University of Edinburgh

School of Informatics

INFORMATICS

THIRD YEAR

(Computer Science, Software Engineering, Artificial Intelligence)

Course Guide 2004–2005

James Clerk Maxwell Building,
The King's Buildings,
Mayfield Road
Edinburgh,
EH9 3JZ

20 September 2004

Contents

Preface

This document describes the organisation of third year Informatics courses, and is for the information of staff and students. Students should consult this document for information throughout the year.

PLEASE NOTE that the Web version of this guide, available via the Informatics teaching pages, is considered to be definitive, and will be updated as necessary throughout the year. Updates will be posted to the newsgroup `eduni.inf.ug3` – *you must read the newsgroup on a regular basis (several times a week) since it is the main means of announcing new information*.

Changes since the start of year

This section describes updates made to the course guide since the version issued in paper form in week 1.

2004-12-03: fixed bug whereby old material wrongly appeared after new material in the HTML version of the guide.

2004-09-22: removed incorrect staff information in COM page. GAGP was mistakenly listed in semester 2 in the Staff list.

Changes due to semesterization

The change to semesterization, and the accompanying complete revision of the University's curriculum, have resulted in many confusing changes in terminology. You may find old documents (or even new documents) still using old terminology. The following are the main changes for third year:

- The former *modules* (e.g. 'Operating Systems') are now **courses**. These are the units of study in our degrees.
- The former *courses* (CS3, AI3, etc.) no longer exist. However, each of our new **courses** has an associated **subject area** or **discipline** which is either 'Computer Science' or 'Artificial Intelligence' (except for Professional Issues, which is 'Informatics'). The terms CS3 and AI3 will still be used to refer to CS and AI courses in third year. Also, the terms CS3, AI3, SE3, AICS3 etc. are sometimes used to refer to the third year of the CS, AI, SE, AI&CS etc. **degree programmes**.
- *Joint honours* degrees have (confusingly) been replaced by **combined honours** degrees. Note that AICS and AISE are **single honours** degrees in the new system, as they are entirely within the School of Informatics.
- The term **course organizer** still exists; however, formally every new **course** (old *module*) now has its own course organizer. In practice, however, all third year Informatics courses have the same course organizer, who is referred to as the **UG3 course organizer**.

1 Introduction

The *Introductory Meeting* for all students taking Informatics Third Year courses (including combined degree students) is held at 11:10 on Tuesday 21 September 2004, in JCMB Lecture Theatre A.

Some combined degree students may find that the introductory meetings for the two components of their degree clash. These students should attend one of the two meetings, and read with particular care the course guide for the other component. Questions may be posed on the UG3 newsgroup `eduni.inf.ug3`.

1.1 Computer Science courses

Computer Science offers the following courses and projects for third year students: 13 technical courses, a course on Professional Issues, and two major projects. Each technical course comprises up to 20 lectures[1] and is assessed by examination and coursework. The Professional Issues course has fewer formal lectures and a different coursework weighting. All students take a subset of the courses and projects available, subject to the constraints specified in Section 3 below. Details of all courses and major projects are in Section 8.

1.2 Artificial Intelligence courses

Artificial Intelligence offers another 6 technical courses, assessed as for CS courses, and a Large Practical, corresponding to the CS3 major project. The available choice of courses varies between the different joint degrees, and is specified below.

[1] Each course has 20 timetabled lecture slots. However, it is not expected that all 20 slots will be used for lectures. Lecturers may use some slots for other activities, or simply leave them free. The normal number of formal lectures is 16 to 18.

2 Staff

Unless otherwise stated, email addresses are all of the form `name@inf.ed.ac.uk`. JCMB means the James Clerk Maxwell Building, AT means Appleton Tower, FH means Forrest Hill, and BP means Buccleuch Place.

Administration

The course organiser is Julian Bradfield (`cs3co`, room JCMB–2610).

The *Informatics Teaching Office* or ITO (`ito`, room JCMB–1502 and AT–5.03) handles enquiries regarding all aspects of the third-year course, including examinations, coursework and the interpretation of marks, for all Computer Science (in JCMB) degrees (including combined degrees) and Artificial Intelligence (in AT) joint degrees.

The staff teaching the third year courses are:

Semester 1 – CS (all rooms in JCMB)

- Algorithms & Data Structures: Mary Cryan (`mcryan`, 2616).
- Computer Design: Eric McKenzie (`ram`, 2507); Archie Howitt (`arch`, 1508) is responsible for the Hardware Laboratory.
- Functional Programming & Specification: Don Sannella (`dts`, 2618).
- Individual Project: Paul Jackson (`pbj`, 1602);
- Operating Systems: Julian Bradfield (`jcb+os`, 2610).
- Professional Issues: Roland Ibbett (`rni`, 3420).
- Software Engineering with Objects and Components 1: Massimo Felici (`mfelici`, 1402);

Semester 1 – AI

- AI Large Practical: Miles Osborne (`miles`, 2BP–2R17) and Manuel Marques-Pita (`m.marques-pita@ed.`
  2BP–3L06).
- Automated Reasoning: Jacques Fleuriot (`jdf`, AT–3.07).
- Genetic Algorithms and Genetic Programming: Gillian Hayes (`gmh`, JCMB–2107C).
- Introduction to Computational Linguistics: Ewan Klein (`ewan`, 2BP–G16), and Steve Renals (`srenals`, 5BP–2L04).
- Introduction to Vision and Robotics: Barbara Webb (`bwebb`, JCMB–1107).

Semester 2 – CS (all rooms in JCMB)

- Compiling Techniques: Kousha Etessami (`kousha`, 1509).
- Computability & Intractability: Kyriakos Kalorkoti (`kk`, 2612).
- Computer Architecture: Marcelo Cintra (`mc`, 3419).
- Computer Communications: Nigel Topham (`npt`, 2502).
- Computer Security: David Aspinall (`da`, 2606).
- Database Systems: Peter Buneman (`opb`, 1511).
- Enterprise Computing: Stephen Gilmore (`stg`, 1608).
- Language Semantics & Implementation: Gordon Plotkin (`gdp`, 2620).
- Systems Design Project: John Butler (`jhb`, 2504).

Semester 2 – AI

- Introduction to Cognitive Science: Jon Oberlander (`jon`, 6BP–G15).
- Knowledge Representation and Engineering: Alan Smaill (`smaill`, AT–3.08).

3 Degrees and Degree Requirements

The curriculum for a degree is specified partly by University Regulations (the Degree Programmes), and partly by additional requirements imposed by the Schools offering the degree. The degrees also have a normal curriculum which most students will follow, although the Regulations allow a considerable variation from the normal curriculum.

As a consequence, the precise regulations are rather complicated. For most students, it is enough to follow the "normal curriculum" and satisfy the Informatics requirements on choice of courses; those who want to do something else must read all the requirements carefully.

This section of the Course Guide gives the requirements from the Degree Programmes and the School of Informatics internal regulations, and also states the normal curriculum. Requirements are identified by the word 'must'. This document does not give the internal regulations of other Schools; the other School's own documents should be consulted.

It is *your* responsibility to make sure that your course selection is correct. By the end of week 1 of semester 1 you are required to submit an initial choice of the courses you intend to take during the year. If you change any choice, it is your responsibility to notify the ITO. Your choices as at **19 January 2005** are final, and will be used by Registry to timetable your exams. **Note** that if your Director of Studies is not in Informatics, you will have to ask them to change your courses; the ITO can only act on behalf of Informatics DoSes. The choice of courses is stored on WISARD, which you can access through the Edinburgh Student Portal.

In the first subsection, the requirements for the Informatics single honours degrees are presented. The second and third subsection cover the requirements for combined degrees in Computer Science or Artificial Intelligence. Some timetable clashes for combined honours students are unavoidable, thereby restricting choice somewhat: note that the University will not permit students to take courses that clash in the timetable.

Although the regulations permit some flexibility, there may be cases where a student has a good reason to want to take a curriculum outwith the regulations (for example, a single honours student wishing to take a 20 point outside course). In this case, permission should be sought from the Director of Teaching. Such permission will not necessarily be granted, and is formally at the discretion of the Head of College.

In order to simplify the description of the requirements for the Computer Science degrees, certain subsets of CS technical courses are identified and assigned colours. All these courses are 10 credit points.

- The *red* courses are: Algorithms & Data Structures (ADS), Computability & Intractability (CI) and Language Semantics & Implementation (LSI).
- The *yellow* courses are: Computer Design (CD), Computer Architecture (CAR) and Operating Systems (OS).
- The *green* courses are: Computer Security (CS), Functional Programming & Specification (FPS), Software Engineering with Objects and Components 1 (SEOC1) and Enterprise Computing (EC).
- The *blue* courses are: Compiling Techniques (CT), Computer Communications (COM) and Database Systems (DBS).

The AI technical courses form one group.

There are also three major projects, combinations of which are compulsory for most degrees. The CS3 System Design Project (SDP) is a group project; it is a double weight course, carrying 20 credit points. The CS3 CS/SE Individual Project (IP) is a single course, carrying 10 credit points. The AI3 AI Large Practical (AILP) is a single course, carrying 10 credit points.

Finally, there is a 10 credit point course on Professional Issues (PI), which is compulsory for fully accredited degrees. This course may be counted as either CS3 or AI3.

3.1 Single Honours Degree Programmes

The requirements for the Informatics Single Honours and Ordinary degrees in third year are described in this section.

Summary: follow the normal curriculum, and observe the restrictions on choice of courses. But if you want to take an outside course, you may, instead of one of your CS or AI courses.

Degree programme requirements for all single honours degrees: students must take 120 credits at level 9, of which 110 credits must be from the School of Informatics.

3.1.1 Computer Science Honours (B.Sc. and B.Eng.)

Additional Degree programme requirements Students must take both CS major practicals and Professional Issues. Students must take at least 60 additional credits from CS3.

Additional Informatics requirements Students must take at least one red, at least one yellow and at least one blue course from CS3.

Normal curriculum SDP, IP, PI, and eight technical CS3 courses (perhaps including one or two AI3 courses).

3.1.2 Computer Science Ordinary

Transitional regulations for 2004–5. Students take six CS3 technical courses, one CS3 major project, and Professional Issues. There are no restrictions on the choice from within the Computer Science 3 syllabus. *Students must have a total of at least 320 credit points over the course of their degree.*

3.1.3 Software Engineering Honours (B.Eng.)

Additional Degree programme requirements Students must take both CS major practicals and Professional Issues. Students must take at least 60 additional credits from CS3.

Additional Informatics requirements Students must take at least one green course and at least one blue course course from CS3.

Normal curriculum SDP, IP, PI, and eight CS3 technical courses (perhaps including one or two AI3 courses).

3.1.4 Artificial Intelligence and Computer Science Honours (B.Sc.)

Additional Degree programme requirements Students must take the AI Large Practical, the System Design Project, and Professional Issues. Students must take at least 30 additional credits from each of CS3 and AI3.

Additional Informatics requirements Students must take at least one red course from CS3.

Normal curriculum AILP, SDP, PI and eight technical courses, split 3:5, 4:4 or 5:3 from AI3 and CS3

3.1.5 Artificial Intelligence and Computer Science Ordinary Course

Transitional regulations for 2004–5. Students take three CS3 technical courses, three AI3 technical courses, one CS3 or AI3 major project, and Professional Issues. *Students must have a total of at least 320 credit points over the course of their degree.*

3.1.6 Artificial Intelligence and Software Engineering 3

Additional Degree programme requirements Students must take the AI Large Practical, the System Design Project, and Professional Issues. Students must take at least 30 additional credits from each of CS3 and AI3.

Additional Informatics requirements Students must take at least one green course from CS3.

Normal curriculum AILP, SDP, PI and eight technical courses, split 3:5, 4:4 or 5:3 from AI3 and CS3

3.2 Combined Honours Degrees in Computer Science

Summary: take half a programme from each School; the normal Informatics half is described here. Observe the restrictions. If you want to take an outside course, you may, but it is not recommended in combined degrees.

Degree programme requirements for all combined degrees in computer science: Students must take 120 credits at level 9, of which at least 110 credits must be from the Schools of the degree. At least 40 credits must be taken from each subject of the degree (i.e. CS3 for the Informatics half).

Informatics requirements for all combined degrees in computer science: students must take at least three technical courses from CS3, and one CS major practical.

3.2.1 Computer Science and Electronics Honours (B.Eng.)

Additional Degree programme requirements Students must take Electronic Engineering 3.

Additional Informatics requirements Students must take at least two yellow courses from CS3.

Accreditation requirements To obtain BCS exemptions and Engineering Council accreditation, students must take Professional Issues during the degree. If not taken in third year, it must be taken in fourth year.

Normal curriculum within Informatics Four technical CS3 courses, plus either SDP or both IP and PI.

3.2.2 Computer Science and Management Science (B.Sc.)

Additional Degree programme requirements None. (Note: the subject area of the Management Science side is actually called 'Business Studies'.)

Additional Informatics requirements None.

Normal curriculum within Informatics Four technical CS3 courses, plus either SDP or both IP and PI.

3.2.3 Computer Science and Mathematics (B.Sc.)

Additional Degree programme requirements Students must take 'Complex Variable and Differential Equations', and *one of* 'Algebra' or 'Pure and Applied Analysis'.

Additional Informatics requirements Students must take at least two red courses from CS3.

Normal curriculum within Informatics Four technical CS3 courses, plus either SDP or both IP and PI.

3.2.4 Computer Science and Physics (B.Sc.)

Additional Degree programme requirements Students must take 'Dynamics & Relativity', 'Quantum Mechanics', 'Statistical Mechanics' and 'Electromagnetism'.

Additional Informatics requirements Students must take at least one red course and at least one yellow course from CS3.

Normal curriculum within Informatics Four technical CS3 courses, plus either SDP or both IP and PI.

3.2.5 Electronics and Computer Science (M.Eng.)

As for B.Eng. in Computer Science and Electronics.

3.2.6 Electronics and Software Engineering (B.Eng. and M.Eng.)

Additional Degree programme requirements Students must take Electronic Engineering 3.

Additional Informatics requirements Students must take at least one yellow course and at least one green course from CS3.

Accreditation requirements To obtain BCS exemptions and Engineering Council accreditation, students must take Professional Issues during the degree. If not taken in third year, it must be taken in fourth year.

Normal curriculum within Informatics Four technical CS3 courses, plus either SDP or both IP and PI.

3.3 Combined Honours Degrees in Artificial Intelligence

Summary: take half a programme from each School; the normal Informatics half is described here. Observe the restrictions. If you want to take an outside course, you may, but it is not recommended in combined degrees.

Degree programme requirements for all combined degrees in artificial intelligence: Students must take 120 credits at level 9, of which at least 110 credits must be from the Schools of the degree. At least 40 credits must be taken from each subject of the degree, *except* that in Artificial Intelligence with Psychology at least 50 credits must be taken from AI3 and at least 30 credits from Psychology.

Informatics requirements for all combined degrees in artificial intelligence: students must take at least three technical courses from AI3 (except in AI/Maths), and must take the AI Large Practical.

Informatics suggestion for all combined degrees in artificial intelligence: Professional Issues.

3.3.1 Linguistics and Artificial Intelligence (M.A.)

Note: this degree is owned by the School of Philosophy, Psychology and Language Sciences.

Additional Degree programme requirements Note: Language Sciences courses are at level 10, even in third year.

Additional Informatics requirements Students must take Introduction to Computational Linguistics.

Normal curriculum within Informatics AILP, ICL, PI and three other technical courses from AI3.

3.3.2 Artificial Intelligence and Mathematics (B.Sc.)

Additional Degree programme requirements Students must take 'Complex Variable and Differential Equations', and *one of* 'Algebra' or 'Pure and Applied Analysis'. Students must take 'Automated Reasoning'.

Additional Informatics requirements Students must take at least three technical courses from AI3 or CS3, including at least two from AI3.

Normal curriculum within Informatics AILP, AR, PI, and three other technical courses, split 3:0, 2:1 or 1:2 between AI3:CS3. The CS3 courses taken in this degree are usually, but not necessarily, red courses.

3.3.3 Artificial Intelligence and Psychology (B.Sc.)

Additional Degree programme requirements Note: Psychology courses are at level 10, even in third year. Students must take 'Psychology Methodology 1', and *one of* 'Psychology Group Project 1' or 'Psychology Group Project 2'. Students must take at least 50 credits from Psychology.

Additional Informatics requirements Students must take Introduction to Cognitive Science.

Normal curriculum within Informatics AILP, ICS, PI and three other AI3 courses.

3.3.4 Artificial Intelligence with Psychology (B.Sc.)

Additional Degree programme requirements Note: Psychology courses are at level 10, even in third year. Students must take at least 30 credits from Psychology, and at least 50 credits from AI3.

Additional Informatics requirements Students must take Introduction to Cognitive Science.

Normal curriculum within Informatics AILP, ICS, PI and four other AI3 courses.

4 Assessment

4.1 Coursework

Each technical course has coursework, which accounts for 25% of the assessment for the course. Professional Issues has a 15% coursework component.

A 10 credit course is nominally assigned 100 hours of work, taking a full year's work to be thirty 40-hour weeks. In practice, this probably means around 50–80 hours of time outside lectures; thus you should expect to spend a few hours per week per course on coursework (the rest of the time being taken up by the lectures themselves and by note-reading and revision). Of course this is only a rough guide.

It is important that you organise your time carefully during the year; not only will this help you to do better coursework, but it's a useful skill in itself. Remember that access to machines will be most difficult near deadlines. It is OK to hand in work before the deadline!

All course work must be handed back to the ITO (room JCMB–1502 for CS, and AT–5.03 for AI) late in Semester 2 so that it is available for the examiners to inspect during the assessment process. The deadline for handing back coursework is **4 April 2005**.

Plagiarism You *must* read and follow both the University's policy on plagiarism, and the School of Informatics Guidelines on Plagiarism. See the course home page for links.

Handing in coursework Coursework will usually be submitted either electronically, or in paper form to the relevant ITO.

The policy for late coursework is currently imposed on us by the University. The normal situation is given by Undergraduate Assessment Regulation 3.8:

> Late coursework will not be accepted without good reason, will be recorded as late and a penalty will be exacted. That penalty should be a reduction of the mark by 5% of the maximum obtainable mark per working day (e.g. a mark of 65% on the common marking scale would be reduced to 60% after 24 hours). This would apply for up to five working days (or to the time when feedback is given, if this is sooner), after which a mark of zero should be given. *[etc.]*

However, under a blanket exemption from College, small exercises (worth 5% or less of the credit for a course) may have a strict deadline policy. Lecturers will advise you if this is the case when such a piece of coursework is issued.

Requests with good reason for extension should be made to the UG3 Course Organizer. 'Good reason' includes illness, serious personal matters etc. It does not include failure to organize your time effectively!

Return of coursework Work will normally be handed back within two weeks (four in the case of the two major projects).

Coursework will be annotated with both a grade and a mark. Please note that these are provisional and may be revised by the Board of Examiners (for example, this might happen if it became clear that coursework had been much more harshly marked on one course than another).

4.2 Examinations

Each lecture course is examined by a separate 105 minute examination paper in the examination block of semester 2 (hereinafter referred to as "the May examinations", although the diet starts in mid-April). Usually, each of these examinations contains three questions; each candidate is required to attempt two of these. Some courses may have a different format; the course lecturer will advise you if this is the case.

For Honours courses (including combined honours), the examinations *must* be passed at the first attempt, although failures on individual papers are permitted. That is, students failing to obtain an overall Honours pass mark after the May examinations will not be permitted to enter fourth year, even if they resit the examinations.

4.3 Overall assessment

Examinations, coursework and projects are taken into account in assessing each student's performance.

In the case of Honours courses, a combined mark is carried forward to be used together with the final Honours year assessment in awarding the appropriate class of degree. The third year mark accounts for half the final Honours mark.

The final grade for the year is obtained by combining the marks for all courses taken during the year, according to their credit weightings (i.e. 10 points for most Informatics UG3 courses, and 20 for SDP). Note that combined degree marks are calculated strictly by credit weighting of courses; there is no separation of the mark into, for example, an overall CS mark and an overall Maths mark. This is a change from the previous assessment system.

The Board of Examiners has the discretion to take into account criteria other than the raw mark in deciding admission to Honours or the award of an Ordinary degree. The criteria used by the third year Board include:

- did the candidate suffer serious medical, psychological or emotional stresses that adversely affected their performance? For this reason, **it is most important** that you keep your Director of Studies informed of any such circumstances *when they happen*.
- did the candidate's choice of courses include some with unusually low averages, or that encountered serious problems during delivery?
- does the candidate display particular strengths in some areas that mitigate the overall failure?

4.4 Progression to Fourth Year; Ordinary degrees; resits

To be admitted to the fourth year of the Honours degrees, students must achieve an overall mark of 40% in third year, with examinations taken at the first attempt. In addition, it is necessary to pass at least 80 credits' worth of courses. In combined degrees, the Board of Examiners may consider that a clear fail in one half of the degree will debar the candidate from progressing to fourth year. Further, failure on certain courses (notably the major practicals) may reduce the candidate's BCS exemptions (see below).

A candidate who fails to reach the Honours pass mark after the May examinations may nevertheless be awarded a pass at the Ordinary degree level without taking the resit examinations if a performance commensurate with an Ordinary degree has already been obtained.

For students who fail to achieve a pass at the Ordinary degree level in May, resit examinations for the Ordinary degree take place in August. Students needing to resit should consult their Directors of Studies to make sure they are registered for the examinations. Students are not expected to resit papers that they passed in May, as the mark from the May examination will be carried forward. Candidates are not permitted to re-do coursework.

4.5 BCS Exemptions/ CEng accreditation

The rules for accreditation for CEng and exemption from parts of the British Computer Society examinations are somewhat complicated. The following is a summary: if in doubt, consult Roland Ibbett (`rni`).

If you gain an Honours degrees in SE, CS, AI&CS, AI&SE, CS&E or SE&E, including the Professional Issues course, and pass the CS4 project, then you are eligible for CEng accreditation and exemption from Parts I and II (and Project) of the British Computer Society Examinations.

If you gain an Honours degrees in CS&ManSci, CS&Maths, or CS&Physics, and pass the CS4 project, then you are eligible for exemption from Part I (and Project) of the British Computer Society Examinations.

If you gain an Ordinary degree in Informatics, and pass one of the CS3 major practicals, then you are eligible for exemption from Part I (and Project) of the British Computer Society Examinations.

5 Facilities and Information

Computing

You *must* read and follow the University's Computing Regulations: see the home page for links. Please note that any files in student accounts, and their email etc., may be inspected when any breach of the Regulations is suspected.

In the JCMB, all the Linux machines in the Machine Halls are generally available to all students from CS2 onwards. However, special arrangements may apply to certain groups of machines near deadlines, to ensure adequate resources are available to the course in question. You may also obtain accounts for EUCS public laboratories throughout the University. (Support has registration forms for EUCS services.)

You are allowed to use JCMB facilities until 22:00 without any special permission. Anyone who wants to remain in the building after that time must have a letter of authorization from the Head of Division. These can be obtained from the ITO. If you do not have written permission you may be asked to leave by the servitors after 22:00. Access to the machine halls is controlled by your University card at all times.

There are labs available to all Informatics users in Forrest Hill (swipe card required). There are also labs in Appleton Tower. Please check the Computing Support Web pages for up to date information.

Note that you are welcome to use your own computers for doing assessed work, but please bear in mind that: All programs that you write need to run on the School machines, using the software installed there. Some assessed work needs to be submitted via the School machines. You are responsible for making regular backups for work that you do away from the School machines. If you lose work because of a hardware crash on a non-School machine, you are liable to get a zero mark for the relevant exercise. In summary, it really makes sense to use School machines in preference to any other University machines.

Printers, Lecture Notes and Photocopying

You are occasionally required to print documents using laser printers (for example, during the System Design Project). You are therefore permitted a moderate amount of printing and photocopying without cost, provided it is connected with the work of the course. Printer and photocopier use is monitored so that excessive use can be detected.

For some courses, printed lecture notes are provided. Normally, a small number of extra copies will be placed in the CS3 pigeon holes (at the end of the 15 corridor) or by the ITO in Appleton Tower. When these are exhausted it is your responsibility to find copies of the lecture notes (the ITO may be able to help, or the notes may be available on the courses's web page) and make copies yourself – but please do not print copies yourself unless the preprinted copies have run out! *Do not* take more than one copy of each lecture note when they are handed out during lectures. *Please* ensure that spare copies are returned to the lecturer at the end of the lecture.

Technical Support

There is extensive documentation of the computing systems and software available to you (go to `http://www.inf.ed.ac.uk/systems/`, or follow the link from the CS home page). You may also find newsgroups, especially `eduni.inf.ug3` and `eduni.inf.questions`

helpful. For further support use the support request form on the previously mentioned page. If you come across any computing equipment that is faulty please report it via the same form.

Email, news and the Web

The newsgroup `eduni.inf.ug3` will be used for information on CS3 and AI3. You should check this several times a week, preferably every day, as there will often be notices concerning current assignments and running of the course. As electronic mail will frequently be used for communication with individual students, you should also check your email (almost) every day, and arrange for your email to be forwarded if you mainly use machines in other departments.

Many UG3 courses also use their own newsgroups: these are named `eduni.inf.course.`*course-code*; for example the Operating Systems group is `eduni.inf.course.os`.

You may use email, ftp and the Web, both within the department and outside. However, please remember that the UK national networks are strictly for academic business. Also be aware that traffic from outside Europe ultimately results in real-money costs. Therefore you must keep the volume of data low. Traffic is logged and heavy users may find their privilege revoked.

Extra-curricular computing activities will normally be permitted provided they

- do not involve harassment or anti-social behaviour;
- do not break the law or the Computing Regulations;
- use minimal amounts of computing/network resources.

In particular, they must not involve the "holding or distribution of any material which is defamatory, discriminatory, obscene or otherwise illegal or is offensive or calculated to make others fearful, anxious or apprehensive", and must not involve commercial activity.

Please do not abuse these facilities. The School retains the right to withdraw computing facilities if necessary.

Information relating to CS3, SE3 and AI3, including the on-line version of this document, is accessible via the Web from the Informatics teaching page.

6 Complaints and Problems

6.1 Feedback mechanisms

Constructive feedback from staff or students is always welcome.

Informal comments and suggestions can be made by anyone to the UG3 Course Organiser at any time. The classes elect some Class Representatives, who act as a channel to the Course Organiser on class-wide issues.

More formally, Staff-Student liaison meetings are held once per term, normally towards the end of term. Items for the agenda are submitted to either the Course Organiser or the Class Representative. Minutes of these meetings will be available on the web.

Complaints can be made confidentially: for example, if you complain to the Course Organiser about a lecturer failing to mark work on time, the Course Organiser will take the matter up with the member of staff without divulging your identity. (If the lecturer concerned is the Course Organiser, you may address your complaint to the Director of Teaching.)

6.2 Harassment

If you are a victim of harassment, raise the matter quickly with the Course Organiser and/or your Director of Studies. The University has a Code of Practice on Personal Harassment; see course home page for link.

6.3 Academic problems or medical circumstances

If you find yourself experiencing special difficulties (for example, trouble coping with the workload of third year), you should approach your Director of Studies for guidance as soon as a problem becomes apparent.

If you experience a medical problem during the year, and if you believe it has either prevented you from completing your coursework satisfactorily or that it will impair your performance in the May examinations, you should notify your Director of Studies. Your Director will normally advise you to obtain a medical certificate, and in due course your circumstances will be made known to the Board of Examiners (who always treat such information confidentially).

6.4 Appeals

The results of degree examinations for all courses are decided by Boards of Examiners. Each Board (normally) includes all of the lecturers on the course (the lecturers set and mark the degree examinations) and (always) an External Examiner from another university. Directors of Studies normally attend Board meetings which involve their directees. Decisions of Boards of Examiners, once certified in writing, are final except that candidates have the right of appeal: (a) on the grounds of substantial information which for good reason was not available to the examiners when their decision was taken, or (b) on the grounds of alleged improper conduct of an examination (this includes conduct of a meeting of a Board of Examiners). Students are advised to consult their Director of Studies before initiating an appeal. An appeal must be submitted in writing to the Secretary to the University as soon as possible; only in exceptional circumstances are appeals considered more than three months after the results of an examination have been made known to the appellant.

7 Timetables

The timetables for CS/SE/AI3 are now maintained centrally. They can be found at
`http://www.inf.ed.ac.uk/teaching/years/ug3/timetable04.html`

7.1 Timetable during Week 1

In week 1, the following changes apply:

- Monday is a public holiday, so there is no teaching.
- On Tuesday at 11:10, the UG3 Introductory Lecture replaces SEOC1 in JCMB Lecture Theatre A.
- The Hardware Lab sessions for Computer Design do not run.

8 Descriptions of Courses and Projects – CS3

The list of references associated with each course summarizes the main recommended reading. Each reference is annotated according to the scheme:

(***) Essential
(**) Strongly recommended
(*) Useful

8.1 Algorithms and Data Structures

Description

The course aims to provide general techniques for the design of efficient algorithms and, in parallel, develop appropriate mathematical tools for analysing their performance. In this, it broadens and deepens the study of algorithms and data structures initiated in CS2. Along the way, problem solving skills are exercised and developed.

Syllabus

Introductory concepts Review of CS2. Models of computation; time and space complexity; upper and lower bounds, big-O and big-Omega notation; average and worst case analysis.

Divide and conquer Matrix multiplication: Strassen's algorithm; the discrete Fourier transform (DFT), the fast Fourier transform (FFT). Expressing the runtime of a recursive algorithm as a recurrence relation; solving recurrence relations.

Sorting and selection Quicksort and its analysis; selection in expected linear time.

Data structures: Disjoint sets The "disjoint sets" (union-find) abstract data type: specification and implementations as lists and trees. Union-by-rank, path-compression, etc., "heuristics". Applications to finding Minimum Spanning Trees.

Dynamic programming Introduction to the technique; application to Matrix-chain multiplication.

Graph/Network algorithms Network flow, Max-flow/min-cut theorem, Ford-Fulkerson algorithm.

Geometric algorithms Intersecting line segments in two dimensions; convex hull of a set of 2-D points (Graham's scan algorithm).

Assessed Coursework

There will be three assignments designed to aid students' understanding of the lecture material. At least two of these will be pencil and paper exercises. One of them may be a programming exercise.

References

(***) Cormen, Leiserson, Rivest, Stein: *Introduction to Algorithms* (2nd Edition). MIT Press, 2002.

(*) Gibbons: *Algorithmic Graph Theory*. Cambridge University Press, 1985.

(*) Goodrich, Tamassia: *Algorithm Design - Foundations, Analysis, and Internet Examples.* Wiley, 2002.

8.2 Compiling Techniques

This course describes the phases of a modern programming language compiler with an emphasis on widely-used techniques. The course project will require students to implement fragments of a compiler for an imperative programming language.

Context: There are no formal prerequisites other than CS2.

Syllabus

- Introduction: structure of a compiler.
- Lexical analysis: tokens, regular expressions, Lex.
- Parsing: context-free grammars, predictive and LR parsing, Yacc.
- Abstract syntax: semantic actions, abstract parse trees.
- Semantic analysis: symbol tables, bindings, type-checking.
- Stack frames: representation and abstraction.
- Intermediate code: representation trees, translation.
- Basic blocks and traces: canonical trees and conditional branches.
- Instruction selection: algorithms for selection, RISC and CISC.
- Liveness analysis: solution of dataflow equations.
- Register allocation (time permitting): colouring by simplification, coalescing.
- Advanced topics.

Activities

Lectures are given by Kousha Etessami. They will loosely follow Part 1 of the textbook by Andrew W. Appel with references to other textbooks and papers where appropriate.

Assessed Coursework

Two practical compiler exercises.

References

(***) Andrew W. Appel *Modern Compiler Implementation in Java*, 2nd Edition (Cambridge University Press, 2002). ISBN 052182060X. Be aware that this book also exists in versions for the C and Standard ML languages. Since the compiler project for the course is based on Java, it is recommended that students use the Java version of the book. Multiple copies of these books are available in the JCM library.

(**) Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman *Compilers: Principles, Techniques and Tools* Addison Wesles, 1986.

(**) Steven Muchnick *Advanced Compiler Design and Implementation* Morgan Kaufmann, 1997.

(**) Reinhard Wilhelm, Dieter Maurer *Compiler Design* Addison Wesley, 1995.

(*) Charles N. Fischer, Richard J. LeBlanc, Jr. *Crafting a Compiler in C* Benjamin/Cummings, 1991.

(*) J.P. Bennett *Introduction to Compiling Techniques* Cambridge University Press, 1998.

8.3 Computability and Intractability

Description

The course centres around the so-called *Church-Turing Thesis* which proposes that each one of a variety of different formal systems adequately captures the intuitive concept of (*effective*) *computability*. This thesis implies that by studying any one of these systems, for example Turing machines, we can learn about the inherent theoretical limitations of computers. If a problem cannot be solved by a Turing machine, then there is no computable solution to it at all.

Not all the computational problems which can be solved *in principle* can be solved *in practice*: the computational resources required (time or space) may be prohibitive. This observation motivates the study of *resource-bounded* computation. Having accepted an extended version of the Church-Turing thesis—that the computationally tractable problems are those which can be efficiently solved by a Turing machine—we are led inevitably to conclude the existence of natural problems which can be solved in principle but not within any reasonable resource bound.

The fundamental concepts and techniques encountered in this course resonate around the whole of Computer Science and indeed beyond: effective procedures and decidability, simulation methods, machine encodings and universality, diagonalization arguments, and reductions between problems.

Syllabus

Finite state machines Brief reminder of a model which does *not* encompass the intuitive notion of effective procedure.

Turing machines Definition; some programming techniques and example machines; variants of the Turing machine and their equivalence.

The Church-Turing Thesis Empirical evidence for the Thesis; equivalence of Turing machines to other models of computation, both simpler (two-register machines) and more complex (models of computation which are akin to real computers). Overview of the basic ideas; proofs are supplied in appendices to the course notes.

Universal machines Encoding Turing machines; a universal Turing machine.

Decidability and partial decidability The Halting Problem; reductions between problems.

Nondeterministic Turing machines Definition and equivalence to deterministic machines.

Resource bounded computation The class P and its interpretation as the class of computationally tractable decision problems; the class NP and its interpretation as the class of search problems with efficiently verifiable solutions.

NP-completeness Cook's theorem; examples of problems which are complete for NP.

Connections with Logic Brief introduction to first order and second order Logic and discussion of how the ideas relate to complexity classes, mainly the NP-complete problems.

Assessed Coursework

Three sets of assessed problem sets with solutions due in weeks 4, 7, and 10. The problem sets will be handed out in weeks 1, 4 and 7. Marked solutions to problem sets will be returned according to the following schedule: problem set 1 by week 6, problem set 2 by week 9 and problem set 3 at the beginning of the final examination block.

References

(*) Ch. H. Papadimitriou *Computational Complexity*, Addison-Wesley 1994. Useful for those who intend to progress to the CS4 *Computational Complexity* course. Includes good discussion of Logic and its connections to computability. Also useful for those who intend to progress to the CS4 *Computational Complexity* course.

(*) J. E. Hopcroft and J. D. Ullman *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley 1979.

(*) M. Minsky *Computation: Finite and Infinite Machines*, Prentice-Hall 1972. An excellent book and strong on motivation. Unfortunately, it is now only available as an expensive hardback. There are some copies in the reserve section of the JCMB library.

(*) M. R. Garey and D. S. Johnson *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman. A classic book, again strong on motivation covering the latter parts of the course.

8.4 Computer Architecture

Description

Computer architecture is about making computing hardware and software operate as fast as possible and for the minimum cost. Over the years improvements in technology and advances in computer architecture have resulted in huge increases in computer performance. This course examines the fundamentals of high-performance computer architecture and looks at how the interface between hardware and software (architecture and compiler) influences performance.

Syllabus

Fundamentals Performance evaluation methods and metrics, principles of high performance design, technology issues.

Instruction Set Design instruction set classes, registers, memory addressing, instruction set measurements.

Processor Design, Pipelines and Parallel Functional Units Essential elements of a high performance processor. Pipeline design, pipeline hazards & interlocks, out-of-order execution, scoreboards and reservation stations. Control prediction techniques and their exploitation.

Memory System Design Memory hierarchies. Basic cache design and improvements.

I/O I/O interface. RAIDS. Buses.

Multiprocessors Multiprocessor organisations. Cache coherence.

Assessed Coursework

Two exercises are set during the course.

References

(***) J.L. Hennessy & D.A. Patterson *Computer Architecture: A Quantitative Approach* (third edition), Morgan Kaufmann Publishers Inc., 2003. (This is an excellent book which covers the course material so closely that there are **no printed lecture notes for this module**. You are strongly advised to obtain a copy of this book).

8.5 Computer Communications

Introduction

Historically, computing and telecommunications were viewed as distinct technical entities. Now, with the advent of cheap computers, mobile devices and sophisticated computer networks, these have merged to provide a set of information resources and facilitaters which pervade almost all aspects of our lives. The course examines the *fundamental techniques* used to implement the sharing of information between computers, and applies them to all levels of communication, from the transmission of bits along physical connections to the distribution of computations over many processors.

Context

The course requires a general knowledge of how computers work and how humans communicate. A formal requirement is programming expertise. The course is a precursor for the final-year Computer Networking module which covers more recent developments and future trends in computer networking.

Syllabus

Introduction and overview information, time, space, protocols;

Information sharing information in a distributed system;

Time achieving synchronisation in a distributed system;

Space achieving connectivity in a distributed system;

Message broadcast networks characteristics, architectures, standards;

Message switching networks characteristics, architectures, standards;

Inter-networks characteristics, architectures, standards;

Case studies perhaps two reasinably sized examples;

Real world issues Internet, OSI, social implications, local experience.

Activities

The method of delivery is two lectures per week. There are no formal tutorials.

Assessed Coursework

There is a piece of assessed coursework to implement components of an internationally standard connection-oriented protocol during the second half of the term; a set of short exercises in the first half of the term are used as a basis for this work. The coursework involves understanding a protocol specification, and then the design and implementation of software modules to complete a software system that simulates real-time communication with a trusted implementation over a channel with non-ideal properties.

The short exercises count for 20% of the overall coursework mark; they are promulgated at the beginning of week 3 of term, must be submitted at the beginning of week 5 of term, and are returned at the end of week 5 of term.

The main exercise counts for the remaining 80% of the overall coursework; it is promulgated during week 6 of term, must be submitted by the end of term, and is returned at the beginning of the next term.

References

(**) Andrew Tanenbaum *Computer Networks* (4th edition), Prentice Hall 2003.

(**) William Buchanan *Distributed Systems and Networks*, McGraw Hill, 2001.

(*) Gordon Brebner *Computers in Communication*, McGraw Hill, 1997. *Book out of print, but electronic copy will be available.*

(*) Douglas Comer *Computer Networks and Internets with Internet Applications* (3rd edition), Prentice Hall 2001.

(*) Fred Halsall *Data Communications, Computer Networks and Open Systems* (4th edition), Addison Wesley 1996.

(*) William Stallings *Data and Computer Communications* (6th edition) Prentice Hall, 2000.

8.6 Computer Design

NOTE: Owing to constraints on laboratory space, this module is limited to **100** students. Visiting students must obtain the permission of the course lecturer before taking this course, unless it has already been agreed before entry to the University. The places will be allotted first come, first served to degree students.

Description

This course is designed to provide an understanding of the different ways computers can be analysed and designed, starting with the basic logic elements you are familiar with (NAND, NOR *etc.*). The course does *not* (mostly) look at the differences between machines with different types of instruction set, nor does it cover design techniques for extracting maximum performance from computers – these aspects of computer hardware are often referred to as *computer architecture*, and are covered in the Inf3 course of that name. The issues and techniques covered in the Computer Design course are relevant to the design of all computers, regardless of their particular architecture.

The course splits fairly well into three sections. By the end of the first section ($\sim$6 lectures) you should be able to design combinational and sequential logic blocks using a variety of design techniques and implementation methods, be able to choose suitably between the various design and implementation options, and be able to analyse other people's designs; by the end of the second section ($\sim$6 lectures), you should be able to analyse and design systems of the complexity of a simple CPU or I/O controller; and by the end of the course, you should be able to analyse and design at the level of a computer capable of executing assembly code and performing I/O.

Syllabus

Introduction Levels of abstraction in computer design.

Logic Design *Combinational logic design:* **1** & **0** points, maxterms, minterms, canonical forms, Karnaugh maps, implicants, prime implicants, essential prime implicants, hazards, implementations (SSI, MSI, programmable logic). *Sequential logic design:* **SR** latch, flip-flops: level- and edge-triggerred, **D**- and **JK**-type, Huffman model, Moore and Mealy machines, sequential machine design procedure, examples, implementations (SSI, MSI, programmable logic).

Processor Design Data path and control. *Fixed program controllers:* example and design procedure. *Instruction set processors:* data path design, simple control, microprogrammed control, pipelining. *ALU design:* addition — ripple carry and look ahead adders, negative numbers & subtraction. Multiplication — sequential multiplier, modification for 2's complement, combinational multiplier, division. *Floating point numbers:* fp addition, multiply and divide, implementations.

Memory Design Byte vs. word addressing, memory system design, error detection and correction.

I/O Design I/O controller design. Connection of I/O controllers to CPU, synchronization of I/O and CPU — polling, interrupts. Direct Memory Access — bus arbitration, DMA controller implementation. I/O processors. Synchronous and asynchronous buses

Assessed Coursework

The coursework mark is based entirely on the hardware lab practicals – from 2 to 5 one afternoon a week from week 2 to week 9 of the first semester. The practicals in the hardware lab are designed to give you familiarity with design techniques covered in the course. Students work in pairs on the first two exercises and individually on the third of three exercises.

References

(**) V. C. Hamacher, Z. G. Vranesic & S. G. Zaky *Computer Organization,* fifth edition, McGraw-Hill, 2001. Covers almost all the syllabus (and more). Similar in approach to the lectures.

(**) D. A. Patterson & J. L. Hennessy *Computer Organization & Design: The Hardware/Software Interface,* second edition, Morgan Kaufmann, 1998. A different view of most of the material in the syllabus, and a lot of other interesting stuff.

(*) M. M. Mano *Digital Design,* second edition, Prentice-Hall, 1991. A very good book on logic design, with much more coverage of that part of the syllabus than the previous two books.

(*) A. S. Tanenbaum *Structured Computer Organization,* fourth edition Prentice-Hall, 1999. More a CS2 level book, but worth referring to.

8.7 Computer Security

NOTE

In the current session 04–05, Computer Security is listed as a level 10 course so that it may be taken as a CS4 course. However, it is primarily a CS3 course – no special permissions are required to take it, as would normally be needed for level 10 courses. Note that (i) in the next session 05–06 Computer Security will be a level 9 course, so will *not* be available to next year's CS4; and (ii) since it is a level 10 course this year, the coursework may be slightly more challenging than other CS3 courses.

Description

Computer Security is concerned with the protection of computer systems and their data from threats which may compromise integrity, availability, or confidentiality; the focus is on threats of a malicious nature rather than accidental. This course aims to give a broad understanding of computer security. Topics range from security risks, attacks, prevention and defence, through some current technology solutions, and down to formal approaches to validating security protocols and the mathematical principles underlying cryptography and cryptographic algorithms.

Syllabus

Introduction and background. Risks and attacks: to privacy (theft, surveillance); integrity (fraud); availability (vandalism, denial of service). Additional security properties: authentication, accountability.

Cryptography: basic functional foundations. Symmetric algorithms, for example: DES, Rijndael, RC4

Public key cryptography. Algorithms including RSA, ElGamal. Hash functions, including SHA-1. Digital signatures and certificates.

Authentication: mechanisms and attacks. Protocols for authentication and key exchange, including Needham-Schroeder, Otway-Rees, Kerberos, Diffie-Hellman.

Formal approaches, including Burrows-Abadi-Needham logic for authentication and its application to security protocol analysis.

Malicious code and network defences: Trojan horses, viruses and worms, attacks on faulty code. Auditing, intrusion detection, alarms and honey pots.

Security engineering: security policy models, multi-level systems. Secure kernels and trusted computing bases. Anatomy of attacks, risk assessment, attack trees.

Present internet technologies, for example: PGP, SSL, SSH, SMIME, DNSSEC, IPsec, firewalls and VPNs. The Java Security Model and security programming in Java.

Copyright protection. Secure hardware and tamper resistance. Steganography and covert communication. Anonymity.

Security futures, real-world issues. Topics chosen from: web security, e-commerce and e-cash; legalities; export control, key escrow; information warfare and cyber terrorism; human factors. Recent research areas.

Assessed Coursework

Two exercises, one involving security protocol analysis, and the other involving implementing a simple application-level security feature using Java's security APIs.

References

(*) Ross Anderson, *Security Engineering*, John Wiley & Sons, 2001

(*) Dieter Gollman, *Computer Security*, John Wiley & Sons, 1999

(*) Nigel Smart, *Cryptography: An Introduction*, McGraw-Hill, 2003

(*) John Viega and Gary McGraw, *Building Secure Software: How to Avoid Security Problems the Right Way*, Addison-Wesley, 2003

8.8 CS/SE Individual Practical

Description

The Individual Practical exposes students to the problems that arise with the design and implementation of large scale computer systems, and to methods of coping with such problems. Students will gain experience in how to:

- Design clearly and coherently structured systems
- Choose the appropriate means of implementation
- Discover and use relevant information
- Schedule their work load
- Present their work in a clear and concise way.

Syllabus

The project is structured into two parts:

1. Becoming familiar with the problem area, refining the specification of the system, undertaking some small programming tasks that are relevant to the problem in hand and providing a design for the system together with some justification for the design.
2. Providing an implementation for the system together with supporting documentation and analysis of the implementation.

This year the project will involve the design of a peer-to-peer filesystem. The implementation language will be Java.

Assessed Coursework

Project assessment is completely based on submitted documents and Java code relating to the project parts outlined above.

8.9 Database Systems

Description

Databases are essential to maintaining the information base in almost all modern business enterprises and to electronic commerce. They are also becoming increasingly important as a fundamental tool in much scientific research. Some knowledge of databases is now essential in any of these areas. The study of databases draws on several areas of computer science: logic, algorithms, programming languages and operating systems.

This module is an introduction to the principles underlying the design and implementation of databases and database management systems. It will cover the languages that have been developed for relational databases, their implementation and optimisation. It will also introduce some recent developments in databases including object-oriented, object-relational systems, semistructured data and the relationship between databases and XML. The bare essentials of transaction processing will also be covered.

Students intending to take this option should note that that it requires basic knowledge of logic and theory of computation.

Syllabus

The main topics covered – not necessarily in the order given – are as follows:

- Relational Model and Relational Algebra
- SQL and SQL DDL
- Designing databases: E-R diagrams and functional dependencies
- Datalog, recursive queries and graph queries
- Comprehensions and OQL
- XML: types and languages for XML
- HTML and PHP programming
- Query optimization and indexing
- Implementation of relational operations
- Transaction processing, concurrency and serialization

Assessed Coursework

There will be a series of homeworks and practical exercises.

References

(*) Raghu Ramakrishnan and Johannes Gehrke *Database Management Systems* (Third Edition) McGraw-Hill 2002. Please note that all database textbooks are expensive. At the time (Oct '02) of writing, this was one of the cheapest database textbooks.

8.10 Enterprise Computing

Description

Enterprise computing is the name given to distributed computing as practised in medium-sized or large organisations where the need to share data between physically-distributed sites is the primary motivator for the creation of a distributed system. The course is biased towards the acquisition of practical skills rather than an investigation of the theoretical limitations of distributed systems. The aim is to treat the dominant relevant technologies in depth rather than to give a more superficial survey of a larger number of technologies. The technologies studied are based on XML (the eXtensible Markup Language) as a data representation language and Java as a companion programming language for distributed programming.

Syllabus

The syllabus below is provisional; deviations may occur and will be documented in the Lecture Log for the module.

- The structure of an enterprise computing system. Basic definitions and concepts.
- Using XML (the eXtensible Markup Language) to represent data.
- Validating XML with DTDs (Document Type Definitions) and Schemas.
- Parsing XML. Document Object Model (DOM) parsers. Simple API for XML (SAX) parsers.
- Java's distributed computing technology. Remote method invocation (RMI). Server-side computing.
- Java enterprise computing technologies. Servlets, Java Server Pages, Enterprise Java Beans.
- Java Web Services.

Assessed Coursework

There will be two pieces of assessed practical work.

References

(*) XML: How to program. Deitel and Associates. Published by Prentice-Hall.

(*) Java Web Services tutorial. Published by Addison-Wesley. Also available on-line at `http://java.sun.com/webservices/docs/1.0/tutorial/`

(*) Java Web Services for Experienced Programmers. Deitel and Associates. Published by Prentice-Hall.

8.11 Functional Programming and Specification

Description

The course has two aims. The first is to provide an introduction to programming in Standard ML including the use of the facilities it offers for structuring programs into modules. The clean functional programming paradigm represented by ML is quite different from the imperative object-oriented paradigm represented by Java, making it more suitable for many applications. The second aim is to provide an introduction to formal methods for specification and development of programs, using the Extended ML specification framework as a vehicle. Simple proofs of properties of functions are interwoven with the first part of the course to link it with the second part.

Syllabus

- Functional programming in Standard ML: The functional paradigm. Polymorphic types, static typing and type inference. Recursion and induction. List processing. Higher-order functions. Eager and lazy evaluation. Imperative features. Signatures, structures, functors. Module hierarchy, sharing and data abstraction.
- Specification and formal program development in Extended ML: Specification of ML functions and modules. The EML specification language. Proving that a program is correct with respect to a specification of its intended behaviour. Refinement of specifications. Formal development of ML programs from EML specifications by modular decomposition and stepwise refinement.

Assessed Coursework

Four written assignments, weighted equally, from which students choose three: two on ML programming; one on the ML module system; one on specification and proof.

References

(***) L. Paulson. ML for the Working Programmer, second edition. Cambridge University Press, 1996. Currently £23.95 in paperback.

(***) D. Sannella. Formal program development in Extended ML for the working programmer.

(**) R. Harper. Programming in Standard ML. Carnegie Mellon University. Soon to be a book. Available on-line.

(**) S. Gilmore. Programming in Standard ML'97: A tutorial introduction. Edinburgh report ECS-LFCS-97-364, 1997.

(*) J. Ullman. Elements of ML Programming, second edition. Prentice-Hall, 1997.

8.12 Language Semantics and Implementation

Description

The aim of the course is to present a unified view of programming language semantics and implementation, based upon the linked notions of structured operational semantics and abstract machines. Different styles of languages (such as declarative and object oriented) will be treated. Lecture notes will be available on the module web page.

Syllabus

Dynamic Language Semantics Semantic rules as an inference system; treatment of variable assignment, iteration, scope, function declaration and application, parameter passing, records, recursion.

Static Semantics Semantic rules for type checking as an inference system.

Abstract Machines The SMC machine for an imperative while language

Assessed Coursework

The coursework consists mostly of paper and pencil exercises, but there may be programming exercises.

References

(*) M. Hennessy, *The Semantics of Programming Languages*, Wiley, 1990.

(*) G. D. Plotkin, (Parts of) *A Structural Approach to Operational Semantics*, Aarhus Research Report.

(***) A. Pitts, *Semantics of Programming Languages*, Lecture Notes, University of Cambridge, **http://www.cl.cam.ac.uk/Teaching/2000/Semantics/**

8.13 Operating Systems

Overview

This course provides an introduction to the design and implementation of general purpose multi-tasking operating systems. It concentrates on the kernel aspects of such systems with the emphasis being on concepts which lead to practical implementations. Throughout the course reference is made to a number of significant actual operating systems (Linux, Windows, etc.) to illustrate real implementations.

Syllabus

Process management The process concept, synchronisation, mutual exclusion, semaphores and monitors. Threads. Inter-process communication.

Resource Allocation Deadlock prevention, avoidance and detection.

The OS Kernel Micro and Monolithic kernels. Multi-tasking, privilege, interrupt handling. System and user processes. System calls.

Memory Management Description of problems of allocation, protection and sharing. Virtual $\rightarrow$ Physical memory mapping schemes. Segmented paged virtual memory. Paging control, replacement algorithms; the working set model. Sharing code and data.

Time Management CPU scheduling algorithms. Real-time scheduling. Disc access scheduling.

File Management Naming and Directory schemes. Disc space allocation. File protection and access control. System security.

Assessed Coursework

The coursework is in two parts, both of four weeks duration, due in weeks 7 and 11 of the semester. The first will consist of a system programming related exercise and the second will be an essay on some topic covered only briefly in the lectures.

References

(**) W. Stallings *Operating Systems, Internals and Design Principles* (4th edition), Prentice-Hall, 2001. The course will follow this book part of the time, and access is highly recommended. However, it is not essential.

(**) A. Silberschatz and P. Galvin *Operating Systems Concepts* (5th edition), Addison-Wesley, 1998.

(**) Harvey M. Deitel, Paul J. Deitel and David R. Choffnes *Operating Systems* (3rd edition), Prentice-Hall, 2004. This is also a well recommended book.

(*) Gary Nutt *Operating Systems - A Modern Perpective* (2nd edition), Addison-Wesley, 2000.

(*) D.A. Solomon and M.E. Russinovich *Inside Microsoft Windows 2000* (3rd Edition), Microsoft Press, 2000.

8.14 Professional Issues

There are many commercial, engineering and professional issues, complementary to the necessary scientific knowledge and technical skills, that impinge on the work of the computing professional. The Professional Issues course aims to provide a general awareness of these issues and to cover some of them in depth. The course will mostly involve directed reading but there will be some lectures from members of staff and visitors.

Syllabus

Personal Attributes: study skills, personal development, interpersonal skills; employers' views and expectations of graduates; study skills, writing skills, presentation skills.

The Computing Profession: professional bodies; codes of conduct and practice.

Social and Ethical Issues: security, privacy, software ownership

Legal Issues: legal and regulatory frameworks; software contracts and liability; intellectual property, copyright and patents; computer misuse, data protection; health and safety.

Commercial Issues: organisational structures; finance, accounting, audit; resource management.

Computing Projects: design, prototype and product; product development cycle; marketing and market research; project management and team working.

Assessment

The examination will involve essay-style questions. The assessed coursework will also be an essay to allow practice in essay writing. All essays will be assessed on their quality of English as well as content. Note that this course has a non-standard coursework weighting (15% coursework, 85% exam).

References

(***) *Professional Issues in Software Engineering*, 3rd edition, Bott, Coleman, Eaton and Rowland, Pitman, 2001

(***) *Scientists Must Write*, R. Barrass, Routledge, 2002

(**) *Computer Ethics and Professional Responsibility*, Bynum and Rogerson (eds), Blackwell, 2004

(**) *The Essence of Professional Issues in Computing*, R. Ayres, Prentice Hall, 1999

(*) *Law and the Internet*, L. Edwards and C. Waelde, Hart, 1997

(*) *Zen and the Art of Motorcycle Maintenance*, Robert M Pirsig, Corgi, 1974/1999

8.15 Software Engineering with Objects and Components 1

Description

This course provides an introduction to the design and implementation of software systems using object oriented techniques. The techniques we consider are oriented to creating component based designs. The course will review basic object oriented techniques and how they support the creation of component based designs. We also consider the high level modelling of systems as a means of supporting the Software Engineering process. Here we study the Unified Modelling Language (UML), which provides programming language independent notations for design.

Context. *the following prerequisites are covered in CS2:* Basic knowledge of Software Engineering and Object-Orientation concepts; Java programming.

The Software Engineering process. We briefly consider how taking objects and components as a central organising theme influences the Software Engineering process. A number of case studies of "classic" software-related failures will be used as illustrative examples throughout the course. We also briefly consider the arguments for and against insisting upon *any* specific approach to Software Engineering, and those for and against object orientation in particular.

- SOFTWARE DEVELOPMENT PROCESS:
  - Requirements analysis
  - Specification
  - Validation
  - Static verification (ie, Testing; Formal verification is studied in the module Functional Programming and Specification)
  - Design and Implementation: including software architecture, and software integration techniques
  - Maintenance and evolution
- PROJECT MANAGEMENT AND PLANNING

Both the development process and project management and planning will be illustrated and utilised throughout the practical work

Elements of UML. Here we outline the main phases of an object oriented development: analysis, design and implementation. Each of these phases is supported by various diagrammatic notations embodied in UML. We consider a small subset of the full collection.

- ANALYSIS: A brief introduction to the use-case diagram as a means of analysing the external behaviour of systems from various viewpoints.
- DESIGN: Here there are various diagrams aimed at capturing the static and dynamic structure of systems. We will study:

  - Class diagrams: these describe the static structural relationship between object classes.
  - Behaviour diagrams, these include:
    * state diagrams,
    * activity diagrams,
    * sequence diagrams,
    * collaboration diagrams.

- IMPLEMENTATION: We provide a brief overview of the implementation process, considering:
  - component diagrams, and
  - deployment diagrams

Assessed Coursework

Exercises, reports and student presentations taken as group work in the tutorials.

References

(***) Bennett, Skelton and Lunn *UML*, Schaum's Outline Series, McGraw-Hill, London, 2001. Access to this text is a necessity.

(**) Sommerville *Software Engineering* (5th Edition), Addison-Wesley.

(**) Various useful Web resources are given on the course homepage.

8.16 System Design Project

The System Design Project is intended to give students practical experience of (a) building a large scale system (b) working as members of a team. The project involves applying and combining material from several courses to complete a complex design and implementation task. At the end of course each group demonstrates its implemented system and gives a formal presentation to an audience of the students, supervisors, and visitors from industry.

During this project students work in groups of about ten on the design and implementation of a complete system to solve some practical and useful problem. All groups perform the same task. This primarily involves software implementation but may potentially also involve hardware design and construction where this is relevant. Recent examples of projects include: an automated on-line supermarket; building webcam-based home and commercial security systems; constructing Mars and Lunar rovers controlled from an Earth-based web browser interface, etc.

Each group is provided with the same facilities. These include one or more PCs dedicated to them and other equipment depending on the particular project, for instance a webcam, a Lego robot construction kit, hardware prototyping kit, diagnostic equipment etc. They also have a small amount of money to spend in any way they choose on any extra items they feel might enhance their particular design. Project management software is also available to them.

A staff member is assigned to each group as a supervisor. The supervisor's task is to advise and provide feedback on the progress of the group during the project but not to provide technical support. Consultants from amongst the academic and support staff are made available to advise on aspects of the task such as management, specific pieces of software and hardware etc. Groups meet with their supervisors at least once a week. They also meet amongst themselves more frequently to plan and coordinate their activities.

Towards the end of the semester, a day is set aside for groups to demonstrate their implemented system and to give a formal presentation of it to an audience of the students, supervisors, and visitors from industry.

9 Descriptions of Courses and Projects – AI3

9.1 AI Large Practical

Why do we ask you to do a Large Practical such as this in the third year? There are several reasons. It provides you with a gentle introduction to the issues and requirements of the more demanding fourth-year project that is worth a third of the final year's marks. It gets you some experience of reading published papers and trying to figure out what the important content is. It requires you to think about how to write a report on a modest piece of scientific work: you have to explain what you did, and why, and what conclusions you reached, and why, and you have to do this clearly and convincingly. It compels you to write programs to investigate a certain question; you must write well-structured, well-documented programs because they too are acts of scientific communication. If you wrote programs which some reasonably intelligent reader, who was not necessarily a specialist in the topic, was unable to follow then there would be some doubt remaining as to whether your programs really implemented what you claimed to be investigating.

These are the main considerations that will be used to judge your work.

Description

Students are divided up into tutorial groups with a demonstrator allocated to each group. Tutorial groups will meet at least once a week at a time of mutual convenience. The LP coordinator and demonstrators will assist students with the planning and organisation of their LP.

Assessment

No formal written examination; the assessment is based on practical work and a written report submitted at the end of the project period.

9.2 Automated Reasoning

Description

The aim of the module is to describe how reasoning can be automated. Major emphases are on: how knowledge can be represented using logic; how these representations can be used as the basis for reasoning and how these reasoning processes can be guided to a successful conclusion. Many of the examples are drawn from mathematics because this domain contains lots of challenging reasoning problems which can be succinctly stated.

Syllabus

The module combines an exposition of theory with the analysis of particular computer programs for reasoning. A few of the topics that will be looked at:

Logic Propositional, first order and higher order.

Reasoning Resolution, term rewriting.

Decision Procedures BDDs, Davis-Putnam proof procedure.

Unification and matching algorithms

Strategies for Search Control in Resolution

Term Rewriting Confluence and Termination.

Guiding search Current Edinburgh research on proof plans and rippling.

Automatic formalisation How formal problem representations can be extracted from informal ones.

Interactive theorem proving with Isabelle

Applications Uses of theorem proving to: reason automatically and interactively, teach mathematics, prove properties of computer programs, assist mathematicians, scientists and engineers.

Assessed Coursework

Practical exercises with theorem provers.

References

(*) A. Bundy: *The Computer Modelling of Mathematical Reasoning. Academic Press 1983*

(**) T. Nipkow, L. C. Paulson, and M. Wenzel. Isabelle/HOL: A Proof Assistant for Higher-Order Logic , Springer-Verlag, 2002.

Other resources are on the module web page.

9.3 Genetic Algorithms and Genetic Programming

Description

This module teaches you about genetic algorithms (GAs), genetic programming (GP) and other such evolutionary computing (EC) ideas based on the idea of solving problems through simulated evolution. These techniques are useful for searching very large spaces. For example, they can be used to search huge parameter spaces in engineering design and spaces of possible schedules in scheduling. However, they can also be used to search for rules and rule sets, for data mining, for good feed-forward or recurrent neural nets and so on. The idea of evolving, rather than designing, algorithms and controllers is especially appealing in AI. The module will also introduce other biologically inspired algorithms, particularly Ant Colony Optimisation methods.

Syllabus

- Basics of biological evolution: Darwin, DNA, etc.
- Basics of GAs: selection, recombination and mutation. Choices of algorithm: (mu,lambda), (mu+lambda), steady-state, CHC, etc. Linkage and epistasis. The standard test functions. Fitness and objective functions: scaling, windowing etc.
- Representational issues: binary, integer and real-valued encodings; permutation-based encodings.
- Operator issues: different types of crossover and mutation, of selection and replacement. Inversion and other operators.
- Constraint satisfaction: penalty-function and other methods; repair and write-back; feasibility issues.
- Experimental issues: design and analysis of sets of experiments by t-tests, F-tests, bootstrap tests etc.
- Some theory: the schema theorem and its flaws; selection takeover times; optimal mutation rates; other approaches to providing a theoretical basis for studying GA issues.
- Rival methods: hill-climbing, simulated annealing, population-based incremental learning, tabu search, etc. Hybrid/memetic algorithms.
- Multiple-solutions methods: crowding, niching; island and cellular models.
- Multi-objective methods: Pareto optimisation; dominance selection; VEGA; COMOGA.
- Genetic programming: functions and terminals, S-expressions; parsimony; fitness issues; ADFs.
- Evolving rules and rule-sets. SAMUEL and related methods. Classifier systems: the Pittsburgh and Michigan approaches. Credit allocation: bucket-brigade and profit-sharing. Hierarchic classifier systems.
- Genetic planning: evolving plans, evolving heuristics, evolving planners, optimising plans.
- Ant Colony Optimization: Basic method for the TSP, local search, application to bin packing.

- Applications: engineering optimisation; scheduling and timetabling; data-mining; neural net design; etc.
- Some further ideas: co-evolution; evolvable hardware; multi-level GAs; polyploid GAs.

Assessed Coursework

There will be one assessed practical project.

References

(***) M. Mitchell: *An Introduction to Genetic Algorithms. MIT Press, 1996.*

(*) W. Banzhaf, P. Nordin, R. E. Keller, F. D. Francone: *Genetic Programming: An Introduction. Morgan Kaufmann, 1988.*

(*) E. Bonabeau, M. Dorigo, G. Theraulez: *Swarm Intelligence: From Natural to Artificial Systems. Oxford University Press, 1999.*

9.4 Introduction to Cognitive Science

Description

This course is principally aimed at third-year undergraduates in the School of Informatics, and at third- and fourth-year undergraduates in the School of Philosophy, Psychology and Language Sciences. It is intended to give them an introduction to some of the current research issues in Cognitive Science, together with examples of the different research paradigms by which they might be investigated. Cognitive Science is an inherently interdisciplinary enterprise and this is reflected in the course, which brings together issues relating to the disciplines of Cognitive Psychology, Linguistics, Neuroscience, Philosophy and Artificial Intelligence.
Specific aims: Knowledge: broad knowledge of previous work in field; understanding of main issues; appreciation of difficulties. Methodology: understanding of diversity of methods appropriate in the field. State of the Art: awareness of emerging trends on cognitive research, and limitations of current solutions.

Syllabus

- The scope of Cognitive Science.
- Vision and imagery.
- Speech perception and production.
- Reading and writing.
- Memory, thinking and learning.
- Communication and co-ordination.
- Reasoning and problem solving, with and without tools.
- Rationality and evolutionary psychology.
- Emotion and consciousness.

Examination

The examination for this module will not be in a standard format. Instead, it will require two questions to be answered. At least one of these will require an essay-style response, based on the student's personal study, as developed via the tutorials. Suggestions for titles will be provided in the early part of the course and as different topics are introduced.

Assessed Coursework

The coursework comprises an individual report on independent reading for the task-based tutorials, due in week 6, and carrying 10% out of the 25% for coursework; and a group report on the task undertaken in task-based tutorials, due in week 10, and carrying the remaining 15% contribution.

9.5 Introduction to Computational Linguistics

Description

The course provides: 1) an introduction to the theory and practice of computational approaches to natural language processing. Topics include formal models of natural language; standard approaches to processing (morphological analysis, part-of-speech tagging, syntactic parsing and chunking) and word sense disambiguation, 2) exposure to techniques and tools used to develop practical, robust systems that can communicate with users, 3) experience in programming with Python, 4) insight into many open research problems in natural language processing, e.g., summarization, machine translation, information extraction, question answering, response generation, and statistical corpus analysis.

Syllabus

- Overview of Computational Linguistics.
- Introduction to scripting for text processing.
- Lexical and morphological structure.
- Using Regular Expressions.
- Tokenization, stemming and part-of-speech tagging.
- Context free grammars, verb subcategorization and parsing.
- Partial parsing and chunking.
- Word sense disambiguation.

Assessed Coursework

Two assignments throughout the course. These will require some programming in Python.

References

(***) D. Jurafsky, J. H. Martin: *Speech and Language Processing. Prentice-Hall, 2000.*

(*) M. Lutz, D. Ascher: *Learning Python. O'Reilly, 1999.*

(*) C. Manning, H. Schutze: *Foundations of Statistical Natural Language Processing. MIT Press, 1999*

9.6 Introduction to Vision and Robotics

Description

Robotics and Vision applies AI techniques to the problems of making devices capable of interacting with the physical world. This includes moving around in the world (mobile robotics), moving things in the world (manipulation robotics), acquiring information by direct sensing of the world (e.g. machine vision) and, importantly, closing the loop by using sensing to control movement. Applying AI in this context poses certain problems, and sets certain limitations, which have important effects on the general software and hardware architectures. For example, a robot with legs must be able to correct detected imbalances before it falls over, and a robot which has to look left and right before crossing the road must be able to identify approaching hazards before it gets run over. These constraints become much more serious if the robot is required to carry both its own power supply and its own brain along with it. This module introduces the basic concepts and methods in these areas, and serves as an introduction to the more advanced robotics and vision modules.

Syllabus

The issues addressed will include the following:

- Applications of robotics and vision; the nature of the problems to be solved; historical overview and current state of the art.
- Robot actuators and sensors. Parallels to biological systems.
- Robot control: Open-loop, feed-forward and feedback; PID (proportional integral differential) control.
- Image formation, transduction and simple processing; thresholding, filtering and classification methods for extracting object information from an image.
- Active vision and attention. Range sensing.
- Sensors for self monitoring.
- General approaches and architectures. Classical vs. behaviour-based robotics. Wider issues and implications of robot research.

Assessed Coursework

A report on a practical project and a research review exercise.

References

(***) Russell & Norvig: Chapters 24 and 25 in *Artificial Intelligence: A Modern Approach. Prentice Hall International Editions, 1995.*

(**) R. R. Murphy: *Introduction to AI robotics, A Bradford Book. MIT Press, 2000.*

(**) Ramesh Jain, Rangachar Kasturi and Brian G. Schunck: *Machine vision*. McGraw-Hill, 1995.

(*) Nevins and Whitney: 'Computer Controlled Assembly', in *Scientific American*, Feb 1978.

(*) Phillip J. McKerrow: *Introduction to Robotics*. Addison Wesley, 1991.

9.7 Knowledge Representation and Engineering

This is the description submitted for approval to the Board of Studies. At the time of printing, the Board had not yet met. Any changes will be notified.

Description

The module looks at languages and techniques for encoding, sharing, and reasoning with information represented in symbolic fashion. It also covers methodologies important to the development and application of knowledge based systems in the context of the internet.

Syllabus

- The notion of symbolic representation
- Logic as a representation language and deduction as inference
- Modal logic and associated decision procedures
- Reason maintenance systems
- Distributed constraints and distributed search
- Knowledge acquisition methodologies
- Ontologies as a basis for structuring knowledge bases
- Distributed multi-agent architectures
- Reasoning about intentional agents

Assessed Coursework

Two practical exercises.

References

Course notes will be made available.

(**) Ronald Fagan et al., *Reasoning about knowledge*, MIT Press.