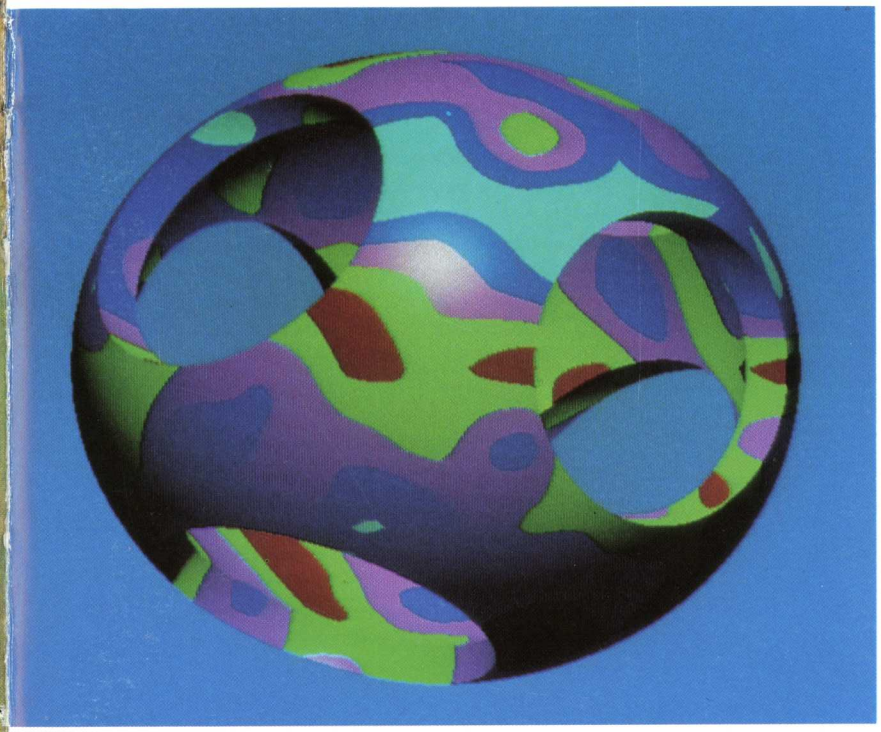
THE UNIVERSITY *of* EDINBURGH

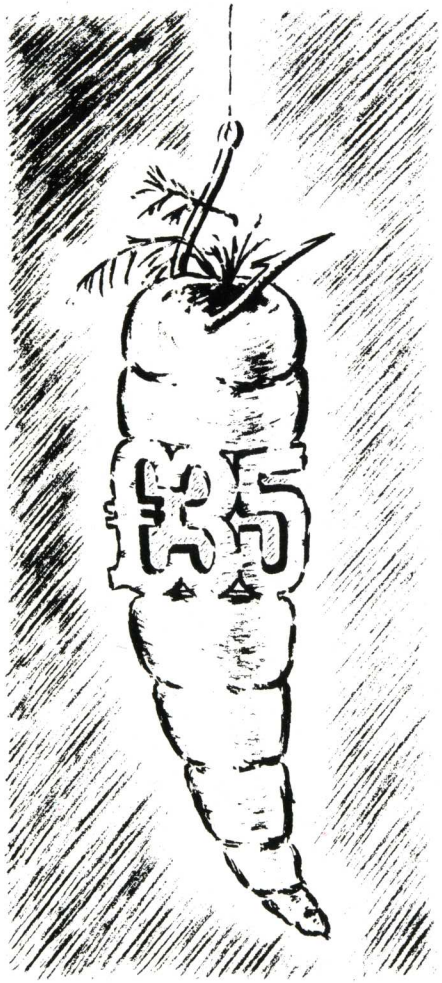


DEPARTMENT *of* COMPUTER SCIENCE

HANDBOOK 1991-92

WHAT CAN THE ROYAL BANK OF SCOTLAND OFFER STUDENTS?

HERE'S £35 FOR A START.



Like every high street bank in 1991, we're offering an incentive for new students to join us: deposit your grant cheque and we'll add £35 on top.* But we also think that your choice of bank should hinge on a little more than who's offering the best carrot.

In our view, you should choose the bank which will be most helpful, not just in your first few weeks, but throughout your years at college or university and beyond.

At The Royal Bank of Scotland we have a strong tradition of personal service.

We take a genuine interest in your circumstances and your future, and do all we can to help you fulfil your plans.

And you'll find our services just as helpful.

For example, we can grant interest free overdrafts of up to £400 in your first year (as long as you clear it with your branch first).

We also have "tiered" rates of interest when you're in credit, and there are no account charges provided you stay in credit, or within your overdraft limit.

Call in at your nearest branch for details of our 1991 student package. Alternatively complete the coupon or call us free on **0800 636 626.**

In the long run, you won't find a better place to start.

To: The Royal Bank of Scotland plc, FREEPOST (SO), PO BOX 31, Edinburgh EH2 0DG.
Please send me more information on your student services.

Name_____

Address_____

_____

_____ Post code _____

DCS/91

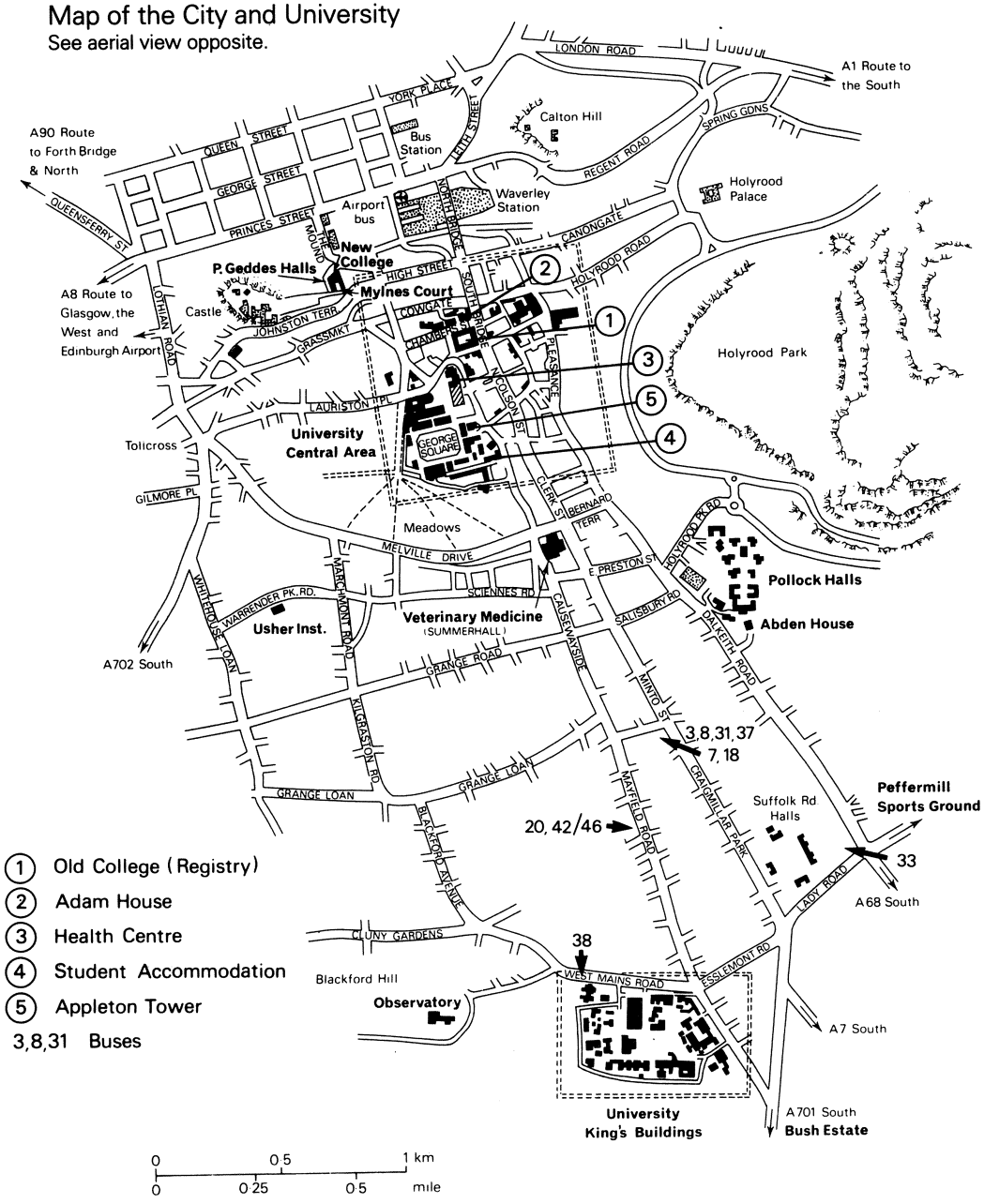
The Royal Bank of Scotland

WHERE PEOPLE MATTER

*To qualify for the £35 offer, applicants must be about to start a full-time higher education course (Degree, HNC and HND), or first year training as a nurse. Applicants must deposit a grant cheque or provide proof that tuition fees are being paid by SED or LEA. Students must be 18 or over to apply for overdraft facilities.

The Royal Bank of Scotland plc. Registered Office: 36 St. Andrew Square, Edinburgh EH2 2YB. Registered in Scotland No. 90312.

UNIVERSITY OF EDINBURGH
Map of the City and University
See aerial view opposite.

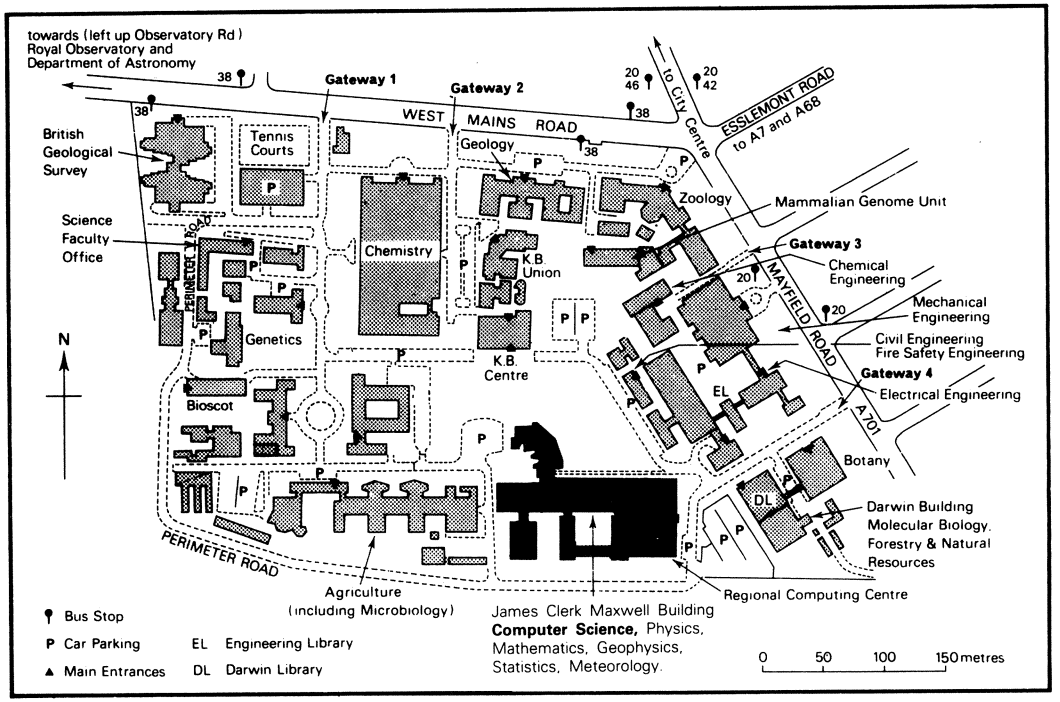


1. Old College (Registry)
2. Adam House
3. Health Centre
4. Student Accommodation
5. Appleton Tower

3,8,31 Buses

The back cover is an aerial view of the Kings Buildings site.

UNIVERSITY OF EDINBURGH
Map of Kings Buildings



Front cover photograph: holey sphere created using Mistral CAD software.

DEPARTMENTAL HANDBOOK

The University of Edinburgh
Department of Computer Science
The King's Buildings
Edinburgh EH9 3JZ
Tel: 031 650 5112 (enquiries)
Fax: 031 667 7209
E-mail: lmp@uk.ac.ed.dcs
Tlx: 727442 (UNIVED G)

Software Systems Concepts

Topics include: Abstract data types; object oriented programming; functional programming; concurrency and communications; formal languages.

Database Systems

Topics include: Data models and query languages; recursive query evaluation; the closed world assumption; physical data organisation; dependency theory and logical database design; transaction processing and concurrency control; distributed databases; query optimisation; database performance analysis.

INTRODUCTION

Computer Aided Formal Reasoning

Topics include: Natural deduction proofs for first and higher order logic; proof by induction; propositions and types; representing mathematical objects; tactics; rewrite rules.

Formal Programming Language Semantics

Topics include: Introduction to domains and the λ calculus; types and polymorphism; denotational semantics.

Communications and Concurrency

Topics include: The language CCS and its operational semantics, derivation trees; flowgraphs and flow algebra; observational equivalence and congruence, weak bisimulation; a complete axiomatic system for congruence of finite behaviours; semantics of an imperative language by translation into CCS.

Operating Systems

Topics include: Process management; the OS kernel; resource allocation; memory management; file management; distributed systems.

Graphics

Topics include: Applications and overview of computer graphics; graphics devices; 2D graphics - shape generation by scan conversion into a raster, including lines, circles and polygons, anti-aliasing, area fills, linear transformations of shapes and homogeneous coordinates, line and polygon clipping, dithering; 3D Graphics - linear transformations, homogeneous coordinates and clipping, methods and data structures for modelling objects, boundary representations, implicit solids, set theoretic constructs, volume data, methods of rendering images, wire frames, rendering with hidden edge, surfaces and volumes, illumination models, shaded images, texture, advanced illumination models for photo-realistic images synthesis with shadows, reflections and transparency.

Computer Communications

Topics include: Communications architectures; data link control; network layer protocols; internetworking; transport layer protocols; session, presentation and application layer protocols; integrated services digital networks; medium access control; error control; routing and flow control.

Software for Parallel Computers

Topics include: Languages based on message passing; languages based on shared memory; languages for data-parallel machines; parallel algorithms; abstract models of parallel computers and their emulation.

Distributed Systems

Topics include: Process communication and synchronisation; remote procedure calls; the client server model; MACH system overview; name/location service, authentication service, file service; the Grapevine mail server.

Theoretical Computer Science

Algebraic Complexity

Topics include: Quadratic and bilinear maps; non-Linear lower bounds; complexity of partial derivatives; reductions.

Applicative Programming and Specification

Topics include: Programming in Standard ML; formal program specification and development in Extended ML; selected topics in algebraic specification.

Computational Complexity

Topics include: The Turing machine model; complexity classes and hierarchies; reductions between problems and completeness; the classes P, NP, PSPACE, #P: complete problems; provably intractable problems; randomised algorithms and classes; circuits and non-uniformity; parallel computation.

INTRODUCTION

The City and its University

The University of Edinburgh was founded over 400 years ago in 1583, some twenty years before the Union of the Crowns which made James VI (I) king of both Scotland and England. Throughout its history it has numbered a host of distinguished scholars amongst its teachers and students. Its situation in the capital of Scotland, one of the world's most beautiful cities, has helped maintain this tradition.

Today, the University of Edinburgh is one of the largest in the UK. Its students enjoy not only the advantages of studying at a well-endowed university but also the many cultural, sporting and social aspects of the city. One advantage of studying at Edinburgh is the unusually large number of optional courses which the University, because of its size, can offer to its students and the resulting flexibility of its degree system.

The Department of Computer Science

The Department of Computer Science at Edinburgh was established with six staff in 1966 when the functions of the Computer Unit (established in 1963) were divided between teaching (in the Department) and service (in the Edinburgh Regional Computing Centre, now the Edinburgh University Computing Service). The Department has since grown into one of the largest departments in the Science and Engineering Faculty with around 32 teaching staff supported by a similar number of computing officers, technicians and secretaries.

At undergraduate level the department offers a four-year Honours degree in Computer Science together with joint Honours degrees with Artificial Intelligence, Electronics, Management Science, Mathematics, Physics, and Statistics. There is also an alternative first year course more suited to students of other scientific and engineering disciplines and an Information Systems course aimed at students in other Faculties.

At postgraduate level there is a flourishing community of research students working towards the degrees of Ph.D. The department also runs an M.Sc. in Computer Science with three themes: Computer Systems Engineering, Parallel Systems, and Theoretical Computer Science.

Research in the Department of Computer Science started in 1966 with a multi-access project from which developed the interactive computing system

EMAS (Edinburgh Multi Access System), which was used for some twenty years on a succession of mainframe computers to provide the University's main computing service. Since then the Department has gone on to gain an international reputation for the quality of its research, receiving top rating in both the 1986 UGC and 1989 UFC Research Selectivity Exercises. There are several very active groups working in a number of different fields including: computational complexity, graphics, parallel and distributed systems, theory of computation and VLSI design using both formal and informal methods.

The Department is involved with two major research units within the University. The Laboratory for the Foundations of Computer Science is a federation of research projects within the Department unified by their interest in computational theory and its applications. The main areas of interest are proof engineering, languages and programming, concurrency and communication, specification and formal development of systems, and semantics. The Edinburgh Parallel Computing Centre, which was set up jointly by the Department of Computer Science, the Physics Department and EUCS, encompasses the activities of a variety of research groups. It has as one of its main aims the support of applications work by investigating those fundamental aspects of parallel computing which can lead to the development of new and improved systems and user support tools.

The Department is located in the James Clerk Maxwell Building (JCMB), a multi-user building shared by several academic departments including Mathematics, Meteorology, Physics and Statistics together with the main facilities of the Edinburgh University Computing Service (EUCS). The JCMB provides extensive accommodation including offices, laboratories, lecture theatres, a library and several common rooms. It is situated in semi-rural surroundings at the King's Buildings campus of Edinburgh University, about 2 miles from the city centre.

Computing Facilities

There is a wide range of computer systems in the Department of Computer Science interconnected by an extensive local area network. Facilities include over 160 SUN workstations plus Apple Macintoshes and IBM PCs.

The EUCS, which houses several mainframes (including a pair of multiprocessor Sequent symmetries), supports a distributed computing environment for the University and manages a 200 Megabit FDDI LAN which interconnects the various University sites and buildings at Kings Buildings and

M.Sc. Modules

Computer Systems Engineering Theme

ASIC Design

Topics include: Introduction to IC Technology; principles of VLSI design; design styles; programmable logic arrays; memory designs; microprocessor design; ASIC tools.

Behavioural Specification and Verification

Topics include: Behavioural design tools; modelling digital behaviour; formal specification and verification; synthesis; data abstraction; temporal abstraction.

Computer-Aided Design

Topics include: Solid modelling; CNC machining; kinematic analysis; finite element analysis ; spatial layout in architectural design; databases for ECAD; hardware description languages; algorithms for automatic Printed Circuit Board layout; placement & Routing algorithms for VLSI layout; switch-level simulation algorithms; behavioural synthesis for VLSI.

Systems Integration

Topics include: Systolic algorithms and systolic arrays; parallel processor arrays; reduced instruction set computers and optimising compilers; language-oriented architectures; Analog VLSI networks; area-time complexity of VLSI circuits; simulation of VLSI systems on Multiprocessors.

Systolic Design

Topics include: Systolic design; systolizing compilation; related issues and research problems.

Parallel Systems Theme

Concurrent Architectures

Topics include: Vector and Array architectures; shared-memory multiprocessors; message-passing multiprocessors; novel architectures.

Computer Simulation and Modelling Techniques

Topics include: Modelling; performance analysis; queueing theory; networks of queues; programming a simulation; data collection; case studies.

Parallel Architectures

Topics include: Performance and cost measurement; advanced pipelines; vector processing; interconnection structures; array processors; shared-memory multiprocessors; distributed-memory multiprocessors; technology issues.

Program Logics

Topics include: Basic propositional modal logics; meta-theory; detailed case study of propositional dynamic logic; basic propositional temporal logic; propositional modal and temporal logics with fixpoint operators; meta-theory of temporal logics; detailed case study of model checking and its application to proving.

Software for Parallel Computers

Topics include: Application areas, different types of parallelism; languages for parallel machines; examples of numerical and non-numerical algorithms; program decomposition, placement, load-balancing; software development tools; design methodologies for parallel programs.

Communication and Concurrency

Topics include: The language CCS and its operational semantics, derivation trees; flowgraphs and flow algebra; observational equivalence and congruence, weak bisimulation; a complete axiomatic system for congruence of finite behaviours; semantics of an imperative language by translation into CCS.

VLSI Design

Topics include: Introduction (MOS devices, CMOS circuits, layout and fabrication); design issues (topological, temporal and behavioural views); synthesis tools (cell generators, ASICs, routing generation, silicon compilation); analysis tools (design-rule and electric-rule checkers, verification, simulation).

in the centre of town.

The facilities of the Edinburgh Parallel Computing Centre include a 400-transputer Meiko Computing Surface, two AMT Distributed Array Processors (DAPs), a 64-node i860 'Grand Challenge' machine and a Thinking Machines Corporation Connection Machine.

Computer-Aided Design

Topics include: Solid modelling; CNC machining; kinematic analysis; finite element analysis ; spatial layout in architectural design; databases for ECAD; hardware description languages; algorithms for automatic Printed Circuit Board layout; placement & Routing algorithms for VLSI layout; switch-level simulation algorithms; behavioural synthesis for VLSI.

Computer Algebra

Topics include: Operations on integers; rational; real and complex numbers; operations or symbolic elements; solution of systems of equations (linear, non-linear, differential); manipulation of formulae; analysis; other facilities.

Computer Graphics

Topics include: Algorithms for display; line drawings; curves and surfaces; body modelling; shading; improved rendering; texture; colour.

Denotational Semantics

Topics include: Abstract syntax and the denotational format; scope and environments; stores and data types; basic domains; recursion and iteration; procedures, functions and parameter mechanisms; continuations.

Distributed Systems

Topics include: Process communication and synchronisation; remote procedure calls; the client server model; MACH system overview; name/location service, authentication service, file service; the Grapevine mail server.

Human-Computer Interaction

Topics include: Psychology of HCI; task analysis and specification; design methodologies; design evaluation; techniques; technologies.

Microprocessor Design Practical

This is a practical module in which students work in small teams, each designing their own microprocessor using one of several sets of VLSI design tools.

Knowledge Based Systems

Topics include: Issues in inference; organising knowledge bases; extended logics; handling uncertainty; human interface; Expert Systems toolkits; alternative architectures; Software Engineering applications.

Language Semantics and Implementation

Topics include: Language semantics; abstract machines; implementation.

Operating Systems

Topics include: Process management; the OS kernel; resource allocation; memory management; file management; distributed systems.

Programming Methodology

Topics include: Description of software engineering; project planning; software creation; specification languages; a review of specification techniques; composing specifications; specification consistency and completeness; refining specifications.

Professional Issues

Topics include: Codes of conduct and practice; law relating to engineering and computing; lectures by visiting industrialists.

CS4 Modules

Compiling Techniques

Topics include: Structure of a compiler; lexical analysis; syntax analysis; semantic analysis; run-time environments; intermediate code generation; optimisation.

Computational Complexity

Topics include: The Turing machine model; complexity classes and hierarchies; reductions between problems and completeness; the classes P, NP, PSPACE, #P: complete problems; provably intractable problems; randomised algorithms and classes; circuits and non-uniformity; parallel computation.

UNDERGRADUATE COURSES IN COMPUTER SCIENCE

APPENDIX

COURSE MODULE CONTENTS

CS3 Modules

Algorithms and Data Structures

Topics include: Algebraic algorithms; sorting and selection; advanced data structures; graph algorithms; geometric algorithms; number-theoretic algorithms; approximation algorithms for hard problems.

Computability and Intractability

Topics include: Finite state machines; Turing machines; the Church-Turing Thesis; universal machines; decidability and partial decidability; nondeterministic Turing machines; resource bounded computation; NP-completeness.

Computer Architecture

Topics include: Instruction formats; storage hierarchies; pipelines; instruction buffering; parallel functional units; vector processors; novel architectures.

Computer Communications

Topics include: Information; time; space; local area networking; wide area networking; case studies; real world issues.

Computer Design

Topics include: Logic design; processor design; memory design; I/O design; buses; microprocessors.

Database Systems

Topics include: The relational model of data; concurrency control and distributed databases.

UNDERGRADUATE COURSES

IN COMPUTER SCIENCE

The Department offers several Honours and Ordinary degrees, as well as providing undergraduate courses contributing to other degrees, outside of Computer Science. Degrees are offered in the following disciplines:

- Computer Science
- Artificial Intelligence and Computer Science
- Computer Science and Electronics
- Computer Science and Management Science
- Computer Science and Mathematics
- Computer Science and Physics
- Computer Science and Statistics

An Honours degree involving Computer Science (single or joint Honours) lasts four years. During the first two years students take three subjects, one being Computer Science. Students intent on studying for a joint Honours degree must also take the prescribed course(s) relevant to that subject. During the first two years, a substantial fraction of time can be spent on support courses. This is an important element in the education of an Edinburgh graduate since it presents an opportunity for the students to broaden their interests. In the final two years, students study courses in their chosen speciality/ies only.

An Ordinary Degree may be obtained after three years of study. The first two years of study are identical to the corresponding Honours degree; in the third year students are assessed on a work load equivalent to approximately 70% of the Honours course.

The degree structure at Edinburgh is very flexible and students can usually keep their options open (as far as final degree is concerned) to the end of the first and sometimes even the second year.

Course Philosophy

The philosophy underlying the Single Honours Computer Science course is that it should produce graduates of the highest quality with a thorough understanding of the software, hardware and underlying science of computer systems and with the engineering skills required to design and implement them. Theory and practice are integrated throughout the course, with students undertaking a wide variety of practical exercises and projects which reinforce and build on lecture material. Communication skills, initiative, professionalism and the ability to work with others are developed during the course as integral parts of the learning process.

The Joint Honours courses offer the opportunity to study part of the Single Honours Computer Science course in conjunction with another (cognate) discipline. In years 1 and 2 Joint Honours students attend the same qualifying Computer Science courses as Single Honours students. In the Honours years (years 3 and 4) the amount of Computer Science studied is typically half of that of the Honours course, but for each Joint Honours course the modules studied form an educationally sound match with the other discipline. The major individual project undertaken in the final year brings together elements of both disciplines.

Computer Science is a rapidly evolving discipline and course and syllabus revision is an on-going activity within the Department[1]. An important influence on the contents of the undergraduate courses is the research being carried out in the Department. In the Honours years each module is normally taught by a member of staff who is involved in research in the subject, or one closely related to it. In the final year, many of the projects are closely linked to research work in the Department, and final year students in particular are encouraged to attend the research seminars given by both internal and external speakers throughout the year.

Qualifying Courses

All students at Edinburgh take three separate courses in each of the first two years. To enter the third year of a Computer Science course students in their first year must take and pass Computer Science 1A (CS1A), and

[1] One consequence of this is that this Handbook does not form a guarantee that any particular course or module will be available, or that its content will be precisely as described herein.

APPENDIX
COURSE MODULE CONTENTS

David Turner *R. Milner, K. Mitchell*	Functions, modules and concurrency.
Stephen Tweedie *E.R.S. Candlin*	Efficient operating systems for parallel programs.
Wang Li-Guo *M.P. Fourman, R. Burstall*	Proof-based computation synthesis: its theory, methodology and application in computer science. Formal methods in design of hardware and software.
Amanda Welsh *R.J. Pooley, D.A. Welch*	Design of an integrated support environment for scientific computing.
Andrew Wilson *K. Mitchell*	Programming language semantics.
Greg Wilson *R.J. Pooley, A.J. Sinclair*	Efficient high level support for parallel programming.
Xue Jingling *D.J. Rees, C. Lengauer*	Integrated object oriented open DBMS, the user-friendly interface, AI-based CAD tools and the study of data structures and algorithms for such tools
Yeung Ping *D.J. Rees, R.A. McKenzie*	High-level hardware synthesis, system level VLSI design, hardware description languages and logic synthesis.

in their second year Computer Science 2 (CS2). The second course in each year will be in Mathematics, with the third being left to the student's own choice. Joint Honours students normally take a course in the other discipline in each year.

Both CS1A and CS2 involve three lectures and one tutorial per week throughout the teaching year. Students also carry out practical assignments related to the lecture material; this usually involves using the appropriate computing laboratories at scheduled times. The skills acquired in these assignments satisfy the EA1 requirements of the Engineering Council[2].

CS1A concentrates on providing a basic competence in developing, and reasoning about, programs. Students are expected to acquire knowledge of (at least) the following topics:

- Programming in both an imperative and a functional language; the use of procedural and data abstraction; stepwise refinement
- Elementary propositional logic
- Verification of simple programs; invariants; induction
- An introduction to algorithms and data structures; graph algorithms, sorting, searching; analysis of algorithms.
- Using a Sun/X-windows environment

In CS2, the knowledge acquired in CS1A is used as a basis for more advanced study of computer systems. The course introduces the basic hardware and software components of computer systems, and examines how these components constrain the design of such systems. This study is performed at various levels of abstraction, ranging from binary operations, through low-level and high-level language operations, to abstract operations. The course also covers the translation of component descriptions between different levels of abstraction, in particular the compilation of high-level languages to machine codes. At all times, an important concern is the analysis of component designs to determine their properties, and to enable evaluation against alternative designs. Such analyses often involve the application of theoretical concepts.

[2]Engineering Applications 1 as laid down in the Council's policy document *Standards and Routes to Registration*, and interpreted by the British Computer Society in Handbook No. 11.

Assessment

CS1A is assessed in two parts, CS1A1h and CS1A2h. For each there is an examination in June, the results of which form half of the total marks, with the marks for the practical work forming the other half. A class examination is held at the end of each term. Entry to CS2 requires a pass in CS1A.

Performance in CS2 is measured by the continuous assessment of practical work and by examinations at the end of the year. The examination consists of two $2\frac{1}{2}$ hour papers, one for each half of the course. The examination mark contributes two thirds of the final mark and the continuous assessment mark one third.

An overall pass at grade C or better is required for entry into the Honours Years courses. A student who fails CS2 as a whole may be awarded a half course pass and may re-sit either or both examinations in September.

Computer Programming 1h

The aim of this first half year course is to provide students in other science and engineering disciplines with a basic competence in programming techniques and problem solving. Pascal is used for implementations and through its use students will encounter overall program structure, procedures, control structures, assignment, simple numerical problems, data types, control of program execution, user-defined types, files, dynamic data structures and programming methodology.

Principles of Software Engineering 1h

This second half year course is designed for students in other science and engineering disciplines with a reasonable competence in computer programming and who wish to develop their understanding of the design, construction and properties of complex computer software systems. Topics include: techniques for building large scale software systems; data structures; reliability and performance of computer systems.

Information Systems 1

In two parts (1Ah and 1Bh), Information Systems 1 is a course aimed at students in other Faculties. The first half of the course introduces computer technologies, the development of skills and understanding of the application of these technologies, and assesses their social impact. The second half

Rob Pooley
E.R.S. Candlin, A.S. Wight

A unified framework to support cross paradigm performance modelling.

R. Rangaswami
M.I. Cole, K. Mitchell

Parallelism in functional languages.

Vinod Rebello
D.K. Arvind, R.N. Ibbett

Hardware aspects of HCI, computer systems.

Sandy Robertson
R.N. Ibbett, N.P. Topham

High performance microprocessor architecture simulation experiments.

Davide Sangiorgi
R. Milner, G. Brebner

Semantics of concurrent processes.

Roger Sayle
M. Fourman

Use of formal methods, in particular higher order logic in the design of VLSI circuits.

Peter Sewell

Investigation of the semantics of programming languages and techniques for program verification using categorical and domain-theoretic methods.

Fabio da Silva
K. Mitchell, D.T. Sannella

Operational semantics of programming languages, applications of operational semantics to the design and implementation of programming -debugging- tools, using algebraic methods and operational semantics for compiler correctness.

Alex Simpson
G.D. Plotkin

Investigation of the use of formal methods including logic systems based on type theory for the systematic development of computer programs.

Colin Smart
D.K. Arvind

Research interest unspecified.

Makoto Takeyama
R. Burstall

Type theory and category theory with a particular interest in fibred categories, applications to proof.

Arshad Mahmood *K. Mitchell, S.O. Anderson*	Algebraic specifications particularly in relation to concurrency.
Sathiamoorthy Manoharan *N.P. Topham, R.N. Ibbett*	Object-oriented simulation modelling; performance issues in parallel systems - assignment and scheduling problems.
Paul Martin *R. Candlin, A.S. Wight*	Hardware monitoring of parallel computers for characterising the efficiency of program execution and improving performance.
Neil MacDonald *S.O. Anderson, M.I. Cole*	Investigation of methods of concurrent computing particularly implementation aspects of programming languages such as Standard ML and the applications of the Calculus of Communicating Systems (CCS).
James McKinna *R. Burstall, B. Jay*	Category theory applied to the development of proven programs in type theory.
Michael Mendler *R. Burstall, M.P. Fourman*	Logics and models for concurrent systems; specification and verification of hardware.
Matthew Morley *R. Milner, G. Cleland*	Genuine case studies of safety critical systems from industrial collaborators within the framework of the MPSS project.
Michael Norman *P. Thanisch, D. Wallace (Physics)*	Parallel processing, optimisation of parallel programs.
Christopher Owens *S.T. Gilmore*	Assisted synthesis of implementations from formal specifications.
Joseph Phillips *R. Candlin, A.S. Wight*	The effectiveness of dynamic process allocation strategies within an occam environment.
Randy Pollack *R. Burstall, M. Fourman*	Computer-assisted proof development, in particular machine-checked mathematics using the Logical Framework and the Calculus of Constructions.

concentrates on the role of computing in television, digital communications, publishing and printing, together with a demonstration of the broad spectrum of uses of microprocessors and conventional computers.

B.Sc./B.Eng. in Computer Science

Computer Science 3

The first of the two Honours Years of the B.Sc/B.Eng. in Computer Science builds on the knowledge and skills acquired in the Qualifying Courses and involves studying core computer science material in depth. The course comprises eight technical modules (each involving 18 lectures and relevant coursework), a Professional Issues module and two major practicals. The technical modules currently offered are:

- Algorithms & Data Structures
- Programming Methodology

plus either or both of:

- Computer Design
- Computer Architecture

plus either or both of:

- Operating Systems
- Computer Communications

plus either or both of:

- Language Semantics & Implementation
- Computability and Intractability

Additional options are:

- Database Systems
- Knowledge Based Systems

The module on Knowledge Based Systems is appropriate for students who have an interest in Artificial Intelligence but who do not want to study AI to the depth of the Joint Honours degree in Artificial Intelligence & Computer Science.

The nature of the coursework varies considerably from module to module, ranging from pencil and paper exercises through a variety of programming assignments relevant to particular modules, to practical work in a hardware systems laboratory associated with the Computer Design module.

The two major practicals are quite different in nature. One is an individual operating systems project which introduces the student to the problems of writing large piece of software and of understanding how software and hardware interact to form a total system.

The second is a group project which involves the specification, design, implementation and presentation of a computer system, in a simulated industrial environment. This project is important not just because it represents a transition from EA1 to the EA2 skills[3] which are required for the fourth year individual project, but also because it develops interpersonal relationship skills. At the end of the project each group makes a presentation to the whole class, to which industrial representatives are invited. The marking of this project is a complex process and includes elements of self assessment and peer review.

Computer Science 4

In the final honours Year (year 4), the knowledge and techniques acquired in the first three years are used as a basis for advanced study of both familiar and new topics. Half of the time is spent on individual project work and half on taught modules. Each student takes 6 taught modules selected from a range which reflects both staff expertise and student interest. The modules currently offered are:

- Communication and Concurrency
- Computer Graphics
- Compiling Techniques
- Computational Complexity

[3]The Engineering Applications 2 requirement of the Engineering Council as interpreted by the BCS

Leslie Henderson Goldberg *M.R. Jerrum, A.J. Sinclair*	Designing algorithms for solving combinatorial -listing and counting- problems and determining the inherent computational difficulty of such problems.
Kees Goossens *S.O. Anderson, K. Mitchell*	The embedding of computer hardware design and description languages in proof systems.
Timothy Harris *M.I. Cole*	Abstract models of parallel computers. Cost consistency in PRAM emulations.
Claudio Hermida *D. Sannella, S. Anderson*	Functional programming and program development by transformation.
Roberto Hexsel *M. Fourman, D.J. Rees*	Formal verification of low-level timing in VLSI circuits.
Jane Hillston *R.J. Pooley, J.C. Bradfield*	Semantics of component-based stochastic models for performance evaluation.
Anna Hondroudakis *R.N. Procter*	Parallel systems, computer architecture, graphics and distributed networks.
Hans Hüttel *C.P. Stirling, F. Moller*	Semantics of non-determinism and concurrency.
Kayhan Imre *R.N. Ibbett, R. Candlin*	Performance monitoring and analysis of parallel programs.
Lee Yong Woo *A.S. Wight*	Operating systems, performance modelling and analysis of computer systems.
John Longley *M.P. Foreman*	Formal methods in language semantics and system design.
Christopher Luth *D.T. Sannella*	Functional programming and algebraic specifications.
Savitri Maharaj *S.O. Anderson*	Extension of the calculus of communicating systems (and associated logics) in a theoretical or a practical sense.

Søren Christensen *C.P. Stirling*	Models for distributed systems—formal frameworks useful for describing concurrent and distributed systems.
Alan Crawford *N.P. Topham, R.N. Ibbett*	Design of high performance decoupled architectures; compilation techniques for decoupled architectures.
Angus Duggan *R.J. Pooley, K. Mitchell*	Improving code/documentation consistency in literate programming environments
Richard Eyre-Todd *R.J. Pooley, D.J. Rees*	Tools and techniques for visualising parallel systems
Marcelo Fiore *G.D. Plotkin, C.B. Jay*	Semantics of programming languages and logics to reason about programs; categorical treatment of non-terminating computations and recursive definitions, and categorical logics for computable functions.
Gao Bo *D.J. Rees, R.A. McKenzie*	Asynchronous VLSI array architectures for algorithm embedding.
Philippa Gardner *G.D. Plotkin, A.J. Power*	Formal specification of logics.
Neil Ghani *D.T. Sannella, C.B. Jay*	Applications of type theory and category theory to the systematic development of computer programs.
Healfdene Goguen *R. Burstall, M. Fourman*	Category theory and computer-aided proof systems to parallel processing and the science of design.
Paul Goldberg *M.R. Jerrum, K. Kalorkoti*	The learnability of various concept classes according to the "probably approximately correct" model, in particular polynomial-time learning algorithms.

- Computer-Aided Design
- Computer Algebra
- Denotational and Axiomatic Semantics
- Distributed Systems
- Human Computer Interaction
- Microprocessor Design (Practical)
- Parallel Architectures
- Program Logics
- Computer Simulation & Modelling Techniques
- Software for Parallel Computers
- VLSI Design

The project involves both the application of skills learnt in the past and the acquisition of new skills, and is intended to allow students to demonstrate their ability to organise and carry out a substantial piece of independent work. The types of skill required vary from project to project, but six main areas of work are required:

- gathering and understanding background information
- solving conceptual problems
- design
- implementation
- experimentation and evaluation
- report writing

Project topics are normally proposed by members of staff (though students may propose projects of their own) and vetted by a project committee. They span the range of Computer Science, so that students can choose projects which they find useful and interesting, and goals are usually flexible

so that students can work to the best of their ability. The project occupies about 500 hours over the year.

One or more members of staff supervise each project and assist students with both technical and managerial matters. Students typically meet their supervisors once a week during term time. In addition students are organised into groups of between four and six which meet regularly, with supervisors in attendance, to discuss progress. These meetings help to develop presentation skills and provide students with a wider range of feedback on their projects. At the end of the project each student gives a demonstration of his/her work to members of the project committee.

Assessment

In assessing each student's overall degree classification at the end of year 4, the mark for CS3 carries equal weight with the mark for CS4. Examinations, coursework and projects are taken into account in assessing a student's performance and students are expected to perform satisfactorily in all three aspects of the course.

In CS3 each lecture module is examined by a separate one-and-a-half hour examination held in June and by continuous assessment of the coursework[4]. For Honours courses (including joint honours), CS3 must be passed at the first attempt, although marginal failures on individual papers may be permitted. Resits in September are available for the Computer Science Ordinary course. A candidate who fails the Honours examination may be awarded a pass at the Ordinary degree level.

In CS4 each lecture module is examined by a separate one-and-a-half hour examination in April and by continuous assessment of the coursework. The marks for the six modules contribute 60% one half of the mark for CS4, and the project the remaining 40%. The project is assessed by the supervisor and by a second marker, and their marks are moderated by the project committee. To be awarded an Honours degree, a student must obtain a pass in the project.

[4] A class examination is held at the start of the second term, but the results are only intended to give some guide to the student's own progress.

RESEARCH STUDENTS

Student/Supervisor[1]	Project
Thorsten Altenkirch *R.M. Burstall, C.P. Stirling*	Theory and application of functional programming, type polymorphism
David Aspinall *D.T. Sannella*	Development and applications of computer-asssisted proof-checkers.
Jo Blishen *K. Mitchell*	Tools for analysis of concurrent systems and applications of theories of concurrency
Glenn Bruns *G.L. Cleland, S.O. Anderson*	Formal approaches to safety critical systems design
Albert Burger *P. Thanisch*	Concurrency control mechanisms in distributed database systems.
Louise Burnham *M. Fourman, M.F. Dam*	Formal specification and verification of systems
Claudio Calvelli *K. Mitchell, R. Milner*	Theory of concurrent systems, time-dependent concurrent systems, process-algebras, operational and denotational semantics
Pietro Cenciarelli	Theory and development of systems to support proofs, formal development and verification of programs
Chen Liang *S. Anderson, R. Candlin*	Concurrency, process algebras, specification and verification of real-time systems and models of timed calculi.
Conrad Chin *R. Candlin, N. Rothwell*	Distributed ML and the semantics of programming languages

[1] At the time of going to press, some new students had not yet been assigned supervisors

B.Sc. in Artificial Intelligence & Computer Science

The Joint Honours degree with Artificial Intelligence is designed to combine a strong basis in the practice and theory of Computer Science together with knowledge of a full range of Artificial Intelligence techniques, ranging from Vision and Robotics to Natural Language and Knowledge Based Systems. There is opportunity to study chosen areas in depth.

Artificial Intelligence & Computer Science 3

The course involves nine technical modules weighted 4/5 or 5/4 between the two disciplines, together with the Professional Issues module and two major practicals. From the Single Honours (CS3) course students take the following modules:

- Algorithms & Data Structures
- Computability & Intractability

together with two or all three of the following modules [5] together with one of the major practicals from Computer Science 3:

- Language Semantics and Implementation
- Operating Systems
- Programming Methodology

Artificial Intelligence & Computer Science 4

Students spend half of their time working on an individual project, chosen from a set of topics which relate artificial intelligence and computer science, and half on taught modules. Each student takes 6 taught modules, with at least 2 being from each department. The Computer Science modules may be chosen from the set available to Single Honours students (subject to pre-requisite third year module constraints).

[5] Students may substitute other CS3 modules at the discretion of the Head of Department, subject to timetabling constraints.

Assessment

In assessing each student's overall degree classification at the end of year 4, the mark for AI & CS3 carries equal weight with the mark for AI & CS4. In AI & CS4 the project mark is worth 40%, and the total of the examination/coursework marks, 60%.

B.Eng. in Computer Science & Electronics

Many of the advances in computer and communications technology over the past four decades have resulted from the interaction between computer science and electronics. These two disciplines therefore form a natural combination for a joint honours course, and in today's world of complex high-speed devices it is vital that there are engineers who understand how to design and implement both the hardware and software of general purpose and embedded computer systems.

Computer Science & Electronics 3

The course is weighted approximately equally between the two disciplines. From the Single Honours (CS3) course students take four technical modules plus the Professional Issues module, and choose one of the two major practicals. The technical modules must include two major practicals. The technical modules must include Computer Design, and Programming Methodology; for the remaining two modules, some degree of choice is available.

Computer Science & Electronics 4

Students take 6 taught modules, with at least 2 being taken from each department. The Computer Science modules may be chosen from the set available to Single Honours students (subject to pre-requisite third year module constraints).

The major project is normally taken in Computer Science and typically involves some computer hardware construction. Students are expected to spend between 300 and 400 hours on the project, i.e. less than Single Honours students, in order to allow time for a dissertation in electronics.

RESEARCH STUDENTS

Mr J.S. Turnbull: IS1 support, Mac, Apple fileservers and gateways, Animator generator, c-prolog, ML teaching support

Administrative and Secretarial Staff

Mrs T.L. Combe	LFCS Secretary
Mrs M. Davis	Enquiries and General Office
Ms L.M. Edgar	Secretary to Professor Fourman
Mr S.S. Falconer	EPCC/CS Accounts
Mrs A.M. Fleming	Secretary to Head of Department
Miss E.A. Kerse	Secretary to Professors Burstall & Plotkin
Ms M. Lekuse	LFCS Administration
Mrs D.A. McKie	Secretary to Professor Milner
Ms L.M. Paterson	Administrative Officer, Schools Liaison

Technical Staff

Mr J.C. Dow	Laboratory Superintendant
Mr A.M. Duncan	General workshop
Mr M.L. Graham	APM Production
Mr D.C. Hamilton	Workshop Chief Technician
Mr G. Inkster	Junior Technician
Mr J. Johnstone	Hardware systems development
Mr P.J. Lindsay	Special systems
Mr T.S. Wigham	General workshop

Assessment

For Single Honours Electrical Engineering students the weighting of the third year mark as a contribution to the final result is lower than the weighting of CS3 for Single Honours Computer Science students. CS & E3 therefore carries an intermediate weighting of about one third. The CS & E4 project is weighted at 20% of the total and the dissertation at 15%.

B.Sc. in Computer Science & Management Science

This joint degree allows students to develop a strong background across a wide range of topics in both Computer Science and Business Studies. There is a central core which develops the use of quantitative methods in management, and there are interdisciplinary links in analysis of algorithms, capacity issues, communications, modelling and simulation and new technologies.

Computer Science & Management Science 3

Computer Science & Management Science 3 may be taken in two different ways. Students take

EITHER

Management Science B [6], and eight lecture modules and the two major projects from Computer Science 3

OR

Management Science B [6], one other Business Studies half course, seven lecture modules and one of the major projects from Computer Science 3. In either case the CS modules should include:

- Algorithms & Data Structures
- Programming Methodology
- Professional Issues

Computer Science & Management Science 4

In Computer Science & Management Science 4, students spend just under half of their time working on an individual project on a topic relating

[6] or another Business Studies half course if Management Science F has already been taken.

Computer Science and Management Science and the remainder on taught modules. Each student takes 6 taught modules, with at least 2 being from each department. The Computer Science modules may be chosen from the set available to Single Honours students (subject to pre-requisite third year module constraints).

Assessment

In assessing each student's overall degree classification at the end of year 4, the mark for CS & MS3 carries equal weight with the mark for CS & MS4. The CS & MS4 project is weighted at 20% of the overall total.

B.Sc. in Computer Science & Mathematics

As Computer Science matures as a discipline, the connection between Computer Science and Mathematics deepens. On the one hand, computer scientists are increasingly turning to mathematical tools in their work; on the other, Computer Science is enriching Mathematics by providing both new problems to study, and new light on old problems. Computability and logic are inextricably linked, computational complexity interacts with combinatorics, and the theory of concurrency with abstract algebra. This joint degree aims to equip students to understand and appreciate these connections.

Computer Science & Mathematics 3

The course involves eight technical modules divided equally between the two disciplines together with the Professional Issues module. From the Single Honours (CS3) course students take the following four modules [7] together with one of the major projects from Computer Science 3:

- Computability & Intractability
- Programming Methodology
- Algorithms & Data Structures,
- Language Semantics and Implementation

[7] Students may substitute other CS3 modules at the discretion of the Head of Department, subject to timetabling constraints.

Computing Officers

Mr P. Anderson: LFCS Systems Development Manager

Mr D.W. Baines: CS2 support, Gandalf, communications support, Mail, "Skye" service

Ms J.T. Blishen: Unix system and X-terminal management (in particular HP), Concurrency Workbench

Mr J.H. Butler: Service Manager, EPCC liason, PC-NFS

Dr G.L. Cleland: Assistant Director of LFCS

Mr C. Cook: GNU, KB teaching cluster, First-line fault response, Databases

Ms C. Dow (p-t: Computing Support Officer): Faults, Documentation, Advisory, Accreditation

Mr A.J.C. Duggan: TeX, Graphics packages and course support, printers, plotters and spooling, DTP

Ms M. Findlay: LFCS workstation support, Mail

Mr A. Howitt: Hardware laboratory support, CS3/4 projects, P-CAD, Posie, IBM PCs

Mr D.J. Rogers: VLSI support, SOLO, Cadence, ELLA, Faculty representative for ECAD, Eurochip and EASE

Dr G.D.M. Ross: CS1 teaching cluster, Sun, X11, APM support, Mail, X toolkits

Mr A.J. Scobie: KB departmental Unix manager, Annexes, Source server, Unix device drivers, Technical support to Service Manager, EUCS liason

Ms J.J. Smith (Computing Support Officer): Fault reporting, CS1, Accreditation

Ms R. Soutar (p-t): CS1 support, Admin database, Compilers

Mr R.W. Thonnes: Hardware/software systems development, BEPI machine, Sparse vector machine

Dr B. Jay: *BP/RSE Research Fellow* Categorical semantics of programming languages, rewriting systems, domain theory, ordered categories.

Mr E. Kazmierczak: System specification and verification, Extended ML, theorem proving, functional programming.

Dr Z Luo: Type theories, theory of computation, model theory and semantics, program theory and semantics, program specification and development, computer-assisted reasoning.

Dr D. Matthews: Poly ML, concurrent implementations of functional programming languages.

Dr F. Moller: Process calculi, timed models of concurrency, the concurrency workbench, asynchronous circuit design.

Mr I. Omorogbe: ML and proof systems implementation.

Mr R.A. Pollack: Design of proof systems, type theory, proof theory.

Dr J. Power: Category theory and its application to computer science, logical frameworks.

Dr D. Pym: Proof theory, logical frameworks,categorical logic/models, search spaces, logic programming, linear logic, recursion theory.

Dr N.J. Rothwell: Compiling and interpreting techniques, functional programming, language theory and semantics, concurrency.

Dr M.C. Shum: Category theory, proof theory.

Mr N. Stevenson: The Integrated Modelling Support Environment Project.

Dr C. Tofts: Algebraic concurrency models including temporal, probabilistic and priority properties, correctness of real-time systems, probabilistic fault tolerant algorithms, the description of behaviour in complex biological systems.

Computer Science & Mathematics 4

Students spend just under half of their time working on an individual project on a topic relating Computer Science and Mathematics and the remainder on taught modules. Each student takes 6 taught modules, with at least 2 being from each department. The Computer Science modules may be chosen from the set available to Single Honours students (subject to pre-requisite third year module constraints).

Assessment

In assessing each student's overall degree classification at the end of year 4, the mark for CS & Maths 3 carries equal weight with the mark for CS & Maths 4. In CS & Maths 3 the contributions from the two disciplines are equally weighted. In CS & Maths 4 all modules are equally weighted and the project is weighted at just under 40%.

B.Sc. in Computer Science & Physics

Computer systems are now a crucial resource for both experimental and theoretical physics. Experiment design and control require an in-depth understanding of computer system engineering, while analysis of experimental results may need sophisticated computer algorithms. The theoretical investigations of computational physicists call for effective use of the most powerful state-of-the-art parallel computers, such as those in the Edinburgh Parallel Computing Centre, and also provide new problems and insights for computer scientists and engineers. Edinburgh University has a long tradition at the forefront of Computational Physics research, and this joint degree aims to provide the range of skills and knowledge required for a thorough understanding of this expanding field.

Computer Science & Physics 3

The course consists of four lecture modules from each discipline, one major project in Computer Science, one laboratory project in Physics, plus the Professional Issues module. The four CS3 lecture modules must include Algorithms and Data Structures, and Programming Methodology; students may choose the remaining two CS3 modules subject to timetabling constraints.

For experimental physicists, the Computer Architecture module is particularly recommended, while for theoretical physicists, the Computability and Intractability module. Of the two CS3 major projects, the Operating Systems topic is recommended as appropriate for most CS and Physics students; those choosing this project are also strongly recommended to select the Operating Systems lecture module.

Computer Science & Physics 4

In Computer Science & Physics 4 students spend half of their time working on an individual project (normally on a Computational Physics topic suggested by the Physics department) and half on taught modules. Each student takes 6 taught modules, with at least 2 being from each department. The Computer Science modules may be chosen from the set available to Single Honours students (subject to pre-requisite third year module constraints).

Assessment

In assessing the final degree classifications, the marks obtained for CS & Physics 3, the Project and the marks for the taught modules in CS & Physics 4 are weighted at 40% and 60% respectively.

B.Sc. in Computer Science & Statistics

The Computer Science and Statistics course is asymmetrical between the two honours years. In third year students take mainly computer science courses, and in fourth year mainly statistics.

Computer Science & Statistics 3

Students take Statistics 3 (equivalent to one third of an honours course) and the following modules from the Single Honours (CS3) course, together with one of the major projects from Computer Science 3:

- Operating Systems
- Computability & Intractability
- Programming Methodology
- Algorithms & Data Structures

Dr P. Thanisch: Parallel computing systems: concurrency in object-oriented languages and load balancing. Database systems: performance analysis, query omptimisation and automatic database design.

Dr N.P. Topham: Computer architecture, and parallel computers. The design of high-performance computing systems, architecture simulation tools, and quantitative analysis of high performance systems.

Dr A.S. Wight: Computer systems performance evaluation. Flow and congestion control in computer communications networks. Workload characterisation.

Honorary Fellow

Mr P.D. Schofield: The use of computers in teaching, assessment and departmental administration particularly the automatic marking of and detection of plagiarism in practical assignments. Data structures; sorting and searching algorithms.

Research Workers

Dr G. Bellin: Proof theory, mathematical logic, linear logic, program verification and extraction.

Dr D.M. Berry: Structured operational semantics, concurrency primitives for Standard ML.

Dr J. Bradfield: Techniques for reasoning about infinite state space systems, Petri nets.

Mr G. Bruns: Temporal logic, concurrency theory and their application to the design of safety-critical systems.

Dr M.F. Dam: Logics for concurrency, applications of temporal logic, modal and temporal μ-calculi.

Ms J. Hillston: The Integrated Modelling Support Environment Project.

Dr C.C.M. Jones: Mathematical applications of computer assisted formal reasoning, constructive mathematics, semantics.

Dr C. Lengauer: Programming methodology with focus on parallelism, systolic design, semantics of concurrency, applications of formal verification and synthesis methods and their implementations.

Mr R.A. McKenzie: Computer graphics: algorithms, scene rendering, object modelling on various parallel architectures. Computer architectures. VLSI design: synthesis and simulation tools.

Dr K.N.P. Mitchell: Functional programming language design and implementation; the application of operational semantics to compiler generators.

Mr R.J. Pooley: Simulation methodologies, support environments for modelling and design, performance modelling, integration of quantitative and qualitative system design approaches, object-oriented language design and implementation.

Dr R.N. Procter: Human Computer Interaction, especially interfaces for real-time control and collaborative tasks. Organisational factors in systems design and design methodologies. Social shaping of Information Technology.

Dr A.J. Sinclair: Complexity theory, design and analysis of algorithms, randomised computation and probabilistic algorithms, approximation algorithms for combinatorial problems.

Dr A.D. Smaill: Theorem proving, automated inference and its control; constructive logics.

Dr J.N. Spackman: Computer synthesis of photo-realistic images: ray tracing, and acceleration thereof. Object modelling, boundary representations, analytic set-theoretic models, oct-tree representations for efficient ray tracing.

Mr F. Stacey: All aspects of distributed operating systems, particularly the provision of file services. The specification and verification of (parallel) algorithms used in the implementation of such services.

Dr C.P. Stirling: Semantics of concurrency and the application of modal and tense logics to verification of concurrent programs.

- Language Semantics and Implementation
- Professional Issues

Computer Science & Statistics 4

In Computer Science & Statistics 4, students spend just under half of their time working on an individual project on a topic relating Computer Science and Statistics and the remainder on taught modules. Each student takes 7 taught modules, 2 from Computer Science and 5 from Statistics. The Computer Science modules may be chosen from the set available to Single Honours students (subject to pre-requisite third year module constraints).

Assessment

In assessing each student's overall degree classification at the end of year 4, the mark for CS & Statistics 3 carries equal weight with the mark for CS & Statistics 4. In CS & Statistics 4 the project carries a weighting of approximately 40%.

Resources

For practical work in CS1A students have access to a laboratory equipped with Sun workstations in the Appleton Tower. In CS2 students use facilities in the JCMB. These include a laboratory equipped with in-house designed APM (Advanced Personal Machine) workstations and access from a dedicated terminal room to a timesharing mainframe in the Edinburgh University Computing Centre.

In third year most of the programming practical work is carried out on Sun workstations in the JCMB Machine Halls, though some work is carried out in a digital hardware design laboratory.

In fourth year most of the programming practical work is again carried out on Sun workstations, though for project work students may use other computing facilities such as the multi-transputer Meiko Computing Surface, housed in the Edinburgh Parallel Computing Centre, or may use a hardware project laboratory equipped with development systems, logic analysers, CAD/CAM facilities, etc.

Communication Channels

For each year of the course a member of academic staff acts as course organiser, and he/she interacts with the students in an informal manner throughout the year on matters relating to the course. More formally there is a Staff-Student Liaison Committee for each year, to which students elect representatives. These committees report to the Teaching Committee, which is largely made up of the course organisers from all years.

At the start of each year students are given a document containing details of the course, including the timetable, syllabuses, list of relevant staff, etc. During the year, additional information is normally supplied through the *alert* mechanisms which display general messages whenever a student logs on to a computer in the department.

Involvement with Industry

There is a strong demand from industry for graduates with Computer Science qualifications generally, but Edinburgh graduates are particularly sought after by IT suppliers and industrial users of IT. As a means of attracting graduates, employers are increasingly offering sponsorship and vacation placements, and the Department's own Careers Officer (who works closely with the University Careers Service) coordinates the arrangements for these activities.

A number of industrialists contribute to the Professional Issues lectures, The Department benefits from the following prizes which are donated by industry:

Best result in CS4	Sun Microsystems
Best result in CS3	FI Group
Best CS3 Group Project	Ford Motor Company
Best result in CS & Electronics 3 & 4	Hewlett Packard
Best result in CS2	Proctor & Gamble
Best performance in CS1	Prentice Hall

Within the School of Engineering and Information Technology the Department is a member of the Information Technology Educational Advisory Board which has representation from up to 10 industrial concerns. This

Lecturers

Mr S.O. Anderson: Use of programming logics in the specification and development of verified programs; the use of Martin-Lof's type theory and the provision of proof assistants for the theory. Development of theories of particular application areas within type theory (e.g. user interface design).

Mr D.K. Arvind: Computer vision: perception of 3D scenes. Learning in parallel networks. Algorithms and environments for parallel computation. Parallel and distributed simulation.

Dr G.J. Brebner: Computer networking, protocol specification and verification, distributed computing environments. Configurable cellular arrays. Computational complexity.

Dr E.R.S. Candlin: Operating systems for MIMD machines, with particular interest in the support of process migration. Models and measurement of parallel computations.

Dr M.I. Cole: The design and implementation of programming languages and frameworks for parallel computers. The implementation of functional programs on message-passing architectures through distributed graph reduction.

Dr S.T. Gilmore: Formal methods of program development, formal specifications, software engineering, software tools,specification of concurrent systems.

Mr T.M. Hopkins: High bandwidth local area networks. Interconnection of local area networks. Network support for special purpose high performance computing engines. Specialised processor design for sparse vector computation.

Dr M.R. Jerrum: Computational complexity: data structures, efficient algorithms (including those involving randomisation), resource bounded complexity classes. Combinatorial mathematics: random graphs. Quantitative treatment of rates of convergence in stochastic systems. Learning theory.

Dr K. Kalorkoti: Computational complexity with special interest in algebraic complexity. Computer algebra. Decision problems in group theory.

MEMBERS OF STAFF

Professor R.M. Burstall: Computer-aided proof and its relation to type theory. Development of correct programs with respect to a specification. Applications of category theory in computer science.

Professor M.P. Fourman: System specification, verification and synthesis: formal models of digital behaviour and of the design process. Temporal and data abstraction as used in digital design. Design and implementation of formally-based system-design tools. Proof and proof assistants.Specification languages and their semantics.

Professor R.N. Ibbett: *Head of Department; Associate Director, EPCC* Computer and Network Architectures. Design, simulation and performance evaluation of parallel and novel computer systems.

Professor A.J.R.G. Milner: *Senior SERC Research Fellow; Co-Director, LFCS* Mathematical models of computation: models and calculi for concurrent computation; how to present logics to machines; semantics of high-level programming constructs.

Professor G.D. Plotkin: *Director, LFCS*
Applications of logic, especially denotational and operational semantics of programming languages. Semantics of type systems, monadic theories of computation, general proof theory and the semantics of natural language, particularly type-free intensional logics for situation theory.

Readers

Dr D.T. Sannella: Algebraic specification and formal program development, mechanised reasoning, programming methodology and functional programming languages.

Senior Lecturers

Dr D.J. Rees: Computer Aided Design tools for VLSI design. Silicon compilation from behavioural descriptions. Hardware description languages. Silicon assembly. Novel VLSI architectures. Configurable cellular arrays.

Board meets annually to review strategic issues in IT Education within the departments concerned. Members of the Board are also invited to attend the CS3 Group Project presentations.

Accreditation

The Department was visited by an Accreditation Panel of the British Computer Society in February 1988. Graduates from courses accredited by the Society may be exempted from Parts I and/or II of the Society's Professional Examinations. Candidates for professional Membership of the Society are required to have passed or been exempted from Parts I and II and subsequently to have completed four years of monitored work experience.

As a consequence of the 1988 visit, accreditation was granted as follows for all student intakes up to and including that of 1992:

B.Sc/B.Eng. (Hons) in Computer Science	Parts I & II
B.Sc/B.Eng. (Ord) in Computer Science	Part I
B.Sc. (Hons) in Artificial Intelligence and Computer Science	Parts I & II
B.Eng. (Hons) in Computer Science and Electronics	Parts I & II
B.Eng. (Ord) in Computer Science and Electronics	Part I
B.Sc. (Hons) in Computer Science and Management Science	Part I
B.Sc. (Hons) in Computer Science and Mathematics	Part I
B.Sc. (Hons) in Computer Science and Physics	Part I

In addition, the B.Eng in Computer Science and Electronics is accredited by the IEE for student intakes up to and including 1992. Graduates from this course with first or second class honours degrees meet the educational requirements for corporate Membership of the IEE.

MEMBERS OF STAFF

tured to exploit hidden parallelism, or new algorithms have to be designed. Of particular interest are algorithms for linear algebra, which are used in many areas. Research in EPCC is focusing particularly on parallelising traditional optimisation techniques.

Parallel Database Applications

EPCC is looking at the parallel evaluation and optimisation of complex queries. This work will provide algebraic operators for certain classes of queries which permit efficient parallel evaluation. Also, optimisation techniques will be developed for expressions involving these operators.

POSTGRADUATE STUDY

The Centre has a flourishing Student Scholarship Scheme. The number of scholarships has grown from two, in 1987, to this year's figure of 26.

Research Programme

The research programme of the Centre includes the SERC-funded projects, some projects supported by the IED and by industry, and relevant projects within Computer Science and other departments. The main areas of research in the Centre include the following:

Universal Models of Parallel Computing

Models of parallel computation are essential for algorithmic development, especially given the rapid evolution of hardware. However, many of these models make unrealistic assumptions, i.e. that all memory is shared. EPCC is currently studying techniques which allow efficient emulation of idealised parallel machines on realisable ones, with the aim of providing a bridge between theory and practice.

Mapping Parallel Computations

Task and process based models are often used in addressing the problem of associating computation with processors. While task based models with communication delays seem to be the most generally applicable, neither are entirely appropriate, and work at EPCC is attempting to characterise the range of their respective applicability.

Programming Environments

One of the greatest obstacles to the wider use of parallel computers is the lack of simple, yet powerful, programming tools for them. Ideally, a complete toolset would allow programmers trained on serial machines to write, debug, profile and blaise programs on a wide range of parallel platforms. One of EPCC's goals is to provide guidance on which tools are appropriate, and can be implemented efficiently on real machines.

Numerical Algorithms

Many algorithms for numerical analysis were designed with a sequential model of computing in mind. Many of these algorithms have to be restruc-

Organisation and Management

EPCC is supported by major grants from the Department of Trade and Industry, the Information Systems Committee of the UFC and the Science and Engineering Research Council. It also receives substantial support from the University and from its industrial affiliates. It is run under the auspices of an Advisory Board made up of representatives from the funding bodies, the University, industry and users.

The work of the Centre is overseen by the Director, (Professor D.J. Wallace of Physics), who works in consultation with an Executive Committee which includes the Chairman (Professor J. Collins, formerly of Electrical Engineering), the Assistant Director for Research (Professor R.N. Ibbett of Computer Science) and the Director of the EUCS and a representative from the DTI. The Committee meets regularly to review research, resource allocation, industrial involvement, and bids for future funding.

Membership of the Centre is open to all staff and research students whose work relates to the interests of the Centre. Many members of the Department of Computer Science are also members of the Centre.

The Work of the Centre

Work in parallel computing in the University covers a wide span of activities ranging from long-term research on the theory of concurrency (in the LFCS), through to applications development by a wide range of users within the University and elsewhere.

The Centre provides a service for users on the ECS and DAP facilities (with EUCS support), together with network access to these machines for both local and remote users connected over JANET, and user support relevant to these facilities. Its most recent acquisition is a Connection Machine as part of a collaboration agreement with its manufacturer, the Thinking Machines Corporation of Cambridge, Mass.

With funding from the SERC the Centre is pursuing research on support tools for users of parallel computers. This work complements the research work on parallel systems in the Department of Computer Science. The Centre produces a series of technical reports describing its research and organises a regular series of lunchtime seminars, at which speakers present short talks on current aspects of their work. (It also publishes a quarterly newsletter and runs regular courses on all aspects of parallel computing).

POSTGRADUATE STUDY

The Department offers facilities for postgraduate study by research leading to the degree of Ph.D. and for study towards the degree of M.Sc. in Computer Science. Study for these degrees is normally undertaken on a full-time basis, but it is also possible to study for them on a part-time basis, over longer periods of time.

Degrees by Research

Students studying for the degree of Ph.D. undertake full-time, supervised research for a minimum period of 36 months. This research will normally be in an area of existing research interest within the Department (outlined later in this Handbook) but proposals for research in other areas may be considered.

The minimum period of study includes a first year of study in which students are registered as *Supervised Postgraduate Students* prior to registration as Ph.D. candidates. During this first year, students are expected to participate in an appropriate study programme. For students registered for research in the computational theory area, a specific course on advanced topics is provided. Students intending to pursue research into computer systems or other non-theoretical topics may be directed to particular advanced courses and are expected to attend postgraduate study seminars. In order to register as a Ph.D. candidate, students must submit for approval a research proposal which is discussed with a small panel of staff. The degree itself is examined on the basis of a thesis and oral examination.

M.Sc. in Computer Science

The Department offers an M.Sc. in Computer Science which takes the form of one of three themes: Computer Systems Engineering, Parallel Systems, and Theoretical Computer Science.

The course runs from October to September. During the first half of the academic year, eight chosen lecture modules are studied and examined, four of which must be 'core' modules within one of the themes. The following six months are spent on a full-time project, examined by dissertation, usually supervised within the department. Most lecture modules have practical work associated with them. The modules currently offered are:

Computer Systems Engineering

- ASIC Design
- Behavioural Specification and Verification
- CAD/CAM
- Systems Integration
- Systolic Design

Parallel Systems

- Concurrent Architectures
- Computer Communications
- Distributed Systems
- Software for Parallel Computers

Theoretical Computer Science

- Algebraic Complexity
- Applicative Programming and Specification
- Communication and Concurrency
- Computational Complexity
- Computer Aided Formal Reasoning
- Formal Programming Language Semantics

General

- Software Systems Concepts
- Graphics
- Operating Systems
- Database Systems

THE EDINBURGH

PARALLEL COMPUTING CENTRE

The University of Edinburgh has a long history of involvement in parallel computing and is one of the largest centres of research in the theory and application of parallelism in Europe. Work started in the early 1980s with the use of ICL Distributed Array Processors (DAPs) by the Departments of Physics and Molecular Biology. When a Meiko Computing Surface was acquired in 1986, the Edinburgh Concurrent Supercomputer Project was set up. In 1989, as a way of bringing more closely together the many different aspects of parallel computing research being carried out in the University, the Department of Physics entered into a collaborative venture with the Department of Computer Science and the Edinburgh University Computing Service to establish the Edinburgh Parallel Computing Centre (EPCC). The major goal of EPCC is to ensure that as parallel computers become generally accessible, computational scientists and engineers become able to think naturally of solving their problems on parallel machines, and have the tools and skills they need to do this. To achieve this goal, EPCC supports the applications work being carried out on parallel systems and is investigating those fundamental aspects of parallel computing which can lead to the development of new and improved user support tools.

The Centre is interdisciplinary and acts as a focus for work being carried out within a number of departments. It has subsumed activities on the Edinburgh Concurrent Supercomputer (ECS) and the AMT DAPs. Other groups within the University, including the Parallel Architectures Laboratory in the Department of Geography and the Biocomputing Research Unit, contribute to the work of the Centre and are able to offer a variety of additional facilities to members of the Centre.

An increasing amount of EPCC's work is being done through industrial and commercial contracts and collaborations. An Industrial Affiliates scheme was set up in 1987 and has already attracted 20 members. EPCC has also been chosen as one of the four UK centres taking part in the DTI/SERC Parallel Applications Programme, which aims to bring together pre-eminent centres for parallel processing, industrial and commercial end-users, system vendors and software houses, to produce applications and tools. EPCC has just been awarded £3.5 million, and plans to attract industrial sponsorship to bring the total to £9 million over the next four years.

RESEARCH IN
COMPUTER SCIENCE

THE EDINBURGH
PARALLEL COMPUTING CENTRE

to their own work that this brings it also provides valuable feedback to the research of the Laboratory and deepens the Laboratory's understanding of the needs of Industry.

Some **Collaborative Projects** are in place and others will follow. These allow Industry and the Laboratory to work jointly on a project – the Laboratory providing a rich environment and a wide range of ability with the industrial partner providing the 'real world' application and the commercial incentive to use it. The result of this kind of project is rapid transfer of skills and systems to the industrial environment and feedback to the Laboratory on the usefulness and the applicability of its work. These effects are similar to those of the Industrial Visitors scheme, but in greater depth.

The groups involved in active collaboration include British Rail (Safety Critical Systems Design), BP (Concurrency), DEC (the Pebble Language and Concurrent ML), AT&T Bell Laboratories (ML Compiler), Harlequin (ML and Specification Systems) and Abstract Hardware (Proof Systems).

The Laboratory publishes a report series comprising both research and expository material. Copies of these and other publications such as the Annual Report and the Prospectus can be obtained from George Cleland, the Assistant Director, from whom further general information on the Laboratory may also be obtained.

Research in Computer Science

Computer Science is a subject which is less easily compartmentalised than most other scientific and engineering disciplines, and so although researchers in the Department form themselves into groups such as the Complexity Group, the Parallel Computing Group, the VLSI Group, and the various 'clubs' within the Laboratory for Foundations of Computer Science (which itself organises a coherent subset of the research activities of the Department as a whole), there is considerable cross representation, and the research of any one group may well draw on the experience and expertise of others. Research in computer systems, for example, involves a number of interrelated activities, including work on architectures, networks, languages, etc., and the use of VLSI, no one of which can be carried out independently of all the others. The division of research into the areas described here does not therefore represent a strict compartmentalisation of activities within the Department. It shows, rather, the major emphases of one or more projects contributing to the overall programme of computer science research.

Communication Protocols

Research in this area concentrates on the rigorous specification and verification of practical communication protocols. Within the Department, the particular concern is to bridge the gap between work on practical protocols currently in use and work on concurrency (e.g. the Calculus of Communicating Systems (CCS) and Concurrency Workbench described later). Major areas of work are the decomposition of protocols in terms of common communication paradigms, and an investigation of the extent to which computer assistance is possible when verifying protocols. Co-operation with other researchers includes work related to LOTOS, the international standard protocol specification language, which is based on CCS.

Computational Complexity

Computational Complexity is the quantitative study of the 'inherent difficulty' of solving computational problems. The study is motivated by the empirical observation that the resources required to accomplish various computational tasks vary dramatically between tasks. The meaning of 'resource' depends on the setting, but typical examples are time, space or hardware

size. The study is wide-ranging, and its techniques rest on increasingly sophisticated mathematics.

Algebraic Complexity

Algebraic complexity studies intrinsic requirements in the computation of algebraic functions, such as matrix multiplication, polynomial evaluation and interpolation. A machine independent model is used in which the basic operations are addition, subtraction, multiplication and division (sometimes extended or even contracted). Most work assumes a uniprocessor, although parallelism is also studied.

The most highly developed area is concerned with bilinear forms. However, despite substantial progress, non-linear lower bounds are still elusive. For example, the best known lower bound for matrix multiplication over arbitrary fields is linear and has seen no improvement for at least ten years (all the advances in this much studied topic have been on upper bounds).

The only general technique for non-linear lower bounds has its basis in Algebraic Geometry. This has yielded optimal results for such problems as polynomial interpolation, but in the case of bilinear forms it is of no help. There is a pressing need for more techniques, especially ones which can handle bilinear problems.

Algorithms and Data Structures

In contrast to algebraic complexity, the problems here are combinatorial, and the model of computation is machine-based. The problems considered come from many areas, including graph theory, combinatorial enumeration and geometry. A particular concern at Edinburgh has been the study of *randomised* algorithms whose progress is influenced by the outcome of a sequence of coin tosses. Is it possible that randomised algorithms have greater computational power than deterministic ones? Surprisingly the answer appears to be 'yes', and several probabilistic algorithms have been discovered which are faster than the best known deterministic counterparts.

Recent work has focused on randomised algorithms which involve the simulation of an appropriate Markov chain. Such algorithms arise in the analysis of systems in statistical mechanics, and in stochastic optimisation techniques such as simulated annealing. Moreover they represent the only known class of efficient approximation algorithms for a number of problems in combinatorial enumeration.

language.

Collaborative research is being conducted to demonstrate the application of LFCS theories and method to industrial problems. Areas being tackled include systems design (ICL) and safety critical systems (British Rail, Adelard and Praxis).

The intense systems work in LFCS requires powerful state of the art computing facilities. As well as being supported by both the University and funding agencies, Digital and Hewlett Packard have each provided significant resources.

Much of the systems work of LFCS is exported through its system dissemination programme. Systems are available at cost for research or educational use, with commercial use being subject to individual licencing agreement. A number of sub-licencing arrangements are in place.

Industrial Interaction

The element of the Laboratory which distinguishes it from merely being a collection of research projects is the formal structures which it uses to interact with industrial research organisations. This takes place in a number of ways:

The **Affiliation Programme** provides, to Industrial R & D groups, a low cost means of keeping in touch with both the work of the Laboratory and with other significant developments through means such as: distributing copies of all LFCS reports (these include both research reports and expository publications); regular conferences for Affiliates; access to the Laboratory's prototype software systems; bulletin boards; informal discussion of affiliates problems, etc.

The Laboratory's **Course Programme** is now strongly developed with about seven or eight courses run each year. These range from practical courses in Functional Programming or Interactive Theorem Proving to those on theoretical topics such as Type Theory, Domain Theory and Denotational and Operational Semantics, all of which are now seen as essential in understanding the structure and behaviour of systems. The courses are intensive and are continually being updated to reflect current research.

Industrial Visitors are encouraged to spend periods of time in the Laboratory. These are typically industrial researchers intent on transferring the current technology of the Laboratory back to their parent company. They study both the work of the Laboratory in breadth and also apply the theories and methods of the Laboratory to particular problems. Besides the benefit

and external academics and industrialists helps provide a strategic direction. The research committee, composed of all academic staff members of LFCS, and the business meeting, composed of all members of the Laboratory, meet regularly and provide further input to the running of the Laboratory.

LFCS Research Programme

The mainspring of research in LFCS is the study of theories which underlie, or should in future underlie, the analysis of computing systems. Many theories are under present study: general semantic theories, theories specifically for concurrent systems, theories for system specification, theories of programming language design, and theories of general logic (to underlie computer-assisted system analysis). In almost all these cases, a significant software tool has been or is being built to mediate the theory to applied computer scientists, including industrial scientists.

The individual areas of research are described elsewhere in this document but it is worth noting that they unite into a large and coherent research effort. This effort has a core of theoretical research and a practical component which explores application and implementation of the theory. The research is unified by a common culture. It is also unified by the predominant use of the same programming language (Standard ML) in software development. Standard ML is in widespread use in Academia and Industry, particularly among those who wish to build upon LFCS research.

The practical part of the research is typified by constructing tools for proving properties of concurrent systems or for constructing and analysing specifications of complex systems. These systems or their descendants are typically used by software engineers and provide methods which allow rigorous proof of properties of the resulting system. These tools must support reasoning in a variety of mathematical logics, and almost half of the Laboratory's resource is committed to projects concerned with either building the interface mechanisms for constructing proofs or tackling the mathematical problems in integrating a number of logics into a single framework.

Investigation into Programming Methodology has resulted in the functional programming language Standard ML, which is now used for most of the Laboratory's programming. It has won wide acclaim and in 1987 was awarded the British Computer Society Award for Technical Achievement. Work in this area continues with formalising the semantics of this and other languages along with further language development both of ML and of development of tools and methods for formal program development in the

Computer Architecture and Computer Systems

The Department has been active in computer systems design for many years and has expertise in operating systems, language support and hardware design and implementation, particularly in relation to local area networks, and personal workstations. At any one time a variety of small projects are undertaken in this general area, together with one or two more major ones. Interest is focusing increasingly on parallel systems and components for constructing them.

The Sparse Project

The Sparse Project is concerned with the design and implementation of hardware and software for a sparse vector processing system. The hardware is incorporated into an existing workstation where it acts as an accelerator providing specialist support for the processing of sparse vectors. The project also involves work on compilers and on language extensions to give programmers access to the sparse vector support mechanisms provided in hardware. The applicability of these mechanisms to other types of computing is also being investigated.

Architectural Simulation

A computer architecture and its implementation can be represented in a number of ways and at different levels of abstraction. At the highest level are multiprocessor systems, while at the lowest an architecture consists of an ensemble of transistors on a VLSI chip (or set of chips). A variety of simulators has been designed for each different level of abstraction, and some design tools can generate implementations automatically at lower levels. The Hierarchical Architectural Simulation Environment (HASE) project aims to allow designers to create architectural designs at different levels of abstraction, to explore designs through a graphical interface system and to invoke the appropriate simulator at each level. Working at the processor architecture level, for example, a designer could investigate bottlenecks in the flow of instructions and try alternative design strategies to eliminate them.

The component parts of a computer can be treated very naturally as objects, and so the environment is being created using the object oriented simulation language Sim++. The environment will include object libraries,

design editors etc., and instrumentation facilities to allow performance measurements to be derived from simulation runs.

Concurrent Architectures

Research in this area is centred around the notion of semantically-driven architectures, an implementation technique for parallel architectures in which concepts of action semantics are used to decompose the structure of conventional uniprocessor into three closely coupled and highly specialised subprocessors which in combination form a high-performance MISD architecture. Active research topics include: semantically-driven architectures with superscalar instruction issue; compilation for semantically-driven machines and memory subsystems for semantically-driven architectures. Tools for investigating these machines are under construction, and include a low-level simulator and compiler.

Concurrency and Communication

Edinburgh's work on models for concurrent communicating programs and hardware arose in the 1970s, in the course of trying to develop an all-embracing semantic theory for computation. It was soon found that existing general theories did not naturally embrace concurrency. This led to the theory of power domains, and to the Calculus of Communicating Systems (CCS). In the past few years this foundational work has been broadened to include new elements, both theoretical and practical. On the theoretical side, significant progress has been made with various logical approaches and the algebraic theory has been enriched. On the practical side it has led to the development of the Concurrency Workbench.

The Concurrency Workbench

The Concurrency Workbench, a set of formally based tools for analysing and designing concurrent systems, has been built by a collaboration between Edinburgh and Sussex Universities and the Swedish Institute of Computer Science. In addition to a number of facilities for examining the operational behaviour of CCS terms, the Workbench incorporates equivalence checkers and preorder checkers that test for various relations between CCS terms. Also it includes a temporal logic model checker that tests for temporal properties of CCS processes. Furthermore, the system includes a feature which

LABORATORY FOR FOUNDATIONS OF COMPUTER SCIENCE

The groups which now form the Laboratory (LFCS) have a long record of seminal research into the Foundations of Computer Science. This has included both research into the fundamental theory and, in amplification of this, construction of prototype systems.

This work has exerted a strong influence on the development of Computation Theory particularly in the fields of semantics, specification, machine assisted proof and implementations (both of languages and of proof systems). It is naturally the software products which have been most visible, in particular the functional programming language Standard ML, which is now in use in many locations around the world both in educational and in research and development environments. Methodologies have also been exported; examples are the Specification Language CLEAR and the algebraic Calculus of Communicating Systems.

Against this light the Laboratory was formed in January 1986 with the twin aims of intensifying the research and establishing formal links with industry to ensure the application of this work in practical environments. Significant funding from the Science and Engineering Research Council, national and international strategic programmes in Information Technology and Industry has been awarded to establish the Laboratory and strengthen its research programme. After its initial period of operation it is approaching its planned size and is generally accepted as one of the leading centres in the world in Theoretical Computer Science and its application.

LFCS Structure

The Laboratory is a research unit of the University, but contained within the department. It has currently about 75 members: 12 members of teaching staff, 25 research fellows, 8 technical and administrative support and about 30 research students whose interest lie within the research areas of the Laboratory. In addition there are a number of other members of the University who are associate members of the Laboratory (from the Artificial Intelligence Department and the Centre for Cognitive Science).

The Laboratory is managed on a day to day basis by its Directorate: Prof Gordon Plotkin, Director; Dr. George Cleland, Assistant Director; and Profs. Robin Milner and Rod Burstall, Co-Directors. An Advisory Board composed of representatives of the rank and file of the Laboratory, internal

supports the hierarchical development of complex systems by means of interactive equation solving.

The Workbench has been used to good effect as a teaching aid on courses offered to industrial and academic computer scientists by the LFCS, and has been the topic of seminars for postgraduate students. It has recently been introduced to undergraduate teaching. The analysis and verification facilities have been successfully used to verify a number of classical mutual exclusion algorithms and to analyse communications protocols (one result of these analyses being the discovery of deadlocks in a multi-process implementation of a particular protocol).

A front-end has been constructed which extends the Concurrency Workbench to cover CCS with a restricted form of value passing; this brings a much wider class of applications within its scope. A procedure for minimizing the state space of a process with respect to a modal formula has been implemented. It is also planned to extend the syntax of the process language, which is currently restricted to CCS, by adding operators from other languages, such as CSP and ACP. Finally, the existing user interface is being extended with a graphical interface.

Foundational Models

There is a continued strong interest in establishing foundational models, including relating different foundational theories of concurrency such as the algebraic theory of concurrency, net theory, and power domains. Work is progressing on probabilistic versions of Plotkin's power domains. An ESPRIT Basic Research Action seeks to unify various concurrency theories, particularly the algebraic ones, in collaboration with Oxford, Amsterdam, Sussex, SICS (Sweden), and INRIA (Sophia Antipolis).

Algebraic Process Calculi

There have been new results extending and elaborating Milner's CCS. A long-standing open problem has been settled (negatively), namely the problem of whether the algebra of finite processes under a certain natural set of operators can be finitely axiomatised. The phenomenon of divergence (infinite internal computation) has been further examined. This work has established a complete algebraic theory (for finite-state processes) for the appropriately refined notion of observation equivalence. CCS has been extended to "mobile processes", i.e. to include the passing of labels as parameters;

this has important applications to systems (e.g. operating systems) in which processes migrate among processors. Work on formally relating imperative concurrent while languages to CCS is also being conducted. An emerging major theme in language research is to develop a common theoretical base which fuses functional programming and concurrency.

Logics of Concurrency

Hennessy-Milner logic has been extended to include fixpoints. This results in a natural temporal logic for CCS. A new model-checking algorithm has been discovered, and implemented on the Concurrency Workbench. Characteristic formulae for CCS processes have been constructed. General work on defining modal and temporal logics has been undertaken (on relevant modal logics). Interest has also extended to Hoare Logics of concurrent while programs. A study of compositionality in reasoning about concurrent systems is being pursued using these logics.

Declarative Languages and Applied Semantics

Standard ML

ML has evolved over a period of about fifteen years. Since 1985 there has been a major effort at Edinburgh to give a precise definition of the syntax and semantics of the language, both of the core language and of the modules language, which is based on David MacQueen's original modules proposal. This work is now completed and in 1987 Robin Milner and Rod Burstall were awarded the BCS Technical award for 'Standard ML'.

Currently, research is focussed on integrating concurrency and ML. Several implementations of ML have included experimental features for creating processes and passing values between processes. However, none of these have been given a formal semantic definition. The ML team has developed a simple, but expressive, description for a powerful set of concurrent operations, and are developing proof techniques to support reasoning about this definition.

Having developed a concurrent extension of ML on a shared memory system (based upon the POLY/ML system), the ML team is now working on a support for Standard ML on distributed memory systems, such as networks of workstations or transputer arrays. Part of this work involves proving that the synchronisation algorithm implements the formal semantics correctly.

LABORATORY FOR FOUNDATIONS OF COMPUTER SCIENCE

The ML team is also exploring ways of defining its concurrent extension of ML in terms of the Π-calculus of Milner, parrow and Walker, which provides a common framework for functions and communicating processes. This approach could yield a semantics that better supports formal reasoning about concurrent programs.

A Support Tool for Operational Semantics

The experience of defining Standard ML in terms of operational semantics highlighted the need for tools to manipulate and reason about semantic descriptions. An attempt is being made to encode the operational semantics formalism as a logic in a theorem prover such as HOL. This will be augmented with a front end that will let the language designer enter and manipulate semantic rules in their usual syntax. Proof tactics will be developed to support reasoning about large examples in this system, with the eventual aim being to handle large parts of the definition of Standard ML.

Categorical Semantics

A categorical semantics for a programming language interprets programs by the morphisms of a category. Current research at the University of Edinburgh (funded by BP and The Royal Society of Edinburgh) considers interpretations in *ordered* categories, so that programs having the same denotation (or satisfying the same specification) can be ordered according to their ancillary properties, e.g. reduction order (in a rewrite system), resource use, degree of determinism. The intention is to combine the operational flavour of the order with the powerful mathematical understanding of the usual unordered semantics.

Other recent work uses elementary limits and colimits to provide clean definitions and manipulations of datatypes. Examples include the use of pullbacks and express side-conditions, and invariants of loops to interpret tail recursion.

Axiomatic and Categorical Models for ML

It is important that new users can be introduced to ML without having to master the technicalities of the formal semantics. An attempt is currently being made to formalise naive models of ML which provide a sound basis for working with portions of the language without all the machinery needed for the formal semantics of the language.

Computer Graphics

Computer graphics research interests are concentrated in two main areas: graphics on parallel machines and graphics in a functional environment. Wider access to such computations has spawned a resurgent interest in 'visualisation' - graphical presentation of models and experimental data. A wide range of graphics and imaging software has, and is being developed for most users of the parallel machines. Both large and small grain parallelism is available and in use.

On the functional programming theme, which is also a major part of the work of this Department, there are a number of interests in computer graphics. Work is being undertaken on a graph editor which employs animated graphics to reason about running processes which are typically defined as finite-state machines. Both hardware and software is being developed to support graphics through a pure functional language. Work is beginning in persistent, object-oriented graphics where any graphics construct from a point to a complex three-dimensional model is treated orthogonally as an entity on which a set of functions can be performed. Not all functions are appropriate to all objects. Descriptions of the graphics objects are held permanently and updated when necessary.

Parallel Computing

Parallelization Tools

Parallelization tools form essential components of the support environment on parallel MIMD computing systems, but on many systems these tools are still either non-existent or relatively unsophisticated. Parallelisation tools can be categorised as either synthesis tools or analysis tools. Synthesis tools may be interactive or automatic, and both categories are being investigated. It is expected that the design of automatic tools will benefit considerably from experience gained with the use of the interactive tools. Thus the design of parallelising compilers will benefit from experience gained with code browsers, and automatic process/data mapping tools from experience with process/data placement aids.

The analysis tools will gather data during the running of programs and provide various types of measurement relating to the performance of programs and the details of run-time activity within them. Such measurements will provide feedback on the efficacy of the synthesis tools and will also lead

project is studying the problems organisations face of meeting the growing demand for specialist IT expertise, and integrating it effectively with the more traditional forms of business expertise such as that held by managerial and administrative staff. Case studies of firms in the financial services sector are being used to explore the difficulties of communication across expertise boundaries.

The concept of the 'hybrid' engineer, offering a combination of business and IT skills, has been proposed as a solution to many of the problems organisations face when trying to exploit new developments in IT. As a follow-up to the present work, a research programme is being planned to study how such hybrid skills might best be acquired. In particular, it will seek to contrast the effectiveness of educational provision with occupational experience.

design. This work is applied to the development of sound design methodologies and to the development of design tools for interactive and automated behavioural synthesis and verification of digital systems, both hardware and software. These tools will provide high-level interfaces to the silicon compilers described above.

Research is also being carried out, funded by the Alvey Directorate (Software Engineering) and in cooperation with the University of Strathclyde, on the modelling capabilities of CIRCAL, a model of concurrent systems, both hardware and software.

Novel Architectures

A cellular array structure has been developed in which the array elements are simple but configurable logic blocks. Not only is the function of each element configurable, but also its communication with adjacent elements. The increasing density of fabrication technologies means that increasingly large arrays of elements can be fabricated, and so realistically-sized computations can be mapped onto the arrays with large speed gains resulting from concurrency. Various applications of arrays are being investigated, including the design of program-specific arithmetic pipelines, the emulation of hypercube algorithms and the implementation of data compression and encryption algorithms.

Inter-Disciplinary Research

Technical knowledge alone is seldom an adequate foundation for either the design of computer systems, or the evaluation of performance. Many of the shortcomings of current systems may be attributed to a failure to recognise this fact. Understanding users and their working environment is critical to the success of any system. This aspect of system design is addressed by Human-Computer Interaction (HCI). The study of HCI is inter-disciplinary in nature. It covers issues ranging from technologies and techniques for interaction, to user cognition, perception and learning, and the context of computer use, including job design and organisational factors.

Amongst current areas of research interest are computer support for collaborative tasks, and interface design for real-time control of scientific instrumentation.

Research into organisational factors is being undertaken in partnership with the University's Research Centre for Social Sciences. One current

to a better understanding of parallel computation generally.

Parallelization Tools for Object-Oriented Programming Systems

The provision of general purpose parallelization tools, though desirable, is expected to occur through a generalisation of special purpose tools for particular languages and applications. The work here involves an investigation of the suitability of object-oriented programming languages for parallel computer systems and the design of special purpose tools for use in non-numerical applications. These tools will be restricted in the sense that (a) they are intended for use by an expert programmer who is assumed to have a detailed knowledge of the computational behaviour of the objects; (b) the objects themselves are assumed to be *persistent*, in the sense that their lifetime is typically longer than that of the execution of the applications software; (c) the class hierarchy and the class composition hierarchy are assumed to be relatively stable.

Skeletal Parallel Programming

Many efficient algorithms for message-passing parallel computers share common patterns of data distribution, process distribution and communication. The complexity of programming such machines can be reduced (for some problems) by defining a library of such patterns of computation, expressed imperatively as equivalent sequential program skeletons, or declaratively as higher order functions. Work in this area involves the collection of a set of such structures and an investigation of the implementation decisions involved when a program is constructed as a complex composition of several skeleton instances. A possible implementation medium, springing from the functional characterisation, is to describe each skeleton as a statically distributed graph intended for evaluation by parallel graph reduction. An associated problem is, therefore, to investigate how closely a distributed graph reducer can be made to emulate the dynamic behaviour (in terms of movement and generation of data) of the explicitly parallel algorithm to which the initial graph distribution is intended to correspond. Such experiments are being conducted on a transputer-based Meiko Computing Surface.

The POSIE Project

The POSIE project is directed towards designing operating systems that efficiently support concurrent computations on multiprocessor machines. There are a number of strands to this work.

A hardware test-bed with real-time monitoring facilities has been constructed, and an operating system designed for it. This will permit non-intrusive monitoring, and open up the possibility of effective dynamic load-balancing by process migration. Work is currently being carried out to evaluate the usefulness of such a feedback system in improving performance. The monitor can be made available to an individual user who wishes to tune up a particular program, and the problem of filtering out significant events and presenting information in a helpful way is being investigated. Research is also being carried out to find strategies for automatic run-time reallocation. This ties up with work on the statistical properties of parallel programs and the attempt to find useful quantitative models for predicting program performance.

Simulation is a very useful tool for finding those properties which most affect performance. A number of simple models have been constructed which represent particular types of parallel program, and an investigation is being carried out to see how well they respond to real programs. The aim is to characterise a program in terms of a small parameter set, and to find the most effective allocation strategies for different classes of program. By deriving parameter values at compile-time, or measuring them at run-time, it would then be possible to select an allocation strategy that would have a high probability of leading to good performance.

A simulation tool has been developed which supports the easy modelling of different types of machines and programs. Current work is concerned with evaluating different compile-time mapping heuristics for programs which have a stable behaviour, and with extending the technique to handle run-time reallocation to improve the performance of time-varying programs.

Performance Modelling

The emergence of practical distributed and parallel computer systems is generating new classes of problem, which are not solvable by existing analytic and numerical methods and are driving the search for new solution methods and more importantly focussing attention again on simulation programming.

are linked together by channels in a particularly regular structure. Typical applications are highly repetitive algorithms on large data structures such as those which occur in image or signal processing, meteorology, etc.

Automatic methods distribute the operations of source programs that do not specify concurrency or communication into time and space (*systolic design*). The target descriptions of these methods are distribution functions that form an abstract specification of the systolic array. The array can then be refined into software (i.e. into a distributed program) by a process of *systolizing compilation* or into hardware (i.e. into a chip layout).

Current research includes (1) the systolization of application problems, (2) classifications and extensions of systolic design methods and (3) the development of systolizing compilation techniques. An implementation with graphics facilities of a method that transforms imperative programs into systolic arrays is in use and systolic programs are being run on the Meiko Computing Surface in EPCC.

Very Large Scale Integration

The main themes of activity are Computer Aided Design (CAD) of VLSI circuits and novel VLSI Architectures. CAD tools being developed include those involving the use of formal techniques for synthesis and verification of designs. Behavioural (high-level) synthesis is also under investigation.

Formal Techniques

Current CAD tools for VLSI manage complexity by manipulating a structural hierarchy. To manage the increasing complexity resulting from technological advances of VLSI fabrication, tools are required which can manage the hierarchy of behavioural abstractions used in system design. A formal model which naturally and accurately represents circuit behaviour is central to such tools.

Behavioural models with the generality necessary to describe behaviour at many levels have been introduced by researchers studying formal verification of hardware designs. In this work, techniques from formal logic have been used to establish the correctness of a circuit design by mathematical proof prior to fabrication.

The major thrust of current research at Edinburgh involves the development, in cooperation with industry, of practical formal models of digital behaviour, and of the temporal and data abstractions employed in system

Mathematically Proven Safety Systems (MPSS)

Safety Critical Systems are well suited to formal modelling and analysis. They are usually small, precisely (though informally) specified, and are developed by engineers who are open to any methods which might help to increase confidence in the safety of systems.

A principal aim of the MPSS project is to gain empirical experience of the application of theories, methods and tools developed within the LFCS to "real life" problems. Case studies are being developed cooperatively with industrial groups working on high integrity systems. Initially the focus is on the application of process calculi: CCS, timed CCS, and the π-calculus. Already experience with the Concurrency Workbench has highlighted its strengths and weaknesses as an analysis tool and some work on extensions has begun.

As the project develops, re-use of theories, proofs and system components will become a major focus. This later phase of the project will draw on the proof- and model-theoretic work of the LFCS as well as continuing work on the application of process calculi. The identification of classes of system and their uniform treatment is an important aspect of this work.

The analysis of a message passing protocol used in a reactor safety system is already complete. This analysis uncovered a failure of "liveness" under certain circumstances and suggested a modified protocol. Currently a number of case studies are being developed in collaboration with British Rail. These include: formal modelling of signalling systems, a semantic analysis of a signalling specification language, the analysis of multi-media communication protocols, and an investigation of the use of replication to obtain high availability.

Formal and Mechanical Methods of Parallelisation

Programming should be a problem solving activity. Issues that have nothing to do with the problem should, if at all possible, not be the burden of the programmer but should be part of the compilation process. In many applications, parallelism and communication are implementation, not problem solving issues. In this sense, they are 'low-level' concepts like gotos or pointers, except that they are more difficult to use and verify.

There are classes of programs into which maximum parallelism can be infused mechanically. The resulting parallel program emulates a *systolic array*. A systolic array is a distributed network of sequential processors that

Also the arrival of powerful and inexpensive workstations with good graphics capabilities makes it worthwhile to explore the construction of novel tools to support simulation and analysis.

The performance group at Edinburgh is involved in an ESPRIT 2 funded project 'Integrated Modelling Support Environment', where Edinburgh is a collaborator with organisations from five European countries. The project involves the design and implementation of an Integrated Modelling Support Environment (IMSE) which includes novel tools to support systematic modelling and experimentation. The design is intended to be readily extensible. Edinburgh is particularly involved in a tool to support experiments.

Related work includes performability and qualitative modelling, with particular interest in Petri nets and timed process algebras, such as TCCS.

Proof Engineering

IPE: An Interactive Proof System for Mathematics and Software Development

We have developed an interactive proof editor, Lego, but based on an extension of Coquand and Huet's Calculus of Constructions. This is a type theory which gives us higher order intuitionist logic. The proof system supports type inference and has both dependent product and dependent sum type. The dependent sum types can be used to represent abstract mathematical structures such as groups and also program inferences. We have been exploring applications to mathematics, for example, theorems about lattices, the construction of real numbers and metric spaces; also about software development, using both categorical ideas and ideas drawn from research on data refinement and algebraic specification.

Logical Frameworks

The aim of the Logical Frameworks project is to study theories of formal reasoning, addressing such questions as: What is a logical system? What is a presentation of a logical system? These questions (and their answers) are motivated, in particular, by the requirements of mechanical implementation of formal reasoning systems.

One approach to these issues is to consider a type-theoretic metalanguage, which can be implemented on a machine, in which logical systems

can be represented by utilizing encoding paradigms based upon the Curry-Howard identification of formulae and types. This is the approach taken by the Edinburgh Logical Framework – ELF. An alternative approach is to implement a particular, powerful logic. This approach can also exploit type theory, an example being Huet and Coquand's Calculus of Constructions. Recent work in these areas concerns the exploitation of Generalized Type Systems as metalanguages for Logical Frameworks. It remains a question of active research to determine which are the better type theories to use as logical frameworks. An alternative approach is provided by studying algebraic (categorical) presentations of type theory. In a similar vein it is hoped that functional calculi, in the manner of Kelly and Hagino, might provide a language for the definition of logical systems in which the basic structural properties of logical systems, in particular for Sequent calculi, can be studied.

Semantics

Research in this area comprises work on the mathematical foundations of the semantics of computationally-oriented languages and the investigation of particular areas which are not the focus of any large research project. One needs to understand the operational and denotational semantics of languages, and logics to reason with them. The aims are, for example, to pursue language design (whether for hardware, programming or specification) and program and specification design and to prove systems meet requirements. The mathematical tools available are varied and now quite sophisticated and are taken from logic, lattice theory, topology, and algebra, and especially category theory which provides a rich source of unifying language.

Specifications and Formal Development

Specifications in ML

Extended ML is an algebraic specification language for specifying Standard ML programs. Specifications in Extended ML look just like programs in Standard ML except that axioms are allowed in module interface declarations (signatures) and in place of code in program modules (structures and functors). Some Extended ML specifications are executable, since Standard ML function definitions are just axioms of a certain special form. This makes Extended ML a "wide-spectrum" language which can be used to express every stage in the development of a Standard ML program from the initial high-level specification to the final program itself and including intermediate stages in which specification and program are intermingled.

The semantics of Extended ML are based on the primitive specification-building operations of the ASL kernel specification language. From this semantic basis Extended ML inherits complete independence from the logical system (institution) used to write specifications. This allows the use of different logical systems for writing specifications, including logics with special facilities for specifying error conditions, partiality, etc. In principle it also allows any programming language, enriched with ML-style modules, to be adopted for writing code.

Formal Development of Programs

It is usually easiest to specify programs (in Extended ML or any other specification language) at a relatively abstract level. It is then possible to work gradually and systematically toward a low-level program which satisfies the specification. This will normally involve the introduction of auxiliary functions, particular data representations and so on. This approach to program development is related to the well-known programming discipline of stepwise refinement. The establishment of an appropriate formal notion of the refinement of one specification by another enables this programming methodology to be formalized. If all the refinement steps can be proved correct then the finished program is guaranteed to satisfy the original specification. Since specifications have a modular structure it is possible to develop different parts of the specification independently and assemble the resulting programs later.

Foundational and methodological work has continued into a framework for formal program development. Foundational issues include the formulation of a formal definition of the refinement relation which reflects our programming intuition, proving that refinements compose as required to ensure the correctness of stepwise and modular program development, and generalizing all aspects of the formal program development framework to achieve independence from the logical system used to write specifications. Methodological issues include the use of modular decomposition during the refinement process and the role of interface specifications in the program development process.