Computer Science Tripos
Syllabus and Booklist 2008–2009

Contents

Introduction to Part IA **2**
- Entry to the three-year Computer Science Tripos . . . 2
- Computer Science Tripos Part IA . . . 2
- Natural Sciences Part IA students . . . 2
- Politics, Psychology and Sociology Part I students . . . 2
- The curriculum . . . 3

Michaelmas Term 2008: Part IA lectures **4**
- Paper 2: Digital Electronics . . . 4
- Paper 1: Discrete Mathematics I . . . 6
- Paper 1: Foundations of Computer Science . . . 7
- How to Study Computer Science . . . 9
- Paper 2: Operating Systems I . . . 10

Lent Term 2009: Part IA lectures **12**
- Paper 2: Discrete Mathematics II . . . 12
- Paper 1: Floating-Point Computation . . . 13
- Paper 2: Professional Practice and Ethics . . . 15
- Paper 1: Programming Methods . . . 17
- Paper 1: Programming in Java . . . 18
- Paper 1: Software Design . . . 19

Easter Term 2009: Part IA lectures **22**
- Paper 1: Algorithms I . . . 22
- Paper 2: Probability . . . 23
- Paper 2: Regular Languages and Finite Automata . . . 24
- Part IB Assessed Exercise Briefing . . . 25

Preparing to Study Computer Science **25**

Introduction to Part IB **26**

Michaelmas Term 2008: Part IB lectures **27**
Algorithms II 27
Computation Theory 28
Computer Design 30
Concurrent Systems and Applications 31
ECAD 33
Group Project 34
Logic and Proof 35
Mathematical Methods for Computer Science 36
Prolog 38
Software Engineering 39
Unix Tools 40

Lent Term 2009: Part IB lectures **42**
Compiler Construction 42
Computer Graphics and Image Processing 43
Concepts in Programming Languages 45
Databases 46
Digital Communication I 48
Floating-Point Computation 50
Foundations of Functional Programming 51
Programming in C and C++ 53
Semantics of Programming Languages 54

Easter Term 2009: Part IB lectures **56**
Artificial Intelligence I 56
Complexity Theory 57
Economics and Law 59
Introduction to Security 60

Introduction to Part II **62**

Michaelmas Term 2008: Part II lectures **63**
Business Studies 63
Computer Systems Modelling 64
Denotational Semantics 66
Digital Communication II 67
Human-Computer Interaction 69
Information Retrieval 70
Information Theory and Coding 71
Optimising Compilers 73
Quantum Computing 75
Security 76
Types 78

Lent Term 2009: Part II lectures **80**
Advanced Graphics 80
Advanced Systems Topics 81
Bioinformatics 83
Comparative Architectures 84
Computer Vision 86
Digital Signal Processing 88
Natural Language Processing 90
Specification and Verification I 91
Specification and Verification II 92
System-on-Chip Design 93

Easter Term 2009: Part II lectures **95**
Additional Topics 95
Artificial Intelligence II 96
Business Studies Seminars 97
Distributed Systems 98
E-Commerce 99
Topics in Concurrency 101

Introduction to Part IA

Entry to the three-year 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

Besides the lectures listed in this document, students are required to attend 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*, *Geology*, *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, Geology 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.

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.

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 2008: Part IA lectures

Paper 2: Digital Electronics

This Paper 2 course is taken by Part IA Computer Science Tripos students only.

Lecturer: Dr I.J. Wassell

No. of lectures and practical classes: 11 + 7

This course is a prerequisite for Operating Systems I and ECAD (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.
- **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 1: Discrete Mathematics I

Lecturer: Dr P.M. Sewell

No. of lectures: 8

This course is a prerequisite for all theory courses as well as Discrete Mathematics II, Algorithms I, 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 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

* Rosen, K.H. (1999). *Discrete mathematics and its applications.* McGraw-Hill (6th ed.).
* Velleman, D.J. (1994). *How to prove it (a structured approach).* CUP.
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 1: Foundations of Computer Science

Lecturer: Professor L.C. Paulson

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 Manual*. Cambridge University Press. ISBN: 0521794781

How to Study Computer Science

Lecturer: Professor A. Mycroft

No. of lectures: 1

Aims

This single lecture gives an overview of how the course works and a range of advice in how to get the best out of your time at Cambridge.

Lecture

- **Overview.** How Cambridge works. The interaction and split of responsibilities between College and University. The structure of the computer science courses.
- **Support structures.** Directors of studies, tutors, lecturers, supervisors, friends, classmates: your support network and the circumstances in which you should call on each type of person.
- **Being organised.** Why a diary is essential. A typical week's activities analysed and diarised.
- **Lectures.** How to take notes. How to handle the handouts.
- **Supervisions.** What is expected of you. What should you expect.
- **Practical work.** Learning by doing.
- **Health and safety.** How to avoid repetitive strain injury. Support structures for if you get affected.

Objectives

Students should understand the expectations that the Computer Laboratory has of them, should be aware of the resources and support available to them and should understand the importance of good working habits.

Recommended reading

Northedge, A., Thomas, J., Lane, A. & Peasgood, A. (1997). *The sciences good study guide*. Open University.
Fairbanks, A. (1932). *A handwriting manual*. Faber and Faber.
* Pascarelli, E.F. & Quilter, D. (1994). *Repetitive strain injury: a computer user's guide*. New York: Wiley.

Paper 2: Operating Systems I

This Paper 2 course is taken by Part IA Computer Science Tripos students only.

Lecturer: Dr S.M. Hand

No. of lectures: 17 *(Continued into Lent Term)*

Prerequisite course: Digital Electronics.

This course is a prerequisite for the Part IB courses Concurrent Systems and Applications and Introduction to Security, and the Part II courses Advanced System Topics, Distributed Systems and Security.

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 as well as the structure and key functions of the operating system. Case studies will be used to illustrate and reinforce fundamental concepts.

Lectures

- **Computer organization.** History: from vacuum tubes to VLSI. Von Neumann architecture. Hardware/software layers and languages. Operation of a simple computer (processors, memory, buses, devices). Memory: concepts, structures, hierarchy. 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. General I/O architecture. Example devices. Buses: general operation, hierarchy, synchronous *versus* asynchronous. Interrupts. Direct Memory Access. Programming in assembly. [5 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. [2 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 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
- 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

Tanenbaum, A.S. (1990). *Structured computer organisation*. Prentice Hall (3rd ed).
Patterson, D. & Hennessy, J. (1998). *Computer organisation and design*. Morgan Kaufmann (2nd ed.).
* 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 2009: Part IA lectures

Paper 2: Discrete Mathematics II

This Paper 2 course is taken by Part IA Computer Science Tripos students only.

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 1: Floating-Point Computation

Lecturer: Professor A. Mycroft

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: Professional Practice and Ethics

This Paper 2 course is taken by Part IA Computer Science Tripos students only.

Lecturer: Dr R.C. Jennings

No. of lectures: 8

Aims

This course will develop the ethical foundations of good professional practice in computing and generally look at the people side of IT. It will provide a basic survey of ethical theories and discuss the role of professional organisations in maintaining good practice, both in general and then specifically in the computing industry. It will then consider legislation that applies in the computing industry, including three major areas of ethical concern in computing: computer misuse, data privacy and software ownership.

Lectures

- **Ethical theory.** Basic questions in ethics. Survey of ethical theories: authoritarian, intuitionist, egoist, utilitarian, deontologist. Advantages and disadvantages of the two main theories: utilitarian and deontological. [1.5 lectures]

- **Local ethics.** Rules and Regulations made by the Information Syndicate. [0.5 lectures]

- **Professions and professional ethics.** Origin and purpose of professions. Internal regulation *versus* external regulation. Professionalism and large IT projects. Varieties of professional responsibility and professional relations. Professional organisations: ethics and codes of conduct. British Computer Society Code of Conduct. [2 lectures]

- **Relevant legislation: Health and Safety.** Potential problems in the use of VDUs, keyboards and workplaces. [0.2 lectures]

- **Relevant legislation: Disability Discrimination Act.** Ensuring accessibility of Web pages. [0.2 lectures]

- **Environmental legislation:** Waste Electrical and Electronic Equipment Directive. Energy use. [0.2 lectures]

- **Relevant legislation: Computer misuse.** Computer hacking, computer cracking: when does the fun become crime? Computer Misuse Act 1990 and Police and Justice Act 2006. Difficulties with traditional legal concepts. The Human side of security – social engineering. [1.4 lectures]

- **Relevant legislation: Privacy and data protection.** What is Privacy? Computer data and human dignity. The problematic status of information stored on computers. The Data Protection Act 1998. [1 lecture]

- **Relevant legislation: Property ownership.** Theories of property and ownership: Patent, Copyright, and trade secrets. Ownership of computer software: a new problem in intellectual property rights. [1 lecture]

Objectives

At the end of the course students should

- be able to recognise and distinguish different kinds of ethical arguments
- know why professions have codes of conduct, and what is included in the British Computer Society code of conduct
- recognise potential health and safety issues in computing
- be aware of the environmental impact of the IT industry
- understand the need for making web pages accessible
- appreciate the dangers in computer cracking and know the contents of the Computer Misuse Act of 1990, including changes introduced by the Police and Justice Act 2006
- be able to explain the nature of privacy and how it is protected by the Data Protection Act of 1998
- be able to justify the existence of property laws and explain the legal mechanisms which protect software as property
- know the disadvantages of private ownership of software and the legal mechanisms by which private ownership can be blocked

Recommended reading

* Kling, R. (1991). *Computerization and controversy: value conflicts and social choices*. London: Academic Press (2nd ed.).
Forester, T. & Morrison, P. (1990). *Computer ethics: cautionary tales and ethical dilemmas in computing*. Cambridge, MA: MIT Press.
Johnson, D.G. (1985). *Computer ethics*. Englewood, NJ: Prentice Hall.
Johnson, D.G. & Snapper, J.W. (1985). *Ethical issues in the use of computers*. Belmont, CA: Wadsworth.

WWW pages:

Centre for Computing and Social Responsibility (CCSR):
`http://www.ccsr.cse.dmu.ac.uk/`

Computer Professionals for Social Responsibility (CPSR):
`http://www.cpsr.org/`

Paper 1: Programming Methods

Lecturer: Dr R.K. Harle

No. of lectures: 6

Prerequisite courses: Software Design, Programming in Java

Aims

This course is primarily intended to reinforce the concepts introduced in the Software Design lecture course and the Programming in Java practical course. Students will be led through a series of example Java-based projects, from requirements to testing, as well as being introduced to important concepts useful in larger programming projects (design patterns, debuggers, testing methodologies, common Java errors).

Syllabus

This course will cover the elements listed below, but not necessarily in a linear fashion. New material will be discussed in the context of examples and may not be treated in depth.

- **Design cycle illustration.** Specification to requirements to implementation to testing. Identifying Objects. Refactoring.
- **Java class libraries.** Correctly using in-built Java data structures. Extending a standard structure.
- **Design patterns.** History. Usage in Java class libraries and common frameworks. Major patterns with examples in Java.
- **Debuggers.** Basic debugging (breaking, stepping, local variables). JPDA as an example of a software architecture.
- **Testing.** Unit testing. Black box testing. White box testing. Regression testing.
- **Common Java errors.** The need for care with syntax. Numerical overflow and other common problems.

Objectives

At the end of the course students should

- be able to describe and implement the major design patterns
- be able to take a small software project from specification to implementation and test it conforms to the specification
- be able to demonstrate good object-oriented programming skills
- be able to define a suitable test procedure for code they have written
- be familiar with common errors in Java and its associated libraries

- be familiar with debugging terminology

Recommended reading

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: Programming in Java

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

No. of practical classes: 8 × 2-hour sessions

Prerequisite course: Foundations of Computer Science

Companion courses: Software Design, Floating-Point Computation, Programming Methods

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

Aims

The goal of this course is to provide students with the ability to write programs in Java and apply concepts from the Software Design course to concrete examples. 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 and exceptions.** Students will explore the control structures found in Java. Students will also explore exceptions and how they affect the control structure and calling pattern in Java programs.
- **Input/Output.** This class will examine streams. Students will read and write data to and from the filesystem and network.
- **Classes and interfaces.** This class will explore object-oriented programming as expressed in Java. Students will define new classes and instantiate them, as well as specify and provide implementations for Java interfaces.

- **Inheritance and inner classes.** Students will examine code-reuse through inheritance and the use of inner classes for encapsulation. Students will revisit exception handling and define their own exceptions.
- **Packages and access modifiers; Generics; JAR files.** This class will explain how Java packages aid encapsulation and help in the construction of libraries. Students will also explore the packaging of library routines into JAR files and the use of generics to enable generalization and aid code reuse.
- **Extended exercise.** Students will design and begin the implementation of a program design task first described in the Programming Methods course. Students will complete the implementation and testing over the Easter vacation.

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 and use JUnit
- 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: Software Design

Lecturer: Dr A.F. Blackwell

No. of lectures: 12

This course is a prerequisite for Programming Methods (Part IA), Programming in Java (Part IA) and 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.

The course also provides an introduction to the principles of object-oriented programming, as a complementary perspective to the functional and procedural programming styles that have been presented in the Foundations of Computer Science course. Object-oriented programming principles will be applied in the Programming in Java practical classes.

Lectures

- **Introduction to objects.** Classes, instances and state. Fields, methods and constructors. Collections and iterators. Inheritance and interfaces.
- **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

After the first few lectures, students should understand the steps necessary to create simple object-oriented programs in the Java language. They will have the opportunity to practise this in the Programming in Java course that runs concurrently with these lectures.

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 (2nd Edition)*. Microsoft Press.
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 2009: Part IA lectures

Paper 1: Algorithms I

Lecturer: Dr F.M. Stajano

No. of lectures + practicals: 12 + 3

Prerequisite course: Discrete Mathematics I

This course is a prerequisite for Algorithms II 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

- **Algorithm fundamentals.** Algorithm analysis and design. Models of a computer; costs. Asymptotic notation. Computational complexity. Recurrences. Ideas for algorithm design: divide and conquer, dynamic programming, greedy algorithms and other useful paradigms. [Ref: Ch 1, 2, 3, 4, 15, 16] [2–3 lectures]

- **Sorting.** Insertion sort. Merge sort. Heapsort. Quicksort. Other sorting methods. Finding the minimum and maximum. [Ref: Ch 2, 6, 7, 8, 9] [4–5 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] [5–7 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. (2001). *Introduction to Algorithms*. MIT Press (2nd 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 and make extensive use of one of the above textbooks: those not doing so will be severely disadvantaged. The recommended choice is Cormen *et al.* which covers all the topics in the syllabus of Algorithms I and II and, in spite of its quality, is the cheapest. The pointers in the syllabus are to chapters in that book. The other textbooks are all excellent alternatives and are sometimes clearer or more detailed than Cormen, but they are not guaranteed to cover every item in the syllabus. Their relative merits are discussed in the course handout.

Paper 2: Probability

This Paper 2 course is taken by Part IA Computer Science Tripos students only.

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 course in Probability with particular emphasis to further applications in Computer Science.

Lectures

- **Review of elementary probability theory.** Random variables. Discrete and continuous distributions. Means and variances, 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 the concepts of probability and practical knowledge of associated calculations
- be aware of applications of probability across the field of modern computer science

Recommended reading

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

Paper 2: Regular Languages and Finite Automata

This Paper 2 course is taken by Part IA Computer Science Tripos students only.

Lecturer: Professor A.M. Pitts

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.).

Part IB Assessed Exercise Briefing

No. of lectures: 1

Aims

The aim of the exercise is to form a bridge between the Part IA Java course and the use in Part IB of that language.

Lecture

This lecture describes the requirements for the first "tick" of the Part IB course.

Objectives

On completing the exercise students should

- be prepared for the Part IB Concurrent Systems course's coverage of more advanced topics
- have kept their practical Java expertise refreshed so as to be ready for the Lent Term Group Project in Part IB

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 2008: 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. (2001). *Introduction to Algorithms*. MIT Press (2nd 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 and make extensive use of one of the above textbooks: those not doing so will be severely disadvantaged. The recommended choice is Cormen *et al.* which covers all the topics in the syllabus of Algorithms I and II and, in spite of its quality, is the cheapest. The pointers in the syllabus are to chapters in that book. The other textbooks are all excellent alternatives and are sometimes clearer or more detailed than Cormen, but they are not guaranteed to cover every item in the syllabus. Their relative merits are discussed in the course handout.

Computation Theory

Lecturer: Professor A.M. Pitts

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 recursive functions.** Definition and examples. Primitive recursive partial function are computable and total. [1 lecture]
- **Partial recursive functions.** Definition. Existence of a recursive, but not primitive recursive function. Ackermann's function. A partial function is partial recursive if and only if it is computable. [2 lectures]
- **Recursive and recursively enumerable sets.** Decidability and recursive sets. Generability and recursive enumeration. Example of a set that is not recursively enumerable. Example of a recursively enumerable set that is not recursive. Alternative characterisations of recursively enumerable sets as the images and the domains of definition of partial recursive functions. [2 lectures]

Objectives

At the end of the course students should

- be familiar with the register machine and Turing machine 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
- be able to develop simple mathematical arguments to show that particular sets are not recursively enumerable

Recommended reading

* Hopcroft, J.E., Motwani, R. & Ullman, J.D. (2001). *Introduction to automata theory, languages, and computation*. Addison-Wesley (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 Design

Lecturer: Dr S.W. Moore

No. of lectures: 16

Prerequisite courses: Digital Electronics and ECAD

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 the hardware/software interface models and the hardware structures used in designing computers. The first seven lectures are concerned with the hardware/software interface and cover the programmer's model of the computer. The last nine lectures look at hardware implementation issues at a register transfer level.

Lectures

- Introduction to the course and some background history.
- Historic machines. EDSAC *versus* Manchester Mark I.
- Introduction to RISC processor design and the MIPS instruction set.
- MIPS tools and code examples.
- Operating system support including memory hierarchy and management.
- Intel x86 instruction set.
- Java Virtual Machine.
- Memory hierarchy (caching).
- Executing instructions: an algorithmic viewpoint.
- Pipelining and data paths.
- Implementation of a MIPS processor. [2 lectures]
- Internal and external communication.
- Many-core processors
- Data-flow and comments on future directions.

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. & Harris, S. (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 Systems and Applications

Lecturers: Dr S.M. Hand, Dr T.L. Harris, Dr A.F. Blackwell and Dr T.W. Hong

No. of lectures: 21

Prerequisite courses: Programming in Java, Operating Systems

This course is a prerequisite for the Part II courses Distributed Systems and Advanced Systems Topics.

Aims

The aims of this course are (*a*) to introduce the modular design of application software, using the facilities of the Java programming language as a running example, (*b*) to explore the principal mechanisms of inter-process communication, the concept of transactions and their implementation and uses, (*c*) to introduce the need for concurrency control within a process and the techniques for structuring concurrent programs.

Lectures

- **Java review.** Objects and classes. Packages, interfaces, nested classes. Design patterns. [3 lectures, TWH]
- **Distributed systems.** Introduction. TCP and UDP. RPC and RMI. Transactions. Enforcing isolation. Crash recovery and logging. [6 lectures, SMH]
- **Advanced Java.** Reflection and serialisation. Graphical interfaces. Memory management. Native methods and class loaders. Generic types. [5 lectures, TWH]
- **Testing.** Software testing strategies. [1 lecture, AFB]
- **Concurrency.** Threads. Mutual exclusion. Deadlock. Condition synchronization. Worked examples. Low-level synchronization. [6 lectures, TLH]

Objectives

At the end of the course students should

- understand the terminology of object oriented programming and be able to use it with precision
- be able to illustrate the use of object oriented techniques through examples of the kind seen in the Java standard libraries
- understand different mechanisms for communication between applications and evaluate their trade-offs in different scenarios
- understand the concept of transactions and their application in a range of systems
- understand why software testing is not always easy and the techniques used to achieve thorough testing
- understand the need for and implementation of concurrency control

Recommended reading

* Bacon, J. & Harris, T. (2003). *Operating systems* or Bacon, J. (1997) *Concurrent systems* (2nd ed.). Addison-Wesley.
Myers, G.J. (2004). *The art of software testing*. Wiley (2nd ed.).
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/`
Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1994). *Design patterns*. Addison-Wesley.

ECAD

Lecturer: Dr S.W. Moore

No. of lectures and practicals: 4 + 8

Prerequisite course: Digital Electronics

This course is a prerequisite for Computer Design and System-on-Chip Design (Part II).

Aims

This course aims to introduce electronic computer aided design (ECAD) with a particular emphasis on the Verilog hardware description language (HDL). The course covers eight lectures of material but four of the lectures have been replaced by an interactive tutor system to teach Verilog (the Intelligent Verilog Compiler, or *IVC*). The IVC will be used for the first two laboratory sessions and completed as homework. The material the IVC covers is a prerequisite for the remaining seven practical sessions.

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.
- **Chip, board and system testing.** Production testing, fault models, testability, fault coverage, scan path testing.
- **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.

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 design, prototype and debug circuits using Verilog targeted at programmable gate arrays (FPGA)
- understand circuit simulation, synthesis and testing concepts
- appreciate hardware/software codesign

Recommended reading

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

Pointers to sources of more specialist information are included in the lecture notes and on the associated course web page.

Group Project

Lecturer: Dr A.F. Blackwell

No. of lectures: 3

Prerequisite courses: Software Design, Software Engineering, Programming in 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: Professor L.C. Paulson

No. of lectures: 12

This course is a prerequisite for the Part II courses Artificial Intelligence II, Specification and Verification I 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.).

Gentle introduction:
Barwise, J. & Etchemendy, J. (1999). *Language proof and logic*. CSLI Publications.

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

Oppenheim, A.V. & Willsky, A.S. (1997). *Signals and systems*. Prentice Hall.
* Pinkus, A. & Zafrany, S. (1997). *Fourier series and integral transforms*. Cambridge University Press.
Mitzenmacher, M. & Upfal, E. (2005). *Probability and computing: randomized algorithms and probabilistic analysis*. Cambridge University Press.
* Ross, S.M. (2002). *Probability models for computer science*. Harcourt/Academic Press.

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 software.** Examples of catastrophic failure; particular problems with real-time systems; the difficulty of achieving ultra-high reliability; 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.
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 little 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 background and shell basics.** Brief review of Unix history and design philosophy. Inter-process communication mechanisms and conventions (command-line arguments, environment variables, files, directories, plain-text format, pipes, standard I/O, signals, process groups, locale). Using the shell (`bash`) for file system navigation, program invocation, piping and job control. Finding documentation.

- **Shell script programming and configuration.** Efficient command entry with history and alias functions. Regular expressions. The shell as a simple scripting language with parameter substitution, control structures, functions. Customizing user environments with start-up scripts. 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.
- **Software development tools.** C compiler, linker and debugger. Makefiles, packaging and compression tools.
- **Revision control systems.** Patch generation and application, RCS, Subversion.
- **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. [2 lectures]
- **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 used
- 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 2009: Part IB lectures

Compiler Construction

Lecturer: Professor A. Mycroft

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.

Computer Graphics and Image Processing

Lecturer: 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.

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.
* 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*.
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`

Digital Communication I

Lecturer: Dr A.W. Moore

No. of lectures: 12

This course is a prerequisite for the Part II courses Digital Communication II, Distributed Systems and Security.

Aims

The aims of this course are to develop an understanding of communications networks from a wide perspective, within a framework of principles rather than technologies or architectures. Technologies and architectures will, however, be used as examples and motivation.

Lectures

- **Scope.** Two example systems: Ethernet and the telephone system: basic operation; common issues; differing constraints; differing approaches.

- **Partitioning the problem.** Abstraction, service *versus* implementation; layering as a restricted form of abstraction; motivation for layering; the channel as an abstraction; layered channels.

- **Fundamental transmission.** Emphasis on the service provided by physical channel; limitations: noise, attenuation. Channel capacity (bandwidth). Modulation techniques for digital systems.

- **Coding.** Coding as a general concept: modulation as a form of coding, A/D, D/A, error correcting and detecting codes, other forms of coding, relation to layering.

- **Multiplexing.** Basic definitions, FDM, synchronous and asynchronous TDM. Circuit switching, packet switching, ATM. Shared media networks with particular emphasis on media access control. Packet scheduling. Non orthogonal multiplexing. Multiplexing and channel characteristics.

- **Switching and routing.** Introduction from LAN perspective (repeaters, bridges, routers). Fundamental view of switching extended to telephone network, connectionless *versus* connection oriented.

- **Protocols and state.** Imperfect view of state at far end of channel. ARQ as an example of an error control protocol; sliding window ARQ as an example of a flow control protocol; flow control in general: X.25 as an example.

- **Naming, addressing and routing.** Service access points, binding. Hierarchical *versus* flat address spaces. Routing classifications and algorithms.

- **The Internet.** Internet architecture, context of development, addressing and routing, transmission control protocol (TCP), higher layer protocols, evolution.

- **Standards.** Role of standards, dynamics of standards process, standards bodies.

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 physical transmission systems

- understand the general principles behind multiplexing, addressing, routing and 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 the internal workings of the Internet

Recommended reading

* 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.).
Schwartz, M. (1987). *Telecommunication networks: protocols, modeling and analysis*. Addison-Wesley.

Floating-Point Computation

Lecturer: Professor A. Mycroft

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.

Foundations of Functional Programming

Lecturer: Dr M.J. Parkinson

No. of lectures: 12

This course is a prerequisite for Types (Part II).

Aims

This course aims (*a*) to show how lambda-calculus and related theories can provide a foundation for a large part of practical programming, (*b*) to present students with one particular type analysis algorithm so that they will be better able to appreciate the Part II Types course, and (*c*) to provide a bridge between the Part IA Foundations of Computer Science course and the theory options in Part II.

Lectures

- **Introduction.** Combinators. Constants and Free Variables. Reduction. Equality. The Church–Rosser theorem. Normal forms. [1 lecture]
- **The Lambda calculus.** Lambda-terms, alpha and beta conversions. Free and bound variables. Abbreviations in the notation. Pure and applied lambda calculi. Relationship between combinators, lambda calculus and typical programming languages. Eager and Lazy evaluation. [2 lectures]
- **Encoding of data.** Booleans, tuples, lists and trees, numbers. The treatment of recursion: the *Y* combinator and its use. [2 lectures]
- **Implementations of lambda calculus.** Conversion from lambda-calculus to combinators. Combinator reduction as tree-rewrites. The treatment of lambda-bindings in an interpreter: the *environment*. Closures. ML implementation of lambda-calculus. SECD machine. Brief survey of performance issues. [3 lectures]
- **Relationship between this and Turing computability.** The halting problem etc. [0.5 lecture]
- **Modelling imperative programming styles.** Handling state information and the continuation-passing style. Return address seen as an additional continuation parameter. [2.5 lectures]
- **Let-polymorphism** reviewed following the Part IA coverage of ML. Unification. A type-reconstruction algorithm. Decidability and potential costs. [1 lecture]

Objectives

At the end of the course students should

- understand the rules for the construction and processing of combinatory terms and terms in the lambda calculus
- know how to model all major aspects of general-purpose computation in terms of these primitives
- know how lambda terms may be efficiently interpreted by machine
- be able to derive ML-style type judgements for languages based upon the lambda-calculus

Recommended reading

Hindley, J.R. & Seldin, J.P. (1986). *Introduction to combinators and lambda-calculus*. Cambridge University Press (now out of print but try a library).
Revesz, G.E. (1988). *Lambda calculus, combinators and functional programming*. Cambridge University Press (now out of print but try a library).

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. STL generic programming. 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. (1994). *The design and evolution of C++.* Addison-Wesley.
Lippman, S.B. (1996). *Inside the C++ object model.* Addison-Wesley.

Semantics of Programming Languages

Lecturer: Dr P.M. Sewell

No. of lectures: 12

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

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 2009: Part IB lectures

Artificial Intelligence I

Lecturer: Dr M. Jamnik

No. of lectures: 12

Prerequisite courses: Algorithms. 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, are likely to be helpful although not essential.

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.

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? What does this course cover, and what is left out?
- **Agents.** A unifying view of AI systems. How could we approach the construction of such a system? How would we judge an AI system? What should such a system do and how does it interact with its environment?
- **Search I.** How can search serve as a fundamental paradigm for intelligent problem-solving? Simple, *uninformed search* algorithms.
- **Search II.** More sophisticated *heuristic search* algorithms.
- **Search III.** Search in an adversarial environment. Computer game playing.
- **Constraint satisfaction problems.**
- **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.
- **Planning.** Methods for planning in advance how to solve a problem. The partial-order planning algorithm.

- **Learning.** A brief introduction to supervised learning from examples, focusing on neural networks. [4 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 (2nd ed.).
Cawsey, A. (1998). *The essence of artificial intelligence*. Prentice Hall.
Luger, G.F. & Stubblefield, W.A. (1998). *Artificial intelligence: structures and strategies for complex problem solving*. Addison-Wesley.
Dean, T., Allen, J. & Aloimonos, Y. (1995). *Artificial intelligence: theory and practice*. Benjamin/Cummings.

Complexity Theory

Lecturer: Dr T.G. Griffin

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.
Sipser, M. (1997). *Introduction to the theory of computation*. PWS.

Economics and Law

Lecturers: Professor R.J. Anderson 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, 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 implications for innovation.

- **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, 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.
Galbriath, 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 Security

Lecturer: Dr M.G. Kuhn

No. of lectures: 8

Prerequisite courses: Discrete Mathematics, Operating Systems

This course is a prerequisite for the Part II courses Distributed Systems and Security.

Aims

This course is a broad introduction to both computer security and cryptography. It covers important basic concepts and techniques.

Lectures

- **Cryptography.** Introduction, terminology, classic ciphers, perfect secrecy, Vernam cipher, pseudo-random functions and permutations, computational security, random bit generation, secure hash functions, birthday problem.
- **Symmetric cryptography.** Block ciphers, modes of operation, message authentication codes, applications of secure hash functions.
- **Asymmetric cryptography.** Key-management problem, signatures and certificates, number theory revisited, discrete logarithm problem, Diffie–Hellman key exchange, ElGamal encryption and signature, hybrid cryptography.

- **Authentication techniques.** Passwords, one-way and challenge–response protocols, Needham–Schroeder, protocol failure examples, hardware tokens.
- **Access control.** Discretionary access control in POSIX and Windows, elevated rights and setuid bits, capabilities, mandatory access control, covert channels, Clark–Wilson integrity.
- **Operating system security.** OS security functions, trusted computing base, security evaluation methodology and standards.
- **Software security.** Malicious software, viruses, common implementation vulnerabilities, buffer overflows, meta characters, integer overflows, race conditions, side channels.
- **Network security.** TCP/IP vulnerabilities, firewalls. [0.5 lecture]
- **Security policies and management.** Application-specific security requirements, targets and policies, security management. [0.5 lecture]

Objectives

By the end of the course students should

- be familiar with the most common security terms and concepts
- have a basic understanding of the most 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

* Gollmann, D. (2006). *Computer Security*. Wiley (2nd ed.).
Stinson, D. (2005). *Cryptography: theory and practice*. Chapman & Hall/CRC (3rd ed.).

Further reading:

Anderson, R. (2001). *Security engineering: a guide to building dependable distributed systems*. Wiley.
Schneier, B. (1995). *Applied cryptography: protocols, algorithms, and source code in C*. Wiley (2nd 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.).

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 2008: Part II lectures

Business Studies

Lecturer: Mr J.A. Lang

No. of lectures: 8

Or "How to Start and Run a Computer Company"

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/`

Computer Systems Modelling

Lecturer: Dr R.J. Gibbens

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. [2 lectures]

- **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. [2 lectures]

- **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]

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

Reference books

* Ross, S.M. (2002). *Probability models for computer science.* Academic Press.
Jain, A.R. (1991). *The art of computer systems performance analysis.* Wiley.
Kleinrock, L. (1975). *Queueing systems, vol.* 1. *Theory.* Wiley.

Denotational Semantics

Lecturer: Dr M. P. Fiore

No. of lectures: 10 *(including revision)*

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

Aims

The aims of this course are to introduce domain theory and denotational semantics, and 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.** ω-complete partial orders (cpos) and least elements. ω-continuous functions and least fixed points.
- **Constructions on domains.** Products of domains. Function domains. Flat 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.
- **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.
- **Full abstraction.** Failure of full abstraction for the domain model. PCF with parallel or. Recent developments.

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

Books:

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

Papers:

Fiore, M., Jung, A., Moggi, E., O'Hearn, P., Riecke, J., Rosolini, G., and Stark, I. (1996). Domains and denotational semantics: History, accomplishments and open problems. *Bulletin of EATCS, 59:227–256*.
Milner, R. (1977). Fully abstract models of typed lambda-calculi. *Theoretical Computer Science, 4:1–22*.
Ong, C.-H. (1995). Correspondence between operational and denotational semantics. *Handbook of Logic in Computer Science, Vol. 4, pp. 269–356*.
Plotkin, G. (1977). LCF considered as a programming language. *Theoretical Computer Science, 5:223–256*.
Scott, D. (1969). A type-theoretical alternative to CUCH, ISWIM, OWHY. (In *Theoretical Computer Science, 121:411–440, 1993*.)

Digital Communication II

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 and Advanced Systems Topics.

Aims

This course aims to provide a detailed understanding of how communications systems operate, through the examples including the Internet amongst others, and presents ways to build such systems. It also covers a selection of topics which relate to recent trends in digital communications systems.

Lectures

- **Introduction.** Course overview. Abstraction, layering. The structure of real networks.
- **The Telephone Net.** It has been around 100 years, and there are important lessons in how it survived and evolved.
- **The Internet.** It is about 25 years old, and looking decidedly shaky. A quick review of where it is at.
- **Asynchronous Transfer Mode networks.** A bold attempt to mix Telephone and Internet.
- **Modular functionality for communications.** Some systems design paradigms, often orthogonal to layers.
- **Naming and addressing.** Reviewing Who is where?
- **A list of common protocols in use today.** To see if we can spot design patterns?
- **Mapping onto common implementation approaches.**
- **Routing.** How many ways can we work out how to get from A to B? [2 lectures]
- **Error control.** What do we do when things go wrong? Retransmit, or pre-transmit?
- **Flow control.** 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. [2 lectures]
- **Switched networks.** What does a switch do and how? [2 lectures]
- **Integrated Service Packet Networks for IP.** APIs to Quality of Service. Scheduling and queue management algorithms for packet forwarding. What about routing with QoS. [2 lectures]
- **The big picture for managing traffic.** Economics, policy and a little MPLS. [2 lectures]

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, eXtensibility, or, ok, Xen:), yield management, and Zeroconf.

Recommended reading

* Keshav, S. (1997). *An engineering approach to computer networking*. Addison-Wesley (1st ed.). ISBN 0201634422
Alternatives to Keshav:
Davie, B.S., Peterson, L.L. & Clark, D. (1999). *Computer networks: a systems approach*. Morgan Kaufmann (2nd ed.). ISBN 1558605142
Stevens, W.R. (1994). *TCP/IP illustrated, vol. 1: the protocols*. Addison-Wesley (1st ed.). ISBN 0201633469

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

- **Interaction techniques.** Historical survey of user interface techniques, leading to the current industry standards of direct manipulation and platform-specific style guidelines.
- **Heuristic evaluation.** A basic approach to systematic analysis of usability from an engineer's perspective.
- **Psychological user models.** Black box models of human performance, including perception, motor control, memory and problem-solving.
- **Quantitative analysis of performance.** The Model Human Processor, Keystroke Level Model, and GOMS descriptions of user performance.
- **Modelling of system understanding.** Mental models and metaphor, use of design prototypes, controlled experiments.
- **Cognitive walkthrough.** Evaluation from the perspective of a novice learning to use the system.
- **Task analysis and design.** Contextual and qualitative studies, use-case driven design.
- **Research techniques.** Cognitive dimensions of notations, CSCW, ubiquitous computing, new interaction techniques, programmability.

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.).
* Carroll, J.M. (ed.) (2003). *HCI models, theories and frameworks: toward a multi-disciplinary science*. Morgan Kaufmann.

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

* Baeza-Yates, R. & Ribiero-Neto, B. (1999). *Modern information retrieval*. Reading, MA: Addison-Wesley and ACM Press.
* Salton, G. & McGill, M. (1983). *Introduction to modern information retrieval*. New York: McGraw-Hill.
Spärck Jones, K. & Willett, P. (eds.) (1997). *Readings in information retrieval*. San Francisco: Morgan Kaufmann.

Information Theory and Coding

Lecturer: Dr J.G. Daugman

No. of lectures + examples classes: 11 + 1

Prerequisite courses: Probability, Discrete Mathematics, Mathematical Methods for Computer Science

Aims

The aims of this course are to introduce the principles and applications of information theory. The course will study how information is measured in terms of probability and entropy, and the relationships among conditional and joint entropies; how these are used to calculate the capacity of a communication channel, with and without noise; coding schemes, including error correcting codes; how discrete channels and measures of information generalize to their continuous forms; the Fourier perspective; and extensions to wavelets, complexity and compression.

Lectures

- **Foundations: probability, uncertainty, information.** How concepts of randomness, redundancy, compressibility, noise, bandwidth, and uncertainty are related to information. Ensembles, random variables, marginal and conditional probabilities. How the metrics of information are grounded in the rules of probability.
- **Entropies defined, and why they are measures of information.** Marginal entropy, joint entropy, conditional entropy, and the Chain Rule for entropy. Mutual information between ensembles of random variables. Why entropy is the fundamental measure of information content.
- **Source coding theorem; prefix, variable-, and fixed-length codes.** Symbol codes. The binary symmetric channel. Capacity of a noiseless discrete channel. Error correcting codes.
- **Channel types, properties, noise, and channel capacity.** Perfect communication through a noisy channel. Capacity of a discrete channel as the maximum of its mutual information over all possible input distributions.
- **Continuous information; density; noisy channel coding theorem.** Extensions of the discrete entropies and measures to the continuous case. Signal-to-noise ratio; power spectral density. Gaussian channels. Relative significance of bandwidth and noise limitations. The Shannon rate limit and efficiency for noisy continuous channels.
- **Fourier series, convergence, orthogonal representation.** Generalized signal expansions in vector spaces. Independence. Representation of continuous or discrete data by complex exponentials. The Fourier basis. Fourier series for periodic functions. Examples.
- **Useful Fourier theorems; transform pairs. Sampling; aliasing.** The Fourier transform for non-periodic functions. Properties of the transform, and examples. Nyquist's Sampling Theorem derived, and the cause (and removal) of aliasing.
- **Discrete Fourier transform. Fast Fourier Transform algorithms.** Efficient algorithms for computing Fourier transforms of discrete data. Computational complexity. Filters, correlation, modulation, demodulation, coherence.
- **The quantized degrees-of-freedom in a continuous signal.** Why a continuous signal of finite bandwidth and duration has a fixed number of degrees-of-freedom. Diverse illustrations of the principle that information, even in such a signal, comes in quantized, countable, packets.
- **Gabor-Heisenberg-Weyl uncertainty relation. Optimal "Logons".** Unification of the time-domain and the frequency-domain as endpoints of a continuous deformation. The Uncertainty Principle and its optimal solution by Gabor's expansion basis of "logons". Multi-resolution wavelet codes. Extension to images, for analysis and compression.

- **Kolmogorov complexity. Minimal description length.** Definition of the algorithmic complexity of a data sequence, and its relation to the entropy of the distribution from which the data was drawn. Fractals. Minimal description length, and why this measure of complexity is not computable.

Objectives

At the end of the course students should be able to

- calculate the information content of a random variable from its probability distribution
- relate the joint, conditional, and marginal entropies of variables in terms of their coupled probabilities
- define channel capacities and properties using Shannon's Theorems
- construct efficient codes for data on imperfect communication channels
- generalize the discrete concepts to continuous signals on continuous channels
- understand Fourier Transforms and the main ideas of efficient algorithms for them
- describe the information resolution and compression properties of wavelets

Recommended reading

* Cover, T.M. & Thomas, J.A. (1991). *Elements of information theory*. New York: Wiley.

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.
- **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.).

Quantum Computing

Lecturer: Dr A. Dawar

No. of lectures: 8

Prerequisite courses: Mathematical Methods for Computer Science, Computation Theory

Aims

The aims of the course are to introduce students to the basics of the quantum model of computation. The model will be used to study algorithms for searching and factorisation. Issues in the complexity of computation will also be explored.

Lectures

- **Bits and qubits.** Introduction to quantum states with motivating examples. Comparison with classical discrete state systems.
- **Linear algebra.** Review of linear algebra. Vector spaces, linear operators, Dirac notation.
- **Quantum mechanics.** Postulates of quantum mechanics. Evolution and measurement. Entanglement.
- **Quantum computation.** Models of quantum computation. Quantum circuits, finite state systems, machines and algorithms.
- **Some applications.** Applications of quantum infomation. Bell States, quantum key exchange, quantum teleportation.
- **Quantum search.** Grover's search algorithm. Analysis and lower bounds.
- **Factorisation.** Shor's algorithm for factorising numbers and analysis. Quantum Fourier transform.
- **Quantum complexity.** Quantum complexity classes and their relationship to classical complexity. Comparison with probabilistic computation.

Objectives

At the end of the course students should

- understand the quantum model of computation and how it relates to quantum mechanics
- be familiar with some basic quantum algorithms and their analysis
- see how the quantum model relates to classical models of computation

Recommended reading

* Nielsen, M.A. & Chuang, I.L. (2000). *Quantum computation and quantum information*. Cambridge University Press.
Mermin, N.D. (2007). *Quantum computer science*. Cambridge University Press.
Kitaev, A.Y., Shen, A.H. & Vyalyi, M.N. (2002). *Classical and quantum computation*. AMS.

Security

Lecturer: Professor R.J. Anderson

No. of lectures: 16

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

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 mode, non-interference and non-deducibility. Composability. Real MLS systems, and real 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. Worms and viruses.

- **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

- **Signal processing issues.** Biometrics: fingerprint readers, iris scanners, signature recognition. Information hiding: watermarks, digital fingerprints, steganography; jam-resistance and low-probability of-intercept communications.

- **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, the filter generator, 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, SSH, SET, PGP and PEM. The BAN logic applied to public key systems.

- **Rights management, interoperability control and economics.** Copyright management systems; accessory control systems; the TC architecture. Security economics. Tensions between security and competition.

- **Security engineering.** Why is security management hard? Risk reduction *versus* transference; due diligence and the role of insurance. Threat trees; risk models; robustness; dependability; engineering disciplines. Verification and evaluation:

TCSEC, ITSEC and the Common Criteria. Interaction with the regulatory environment, from data protection through RIP to export control.

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 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.
Davies, D.W. & Price, W.L. (1984). *Security for computer networks*. Wiley.
Beker, H. & Piper, F. (1982). *Cipher systems*. Northwood.
Cohen, F.B. (1994). *A short course on computer viruses*. Wiley (2nd ed.).

Types

Lecturer: Professor A.M. Pitts

No. of lectures: 8

Prerequisite courses: Semantics of Programming Languages, Foundations of Functional Programming

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 2009: Part II lectures

Advanced Graphics

Lecturer: Dr P.A. Benton

No. of lectures: 8

Prerequisite course: Computer Graphics and Image Processing

Aims

This course provides students with a solid grounding in a variety of three-dimensional rendering and modeling mechanisms. Topics addressed will range from low-level implementation to advanced rendering concepts and current research in computational geometry.

As there will be only eight lectures this term, interested students are expected to pursue further reading and research independently; some references and resources will also be provided during lectures.

Lectures

- **Fundamentals.** OpenGL and JOGL. Scene graphs and bounding boxes. Polygon data structures. Simple hierarchical modeling. [1 lecture]
- **Computational geometry.** Geometric methods for ray tracing and Computational Solid Geometry. Surface interrogations, convexity, centroids, curvature, etc. NURBS. [3 lectures]
- **Rendering.** Ray tracing revisited; reflection, refraction, ray-traced effects, etc. Radiosity. [1 lecture]
- **Subdivision surfaces.** Univariate and bivariate subdivision schemes. [1 lecture]
- **Implicit surfaces.** Octrees and marching cubes. [1 lecture]
- **Shaders.** Current trends in hardware-accelerated 3D rendering. [1 lecture]

Objectives

On completing the course, students should be able to

- produce equations for each geometric primitive, derive a ray/primitive intersection algorithm for each, describe how each can be approximated by polygons
- be able to explain the basic radiosity algorithm
- define NURBS basis functions, understand the use of NURBS curves and surfaces in 2D and 3D modelling

- understand and describe a number of methods of surface modeling and generation, including computational solid geometry, subdivision surfaces and implicit surfaces
- understand algorithms for common operations on geometry, such as finding discrete curvature, convexity operations and others

Recommended reading

Students should expect to refer to one or more of these books. You will also be expected to draw on online resources. Other books may be added to this list as the course evolves.

You may find that it is not necessary to purchase these books, as virtually all of key facts are available on the interweb; however, if you are interested in advanced computer graphics, these texts are generally considered essential for any well-stocked reference shelf. And always remember that wikipedia is not a citable source and is only as correct as its latest update.

Watt, A. (1999). *3D computer graphics*. Addison-Wesley (3rd ed).
Foley, J.D., van Dam, A., Feiner, S.K. & Hughes, J.F. (1990). *Computer graphics: principles and practice*. Addison-Wesley (2nd ed.).
de Berg, M., Cheong, O., van Kreveld, M. & Overmars, M. (2008). *Computational geometry: algorithms and applications*. Springer (3rd ed.).

Advanced Systems Topics

Lecturers: Dr S.M. Hand, Dr T.G. Griffin and Dr C. Mascolo

No. of lectures: 16

Prerequisite courses: Concurrent Systems and Applications, Operating Systems, Digital Communication II

Aims

This course will cover a selection of topics in the general area of systems including networking, operating systems, database systems, mobility and sensor systems. It aims to help students develop and understand complex systems and interactions, and to prepare them for emerging systems architectures.

Lectures

- **Internet routing protocols.** Internet routing protocols from a distributed systems perspective. We will cover the five most commonly used protocols — RIP, EIGRP, OSPF, IS-IS (all roughly based on the shortest-paths model) and BGP (which has evolved organically in the interdomain context). Convergence, scalability, and stability are the main concerns. Live-lock in BGP will be described. [TGG, 6 lectures]

- **Advanced operating systems.** Extensible systems. Capability systems. Distributed and persistent virtual memory. Hypervisors and machine virtualization. Networked storage architectures (NAS, SAN). Database and filing system topics. [SMH, 6 lectures]

- **Mobile and sensor systems.** This section concerns systems and communication in decentralized mobile networks and sensor systems. In mobile systems we will concentrate on issues arising from the application of these systems to vehicular networking, human ad hoc connectivity and decentralized content distribution. In terms of sensor systems we will look at energy and communication tradeoffs for of both mobile and fixed devices with application in various scenarios such as human and animal activity monitoring and environmental monitoring. [CM, 4 lectures]

Objectives

On completing the course, students should be able to

- describe similarities and differences between current Internet routing protocols
- explain the fundamental tradeoffs in attempting to achieve stability, scalability, and fast convergence for routing protocols
- describe three techniques for supporting extensibility
- argue for and against distributed shared virtual memory
- discuss the tradeoffs between energy and communication in mobile and sensor systems.

Recommended reading

Singhal, M. & Shivaratri, N. (1994). *Advanced concepts in operating systems: distributed, database, and multiprocessor operating systems*. McGraw-Hill.
Stonebraker, M. & Shivaratri, N. (1998). *Readings in database systems*. Morgan Kaufmann (3rd ed.). ISBN 1-55860-523-1
Hennessy, J. & Patterson, D. (2006). *Computer architecture: a quantitative approach* (Chapter 4 in particular). Morgan Kaufmann (4th ed.).
Hedrick, C. (1988). *RFC 1058 — Routing information protocol*. `http://www.ietf.org/rfc/rfc1058.txt`.

Basu, A. & Riecke, J.G. (2001). *Stability issues in OSPF routing*. `http://www.sigcomm.org/sigcomm2001/p18-basu.pdf`.
Huston, G. (2007). *Damping BGP*. The ISP Column, June 2007. `http://www.potaroo.net/ispcol/2007-06/dampbgp.html`

Bioinformatics

Lecturer: Dr P. Liò

No. of lectures: 12

Aims

This course focuses on algorithms used in Bioinformatics and System Biology. Most of the algorithms are general and can be applied on multidimensional and noisy data from other fields. All the necessary biological terms and concepts useful for the course and the examination will be given in the lectures.

Lectures

- **Introduction to Bioinformatics**. Primer on molecular biology and "omics". Open problems and types of data.
- **Sequence alignment I**. Dynamic programming. Global and local alignment algorithms. Scoring matrices.
- **Sequence alignment II**. Block alignment and four Russians speedup. Multiple alignment.
- **Database search**. Comparing a sequence against a database. Blast family, PatternHunter, Using spaced seeds.
- **Algorithms for evolutionary trees I**. Parsimony methods.
- **Algorithms for evolutionary trees II**. Distance methods.
- **Algorithms for evolutionary trees III**. Maximum Likelihood methods; bootstrap test.
- **Information theory**. Markov properties of genome sequences.
- **Applications of Hidden Markov Models in Bioinformatics**. Examples on gene and protein structure prediction.
- **Microarray data analysis**. Steady state and time series microarray data. Clustering methods.
- **Introduction to system biology I**. Algorithms for generating genetic networks from microarray data. Identifying regulatory elements using microarray data.

- **Introduction to system biology II.** Analysis of Genetic and biochemical networks; Gillespie family of algorithms.

Objectives

At the end of this course students should

- understand Bioinformatics terminology
- be able to work with with bioinformaticians and biologists
- have some experience of the data used in Bioinformatics
- master the most important algorithms in the field

Recommended reading

* Jones N.C. & Pevzner, P.A. (2004). *An introduction to bioinformatics algorithms*. MIT Press.
Felsenstein, J. (2003). *Inferring phylogenies*. Sinauer Associates.

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, market trends and application characteristics. Power and performance metrics. Environmental impact.
- **Review**. Instruction-set principles; the scalar pipeline
- **Overview of architectural techniques**. Classification of approaches; examples and pitfalls; Amdahl's law
- **Advanced pipelining**. Pipeline hazards; exceptions; branch prediction; avoiding branches; fetch issues; trace cache; limitations of pipelining and optimal pipeline depth [2 lectures]

- **Superscalar techniques.** Instruction-Level Parallelism (ILP); static and dynamic scheduling; register renaming; load/store instruction ordering; advanced speculation techniques; example microarchitectures [2 lectures]

- **Software approaches to exploiting ILP.** VLIW architectures; dynamic binary code translation; exploiting compiler hints

- **Multithreaded processors.** Coarse-grained, fine-grained, SMT

- **The memory hierarchy.** Caches; programming for caches; prefetching; software managed memories; main memory [3 lectures]

- **Vector processors.** Vector machines; short vector/SIMD instruction set extensions; stream processing

- **Chip multiprocessors.** Examples of multi-core processors; Memory, interconnect and programming challenges; performance limits

- **Special-purpose architectures.** Converging approaches to computer design

- **Review and current developments**

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 a basic 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 Vision

Lecturer: Dr J.G. Daugman

No. of lectures + examples classes: 15 + 1

Prerequisite courses: Mathematical Methods for Computer Science, Probability

Aims

The aims of this course are to introduce the principles, models and applications of computer vision, as well as some mechanisms used in biological visual systems that may inspire design of artificial ones. The course will cover: image formation, structure, and coding; edge and feature detection; neural operators for image analysis; texture, colour, stereo, and motion; wavelet methods for visual coding and analysis; interpretation of surfaces, solids, and shapes; data fusion; probabilistic classifiers; visual inference and learning. Several of these issues will be illustrated in the topic of 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, CCD cameras, framegrabbers. Elementary operations on image arrays; coding and information measures.
- Biological visual mechanisms from retina to cortex. Photoreceptor sampling; receptive field profiles; spike trains; channels and pathways. Neural image encoding operators.
- Mathematical operators for extracting image structure. Finite differences and directional derivatives. Filters; convolution; correlation. 2D Fourier domain theorems.
- Edge detection operators; the information revealed by edges. The Laplacian operator and its zero-crossings. Logan's theorem.
- Multi-resolution representations. Active contours. 2D wavelets as visual primitives.
- Higher level visual operations in brain cortical areas. Multiple parallel mappings; streaming and divisions of labour; reciprocal feedback through the visual system.
- Texture, colour, stereo, and motion descriptors. Disambiguation and the achievement of invariances.
- 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; 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.
- Lessons from neurological trauma and visual deficits. Visual illusions and what they may imply about how vision works.
- Bayesian inference in vision; knowledge-driven interpretations. Classifiers. Probabilistic methods in vision.
- Object-centred coordinates. Appearance-based *versus* volumetric model-based vision.
- Vision as a set of inverse problems; mathematical methods for solving them: energy minimization, relaxation, regularization.
- 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 analyse the robustness, brittleness, generalizability, and performance of different approaches in computer vision
- be able to describe key aspects of how biological visual systems encode, analyse, and represent visual information
- be able to think of ways in which biological visual strategies might be implemented in machine vision, despite the enormous differences in hardware
- understand in depth at least one major practical application problem, such as face recognition, detection, and interpretation

Recommended reading

* 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 Information Theory and Coding and 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 a particular view on audio-visual data compression techniques. 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).
- **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 and audio coding standards.** A/μ-law coding, delta coding, JPEG photographic still-image compression.
- **MPEG standards.** Motion compensation, video coding, MP3.

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
- be able to explain the above in time and frequency domain representations
- be competent to use filter-design software
- be able to visualize and discuss digital filters in the z-domain
- be able to use the FFT for convolution, deconvolution, filtering
- be able to implement, apply and evaluate simple DSP applications in MATLAB
- apply transforms that reduce correlation between several signal sources
- understand and explain limits in human perception that are exploited by lossy compression techniques
- provide a good overview of the principles and characteristics of several widely-used compression techniques and standards for audio-visual signals

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.

Natural Language Processing

Lecturer: Dr A.L. Korhonen

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, knowledge-based *versus* probabilistic approaches.
- **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.** Machine translation, email response, spoken dialogue systems.

Objectives

At the end of the course students should

- be able to describe the architecture of and basic design for a generic NLP system "shell"
- be able to discuss the current and likely future performance of several NLP applications, such as machine translation and email response
- be able to describe briefly a fundamental technique for processing language for several subtasks, such as morphological analysis, parsing, word sense disambiguation etc.

- understand how these techniques draw on and relate to other areas of (theoretical) computer science, such as formal language theory, formal semantics of programming languages, or theorem proving

Recommended reading

* Jurafsky, D. & Martin, J. (2000). *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`

Specification and Verification I

Lecturer: Professor M.J.C. Gordon

No. of lectures: 12

Prerequisite course: Logic and Proof

This course is a prerequisite for Specification and Verification II.

Aims

The aim of the course is to motivate and illustrate the use of rigorous methods and mechanised tools for reasoning about the functional behaviour of imperative programs. A goal is to show the similarities and differences between hardware and software verification.

Lectures

- **Program specification.** Partial and total correctness. Hoare notation. [2 lectures]
- **Program verification.** Axioms and rules of Floyd–Hoare logic. Discussion of soundness and completeness. [4 lectures]
- **Mechanised program verification.** Annotation. Verification condition generation. [3 lectures]
- **Program refinement.** Definition of the refinement relation. Derivation of laws. [1 lecture]
- **Semantic embedding.** Deep *versus* shallow embedding. Shallow embedding of Hoare notation in higher order logic. Programming logic as a special case of general logic. [2 lectures]

Objectives

By the end of the course students should have an understanding of some aspects of the following topics. Partial and total correctness. Hoare notation. Axioms and rules of Floyd–Hoare logic. Soundness and completeness. Mechanised program verification using verification conditions. Program refinement. Semantic embedding in higher order logic. Limitations of program verification.

Recommended reading

Comprehensive notes supplied.

Specification and Verification II

Lecturer: Professor M.J.C. Gordon

No. of lectures: 12

Prerequisite course: Specification and Verification I

Aims

The aim of the course is to introduce modern work on formal hardware verification including manual deductive methods and automatic techniques (including model checking). The similarities and differences with software verification will be discussed.

Lectures

- **Introduction to formal methods for hardware.** Use of HDLs (e.g. Verilog) *versus* mathematical logic. Specifying structure and behaviour. Parameterised designs. [4 lectures]
- **Logical models of transistors.** Simple switch models. Threshold switching models. Unidirectional sequential models. Problems with existing methods. [2 lectures]
- **Sequential behaviour.** Modelling on different timescales. Temporal abstraction. [2 lectures]
- **Property specification and checking.** Introduction to temporal logic and model checking algorithms. [4 lectures]

Objectives

Students successfully completing the course will have been introduced to the following topics. The use and limitations of Floyd–Hoare logic for verifying HDL programs. Event and trace semantics of a simple Verilog-like HDL. Formal verification of adder and multiplier examples. The description of hardware directly in higher order logic. Modelling combinational and sequential behaviour. Representing behaviour with predicates. Formal treatment of simple CMOS examples and edge-triggered D-type registers. The strengths and weaknesses of various transistor models. Temporal abstraction. Transition systems. Expressing properties in temporal logic (including CTL, LTL and PSL). Symbolic modelling with BDDs. Model checking.

Recommended reading

Comprehensive notes supplied.

System-on-Chip Design

Lecturer: Dr D.J. Greaves

No. of lectures: 12

Prerequisite courses: Specification and Verification II, Computer Design, ECAD, C and C++

Aims

A current-day system on a chip (SoC) consists of several different microprocessor 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 protocol stacks 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.** Basic RTL to gates synthesis algorithm.
- **Further examples.** Event-driven simulation cycle. Using signals, variables and transactions for component inter-communication.
- **SystemC overview.** Verilog synthesis and high/low-level mapping examples.
- **High-level modelling in SystemC.** Bus and cache structures, DRAM interface. Design exploration.
- **Transactional modelling (ESL).** Electronic systems level design. IP-XACT.
- **Contemporary ASIC design flow.** EDA tools used (timing and power modelling, memory generators, power gating, clock tree, self-test and scan insertion).
- **Design flow continued.** Instruction set simulators, cache modelling and hybrid models.
- **Assertions and monitors.** System Verilog brief tour. PSL/SVA assertions. Temporal logic compilation to FSM. Assertion based design. [2 lectures]

- **Future languages.** Recent developments, lectured as time permits (BlueSpec, H2, Kiwi).
- **On Chip interconnect.** Busses (OPB (BVCI) and AXI). Glue logic synthesis. Transactor synthesis. Network on chip.
- **Engineering aspects.** Scaling, power, logical effort and performance limits.

In addition to these topics, the running example will demonstrate practical aspects of device bus interface design, on chip communication and device control software.

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
- be familiar with SystemC 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/`

Easter Term 2009: Part II lectures

Additional Topics

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.

Lectures

The exact plan of lectures has not been settled at the time of publishing this document, but may include some of the following.

- Location, Technology and Context Aware Systems [Professor A. Hopper]
- Cryptography, Security, and Ubiquitous Computing [Dr F.M. Stajano]
- The Navstar Global Positioning System (GPS) [Dr A. Jones]
- Spatial Computing and Cars [Dr J.K. Fawcett]
- Advanced Protocols [Dr G. Mapp]
- Thin Client Systems [Dr A. Harter]
- Broadband Fixed Wireless Access [Dr I.J. Wassell]

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

Recommended reading

Stajano, F.M. (2002). *Security for ubiquitous computing*. Wiley. ISBN 0-470-84493-0
Campbell, A. & Nahrstedt, K. (1997). *Building QoS into distributed systems*. Chapman and Hall. ISBN 0-412-80940-0
Harter, A., Hopper, A., Steggles, P., Ward, A. & Webster, P. (1999). The anatomy of a context-aware application. *Proceedings of the 5th Annual ACM/IEEE International conference on Mobile computing and Networking (Mobicom '99), Seattle, Washington, USA, August 15–20 1999*. Available for download from
`http://www.cl.cam.ac.uk/research/dtg/publications/public/ah12/acaa.pdf`

Hopper, A. (1999). Sentient computing. *The Royal Society Clifford Paterson Lecture 1999*. Available for download from
http://www.cl.cam.ac.uk/research/dtg/publications/public/files/tr.1999.12.pdf
Yang, S.J., Nieh, J., Selsky, M. & Tiwari, N. (2002). The performance of remote display mechanisms for thin-client computing. In *Proceedings of the 2002 USENIX Annual Technical Conference, Monterey, CA, June 10–15, 2002*, pp. 131–146.
Global Positioning System Overview:
http://www.colorado.Edu/geography/gcraft/notes/gps/gps_f.html
Brown, C.M. & Terzopoulos, D. (ed.) (1995). *Real-time computer vision*. Cambridge University Press.
Webb, W. *Introduction to wireless local loop. Second Edition: Broadband and Narrowband Systems*.
Broadband Wireless Association: http://www.broadband-wireless.org/

Artificial Intelligence II

Lecturer: Dr S.B. Holden

No. of lectures: 12

Prerequisite courses: Artificial Intelligence I, Logic and Proof, Algorithms, Mathematical Methods for Computer Science, Discrete Mathematics, 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 knowledge representation and 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 richer forms of knowledge representation, 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 symbolic knowledge representation.** Representing knowledge using First Order Logic (FOL). The situation calculus. [1 lecture]

- **Further planning.** Incorporating heuristics into partial-order planning. Planning graphs. The GRAPHPLAN algorithm. Planning using propositional logic. [1 lecture]

- **Uncertainty and Bayesian networks.** Review of probability as applied to AI. Bayesian networks. Inference in Bayesian networks using both exact and approximate techniques. Other ways of dealing with uncertainty. [3 lectures]
- **Utility and decision-making.** Maximizing expected utility, decision networks, the value of information. [1 lecture]
- **Uncertain reasoning over time.** Markov processes, transition and sensor models. Inference in temporal models: filtering, prediction, smoothing and finding the most likely explanation. Hidden Markov models. [1 lecture]
- **Further supervised learning.** Bayes theorem as applied to supervised learning. The maximum likelihood and maximum *a posteriori* hypotheses. Applying the Bayesian approach to neural networks. [4 lectures]
- **Reinforcement learning.** Learning from rewards and punishments. [1 lecture]

Objectives

At the end of this course students should

- have gained a deeper appreciation for 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 (2nd ed.).
Bishop, S. (1995). *Neural networks for pattern recognition*. Oxford University Press.

Business Studies Seminars

Lecturer: Mr J.A. Lang and others

No. of seminars: 6

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

Six lectures by six 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.

Distributed Systems

Lecturer: Dr D. Evans

No. of lectures: 8

Prerequisite courses: Concurrent Systems and Applications, Operating Systems, Digital Communication I, Introduction to Security

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; and the resulting details of good distributed algorithms and applications.

Lectures

- **Introduction.** Characteristics specific to distributed systems. Software structure. Modelling, designing, and engineering distributed software. Issues of scale and trust. Why seemingly easy problems suddenly become hard.
- **Time.** Physical and logical time. Event ordering. Clock synchronisation. Message delivery ordering.
- **Communication.** Overview of synchronous, asynchronous, and event-based middleware. Web services.
- **Naming.** Design of names, pure or hierarchical. Interpretation of names in context. Binding. Long-term consistency.
- **Algorithms and application protocols.** Replication management. Strong and weak consistency. Asynchronous and synchronous algorithms. Atomic commitment. Process groups. Election. Mutual exclusion.

- **Access control.** ACLs and capabilities in distributed systems. Role-based access control. Policy expression and enforcement.
- **Storage.** Design issues for network-based storage services.
- **Applications.** Pervasive computing environments: active office, home, and city. Events, composite events, mobility and, location-tracking. Electronic health, police, and transport services.

Objectives

At the end of the course students should

- understand the fundamental properties of distributed systems and how to design software to accommodate them
- understand some basic distributed algorithms and the assumptions on which they are based
- for widely distributed and/or large scale systems, understand how naming and access control can be designed
- understand the tradeoffs involved in selecting various styles of middleware
- and be aware of some emerging application areas, such as pervasive and mobile computing

Recommended reading

* Bacon, J. & Harris, T. (2003). *Operating systems: distributed and concurrent software design*. 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.).
Mullender, S. (ed.) (1993). *Distributed systems*. Addison-Wesley (2nd ed.).

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.

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.