Computer Science Tripos
Syllabus and Booklist 2015–16

Contents

Introduction to Part IA

Entry to the Computer Science Tripos

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

Computer Science Tripos Part IA

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

All other Part IA students are required to attend, besides the lectures listed in this document, the Mathematics course offered for Part IA of the Natural Sciences Tripos (NST), together with **either** Paper 1 of Part IA of the Psychological and Behavioural Sciences Tripos (PBST) **or** *one* other Natural Science subject selected from the following list: Chemistry, Evolution and Behaviour, Earth Sciences, Physics, and Physiology of Organisms.

Physics is recommended for those with an A-level in the subject; potential applicants may note that there is no A-level prerequisite for Evolution and Behaviour, Earth Sciences or Physiology of Organisms, although an AS-level science would be desirable. Laboratory work forms an integral part of the Natural Sciences Part IA course, and students reading the Computer Science Tripos will be required to undertake practical work on the same basis as for the Natural Sciences Tripos. There is no A-level requirement for those taking Paper 1 of the PBS 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.

Psychological and Behavioural Sciences students

There is an "Introduction to Computer Science" option in Part I of the Psychological and Behavioural Sciences 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/timetables/`

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:

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

telephone: 01223 763505
fax: 01223 334678
e-mail: `teaching-admin@cl.cam.ac.uk`

Michaelmas Term 2015: Part IA lectures

Paper 1: Foundations of Computer Science

Lecturer: Professor L.C. Paulson

No. of lectures and practicals: 12 + 5 (NST and PBST students will take 4 practicals)

Suggested hours of supervisions: 3 to 4

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 for 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 introduce traditional (procedural) programming, such as assignments, arrays and references.

Lectures

- **Introduction to Programming.** The role of abstraction and representation. Introduction to integer and floating-point arithmetic. Declaring functions. Decisions and booleans. Example: integer exponentiation.
- **Recursion and Efficiency.** Examples: Exponentiation and summing integers. Overloading. Iteration *versus* recursion. Examples of growth rates. Dominance and O-Notation. The costs of some representative functions. Cost estimation.
- **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.

- **Functions as values.** Nameless functions. Currying. The "apply to all" functional, `map`. *Examples*: matrix transpose and product. The predicate functionals `filter` and `exists`.

- **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.

- **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.

- **Polynomial arithmetic.** Addition, multiplication of polynomials using ideas from sorting, etc.

- **Elements of procedural programming.** Address *versus* contents. Assignment *versus* binding. Own variables. Arrays, mutable or not. Introduction to linked lists.

Objectives

At the end of the course, students should

- be able to write simple ML programs;

- understand the fundamentals of using a data structure to represent some mathematical abstraction;

- 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;

- understand how ML combines imperative and functional programming in a single language.

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.

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

Paper 1: Object-Oriented Programming

Lecturer: Dr R.K. Harle, Dr A.C. Rice and Dr S. Cummins

No. of lectures and practicals: 12 + 5

Suggested hours of supervisions: 3 to 4

Aims

The goal of this course is to provide students with the ability to write programs in Java and make use of the concepts of Object-Oriented Programming. Examples and discussions will use Java primarily, but other languages may be used to illustrate specific points where appropriate. The course is designed to accommodate students with diverse programming backgrounds; it is taught with a mixture of lectures and practical sessions where students can work at their own pace from a course handbook. Each practical class will culminate in an assessed exercise.

Lecture syllabus

- **Types, Objects and Classes** Moving from functional to imperative. Distinguishing state and behaviour. Primitive types. Function prototypes. Objects and classes as custom types. Introduction to parameterised types (templates/Generics).

- **Pointers, References and Memory** Pointers and references. The call stack and heap. Iteration and recursion. Pass-by-value and pass-by-reference. Objects as reference types in Java.

- **Creating Classes** Modularity. Encapsulation. Information hiding. Access modifiers. Advantages of immutability. Creating Generic types in Java. Static data.

- **Inheritance** Inheritance. Casting. Shadowing. Overloading. Overriding. Abstract Methods and Classes.

- **Polymorphism and Multiple Inheritance** Polymorphism in ML and Java. Multiple inheritance. Interfaces in Java.

- **Lifecycle of an Object** Constructors and chaining. Destructors. Finalizers. Garbage Collection. Copying Objects. Shallow and deep copies. Copy constructors. Cloning in Java. Cloneable as a marker interface in Java.
- **Java Collections** Java Collection interface. Key classes. Collections class. Iteration options and the use of Iterator.
- **Object Comparison** Comparing primitive and reference types. Equals. Comparable and Comparator in Java. Operator Overloading.
- **Error Handling** Limitations of return values. Exceptions. Custom exceptions.
- **Design Patterns** Introduction to design patterns. Examples of Singleton, Decorator, State, Strategy, Observer.
- **Case Studies and Worked Examples**

Practical classes

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

Objectives

At the end of the course students should

- be familiar with the main features and limitations of the Java language;
- be able to write a Java program to solve a well specified problem;

- understand the principles of OOP;
- be able to demonstrate good object-oriented programming skills in Java;
- be able to describe, recognise, apply and implement selected design patterns in Java;
- be familiar with common errors in Java and its associated libraries;
- understand a Java program written by someone else;
- be able to debug and test Java programs;
- be familiar with major parts of Java 8 SE libraries;
- understand how to read Javadoc library documentation and reuse library code.

Recommended reading

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

* Deitel, H.M. & Deitel, P.J. (2009). *Java: How to Program*. Prentice Hall (8th ed.).
Flanagan, D. (2005). *Java in a nutshell : a desktop quick reference*. O'Reilly (5th ed.).
Flanagan, D. (2004). *Java examples in a nutshell : a tutorial companion to Java in a nutshell*. O'Reilly (3rd ed.).
Gamma, E., Helm, R., Johnson, R. & Vlissides, A. (1995). *Design patterns: elements of reusable object-oriented software*. Addison-Wesley.
Bloch, J. & Gafter, N. (2005). *Java puzzlers*. Addison-Wesley.

Paper 2: Digital Electronics

This course is not taken by NST or PBST students.

Lecturer: Dr I.J. Wassell

No. of lectures and practical classes: 12 + 7

Suggested hours of supervisions: 3-4

This course is a prerequisite for Operating Systems and Computer Design (Part IB), ECAD and Architecture Practical Classes (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 n and p channel MOSFETs for building logic 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.
- **Binary adders.** Half adder, full adder, ripple carry adder, fast carry generation.
- **Combinational logic design: further considerations.** Multilevel logic. Gate propagation delay. An introduction to timing diagrams. Hazards and hazard elimination. Other ways to implement combinational logic.
- **Introduction to practical classes.** Prototyping box. Breadboard and Dual in line (DIL) packages. Wiring. Use of oscilloscope.
- **Sequential logic.** Memory elements. RS latch. Transparent D latch. Master–slave D flip-flop. T and JK flip-flops. Setup and hold times.
- **Sequential logic.** Counters: Ripple and synchronous. Shift registers.
- **Synchronous State Machines.** Moore and Mealy finite state machines (FSMs). Reset and self starting. State transition diagrams.
- **Further state machines.** State assignment: sequential, sliding, shift register, one hot. Implementation of FSMs.
- **Electronics, Devices and Circuits.** Current and voltage, resistance, basic circuit theory, the potential divider. Solving non-linear circuits. Resistor-Capacitor (RC) circuits. Materials, semiconductors and the p-n junction, i.e., the diode. n and p channel MOSFETs and n-MOSFET logic, e.g., n-MOSFET inverter. CMOS logic. Logic families. Noise margin. [3 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 MOSFETs to build digital logic circuits.
- understand the effect of finite load capacitance on the performance of digital logic circuits.

Recommended reading

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

Books for reference:

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

Paper 2: Discrete Mathematics

Lecturers: Professor M.P. Fiore and Professor I.M. Leslie

No. of lectures: 24 (continued into Lent term)

Suggested hours of supervisions: 6 to 8

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

Aims

The course aims to introduce the mathematics of discrete structures, showing it as an essential tool for computer science that can be clever and beautiful.

Lectures

- **Proof [5 lectures].**

  Proofs in practice and mathematical jargon. Mathematical statements: implication, bi-implication, universal quantification, conjunction, existential quantification,

disjunction, negation. Logical deduction: proof strategies and patterns, scratch work, logical equivalences. Proof by contradiction. Divisibility and congruences. Fermat's Little Theorem.

- **Numbers [5 lectures].**

  Number systems: natural numbers, integers, rationals, modular integers. The Division Theorem and Algorithm. Modular arithmetic. Sets: membership and comprehension. The greatest common divisor, and Euclid's Algorithm and Theorem. The Extended Euclid's Algorithm and multiplicative inverses in modular arithmetic. The Diffie-Hellman cryptographic method. Mathematical induction: Binomial Theorem, Pascal's Triangle, Fundamental Theorem of Arithmetic, Euclid's infinity of primes.

- **Sets [7 lectures].**

  Extensionality Axiom: subsets and supersets. Separation Principle: Russell's Paradox, the empty set. Powerset Axiom: the powerset Boolean algebra, Venn and Hasse diagrams. Pairing Axiom: singletons, ordered pairs, products. Union axiom: big unions, big intersections, disjoint unions. Relations: composition, matrices, directed graphs, preorders and partial orders. Partial and (total) functions. Bijections: sections and retractions. Equivalence relations and set partitions. Calculus of bijections: characteristic (or indicator) functions. Finite cardinality and counting. Infinity axiom. Surjections. Enumerable and countable sets. Axiom of choice. Injections. Images: direct and inverse images. Replacement Axiom: set-indexed constructions. Set cardinality: Cantor-Schoeder-Bernstein Theorem, unbounded cardinality, diagonalisation, fixed-points. Foundation Axiom.

- **Formal languages and automata [7 lectures].**

  Introduction to inductive definitions using rules and proof by rule induction. Abstract syntax trees.

  Regular expressions and their algebra.

  Finite automata and regular languages: Kleene's theorem and the Pumping Lemma.

Objectives

On completing the course, students should be able to

- prove and disprove mathematical statements using a variety of techniques;
- apply the mathematical principle of induction;
- know the basics of modular arithmetic and appreciate its role in cryptography;
- understand and use the language of set theory in applications to computer science;
- define sets inductively using rules and prove properties about them;
- convert between regular expressions and finite automata;
- use the Pumping Lemma to prove that a language is not regular.

Recommended reading

Biggs, N.L. (2002). *Discrete mathematics.* Oxford University Press (Second Edition).
Davenport, H. (2008). *The higher arithmetic: an introduction to the theory of numbers.* Cambridge University Press.
Houston, K. (2009). *How to think like a mathematician: a companion to undergraduate mathematics.* Cambridge University Press.
Kozen, D.C. (1997). *Automata and computability*. Springer.
Lehman, E.; Leighton, F.T.; Meyer, A.R. (2014). *Mathematics for computer science.* Available on-line.
Velleman, D.J. (2006). *How to prove it: a structured approach.* Cambridge University Press (Second Edition).

Lent Term 2016: Part IA lectures

Paper 1: Algorithms

Lecturer: Dr F.M. Stajano and Dr T.M. Sauerwald

No. of lectures and practical classes: 24 + 3 (NST and PBST students take 1 practical)

Suggested hours of supervisions: 6 to 8

Prerequisite courses: Foundations of Computer Science, Object-Oriented Programming

This course is a prerequisite for: Artificial Intelligence, Complexity Theory, Computer Graphics and Image Processing , Prolog, Advanced Algorithms

Aims

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

Lectures

- **Sorting.** Review of complexity and O-notation. Trivial sorting algorithms of quadratic complexity. Review of merge sort and quicksort, understanding their memory behaviour on statically allocated arrays. Heapsort. Stability. Other sorting methods including sorting in linear time. Median and order statistics. [Ref: CLRS3 chapters 1, 2, 3, 6, 7, 8, 9] [about 4 lectures]
- **Strategies for algorithm design.** Dynamic programming, divide and conquer, greedy algorithms and other useful paradigms. [Ref: CLRS3 chapters 4, 15, 16] [about 3 lectures]
- **Data structures.** Primitive data structures. Abstract data types. Pointers, stacks, queues, lists, trees. Binary search trees. Red-black trees. B-trees. Hash tables. Priority queues and heaps. [Ref: CLRS3 chapters 6, 10, 11, 12, 13, 18] [about 5 lectures]
- **Advanced data structures.** Amortized analysis: aggregate analysis, potential method. Fibonacci heaps. Disjoint sets. [Ref: CLRS3 chapters 17, 19, 20, 21] [about 4 lectures]
- **Graph algorithms.** Graph representations. Breadth-first and depth-first search. Topological sort. Minimum spanning tree. Kruskal and Prim algorithms. Single-source shortest paths: Bellman-Ford and Dijkstra algorithms. All-pairs shortest paths: matrix multiplication and Johnson's algorithms. Maximum flow: Ford-Fulkerson method, Max-Flow Min-Cut Theorem. Matchings in bipartite graphs. [Ref: CLRS3 chapters 22, 23, 24, 25, 26] [about 7 lectures]
- **Geometric algorithms.** Intersection of segments. Convex hull: Graham's scan, Jarvis's march. [Ref: CLRS3 chapter 33] [about 1 lecture]

Objectives

At the end of the course students should

- have a thorough understanding of several classical algorithms and data structures;
- be able to analyse the space and time efficiency of most algorithms;
- 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 design new algorithms or modify existing ones for new applications and reason about the efficiency of the result.

Recommended reading

* Cormen, T.H., Leiserson, C.D., Rivest, R.L. & Stein, C. (2009). *Introduction to Algorithms*. MIT Press (3rd ed.). ISBN 978-0-262-53305-8
Sedgewick, R., Wayne, K. (2011). *Algorithms* Addison-Wesley. ISBN 978-0-321-57351-3.
Kleinberg, J. & Tardos, É. (2006). *Algorithm design*. Addison-Wesley. ISBN 978-0-321-29535-4.
Knuth, D.A. (2011). *The Art of Computer Programming*. Addison-Wesley. ISBN 978-0-321-75104-1.

Students hoping to receive a computer science degree from Cambridge are expected to buy, make extensive use of, and keep as reference for their future career, one of the above fundamental textbooks: those not doing so will be severely disadvantaged. The recommended choice is Cormen, Leiserson, Rivest and Stein (CLRS3, starred in the above list) which covers all topics listed and, in spite of its superb quality, is the cheapest: about 35 GBP new for over 1300 pages. The references in the syllabus are to this textbook. The other textbooks listed are excellent additions for further study but might cost more and yet not cover the entire syllabus.

Paper 2: Operating Systems

This course is not taken by NST or PBST students.

Lecturer: Dr. R. Mortier

No. of lectures: 12

Suggested hours of supervisions: 3 to 4

Prerequisite courses: Computer Fundamentals, Digital Electronics

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

Aims

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

Lectures

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

Objectives

At the end of the course students should be able to

- describe the general structure and purpose of an operating system;
- explain the concepts of process, address space, and file;
- compare and contrast various CPU scheduling algorithms;

- understand the differences between segmented and paged memories, and be able to describe the advantages and disadvantages of each;
- compare and contrast polled, interrupt-driven and DMA-based access to I/O devices.

Recommended reading

* Bacon, J. & Harris, T. (2003). *Operating systems*. Addison-Wesley (3rd ed.).
Silberschatz, A., Peterson, J.L. & Galvin, P.C. (2008). *Operating systems concepts*. Wiley (8th ed.).
Leffler, S. (1989). *The design and implementation of the 4.3BSD Unix operating system*. Addison-Wesley.
McKusick, M.K., Neville-Neil, G.N. & Watson, R.N.M. (2014) *The Design and Implementation of the FreeBSD Operating System*. Pearson Education. (2nd end.).
Solomon, D. & Russinovich, M. (2000). *Inside Windows 2000*. Microsoft Press (3rd ed.).

Easter Term 2016: Part IA lectures

Paper 1: Numerical Methods

Lecturer: Dr D.J. Greaves

No. of lectures: 11

Suggested hours of supervisions: 3 to 4

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

Aims

This course provides:

1. an introduction to (IEEE) floating-point data representation and arithmetic;
2. illustrations of how naïve implementations of obvious mathematics can go badly wrong;
3. a study of several standard numerical algorithms.

An overall implicit aim is to encourage caution when using any floating-point value produced by a computer program. A variety of code fragments are provided and most are available in multiple languages. Students are strongly encouraged to experiment with these fragments.

Lectures

- **Integer and floating-point representation and arithmetic.** Signed and unsigned integers and fixed-point; arithmetic, saturating arithmetic. Long division and multiplication. Floating point I/O in ASCII. What numbers are exactly representable in bases 2 and 10. Accuracy in terms of significant figures.
- **IEEE floating-point arithmetic.** Floating-point arithmetic, and the IEEE requirements. IEEE 754/854 floating point (32 and 64 bit); zeros, infinities, NaN. Overflow, underflow, progressive loss of significance. Rounding modes. Floating-point arithmetic is non-associative, and mathematical equivalences fail. Nonsensical results, e.g. `sin(1e40)`. Difficulty in obtaining IEEE-quality in libraries.
- **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. Order of convergence. Limit cycles. Why summing a Taylor series is problematic. Condition number, partial derivatives, backwards stability and chaos.

- **Matrix Form Simultaneous Equations.** Gaussian Elimination. Stability and pivoting improvements. Positive-definite. L/U and Cholesky decompositions. Doolittle/Crout method.

- **Efficient Implementation** Chebychev orthogonal basis (for power series economisation) Practical implementation of scientific (trig/log) functions. Comparison of Taylor, Chebychev and Cordic.

- **Finite-Difference Time-Domain Simulation.** Numerical simulation of SHM, charge/discharge, waves and other various examples (such as a Moniac Simulator).

- **Circuit/Flow Analysis.** Using a matrix representation of a linear flow circuit (water, electricity etc) to find steady state. Extensions for non-linear and time-varying branches as used by SPICE.

- **Adaptive Methods and Custom Encodings** A subset of the following topics will we be lectured. Details in handout. Arbitrary precision floating point, adaptive floating point, interval arithmetic. Rounding errors in PCM. Logarithmic and other non-linear representations. Their use in a-posteriori decision algorithms. Eg for rapid multiplication in Viterbi/Bayes and specialist ALUs (e.g. for low-density parity). Simulated Annealing. Non-linear spatial quantisation.

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 and know when incorrect solution is likely;

- be familiar with several key algorithms from the history of numerical analysis;

- decide how and when computation energy should be traded for accuracy;

- 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: Software and Interface Design

This course is not taken by NST or PBST students.

Lecturer: Professor A.F. Blackwell

No. of lectures: 11

Suggested hours of supervisions: 3 to 4

Companion courses: Object-Oriented Programming, Programming in Java

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

Aims

This course introduces principles and methods for the design of software systems in professional contexts. The whole of the software development lifecycle is considered, but with special emphasis on techniques for requirements capture and testing of interactive systems.

Lectures

- **Introduction.** Design process overview: Inception phase, Elaboration phase, Construction phase, Transition phase (1 lecture).
- **Inception.** Structured description of system usage and function (2 lectures).
- **Elaboration.** Development and evaluation of interactive prototypes (2 lectures).
- **Construction.** Use of source code as a design model (1 lecture).
- **Transition.** Testing and debugging techniques (2 lectures).
- **Evaluation.** Measurement with respect to design objectives (2 lectures).
- **Iteration.** Design process responses to uncertainty and requirements change (1 lecture).

Objectives

At the end of the course, students should be able to undertake design of an interactive system in a methodical manner, starting from a general requirement, analysing user needs, developing a design model, approaching iterative model refinement and implementation in a manner that minimises risk, and using appropriate methods to identify and prevent faults.

Recommended reading

Pressman, R.S. (2010). *Software engineering*. McGraw-Hill (7th international ed.). ISBN 9780073375977
Sharp, H., Rogers, Y. & Preece, J. (2007). *Interaction design: beyond human–computer interaction*. Wiley (2nd ed.).
Further reading

McConnell, S. (2004). *Code complete: a practical handbook of software construction*. Microsoft Press (2nd ed.).
Broy, M. & Denert, E. (ed.) (2002). *Software pioneers: contributions to software engineering*. Springer-Verlag.
Schon, D.A. (1990). *Educating the reflective practitioner*. Jossey-Bass.

Further Java Briefing

Lecturer: Dr A.C. Rice

No. of lectures: 1

Prerequisite course: Object-Oriented Programming

This course is a prerequisite for Further Java.

Aims

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

Lecture

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

Objectives

On completing the exercise students should

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

Preparing to Study Computer Science

For general advice about preparing for the Computer Science course at Cambridge and for details of the pre-arrival course, please see: `http://www.cl.cam.ac.uk/freshers/`

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 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/timetables/`

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:

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

telephone: 01223 763505
fax: 01223 334678
e-mail: `teaching-admin@cl.cam.ac.uk`

Michaelmas Term 2015: Part IB lectures

Computer Design

Lecturers: Professor S.W. Moore and Dr T. Jones

No. of lectures: 18 (plus 4 via a web-based tutor)

Suggested hours of supervisions: 5

Prerequisite course: Digital Electronics

Companion course: Electronic Computer Aided Design (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 a hardware description language (SystemVerilog) and computer architecture concepts in order to design computer systems. The parallel ECAD+Arch practical classes will allow students to apply the concepts taught in lectures.

The course starts with a web-based SystemVerilog tutor which is a prerequisite for the ECAD+Arch practical classes. There are then eighteen lectures in three six-lecture parts. Part 1 goes from gates to a simple processor. Part 2 looks at instruction set and computer architecture. Part 3 analyses the architecture of modern systems-on-chip.

Lectures

Part 0 - SystemVerilog Web tutor

- This web tutor is a prerequisite to starting the ECAD+Arch laboratory sessions [equivalent to approximately 4 lectures]

Part 1 - Gates to processors [lecturer: Simon Moore]

- **Introduction and motivation.** [1 lecture] Current technology, technology trends, ECAD trends, challenges.
- **Logic modelling, simulation and synthesis.** [1 lecture] Logic value and delay modelling. Discrete event and device simulation. Automatic logic minimization.
- **SystemVerilog FPGA design.** [1 lecture] Practicalities of mapping SystemVerilog descriptions of hardware (including a processor) onto an FPGA board. Tips and pitfalls when generating larger modular designs.
- **Chip, board and system testing.** [1 lecture] Production testing, fault models, testability, fault coverage, scan path testing, simulation models.

- **Building a simple computer.** [2 lectures]

Part 2 - Instruction sets and introduction to computer architecture [lecturer: Simon Moore]

- **Historical perspective on computer architecture.** [1 lecture] EDSAC *versus* Manchester Mark I.
- **RISC machines.** [1 lecture] Introduction to ARM and MIPS RISC processor designs.
- **CISC and virtual machines** [1 lecture] The Intel x86 instruction set and the Java Virtual Machine (JVM).
- **Memory hierarchy.** [1 lecture] Caching, etc.
- **Hardware support for operating systems.** [1 lecture] Memory protection, exceptions, interrupts, etc.
- **Pipelining and data paths.** [1 lecture]

Part 3 - Systems-on-chip [lecturer: Timothy Jones]

- **Overview of Systems-on-Chip (SoCs).** [1 lecture] What are they and how do we program them?
- **Multicore Processors.** [2 lectures] Communication, cache coherence, barriers and synchronisation primitives
- **Graphics processing units (GPUs)** [2 lectures] Basic GPU architecture and programming
- **Future Directions** [1 lecture] Where is computer architecture heading?

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 and coherency needed for multicore systems;
- understand how to implement a processor in SystemVerilog;
- appreciate the use of pipelining in processor design;
- have an appreciation of control structures used in processor design;
- have an appreciation of system-on-chip design.

Recommended reading

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

Recommended further reading:

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

Computer Graphics and Image Processing

Lecturer: Professor P. Robinson

No. of lectures: 16

Suggested hours of supervisions: 4

Prerequisite courses: Algorithms

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? Colour. Human vision. Resolution and quantisation. Storage of images in memory. [2 lectures]
- **Rendering.** Perspective. Reflection of light from surfaces. Geometric models. Ray tracing. [2 lectures]
- **Graphics pipeline.** Polygonal mesh models. Transformations using matrices in 2D and 3D. Homogeneous coordinates. Projection: orthographic and perspective. Graphics hardware and OpenGL. Lighting: flat shading, Gouraud shading, Phong shading. Texture mapping. [4 lectures]
- **Underlying algorithms.** Drawing a straight line. Drawing circles and ellipses. Cubic curves: specification and drawing. Clipping lines. Clipping polygons. Filling

polygons. 3D scan conversion using the z-buffer. Anti-aliasing and the A-buffer. [4 lectures]

- **Technology.** Colour spaces. Output devices: brief overview of two display technologies (LCD, DMD) and two printer technologies (ink jet and laser printer). Image capture. [2 lectures]
- **Image processing.** Operations on images: filtering, point processing, compositing. Half-toning and dithering, error diffusion. [2 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 two display technologies and two printer technologies;
- describe and explain the following algorithms: mid-point line drawing, mid-point circle drawing, Bezier cubic drawing, Cohen-Sutherland line clipping, scanline polygon fill, Sutherland-Hodgman polygon clipping, *z*-buffer, *A*-buffer, texture mapping, 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 *z*-buffer polygon scan conversion; 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.

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. (2008). *Digital image processing*. Addison-Wesley (3rd ed). [The second edition (1992) and the first edition (Gonzalez & Wintz, 1977) are as useful for this course.]
* Shirley, P. & Marschner, S. (2009). *Fundamentals of Computer Graphics*. CRC Press (3rd ed.).
Slater, M., Steed, A. & Chrysanthou, Y. (2002). *Computer graphics and virtual environments: from realism to real-time*. Addison-Wesley.

Computer Networking

Lecturer: Dr A.W. Moore

No. of lectures: 24 (Continued in Lent Term)

Suggested hours of supervisions: 6

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

Aims

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

Lectures

- **Introduction.** Overview of networking using the Internet as an example. LANs and WANs. OSI reference model, Internet TCP/IP Protocol Stack. Client/server paradigm, circuit-switching, packet-switching, Internet structure, networking delays and packet loss. [3 lectures]
- **Link layer and local area networks.** Link layer services, error detection and correction, Multiple Access Protocols, link layer addressing, Ethernet, hubs and switches, Point-to-Point Protocol. [3 lectures]
- **Wireless and mobile networks.** Wireless links and network characteristics, Wi-Fi: IEEE 802.11 wireless LANs, mobility management and mobile IP. [2 lectures]
- **Network layer addressing.** Network layer services, IP, IP addressing, IPv4, DHCP, NAT, ICMP, IPv6. [3 lectures]
- **Network layer routing.** Routing and forwarding, routing algorithms, routing in the Internet, RIP, OSPF, BGP, multicast. [3 lectures]
- **Transport layer.** Service models, multiplexing/demultiplexing, connection-less transport (UDP), principles of reliable data transfer, connection-oriented transport (TCP), TCP congestion control, securing TCP (SSL), TCP variants. [3 lectures]
- **Application layer.** Service requirements, WWW, HTTP, electronic mail, Domain Name System, P2P, socket programming API. [3 lectures]

- **Multimedia networking.** Networked multimedia applications, best-effort service and multimedia delivery requirements, multimedia protocols (RTSP, RTP, RTCP, SIP), content distribution networks. [2 lectures]

- **Datacenter Networking** Datacenter introductions, architecting a datacenter, datacenter network and workload issues, datacenter transport issues. [2 lectures]

Objectives

At the end of the course students should

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

Recommended reading

* Peterson, L.L. & Davie, B.S. (2011). *Computer networks: a systems approach*. Morgan Kaufmann (5th ed.). ISBN 9780123850591
Kurose, J.F. & Ross, K.W. (2009). *Computer networking: a top-down approach*. Addison-Wesley (5th ed.).
Comer, D. & Stevens, D. (2005). *Internetworking with TCP-IP, vol.* 1 *and* 2. Prentice Hall (5th ed.).
Stevens, W.R., Fenner, B. & Rudoff, A.M. (2003). *UNIX network programming, Vol.I: The sockets networking API*. Prentice Hall (3rd ed.).

Concurrent and Distributed Systems

Lecturer: Dr R.M. Watson

No. of lectures: 16 (Continued in Lent Term)

Suggested hours of supervisions: 4

Prerequisite courses: Operating Systems, Object-Oriented Programming

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

Aims of the Michaelmas Term part of the course

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

Michaelmas Term Lectures (Concurrency)

- **Introduction to concurrency, threads, and mutual exclusion** Introduction to concurrent systems; threads; interleaving; preemption; parallelism; execution orderings; processes and threads; kernel vs. user threads; M:N threads; atomicity; mutual exclusion; and mutual exclusion locks (mutexes).
- **More mutual exclusion, semaphores, producer-consumer, and MRSW** Hardware foundations for atomicity; locks and invariants; semaphores; condition synchronisation; N-resource allocation; two-party and generalised producer-consumer; Multi-Reader, Single-Write (MRSW) locks.
- **CCR, monitors, and concurrency in practice** Conditional critical regions (CCR); monitors; condition variables; signal-wait vs. signal-continue semantics; concurrency in practice (kernels, pthreads, Java).
- **Safety and liveness** Safety vs. liveness; deadlock; the Dining Philosophers; resource allocation graphs; deadlock prevention, avoidance, detection, and recovery; livelock; priority inversion; priority inheritance.
- **Concurrency without shared data; transactions** Active objects; message passing; tuple spaces; CSP; and actor models. Composite operations; transactions; ACID; isolation; and serialisability.
- **Further transactions** History graphs; good and bad schedules; isolation vs. strict isolation; 2-phase locking; rollback; timestamp ordering (TSO); and optimistic concurrency control (OCC).
- **Crash recovery, lock-free programming, and transactional memory** Write-ahead logging, checkpoints, and recovery. Lock-free programming and software-transactional memory (STM).
- **Concurrent systems case study.** Concurrency in the FreeBSD kernel; kernel synchronisation before parallelism; Giant-locked kerels; fine-grained locking; primitives and strategies; lock order checking; network-stack work flows; performance scalability; the impact of changing hardware.

Objectives

At the end of the course students should

- understand the need for concurrency control in operating systems and applications, both mutual exclusion and condition synchronisation;

- understand how multi-threading can be supported and the implications of different approaches;
- be familiar with the support offered by various programming languages for concurrency control and be able to judge the scope, performance implications and possible applications of the various approaches;
- be aware that dynamic resource allocation can lead to deadlock;
- understand the concept of transaction; the properties of transactions, how they can be implemented, and how their performance can be optimised based on optimistic assumptions;
- understand how the persistence properties of transactions are addressed through logging; and
- have a high-level understanding of the evolution of software use of concurrency in the operating-system kernel case study.

Recommended reading

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

Aims of the Lent Term part of the course

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

Lent Term Lectures (Distributed Systems)

- **Introduction to distributed systems; RPC** Avantages and challenges of distributed systems; “middleware”; transparency goals; client-server systems; failures and retry semantics (all-or-nothing; at-most-once; at-least-once). Remote procedure call (RPC); marshalling; interface definition languages (IDLs); SunRPC; external data representation (XDR).
- **Network File System and Object-Oriented Middleware** Network File System (NFS); NFSv2; NFSv3; scoping; the implications of a stateless design; performance optimisations. Object-oriented middleware (OOM); Corba ORBs, IDL; DCOM.

- **Practical RPC systems; clocks** Remote method invocation (RMI); remote classes vs. serialisable classes; distributed garbage collection; XML-RPC; SOAP and web services; REST. Physical clocks; UTC; computer clocks; clock synchronisation.
- **Clock synchronisation; logical clocks** Clock drift and compensation; Cristian's Algorithm; Berkeley Algorithm; Network Time Protocol (NTP). Logical time, "happens-before"; Lamport clocks; vector clocks.
- **Consistent cuts, process groups, and mutual exclusion** Consistent global state; consistent cuts. Process groups; FIFO ordering; receiving vs. delivering; causal ordering; total ordering. Distributed mutual exclusion; central lock servers; token passing; totally ordered multicst.
- **Elections, consensus, and distributed transactions** Leader elections; ring-based algorithm; the Bully algorithm. Consensus. Distributed transactions; atomic commit protocols; 2-phase commit. Replication and consistency.
- **Replication in distributed systems, CAP, case studies** Replication and consistency (cont); strong consistency; quorum systems; weak consistency; FIFO consistency; eventual consistency; Amazone's Dynamo; session guarantees; Consistency, Availability and Partitions (CAP); Google datacentre technologies (MapReduce).
- **Further case studies, PubSub, security, NASD/AFS/Coda** Google datacentre technologies (BigTable, Spanner). Access control and the access-control matrix; ACLs vs capabilities; cryptographic capabilities; role-based access control (RBAC); single-system sign-on. NASD, AFS, and Coda.

Objectives

At the end of the course students should

- understand the difference between simple concurrent systems and distributed systems;
- understand the fundamental properties of distributed systems and their implications for system design;
- understand notions of time synchronisation, including logical clocks, vector clocks, and physical time;
- be familiar with various approaches to data and service replication, as well as the concept of data consistency;
- understand the effects of large scale on the provision of fundamental services and the tradeoffs arising from scale;
- appreciate the implications of individual node and network communications failures on distributed computation;

- be aware of a variety of tools used by distributed-system creators, such as RPC and object-oriented middleware (OOM);
- be familiar with a range of distributed algorithms;
- be familiar with a number of case studies in distributed-system design including: the Network File System (NFS), the Network Time Protocol (NTP), Java Remote Method Invocation (RMI), CORBA, the AFS and Coda filesystems, Network-Attached Secure Disks (NASD), and Google's MapReduce, BigTable, and Spanner systems.

Recommended reading

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

ECAD and Architecture Practical Classes

Lecturer: Dr S.W. Moore

No. of practical classes: 8

Prerequisite course: Digital Electronics

Companion course: Computer Design

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 enable students to apply the concepts learned in the Computer Design course. In particular a web based tutor is used to introduce the SystemVerilog hardware description language, while the remaining practical classes will then allow students to implement the design of components in this language.

Practical Classes

- **Web tutor** This first class uses a web based tutor to rapidly teach the SystemVerilog language (this is part of the lectured component of Computer Design).
- **FPGA design flow** Test driven hardware development for FPGA including an embedded processor and peripherals [3 classes]
- **Embedded system implementation** Embedded system implementation on FPGA [3-4 classes]

Objectives

- Gain experience in electronic computer aided design (ECAD) through learning a design-flow for field programmable gate arrays (FPGAs).
- Learn how to interface to peripherals like a touch screen.
- Learn how to debug hardware and software systems in simulation.
- Understand how to construct and program a heterogeneous embedded system.

Recommended reading

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

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

Further Java

Lecturer: Dr A.C. Rice

No. of practical classes: 5 x 2-*hour sessions*

Prerequisite course: Object-Oriented Programming, Further Java Briefing

Companion courses: Concurrent and Distributed Systems

This course is a prerequisite for the Group Project.

Aims

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

Practical classes

- **Communication and client applications.** This class will introduce the Eclipse development environment. Students will write a simple client to send and receive data to a server via TCP.

- **Serialisation, reflection and class loaders.** This class will introduce object serialisation. Students will use a class loader and reflection to inspect an object which is only available at run-time.

- **Concurrency and synchronisation.** This class introduces the concurrency and synchronisation primitives found in Java. Students will implement a thread-safe first-in-first-out queue and learn about Java generics.

- **Server applications**. Students implement a server in Java which is capable of communicating concurrently with mulitple clients.

- **Databases**. This week students will use Java annotations and a relational database to build a persistent store.

Objectives

At the end of the course students should

- understand different mechanisms for communication between distributed applications and be able to evaluate their trade-offs;

- be able to use Java generics and annotations to improve software usability, readability and safety;

- understand and be able to exploit the Java class-loading mechansim;

- understand and be able to use concurrency control correctly;

- understand the concept of transactions and their application in a range of systems.

Recommended reading

* Goetz, B. (2006). *Java concurrency in practice*. Addison-Wesley. Gosling, J., Joy, B., Steele, G., Bracha, G. & Buckley, A. (2014). *The Java language specification, Java SE 8 Edition*. Addison-Wesley.
`http://docs.oracle.com/javase/specs/jls/se8/html/`

Group Project

Lecturer: Professor I.M. Leslie, Professor A.F. Blackwell

No. of lectures: 1

Prerequisite courses: Software Design, Software Engineering, Further Java

Aims

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

Lectures

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

Objectives

At the end of the course students should

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

Mathematical Methods for Computer Science

Lecturers: Professor J.G. Daugman and Dr R.J. Gibbens

No. of lectures: 16

Suggested hours of supervisions: 4

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

Lectures

- **Probability methods** (Dr R.J. Gibbens)
  - **Probability generating functions.** Definitions and properties. Use in calculating moments of random variables and for finding the distribution of sums of independent random variables. [2 lectures]
  - **Inequalities and limit theorems.** Bounds on tail probabilities, moment generating functions, notions of convergence, laws of large numbers, the central limit theorem, statistical applications, Monte Carlo simulation. [3 lectures]
  - **Stochastic processes.** Random walks. Recurrence and transience. The Gambler's Ruin problem. Discrete-time Markov chains, Chapman–Kolmogorov equations, classifications of states, limiting and stationary behaviour, time-reversible Markov chains. Examples and applications. [5 lectures]
- **Fourier and related methods** (Professor J. Daugman)
  - **Fourier representations.** 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, efficient algorithms implementing it, and applications. [2 lectures]
  - **Wavelets.** Introduction to wavelets, with applications in signal processing, coding, communications, and computing. [1 lecture]

Objectives

At the end of the course students should

- understand the use of probability generating functions;
- 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 random walks and discrete-time Markov chains;

- 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, algorithms, and applications;

Reference books

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

Programming in C and C++

Lecturer: Dr A. Madhavapeddy

No. of lectures: 10

Suggested hours of supervisions: 3

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, meta-programming and the STL. Examples. [2 lectures]

- **Linkers and loaders.** Executable sections. Debug symbols. Inspecting program state. [1 lecture]

- **C semantics.** Undefined vs implementation-defined behaviour. Common optimisation problems. Buffer and integer overflows. Examples. [1 lecture]

Objectives

At the end of the course students should

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

Recommended reading

* Eckel, B. (2000). *Thinking in C++, Vol. 1: Introduction to Standard C++*. Prentice Hall (2nd ed.). Also available at
`http://www.mindview.net/Books/TICPP/ThinkingInCPP2e.html`
Kernighan, B.W. & Ritchie, D.M. (1988). *The C programming language*. Prentice Hall (2nd ed.).
Stroustrup, B. (2008). *Programming — principles and practice using C++*. Addison-Wesley.
Stroustrup, B. (1994). *The design and evolution of C++*. Addison-Wesley.
Lippman, S.B. (1996). *Inside the C++ object model*. Addison-Wesley.

Prolog

Lecturer: Dr. A.C. Rice

No. of lectures: 8

Suggested hours of supervisions: 2

Prerequisite courses: Foundations of Computer Science, Algorithms

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 or 4th ed.).
Sterling, L. & Shapiro, E. (1994). *The art of Prolog*. MIT Press (2nd ed.).

Further reading:

O'Keefe, R. (1990). *The craft of Prolog*. MIT Press. [This book is beyond the scope of this course, but it is very instructive. If you understand its contents, you're more than prepared for the examination.]

Semantics of Programming Languages

Lecturer: Professor P. Sewell

No. of lectures: 12

Suggested hours of supervisions: 3

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

Aims

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

Lectures

- **Introduction.** Transition systems. The idea of structural operational semantics. Transition semantics of a simple imperative language. Language design options. [2 lectures]
- **Types.** Introduction to formal type systems. Typing for the simple imperative language. Statements of desirable properties. [2 lectures]
- **Induction.** Review of mathematical induction. Abstract syntax trees and structural induction. Rule-based inductive definitions and proofs. Proofs of type safety properties. [2 lectures]
- **Functions.** Call-by-name and call-by-value function application, semantics and typing. Local recursive definitions. [2 lectures]
- **Data.** Semantics and typing for products, sums, records, references. [1 lecture]
- **Subtyping.** Record subtyping and simple object encoding. [1 lecture]
- **Semantic equivalence.** Semantic equivalence of phrases in a simple imperative language, including the congruence property. Examples of equivalence and non-equivalence. [1 lecture]

- **Concurrency.** Shared variable interleaving. Semantics for simple mutexes; a serializability property. [1 lecture]

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

* Pierce, B.C. (2002). *Types and programming languages*. MIT Press.
Hennessy, M. (1990). *The semantics of programming languages*. Wiley. Out of print, but available on the web at
`http://www.cs.tcd.ie/matthew.hennessy/splexternal2015/resources/sembookWiley.pdf`
Winskel, G. (1993). *The formal semantics of programming languages*. MIT Press.

Software Engineering

Lecturer: Professor R.J. Anderson and Dr R.M. Brady

No. of lectures: 6

Suggested hours of supervisions: 2

This course is a prerequisite for the Group Project.

Aims

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

Lectures

- **The software crisis.** Examples of large-scale project failure, such as the London Ambulance Service system and the NHS National Programme for IT. Intrinsic difficulties with software.

- **The software life cycle.** Getting the requirements right; requirements analysis methods; modular design; the role of prototyping; the waterfall, spiral and evolutionary models.
- **Critical systems.** Examples of catastrophic failure; particular problems with real-time systems; usability and human error; verification and validation.
- **Quality assurance.** The contribution of reviews and testing; reliability growth models; software maintenance and configuration management; life-cycle costs.
- **Tools.** The effect of high-level languages; object-oriented systems and object reuse; an overview of formal methods with some application examples; project planning tools; automated testing tools.
- **Guest lecture.** A guest lecture from an industry speaker about the realities of managing software development in a commercial environment.

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. (2010). *Software engineering*. McGraw-Hill (7th international ed.). ISBN 9780073375977
Leveson, N. (1994). *Safeware*. Addison-Wesley.
Maguire, S. (1993). *Writing solid code*. Microsoft Press.

Further reading:

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

Unix Tools

Lecturer: Dr M.G. Kuhn

No. of lectures: 8

Suggested hours of supervisions: 0–1 (non-examinable course with exercises)

Operating Systems provides a useful foundation for this course.

Aims

This course gives students who have already basic Unix/Linux experience some additional practical software-engineering knowledge: how to use the shell and related tools as an efficient working environment, how to automate routine tasks, and how version control and automated-build tools can help to avoid confusion and accidents, especially when working in teams. These are essential skills, both in industrial software development and student projects.

Lectures

- **Unix concepts.** Brief review of Unix history and design philosophy, documentation, terminals, inter-process communication mechanisms and conventions, shell, command-line arguments, environment variables, file descriptors.
- **Shell concepts.** Program invocation, redirection, pipes, file-system navigation, argument expansion, quoting, job control, signals, process groups, variables, locale, history and alias functions, security considerations.
- **Scripting.** Plain-text formats, executables, `#!`, shell control structures and functions. Startup scripts.
- **Text, file and networking tools.** sed, grep, chmod, find, ssh, rsync, tar, zip, etc.
- **Software development tools.** C compiler, linker, debugger, make.
- **Revision control systems.** diff, patch, RCS, Subversion, git.
- **Perl.** Introduction to a powerful scripting and text manipulation language. [2 lectures]

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 simple tools can be combined to perform a large variety of tasks;

- be familiar with the most common tools, file formats and configuration practices;
- be able to understand, write, and maintain shell scripts and makefiles;
- appreciate how using revision control systems and fully automated build processes help to maintain reproducibility and audit trails during software development;
- know enough about basic development tools to be able to install, modify and debug C source code;
- have understood the main concepts of and gained initial experience in writing Perl scripts (excluding the facilities for object-oriented programming).

Recommended reading

Robbins, A. (2005). *Unix in a nutshell*. O'Reilly (4th ed.).
Schwartz, R.L., Foy, B.D. & Phoenix, T. (2011). *Learning Perl*. O'Reilly (6th ed.).

Lent Term 2016: Part IB lectures

Compiler Construction

Lecturer: Dr T.G. Griffin

No. of lectures: 16

Suggested hours of supervisions: 4

Prerequisite: Discrete Mathematics (Part IA)

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

Aims

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

Lectures

- **Overview of compiler structure** The spectrum of interpreters and compilers; compile-time and run-time. Structure of a simple compiler: lexical analysis and syntax analysis (details postponed to the last four lectures of term), type checking, intermediate representations, optimisations, code generation. Overview of run-time data structures: stack and heap. Virtual machines. [1 lecture]

- **A Small LANGguage (SLANG)** A complete compiler for a small language will illustrate some of the main concepts. Compilation as a sequence of translations from *higher-level* to *lower-level* intermediate languages. Code generation from intermediate code. [3 lectures]

- **Data structures, procedures/functions** Representing tuples, arrays, references. Procedures and functions: calling conventions, nested structure, non-local variables. Functions as *first-class* values represented as *closures*. Simple optimisations: inline expansion, constant folding, elimination of tail recursion, peephole optimisation. [4 lectures]

- **Advanced topics** Run-time memory management (garbage collection). Static and dynamic linking. Objects and inheritance; implementation of method dispatch. Try-catch exception mechanisms. How to compile a compiler via bootstrapping. [4 lectures]

- **Lexical analysis and syntax analysis.** Lexical analysis based on regular expressions and finite state automata. Using LEX-tools. How does LEX work? Parsing based on context-free grammars and push-down automata. Grammar ambiguity, left- and right-associativity and operator precedence. Using YACC-like tools. How does YACC work? LL(k) and LR(k) parsing theory. [4 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

* Aho, A.V., Sethi, R. & Ullman, J.D. (2007). *Compilers: principles, techniques and tools*. Addison-Wesley (2nd ed.).
Mogensen, T. Æ. (2011). *Introduction to compiler design*. Springer.
`http://www.diku.dk/ torbenm/Basics`.

Computation Theory

Lecturer: Professor A.M. Pitts

No. of lectures: 12

Suggested hours of supervisions: 3

Prerequisite course: Discrete Mathematics

This course is a prerequisite for Complexity Theory (Part IB).

Aims

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

Lectures

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

- **Register machines.** Definition and examples; graphical notation. Register machine computable functions. Doing arithmetic with register machines. [1 lecture]

- **Universal register machine.** Natural number encoding of pairs and lists. Coding register machine programs as numbers. Specification and implementation of a universal register machine. [2 lectures]

- **Undecidability of the halting problem.** Statement and proof. Example of an uncomputable partial function. Decidable sets of numbers; examples of undecidable sets of numbers. [1 lecture]
- **Turing machines.** Informal description. Definition and examples. Turing computable functions. Equivalence of register machine computability and Turing computability. The Church-Turing Thesis. [2 lectures]
- **Primitive and partial recursive functions.** Definition and examples. Existence of a recursive, but not primitive recursive function. A partial function is partial recursive if and only if it is computable. [2 lectures]
- **Lambda-Calculus.** Alpha and beta conversion. Normalization. Encoding data. Writing recursive functions in the lambda-calculus. The relationship between computable functions and lambda-definable functions. [3 lectures]

Objectives

At the end of the course students should

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

Recommended reading

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

Databases

Lecturer: Dr T.G. Griffin

No. of lectures: 12

Suggested hours of supervisions: 3

Prerequisite courses: None

Aims

The aim of the course is to cover the fundamentals of databases as seen from the perspective of application writers. The course covers schema design techniques, SQL, data warehouses, On-line Analytical Processing (OLAP), federated databases, and some aspects of the NoSQL movement.

Lectures

- **Introduction.** What is a database system? Different roles : database implementor, database administrator, database applications developer, database user. This course will focus on database systems from the perspective of *application developers*. [1 lecture]
- **Data models and query languages** We explore and contrast two distinct data models: *relational* and *semi-structured* models. Relational database systems are the most widespread and typically provide SQL-based data querying and modification. We illustrate the basic constructs of SQL and On-Line Transaction Processing (OLAP). Some NoSQL databases (such as MongoDB) are based on semi-structured models, and have a more procedural means of inspecting data. [4 lecture]
- **Modeling (some bit of) the real world in a database** Entity-Relationship (ER) modeling for the relational model. Object-oriented modeling for semi-structured data. What is a *good* representation? What are the design tradeoffs involved? A persistent problem: data redundancy leads to update anomalies. Relational solution: normalization! Semi-structured solution: good luck pal! [3 lecture]
- **Relational normal forms** Functional dependencies (FDs) as a formal means of investigating redundancy. Relational decomposition. Schema normalization. Third normal form and Boyce–Codd normal form. [1 lecture]
- **Schema evolution** The real world is constantly changing and your database design has to change with it. Semi-structured solution: look mom, no schema! Relational solution: good luck pal! [1 lecture]
- **Embracing redundancy** If rarely updating but almost always reading, then normalization is too costly. On-line Analytical Processing (OLAP). OLTP *versus* OLAP. A somewhat related topic: federated databases. Data exchange languages such as XML and JSON. A close look at some federated databases from EBI (`http://www.ebi.ac.uk/services`). [2 lecture]

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
- be able to program simple database applications
- 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

* 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.
Date, C.J. (2004). *An introduction to database systems*. Addison-Wesley (8th ed.).

Logic and Proof

Lecturer: Professor L.C. Paulson

No. of lectures: 12

Suggested hours of supervisions: 3

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

Aims

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

Lectures

- **Introduction to logic.** Schematic statements. Interpretations and validity. Logical consequence. Inference.

- **Propositional logic.** Basic syntax and semantics. Equivalences. Normal forms. Tautology checking using CNF.
- **The sequent calculus.** A simple (Hilbert-style) proof system. Natural deduction systems. Sequent calculus rules. Sample proofs.
- **First order logic.** Basic syntax. Quantifiers. Semantics (truth definition).
- **Formal reasoning in FOL.** Free *versus* bound variables. Substitution. Equivalences for quantifiers. Sequent calculus rules. Examples.
- **Clausal proof methods.** Clause form. A SAT-solving procedure. The resolution rule. Examples. Refinements.
- **Skolem functions, Unification and Herbrand's theorem.** Prenex normal form. Skolemisation. Most general unifiers. A unification algorithm. Herbrand models and their properties.
- **Resolution theorem-proving and Prolog.** Binary resolution. Factorisation. Example of Prolog execution. Proof by model elimination.
- **Satisfiability Modulo Theories.** Decision problems and procedures. How SMT solvers work.
- **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;
- understand the concept of a decision procedure and the basic principles of "satisfiability modulo theories".
- 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.).

Easter Term 2016: Part IB lectures

Artificial Intelligence I

Lecturer: Dr S.B. Holden

No. of lectures: 12

Suggested hours of supervisions: 3

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. Similarly, elements of Mathematical Methods for Computer Science, Logic and Proof, Prolog and Complexity Theory are likely to be useful.

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

Aims

The aim of this course is to provide an introduction to some fundamental 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 fundamental tools and algorithms required to produce AI systems able to exhibit limited human-like abilities, particularly in the form of problem solving by search, game-playing, representing and reasoning with knowledge, planning, and learning.

Lectures

- **Introduction.** Alternate ways of thinking about AI. *Agents* as a unifying view of AI systems. [1 lecture]
- **Search I.** Search as a fundamental paradigm for intelligent problem-solving. Simple, *uninformed search* algorithms. Tree search and graph search. [1 lecture]
- **Search II.** More sophisticated *heuristic search* algorithms. The A* algorithm and its properties. Improving memory efficiency: the IDA* and recursive best first search algorithms. Local search and gradient descent. [1 lecture]
- **Game-playing.** Search in an adversarial environment. The minimax algorithm and its shortcomings. Improving minimax using alpha-beta pruning. [1 lecture]
- **Constraint satisfaction problems (CSPs).** Standardising search problems to a common format. The backtracking algorithm for CSPs. Heuristics for improving the search for a solution. Forward checking, constraint propagation and arc consistency. [1 lecture]

- **Backjumping in CSPs.** Backtracking, backjumping using Gaschnig's algorithm, graph-based backjumping. [1 lecture]
- **Knowledge representation and reasoning I.** 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. [1 lectures]
- **Knowledge representation and reasoning II.** Knowledge representation and reasoning using first order logic. The frame, qualification and ramification problems. The situation calculus. [1 lectures]
- **Planning I.** Methods for planning in advance how to solve a problem. The STRIPS language. Achieving preconditions, backtracking and fixing threats by promotion or demotion: the partial-order planning algorithm. [1 lecture]
- **Planning II.** Incorporating heuristics into partial-order planning. Planning graphs. The GRAPHPLAN algorithm. Planning using propositional logic. Planning as a constraint satisfaction problem. [1 lecture]
- **Neural Networks I.** A brief introduction to supervised learning from examples. Learning as fitting a curve to data. The perceptron. Learning by gradient descent. [1 lecture]
- **Neural Networks II.** Multilayer perceptrons and the backpropagation algorithm. [1 lecture]

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 the fact that the computational complexity of most AI problems requires us regularly to deal with approximate techniques;
- be able to design basic problem solving methods based on AI-based search, knowledge representation, reasoning, planning, and learning algorithms.

Recommended reading

The recommended text is:

* Russell, S. & Norvig, P. (2010). *Artificial intelligence: a modern approach*. Prentice Hall (3rd ed.).
There are many good books available on artificial intelligence; one alternative is:

Poole, D. L. & Mackworth, A. K. (2010). *Artificial intelligence: foundations of computational agents*. Cambridge University Press.

For some of the material you might find it useful to consult more specialised texts, in particular:

Dechter, R. (2003). *Constraint processing*. Morgan Kaufmann.
Cawsey, A. (1998). *The essence of artificial intelligence*. Prentice Hall.
Ghallab, M., Nau, D. & Traverso, P. (2004). *Automated planning: theory and practice*. Morgan Kaufmann.
Bishop, C.M. (2006). *Pattern recognition and machine learning*. Springer.

Complexity Theory

Lecturer: Professor A. Dawar

No. of lectures: 12

Suggested hours of supervisions: 3

Prerequisite courses: Algorithms, Computation Theory

Aims

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

Lectures

- **Algorithms and problems.** Complexity of algorithms and of problems. Lower and upper bounds. Examples: sorting and travelling salesman.
- **Time and space.** Models of computation and measures of complexity. Time and space complexity on a Turing machine. Decidability and complexity.
- **Time complexity.** Time complexity classes. Polynomial time problems and algorithms. P and NP.
- **Non-determinism.** Non-deterministic machines. The class NP redefined. Non-deterministic algorithms for reachability and satisfiability.
- **NP-completeness.** Reductions and completeness. NP-completeness of satisfiability.
- **More NP-complete problems.** Graph-theoretic problems. Hamiltonian cycle and clique.
- **More NP-complete problems.** Sets, numbers and scheduling. Matching, set covering and bin packing.

- **coNP.** Validity of boolean formulae and its completeness. NP ∩ coNP. Primality and factorisation.
- **Cryptographic complexity.** One-way functions. The class UP.
- **Space complexity.** Deterministic and non-deterministic space complexity classes. The reachability method. Savitch's theorem.
- **Hierarchy.** The time and space hierarchy theorems and complete problems.
- **Descriptive complexity.** Logics capturing complexity classes. Fagin's theorem.

Objectives

At the end of the course students should

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

Recommended reading

* Papadimitriou, Ch.H. (1994). *Computational complexity*. Addison-Wesley.
Goldreich, O. (2010). *P, NP, and NP-Completeness: the basics of computational complexity*. Cambridge University Press. Sipser, M. (1997). *Introduction to the theory of computation*. PWS.

Concepts in Programming Languages

Lecturer: Professor A. Mycroft

No. of lectures: 8

Suggested hours of supervisions: 2

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 ancestors: Fortran, Lisp, Algol and Pascal.** Key ideas: procedural (Fortran), declarative (Lisp), block structured (Algol and Pascal). Execution models (abstract machines), data types, control structures, storage, arrays and pointers, procedures and parameter passing, scope, strict and lazy evaluation, garbage collection. Programs as data (Lisp).
- **Object-oriented languages — Concepts and origins: Simula (1964–67) and Smalltalk (1971–80).** Dynamic lookup. Abstraction. Subtyping. Inheritance. Object models.
- **Languages for parallel processing.** Shared-memory concurrency with spawn/sync (OpenMP, Cilk, X10). Distributed-memory models (the actor model, Erlang).
- **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.
- **Scala: a principled feature-rich language.** Procedural and declarative aspects. Blocks and functions. Classes and objects. Generic types and methods. Variance annotations. Mixin-class composition.
- **More-advanced concepts and idioms.** Haskell monads, type classes. Continuation passing style and call/cc. Dependent types.

Objectives

At the end of the course students should

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

Recommended reading

Books:

* Mitchell, J.C. (2003). *Concepts in programming languages*. Cambridge University Press.

* Scott, M.L. (2009). *Programming language pragmatics*. Morgan Kaufmann.
Odersky, M. (2008). *Scala by example*. Programming Methods Laboratory, EPFL.
Pratt, T.W. & Zelkowitz, M.V. (2001). *Programming languages: design and implementation*. Prentice Hall.

Papers:

Kay, A.C. (1993). The early history of Smalltalk. *ACM SIGPLAN Notices*, Vol. 28, No. 3.
Kernighan, B. (1981). Why Pascal is not my favorite programming language. AT&T Bell Laboratories. *Computing Science Technical Report* No. 100.
Koenig, A. (1994). An anecdote about ML type inference. *USENIX Symposium on Very High Level Languages*.
Landin, P.J. (1966). The next 700 programming languages. *Communications of the ACM*, Vol. 9, Issue 3.
Odersky, M. *et al.* (2006). An overview of the Scala programming language. *Technical Report LAMP-REPORT-2006-001*, Second Edition.
McCarthy, J. (1960). Recursive functions of symbolic expressions and their computation by machine. *Communications of the ACM*, 3(4):184–195.
Stroustrup, B. (1991). What is Object-Oriented Programming? (1991 revised version). *Proceedings 1st European Software Festival*.

Economics, Law and Ethics

Lecturers: Professor R.J. Anderson and Dr R.N. Clayton

No. of lectures: 8

Suggested hours of supervisions: 2

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

Aims

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

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.** Definitions: preference, utility, choice and budget. Pareto efficiency; the discriminating monopolist; supply and demand; elasticity; utility; the marginalist revolution; competitive equilibrium and the welfare theorems. Trade; monopoly rents; public goods; oligopoly.
- **Market failure.** Asymmetric information: the market for lemons; adverse selection; moral hazard; signalling; and brands. Transaction costs and the theory of the firm.

Real and virtual networks, supply-side *versus* demand-side scale economies, Metcalfe's law, the dominant firm model, price discrimination. Behavioural economics: bounded rationality, heuristics and biases.

- **Auctions.** English auctions; Dutch auctions; all-pay auctions; Vickrey auctions. The winner's curse. The revenue equivalence theorem. Mechanism design and the combinatorial auction. Problems with real auctions. Applicability of auction mechanisms in computer science.
- **Principles of law.** Contract and tort; copyright and patent; binding actions; liabilities and remedies; competition law; choice of law and jurisdiction.
- **Law and the Internet.** EU directives including distance selling, electronic commerce, data protection, electronic signatures and copyright; their UK implementation. UK laws that specifically affect the Internet, including RIP.
- **Ethics.** Philosophies of ethics: authority, intuitionist, egoist and deontological theories. Utilitarian and Rawlsian models. Insights from evolutionary psychology and neorology. The Internet and social policy; current debates on privacy, surveillance, censorship and export control.

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, legislation and ethics place on firms dealing in information goods and services.

Recommended reading

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

Further reading:

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

Security I

Lecturer: Dr M.G. Kuhn

No. of lectures: 12

Suggested hours of supervisions: 3

Prerequisite courses: Mathematical Methods I, Discrete Mathematics, Operating Systems, Complexity Theory

This course is a prerequisite for Security II.

Aims

This course covers some essential computer-security techniques, focussing mainly on private-key cryptography, discretionary access control and common software vulnerabilities.

Lectures

- **Introduction.** Malicious intent. Security policies, targets, mechanisms. Aspects of confidentiality, integrity, availability, privacy. Requirements across different applications.
- **Cryptography.** Overview, private vs. public-key ciphers, MACs vs. signatures, certificates, capabilities of adversary, Kerckhoffs' principle.
- **Classic ciphers.** Attacks on substitution and transposition ciphers, Vigenére. Perfect secrecy: one-time pads.
- **Private-key encryption.** Stream ciphers, pseudo-random generators, attacking linear-congruential RNGs and LFSRs. Semantic security definitions, oracle queries, advantage, computational security, security proofs.
- **Block ciphers.** Pseudo-random functions and permutations. Birthday problem, random mappings. Feistel/Luby–Rackoff structure, DES, TDES, AES.
- **Chosen-plaintext attack security.** Security with multiple encryptions, randomized encryption. Modes of operation: ECB, CBC, OFB, CNT.
- **Message authenticity.** Malleability, MACs, existential unforgeability, CBC-MAC, ECBC-MAC, CMAC, birthday attacks, Carter-Wegman one-time MAC.
- **Authenticated encryption.** Chosen-ciphertext attack security, ciphertext integrity, encrypt-and-authenticate, authenticate-then-encrypt, encrypt-then-authenticate, padding oracle example, GCM.
- **Entity authentication.** Passwords, trusted path, phishing, CAPTCHA. Authentication protocols: replay attacks, one-way and challenge–response protocols, Needham–Schroeder, protocol failure examples.

- **Operating system security.** Trusted computing base, domain separation, reference mediation, residual information protection.
- **Discretionary access control.** Matrix model, DAC in POSIX and Windows, elevated rights and setuid bits, capabilities, Clark–Wilson integrity.
- **Software security.** Malicious software. Common implementation vulnerabilities: buffer overflows, integer overflows, meta characters, syntax incompatibilities, race conditions, unchecked values, side channels, random-bit sources.

Objectives

By the end of the course students should

- be familiar with core security terms and concepts;
- understand security definitions of modern private-key cryptographic primitives;
- understand the POSIX and Windows NTFS discretionary access control system;
- understand the most common security pitfalls in software development.

Recommended reading

Katz, J., Lindell, Y. (2015). *Introduction to modern cryptography*. Chapman & Hall/CRC (2nd ed.).
Paar, Ch. & Pelzl, J. (2010). *Understanding cryptography*. Springer.
Gollmann, D. (2010). *Computer security*. Wiley (3rd 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 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/timetables/`

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:

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

telephone: 01223 763505
fax: 01223 334678
e-mail: `teaching-admin@cl.cam.ac.uk`

Michaelmas Term 2015: Part II lectures

Bioinformatics

Lecturer: Dr P. Liò

No. of lectures: 12

Suggested hours of supervisions: 3

Aims

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

Lectures

- **Introduction to biological data:** Bioinformatics as an interesting field in computer science.
- **Dynamic programming.** Longest common subsequence, DNA, RNA, protein structure alignment, linear space alignment, heuristics for multiple alignment.
- **Sequence database search.** Blast, Patternhunter.
- **Next Generation Sequencing.** De Bruijn graph, BurrowsWheeler transform.
- **Phylogeny – parsimony-based.** Fitch, Wagner, Sankoff parsimony.
- **Phylogeny – distance-based.** UPGMA, Neighbour Joining.
- **Clustering.** K-means, Markov Clustering algorithm.
- **Applications of Hidden Markov Models.**
- **Searching motifs in sequence alignment.** Gibbs sampling.
- **Biological networks:** reverse engineering algorithms and dynamics; Wagner, Gillespie.

Objectives

At the end of this course students should

- understand Bioinformatics terminology;
- have mastered the most important algorithms in the field;

- be able to work with bioinformaticians and biologists;
- be able to find data and literature in repositories.

Recommended reading

* Durbin, R., Eddy, S., Krough, A. & Mitchison, G. (1998). *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge University Press.
Jones, N.C. & Pevzner, P.A. (2004). *An introduction to bioinformatics algorithms*. MIT Press.
Felsenstein, J. (2003). *Inferring phylogenies*. Sinauer Associates.

Business Studies

Lecturer: Mr J.A. Lang

No. of lectures 8

Suggested hours of supervisions: 2

Prerequisite course: Economics and Law

This course is a prerequisite for E-Commerce.

Aims

How to start and run a computer company; 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/`

Denotational Semantics

Lecturer: Professor M.P. Fiore

No. of lectures: 10

Suggested hours of supervisions: 3

Aims

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

Lectures

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

Objectives

At the end of the course students should

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

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

Recommended reading

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

Digital Signal Processing

Lecturer: Dr M.G. Kuhn

No. of lectures: 12

Suggested hours of supervisions: 3

Prerequisite courses: Mathematical Methods I–III, Mathematical Methods for Computer Science, LaTeX and MATLAB.

Aims

This course teaches the basic signal-processing principles necessary to understand many modern high-tech systems, with digital-communications examples. Students will gain practical experience from numerical experiments in MATLAB-based programming assignments.

Lectures

- **Signals and systems.** Discrete sequences and systems, their types and properties. Linear time-invariant systems, convolution.
- **Phasors.** Eigen functions of linear time-invariant systems. Review of complex arithmetic. Some examples from electronics, optics and acoustics.
- **Fourier transform.** Phasors as orthogonal base functions. Forms and properties of the Fourier transform. Convolution theorem.
- **Dirac's delta function.** Fourier representation of sine waves, impulse combs in the time and frequency domain.

- **Discrete sequences and spectra.** Periodic sampling of continuous signals, periodic signals, aliasing, interpolation, sampling and reconstruction of low-pass and band-pass signals, spectral inversion.
- **Digital modulation.** IQ representation of band-pass signals, in particular AM, FM, PSK, and QAM signals.
- **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 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.
- **Infinite impulse-response filters.** Sequences as polynomials, z-transform, zeros and poles, some 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.

Objectives

By the end of the course students should be able to

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

Recommended reading

* Lyons, R.G. (2010). *Understanding digital signal processing.* Prentice Hall (3rd ed.).
Oppenheim, A.V. & Schafer, R.W. (2007). *Discrete-time digital signal processing.* Prentice Hall (3rd ed.).

HumanComputer Interaction

Lecturer: Professor A.F. Blackwell

No. of lectures: 8

Suggested hours of supervisions: 2

Aims

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

Lectures

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

Objectives

On completing the course, students should be able to

- propose design approaches that are suitable to different classes of user and application;

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

Recommended reading

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

Further reading:

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

Information Theory

Lecturer: Professor J.G. Daugman

No. of lectures: 12

Suggested hours of supervisions: 3

Prerequisite courses: Mathematical Methods for Computer Science

Aims

The aims of this course are to introduce the principles and applications of information theory. The course covers how information is measured in terms of probability and various entropies, and how these are used to calculate the capacity of communication channels, continuous or discrete, with or without noise. Coding schemes including error correcting codes are studied along with data compression, spectral analysis, and efficient coding using wavelets. Applications of information theory are also reviewed, from bioinformatics to pattern recognition.

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.** Markov sources. Entropy rate of a Markov process. Symbol codes. Huffman codes and the prefix property. Binary symmetric channels. Capacity of a noiseless discrete channel.

- **Discrete channel properties, noise, and channel capacity.** Perfect communication through a noisy channel: error-correcting codes. Capacity of a discrete channel as the maximum of its mutual information over all possible input distributions.

- **Spectral properties of continuous-time signals and channels.** Signals represented as combinations of complex exponential eigenfunctions; channels represented as spectral filters that add noise. Applying Fourier analysis to signal communication. Continuous versus discrete, and periodic versus aperiodic signals and their transforms. Duality properties.

- **Continuous information; density; noisy channel coding theorem.** Extensions of 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 for noisy continuous channels.

- **Signal coding and transmission schemes using Fourier theorems.** Nyquist Sampling Theorem. Aliasing and its prevention. Modulation and shift theorems; multiple carriers; frequency and phase modulation codes; ensembles. Filters, coherence, demodulation; noise removal by correlation.

- **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.

- **Data compression codes and protocols.** Run-length coding; dictionary methods on strings; vector quantisation; JPEG and JP2K image compression; orthogonal subspace projections; predictive coding; the Laplacian pyramid; and wavelet scalar quantisation.

- **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.

- **Applications of information theory in other sciences.** Use of information metrics and analysis in: genomics; neuroscience; astrophysics; noisy signal classification; and pattern recognition including biometrics.

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 encoding and communication schemes in terms of the spectral properties of signals and channels;
- describe compression schemes, and efficient coding using wavelets and other representations for data.

Recommended reading

* Cover, T.M. & Thomas, J.A. (2006). *Elements of information theory*. New York: Wiley.

Natural Language Processing

Lecturer: Dr E. Shutova

No. of lectures: 12

Suggested hours of supervisions: 3

Prerequisite courses: Mathematical Methods for Computer Science, Logic and Proof, and Artificial Intelligence I

Aims

This course introduces the fundamental techniques of natural language processing. It aims to explain the potential and the main limitations of these techniques. Some current research issues are introduced and some current and potential applications discussed and evaluated.

Lectures

The order of delivery of the lectures is provisional.

- **Introduction.** Brief history of NLP research, current applications, components of NLP systems.
- **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.
- **Context-free grammars and parsing.** Generative grammar, context-free grammars, parsing with context-free grammars, weights and probabilities. Limitations of context-free grammars.
- **Constraint-based grammars.** Constraint-based grammar, unification.
- **Compositional semantics.** Compositional semantics with FOPC and lambda calculus. Simple compositional semantics in constraint-based grammar. Dependency structures. Inference and robust entailment.
- **Lexical semantics.** Semantic relations, WordNet, word senses, word sense disambiguation.
- **Distributional semantics.** Representing lexical meaning with distributions. Similarity metrics. Clustering.
- **Discourse and dialogue.** Anaphora resolution, discourse relations.
- **Language generation.** Generation and regeneration. Components of a generation system.
- **Computational psycholinguistics.** Modelling human language use.
- **Recent trends and applications.** Recent trends in NLP research and examples of practical applications of NLP techniques.

Objectives

At the end of the course students should

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

Recommended reading

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

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

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

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

Optimising Compilers

Lecturer: Professor A. Mycroft

No. of lectures: 16

Suggested hours of supervisions: 4

Prerequisite course: Compiler Construction

Aims

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

Lectures

- **Introduction and motivation.** Outline of an optimising compiler. Optimisation partitioned: *analysis* shows a property holds which enables a *transformation*. The flow graph; representation of programming concepts including argument and result passing. The phase-order problem.
- **Kinds of optimisation.** Local optimisation: peephole optimisation, instruction scheduling. Global optimisation: common sub-expressions, code motion. Interprocedural optimisation. The call graph.
- **Classical dataflow analysis.** Graph algorithms, *live* and *avail* sets. Register allocation by register colouring. Common sub-expression elimination. Spilling to memory; treatment of CSE-introduced temporaries. Data flow anomalies. Static Single Assignment (SSA) form.
- **Higher-level optimisations.** Abstract interpretation, Strictness analysis. Constraint-based analysis, Control flow analysis for lambda-calculus. Rule-based

inference of program properties, Types and effect systems. Points-to and alias analysis.

- **Target-dependent optimisations.** Instruction selection. Instruction scheduling and its phase-order problem.
- **Decompilation.** Legal/ethical issues. Some basic ideas, control flow and type reconstruction.

Objectives

At the end of the course students should

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

Recommended reading

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

Principles of Communications

Lecturer: Professor J.A. Crowcroft

No. of lectures: 24

Suggested hours of supervisions: 6

Prerequisite course: Computer Networking

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

This course may be useful for the Part III course on Network Architectures.

Useful related courses: Computer Systems Modelling, Information Theory, Digital Signal Processing

Aims

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

Lectures

- **Introduction.** Course overview. Abstraction, layering. The structure of real networks, links, end systems and switching systems. [1 lecture]
- **Modular functionality for communications.** Some systems design paradigms, often orthogonal to layers. [2 lectures]
- **Topology and Routing.** How many ways can we get from A to B? We review relevant graph theory, including recent advances in understanding the topology of the Internet and similar networks. We then look at the two most common class of routing algorithms. [4 lectures]
- **Error control.** What do we do when things go wrong? Information can be coded and transmitted in a number of ways to survive interference. Retransmit, or pre-transmit? [2 lectures]
- **Flow control.** Control theory is a branch of engineering familiar to people building dynamic machines. It can be applied to network traffic. Stemming the flood, at source, sink, or in between? Optimization as a model of network& user. [4 lectures]
- **Shared media networks.** Ethernet and Radio networks: some special problems for media access and so forth. We revisit the problem of capacity of a channel in the context of a radio network. [2 lectures]
- **Switched networks.** What does a switch have to do, and how? [2 lectures]
- **Integrated Service Packet Networks for IP.** Traffic may be adaptive to feedback control, or it may be a given; characteristics may be quite complex in terms of time series. This has an impact on the design choices for scheduling and queue management algorithms for packet forwarding, including APIs to Quality of Service and routing with QoS. [2 lectures]
- **The big picture for managing traffic.** Economics and policy are relevant to networks in many ways. Optimisation and game theory are both relevant topics discussed here. [2 lectures]

Objectives

At the end of the course students should be able to explain the underlying design and behaviour of networks, including capacity, topology, control and use.

Recommended reading

* Keshav, S. (2012). *Mathematical Foundations of Computer Networking*. Addison Wesley. ISBN 9780321792105
Background reading:
Keshav, S. (1997). *An engineering approach to computer networking*. Addison-Wesley (1st ed.). ISBN 0201634422
Stevens, W.R. (1994). *TCP/IP illustrated, vol. 1: the protocols*. Addison-Wesley (1st ed.). ISBN 0201633469

Quantum Computing

Lecturer: Professor A. Dawar

No. of lectures: 8

Suggested hours of supervisions: 2

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. (2010). *Quantum computation and quantum information*. Cambridge University Press (2nd ed.).
Mermin, N.D. (2007). *Quantum computer science*. Cambridge University Press.

LaTeX and MATLAB

Lecturer: Dr M.G. Kuhn

No. of lectures: 2

Suggested hours of supervisions: 0–1 (non-examinable course with exercises)

LaTeX skills are useful for preparing the Part II dissertation. MATLAB skills are useful for programming exercises in some Part II courses (e.g. Digital Signal Processing).

Aims

Introduction to two widely-used languages for typesetting dissertations and scientific publications, for prototyping numerical algorithms and to visualize results.

Lectures

- **LaTeX.** Workflow example, syntax, typesetting conventions, non-ASCII characters, document structure, packages, mathematical typesetting, graphics and figures, cross references, build tools.
- **MATLAB.** Tools for technical computing and visualization. The matrix type and its operators, 2D/3D plotting, common functions, function definitions, toolboxes, vectorized audio demonstration.

Objectives

Students should be able to avoid the most common LaTeX mistakes, to prototype simple image and signal processing algorithms in MATLAB, and to visualize the results.

Recommended reading

* Lamport, L. (1994). *LaTeX – a documentation preparation system user's guide and reference manual*. Addison-Wesley (2nd ed.).

Mittelbach, F., et al. (2004). *The LaTeX companion*. Addison-Wesley (2nd ed.).

Types

Lecturer: Professor A.M. Pitts

No. of lectures: 12

suggested hours of supervisions: 3

Prerequisite courses: Computation Theory, Semantics of Programming Languages

Aims

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

Lectures

- **Introduction.** The role of type systems in programming languages. Review of rule-based formalisation of 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 (PLC).** Explicit versus implicitly typed languages. PLC syntax and reduction semantics. Examples of datatypes definable in the polymorphic lambda calculus. [3 lectures]
- **Dependent types.** Dependent function types. Pure type systems. System F-omega. [2 lectures]
- **Propositions as types.** Example of a non-constructive proof. The Curry-Howard correspondence between intuitionistic second-order propositional calculus and PLC. The calculus of Constructions. Inductive types. [3 lectures]

Objectives

At the end of the course students should

- be able to use a rule-based specification of a type system to carry out type checking and type inference;
- understand by example the Curry-Howard correspondence between type systems and logics;
- appreciate the expressive power of parametric polymorphism and dependent types.

Recommended reading

* Pierce, B.C. (2002). *Types and programming languages*. MIT Press.
Pierce, B. C. (Ed.) (2005). *Advanced Topics in Types and Programming Languages*. MIT Press.
Girard, J-Y. (tr. Taylor, P. & Lafont, Y.) (1989). *Proofs and types*. Cambridge University Press.

Lent Term 2016: Part II lectures

Advanced Graphics

Lecturers: Dr P.A. Benton, Dr R.A. Mantiuk

No. of lectures: 16

Suggested hours of supervisions: 4

Prerequisite course: Computer Graphics and Image Processing

Aims

This course provides students with a solid grounding in the main three-dimensional modelling and rendering mechanisms. It also introduces supporting topics, including graphics cards, mobile graphics, and animation.

Lectures

The order of delivery of lectures is provisional and subject to change.

- **Computational geometry.** The mathematics of discrete geometry: what can you know, and how well can you know it?
- **Ray tracing.** The fundamentals of raycasting, constructive solid geometry (CSG), and bounding volumes.
- **Implicit surfaces, voronoi diagrams.** Useful graphical and geometric techniques.
- **Splines and subdivision surfaces.** Bézier curves and B-splines, from uniform, non-rational B-splines through to non-uniform, rational B-splines (NURBS). Introduction to subdivision. The key methods. Pros and cons when compared with NURBS.
- **Advanced illumination techniques.** Radiosity and photon mapping.
- **OpenGL, graphics cards, and shaders.** Tools and technologies available today; previews of what's coming tomorrow.
- **Optics and cameras**
- **Light and colour, photometry and colorimetry**
- **Models of early visual perception**
- **Post-processing for graphics**
- **Selected topics of computational photography**
- **Image metrics and quality assessment**

Objectives

On completing the course, students should be able to:

- compare and contrast ray tracing with polygon scan conversion;
- define NURBS basis functions, and explain how NURBS curves and surfaces are used in 2D and 3D modelling;
- describe the underlying theory of subdivision and define the Catmull-Clark and Doo-Sabin subdivision methods;
- understand the core technologies of ray tracing, constructive solid geometry, computational geometry, implicit surfaces, and particle systems;
- understand several global illumination technologies such as radiosity and photon mapping, and be able to discuss each in detail;
- be able to describe current graphics technology and discuss future possibilities.
- understand the components and basic limitations of cameras and optics.
- differentiate between different measures of light and colour, know which to apply to a particular problem.
- be able to choose right display (post-processing) algorithms for a given rendering problem.
- apply suitable visual models in graphics and imaging applications.
- select proper metrics for measuring quality of images and video.

Recommended reading

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

* Shirley, P. & Marschner, S. (2009). *Fundamentals of Computer Graphics*. CRC Press (3rd ed.).
Slater, M., Steed, A. & Chrysanthou, Y. (2002). *Computer graphics and virtual environments: from realism to real-time*. Addison-Wesley.
Watt, A. (1999). *3D Computer graphics*. Addison-Wesley (3rd ed).
de Berg, M., Cheong, O., van Kreveld, M. & Overmars, M. (2008). *Computational geometry: algorithms and applications*. Springer (3rd ed.).
Rogers, D.F. & Adams, J.A. (1990). *Mathematical elements for computer graphics*. McGraw-Hill (2nd ed.).
Warren, J. & Weimer, H. (2002). *Subdivision methods for geometric design*. Morgan Kaufmann.

Artificial Intelligence II

Lecturer: Dr S.B. Holden

No. of lectures: 16

Suggested hours of supervisions: 4

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

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

Lectures

- **Further planning.** Incorporating heuristics into partial-order planning. Planning graphs. The GRAPHPLAN algorithm. Planning using propositional logic. Planning as a constraint satisfaction problem. [3 lectures]

- **Uncertainty and Bayesian networks.** Review of probability as applied to AI. Representing uncertain knowledge using Bayesian networks. Inference in Bayesian networks using both exact and approximate techniques. Other ways of dealing with uncertainty. [2 lectures]

- **Utility and decision-making.** The concept of *utility*. Utility and preferences. Deciding how to act by maximising expected utility. Decision networks. The value of information, and reasoning about when to gather more. [1 lectures]

- **Uncertain reasoning over time.** Markov processes, transition and sensor models. Inference in temporal models: filtering, prediction, smoothing and finding the most likely explanation. The Viterbi algorithm. Hidden Markov models. [2 lectures]

- **Reinforcement learning.** Learning from rewards and punishments. Markov decision processes. The problems of temporal credit assignment and exploration versus exploitation. Q-learning and its convergence. How to choose actions. [2 lecture]

- **Further supervised learning I.** Bayes theorem as applied to supervised learning. The maximum likelihood and maximum *a posteriori* hypotheses. What does this teach us about the backpropagation algorithm? [1 lecture]
- **. How to classify optimally.** Bayesian decision theory and Bayes optimal classification. What does this tell us about how best to do supervised machine learning? [1 lecture]
- **Further supervised learning II.** Applying the Bayes optimal classification approach to neural networks. Markov chain Monte Carlo methods, the evidence and how to choose hyperparameters. [4 lectures]

Objectives

At the end of this course students should:

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

The recommended text is:

* Russell, S. & Norvig, P. (2010). *Artificial intelligence: a modern approach*. Prentice Hall (3rd ed.).
For some material you may find more specialised texts useful, in particular:

Bishop, C.M. (2006). *Pattern recognition and machine learning*. Springer.
Ghallab, M., Nau, D. & Traverso, P. (2004). *Automated planning: theory and practice*. Morgan Kaufmann.
Sutton, R.S., & Barto, A.G. (1998). *Reinforcement learning: an introduction*. MIT Press.

Comparative Architectures

Lecturer: Dr R.D. Mullins

No. of lectures: 16

Suggested hours of supervisions: 4

Prerequisite course: Computer Design

Aims

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

Lectures

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

Objectives

At the end of the course students should

- understand what determines processor design goals;
- appreciate what constrains the design process and how architectural trade-offs are made within these constraints;

- be able to describe the architecture and operation of pipelined and superscalar processors, including techniques such as branch prediction, register renaming and out-of-order execution;
- have an understanding of vector, multithreaded and multi-core processor architectures;
- for the architectures discussed, understand what ultimately limits their performance and application domain.

Recommended reading

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

Computer Systems Modelling

Lecturer: Dr R.J. Gibbens

No. of lectures: 12

Suggested hours of supervisions: 3

Prerequisite courses: 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.** Basic approaches and applications to the modelling computer systems.
- **Random number generation methods and simulation techniques.** Statistical aspects of simulations: confidence intervals, stopping criteria, variance reduction techniques. [2 lectures]
- **Simple stochastic processes.** Introduction and examples. The Poisson process. [2 lectures]
- **Birth-death processes, flow balance equations.** Birth-death processes and their relation to queueing systems. The M/M/1 queue in detail: the equilibrium distribution with conditions for existence and common performance metrics. [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.
Harchol-Balter, M. (2013). *Performance modeling and design of computer systems: queueing theory in action*. Cambridge University Press.
Jain, A.R. (1991). *The art of computer systems performance analysis*. Wiley.
Kleinrock, L. (1975). *Queueing systems, vol.* 1*. Theory*. Wiley.
Mitzenmacher, M. & Upfal, E. (2005). *Probability and computing: randomized algorithms and probabilistic analysis*. Cambridge University Press.

Computer Vision

Lecturer: Professor J.G. Daugman

No. of lectures: 16

Suggested hours of supervisions: 4

Prerequisite courses: Probability, Mathematical Methods for Computer Science

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. Issues will be illustrated using the examples of pattern recognition, image retrieval, and face recognition.

Lectures

- **Goals of computer vision; why they are so difficult.** How images are formed, and the ill-posed problem of making 3D inferences from them about objects and their properties.
- **Image sensing, pixel arrays, CCD cameras.** Image coding and information measures. Elementary operations on image arrays.
- **Biological visual mechanisms, from retina to cortex.** Photoreceptor sampling; receptive field profiles; stochastic impulse codes; channels and pathways. Neural image encoding operators.
- **Mathematical operations 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-scale feature detection and matching.** SIFT (scale-invariant feature transform); pyramids. 2D wavelets as visual primitives. Energy-minimising snakes; active contours.
- **Higher 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. Image and motion segmentation.
- **Lambertian and specular surfaces; reflectance maps.** Geometric analysis of image formation from surfaces. Discounting the illuminant when inferring 3D structure and surface properties.
- **Shape representation.** Inferring 3D shape from shading; surface geometry. Boundary descriptors; codons. Object-centred coordinates and the "2.5-Dimensional" sketch.
- **Perceptual organisation and cognition.** Vision as model-building and graphics in the brain. Learning to see.
- **Lessons from neurological trauma and visual deficits.** Visual agnosias and illusions, and what they may imply about how vision works.
- **Bayesian inference in vision; knowledge-driven interpretations.** Classifiers, decision-making, and pattern recognition.

- **Model estimation.** Machine learning and statistical methods in vision.

- **Applications of machine learning in computer vision.** Discriminative and generative methods. Content based image retrieval.

- **Approaches to face detection, face recognition, and facial interpretation.** Cascaded detectors. Appearance *versus* model-based methods (2D and 3D approaches).

Objectives

At the end of the course students should

- understand visual processing from both “bottom-up” (data oriented) and “top-down” (goals oriented) perspectives;

- be able to decompose visual tasks into sequences of image analysis operations, representations, specific algorithms, and inference principles;

- understand the roles of image transformations and their invariances in pattern recognition and classification;

- be able to describe and contrast techniques for extracting and representing features, edges, shapes, and textures;

- be able to describe key aspects of how biological visual systems work; and be able to think of ways in which biological visual strategies might be implemented in machine vision, despite the enormous differences in hardware;

- be able to analyse the robustness, brittleness, generalizability, and performance of different approaches in computer vision;

- understand the roles of machine learning in computer vision today, including probabilistic inference, discriminative and generative methods;

- understand in depth at least one major practical application problem, such as face recognition, detection, or interpretation.

Recommended reading

* Forsyth, D. A. & Ponce, J. (2003). *Computer Vision: A Modern Approach*. Prentice Hall.

Shapiro, L. & Stockman, G. (2001). *Computer vision*. Prentice Hall.

E-Commerce

Lecturers: Mr J.A. Lang and others

No. of lectures: 8

Suggested hours of supervision: 2 (example classes if requested)

Prerequisite courses: Business Studies, Security, Economics and Law

Aims

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

Lectures

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

Objectives

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

Recommended reading

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

Additional reading:

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

Hoare Logic and Model Checking

Lecturer: Professor A. Mycroft

No. of lectures: 12

Suggested hours of supervisions: 3

Prerequisite courses: Logic and Proof

Aims

The course introduces two *program logics*, Hoare Logic and Temporal Logic, and uses them to formally specify and verify imperative programs and systems.

One main aim is to introduce Hoare logic for a simple imperative language and then to show how it can be used to formally specify programs (along with discussion of soundness and completeness), and also how to use it in a mechanised program verifier.

The second thrust is to introduce temporal properties, show how these can describe the behaviour of systems, and finally to introduce model-checking algorithms which determine whether properties hold or find counter-examples.

Current research trends also will be outlined.

Lectures

- **Part 1: Formal specification of imperative programs.** Formal versus informal methods. Specification using preconditions and postconditions.
- **Axioms and rules of inference.** Hoare logic for a simple language with assignments, sequences, conditionals and while-loops. Syntax-directedness.
- **Loops and invariants.** Various examples illustrating loop invariants and how they can be found. `FOR`-loops and derived rules. Arrays and aliasing.

- **Partial and total correctness.** Hoare logic for proving termination. Variants.
- **Semantics, metatheory, mechanisation** Mathematical interpretation of Hoare logic. Soundness, completeness and decidability. Assertions, annotation and verification conditions. Weakest preconditions and strongest postconditions; their relationship to Hoare logic and its mechanisation.
- **Additional topics.** Discussion of correct-by-construction methods versus post-hoc verification. Proof of correctness versus property checking. Recent developments in Hoare logic such as separation logic.
- **Part 2: Specifying state transition systems.** Representation of state spaces. Reachable states.
- **Checking reachability properties.** Fixed-point calculations. Symbolic methods using binary decision diagrams. Finding counter-examples.
- **Examples.** Various uses of reachability calculations.
- **Temporal properties and logic**. Linear and branching time. Intervals. Path quantifiers. Brief history. CTL and LTL. PSL for clocked hardware.
- **Model checking.** Simple algorithms for verifying that temporal properties hold. Reachability analysis as a special case.
- **Applications and more recent developments** Simple software and hardware examples. CEGAR (counter-example guided abstraction refinement).

Objectives

At the end of the course students should

- be able to prove simple programs correct by hand and implement a simple program verifier;
- be familiar with the theory and use of Hoare logic and its mechanisation;
- be able to write properties in a variety of temporal logics;
- be familiar with the core ideas of model checking.

Recommended reading

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

Information Retrieval

Lecturer: Dr R. Cummins

No. of lectures: 8

Suggested hours of supervisions: 2

Prerequisite courses: Mathematical Methods for CS (Part IB)

Aims

The course is aimed to characterise information retrieval in terms of the data, problems and concepts involved. It follows the text book “Introduction to Information Retrieval”, cf. below. The main formal retrieval models and evaluation methods are described, with an emphasis on indexing. Web search is also covered. We also consider clustering as an application case of IR.

Lectures

- **Introduction.** (Chapters 1; 2.3) Key problems and concepts. Information need. Boolean Operators.
- **Boolean Retrieval and Indexing.** (Chapters 2.2; 2.4) and Implementation of Boolean Operators. Term manipulations; equivalence classes, stemming.
- **Index representation and Tolerant Retrieval.** (Chapter 3, 4.2-4.4). Index construction. Wildcards. Spelling Correction.
- **The Vector Space Model.** (Chapter 6). VSM and Term weighting.
- **Language Models for Information Retrieval and Classification.** (Chapters 12; 13). Query-likelihood, Smoothing. Naive Bayes Classification.
- **Evaluation.** (Chapter 8, p. 139-148). Test Collections. Relevance. Precision, Recall, MAP, 11pt interpolated average precision.
- **Clustering.** Chapters 16.1-16.4; 17.1-17.2). Proximity metrics, hierarchical vs. partitional clustering. Clustering algorithms.
- **Link Analysis.** (Chapter 21.1, 21.2). PageRank.

Objectives

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

- define the tasks of information retrieval, web search and clustering, and the differences between them;
- understand the main concepts, challenges and strategies used in IR, in particular the retrieval models currently used.

- develop strategies suited for specific retrieval, clustering and classification situations, and recognise the limits of these strategies;
- understand (the reasons for) the evaluation strategies developed for the tasks covered.

Recommended reading

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

Security II

Lecturers: Dr F.M. Stajano and Dr M.G. Kuhn

No. of lectures: 16

Suggested hours of supervisions: 4

Prerequisite courses: Security I; Probability; Economics, Law and Ethics; Operating Systems; Computer Networking

This course is a prerequisite for E-Commerce.

Aims

The first half of this course aims to give students additional understanding of security engineering as a systems discipline, from security policies (modelling what ought to be protected) to mechanisms (how to implement the protection goals). It also covers the interaction of security with psychology and usability; anonymity; security economics, and aspects of physical security. The second half gives and introduction to public-key cryptography, including some mathematical prerequisites and applications.

Lectures

Part 1: Security Engineering [lecturer: Frank Stajano and others]

- **Security, human factors and psychology.** Usability failures. Incompatibility between security requests and work practices. Thinking like an attacker/victim. Social engineering. Phishing. Why do scams work? Social psychology. Decision under risk. Prospect theory as a critique of Expected Utility theory. Framing. [Refs: “Why Johnny can’t encrypt”, “Users are not the enemy”, *The art of deception*, “Understanding scam victims”, *Influence: science and practice*, “The compliance budget”, “Maps of bounded rationality”] [2.5 lectures]
- **Security policies.** Terminology: policy, profile, target. Vaporware policies. Influential security policies: Bell-LaPadula (multi-level security, lattices, covert channels,

downgrading), Biba, Clark-Wilson (double-entry bookkeeping, separation of duties), Resurrecting Duckling (ubiquitous computing, bootstrapping a security association). [1.5 lectures]

- **Passwords.** Usability and security problems of passwords. Taxonomy of replacement schemes and their salient features. Why passwords continue to dominate. [Refs: “The quest to replace passwords”, “Pico: no more passwords”, “The password thicket”].

- **Physical security.** Relevance in systems security context. Pin tumbler locks. Lockpicking. Bumping. “Cryptology and physical security: rights amplification in master-keyed mechanical locks”. Burglar alarms. Sensor defeats; feature interactions; attacks on communications; attacks on trust.

- **Security economics.** Why is security management hard? Misaligned incentives. Asymmetric information. Externalities. Adverse selection. Case studies: security seals, markets for vulnerabilities, phishing website takedown, cost of cybercrime.

- **Anonymity and censorship resistance.** Censorship on the web: goals, technology (DNS tampering, IP blocking etc). Blocking through laws or intimidation. Why privacy and anonymity? Remailers, mix networks, attacks. Censorship resistance tools and their architecture: Tor, Freenet, Psiphon.

Part 2: Cryptography [lecturer: Markus Kuhn]

- **Secure hash functions.** One-way functions, collision resistance, Merkle–Damgård construction, padding, MD5, SHA.

- **Applications of secure hash functions.** HMAC, stream authentication, Merkle tree, commitment protocols.

- **Key distribution problem.** Needham–Schroeder protocol, Kerberos, hardware-security modules, public-key encryption schemes, CPA and CCA security for asymmetric encryption.

- **Number theory.** Modular arithmetic, greatest common divisor, Euclid’s algorithm, modular inversion, groups, rings, fields, finite groups, cyclic groups, generators, Euler’s theorem, Chinese remainder theorem, modular roots, subgroup of quadratic residues, modular exponentiation, easy and difficult problems. [2 lectures]

- **Discrete logarithm problem.** Diffie–Hellman key exchange, ElGamal encryption, hybrid cryptography, elliptic-curve systems.

- **Trapdoor permutations.** Security definition, turning one into a public-key encryption scheme, RSA, attacks on “textbook” RSA, RSA as a trapdoor permutation, optimal asymmetric encryption padding, common factor attacks.

- **Digital signatures.** one-time signatures, ElGamal signatures, DSA, RSA signatures, Certificates, PKI.

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. They also should understand the properties and main applications of secure hash functions, as well as the properties of, and some implementation options for, asymmetric ciphers and signature schemes, based on the discrete-logarithm and RSA problems.

Recommended reading

* Anderson, R. (2008). *Security engineering*. Wiley (2nd ed.). Freely downloadable in PDF from `http://www.cl.cam.ac.uk/users/rja14/book.html`
* Katz, J., Lindell, Y. (2015). *Introduction to modern cryptography*. Chapman & Hall/CRC (2nd ed.).

Further reading:
Gollmann, D. (2010). *Computer security*. Wiley (3rd ed.).
Cialdini, R. (2008). *Influence: science and practice*. Pearson (5th ed.)
Stajano, F. (2002). *Security for ubiquitous computing*. Wiley.
Kahneman, D. (2012). *Thinking fast and slow*. Penguin.

System-on-Chip Design

Lecturer: Dr D.J. Greaves

No. of lectures: 12

Suggested hours of supervisions: 3

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

Aims

A current-day system on a chip (SoC) consists of several different processor subsystems together with memories and I/O interfaces. This course covers SoC design and modelling techniques with emphasis on power consumption and partition of functionality between hardware and software. We study high-level modelling techniques for rapid architectural exploration and assertion-driven design for correctness. We also look at the semantics of hardware description languages and discuss why or whether they should differ from software languages.

This year, two or three lectures will be replaced with practical classes. Watch out for room change notifications. An industrial-strength System-On-Chip demonstrator is available for those who wish to perform in-depth experiments of their own. A highly cut-down version of it can instead be used to learn the key examinable material. The main languages used are Verilog and C++ using the SystemC library, but basic familiarity with assembly language programming is also required.

Lecture Topics (not all may be covered)

- **Verilog RTL design with examples.** Event-driven simulation with and without delta cycles, basic gate synthesis algorithm and design examples. Structural hazards (memories and multipliers) Pipelining and handshake synthesis. [3 lectures]
- **SystemC overview.** The major components of the SystemC C++ class library for hardware modelling are covered with code fragments and demonstrations. Queuing/contention delay modelling. Power, energy and layout high-level modelling. [2 lectures]
- **Basic bus structures.** Bus structure. I/O device structure. Interrupts, DMA and device drivers. Examples. Basic bus bridging.
- **ESL + transactional modelling.** Electronic systems level (ESL) design. Architectural exploration. Firmware modelling methods. Blocking and non-blocking transaction styles. Approximate and loose timing styles. Examples. [2 lectures]
- **ABD: assertions and monitors.** Types of assertion (imperative, safety, liveness, data conservation). Assertion-based design (ABD). PSL/SVA assertions. Temporal logic compilation of fragments to monitoring FSM. [2 lectures]
- **Engineering aspects: FPGA and ASIC design flow.** Cell libraries. Market breakdown: CPU/Commodity/ASIC/FPGA. Further tools used for design of FPGA and ASIC (timing and power modelling, place and route, memory generators, power gating, clock tree, self-test and scan insertion). Dynamic frequency and voltage scaling. [2 lectures]

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 how energy, execution time and silicon area can be traded off;
- understand the hardware and software structures used to implement and model inter-component communication in such devices;
- have basic exposure to SystemC programming and PSL assertions.

Recommended reading

* Keating, M. (2011). *The Simple art of SoC design*. Springer. ISBN 9781441985859.
* OSCI. *SystemC tutorials and whitepapers*. Download from OSCI `http://accellera.org/community/systemc` or copy from course web site.
Ghenassia, F. (2010). *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. (2009). *Modern VLSI design (System-on-chip design)*. Pearson Education (4th ed.).

Topics in Concurrency

Lecturer: Dr. J.M. Hayman

No. of lectures: 12

Suggested hours of supervisions: 3

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.

Easter Term 2016: Part II lectures

Advanced Algorithms

Lecturer: Dr T.M. Sauerwald

No. of lectures: 12

Suggested hours of supervisions: 3

Prerequisite courses: Algorithms

Aims

The aim of this course is to introduce advanced techniques for the design and analysis of algorithms that arise in a variety of applications. A particular focus will be on parallel algorithms and approximation algorithms.

Lectures

- **Sorting networks**. Building blocks of sorting networks. Zero-one principle. Merging network. Bitonic sorter. AKS Network.
- **Load balancing**. Basic approaches including diffusion, first order and second order scheme. Performance measures. Spectral analysis and convergence rate.
- **Matrix multiplication**. Strassen's algorithm. Matrix inversion vs. matrix multiplication. Parallel matrix multiplication. [CLRS3, Chapters 4, 27 and 28]
- **Linear programming**. Definitions and applications. Formulating linear programs. The simplex algorithm. Finding initial solutions. Fundamental theorem of linear programming. [CLRS3, Chapter 29]
- **Approximation algorithms**. (Fully) Polynomial-time approximation schemes. Design techniques. Applications: Vertex cover, Travelling Salesman Problem, parallel machine scheduling. Hardness of approximation. [CLRS3, Chapter 35]
- **Randomised approximation algorithms**. Randomised approximation achemes. Linearity of expectations and randomised rounding of linear programs. Applications: MAX3-CNF problem, weighted vertex cover, MAX-CUT. [CLRS3, Chapter 35]

Objectives

At the end of the course students should

- have an understanding of algorithm design for parallel computers
- be able to formulate, analyse and solve linear programs
- have learned a variety of tools to design efficient (approximation) algorithms

Recommended reading

* Cormen, T.H., Leiserson, C.D., Rivest, R.L. & Stein, C. (2009). *Introduction to Algorithms*. MIT Press (3rd ed.). ISBN 978-0-262-53305-8

Business Studies Seminars

Lecturer: Mr J.A. Lang and others

No. of seminars: 8

Aims

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

Lectures

Eight lectures by eight different entrepreneurs.

Objectives

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

Recommended reading

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

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

Topical Issues

Lecturers: Dr R.K. Harle and others

No. of lectures: 12

Suggested hours of supervisions: 3

Aims

The aim of this course is to broaden the experience of students by exposing them to real-world issues which are of current interest to the computer community. Expert guest lecturers will be used wherever possible to give an external (industrial/commercial) view. The course title changed from “Additional Topics” to “Topical Issues” in 2010–11 for clarity only: the substance of the course remains the same. To remain topical, the exact syllabus is confirmed over the Lent term.

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.
