Computer Science Tripos
Syllabus and Booklist 2019–20

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. Further mathematics should be taken if available, and a physical science (Physics, Chemistry or Geology) is desirable.

Computer Science Tripos Part IA

Part IA students taking the *75% Computer Science option* will attend all lectures for Papers 1, 2 and 3. In addition they attend the Mathematics course offered for Part IA of the Natural Sciences Tripos (NST).

Students taking the *50% Computer Science option* will take one of the following:

Part IA students accepted to read *Computer Science with Mathematics* will attend the lectures for Papers 1 and 2 of the Computer Science Tripos in addition to Papers 1 and 2 of Part IA of the Mathematical Tripos.

Part IA students who take a Natural Science option selected from Chemistry, Evolution and Behaviour, Physics, and Physiology of Organisms will attend Papers 1 and 2 of the Computer Science Tripos as well as the Mathematics course offered for Part IA of the Natural Sciences Tripos (NST).

An A level in a science subject is desirable for students taking an NST option. Computer Science students taking an NST option are expected to undertake practical work on the same basis as for the Natural Science 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 should take the pre-arrival course and attend all the lectures and practicals for Paper 1.

The curriculum

This document lists the courses offered by the Computer Laboratory for Papers 1, 2 and 3 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 2019: Part IA lectures

Paper 1: Foundations of Computer Science

Lecturers: Dr A. Madhavapeddy, Dr A. Prorok

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

Suggested hours of supervisions: 3

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 (OCaml). OCaml is particularly appropriate for inexperienced programmers, since a faulty program cannot crash and OCaml's unobtrusive type system captures many program faults before execution. 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. 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*: 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 OCaml. 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.
- **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 OCaml 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 OCaml combines imperative and functional programming in a single language.

Recommended reading

John Whitington. *OCaml from the Very Beginning*. (http://ocaml-book.com).
Okasaki, C. (1998). *Purely functional data structures*. Cambridge University Press.

Paper 1: Object-Oriented Programming

Lecturer: Dr A.C. Rice

No. of lectures and practicals: 12 + 5

Suggested hours of supervisions: 3

Aims

The goal of this course is to provide students with an understanding of Object-Oriented Programming. Concepts are demonstrated in multiple languages, but the primary language is Java.

Lecture syllabus

- **Types, Objects and Classes** Moving from functional to imperative. Functions, methods. Control flow. values, variables and types. Primitive Types. Classes as custom types. Objects vs Classes. Class definition, constructors. Static data and methods.
- **Designing Classes** Identifying classes. UML class diagrams. Modularity. Encapsulation/data hiding. Immutability. Access modifiers. Parameterised types (Generics).
- **Pointers, References and Memory** Pointers and references. Reference types in Java. The call stack. The heap. Iteration and recursion. Pass-by-value and pass-by-reference.
- **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: reference counting, tracing.
- **Java Collections and Object Comparison** Java Collection interface. Key classes. Collections class. Iteration options and the use of Iterator. Comparing primitives and objects. Operator overloading.
- **Error Handling** Types of errors. Limitations of return values. Deferred error handling. Exceptions. Custom exceptions. Checked vs unchecked. Inappropriate use of exceptions. Assertions.
- **Copying Objects**. Shallow and deep copies. Copy constructors. Cloning in Java. Cloneable as a marker interface in Java.
- **DesignLanguage evolution** Need for languages to evolve. Generics in Java. Type erasure. Introduction to Java 8: Lambda functions, functions as values, method references, streams.

- **Design Patterns** Introduction to design patterns. Open-closed principle. Examples of Singleton, Decorator, State, Composite, Strategy, Observer. [2 lectures]

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 students.

Lecturer: Dr I.J. Wassell

No. of lectures and practical classes: 12 + 7

Suggested hours of supervisions: 3

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. Quine-McCluskey method.
- **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. Elimination of redundant states.
- **Further state machines.** State assignment: sequential, sliding, shift register, one hot. Implementation of FSMs.
- **Introduction to microprocessors.** Microarchitecture, fetch, register access, memory access, branching, execution time.

- **Electronics, Devices and Circuits.** Current and voltage, resistance, basic circuit theory, the potential divider. Solving non-linear circuits. N and p channel MOSFETs and n-MOSFET logic, e.g., n-MOSFET inverter. Switching speed and power consumption problems in n-MOSFET logic. CMOS logic. Logic families. Noise margin. Analogue interfacing and operational amplifiers. [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.
- understand basic analogue interfacing.

Recommended reading

* Harris, D.M. & Harris, S.L. (2013). *Digital design and computer architecture*. Morgan Kaufmann (2nd ed.). The first edition is still relevant.
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

This course is not taken by NST students.

Lecturers: Professor M. Fiore and Professor F. Stajano

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

Suggested hours of supervisions: 6

This course is a prerequisite for all theory courses as well as: Probability, Security, Artificial Intelligence, Compiler Construction and the following Part II courses: Machine Learning and Bayesian Inference and Cryptography

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 [9 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 [5 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.
Hammack, R. (2013). *Book of proof.* Privately published (Second edition). Available at: `http://www.people.vcu.edu/ rhammack/BookOfProof/index.html`
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).

Paper 3: Databases

This course is only taken by Part IA and Part IB Paper 3 students

Lecturer: Dr T.G. Griffin

No. of lectures and practical classes: 8 + 4

Suggested hours of supervisions: 3

Prerequisite courses: None

Aims

This course introduces basic concepts for database systems as seen from the perspective of application designers. That is, the focus is on the abstractions supported by database management systems and not on how those abstractions are implemented.

The database world is currently undergoing swift and dramatic transformations largely driven by Internet-oriented applications and services. Today many more options are available to database application developers than in the past and so it is becoming increasingly difficult to sort fact from fiction. The course attempts to cut through the fog with a practical approach that emphasises engineering tradeoffs that underpin these recent developments and also guide our selection of "the right tool for the job."

This course covers three approaches. First, the traditional mainstay of the database industry — the relational approach — is described with emphasis on eliminating logical redundancy in data. Then two representatives of recent trends are presented — graph-oriented and document-oriented databases. The lectures are tightly integrated with the associated practical sessions where students gain hands-on experience with all three of these approaches.

Lectures

- **Introduction.** What is a database system? What is a data model? A central tradeoff in the choice of data representation: optimise for ease of updating or for fast query response. On-Line Transaction Processing (OLTP) *versus* On-line Analytical Processing (OLAP). Application independent *versus* application specific data representations. [1 lecture]
- **Conceptual modeling** The Entity-Relationship (ER) approach as an implementation-independent technique for modeling data. [1 lecture]
- **The relational model** Implementing ER models with relational tables. Relational algebra and SQL. Update anomalies caused by logical redundancy. Minimise logical redundancy with normalised data representation. What is transitive closure? Why SQL struggles with transitive closure. [2 lectures]
- **The graph-oriented model** The NoSQL movement. Implementing ER models in a graph-oriented database. Graph databases: optimised for computing transitive closure. Path-oriented queries. [2 lectures]
- **The document-oriented model** Semi-structured data (XML, JSON). Document-oriented databases. Embracing data redundancy: representing data for fast, application-specific, access. [1 lecture]
- **The multi-dimensional model**. Data cubes, star schema, data warehouse. [1 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- and graph-oriented implementations
- understand the fundamental tradeoff between the ease of updating data and the response time of complex queries
- understand that no single data architecture can be used to meet all data management requirements
- be familiar with recent trends in the database area.

Recommended reading

Lemahieu, W., Broucke, S. van den & Baesens, B. (2018) *Principles of database management*. Cambridge University Press.

Paper 3: Introduction to Graphics

This course is only taken by Part IA and Part IB Paper 3 students.

Lecturer: Dr R.K. Mantiuk

No. of lectures and practical classes: 8 + 4

Suggested hours of supervisions: 2

Prerequisite courses: None

This course is a prerequisite for Further Graphics

Aims

To introduce the necessary background, the basic algorithms, and the applications of computer graphics and graphics hardware.

Lectures

- **Background.** What is an image? Resolution and quantisation. Storage of images in memory. [1 lecture]
- **Rendering.** Perspective. Reflection of light from surfaces and shading. Geometric models. Ray tracing. [2 lectures]
- **Graphics pipeline.** Polygonal mesh models. Transformations using matrices in 2D and 3D. Homogeneous coordinates. Projection: orthographic and perspective. [1 lecture]

- **Graphics hardware and modern OpenGL.** GPU rendering. GPU frameworks and APIs. Vertex processing. Rasterisation. Fragment processing. Working with meshes and textures. Z-buffer. Double-buffering and frame synchronization. [3 lectures]
- **Colour**. Perception of colour. Colour spaces. [1 lecture]

Objectives

By the end of the course students should be able to:

- understand and apply in practice basic concepts of ray-tracing: ray-object intersection, reflections, refraction, shadow rays, distributed ray-tracing, direct and indirect illumination;
- describe and explain the following algorithms: Gouraud and Phong shading, z-buffer, texture mapping, double buffering, mip-map, bump- and normal-mapping;
- 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;
- implement OpenGL code for rendering of polygonal objects, control camera and lighting, work with vertex and fragment shaders;
- describe a number of colour spaces and their relative merits.

Recommended reading

* Shirley, P. & Marschner, S. (2009). *Fundamentals of Computer Graphics*. CRC Press (3rd ed.).
Foley, J.D., van Dam, A., Feiner, S.K. & Hughes, J.F. (1990). *Computer graphics: principles and practice*. Addison-Wesley (2nd ed.).
Kessenich, J.M., Sellers, G. and Shreiner, D (2016). *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 4.5 with SPIR-V*, [seventh edition and later]

Scientific Computing Practical Course

This is a practical course taken by Part IA CST students only. It is taught through one introductory lecture, followed by an online course which is equivalent in content to approximately 5 lectures. Students will normally work though the online component at their own pace. The course will be entirely examined through practical exercises.

Lecturer: Dr D. Wischik

No. of lectures: 1 (plus an online course with roughly 5 lectures worth of material in the Lent term)

Suggested hours of supervisions: none

Prerequisite courses: Foundations of Computer Science, NST Mathematics

This course is a prerequisite for Foundations of Data Science (Part IB)

Aims

This course is a hands-on introduction to using computers to investigate scientific models and data.

Syllabus

- Python notebooks. Overview of the Python programming language. Use of notebooks for scientific computing.
- Numerical computation. Writing fast vectorized code in `numpy`. Optimization and fitting. Simulation.
- Working with data. Data import. Common ways to summarize and plot data, for univariate and multivariate analysis.

Objectives

At the end of the course students should

- be able to import data, plot it, and summarize it appropriately
- be able to write fast vectorized code for scientific / data work

---

Lent Term 2020: Part IA lectures

Paper 1: Algorithms

Lecturers: Professor F. Stajano and Dr D. Wischik

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

Suggested hours of supervisions: 6

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

This course is a prerequisite for: Artificial Intelligence, Complexity Theory, Further Graphics, Prolog and the following Part II courses: Advanced Algorithms and Machine Learning and Bayesian Inference

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.** Elementary data structures: pointers, objects, 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]
- **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]
- **Advanced data structures.** Binomial heap. Amortized analysis: aggregate analysis, potential method. Fibonacci heaps. Disjoint sets. [Ref: CLRS3 chapters 17, 19, 20, 21] [about 4 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.

Paper 2: Operating Systems

This course is not taken by NST students.

Lecturer: Dr. R. Mortier

No. of lectures: 12

Suggested hours of supervisions: 3

Prerequisite courses: Pre-arrival course, Digital Electronics

This course is a prerequisite for Concurrent & Distributed Systems (Part IB), Security 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. Elementary computer architecture. OS evolution: multi-programming, time-sharing. [1 lecture]

- **Protection.** Dual-mode operation. Protecting I/O, memory, CPU. Kernels and micro-kernels. Virtual machines and containers. Subjects and objects. Authentication. Access matrix: ACLs and capabilities. Combined scheme. Covert channels. [1 lecture]

- **Processes.** Job/process concepts. Lifecycle. Process management. Inter-process communication. [1 lecture]

- **Scheduling.** 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]

- **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 various mechanisms involved in memory management, 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.).
Anderson, T. & Dahlin, M. (2014). *Operating Systems: Principles & Practice*. Recursive Books (2nd 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 ed.).

Paper 3: Machine Learning and Real-world Data

This course is only taken by Part IA and Part IB Paper 3 students.

Lecturers: Professor S.H. Teufel, Dr A. Vlachos and Dr R. Cotterell

No. of lectures and practical classes: 16

Suggested hours of supervisions: 4

Prerequisite courses: NST Mathematics

Aims

This course introduces students to machine learning algorithms as used in real-world applications, and to the experimental methodology necessary to perform statistical analysis of large-scale data from unpredictable processes. Students will perform 3 extended practicals, as follows:

- Statistical classification: Determining movie review sentiment using Naive Bayes (7 sessions);
- Sequence Analysis: Hidden Markov Modelling and its application to a task from biology (predicting protein interactions with a cell membrane) (4 sessions);
- Analysis of social networks, including detection of cliques and central nodes (5 sessions).

Syllabus

- **Topic One: Statistical Classification [7 sessions].**
  Introduction to sentiment classification.
  Naive Bayes parameter estimation.
  Statistical laws of language.
  Statistical tests for classification tasks.
  Cross-validation and test sets.
  Uncertainty and human agreement.

- **Topic Two: Sequence Analysis [4 sessions].**
  Hidden Markov Models (HMM) and HMM training.
  The Viterbi algorithm.
  Using an HMM in a biological application.
- **Topic Three: Social Networks [5 sessions].**
  Properties of networks: Degree, Diameter.
  Betweenness Centrality.
  Clustering using betweenness centrality.

Objectives

By the end of the course students should be able to:

- understand and program two simple supervised machine learning algorithms;
- use these algorithms in statistically valid experiments, including the design of baselines, evaluation metrics, statistical testing of results, and provision against overtraining;
- visualise the connectivity and centrality in large networks;
- use clustering (i.e., a type of unsupervised machine learning) for detection of cliques in unstructured networks.

Recommended reading

Jurafsky, D. & Martin, J. (2008). *Speech and language processing*. Prentice Hall.
Easley, D. and Kleinberg, J. (2010). *Networks, crowds, and markets: reasoning about a highly connected world*. Cambridge University Press.

Easter Term 2020: Part IA lectures

Paper 1: Introduction to Probability

Lecturer: Prof. M. Jamnik, Dr T. Sauerwald

No. of lectures: 12

Suggested hours of supervisions: 3

This course is a prerequisite for Probability and Computation (Part II).

Aims

This course provides an elementary introduction to probability and statistics with applications. Probability theory and the related field of statistical inference provide the foundations for analysing and making sense of data. The focus of this course is to introduce the language and core concepts of probability theory. The course will also cover some applications of statistical inference and algorithms in order to equip students with the ability to represent problems using these concepts and analyse the data within these principles.

Lectures

Part 1 - Introduction to Probability

- **Introduction.** Counting/Combinatorics (revision), Probability Space, Axioms, Union Bound.
- **Conditional probability.** Conditional Probabilities and Independence, Bayes'Theorem, Partition Theorem

Part 2 - Discrete Random Variables

- **Random variables.** Definition of a Random Variable, Probability Mass Function, Cumulative Distribution, Expectation.
- **Probability distributions.** Definition and Properties of Expectation, Variance, different ways of computing them, Examples of important Distributions (Bernoulli, Binomial, Geometric, Poisson), Primer on Continuous Distributions including Normal and Exponential Distributions.
- **Multivariate distributions.** Multiple Random Variables, Joint and Marginal Distributions, Independence of Random Variables, Covariance.

Part 3 - Moments and Limit Theorems

- **Introduction.** Law of Average, Useful inequalities (Markov and Chebyshef), Weak Law of Large Numbers (including Proof using Chebyshef's inequality), Examples.

- **Moments and Central Limit Theorem.** Introduction to Moments of Random Variables, Central Limit Theorem (Proof using Moment Generating functions), Example.

Part 4 - Applications/Statistics

- **Statistics.** Classical Parameter Estimation (Maximum-Likelihood-Estimation), bias, sample mean, sample variance), Examples (Collision-Sampling, Estimating Population Size).
- **Algorithms.** Online Algorithms (Secretary Problem, Odd's Algorithm).

Objectives

At the end of the course students should

- understand the basic principles of probability spaces and random variables
- be able to formulate problems using concepts from probability theory and compute or estimate probabilities
- be familiar with more advanced concepts such as moments, limit theorems and applications such as parameter estimation

Recommended reading

* Ross, S.M. (2014). *A First course in probability*. Pearson (9th ed.).
Bertsekas, D.P. & Tsitsiklis, J.N. (2008). *Introduction to probability*. Athena Scientific.
Grimmett, G. & Welsh, D. (2014). *Probability: an Introduction*. Oxford University Press (2nd ed.).
Dekking, F.M., et. al. (2005) *A modern introduction to probability and statistics*. Springer.

Paper 2: Software and Security Engineering

Lecturer: Professor R. Anderson

No. of lectures: 11

Suggested hours of supervisions: 3

This course is a prerequisite for the Group Project.

Aims

This course aims to introduce students to software and security engineering, and in particular to the problems of building large systems, safety-critical systems and systems that must withstand attack by capable opponents. Case histories of failure are used to illustrate what can go wrong, while current software and security engineering practice is studied as a guide to how failures can be avoided.

Lectures

- **What is a security policy or a safety case?** Definitions and examples; one-way flows for both confidentiality and safety properties; separation of duties. Top-down and bottom-up analysis methods. What architecture can do, versus benefits of decoupling policy from mechanism.
- **Examples of safety and security policies.** Safety and security usability; the pyramid of harms. Predicting and mitigating user errors. The prevention of fraud and error in accounting systems; the safety usability of medical devices.
- **Attitudes to risk:** expected utility, prospect theory, framing, status quo bias. Authority, conformity and gender; mental models, affordances and defaults. The characteristics of human memory; forgetting passwords versus guessing them.
- **Security protocols;** how to enforce policy using structured human interaction, cryptography or both. Middleperson attacks. The role of verification and its limitations.
- **Attacks on TLS,** from rogue CAs through side channels to Heartbleed. Other types of software bugs: syntactic, timing, concurrency, code injection, buffer overflows. Defensive programming: secure coding, contracts. Fuzzing.
- **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 complex software.
- **Software engineering as the management of complexity.** The software life cycle; requirements analysis methods; modular design; the role of prototyping; the waterfall, spiral and agile models.
- **The economics of software as a Service (SaaS);** the impact SaaS has on software engineering. Continuous integration, release engineering, behavioural analytics and experiment frameworks, rearchitecting systems while in operation.
- **Critical systems:** safety as an emergent system property. Examples of catastrophic failure: from Therac-25 to the Boeing 737Max. The problems of managing redundancy. The overall process of safety engineering.
- **Managing the development of critical systems:** tools and methods, individual versus group productivity, economics of testing and agile development, measuring outcomes versus process, the technical and human aspects of management, post-market surveillance and coordinated disclosure. The sustainability of products with software components.

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 understand the different models of software development described in the course as well as the value of various development and management tools. They

should understand the development life cycle and its basic economics. They should understand the various types of bugs, vulnerabilities and hazards, how to find them, and how to avoid introducing them. Finally, they should be prepared for the organizational aspects of their Part IB group project.

Recommended reading

Anderson, R. (2020). *Security engineering* (Part 1 and Chapters 27-28). Wiley. Available at: `http://www.cl.cam.ac.uk/users/rja14/book.html`

Paper 3: Interaction Design

This course is only taken by Part IA and Part IB Paper 3 students.

Lecturer: Dr H. Gunes

No. of lectures and practical classes: 8 + 8

Suggested hours of supervisions: 2

Prerequisite courses: Java

This course is a prerequisite for Human-Computer Interaction (Part II)

Aims

The aim of this course is to provide an introduction to interaction design, with an emphasis on understanding and experiencing the user interface design process from requirements and data gathering to implementation and evaluation, while gaining an understanding of the background to human factors. This course focuses equally on design and implementation.

Lectures

- **Course overview and requirements analysis.** Introduction to the course and the practicals. Identifying potential users and understanding their tasks. Identifying and establishing non-functional and functional requirements.
- **Data gathering.** Data collection and quantitative/qualitative analysis techniques.
- **Design and prototyping.** Participatory design process. Conceptual versus physical design. Concept development. Prototyping and different kinds of prototypes. Personas and storyboards.
- **Case studies from the industry.** Guest lecture (the schedule of this lecture is subject to change).
- **Cognitive aspects.** Attention, perception/recognition, memory, context and grouping, and their implications for interaction design. Cognitive frameworks.

- **Evaluation.** Introduction, evaluation techniques, and an evaluation case study.
- **Student (group) presentations**

Objectives

By the end of the course students should

- have a thorough understanding of the iterative design process and be able to apply it to interaction design;
- be able to design new user interfaces that are informed by principles of human visual perception and cognition;
- be able to construct user interfaces using Java with a strong emphasis on users, usability and appearance;
- be able to evaluate existing or new user interfaces using multiple techniques;
- be able to compare and contrast different design techniques and to critique their applicability to new domains.

Recommended reading

* Preece, J., Rogers, Y. & Sharp, H. (2015). *Interaction design*. Wiley (4th ed.).

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.

Some courses are specific to either Paper 3 or Paper 7, and have been marked as such in this booklet. Those students following the *75% Computer Science option*, who have taken Paper 3 in Part IA, will attend lectures for the Paper 7 courses. Those students taking the *50% Computer Science option*, will attend the Paper 3 courses instead. The remaining courses are taken by all Part IB students.

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 2019: Part IB lectures

Computer Design

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

No. of lectures: 16

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 course Comparative Architectures.

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.

Lectures

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

- **Technology trends and design challenges.** Current technology, technology trends, ECAD trends, challenges. [1 lecture]
- **Digital system design.** Practicalities of mapping SystemVerilog descriptions of hardware (including a processor) onto an FPGA board. Tips and pitfalls when generating larger modular designs. [1 lecture]
- **Eight great ideas in computer architecture.** [1 lecture]
- **Reduced instruction set computers and RISC-V.** Introduction to the RISC-V processor design. [1 lecture]
- **Executable and synthesisable models.** [1 lecture]
- **Pipelining.** [2 lectures]
- **Memory hierarchy and caching.** Caching, etc. [1 lecture]
- **Support for operating systems.** Memory protection, exceptions, interrupts, etc. [1 lecture]
- **Other instruction set architectures.** CISC, stack, accumulator. [1 lecture]

Part 2 - Lecturer Dr Timothy Jones

- **Overview of Systems-on-Chip (SoCs) and DRAM.** [1 lecture] High-level SoCs, DRAM storage and accessing.
- **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-chips and their components;
- understand how DRAM stores data;
- understand how multicore processors communicate;
- understand how GPUs work and have an appreciation of how to program them.

Recommended reading

* Patterson, D.A. & Hennessy, J.L. (2017). *Computer organization and design: The hardware/software interface RISC-V edition*. Morgan Kaufmann.
ISBN 978-0-12-812275-4.

Recommended further reading:

Harris, D.M. & Harris, S.L. (2012). *Digital design and computer architecture*. Morgan Kaufmann. ISBN 978-0-12-394424-5.
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.)
Pointers to sources of more specialist information are included in the lecture notes and on the associated course web page.

Concurrent and Distributed Systems

Lecturer: Dr D. Greaves and Dr M. Kleppmann

No. of lectures: 16

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

This course considers two closely related topics, Concurrent Systems and Distributed Systems, over 16 lectures. The aim of the first half of the course is to introduce concurrency control concepts and their implications for system design and implementation. The aims of the latter half of the course are to study the fundamental characteristics of distributed systems, including their models and architectures; the implications for software design; some of the techniques that have been used to build them; and the resulting details of good distributed algorithms and applications.

Lectures: 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).

- **Automata Composition** Synchronous and asynchronous parallelism; sequential consistency; rendezvous. Safety, liveness and deadlock; the Dining Philosophers; Hardware foundations for atomicity: test-and-set, load-linked/store-conditional and fence instructions.

- **Common design patterns: semaphores, producer-consumer, and MRSW** 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).

- **Deadlock and liveness guarantees** Offline vs. online; model checking; resource allocation graphs; lock order checking; 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. Hardware and software transactional memories.

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 multicast.

- **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 Concurrent Systems portion 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.

At the end of the Distributed Systems portion 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.
Kleppmann, M. (2017). *Designing data-intensive applications*. O'Reilly.
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. T. Markettos, Professor S.W. Moore & Dr R. Mullins

No. of practical classes: 8

Prerequisite course: Digital Electronics

Companion course: Computer Design

This course is a prerequisite for the Part II course Comparative Architectures.

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** The first class uses a web based tutor to rapidly teach the SystemVerilog language.
- **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.

Paper 7: Economics, Law and Ethics

This course is only taken by Part IB and Part II Paper 7 students.

Lecturers: Dr A. Hutchings

No. of lectures: 8

Suggested hours of supervisions: 2

This course is a prerequisite for the Part II courses 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.
- **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`
Thaler, R.H. (2016). *Misbehaving*. Penguin.
Galbraith, J.K. (1991). *A history of economics*. Penguin.
Poundstone, W. (1992). *Prisoner's dilemma*. Anchor Books.
Pinker, S (2011). *The Better Angels of our Nature*. Penguin.
Anderson, R. (2008). *Security engineering* (Chapter 7). Wiley.
Nuffield Council on Bioethics (2015) *The collection, linking and use of data in biomedical research and health care*.

Foundations of Data Science

Lecturer: Dr D.J. Wischik

No. of lectures and practical classes: 12 + 4

Suggested hours of supervisions: 3

Prerequisite courses: either Mathematics for Natural Sciences, or the equivalent from the Maths Tripos

This course is a prerequisite for: Part II Machine Learning and Bayesian Inference, Information Retrieval, Quantum Computing, Natural Language Processing.

Aims

This course introduces fundamental tools for describing and reasoning about data. There are two themes: describing the behaviour of random systems; and making inferences based on data generated by such systems. The course will survey a wide range of models and tools, and it will emphasize how to design a model and what sorts of questions one might ask about it.

Lectures

- **Likelihood.** Random variables. Random samples. Maximum likelihood estimation, likelihood profile.
- **Random variables.** Rules for expectation and variance. Generating random variables. Empirical distribution. Monte Carlo estimation; law of large numbers. Central limit theorem.
- **Inference.** Estimation, confidence intervals, hypothesis testing, prediction. Bootstrap. Bayesianism. Logistic regression, natural parameters.
- **Feature spaces.** Vector spaces, bases, inner products, projection. Model fitting as projection. Linear modeling. Choice of features.
- **Random processes.** Markov chains. Stationarity and convergence. Drift models. Examples, including estimation and memory.
- **Probabilistic modelling.** Independence; joint distributions. Descriptive, discriminative, and causal models. Latent variable models. Random fields.

Objectives

At the end of the course students should

- be able to formulate basic probabilistic models, including discrete time Markov chains and linear models
- be familiar with common random variables and their uses, and with the use of empirical distributions rather than formulae
- be able to use expectation and conditional expectation, limit theorems, equilibrium distributions
- understand different types of inference about noisy data, including model fitting, hypothesis testing, and making predictions
- understand the fundamental properties of inner product spaces and orthonormal systems, and their application to model representation

Recommended reading

* F.M. Dekking, C. Kraaikamp, H.P. Lopuhaä, L.E. Meester (2005). *A modern introduction to probability and statistics: understanding why and how.* Springer.

S.M. Ross (2002). *Probability models for computer science.* Harcourt / Academic Press.

M. Mitzenmacher & E. Upfal (2005). *Probability and computing: randomized algorithms and probabilistic analysis.* Cambridge University Press.

Paper 7: Further Graphics

This course is only taken by Part IB and Part II Paper 7 students.

Lecturer: Dr A. Benton

No. of lectures: 8

Suggested hours of supervisions: 2

Prerequisite courses: Introduction to Graphics

This course is a prerequisite for: Advanced Graphics

Aims

This course explores the modern state of computer graphics, applying long-standing techniques to cutting-edge hardware.

Lectures

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

- **Ray Marching.** Signed distance fields and GPU-based realtime rendering. [1 lecture]
- **Implicit surfaces.** Organic and soft surface modelling. [1 lecture]
- **Computational Geometry.** Mathematics of surfaces. [1 lecture]
- **Bezier curves and NURBS.** These points of data make a beautiful line. [2 lectures]
- **Subdivision Surfaces.** Smooth modeling of continuous surfaces. [1 lecture]
- **Global Illumination.** Realistic global lighting techniques. [1 lecture]
- **Virtual Reality.** Technology and best practices for an emerging medium. [1 lecture]

Objectives

On completing the course, students should be able to

- use graphics hardware to render interactive images, both polygonal and implicit;
- understand the core technologies of ray tracing, rendering, and implicit surfaces;
- learn techniques of computational geometry and their applications to visualization;
- describe the underlying theory of splines and subdivision and define the Catmull-Clark and Doo-Sabin subdivision methods;
- understand several global illumination technologies such as radiosity and ambient occlusion;

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.).

Watt, A. (2000). *3D Computer Graphics*. Addison-Wesley (3rd ed).

Hughes, van Dam, McGuire, Skalar, Foley, Feiner & Akeley (2013). *Computer Graphics: Principles & Practice*. Addison-Wesley (3rd edition)

Akenine-Möller, et. al. (2018). *Real-time rendering*. CRC Press (4th ed.).

Further Java

Lecturer: Dr A.R. Beresford and Dr A. Rice

No. of practical classes: 5 x 2-*hour sessions* (for which supervisions should be given)

Suggested hours of supervisions: 2

Prerequisite course: Object-Oriented Programming

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 an integrated 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.

- **Vector clocks**. This week students will use the concept of vector clocks to make their client and server robust to message delays and reordering.

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;

- be able to implement a vector clock algorithm and the happens-before relation.

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: Dr R. Mortier and Dr R. Harle

No. of lectures: 1

Prerequisite courses: Software and Security 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.

Programming in C and C++

Lecturer: Dr D. Greaves and Professor A. Mycroft

No. of lectures: 12

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 to provide an overview of the principles and constraints that affect the way in which the C programming language has been designed and is used, including the differences betweeen it and C++.

Lectures

- **Introduction to the C language.** Background and goals of C. Types and variables. Expressions and statements. Functions. Multiple compilation units. Tooling for C programming. [2 lectures]
- **Further C concepts.** Preprocessor. Pointers and pointer arithmetic. Data structures. Dynamic memory management. Examples. [2 lectures]
- **Memory Management** Unique ownership. Object graphs and graph traversals. Aliasing and deallocation. Mark and sweep algorithms. Reference counting. Arenas. Stack allocation. Handles and compaction. [3 lectures]
- **Memory Hierarchy and Cache Optimization** Cache hierarchy. Data structure layouts. Intrusive lists. Array-of-structs vs struct-of-array representations. [1 lecture]
- **Linkers, loaders and debugging.** 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]
- **Introduction to C++.** Goals of C++. Differences between C and C++. References versus pointers. Overloading functions. [1 lecture]
- **Objects in C++** Classes and structs. Exceptions. Destructors. Operator overloading. Virtual functions. Casting. Multiple inheritance. Virtual base classes. Templates and meta-programming. [1 lecture]

Objectives

At the end of the course students should

- be able to read and write C programs;
- understand the interaction between C programs and the host operating system;
- be familiar with the structure of C program execution in machine memory;
- understand the potential dangers of writing programs in C;
- understand the main differences between C and C++.

Recommended reading

* Kernighan, B.W. & Ritchie, D.M. (1988). *The C programming language*. Prentice Hall (2nd ed.).

Semantics of Programming Languages

Lecturer: Dr N. Krishnaswami

No. of lectures: 12

Suggested hours of supervisions: 3

This course is a prerequisite for the Part II courses Topics in Concurrency, Hoare Logic and Model Checking 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.

Unix Tools

Lecturer: Dr M.G. Kuhn

No. of lectures: 8

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

Prerequisite courses: Operating Systems.

This course is a prerequisite for Security.

Aims

This video-lecture 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.
- **Version control.** diff, patch, RCS, Subversion, git.
- **Software development tools.** C compiler, linker, debugger, make.
- **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 a version-control system 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 2020: 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. We use a running example called SLANG (a Small LANGuage) inspired by the languages described in 1B Semantics. A toy compiler (written in ML) is provided, and students are encouraged to extend it in various ways.

Lectures

- **Overview of compiler structure** The spectrum of interpreters and compilers; compile-time and run-time. Compilation as a sequence of translations from *higher-level* to *lower-level* intermediate languages, where each translation preserves semantics. The structure of a simple compiler: lexical analysis and syntax analysis, type checking, intermediate representations, optimisations, code generation. Overview of run-time data structures: stack and heap. Virtual machines. [1 lecture]

- **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. [3 lectures]

- **Compiler Correctness** Recursive functions can be transformed into iterative functions using the Continuation-Passing Style (CPS) transformation. CPS applied to a (recursive) SLANG interpreter to derive, in a step-by-step manner, a correct stack-based compiler. [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. [5 lectures]

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

Computer Networking

Lecturer: Dr A.W. Moore

No. of lectures: 20

Suggested hours of supervisions: 5

This course is a prerequisite for the Part II course Principles of Communication

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-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. 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. [2 lectures]
- **Wireless and mobile networks.** Wireless links and network characteristics, Wi-Fi: IEEE 802.11 wireless LANs. [1 lecture]
- **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, 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, TCP variants. [6 lectures]
- **Application layer.** Client/server paradigm, WWW, HTTP, Domain Name System, P2P. [1.5 lectures]
- **Multimedia networking.** Networked multimedia applications, multimedia delivery requirements, multimedia protocols (SIP), content distribution networks. [0.5 lecture]

Objectives

At the end of the course students should

- be able to analyse a communication system by separating out the different functions provided by the network;
- understand that there are fundamental limits to any communications system;
- understand the general principles behind multiplexing, addressing, routing, reliable transmission and other stateful protocols as well as specific examples of each;
- understand what FEC is;
- 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.).

Paper 7: Further Human–Computer Interaction

This course is only taken by Part IB and Part II Paper 7 students.

Lecturer: Dr L. Church

No. of lectures: 8

Suggested hours of supervisions: 2

Prerequisite courses: Interaction Design (Part IA)

Aims

This aim of this course is to provide an introduction to the theoretical foundations of Human Computer Interaction, and an understanding of how these can be applied to the design of complex technologies.

Lectures

- **Theory driven approaches to HCI.** What is a theory in HCI? Why take a theory driven approach to HCI?
- **Design of visual displays.** Segmentation and variables of the display plane. Modes of correspondence.
- **Goal-oriented interaction.** Using cognitive theories of planning, learning and understanding to understand user behaviour, and what they find hard.
- **Designing smart systems.** Using statistical methods to anticipate user needs and actions with Bayesian strategies.
- **Designing efficient systems.** Measuring and optimising human performance through quantitative experimental methods.
- **Designing meaningful systems.** Qualitative research methods to understand social context and requirements of user experience.
- **Evaluating interactive system designs.** Approaches to evaluation in systems research and engineering, including Part II Projects.
- **Designing complex systems.** Worked case studies of applying the theories to a hard HCI problem. Research directions in HCI.

Objectives

At the end of the course students should be able to apply theories of human performance and cognition to system design, including selection of appropriate techniques to analyse, observe and improve the usability of a wide range of technologies.

Recommended reading

* Preece, J., Sharp, H. & Rogers, Y. (2015). *Interaction design: beyond human–computer interaction*. Wiley (Currently in 4th edition, but earlier editions will suffice).

Further reading:

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

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 Machine Learning and Bayesian Inference, Hoare Logic & Model Checking 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.).

Paper 7: Prolog

This course is only taken by Part IB and Part II Paper 7 students.

Lecturer: Dr. A. Rice and Dr I. Lewis

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. 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.]

Easter Term 2020: Part IB lectures

Artificial Intelligence

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 Machine Learning and Real World Data, Foundations of Data Science, Logic and Proof, Prolog and Complexity Theory are likely to be useful.

This course is a prerequisite for the Part II courses Machine Learning and Bayesian Inference 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 lecture]
- **Knowledge representation and reasoning II.** Knowledge representation and reasoning using first order logic. The frame, qualification and ramification problems. The situation calculus. [1 lecture]
- **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. (2017). *Artificial intelligence: foundations of computational agents*. Cambridge University Press (2nd ed.).

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.
Brachman, R.J & Levesque, H.J. (2004). *Knowledge Representation and Reasoning*. Morgan Kaufmann.

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. Problems on numbers, graphs and formulas.
- **Non-determinism.** Non-deterministic machines. The complexity class NP and its various characterizations. Non-deterministic algorithms for satisfiability and other problems in NP.
- **NP-completeness.** Reductions and completeness. NP-completeness of satisfiability.
- **More NP-complete problems.** Graph-theoretic problems. Independent set, clique and 3-colourability.

- **More NP-complete problems.** Sets, numbers and scheduling. Matching, set covering and knapsack.
- **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.

Paper 7: Concepts in Programming Languages

This course is only taken by Part IB and Part II Paper 7 students.

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 forms of 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. JavaScript prototypal vs Java class-based inheritance.
- **Languages for parallel processing.** Shared-memory concurrency with spawn/sync (OpenMP, Cilk, X10). Distributed-memory models (the actor model, Erlang). External vs. internal iteration.
- **Types.** Types in programming languages. Type safety. Type systems—static vs. dynamic. 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.
- **Combining functional and object-oriented features.** Scala and Java 8. Generic types and methods. Variance annotations. The expression problem. Value types and deep copy.
- **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*.

Paper 7: Formal Models of Language

This course is only taken by Part IB and Part II Paper 7 students.

Lecturers: Professor P. Buttery

No. of lectures: 8

Suggested hours of supervisions: 2

Prerequisite courses: Discrete Maths; Compiler Construction;

Aims

This course studies formal models of language and considers how they might be relevant to the processing and acquisition of natural languages. The course will extend knowledge of formal language theory; introduce several new grammars; and use concepts from information theory to describe natural language.

Lectures

- **Natural language and the Chomsky hierarchy 1.** Recap classes of language. Closure properties of language classes. Recap pumping lemma for regular languages. Discussion of relevance (or not) to natural languages (example embedded clauses in English).
- **Natural language and the Chomsky hierarchy 2.** Pumping lemma for context free languages. Discussion of relevance (or not) to natural languages (example Swiss-German cross serial dependancies). Properties of minimally context sensitive languages. Introduction to tree adjoining grammars.
- **Language processing and context free grammar parsing 1.** Recap of context free grammar parsing. Language processing predictions based on top down parsing models (example Yngve's language processing predictions). Language processing predictions based on probabilistic parsing (example Halle's language processing predictions).
- **Language processing and context free grammar parsing 2.** Introduction to context free grammar equivalent dependancy grammars. Language processing predictions based on Shift-Reduce parsing (examples prosodic look-ahead parsers, Parsey McParseface).
- **Grammar induction of language classes.** Introduction to grammar induction. Discussion of relevance (or not) to natural language acquisition. Gold's theorem. Introduction to context free grammar equivalent categorial grammars and their learnable classes.
- **Natural language and information theory 1.** Entropy and natural language typology. Uniform information density as a predictor for language processing.
- **Natural language and information theory 2.** Noisy channel encoding as a model for spelling error, translation and language processing.
- **Vector space models and word vectors.** Introduction to word vectors (example Word2Vec). Word vectors as predictors for semantic language processing.

Objectives

At the end of the course students should

- understand how known natural languages relate to formal languages in the Chomsky hierarchy;

- have knowledge of several context free grammars equivalents;
- understand how we might make predictions about language processing and language acquisition from formal models;
- know how to use information theoretic concepts to describe aspects of natural language.

Recommended reading

* Jurafsky, D. & Martin, J. (2008). *Speech and language processing*. Prentice Hall.
Manning, C.D. & Schutze, H. (1999) *Foundations of statistical natural language processing*. MIT Press.
Ruslan, M. (2003) *The Oxford handbook of computational linguistics*. Oxford University Press.
Clark, A., Fox, C. & Lappin, S. (2010) *The handbook of computational linguistics and natural language processing*. Wiley-Blackwell.
Kozen, D. (1997) *Automata and computibility*. Springer.

Security

Lecturer: Dr M.G. Kuhn

No. of lectures: 12

Suggested hours of supervisions: 3

Prerequisite courses: Operating Systems; Computer Networking; Programming in C; Unix Tools

Aims

This course provides an overview of technical measures commonly used to enforce security policies, to protect networked and multi-user information systems against malicious user activity, mainly at the level of operating systems and network protocols. It also discusses common security concepts and pitfalls for application programmers and system architects, and strategies for exploiting and mitigating the resulting vulnerabilities.

Lectures

- **Introduction.** Malicious intent: safety vs. security engineering. Security policies, targets, mechanisms. Aspects of confidentiality, integrity, availability, privacy. Requirements across different applications.
- **Operating-system security overview.** Access-control matrix, trusted computing base, domain separation, CPU modes, system calls, residual information protection, virtual machines.

- **POSIX discretionary access control.** User and group databases and identifiers, file permission modes, ownership rights, sticky bit, group inheritance, set-uid, elevation of privileges, root user, NFS root squash, chroot, POSIX.1e ACLs.
- **Windows discretionary access control.** NTFS access rights, security identifiers, access-control entries and lists, inheritance, services, auditing, NFSv4 ACLs.
- **Linux-specific mechanisms.** LSM, Linux capabilities, AppArmor, seccomp, namespaces, containers.
- **Running untrusted code.** Mandatory access control, covert channels, SELinux, type enforcement, iOS/macOS/Android app-store sandboxes, capabilities.
- **Software vulnerabilities.** buffer/integer overflows, ASLR, metacharacter vulnerabilities: shell and SQL injection, side channels, race conditions, environmental exploits, fuzzing.
- **Cryptography overview.** Private/public-key encryption, MACs, digital signatures, certificates, key revocation, secure hash functions, key-establishment schemes, key generation.
- **Entity authentication.** Password verification, guessing user-generated secrets, biometric identification, hardware tokens, challenge-response authentication protocols, Kerberos, ssh, TLS.
- **Internet protocols.** TCP vs UDP, firewalls, iptables, IPSEC/IKE, VPNs, IP options/fragmentation, DDoS.
- **Web security.** HTTP basics, HTTP authentication, cookies, single sign-on (Ucam WebAuth, SAML), delegation (OAuth2), JavaScript, cross-site scripting, cross-site request forgery, same-origin policy, CORS.

Objectives

By the end of the course, students should appreciate the importance of adversarial thinking in systems design and have a good overview of the security mechanisms and attributes of some of the most commonly used operating systems, networking infrastructure and Internet applications. They should also understand commonly exploited vulnerabilities of authentication mechanisms and know how to avoid some common security pitfalls in software development.

Recommended reading

Gollmann, D. (2010). *Computer security*. Wiley (3rd ed.).
Dowd, M.; McDonald, J.; Schuh, J. (2007). *The art of software security assessment*. Addison-Wesley.

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.

Students taking the Part II *50%* option of the CST will read papers 7, 8 and 9 and submit a dissertation. Each of these four is marked out of 100 giving a total available credit in Part II of 400 marks. Alternatively, those taking the Part II *75%* option will take papers 8, 9, submit a dissertation, and offer two units of assessment.

Some courses are specific to either Paper 7 or are Units of Assessment, and have been marked as such in this booklet. Those students following the *75% Computer Science option*, who have taken Paper 7 in Part II, will additionally choose two of the units of assessment. Those students taking the *50% Computer Science option*, will attend the Paper 7 courses instead, full details of which can be found in the Part IB booklet. The remaining courses can be chosen by all Part II students.

The taught modules in Part II are examined in papers 7, 8 and 9 and you answer five questions from each paper. There are no restrictions on which questions you answer. The layout of the papers is announced just before the Michaelmas term starts, but it is generally mostly the same as in previous years, varying only to accomodate new, withdrawn or suspended courses. The units are assessed in a variety of ways, and details can be found in the course descriptions.

It is up to you to make sure you read sufficient courses to be able to answer five questions on each of the papers. Generally, you should aim to be able to answer at least six questions on each paper. You are certainly not expected to go to all the Part II lectures and be able to answer all of the questions on every paper — that would be more or less impossible.

Here is a suggestion for how to plan your courses: In September, just before the start of the year, look through the course list and strike out any course you know you won't do (i.e. remove the definite 'no's - there are always some). Then attend the first lecture of every Part II course to get the feel for it and make a decision on whether to continue after checking that dropping the course doesn't leave you short on any paper. Work on the basis of being able to answer 6 questions, with a 7th as a backup where you are confident of scoring half marks (but probably no more).

It is the duty of your Director of Studies to advise you in course selection so do ask for guidance.

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 2019: Part II lectures

Unit: Advanced Graphics and Image Processing

This course is only taken by Part II 75% students.

Lecturers: Dr R. Mantiuk

No. of lectures and practical classes: 12

Prerequisite courses: Programming in C.

Capacity: 30-50

Aims

Advanced Graphics covers topics related to processing, perception and display of images. The focus of the course is on the algorithms behind new emerging display technologies, such as virtual reality, augmented reality, and high dynamic range displays. It complements two computer graphics courses, Introduction to Graphics and Further Graphics, by introducing problems that became the part of graphics pipeline: tone-mapping, post-processing, displays and models of visual perception.

Lectures

- **GP-GPU**: scheduling and thread mapping, reductions.
- **Advanced image processing**: edge-stopping filters, pyramids, optimization-based image processing.
- **Beyond 2D**: stereo rendering and light fields.
- **Models of visual perception**: visual system, brightness perception, detection and discrimination, contrast sensitivity function, contrast constancy, perceptually uniform spaces, depth perception.
- **High Dynamic Range and tone mapping**: dynamic range, display model, methods of tone-mapping.
- **Display technologies**: 2D displays, 3D displays, temporal display characteristic, HDR displays.
- **Virtual and Augmented Reality**: display technologies, VR rendering, orientation tracking, pose tracking, perceptual considerations, panoramic imaging.

Objectives

By the end of the course students should be able to:

- implement real-time image processing methods on a GPU (OpenCL);

- design and implement a tone-mapping algorithm;
- describe the limitations of display technologies (dynamic range, brightness, visual comfort, VR simulation sickness) and how they can be addressed using computational methods (tone-mapping, HDR displays);
- describe the limitations of the visual system and how those limitation can be exploited in computer graphics and image processing.

Recommended reading

Hainich, R. & Bimber, O. (2016) *Displays: Fundamentals and Applications*. CRC Press (2nd ed.).
Boreskov, A. & Shikin, E. (2013) *Computer Graphics: From Pixels to Programmable Graphics Hardware*. CRC Press.
Reinhard, E., et. al. (2010) *High Dynamic Range Imaging: Acquisition, Display, and Image-Based Lighting*. Morgan Kaufmann (2nd ed.).

Bioinformatics

Lecturer: Professor P. Lio'

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. Computing and storing information with DNA (including Adleman's experiment).
- **Dynamic programming.** Longest common subsequence, DNA global and local alignment, linear space alignment, Nussinov algorithm for RNA, heuristics for multiple alignment. (Vol. 1, chapter 5)
- **Sequence database search.** Blast. (see notes and textbooks)
- **Genome sequencing.** De Bruijn graph. (Vol. 1, chapter 3)

- **Phylogeny.** Distance based algorithms (UPGMA, Neighbour-Joining). Parsimony-based algorithms. Examples in Computer Science. (Vol. 2, chapter 7)

- **Clustering.** Hard and soft K-means clustering, use of Expectation Maximization in clustering, Hierarchical clustering, Markov clustering algorithm. (Vol. 2, chapter 8)

- **Genomics Pattern Matching.** Suffix Tree String Compression and the Burrows-Wheeler Transform. (Vol. 2, chapter 9)

- **Hidden Markov Models.** The Viterbi algorithm, profile HMMs for sequence alignment, classifying proteins with profile HMMs, soft decoding problem, Baum-Welch learning. (Vol. 2, chapter 10)

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

* Compeau, P. & Pevzner, P.A. (2015). *Bioinformatics algorithms: an active learning approach*. Active Learning Publishers.
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: Jack Lang and Stewart McTavish

No. of lectures: 8

Suggested hours of supervisions: 2

Prerequisite course: Economics, Law & Ethics

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 be 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/`

Unit: Data Science: principles and practice

This course is only taken by Part II 75% students.

Lecturers: Dr E. Kochmar, Dr. G. Emerson and Dr D. Wischik

No. of lectures and practical classes: 16

Prerequisite courses: NST Mathematics, Machine Learning and Real-World Data and Foundations of Data Science.

Capacity: 40-50

Aims

The course will develop core areas of Data Science (eg. models for regression and classification) from several perspectives: conceptual formulation and properties, solution algorithms and their implementation, data visualization for exploratory data analysis and the effective presentation of modelling outputs. The lectures will be complemented by practical classes using Python, scikit-learn and TensorFlow.

Lectures

- **Introduction.** Motivation, applications, examples, common data formats (csv, json), loading data with Python, calculating statistics over a dataset with numpy, logistics and overview of the course.
- **Linear Regression.** Defining a model, fitting a model, least squares regression, linear regression, gradient descent, scikit-learn.

- **Practical: Linear Regression**
- **Classification, part I.** Classification, logistic regression, perceptron, multi-class classification, classification performance measures.
- **Practical: Classification I**
- **Classification, part II.** An overview of other classification techniques (e.g., decision trees, SVMs) and more advanced techniques including ensemble-based models (boosting, bagging, exemplified with AdaBoost and Random Forests).
- **Practical: Classification II**
- **Deep learning basics.** Neural networks, applications in the world, optimization, stochastic gradient descent, backpropagation, learning rates
- **Deep learning with TensorFlow.** Introduction to TensorFlow, minimal TensorFlow example, symbolic graphs, training a network, practical tips for deep learning.
- **Practical: Deep learning with TensorFlow**
- **Deep learning architectures.** Convolutional networks, RNNs, LSTMs, autoencoders, regularization.
- **Practical: Deep learning architectures**
- **Visualization, part I.** Scales and coordinates, depicting comparisons.
- **Visualization, part II.** Common plotting patterns, including dimension reduction.
- **Practical: Visualization**
- **Challenges in Data Science.** Summary of the course, ethics and privacy in data science, P-hacking, look-everywhere effect, bias in the training data, interpretability, information about the hand out test.

Objectives

By the end of the course students should be able to:

- demonstrate understanding and practical skills in Data Science;
- be able to specify and work with an analytical model;
- be able to effectively implement Data Science algorithms;
- understand how data visualization underpins exploring datasets as well as communicating the findings of data science models.

Recommended reading

Bishop, C.M. (2008). *Pattern Recognition and Machine Learning*. Springer.
MacKay, D.J. (2003). *Information Theory, Inference and Learning Algorithms*. Cambridge University Press.
Python Basic Tutorial. Available online:
`https://www.tutorialspoint.com/python/index.htm`
Numpy: Quickstart Tutorial. Available online:
`https://docs.scipy.org/doc/numpy/user/quickstart.html`
Get Started with TensorFlow. Available online:
`https://www.tensorflow.org/tutorials/`

Denotational Semantics

Lecturer: Professor M. 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.

Unit: Digital Signal Processing

The whole course is for Part II 75% students, while Part II 50% students can take one question in Paper 7 about part 1 of this course.

Lecturer: Dr M.G. Kuhn

No. of lectures: 16

Prerequisite courses: Mathematical Methods I and III from the NST Mathematics course (or equivalent), LaTeX and MATLAB (recommended).

Aims

This course teaches the basic signal-processing principles necessary to understand many modern high-tech systems, with application examples focussing on audio processing, audio and image coding, and communication systems. Students will gain practical experience from numerical experiments in programming assignments (in MATLAB, NumPy or Julia).

Lectures

Part 1 (about 9–10 lectures) focusses on basic theory and audio applications.

- **Signals and systems.** Discrete sequences and systems: types and properties. Amplitude, phase, frequency, modulation, decibels, root-mean square. Linear time-invariant systems, convolution. Some examples from electronics, optics and acoustics.
- **Phasors.** Eigen functions of linear time-invariant systems. Review of complex arithmetic. Phasors as orthogonal base functions.
- **Fourier transform.** Forms and properties of the Fourier transform. Convolution theorem. Rect and sinc.
- **Dirac's delta function.** Fourier representation of sine waves, impulse combs in the time and frequency domain. Amplitude-modulation in the frequency domain.
- **Discrete sequences and spectra.** Sampling of continuous signals, periodic signals, aliasing, interpolation, sampling and reconstruction, sample-rate conversion, oversampling, spectral inversion.
- **Discrete Fourier transform.** Continuous *versus* discrete Fourier transform, symmetry, linearity, FFT, real-valued FFT, FFT-based convolution, zero padding, FFT-based resampling, deconvolution exercise.
- **Spectral estimation.** Short-time Fourier transform, leakage and scalloping phenomena, windowing, zero padding. Audio and voice examples. DTFM exercise.
- **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.
- **Infinite impulse-response filters.** Sequences as polynomials, *z*-transform, zeros and poles, some analog IIR design techniques (Butterworth, Chebyshev I/II, elliptic filters, second-order cascade form).

Part 2 (about 6–7 lectures) adds material on software-defined radio techniques, statistical signals, audio-visual signal compression and linear feedback control systems.

- **Band-pass signals.** Band-pass sampling and reconstruction, IQ up and down conversion, superheterodyne receivers, software-defined radio front-ends, IQ representation of AM and FM signals and their demodulation.
- **Digital communication.** Pulse-amplitude modulation. Matched-filter detector. Pulse shapes, inter-symbol interference, equalization. IQ representation of ASK, BSK, PSK, QAM and FSK signals. Clock recovery. Spectral characteristics of binary sequences. [2 hours]
- **Random sequences and noise.** Random variables, stationary and ergodic processes, autocorrelation, crosscorrelation, deterministic cross-correlation sequences, filtered random sequences, white noise, periodic averaging.

- **Correlation coding.** Entropy, delta coding, linear prediction, dependence *versus* correlation, random vectors, covariance, decorrelation, matrix diagonalization, eigen decomposition, Karhunen–Loève transform, principal component analysis. Relation to orthogonal transform coding using fixed basis vectors, such as DCT.
- **Lossy versus lossless compression.** What information is discarded by human senses and can be eliminated by encoders? Perceptual scales, audio masking, spatial resolution, colour coordinates, some demonstration experiments.
- **Quantization, image coding standards.** Uniform and logarithmic quantization, A/mu-law coding, dithering, JPEG.

Objectives

By the end of part 1 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;

By the end of part 2, students should be able to discuss and explain many fundamental concepts of techniques commonly used in digital communication systems in terms of the concepts introduced in part 1.

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.).
Stein, J. (2000). *Digital signal processing – a computer science perspective.* Wiley.

Information Theory

Lecturer: Professor J.G. Daugman

No. of lectures: 16

In lieu of supervisions, exercises will be set and reviewed in three Examples Classes.

Aims

This course introduces the principles and applications of information theory: how information is measured in terms of probability and various entropies, how these are used to calculate the capacity of communication channels, with or without noise, and to measure how much random variables reveal about each other. Coding schemes including error correcting codes are studied along with data compression, spectral analysis, transforms, and wavelet coding. Applications of information theory are reviewed, from astrophysics 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 joint and conditional entropy; chain rule for entropy. Cross-entropy and distances between distributions. Mutual information between random variables. Why entropy gives fundamental measures of information content.
- **Source coding theorem; prefix, variable-, and fixed-length codes.** Markov sources. Entropy of a multi-state Markov process. Symbol codes; Huffman codes and the prefix property. Binary symmetric channels. Capacity of a noiseless discrete channel.
- **Noisy discrete channel properties, and channel capacity.** Perfect communication through a noisy channel: error-correcting codes. Capacity of a discrete channel as the maximum of its mutual information.
- **Information represented by projections and in transforms.** Expressing data in vector spaces or as a linear combination of basis functions. Inner product spaces and orthonormal systems. Norms, span, and linear subspaces; dimensionality reduction.
- **Fourier analysis: series and transforms, discrete or continuous.** How periodic and aperiodic data are analysed and represented by Fourier methods. Rates of convergence. Information revealed in the Fourier domain. Discrete, inverse, and Fast Fourier Transforms; butterfly algorithm. Duality properties. Wavelet transforms.
- **Spectral properties of continuous-time signals and channels.** Signals represented as combinations of complex exponential eigenfunctions; channels represented as spectral filters that add noise. Convolution. Applying Fourier analysis to communication schemes.
- **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.

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.).

Unit: Multicore Semantics and Programming

This course is only taken by Part II 75% students.

Lecturers: Professor P. Sewell and Dr T. Harris

No. of lectures and practical classes: 8 x 2-hour sessions

Prerequisite courses: Discrete mathematics, Object-Oriented programing, Semantics of Programming Languages.

Capacity: 20

Aims

In recent years multiprocessors have become ubiquitous, but building reliable concurrent systems with good performance remains very challenging. The aim of this module is to introduce some of the theory and the practice of concurrent programming, from hardware memory models and the design of high-level programming languages to the correctness and performance properties of concurrent algorithms.

Lectures

Part 1: Introduction and relaxed-memory concurrency [Professor P. Sewell]

- **Introduction**. Sequential consistency, atomicity, basic concurrent problems. [1 block]
- **Concurrency on real multiprocessors**: the relaxed memory model(s) for x86, ARM, and IBM Power, and theoretical tools for reasoning about x86-TSO programs. [2 blocks]
- **High-level languages**. An introduction to C/C++11 and Java shared-memory concurrency. [1 block]

Part 2: Concurrent algorithms [Dr T. Harris]

- **Concurrent programming**. Simple algorithms (readers/writers, stacks, queues) and correctness criteria (linearisability and progress properties). Advanced synchronisation patterns (e.g. some of the following: optimistic and lazy list algorithms, hash tables, double-checked locking, RCU, hazard pointers), with discussion of performance and on the interaction between algorithm design and the underlying relaxed memory models. [3 blocks]
- **Research topics**, likely to include one hour on transactional memory and one guest lecture. [1 block]

Objectives

By the end of the course students should:

- have a good understanding of the semantics of concurrent programs, both at the multprocessor level and the C/Java programming language level;
- have a good understanding of some key concurrent algorithms, with practical experience.

Recommended reading

Herlihy, M. & Shavit, N. (2008). *The art of multiprocessor programming*. Morgan Kaufmann.

Unit: Natural Language Processing

This course is only taken by Part II 75% students.

Lecturers: Professor S. Teufel, Professor P. Buttery, Dr A. Vlachos and Dr R. Cotterell

No. of lectures and practical classes: 12+3

Prerequisite courses: Machine Learning and Real-World Data, Formal Models of Language, Foundations of Data Science, Artificial Intelligence.

Capacity: 30

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. Students will also be introduced to practical experimentation in natural language processing.

Lectures

- **Introduction.** Brief history of NLP research, some 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. Some limitations of context-free grammars.
- **Dependency structures.** English as an outlier. Universal dependencies. Introduction to dependency parsing.
- **Compositional semantics.** Logical representations. Compositional semantics and lambda calculus. Inference and robust entailment. Negation.
- **Lexical semantics.** Semantic relations, WordNet, word senses.
- **Distributional semantics.** Representing lexical meaning with distributions. Similarity metrics.
- **Distributional semantics and deep learning.** Embeddings. Grounding. Multimodal systems and visual question answering.
- **Discourse processing.** Anaphora resolution, summarization.
- **Language generation and regeneration.** Generation and regeneration. Components of a generation system. Generation of referring expressions.
- **Recent NLP research.** Some recent NLP research.
- **Practical on sentiment analysis.** Students will build a sentiment analysis system which will be trained and evaluated on supplied data. The system will be built from existing components, but students will be expected to compare approaches and some programming will be required for this.

Objectives

By 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.

Principles of Communications

Lecturer: Professor J.A. Crowcroft

No. of lectures: 16

Suggested hours of supervisions: 4

Prerequisite course: Computer Networking

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. Review of structure of real networks, links, end systems and switching systems. [1 lecture]
- **Routing.** Central versus Distributed Routing Policy Routing. Multicast Routing Circuit Routing [6 lectures]
- **Flow control and resource optimisation.** 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? Optimisation as a model of network& user. TCP in the wild. [3 lectures]
- **Packet Scheduling.** Design choices for scheduling and queue management algorithms for packet forwarding, and fairness. [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]
- **System Structures and Summary.** Abstraction, layering. The structure of real networks, links, end systems and switching. [2 lectures]

Objectives

At the end of the course students should be able to explain the underlying design and behaviour of protocols and networks, including capacity, topology, control and use. Several specific mathematical approaches are covered (control theory, optimisation).

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

Types

Lecturer: Dr N. Krishnaswami

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]
- **Propositions as types.** The Curry-Howard correspondence between intuitionistic propositional calculus and simply-typed lambda calculus. Inductive types and iteration. Consistency and termination. [2 lectures]
- **Polymorphic lambda calculus (PLC).** PLC syntax and reduction semantics. Examples of datatypes definable in the polymorphic lambda calculus. Type inference. [3 lectures]
- **Monads and effects**. Explicit versus implicit effects. Using monadic types to control effects. References and polymorphism. Recursion and looping. [2 lectures]
- **Continuations and classical logic**. First-class continuations and control operators. Continuations as Curry-Howard for classical logic. Continuation-passing style. [2 lectures]
- **Dependent types.** Dependent function types. Indexed datatypes. Equality types and combining proofs with programming. [2 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;
- understand how types can be used to control side-effects in programming;
- 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 2020: Part II lectures

Unit: Cloud Computing

This course is only taken by Part II 75% students.

Lecturers: Dr E. Kalyvianaki and Dr A. Madhavapeddy

No. of lectures and practical classes: 10+3

Prerequisite courses: Operating Systems, Concepts in Programming Languages Concurrent and Distributed Systems, Computer Networking and Unix Tools.

Capacity: 50

Aims

This module aims to teach students the fundamentals of Cloud Computing covering topics such as virtualization, data centres, cloud resource management, cloud storage and popular cloud applications including batch and data stream processing. Emphasis is given on the different backend technologies to build and run efficient clouds and the way clouds are used by applications to realise computing on demand. The course will include practical tutorials on cloud infrastructure technologies. Students will be assessed via a Cloud-based coursework project.

Lectures

- Introduction to Cloud Computing
- Data centres
- Virtualization I
- Virtualization II
- MapReduce
- MapReduce advanced
- Resource management for virtualized data centres
- Cloud storage
- Cloud-based data stream processing

Objectives

By the end of the course students should:

- understand how modern clouds operate and provide computing on demand;

- understand about cloud availability, performance, scalability and cost;
- know about cloud infrastructure technologies including virtualization, data centres, resource management and storage;
- know how popular applications such as batch and data stream processing run efficiently on clouds;

Recommended reading

Marinescu, D.C. *Cloud Computing, Theory and Practice*. Morgan Kaufmann.
Barham, P., et. al. (2003). “Xen and the Art of Virtualization”. In *Proceedings of SOSP 2003*.
Charkasova, L., Gupta, D. & Vahdat, A. (2007). “Comparison of the Three CPU Schedulers in Xen”. In *SIGMETRICS 2007*.
Dean, J. & Ghemawat, S. (2004). “MapReduce: Simplified Data Processing on Large Clusters”. In *Proceedings of OSDI 2004*.
Zaharia, M, et al. (2008). “Improving MapReduce Performance in Heterogeneous Environments”. In *Proceedings of OSDI 2008*.
Hindman, A., et al. (2011). “Mesos: A Platform for Fine-Grained Resource Sharing in Data Center”. In *Proceedings of NSDI 2011*.
Schwarzkopf, M., et al. (2013). “Omega: Flexible, Scalable Schedulers for Large Compute Clusters”. In *EuroSys 2013*.
Ghemawat, S. (2003). “The Google File System”. In *Proceedings of SOSP 2003*.
Chang, F. (2006). “Bigtable: A Distributed Storage System for Structured Data”. In *Proceedings of OSDI 2006*.
Fernandez, R.C., et al. (2013). “Integrating Scale Out and Fault Tolerance in Stream Processing using Operator State Management”. In *SIGMOD 2013*.

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. (2012). *Computer architecture: a quantitative approach*. Elsevier (5th ed.) ISBN 9780123838728. (the 3rd and 4th editions are also good)

Computer Vision

Lecturer: Professor J.G. Daugman

No. of lectures: 16

Exercises will be set and reviewed in three Examples Classes.

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; probabilistic classifiers; visual inference, recognition, and learning.

Lectures

- **Goals of computer vision; why they are so difficult.** Image formation, and the ill-posed problem of making 3D inferences about objects and their properties from images.
- **Image sensing, pixel arrays, CCD and CMOS 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.** Gradient vector field; Laplacian operator and its zero-crossings.
- **Multi-scale contours, feature detection and matching.** SIFT (scale-invariant feature transform); pyramids. 2D wavelets as visual primitives. Active contours. Energy-minimising snakes.
- **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. Colour computation, motion and image 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 volumetric coordinates.
- **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.

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.

Cryptography

Lecturer: Dr M.G. Kuhn

No. of lectures: 16

Suggested hours of supervisions: 3–4

Prerequisite courses: Mathematical Methods I from the NST Mathematics course, Discrete Mathematics, Complexity Theory

Aims

This course provides an overview of basic modern cryptographic techniques and covers essential concepts that users of cryptographic standards need to understand to achieve their intended security goals.

Lectures

- **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, concrete-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.

- **Secure hash functions.** One-way functions, collision resistance, padding, Merkle–Damgård construction, sponge function, duplex construct, entropy pool, SHA standards.

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

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

- **Number theory, finite groups and fields.** Modular arithmetic, Euclid's algorithm, inversion, groups, rings, fields, GF(2^n), subgroup order, cyclic groups, Euler's theorem, Chinese remainder theorem, modular roots, quadratic residues, modular exponentiation, easy and difficult problems. [2 lectures]

- **Discrete logarithm problem.** Baby-step-giant-step algorithm, computational and decision Diffie–Hellman problem, DH key exchange, ElGamal encryption, hybrid cryptography, Schnorr groups, elliptic-curve systems, key sizes. [2 lectures]

- **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, RSA signatures, Schnorr identification scheme, ElGamal signatures, DSA, PS3 hack, certificates, PKI.

Objectives

By the end of the course students should

- be familiar with commonly used standardized cryptographic building blocks;

- be able to match application requirements with concrete security definitions and identify their absence in naive schemes;

- understand various adversarial capabilities and basic attack algorithms and how they affect key sizes;

- understand and compare the finite groups most commonly used with discrete-logarithm schemes;

- understand the basic number theory underlying the most common public-key schemes, and some efficient implementation techniques.

Recommended reading

Katz, J., Lindell, Y. (2015). *Introduction to modern cryptography*. Chapman & Hall/CRC (2nd ed.).

E-Commerce

Lecturers: Jack Lang, Stewart McTavish and others

No. of lectures: 8

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

Prerequisite courses: Business Studies, Security, Economics, Law & Ethics

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.

Machine Learning and Bayesian Inference

Lecturer: Dr S.B. Holden

No. of lectures: 16

Suggested hours of supervisions: 4

Prerequisite courses: Artificial Intelligence, Foundations of Data Science, Discrete Mathematics and Probability, Linear Algebra and Calculus from the NST Mathematics course.

Aims

The Part 1B course *Artificial Intelligence* introduced simple neural networks for supervised learning, and logic-based methods for knowledge representation and reasoning. This course has two aims. First, to provide a rigorous introduction to machine learning, moving beyond the supervised case and ultimately presenting state-of-the-art methods. Second, to provide an introduction to the wider area of probabilistic methods for representing and reasoning with knowledge.

Lectures

- **Introduction to learning and inference.** Supervised, unsupervised, semi-supervised and reinforcement learning. Bayesian inference in general. What the naive Bayes method actually does. Review of backpropagation. Other kinds of learning and inference. [1 lecture]

- **How to classify optimally.** Treating learning probabilistically. Bayesian decision theory and Bayes optimal classification. Likelihood functions and priors. Bayes theorem as applied to supervised learning. The maximum likelihood and maximum *a posteriori* hypotheses. What does this teach us about the backpropagation algorithm? [2 lectures]

- **Linear classifiers I.** Supervised learning via error minimization. Iterative reweighted least squares. The maximum margin classifier. [2 lectures]

- **Gaussian processes.** Learning and inference for regression using Gaussian process models. [2 lectures]

- **Support vector machines (SVMs).** The kernel trick. Problem formulation. Constrained optimization and the dual problem. SVM algorithm. [2 lectures]

- **Practical issues.** Hyperparameters. Measuring performance. Cross-validation. Experimental methods. [1 lecture]

- **Linear classifiers II.** The Bayesian approach to neural networks. [1 lecture]

- **Unsupervised learning I.** The k-means algorithm. Clustering as a maximum likelihood problem. [1 lecture]

- **Unsupervised learning II.** The EM algorithm and its application to clustering. [1 lecture]

- **Bayesian networks I.** Representing uncertain knowledge using Bayesian networks. Conditional independence. Exact inference in Bayesian networks. [2 lectures]

- **Bayesian networks II.** Markov random fields. Approximate inference. Markov chain Monte Carlo methods. [1 lecture]

Objectives

At the end of this course students should:

- Understand how learning and inference can be captured within a probabilistic framework, and know how probability theory can be applied in practice as a means of handling uncertainty in AI systems.

- Understand several algorithms for machine learning and apply those methods in practice with proper regard for good experimental practice.

Recommended reading

If you are going to buy a single book for this course I recommend:

* Bishop, C.M. (2006). *Pattern recognition and machine learning*. Springer.

The course text for Artificial Intelligence I:

Russell, S. & Norvig, P. (2010). *Artificial intelligence: a modern approach*. Prentice Hall (3rd ed.).
covers some relevant material but often in insufficient detail. Similarly:

Mitchell, T.M. (1997). *Machine Learning*. McGraw-Hill.
gives a gentle introduction to some of the course material, but only an introduction.

Recently a few new books have appeared that cover a lot of relevant ground well. For example:
Barber, D. (2012). *Bayesian Reasoning and Machine Learning*. Cambridge University Press.
Flach, P. (2012). *Machine Learning: The Art and Science of Algorithms that Make Sense of Data*. Cambridge University Press.
Murphy, K.P. (2012). *Machine Learning: A Probabilistic Perspective*. MIT Press.

Unit: Mobile Robot Systems

This course is only taken by Part II 75% students.

Lecturers: Dr A. Prorok

No. of lectures and practical classes: 16

Prerequisite courses: NST Mathematics, Artificial Intelligence, Algorithms.

Capacity: 40

Aims

This course teaches the foundations of autonomous mobile robots, covering topics such as perception, motion control, and planning. It also teaches algorithmic strategies that enable the coordination of multi-robot systems and robot swarms. The course will feature several practical sessions with hands-on robot programming. The students will undertake mini-projects, which will be formally evaluated through a report and presentation.

Lectures

- **Robot motion and control.** Kinematics, control models, trajectory tracking.
- **Control architectures.** Sensor-actuator loops, reactive path planning.
- **Sensing.** Sensors, perception.
- **Localization.** Markov localization, environment modeling, SLAM.
- **Navigation.** Planning, receding horizon control.
- **Multi-robot systems I.** Centralization vs. decentralization, robot swarms.
- **Multi-robot systems II.** Consensus algorithms, graph-theoretic methods.

- **Multi-robot systems III.** Task assignment.
- **Multi-robot systems IV.** Multi-robot path planning.

Objectives

By the end of the course students should:

- understand how to control a mobile robot;
- understand how a robot perceives its environment;
- understand how a robot plans actions (navigation paths);
- know paradigms of coordination in systems of multiple robots;
- know classical multi-robot problems and their solution methods;
- Know how to use ROS (Robot Operating System, `http://www.ros.org`).

Recommended reading

Siegwart, R., Nourbakhsh, I.R. & Scaramuzza, D. (2004). *Autonomous mobile robots*. MIT Press.
Thrun, S., Wolfram B. & Dieter F. (2005). *Probabilistic robotics*. MIT Press.
Mondada, F. & Mordechai B. (2018) *Elements of Robotics*. Springer
Siciliano, B. & Khatib, O. (2016) *Springer handbook of robotics*. Springer.
Mesbahi, M. & Egerstedt, M. (2010) *Graph theoretic methods in multiagent networks*. Princeton University Press.

Mobile and Sensor Systems

Lecturer: Prof C. Mascolo, Dr R.K. Harle

No. of lectures: 12

Suggested hours of supervisions: 3

Prerequisite courses: Operating Systems, Concurrent and Distributed Systems

Aims

This course will cover topics in the areas of mobile systems and communications, and sensor systems and sensor networking. It aims to help students develop and understand the additional complexity introduced by mobility and sensing, including energy constraints, communication in dynamic networks and handling measurement errors. The course will be using various applications to exemplify concepts.

Lectures

- **Introduction to Mobile Systems. MAC Layer concepts.** Examples of mobile systems, differences with non mobile systems. Introduction to MAC layer protocols of wireless and mobile systems.
- **Mobile Infrastructure Communication and Opportunistic Networking.** Description of common communication architectures and protocols for mobile and introduction to models of opportunistic networking.
- **Introduction to Sensor Systems, MAC Layer concepts and Internet of Things.** Sensor systems challenges and applications. Concepts related to duty cycling and energy preservation protocols. Emerging concepts and communications protocols for the Internet of Things.
- **Sensor Systems Routing Protocols.** Communication protocols, data aggregation and dissemination in sensor networks.
- **Machine Learning for Mobile Systems and Sensor Data** Mobile and wearable sensing. Machine Learning on Sensor Data. On-device Machine Learning.
- **Mobile Sensing: Systems Considerations** Considerations of energy preservation. Local computation vs cloud computation. Storage and Latency.
- **Privacy in Mobile and Sensor Systems.** Concepts of mobile and sensor systems privacy. Privacy and sensor based activity inference. Mobility prediction and privacy.
- **Localization I.** Basic concepts. Proximity, trilateration, triangulation, ToA, TDoA. Examples including ultrasonic, mobile networks, UWB.
- **Localization II.** GNSS, RSS fingerprinting/WiFi positioning.
- **Tracking.** Kalman filtering, particle filtering, Pedestrian Dead-Reckoning as an example.
- **Mobile/Wearable health sensing.** Health as a key driver for wearables. High availability and low fidelity (wearable) vs low availability and high fidelity (clinical). Sensing challenges (low power, noisy, poor contact). PPG case study. False positives and the base rate fallacy.
- **Robots and Drones** Concepts related to control, communication and coordination of robotic systems.

Objectives

On completing the course, students should be able to

- describe similarities and differences between standard distributed systems and mobile and sensor systems;
- explain the fundamental tradeoffs related to energy limitations and communication needs in these systems;

- argue for and against different mobile and sensor systems architectures and protocols.
- Understand typical error sources for sensing and be aware of techniques to minimise them.
- put concepts into context of current applications of mobile and sensor systems as described in the course.

Recommended reading

The course is based mainly on research papers cited in each lecture. The following books, however, contain some of the more traditional concepts.

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

Optimising Compilers

Lecturer: Dr T.M. Jones

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.).

Unit: Probability and Computation

This course is only taken by Part II 75% students.

Lecturers: Dr. T.M. Sauerwald, Dr. N. Rivera, Dr. J. Sylvester, Dr. L. Zanetti

No. of lectures and practical classes: 16

Prerequisite courses: Algorithms, Foundations of Data Science.

Capacity: 30

Aims

The aim of this course is to introduce the design and analysis of randomised algorithms. It starts by introducing some essential tools and ideas from probability and graph theory, and develops this knowledge through analysing a variety of examples of randomised algorithms and processes. Ultimately the course demonstrates that randomness can be an elegant programming technique, and particularly helpful when time or space are restricted.

Lectures

- **Introduction and review of probability theory**: Introduction and review of probability theory: Review of probability theory: Random variables, Independence, Markov and Chebychev inequalities. Basic Randomised Algorithms
- **Concentration Inequalities**: How to derive Chernoff bounds, Applications to load balancing and quick-sort. Extensions of Chernoff bounds.
- **Random walks and Markov chains**: Transition matrices, Stationary distribution and Convergence. Mixing time and total variation distance. Applications of Markov chains such as Connectivity testing, Solving 2-SAT, Sampling from unknown distributions and MCMC.
- **Spectral Analysis of Random Walks**: Convergence Rate, Cheeger's Inequality, Applications including Graph Clustering.
- **Advanced Randomised Algorithms**: Sublinear-Time Algorithms, Matrix Verification, Random Projection and Dimensionality Reduction.

Objectives

By the end of the course students should be able to:

- learn how to use randomness in the design of algorithms;
- apply randomisation to various problems coming from optimisation, machine learning and distributed computing;
- use results from probability theory to analyse the performance of randomised algorithms.

Recommended reading

Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis.*, Cambridge University Press, 2nd edition.

Quantum Computing

Lecturer: Dr. S. Herbert

No. of lectures: 16

Suggested hours of supervisions: 4

Prerequisite courses: Foundations of Data Science, Computation Theory

Aims

The principal aim of the course is to introduce students to the basics of the quantum model of computation. The model will be used to study algorithms for searching, factorisation and quantum chemistry as well as other important topics in quantum information such as cryptography and super-dense coding. Issues in the complexity of computation will also be explored. A second aim of the course is to introduce student to near-term quantum computing. To this end, error-correction and adiabatic quantum computing are studied.

Lectures

- **Bits and qubits.** Introduction to quantum states and measurements with motivating examples. Comparison with discrete classical states.
- **Linear algebra.** Review of linear algebra: vector spaces, linear operators, Dirac notation, the tensor product.
- **The postulates of quantum mechanics.** Postulates of quantum mechanics, incl. evolution and measurement.
- **Important concepts in quantum mechanics.** Entanglement, distinguishing orthogonal and non-orthogonal quantum states, no-cloning and no signalling.
- **The quantum circuit model.** The circuit model of quantum computation. Quantum gates and circuits. Universality of the quantum circuit model, and efficient simulation of arbitrary two-qubit gates with a standard universal set of gates.
- **Some applications of quantum information.** Applications of quantum information (other than quantum computation): quantum key distribution, superdense coding and quantum teleportation.
- **Deutsch–Jozsa algorithm.** Introducing Deutsch's problem and Deutsch's algorithm leading onto its generalisation, the Deutsch–Jozsa algorithm.
- **Quantum search.** Grover's search algorithm: analysis and lower bounds.
- **Quantum Fourier Transform & Quantum Phase Estimation.** Definition of the Quantum Fourier Transform (QFT), and efficient representation thereof as a quantum circuit. Application of the QFT to enable Quantum Phase Estimation (QPE).
- **Application 1 of QFT / QPE: Factoring.** Shor's algorithm: reduction of factoring to period finding and then using the QFT for period finding.

- **Application 2 of QFT / QPE: Quantum Chemistry.** Efficient simulation of quantum systems, and applications to real-world problems in quantum chemistry.
- **Quantum complexity.** Quantum complexity classes and their relationship to classical complexity. Comparison with probabilistic computation.
- **Quantum error correction.** Introducing the concept of quantum error correction required for the following lecture on fault-tolerance.
- **Fault tolerant quantum computing.** Elements of fault tolerant computing; the threshold theorem for efficient suppression of errors.
- **Adiabatic quantum computing.** The quantum adiabatic theorem, and adiabatic optimisation. Quantum annealing and D-Wave.
- **Case studies in near-term quantum computation.** Examples of state-of-the-art quantum algorithms and computers, including superconducting and networked quantum computers.

Objectives

At the end of the course students should:

- understand the quantum model of computation and the basic principles of quantum mechanics;
- be familiar with basic quantum algorithms and their analysis;
- be familiar with basic quantum protocols such as teleportation and superdense coding;
- see how the quantum model relates to classical models of deterministic and probabilistic computation.
- appreciate the importance of efficient error-suppression if quantum computation is to yield an advantage over classical computation.
- gain a general understanding of the important topics in near-term quantum computing, including adiabatic quantum computing.

Recommended reading

Books:

Kaye P., Laflamme R., Mosca M. (2007). *An Introduction to Quantum Computing*. Oxford University Press.
Nielsen M.A., Chuang I.L. (2010). *Quantum Computation and Quantum Information*. Cambridge University Press.
Mermin N.D. (2007). *Quantum Computer Science: An Introduction*. Cambridge University Press.

Hirvensalo M. (2001). *Quantum Computing*. Springer.
McGeoch, C. (2014). *Adiabatic Quantum Computation and Quantum Annealing Theory and Practice*. Morgan & Claypool. https://ieeexplore.ieee.org/document/7055969

Papers:

Braunstein S.L. (2003). *Quantum computation tutorial*. Available at:
https://www-users.cs.york.ac.uk/~schmuel/comp/comp_best.pdf
Aharonov D., Quantum computation [arXiv:quant-ph/9812037]
Steane A., Quantum computing [arXiv:quant-ph/9708022]
Albash T., Adiabatic Quantum Computing https://arxiv.org/pdf/1611.04471.pdf
McCardle S. *et al*, Quantum computational chemistry
https://arxiv.org/abs/1808.10402

Other lecture notes:

Umesh Vazirani (UC Berkeley): http://www-inst.eecs.berkeley.edu/~cs191/sp12/
John Preskill (Caltech): http://www.theory.caltech.edu/people/preskill/ph229/
Andrew Childs (University of Maryland): http://cs.umd.edu/~amchilds/qa/
John Watrous (University of Waterloo): https://cs.uwaterloo.ca/~watrous/TQI/

Unit: Topics in Concurrency

This course is only taken by Part II 75% students.

Lecturer: Professor G. Winskel

No. of lectures: 16

Suggested hours of supervisions: 3

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

Capacity: no restrictions

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. Students will be assessed by a one-hour test at the end of the course.

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]
- **Event structures**. Their relation with Petri nets and representation via rigid families. The CCS operations on event stuctures. Maps of event structures. [2 lectures]
- **Games and strategies as event structures**, an introduction to Concurrent Games. Composing strategies – interaction and hiding. A special case: nondeterministic dataflow. [2 lectures]
- **Strategies as concurrent processes**. A higher-order language for strategies. May and must equivalence. Probabilistic and quantum strategies briefly. The future? [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, event structures, a language and reasoning techniques for cryptographic protocols, and the basics of concurrent games;
- 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, apply the basics of concurrent games.

Recommended reading

Comprehensive notes will be provided.

Further reading:

* Aceto, L., et. al. (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.
Winskel, G. (2011-) “The ECSYM notes: Event structures, stable families and games”. Notes for the ERC Research project *Events, Causality and Symmetry (ECSYM)*. Available at: `https://www.cl.cam.ac.uk/ gw104/ecsym-notes.pdf`

Easter Term 2020: 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, linear programming and approximation algorithms.

Lectures

- **Sorting Networks.** Zero-one principle. Merging Network, Bitonic Sorter. Counting Networks. [CLRS2, Chapter 27]
- **Linear Programming.** Definitions and Applications. Formulating Linear Programs. The Simplex Algorithm. Finding Initial Solutions. [CLRS3, Chapter 29]
- **Approximation Algorithms.** (Fully) Polynomial-Time Approximation Schemes. Design Techniques. Applications: Vertex Cover, Subset-Sum, Parallel Machine Scheduling, Travelling Salesman Problem (including a practical demonstration how to solve a TSP instance exactly using linear programming), Hardness of Approximation. [CLRS3, Chapter 35]
- **Randomised Approximation Algorithms.** Randomised Approximation Schemes. Linearity of Expectations and Randomised Rounding of Linear Programs. Applications: MAX3-SAT problem, Weighted Vertex Cover, Weighted Set Cover. Summary: MAX-SAT problem and discussion of various approximation algorithms. [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: Jack Lang, Stewart McTavish 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.
Maurya, A. (2012). *Running Lean: Iterate from Plan A to a Plan That Works*. O'Reilly.
Osterwalder, A. & Pigneur, Y. (2010). *Business Model Generation: A Handbook for Visionaires, Game Changers, and Challengers*. Wiley.
Kim, W. & Mauborgne, R. (2005). *Blue Ocean Strategy*. Harvard Business School Press.

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

Hoare Logic and Model Checking

Lecturer: Dr J. Pichon

No. of lectures: 12

Suggested hours of supervisions: 3

Prerequisite courses: Logic and Proof and Semantics of Programming Languages

Aims

The course introduces two verification methods, Hoare Logic and Temporal Logic, and uses them to formally specify and verify imperative programs and systems.

The first 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 aim is to introduce model checking: to show how temporal models can be used to represent systems, how temporal logic can describe the behaviour of systems, and finally to introduce model-checking algorithms to determine whether properties hold, and to find counter-examples.

Current research trends also will be outlined.

Lectures

- **Part 1: Hoare logic.** 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.
- **Partial and total correctness.** Hoare logic for proving termination. Variants.
- **Semantics and metatheory** Mathematical interpretation of Hoare logic. Semantics and soundness of Hoare logic.
- **Separation logic** Separation logic as a resource-aware reinterpretation of Hoare logic to deal with aliasing in programs with pointers.
- **Part 2: Model checking.** Models. Representation of state spaces. Reachable states.
- **Temporal logic**. Linear and branching time. Temporal operators. Path quantifiers. CTL, LTL, and CTL*.
- **Model checking.** Simple algorithms for verifying that temporal properties hold.
- **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 separation logic;
- be able to write simple models and specify them using temporal logic;
- be familiar with the core ideas of model checking, and be able to implement a simple model checker.

Recommended reading

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