate synthesis algorithm and design examples. Structural hazards, pipelining, memories and multipliers. [2 lectures]
- **SystemC overview.** The major components of the SystemC C++ class library for hardware modelling are covered with code fragments and demonstrations. Queuing/contention delay modelling. [2 lectures]
- **Basic bus structures.** Bus structure. I/O device structure. Interrupts, DMA and device drivers. Examples. Basic bus bridging.
- **ESL + transactional modelling.** Electronic systems level (ESL) design. Architectural exploration. Firmware modelling methods. Blocking and non-blocking transaction styles. Approximate and loose timing styles. Examples. [2 lectures]
- **ABD: assertions and monitors.** Types of assertion (imperative, safety, liveness, data conservation). Assertion-based design (ABD). PSL/SVA assertions. Temporal logic compilation of fragments to monitoring FSM. [2 lectures]
- **Further bus structures.** Busses used in today's SoCs (OPB/BVCI, AHB and AXI). Glue logic synthesis. Transactor synthesis. Pipeline Tolerance. Network on chip.
- **Engineering aspects: FPGA and ASIC design flow.** Cell libraries. Market breakdown: CPU/Commodity/ASIC/FPGA. Further tools used for design of FPGA and ASIC (timing and power modelling, place and route, memory generators, power gating, clock tree, self-test and scan insertion). Dynamic frequency and voltage scaling.
- **Future approaches** *Only presented if time permits. Non-examinable.* Recent developments: BlueSpec, IP-XACT, Kiwi, Custom processor synthesis.

In addition to these topics, the running example will demonstrate a few practical aspects of device bus interface design, on chip communication and device control software. Students are encouraged to try out and expand the examples in their own time.

Objectives

At the end of the course students should

- be familiar with how a complex gadget containing multiple processors, such as an iPod or Satnav, is designed and developed;
- understand the hardware and software structures used to implement and model inter-component communication in such devices;
- have basic exposure to SystemC programming and PSL assertions.

Recommended reading

* OSCI. *SystemC tutorials and whitepapers*. Download from OSCI `www.systemc.org` or copy from course web site.
Ghenassia, F. (2006). *Transaction-level modeling with SystemC: TLM concepts and applications for embedded systems*. Springer.
Eisner, C. & Fisman, D. (2006). *A practical introduction to PSL*. Springer (Series on Integrated Circuits and Systems).
Foster, H.D. & Krolnik, A.C. (2008). *Creating assertion-based IP*. Springer (Series on Integrated Circuits and Systems).
Grotker, T., Liao, S., Martin, G. & Swan, S. (2002). *System design with SystemC*. Springer.
Wolf, W. (2002). *Modern VLSI design (System-on-chip design)*. Pearson Education.
`http://www.princeton.edu/~wolf/modern-vlsi/`

Temporal Logic and Model Checking

Lecturer: Professor M.J.C. Gordon

No. of lectures: 8

Prerequisite courses: Logic and Proof

Aims

The aim of the course is to introduce the use of temporal logic for specifying properties of hardware and software and model checking as a method for checking that properties hold or finding counter-examples.

Lectures

- **State transition systems.** Representation of state spaces. Reachable states.
- **Checking reachability properties** Fixed-point calculations. Symbolic methods using binary decision diagrams. Finding counter-examples.
- **Examples.** Various uses of reachability calculations.
- **Temporal properties.** Linear and branching time. Intervals. Path quantifiers.
- **Temporal logic.** Brief history (Prior to Pnueli). CTL and LTL. Standarised logics: PSL.
- **Model checking.** Simple algorithms for verifying that temporal properties hold. Reachability analysis as a special case.
- **Applications.** Software and hardware examples.
- **Advanced methods.** Brief introduction to recent development, e.g. Counter-example guided abstraction refinement (CEGAR).

Objectives

At the end of the course students should

- be able to write properties in a variety of temporal logic;
- be familiar with the core ideas of model checking;
- understand what commercial model checking tools can be used for.

Recommended reading

Huth, M. & Ryan M. (2004). *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press (2nd ed.).

Topical Issues

Lecturer: Professor A. Hopper, Dr R.K. Harle and others

No. of lectures: 12

Aims

The aim of this course is to broaden the experience of students by asking expert guest lecturers to discuss real-world issues which are of current interest to the computer community. The course title has changed from “Additional Topics” to “Topical Issues” in 2010–11 for clarity only: the substance of the course remains the same.

Lectures

This course provides an introduction to wide range of Computer Science subjects that are topical. In order to present topics that are timely, the precise coverage may change at the start of Easter term. However, the basis of the 2010–11 course will be the 2009–10 course, which addressed the following topics:

- Location-Aware Computing I
- Location-Aware Computing II
- RFID
- Mobile RealVNC Development
- Computing for the Future of the Planet
- The GPS System
- Coding in Industry
- Developing Commercial Software
- Affective Computing

- Building and Deploying Google Voice Search
- Advances in Search, Information Retrieval and Extraction

Objectives

At the end of the course students should

- realise that the range of issues affecting the computer community is very broad;
- be able to take part in discussions on several subjects at the frontier of modern computer engineering.

Topics in Concurrency

Lecturer: Professor G. Winskel

No. of lectures: 12

Prerequisite course: Semantics of Programming Languages (specifically, an idea of operational semantics and how to reason from it)

Aims

The aim of this course is to introduce fundamental concepts and techniques in the theory of concurrent processes. It will provide languages, models, logics and methods to formalise and reason about concurrent systems.

Lectures

- **Simple parallelism and nondeterminism.** Dijkstra's guarded commands. Communication by shared variables: A language of parallel commands. [1 lecture]
- **Communicating processes.** Milner's Calculus of Communicating Processes (CCS). Pure CCS. Labelled-transition-system semantics. Bisimulation equivalence. Equational consequences and examples. [3 lectures]
- **Specification and model-checking.** The modal mu-calculus. Its relation with Temporal Logic, CTL. Model checking the modal mu-calculus. Bisimulation checking. Examples. [3 lectures]
- **Introduction to Petri nets.** Petri nets, basic definitions and concepts. Petri-net semantics of CCS. [1 lecture]
- **Cryptographic protocols.** Cryptographic protocols informally. A language for cryptographic protocols. Its Petri-net semantics. Properties of cryptographic protocols: secrecy, authentication. Examples with proofs of correctness. [2 lectures]

- **Mobile computation.** An introduction to process languages with process passing and name generation. [2 lectures]

Objectives

At the end of the course students should

- know the basic theory of concurrent processes: non-deterministic and parallel commands, the process language CCS, its transition-system semantics, bisimulation, the modal mu-calculus, Petri nets, languages for cryptographic protocols and mobile computation;
- be able to formalise and to some extent analyse concurrent processes: establish bisimulation or its absence in simple cases, express and establish simple properties of transition systems in the modal mu-calculus, argue with respect to a process language semantics for secrecy or authentication properties of a small cryptographic protocol, formalise mobile computation.

Recommended reading

Comprehensive notes will be provided.

Further reading:

* Aceto, L., Ingolfsdottir, A., Larsen, K.G. & Srba, J. (2007). *Reactive systems: modelling, specification and verification*. Cambridge University Press.
Milner, R. (1989). *Communication and concurrency*. Prentice Hall.
Milner, R. (1999). *Communicating and mobile systems: the Pi-calculus*. Cambridge University Press.
Winskel, G. (1993). *The formal semantics of programming languages, an introduction*. MIT Press.