University of Edinburgh

School of Informatics

COMPUTER SCIENCE 3

and

SOFTWARE ENGINEERING 3

and

ARTIFICIAL INTELLIGENCE 3

Course Guide 2003–2004

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

October 2003

Contents

Preface

This document describes the organisation of CS3 and AI3, 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.

20 October 2003: Mention the module-specific newsgroups.

14 October 2003: The degree requirements for AI/M3 were mistaken; for students entering AI/M3 in 2003, Algorithms and Data Structures is no longer a compulsory part of the AI/M degree.

1 Introduction

1.1 Computer Science classes

Computer Science offers the following modules and projects for third year students: 12 technical modules, a module on Professional Issues, and two major projects. Each technical module comprises about 18 lectures and is assessed by examination and coursework. The Professional Issues module is assessed by examination. The coursework for the Computer Design module involves practical work in a hardware systems laboratory. All students take a subset of the modules and projects available, subject to the constraints specified in Section 3 below. Details of all modules and major projects are in Section 8.

The *Introductory Meeting* for all students taking Computer Science modules (including joint degree students) is held at 10:00 on Monday 6 October 2003, in JCMB Lecture Theatre A. There may be a small number of joint degree students who 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. The course organiser may call a short meeting to answer questions from students unable to attend the Computer Science Introductory Meeting.

1.2 Artificial Intelligence classes

Artificial Intelligence offers another 7 technical modules, assessed as for CS modules, and a Large Practical, corresponding to the CS3 major project. The available choice of modules varies between the different joint degrees, and is specified below.

The *Introductory Meeting* for all students taking Artificial Intelligence modules is held at 13:30 on Monday 6 October 2003, in room AT 3.04. As with CS, there may be a small number of joint degree students who 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.

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 (room changing)) 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 joint degrees) and Artificial Intelligence (in AT) joint degrees.

The staff teaching the third year modules are:

Term 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).
- Computer Science Individual Project: Phil Wadler (`wadler`, 1616); this project continues in term 2.
- Software Engineering Individual Project: Phil Wadler (`wadler`, 1616); this project continues in term 2.
- Operating Systems: Julian Bradfield (`jcb+os`, 2610).
- Software Engineering with Objects and Components 1: Stuart Anderson (`soa`, 1610);

Term 1 – AI

- AI Programming in Java: Judy Robertson (`judyr`, 2BP–G05).
- Automated Reasoning: Jacques Fleuriot (`jdf`, AT–2.02).
- Introduction to Computational Linguistics: Ewan Klein (`ewan`, 2BP–G16).
- Introduction to Vision and Robotics: Barbara Webb (`bwebb`, JCMB–1107).
- Learning from Data 1: David Barber (`dbarber@anc.ed.ac.uk`, FH–B2).

Term 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`, t.b.a.).
- Database Systems: Peter Buneman (`opb`, 1511).
- Enterprise Computing: Stephen Gilmore (`stg`, 1608).
- Language Semantics & Implementation: Gordon Plotkin (`gdp`, 2620) and Colin Stirling (`cps`, 3422).
- Professional Issues: John Butler (`jhb`, 2504).

Term 2 – AI

- Genetic Algorithms and Genetic Programming: John Levine (`johnl`, AT–3.08).
- Introduction to Cognitive Science: Jon Oberlander (`jon`, 6BP–G15).
- Knowledge Engineering: Chris Malcolm (`cam`, JCMB–2107B).

Term 3 – CS

- Professional Issues: John Butler & Law School (`jhb`, 2504).
- Systems Design Project: David Rees (`djr`, 1510).

All year – AI

- AI Large Practical (AI/CS, AI/M, AI/SE): Qiang Shen (`qiangs`, AT–2.06).
- AI Large Practical (AI/Psy, AI/Ling): Mark Steedman (`steedman`, 2BP–2R14).

3 Degrees and Degree Requirements

In the first subsection, the requirements for the Computer Science component of single and joint degrees are described. The second subsection describes the requirements for the Artificial Intelligence component of joint degrees. The courses that joint honours students take outside the School of Informatics are not defined in this document, and all such joint honours students should consult the course documentation produced by their other school. Some timetable clashes for joint honours students are unavoidable. You are strongly advised to select courses so as to avoid conflicts.

3.1 Computer Science requirements

All students who take a CS degree or joint CS degree take a subset of the CS3 modules and projects, subject to the rules presented in this section. It is *your* responsibility to make sure that your module selection is correct. Before the end of week 1 of terms 1 and 2 you are required to submit a *binding* choice of the technical modules you intend to take during that term.

Joint honours students taking the System Design Project are *strongly* advised to plan their timetable so as to avoid other major commitments in the first half of the summer term; such commitments might include modules that are examined at the beginning of the summer term (e.g., in AI4). The System Design Project is a team effort and team members are expected to work full-time on the project, with the exception of attendance at Professional Issues lectures (some joint degree students will also attend a small number of required classes in their joint degree department).

At the discretion of the Head of School, and subject to timetabling constraints, it may be possible to vary the rules described below. Discuss this with your director of studies to begin with.

In order to simplify the description of the requirements for each of the courses, certain subsets of technical modules are identified and assigned colours.

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

There are also two major projects. The System Design Project is a group project, and is uniform over the various degrees. The Individual Project comes in two flavours: Computer Science (CS) and Software Engineering (SE). All references to the Individual Project (IP) below should be taken to mean IP-CS for students registered for CS degrees and IP-SE for students registered for SE degrees. All references to "both major projects" should be taken to mean the System Design Project together with the appropriate flavour of Individual Project. It is not permitted for students to count both IP-CS and IP-SE for credit towards a degree.

3.1.1 Computer Science Honours Course

Students take eight CS3 technical modules, both CS3 major projects, and Professional Issues. The choice must include at least one red, at least one yellow and at least one blue module.

3.1.2 Computer Science Ordinary Course

Students take six CS3 technical modules, one CS3 major project, and Professional Issues. All students must take 3 technical modules per term. There are no restrictions on the choice from within the Computer Science 3 syllabus.

3.1.3 Software Engineering Honours Course (B.Eng.)

Students take eight CS3 technical modules, both major projects, and Professional Issues. The choice must include at least one green module and at least one blue module.

3.1.4 Artificial Intelligence and Computer Science 3

Students take four CS3 technical modules, the System Design Project, and Professional Issues. The choice must include at least one red module.

3.1.5 Artificial Intelligence and Computer Science Ordinary Course

Students take three CS3 technical modules, one major project, and Professional Issues. There are no restrictions on the choice from within the CS3 syllabus. The project may be from AI or CS. (Thus, in total, students take six technical modules evenly distributed between AI and CS, one project from either AI or CS, and Professional Issues.)

3.1.6 Artificial Intelligence and Software Engineering 3

Students take four CS3 technical modules, the System Design Project, and Professional Issues. The choice must include at least one green module.

3.1.7 Computer Science and Electronics 3

Students take four CS3 technical modules, a CS3 major project, and Professional Issues. The choice must include at least two yellow modules.

3.1.8 Computer Science and Management Science 3

Students take four CS3 technical modules, a CS3 major project, and Professional Issues. There are no restrictions on the choice from within the Computer Science 3 syllabus. Students must take Management Science and Operational Planning and Management Science and Information Systems, if they have not already done so. They may not take Quantitative Research Methods.[1]

[1] With permission of the Head of School and of Business Studies, students may do two more CS3 modules and one fewer Business Studies half-course.

3.1.9 Computer Science and Mathematics 3

Students take four CS3 technical modules, a CS3 major project, and Professional Issues. The choice must include at least two red modules.

3.1.10 Computer Science and Physics 3

Students take four CS3 technical modules, a CS3 major project, and Professional Issues. The choice must include at least one red and at least one yellow module.

3.1.11 Electronics and Software Engineering 3

Students take four CS3 technical modules, a CS3 major project, and Professional Issues. The choice must include at least one yellow module and at least one green module.

3.2 Artificial Intelligence requirements

All students who take a joint AI degree take a subset of the AI3 modules and projects, subject to the rules presented in this section. It is *your* responsibility to make sure that your module selection is correct. Before the end of week 1 of terms 1 and 2 you are required to submit a *binding* choice of the technical modules you intend to take during that term.

3.2.1 Artificial Intelligence and Computer Science 3

Students take 4 modules in AI and the Large Practical.

3.2.2 Artificial Intelligence and Software Engineering 3

Students take 4 modules in AI and the Large Practical.

3.2.3 Artificial Intelligence and Computer Science Ordinary Course

Students take 3 lecture modules in AI and take the AI Large Practical only if they are not taking the CS group project.

3.2.4 Artificial Intelligence and Linguistics 3

Students take Introduction to Computational Linguistics and another 2 AI lecture modules, as well as the Large Practical.

3.2.5 Artificial Intelligence and Mathematics 3

AI/M3 students take 4 modules from those available in AI3 and CS3, as well as a Large Practical. During AI/M3 and AI/M4 the Automated Reasoning module is compulsory. This means that you have to have done it by the end of your 4th year; however, you may choose to do it either in AI/M3 or AI/M4.

3.2.6 Artificial Intelligence and Psychology 3

Students take four lecture modules in AI, which must include Introduction to Cognitive Science, and also take the Large Practical.

3.2.7 Artificial Intelligence with Psychology 3

Students take five lecture modules in AI, which must include Introduction to Cognitive Science, and also take the Large Practical.

4 Assessment

4.1 Coursework

Each lecture module has coursework, which accounts for 25% of the assessment for the module.

A lecture module is nominally assigned 120 hours of work, taking a full year's work to be thirty 40-hour weeks. In practice, this probably means around 60-90 hours of time outside lectures, thus you should expect to spend a few hours per week per module 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 re-submitted to the ITO (room JCMB–1502 for CS, and AT–?? for AI) in June so that it is available for the examiners to inspect during the assessment process. The deadline for resubmission is the end of week 5 of term 3.

Plagiarism You *must* read and follow both the University's (revised, 1999) policy on plagiarism (section 5 of the Degree Examination Regulations), 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.

CS3 and AI3 follow the College of Science and Engineering defined 'normal practice' for late submission:

> Late coursework will not be accepted without good reason; the work will be recorded as late and a mark of zero given. Any extension to the coursework deadline for a particular student must be approved by the Course Organizer, and the reason for the extension must be recorded.

'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 module is examined by a separate one-and-a-half hour examination paper in term 3 (hereinafter referred to as "the June examinations", though there may in fact be examinations before the start of June). Usually, each of these examinations contains three questions; each candidate is required to attempt two of these. Some modules may have a different

format; the module lecturer will advise you if this is the case. The Professional Issues module is examined by a one hour examination in May.

For Honours courses (including joint 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 June 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. In Computer Science Honours, normally, the mark for Computer Science 3 will carry the same weight as the mark for Computer Science 4 in determining the degree classification.

The final grade for the year is obtained by combining the marks on examinations, coursework and CS major projects using the following weightings:

Units of assessment	Points
Examination and coursework for each technical module	12
Examination for Professional Issues module	4
Each CS major project	10
AI large practical	12

Coursework and examination for each technical module form separate units of assessment; the weighting is 9 points for examination and 3 for coursework.

In the case of joint honours courses (including AI/CS), your overall mark is the average of your marks in the two subjects, weighted according to the time allocated to the courses you take. In most, but not all, joint degrees, this means that the two components have equal weight.[2] Your CS or SE mark is calculated according to the weightings given above; your AI mark is calculated separately, again using those weightings. The calculation of the mark in other subjects is the responsibility of the other school.

The Board of Examiners has the discretion to act in any way it sees fit; for example, it may take into account medical circumstances.

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 general, there is no requirement to pass a certain number of individual modules. However, in joint 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. In addition, failure on certain modules (notably the major practicals) may reduce the candidate's BCS exemptions (see below).

[2]Note: owing to the nominal weightings of Professional Issues and CS major projects, students doing joint CS degrees nominally do 62 credit units of work in the CS half of their degree. However, this is scaled to 60 credits (half a year) before being combined with the mark from the other component of the degree.

A candidate who fails to reach the Honours pass mark after the June 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 June, resit examinations for the Ordinary degree take place in September. 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 June, as the mark from the June 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, 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 will be labs in Appleton Tower, but these are not yet ready. 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 modules, 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 modules's web page) and make copies yourself. *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.dcs.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 modules also use their own newsgroups: these are named `eduni.inf.module.`*module-code*; for example the Operating Systems group is `eduni.inf.module.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".

Please do not abuse these facilities. The department 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 CS3/AI3 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 June 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 by the ITO. They can be found at `http://www.inf.ed.ac.uk/teaching/classes/aics3/timetable03.html`

7.1 Timetable during Week 1

In week 1, the following changes apply:

- On Monday at 10:00, the CS3 Introductory Lecture replaces Operating Systems in JCMB Lecture Theatre A.
- On Monday at 13:30, there is the AI3 introductory presentation in Appleton Tower room 3.04.
- On Monday at 15:00, there is an Introductory Lecture for Professional Issues in JCMB LTA.
- On Monday at 16:00, there is a Prizegiving and Reception in JCMB 2511.
- The Hardware Lab sessions for Computer Design do not run.

7.2 Notes on the Term 1 timetables

- The term lasts for ten weeks, but lectures and tutorials usually finish in week 9, at the discretion of staff. Week 10 is spent completing course work. There are variations, noted above, in the first week of term.
- The hardware laboratory sessions for the Computer Design module will be held in the afternoons, starting in week 2, and students will be allocated *one* session by the lecturer for that module. Due to timetabling constraints, joint CS & Electronics students usually attend the Friday afternoon slot.
- The Individual Project is in two parts, each occupying approximately four weeks of terms 1 and 2. There will be lectures in weeks 2 and 9 of term 1.

7.3 Notes on the Term 2 timetable

- The term lasts for ten weeks, but lectures and tutorials usually finish in week 9, at the discretion of staff. Week 10 is spent completing course work.
- The second part of Individual Project starts in week 2.
- Professional Issues lectures start towards the middle of the term; you will be informed via the CS3 alert message when the first lecture will take place.
- Student groups for the Professional Issues (and later the System Design Project) will be formed in week 3. The constitution of groups is a staff responsibility and it is not normally permitted for students to switch groups.

8 Descriptions of Modules 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 Major Projects

These projects aim to introduce you to the methods of coping with the problems that arise with the design and implementation of large scale computer systems. It is intended that students will gain experience in deciding 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 findings and implementations in a clear and concise way.

8.1.1 Computer Science Individual Project

This project gives students experience in developing a non-trivial system and providing some analysis of its behaviour. In particular:

Structure: You should pay careful attention to the structure of you proposed system ensuring that the design is sufficiently simple that it eases verification and analysis.

Verification: You will be asked to ensure that certain properties can be guaranteed of the system and will be asked to provide evidence that the system has these properties.

Analysis: You will be asked to provide an analysis of the timing behaviour of some parts of your system.

The central task is to design and develop a small client that allows peer-to-peer information sharing. The project is divided into two phases: the first is to generate a design, preliminary analysis of its behaviour and some exploratory code; the second is to provide an initial implementation of the system and provide some verification and analysis of the implementation. Java is the preferred language for most of the project but this is not compulsory.

The first phase of the project runs through term 1 and the second in term 2. There will be briefing meetings at the start of each phase, and a support team will be available to help with any problems. In addition, you are strongly encouraged to use the CS3 newsgroup `eduni.inf.module.ip` for discussion of the project. This allows you to share any questions with the largest possible audience, and can highlight any common difficulties with the work.

Assessment Project assessment is primarily based two deliverables. For each deliverable you will be provided with a *pro-forma* that will allow you to audit your submission and provide a self-assessment of your work. These assessments are an integral part of the assessment procedure. The deadline for submission of phase 1 is the start of week 7 of term 1; for phase 2 it is the start of week 7 of term 2. Phase 1 contributes 50% of the project mark, and phase 2, 50%. Note that the CS Individual Project contributes 10 points towards your overall mark for the third-year course, against 12 points for a single technical module. This fact should be borne in mind when dividing your time between the project and the taught modules.

8.1.2 Software Engineering Individual Project

This project aims to provide students with experience in the engineering of a non-trivial system. In particular we will focus on:

Non-functional requirements: The functionality provided by the system will be simple. The emphasis of the project will be in capturing and ensuring the delivery of non-functional attributes of the system. For example: performance, scalability, maintainability, and availability.

Process: You will be required consciously to consider your process, the elements comprising the process, and how they combine. You will be expected explicitly to plan your time and the delivered systems should include documentation, test results etc as well as the code. You will be expected to use tools to support your process where these are available.

Technologies: The project will require you to use some key technologies e.g. some Java distributed system technologies as well as XML. The practical will stress a component-based approach to design and implementation.

The central task is to design and develop a small client that allows distributed peer-to-peer information sharing. The project is divided into two phases: the first is to develop a design document and carry out some preliminary prototyping of the system design; the second is to develop an initial implementation of the system. The preferred language for most of the project is Java but this is not compulsory.

The first phase of the project runs through term 1 and the second in term 2. There will be briefing meetings at the start of each phase, and a support team will be available to help with any problems. In addition, you are strongly encouraged to use the CS3 newsgroup `eduni.inf.module.ip` for discussion of the project. This allows you to share any questions with the largest possible audience, and can highlight any common difficulties with the work.

Assessment Project assessment is primarily based on two deliverables. For each deliverable you will be provided with a *pro-forma* that will allow you to audit your submission and provide a self-assessment of your work. These assessments are an integral part of the assessment procedure. The deadline for submission of phase 1 is the start of week 7 of term 1; for phase 2 it is the start of week 7 of term 2. Phase 1 contributes 50% of the project mark, and phase 2, 50%. Note that the SE Individual Project contributes 10 points towards your overall mark for the third-year course, against 12 points for a single technical module. This fact should be borne in mind when dividing your time between the project and the taught modules.

8.1.3 System Design Project

In addition to the general project aims outlined above, the System Design Project, which runs in term 3, is intended to provide experience of the following issues (many of which are discussed in the term 2 Professional Issues lectures):

- Applying and combining material from several course modules to complete a complex design and implementation task

- Working as a member of a team on a project too complex to be undertaken by one individual
- Working under constraints of time and resources to meet a specification
- Partitioning a problem into well defined sub-problems which interact through clean and carefully defined interfaces and agreeing these within the group
- Helping to form and operate a group structure which maximises the potential of the whole group, rather than any one individual
- Planning the effort of a project, monitoring this and so managing the overall task (using milestones, deliverables etc.)
- Documenting the design, the work performed and the final product
- Understanding the relevance of market research, marketing and other "non-technical" issues
- Mounting a demonstration and presentation of the final product.

Students work in groups of about ten, on the design of both hardware and sofware of a complete system. Full details of the system to be designed will be given to the groups at the start of term 3, but recent examples include a speech synthesis device for blind computer users and a shape recognition device for automation. The project lasts for four weeks.

During term 2, the groups are announced and a staff member is assigned to each group as supervisor. The supervisor's task is to advise and provide feedback on the progress of the group during the project, not to provide technical input.

On the first day of term 3, the details of the project task will be announced, and groups will be allocated a budget (real money) to buy components, along with some initial equipment for their use. They will also receive a second budget (virtual money) which can be used to pay for staff "consultancy" time. Staff consultants will be available to advise on management, design, specification, software and hardware.

During the project itself, each group will meet with their supervisor three times a week and should also hold their own meetings as often as necessary. Most work will be done in the North Machine Hall, where each group will have bench space and a dedicated linux PC, together with other equipment.

On the Friday of week 4 of term 3, groups will demonstrate their implemented system, and give a "sales" presentation of it, to an audience of the other students, supervisors, and visitors from industry.

Assessment will be based on the final product developed, the documentation, the demonstration and presentation, the group's ability at self-management, group self-assessment and supervisor moderation of the above. Both an overall mark for each group (which carries a prize), and derived individual marks will be awarded.

High marks will be given to groups which develop clear, realistic designs which meet the spirit of the requirements. Quality of documentation, success of team working and flexibility in solving problems are also key factors. Individuals who opt out of group efforts will be heavily penalised.

8.2 Algorithms and Data Structures

Description

The CS3 *Algorithms and Data Structures* course is concerned with the application of (mostly elementary) mathematical techniques to the design and analysis of efficient algorithms. The range of techniques for algorithm design introduced in the first and second year courses is augmented and developed. The notion of computational complexity is stressed as an integral part of the problem solving process.

Syllabus

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

Introductory Concepts Models of computation; time and space complexity; upper and lower bounds; average and worst case analysis.

Algebraic Algorithms Matrix multiplication: Strassen's algorithm; polynomial arithmetic: the fast Fourier transform.

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

Advanced Data Structures Priority queues; data structures for disjoint sets.

Graph Algorithms Dijkstra's algorithm and Kruskal's algorithm; network flow and bipartite matching.

Dynamic Programming Matrix-chain multiplication; longest common subsequences.

Geometric Algorithms Intersecting line segments in two dimensions; convex hull of a set of points (in 2-d).

Assessed Coursework

There will be three assignments designed to aid students' understanding of the lecture material. These will consist largely of pencil and paper exercises, but there may also be some small programming tasks.

References

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

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

(*) Sedgewick: *Algorithms in C (Part 1-5)*, Addison Wesley, 2001.

8.3 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.

8.4 Computability and Intractability

Description

The module 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 module 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 Term 3.

References

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

(*) 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 module.

8.5 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 with deadlines at the end of weeks 5 and 10.

References

(***) J.L. Hennessy & D.A. Patterson *Computer Architecture: A Quantitative Approach* (second edition), Morgan Kaufmann Publishers Inc., 1996. (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).

8.6 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 course is delivered by Eric Mckenzie with teaching assistant Irwin Kennedy. 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.

8.7 Computer Design

NOTE: Owing to constraints on laboratory space, this module is limited to **80** students. We regret that visiting students may not take this module, unless it was agreed prior to entry to the University. The places will be allotted first come, first served.

Description

The CS2 course covered some details of the interconnection of CPU and I/O systems in a complete computer, and CS1h provided a brief look at the inside of the CPU itself. 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 module 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$7 lectures) you should be able to design combinational and sequential logic blocks using a variety of design techniques and implementation methods, be able to choose suitably between the various design and implementation options, and be able to analyse other people's designs; by the end of the second section ($\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, programmable logic), hardware description languages, hardware design tools. *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, programmable logic). *IC logic families:* CMOS, bipolar

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, static & dynamic RAM, 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 – students work in pairs on these, from 2 to 5 one afternoon a week from week 2 to week 9 of term. The practicals in the hardware lab are designed to give you familiarity with design techniques covered in the first three sections; you get a chance to try designing at the computer system level during the *System Design Project* in term 3.

References

(**) V. C. Hamacher, Z. G. Vranesic & S. G. Zaky *Computer Organization,* fourth edition, McGraw-Hill, 1996. 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.

8.8 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.

8.9 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.

8.10 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

Three written assignments, weighted equally: one 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.

8.11 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 is in two parts:

1. 3 weeks duration, due end of week 5
2. 3 weeks duration, due end of week 10.

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/**

8.12 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 (Linix, Windows 2000 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 6 and 10 of the term. 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.

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

(**) 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.

8.13 Professional Issues

There are many engineering and professional issues, complementary to technical ones, that computer science students should know something of. These are not always easy to exemplify through coursework, but many of them form an essential underpinning for the System Design Project in term 3. They include:

- the difference between design, prototype and product
- the product lifecycle
- quality assurance and engineering standards
- marketing and marketing research
- projection management and team working
- finance, cashflow, resource management and profitability
- professional certification and ethics
- legal constraints, responsibilities and responsibilities.

The Professional Issues module provides a brief introduction to these issues. It is spread across all three terms.

At the start of term 1, a single introductory lecture is given, covering the nature of degree programmes (self-directed learning, employers' views and requirements of graduates), the computing profession (the British Computer Society, Codes of Conduct and Practice), and the engineer in society (technology in society, industry and commerce, communication skills).

In term 2, lectures from visiting speakers provide an introduction to some of the non-technical issues of importance to someone working in the fields of software or hardware development and engineering. Parts of the module should provide insights in preparing for the CS3 System Design Project, which takes place in term 3.

The theme for this year is that of the I.T. entrepreneur, covering the importance of high-tech startups to Scotland and the problems (and benefits) faced by an entrepreneur in getting a high technology company up and running.

The topics covered should form a good introduction to anyone who has wondered if they should have a go at starting their own company, and there are opportunities in the course to meet people who have specific relevant expertise and access to resources. The module covers much of relevance to business generally and is useful material for discussing with potential employers. During the course students will be asked to submit an essay developing a business idea using material from the course. Essays may also be submitted for the Lehman prize (which last year was a Palm Pilot). Lectures take place weekly during term 2.

In term 3, the Professional Issues module continues with eight lectures on legal issues, given by staff from the Law Faculty. These are given over the first five weeks of term. There will be a 1 hour class examination on this material during week five of term 3.

The syllabus for the term 3 part of the Professional Issues module covers the following:

- Main source of law (statutes, case law); court structures, Scottish civil/criminal courts, English civil courts

- Contracts: nature, formation, validity, interpretation, breach of contract
- Engineering contracts: agency, authority of the agent, role of third parties
- Sale of goods: passing of property, title, securities, forms of consumer finance
- Business organisations: partnerships, companies (limited, public)
- Health and safety at work: negligence, employer's duty, vicarious liability, statutory duty, Health and Safety at Work Act, product liability
- Intellectual property: patents, copyright, trade marks, industrial designs.

Assessment

The final mark given for Professional Issues is the mark for the class exam on legal issues in term 3. However, to pass this module, it is also necessary to submit a satisfactory term 2 essay.

8.14 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.

Web URLs

- Unified Modelling Language: `http://www.rational.com/uml/`
- Argo/UML freeware tool: `http://argouml.tigris.org/`
- Java tutorial: `http://java.sun.com/docs/books/tutorial/`

References

(***) Bennett, Skelton and Lunn *UML*, Schaum's Outline Series, McGraw-Hill, London, 2001

(**) Stevens and Pooley *Using UML: Software Engineering with Objects and Components*, Addison-Wesley, updated UML1.3 edition, Sep 1999.

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

(*) Booch *Object Oriented Design with Applications* (2nd Edition), Benjamin Cummings.

(*) Boone *Java Essentials for C and C++ Programmers*, Addison-Wesley.

(*) Winston and Narasimhan *On to Java*, Addison-Wesley.

9 Descriptions of Modules and Projects – AI3

9.1 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 40gets 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

A large practical (LP) during the third year is a compulsory part of all AI joint honours degrees. It is also compulsory for non-graduating students doing AI honours modules unless a student only stays at Edinburgh for one term. Students on an AI joint ordinary degree do the LP only if they are not taking the CS group project. The large practical is equivalent to a single third year module in terms of the percentage of the final degree mark that depends on it and the amount of effort that students are expected to put into doing it.

The LP does not have a formal written examination as in most other modules; instead, the assessment is based on practical work and a written report submitted at the end of the project period. The practical work *covers part of term 1, and carries on in term 2*.

In 2002–03 there are two different exercises: **3-lpcm** and **3-lpp**. Students in any AI joint degree class may choose either of the two exercises as their LP. However, different LPs follow different routes through the system. The **3-lpcm** LP assumes knowledge of the material in AI2 Module 1 (Using Constraints to Solve Problems) and Prolog programming skills.

Assessment (3-lpcm)

The problem description will be issued to students on in the middle of week 4 of Autumn Term (term 1) , and there will be one deadline in the Autumn term.

There is then a gap, with no further scheduled submissions on the LP during term 1. Work recommences at the start of term 2, with four further deadlines during that term.

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.

References (3-lpcm)

(*) R. Haralick and G. Elliot, 'Increasing tree search efficiency for constraint-satisfaction algorithms', *Artificial Intelligence*, 14, 263–313, 1980.

(*) I. Miguel and Q. Shen, 'Hard, flexible and dynamic constraint satisfaction', *Knowledge Engineering Review*, 14(3), 199–220, 1999.

(*) V. Kumar, 'Algorithms for constraint satisfaction problems: a survey', *AI Magazine*, Fall 1992, 32–44.

(*) M. Ginsberg, 'Dynamic backtracking', *Journal of AI Research*, 1, 25–46, 1993.

(*) P. Prosser, 'Hybrid algorithms for the constraint-satisfaction problem', *Computational Intelligence*, 9(3), 268–299, 1992.

Assessment (3-lpp)

Details will be available via the module web page nearer the time. The practical will involve machine learning and text classification (that is, given some document, automatically working out some category for that document). In particular, the project will look at how supervised and unsupervised techniques can be combined together.

References (3-lpp)

To be announced.

9.2 AI Programming in Java

NOTE: This module is available in AI3 only by special permission, to those entering the third year of certain joint AI degrees who do not have sufficient programming experience. Students requiring this module should consult their Director of Studies. *This module will be examined in the April diet.*

Description

The study of Artificial Intelligence generally involves the formation of hypotheses and theories which can then be tested through the creation of computer models. In order to create these models, students need to be able to write their own computer programs as well as use pre-existing special purpose systems and tools. This module is intended to provide students who do not already have significant computing experience, with the ability and confidence to use Java as their programming tool for their summer project work.

Having completed the module, a student should be able to: 1) write and run programs in Java which use basic Artificial Intelligence techniques, and which can be read and understood by other people; 2) use the standard Java packages: particularly java.io, java.lang, and java.util; 3) create simple graphical interfaces using javax.swing; 4) navigate the standard Java online documentation.

Syllabus

Object-oriented programming concepts Classes, objects, sub-classes, inheritance, encapsulation, polymorphism

The Java programming language and standard library packages Packages, classes, interfaces, instances, fields, methods. Variables, identifiers, types, values. Expressions, statements, conditionals, loops, labels. exceptions, threads, I/O. Strings, collections.

Basic AI programming Data representation. Recursion, search.

Basic Graphics Applets, Components, Events. AWT, Swing.

Assessment

This module is not a core UG3 module, and may have a non-standard coursework/examination weighting. At the time of printing, the lecturer intends a 50/50 weighting. A final decision will be announced at the start of the course.

Assessed Coursework

Practical exercises are set to aid students' understanding of the course material.

References

(*) M. Campione, K. Walrath, A. Huml: *The Java Tutorial (3rd Edition). Addison Wesley, 2001.*

(*) K. Arnold, J. Gosling, D. Holmes: *The Java Programming Language (3rd Edition) 2000.*

(*) D. Flanagan: *Java in a Nutshell (4th Edition). O'Reilly, 2002.*

(*) B. Eckel: *Thinking in Java (2nd Edition). Prentice Hall PTR, 2000.*

(*) H. Schildt: *Java 2: A Beginner's Guide. Osborne/McGraw-Hill, 2001.*

9.3 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.

9.4 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.*

9.5 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 the standard format. Instead, it will require a single 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. The title is to be agreed with the module lecturer.

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 9, and carrying the remaining 15% contribution.

9.6 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 semantic interpretation, 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 and Background Aims and methods of computational linguistics. Introduction to Python.

Tokenisation, Morphology and Tagging Regular expressions and finite state automata. Finite state approaches to morphology. Probabilistic methods and n-grams. Part-of-speech tagging.

Syntactic Analysis and Parsing Context free grammars and parsing. Partial parsing. Unification grammars.

Semantic Interpretation Compositional semantics. Information Extraction. Lexical semantics.

Assessed Coursework

Three assignments throughout the course, which require programming in Python as well as written work.

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*

9.7 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 by moving around in it and moving things about, and being able to make sense of the nature of the immediate circumstances by means of information acquired by sensors, one of the most complex and useful of these being vision. Having to apply AI in this kind of context poses certain kinds of problems, and sets certain kind of limitations, which have important effects on the kinds of 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. It is remarkable how well biological evolution has solved some of these problems, e.g., how well a fly manages to avoid your hand with very limited computational power. We can often acquire good design ideas for artificial systems by studying natural ones. There is also a rapidly growing research area in which robotic implementations of hypothesised biological mechanisms is used as a biological research tool. This module introduces the basic concepts and state of knowledge in some of 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.
- Actuators and motor control; sensors and sensory processing. Parallels to biological systems.
- Image formation, transduction and simple processing; thresholding, filtering and classification methods for extracting object information from an image.
- Active vision and attention.
- Assembly robots (which move things about according to complex previously prepared plans)
- Mobile robots (which explore and construct maps).
- Putting systems together: classical architectures; reactive architectures; hybrid architectures.

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.

9.8 Knowledge Engineering

Description

This module introduces a variety of methodologies important to the development and applications of knowledge-based systems (KBSs). The module covers topics regarding different processes within a KBS life-time, ranging from knowledge capture and analysis, systems design and implementation, to knowledge maintenance and systems evaluation.

Syllabus

- Knowledge, expertise, and knowledge engineering in the development and applications of KBSs.
- Structured interview based knowledge elicitation, automated knowledge acquisition from historical data, knowledge acquisition methodologies.
- KBS synthesis: using KBSs in the context of software development, and building KBs of SE knowledge.
- Ontologies as a basis for structuring KBs, and for combining disparate information.
- Reasoning maintenance systems capable of making hypotheses and exploring their consequences, KBSs that utilise explicit domain models, other problem-solver building approaches.
- Distributed multi-agent architectures, shared and common knowledge, evaluation of reasoning about multiple agents.

Assessed Coursework

Two exercises, probably one written and one practical.

References

Module notes will be provided, available from the ITO.

9.9 Learning from Data 1

Description

Since the early days of AI, researchers have been interested in making computers learn, rather than simply programming them to do tasks. This is the field of machine learning. The main area that will be discussed is supervised learning, which is concerned with learning to predict an output, given inputs. For example, we might wish to classify a handwritten digit as one of the numbers 0 through 9, given an input image of the digit. We will compare and contrast different algorithms for supervised learning tasks. One of the key issues is that of generalization, i.e. how good our performance will be on new input patterns. A second area of study is unsupervised learning, where we wish to discover the structure in a set of patterns; there is no output "teacher signal".
The field of learning from data is of increasing importance, given the large volume of data that can now be collected in many different domains, ranging from time recordings of multiple neurons from regions of the brain, to point-of-sale information from supermarkets; the task is to make sense of this data by understanding its structure and using it to make predictions.

Syllabus

- Introduction
- Mathematical Preliminaries
- Supervised learning: Perceptrons, neural networks, Naive bayes, feature selection, nearest neighbour methods, decision trees.
- Generalization: Assessment of performance, experimental design and methodology, model selection.
- Unsupervised learning: Clustering, PCA/autoencoders.

Assessed Coursework

There will be two assessed practicals. Together, these will carry 20% of the course marks.

References

(*) T. Mitchell: *Machine Learning. McGraw-Hill, 1997.*

(*) C. M. Bishop: *Neural Networks for Pattern Recognition. Oxford University Press, 1995.*

(*) S. Russell and P. Norvig: *Artificial Intelligence: A Modern Approach. Prentice Hall, 1995.*