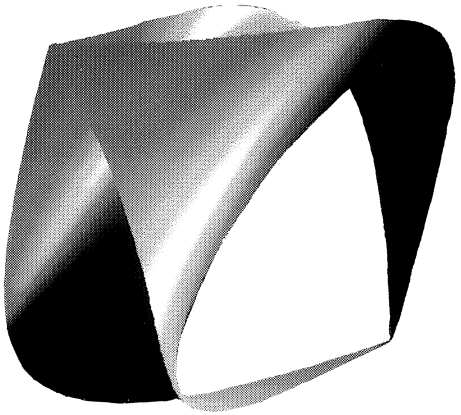
An aerial view of the Kings Buildings site.
Computer Science is housed in the James Clerk Maxwell building (to rear of centre of photograph, overlooking the golf course).



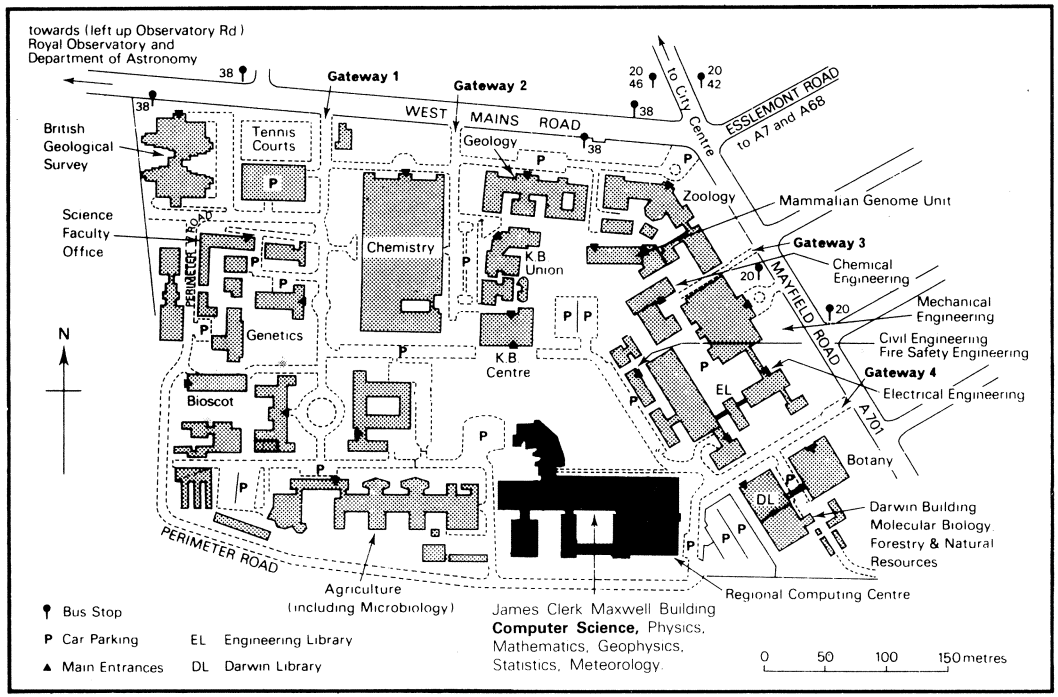
1989

University of Edinburgh

Department of Computer Science

DEPARTMENTAL HANDBOOK

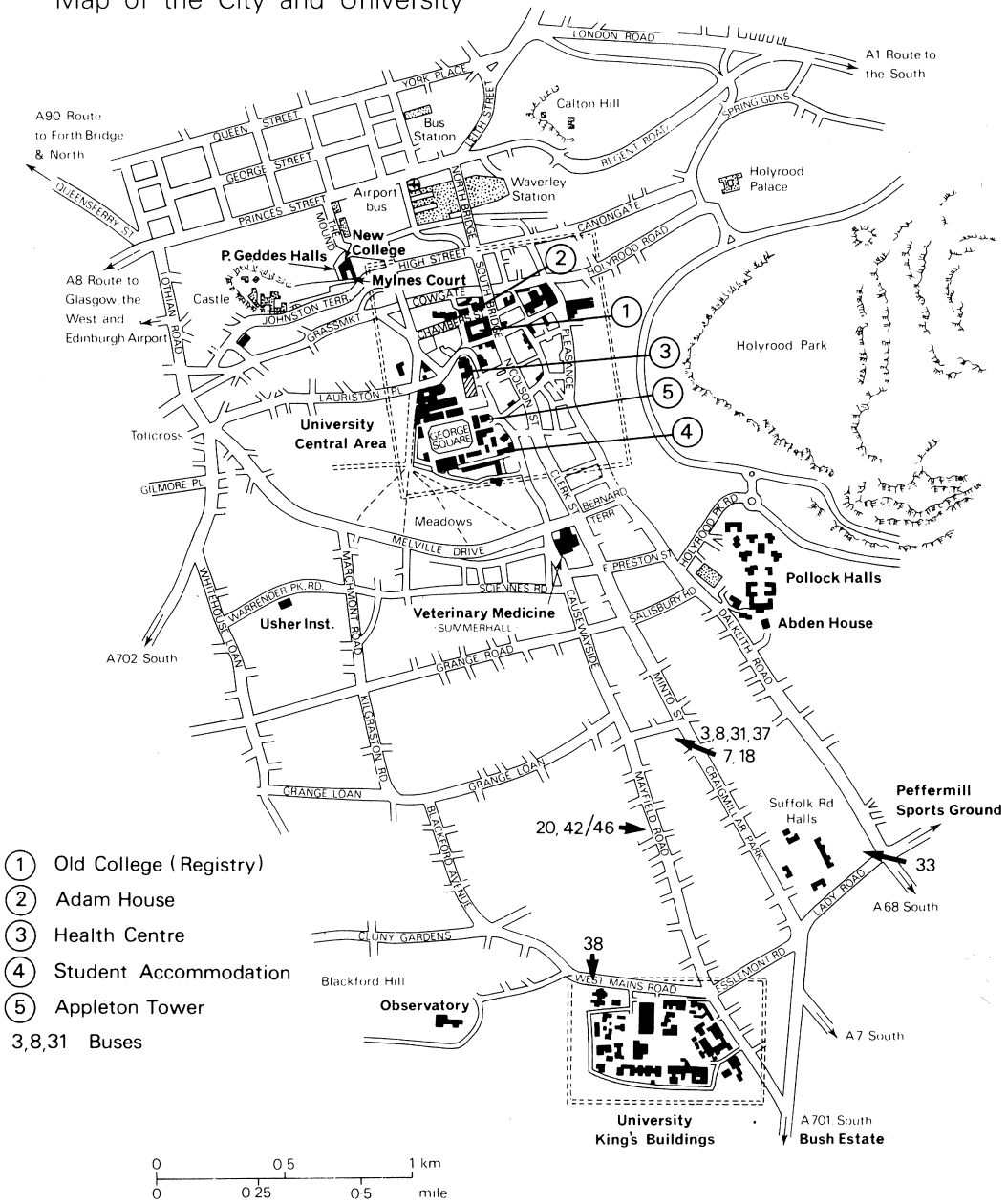
UNIVERSITY OF EDINBURGH
Map of Kings Buildings



See inside rear cover for map of the University within the city.

The front cover inset was produced by a Bezier surface editor from a plane surface by moving the corner control points on a coarse grid. This was then reduced to a finer grid for the plot shown. The image was produced on a departmental Advanced Personal Machine (APM) as part of a final year project.

UNIVERSITY OF EDINBURGH
Map of the City and University



See inside front cover for detailed map of the Kings Buildings site.

CCS, by adding operators from other languages, such as CSP and ACP. Finally, the existing user interface is being extended with a graphical interface.

Languages and Programming

Standard ML

ML has evolved over a period of about fourteen 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 prize for 'Standard ML'.

Currently, research is focussed on separate compilation in ML and on integrating concurrency and ML. The former involves the notion of a module being a program parameterised on functors it may use, but otherwise being completely closed and hence easy to compile by itself and port from one application to another. The latter involves interesting problems of type inference and the idea that exceptions can be used as channels of communication between concurrent ML processes.

An example of value and the status of ML as a general purpose language conducive to highly efficient software design has been its use as a rapid prototyping vehicle to implement a modelling package on the transputer in collaboration with a commercial company. Since there is little or no debugging assistance or memory protection on the transputer, the system was first prototyped in Standard ML, with concurrency being modelled abstractly (this took 6 weeks). The system was then rewritten in C (by hand), with concurrency being modelled concretely and the ML code being treated as a very detailed specification (this also took 6 weeks). A week was spent configuring the transputer networking code in preparation for porting of the resulting 20,000 lines of C code to the transputer. This port was immediate and no bugs have yet emerged in the final system.

Concurrency is the other main area under investigation. A prototype of a concurrent extension of ML, PFL, has already been constructed. However, problems with the semantics of such a language still remain. The aim is to develop a concurrent extension of ML on a shared memory system, before considering machines without shared memory, such as transputer arrays.

Categorical Programming

The use of Category theory in programming has been studied in a collaboration between the Edinburgh and Manchester Computer Science Departments which

considered how categorical constructions should be coded in Standard ML. This work has been published in a book by Rydeheard and Burstall, 'Computational Category Theory'.

Current work is concerned with the use of category theory to design a programming language. This language uses a somewhat generalised form of adjunction as its sole definition method. It uses this to define both data-types and the (primitive recursive) functions over them. From this minimal basis all the facilities of a conventional functional programming language can be defined, including lazy data-types. The computation rules are derived from the rules about the unit and co-unit in an adjunction, and it has been proved that the functions always terminate. The language has been implemented with an interpreter in LISP.

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.

Performance Modelling

The performance of computer systems and communication networks has been extensively studied for many years. Although considerable advances have been made suitable performance modelling tools are not generally available for systems designers and architects or analysts studying working systems. There is currently renewed interest in providing such tools because of three parallel developments. Firstly the emergence of practical distributed and parallel computer systems is generating new classes of problem, which are not solvable by existing analytical and numerical methods and are driving the search for new solution methods and more importantly focussing attention again on simulation programming. Secondly 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. Thirdly the formalisation and automation of the design process for software and computer systems, through Integrated Project Support Environment (IPSE) development is creating a new rôle for performance tools and exposing new problems of data sharing and verification of results.

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. They are designing

and building an Integrated Modelling Support Environment (IMSE) which includes novel tools to to support systematic modelling and experimentation. The design is intended to be readily extensible. Edinburgh is particularly involved in a tool to support experiments.

Proof Engineering

IPE: An Interactive Proof Editor

IPE is an interactive proof editor written at Edinburgh in the period between 1984 and 1987. It provides an easy to use interactive interface for untyped first order logic with equality and sufficient second order to do induction. This system continues to be maintained and has been used to teach two courses to a mixture of industry and academics. The course participants found the system attractive to use and easy to learn. Its logical power is limited, but it is sufficient to learn first order logic and to do proofs of correctness for functional programs.

In 1987 work started on a new version, IPE-2, which would use the insights of the logical framework to prove a much more general system, logic independent, reproducing the attractive interface based on attribute grammars. A pilot system was produced however work on this system in currently in abeyance but the plan is to combine it with work on LEGO.

LEGO: A Prover for the Calculus of Constructions and the Logical Framework

LEGO is an interactive, refinement proof editor for the Edinburgh Logical Framework, and the Calculus of Constructions, a very powerful higher order logic devised by Huet and Coquand. The implementation is based upon one in CAML by Huet, but has greatly extended the original system. The emphasis is on removing the more tedious aspects of interactive proofs; the system is able to infer type parameters which a naive implementation of constructions would demand from the user, and able to infer level numbers in the hierarchy of universes, $type_0, type_1, type_2, \ldots$. An experimental version of sigma types is provided to handle mathematical structures.

The editor provides an ML interface to the user but, unlike LCF (the earlier Edinburgh interactive proof system), retains the proof tree as a data structure. The user types in ML commands, but the input is mediated by a parser to simplify the syntax. Refinement is built into LEGO, and includes some decidable proof search. Tactics can directly access these features.

A number of interesting example theorems have been proved using the editor, for example the deduction theorem for a minimal propositional calculus, a theorem in elementary topology about bases, a metatheorem about normal

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.

Implementation of Concurrency

The extension of ML to include concurrency has been mentioned in the earlier section on ML above. This will take advantage of experience with Kevin Mitchell's prototype extension which has been in existence for some years. It will also interact closely with the work on mobile processes, or label passing, mentioned above.

The Concurrency Workbench

The Concurrency Workbench, a set of formally based tools for analysing and designing concurrent systems, has been built by a colaboration 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 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 a seminar for postgraduate students. This year will see its introduction in 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).

In the near future the equivalences checker will be extended to cover CCS with a restricted form of value passing; this will bring 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 will be implemented. It is also planned to extend the syntax of the process language, which is currently restricted to

Concurrency and Communication

Edinburgh's work on models for concurrent communicating programs and hardware arose in the 1970's, 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 year this foundational work has been broadened to include new elements, both theoretical and practical. These are itemised below. On the theoretical side, significant progress has been made with various logical approaches and the algebraic theory has been enriched.

Most notable on the practical side is the Concurrency Workbench (a joint project with the University of Sussex funded by the SERC). Currently, the Workbench is used as a teaching aid for courses in concurrency, including courses for Industry; in this context it is greeted with enthusiasm. It has been used on some small applications one of which was the locating and fixing of a bug (leading to deadlock) in the VAX VMS mail system. It is hoped to apply it on a large scale in a collaborative project on Safety Critical Systems.

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 will seek 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: whether observation equivalence can be finitely axiomatised for finite processes under a certain set of operations; has settled (negatively). The phenomenon of divergence (infinite internal computation) has been further examined. This work was funded by the Venture Research Unit of BP; it 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.

forms for an equational logic with associativity, and the binomial theorem for rings.

The Logical Framework

The aim of the LF project is to study theories of formal reasoning, especially motivated by the requirements of computer implementation. One approach to this problem has been to implement a particular powerful logic, another has been to implement a metalanguage within which various logics can be implemented and used. The latter approach is taken within the LF project, and a particular language, ELF – the Edinburgh Logical Framework – has been proposed. However both approaches have much in common, as often the languages are very similar type theories: ELF is one such, Huet and Coquand's Calculus of Constructions is another. Recent work concerns the definition and use of fairly strong extensions of the latter calculus (which in turn extends ELF). It remains a question of active research which is the best type theory to use for a logical framework.

Theoretical work on the Calculus of Constructions

Work in this area has provided some theoretical bases for the version of the Calculus of Constructions implemented in the LEGO editor. It has considered several variants of this calculus including ones with a hierarchy of types and with strong sigma types. It has proved results about strong normalisation and given semantics using the ω-set model technique proposed by Moggi. The Extended Calculus of Constructions has a simple notion of principal type and a simple algorithm for type-checking and computation.

Experiments with Hypercard

Some experiments with proof editors have been carried out using Hypercard on the Macintosh. A Hypercard demonstration has been constructed which approximately mimics the IPE behaviour for specific examples, and an editor has been developed for proof maintenance without doing any actual inference rule application. It simply believes the inference steps to be input by the user but aims to provide a convenient way of recording a proof and showing the dependencies on demand. Work has also started on a Hypercard front-end for a LEGO system.

Semantics

Research in this area comprises work on the mathematical foundations of the semantics of computationally-oriented languages and the investigation of partic-

ular 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 principal 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

board for Sun workstations (the 'Rekursiv') which promises to overcome this problem. The main thrust of the awareness initiative is a collaborative programme of development, enhancement and evaluation of object oriented systems, with particular emphasis on applications in AI, Computer Aided Software Engineering and supporting languages, involving a number of universities and other institutions of higher education.

Concurrent Architectures

The majority of research in this area is based on the notion of 'context flow' architectures, an implementation technique for parallel architectures in which the system is defined as a cyclic graph structure, through which parallel contexts flow. Active research topics include: a hardware description language for context flow architectures, VLSI design of context flow systems, and the implementation of declarative architectures using this technique. Tools for investigating these systems are under construction, and include a high-level multiprocessor simulation which runs on the Meiko Computing Surface, a compiler and low-level simulator for the hardware description language, and a prototype implementation of a single context flow processor.

Computer Graphics

Computer graphics research interests are concentrated in two main areas - graphics on parallel machines and graphics in a functional environment. Parallel computing is a major theme of this department and in this University. 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 done 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 3D 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.

work on a number of fronts.

Theoretical work is being carried out to construct abstract models of important classes of parallel computations and formal methods, involving the use of the specification language Z, are being used at all levels of detail.

Simulation is being used to study strategies for process placement and migration. Modelling tools have been built so that a series of experiments can be carried out, and the effect of varying hardware topology and program structure studied systematically. System measurements are being made to determine the statistical properties of real parallel programs. These statistics are to be used as input to the simulation models.

Run-time decisions about resource allocation will be based on real-time measurements of processor and hardware channel usage. This will involve special-purpose hardware which is currently being designed. Ideally, in making real-time or cumulative measurements, the information should be acquired without altering the results of the computation or seriously degrading system performance. The monitoring hardware should thus be an integral part of the system, designed in from the start. For POSIE an experimental testbed is being built, made up of MC68000-based single board computers attached to the Department's experimental Centrenet LAN, and including a special purpose monitor board.

Novel Architectures

As facilities for the generation and implementation of VLSI design improve and become more cost-effective, research into novel architectures (and concurrent architectures) will become increasingly important and will be an on-going activity at Edinburgh. This work is greatly facilitated by the use of host mechanisms which allow access to existing peripherals, filestores, etc. Thus a novel architecture can either be attached to an experimental network as a computing resource or can be incorporated into an existing workstation.

The Sparse Project is concerned with the design and implementation of hardware and software for a sparse vector processing system. This project is supported by SERC and High Level Hardware Ltd., one of whose Orion computers is used in the project. The hardware takes the form of an add-on board for the Orion and will provide specialised support for the processing of sparse vectors.The project also involves a considerable amount of work on compilers and on language extensions to give the programmer access to the sparse vector support mechanisms provided in hardware. The applicability of these mechanisms to other types of computing is also being investigated.

The Department is also involved in running an awareness initiative in object oriented computing funded by the Department of Trade and Industry. Progress in practical applications of object oriented computing is currently limited by hardware performance and Linn Smart Computing have produced an add-on

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.

An important issue concerns systematic methods for proving the correctness of refinement steps, since the definition of refinement used is model-theoretic rather than proof-theoretic. A method is being developed which involves a calculus for transforming specifications.

Stylistic Analysis

Stylometry has been defined as the scientific study of the usage of words in an attempt to resolve literary problems of authorship and chronology. The traditional methods of stylistic analysis have often been based upon subjective evaluation of internal textual evidence, often with unsatisfactory results. With the availability of computers as tools, new ideas have burgeoned and new approaches to these age-old problems have led to an increased need for statistical analysis of observational data. For example, it has now become practicable to study the usage of the most common words (such as determiners, conjunctions and prepositions) in a text: because of their number, such function-words have previously been neglected by literary scholars. Study of the positions of such words within the sentence has revealed that authors can be distinguished in this way. A new approach to problems of authorship and chronology is being brought into being by the study of such habits of composition.

Before any new technique is applied to disputed texts, it must be tested on texts of known provenance - as with other observational sciences. A wide range of texts is available for such testing, as well as an ever-growing collection of unanswered questions ranging from the composition of the Iliad to disputed wills in the U.S.A. Current work is concentrating on the thorny problems of authorship of Elizabethan and Jacobean drama and on the development of techniques for investigating poetry as well as prose.

This research into the methodology of answering such questions necessarily involves the development of further techniques and associated software. The amounts of data to be handled are often large and the processing poses interesting problems for the computer scientist.

Very Large Scale Integration

The main themes of activity are Computer Aided Design (CAD) tools and novel VLSI Architectures. The CAD framework encompasses both Silicon Compilation and the use of Formal Specification, Synthesis and Verification techniques.

Silicon Compilation

Silicon Compilation is the process of automatically synthesising a design right down to the mask level starting from a high-level description of its required behaviour. Considerable progress has already been made in this direction. An early silicon compiler, implemented in cooperation with the Electrical Engineering Department, was the so-called 'FIRST' system. This used a 'structural' description of operators and their connections as input and was aimed at bit-serial signal processing applications. More recently, a silicon compiler named 'U2', originally aimed at nucleonic instrumentation applications, has been developed in cooperation with the UK Atomic Energy Authority. Other silicon compilation research has concerned structures with a built-in-test capability. The lower levels of synthesis, concerned with physical block placement and routing are also under investigation. The focus of future work will lie in the problem of synthesizing designs from behavioural descriptions in the form of an algorithmic specifications rather than from a structural description. This is a much more difficult task but some progress has been made in the first stage, that of 'silicon pre-compilation'.

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.

programmed controllers and personal workstations. Within the Department there are now over sixty Advanced Personal Machines (APMs) which provide substantial local computing capability and use an Ethernet to gain access to network services such as filing and printing. A variety of new projects are now emerging which carry forward from this experience.

Distributed Systems

Current research is directed towards examining the general problems associated with designing a distributed system comprised of many heterogeneous computing elements inter-connected by a communication network. The motivation behind such systems is to provide an improved user environment; potential benefits include improved performance through parallelism, resource sharing, fault tolerance, and the ability to adapt to changing resources. A number of projects are in progress in this area, the largest being the POSIE project, described below.

One of the more popular paradigms to emerge in recent years has been the application of object-oriented programming techniques to distributed system design. A distributed resource scheduler is under development which utilises object-oriented properties to control the allocation of resources in a distributed system. This research is being conducted within the framework of an Object Reference Model (ORM) which has been developed at Edinburgh.

Work is also continuing on the distributed system of APMs in the Department. The APMs have been used as discless workstations sharing a common filestore for a number of years, but a number of deficiencies have been apparent for some time. Work is currently in progress to replace the existing filestore mechanisms with a more sophisticated system which will offer both increased performance and functionality. The new system will facilitate the use of a number of file access and transport protocols in parallel, allowing users full access to all their files from any computer system.

The POSIE Project

The POSIE project involves the design of an operating system to provide efficient run-time support for concurrent, process-based computations. The main interest centres on architectectures which support computations with large-grained parallelism, structured as a set of asynchronous processes communicating by message-passing. Efficient computation requires that a program be distributed in an optimum way so as to take advantage of the processing power of the parallel processors, without increasing unduly the communications traffic between processors. One of the aims of POSIE is to find strategies for load balancing via dynamic process migration in systems where communication may be the bounding factor on performance, e.g. in grids of Transputers. The project involves

Algebraic Complexity

Algebraic Complexity studies certain intrinsic requirements in the computation of algebraic functions, such as matrix multiplication, polynomial evaluation and interpolation. A machine independent model is used whose basic operations are addition, subtraction, multiplication and division (sometimes extended or even contracted). Most work assumes a single processor, 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.

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, micro-

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 design. This work is applied to the devolopment 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 is being developed in which the array elements are simple but configurable logic blocks. Not only is the element function configurable but also its communication with adjacent elements. Such is the density of fabrication technologies becoming available that in the near future it will be possible to fabricate the very large arrays of elements that are necessary to map realistically sized computations onto the array and thereby to make the potentially enormous gains in concurrency the arrays allow. It is expected that this area of research will become increasingly interesting and important.

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 Systems 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.

LABORATORY FOR FOUNDATIONS
OF COMPUTER SCIENCE

RESEARCH IN COMPUTER SCIENCE

selection of which must together form a 'core' of one of the above 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.

It is also possible to study for the M.Sc. degree on a part time basis over a period of up to three years.

Further information on the M.Sc. in Information Technology run by the Computer Science department may be obtained from Dr. N.P. Topham in the department.

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.

LFCS Staff

The Laboratory currently has about 45 members, some categories of these are detailed below.

Academic Staff with research interests close to the Laboratory's:

Professor Robin Milner	Director	
Professor Rod Burstall	Co-Director	
Professor Gordon Plotkin	Co-Director	
Mr Stuart Anderson		Dr Inge-Marie Bethke
Dr Kevin Mitchell		Dr Donald Sannella
Dr Colin Stirling		Dr David Walker

Support staff:

Dr George Cleland	Assistant Director
Mr Paul Anderson	Systems Development Manager
Ms Morna Findlay	Systems Programmer

Dr Cleland is the Assistant Director of the Laboratory and is responsible for management of the Laboratory's infrastructure and for developing its Programme of Industrial Interaction.

The Laboratory has been fortunate in being able to attract first-rate postdoctoral research fellows from all over the world:

Research Staff:

Mr David Berry	Mr Mads Dam	Ms Claire Jones
Dr Hirofumi Yokouchi	Dr Eugenio Moggi	Dr Christian-Emil Ore
Mr Robert Pollack	Dr Paul Taylor	Dr Nick Rothwell
Dr Mark Tarver	Dr Mads Tofte	Dr Anne Salvesen
Dr Berhard Steffen		

There are also some twenty research students (also from many countries) working on predominantly theoretical research topics.

LFCS Research Programme

The mainspring of research in LFCS (whose structure is described elsewhere in this Handbook) 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 (ML) in software development. ML is in widespread use

POSTGRADUATE STUDY

Facilities are offered in the Department of Computer Science for postgraduate study by research (towards the degrees of M.Phil. and Ph.D.) and for study towards the degree of M.Sc. in Information Technology: Computer Systems Engineering.

Degrees by Research

Students can undertake full-time, supervised research leading towards the degrees of Master of Philosophy (minimum study period 21 months; normally 2 years) and Doctor of Philosophy (minimum period 33 months; normally 3 years). For the first year of study, candidates are registered as 'Supervised Postgraduate Students' prior to transfer to the appropriate degree course. (The minimum study periods noted above include this year spent as a Supervised Postgraduate Student.) It is also possible to study for these degrees on a part-time basis, over a longer period of time.

During the first year, postgraduate students are expected to participate in an appropriate study programme. For those students registered for research in the computational theory area, a specific course is provided. Students intending to pursue research into computer systems or other non-theoretical topic may be directed to particular advanced courses and are expected to attend postgraduate study seminars. Before registration for a higher degree students are obliged to submit a research proposal which is subsequently examined. Research interests in the Department of Computer Science are outlined in pp 25-37.

M.Sc. in Information Technology: Computer Systems Engineering

The School of Information Technology in the University of Edinburgh offers a 12 month program of postgraduate study leading to the M.Sc. in Information Technology. Each of the constituent departments of the School (Computer Science, Artificial Intelligence and Electrical Engineering) offers its own degree, with a student based in one department able to take courses offered by the other departments. The Computer Science Department offers the M.Sc. in Computer Systems Engineering which takes the form of one of three themes: Parallel and Distributed Systems, VLSI Design or Applied Theory of Computation.

The course runs from October to September. At present there are around 50 modules available to students studying within the modular M.Sc. scheme ranging from Astrophysics, through Architectures, Databases, VLSI Design and Functional Programming to Knowledge Representation. During the first half of the academic year, eight chosen lecture modules are studied and examined, a

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 language

LFCS Computing Facility

The Laboratory has been fortunate in acquiring from a number of sources a powerful and flexible facility to support its systems-based activities. It is based upon a mixture of Sun, Vax and Pyramid equipment to provide the computational intensive resource. A number of multiprocessor workstations are also used for experimentation with parallel computing. This is augmented by a number of Macintosh systems and a small number of IBM PC machines used for general purpose work. The facility to date has cost about £1.5m

The central file server is a Pyramid 98XE system with 2 Gb of disk store. This runs Sun's NFS system to provide central filestore facilities for the Sun workstations: these consist of 4 fileservers and about 30 workstations. This is backed up by a small number of Vax/Unix systems which allows us to maintain Vax code compatibility.

There are also a number of Firefly Systems available. These are multiprocessor workstations with bitmapped screens. Each system has between 5 and 9 microvax-III processors and up to 64 Mb memory. They are supported by a distributed filestore running on the workstations themselves and on two central servers. The operating system (TOPAZ) is Unix compatible but has extensions

to support "Threads" which support concurrency. The equipment has been provided by the DEC Systems Research center in Palo Alto with whom we have had a longstanding collaboration.

Virtually all the Laboratory's software development is based upon window based interfaces. The use of MIT's X-windows system along with Sun's Network File System is allowing the Laboratory to build a range of applications and support environments which are portable, but still provide a good user interface.

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 would typically be be industrial researchers intent on transferring the current technology of the Laboratory back to their parent company. It is expected that they will both study the work of the Laboratory in breadth and also apply the theories and methods of the Laboratory to particular problems. Besides the benefit to their own work that this will bring it will also provide valuable feedback to the research of the Laboratory and deepen the Laboratory's understanding of the needs of Industry

Some **Collaborative Projects** are in place and other 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

POSTGRADUATE STUDY

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.

Some of the groups with which we have an active collaboration include Software Sciences (Specifications in ML), STL (ML Development), BP (Concurrency), DEC (The Pebble Language and Concurrent ML), AT&T Bell Laboratories (ML Compiler). A small project was carried out with a commercial software company Risk Decisions Ltd. An interaction with Meta Software Corporation, Cambridge (Mass.) is steadily growing over graphical tools for concurrency.

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 obtain from George Cleland, the Assistant Director, from whom further general information on the Laboratory may be obtained.

Information Systems 1

The motive in presenting this course is to increase the number of people in society who are well-informed about computers. The computer is becoming very much cheaper, a process which has been accelerated by the microprocessor. This removes one of the impediments to its increased application: it is already a significant tool in many aspects of our society. There are other impediments to its application which are not so easily overcome. Deciding what it should be used for, precisely what is required and how these facilities should be made available, are typical examples. Such decisions cannot become or remain the prerogative of the computer scientist alone. It is important that managers, politicians, lawyers, doctors, journalists etc., should be well equipped to make or influence such decisions.

It is quite impractical for most of the people who will find themselves in these positions to become experts in the construction of computer systems. It is, however, entirely sensible and desirable that as many as possible should be well informed, in general terms, about computer-based systems. It is this function which the course sets out to fulfil. Consequently, students from any faculty, in any year and with any background (other than having already attended a Computer Science course) will be welcomed on the course.

The course is organised as two half courses: the first half course is designed to introduce the fundamental technologies and component parts of information systems and the basic skills of computer-based problem solving.

Topics: *A brief history of IT, Operating Systems, Computer Architecture, Principles of programming, Programming in Prolog, Digital Communications, The IT industry: past, present and future.*

The principle theme of the second half course is information systems applications. Major applications are examined and discussed in detail.

Topics: *Computer graphics, Information retrieval: database theory and implementation, Hypertext, Artificial Intelligence: Knowledge-based systems, natural language and speech processing, Human aspects of IT: social implications and the management of change.*

The ratio of assessment is 20% for essays, 40% for practical work (divided between completion of set exercises and submission of written reports relating to those exercises) and 40% for the degree examination.

CS480 : Human-Computer Interaction

The interactive use of computing systems requires the cooperative efforts of both humans and machines. Success depends upon system designers understanding and applying knowledge of Human Factors and Computer Science in equal measure, so that systems are matched to the skills and knowledge of the human user. Human-Computer Interaction (HCI) combines the insights into human performance offered by Cognitve Psychology and Ergonomics with the formal methods, tools and technologies of Computer Science. HCI is now recognised as an aspect of design which is critical to the success of computing applications. The course aims to provide (a) an appreciation of the issues and problems in HCI, (b) knowledge of design methodologies, techniques and skills, and (c) awareness of current areas of research.

Topics include: *Psychology of HCI, Task analysis and specification, Formal specification methods, Design evaluation, Prototyping tools, Techniques, Technologies.*

CS490 : Computational Complexity

The course extends a line of study, begun in CS3 Analysis of Algorithms, which attempts to classify computational problems according to their difficulty. This idea is formalised using as a measure of problem complexity the amount of time (or space) required to solve the problem on a simple universal computing device, namely a Turing machine. A number of natural complexity classes, which are essentially independent of the Turing machine model, are introduced and characterised by identifying some of their complete (i.e., hardest) problems. Finally, the rôle of randomisation (occasionally allowing incorrect answers) in making apparently intractable problems easier is examined: this leads on to a discussion of non-uniform models, such as Boolean circuits.

Topics include: *The Turing machine model, Time and space as complexity measures, Complexity classes and hierarchies, Reductions between problems and completeness, The classes P, NP, PSPACE: complete problems, Oracles and the polynomial time hierarchy, Counting problems and the class #P, Provably intractable problems, Randomised algorithms and classes, Circuits and nonuniformity.*

DEPARTMENTAL COMPUTING FACILITIES

CS460 : Theory of Communicating Systems

An introduction to the Calculus of Communicating Systems, giving both the Theoretical Foundation and examples of practical application.

Topics include: *Discussion of models of communication; The language CCS and its operational semantics; algebraic laws; Flowgraphs and Flow algebra; Proofs of example systems, using strong bisimulation; Observational equivalence and congruence; A complete axiomatic system for congruence of finite behaviours. Semantics of an imperative language (like concurrent PASCAL) by translation into CCS; The notions of confluent and determinate behaviour.*

Two or three practical assignments will be set and marked. No computing facilities needed.

CS470 : Computer Graphics

The course aims to teach fundamental techniques for generating line-drawings and 3D images. Various rendering models are introduced. A major part of the course is devoted to curves and surfaces. There are practical exercises to illustrate the principles learnt in the course.

Topics include: *Algorithms for display, Line drawings, Curves and surfaces, Body modelling, Shading, Ray-tracing, Texture, Colour.*

CS440 : Denotational Semantics

Denotational Semantics is a powerful method for defining realistic programming languages and for specifying the meaning or actions of programs. It is also open to mathematical analysis and gives results in the theory of computation.

Topics include: *Abstract syntax, Sets and semantic domains, The denotational format, Definitions using stores, Dealing with recursion and iteration, Environments, Procedures, functions, and other compound data types, Continuations.*

Besides weekly pencil-and paper exercises, which are strongly recommended, the coursework consists of three assignments.

CS450 : Modelling and Simulation

The course studies the development and use of modelling and simulation techniques to study the performance of computer and communication systems. The central topic is discrete-event-oriented simulation but process-oriented simulation and analytic modelling are also discussed.

System designers, builders and managers are interested in performance evaluation. The topics introduced in this course are an introduction to both performance evaluation of computer systems and modelling techniques which are also more widely applicable. A simulation modelling package is used to develop models of a range of systems and is itself studied and developed.

Topics include: *Discrete event simulator, Building a model, Output analysis Multipricessor system model, Modelling local area networks, Process-oriented modelling, Queueing network modelling, Modelling packages.*

Modelling exercises will be carried out using an existing package.

DEPARTMENTAL COMPUTING FACILITIES

The computing resources of the Department of Computer Science include both mainframes and powerful personal workstations, the latter sharing common filestores *via* Ethernet local area communication networks. At present, two types of single user machine are used – the SUN workstation and the Advanced Personal Machine (APM).

The SUN workstation is a proprietary machine with a high resolution screen running UNIX BSD4.2. About 80 of these are in use in the department, 30 within LFCS, 8 associated with the VLSI group, 36 for 1st year undergraduate teaching and a further 8 for teaching later years.

The Advanced Personal Machine was designed and built within the department; the first workstation coming into use in 1981. These are modular machines currently based on a Motorola 68000 or 68010 main processor and have between 2 and 7.5 Megabytes of main store. At present, sixty-four of these are operated in personal offices and public terminal areas. The APM network has seven filestores with over a Gigabyte of total storage; twenty-five of the systems have 1024 × 1024 × 8 plane colour graphic framestores. The APMs are each about as powerful as a Mac SE and are used for second, third and fourth year undergraduate course work in subjects such as Real Time Systems, Graphics, VLSI design and Operating Systems and for final year and M.Sc. projects.

The departmental mainframes are a VAX 11/780 and MicroVax cluster running VMS supporting up to fifty simultaneous users; its main uses being fourth year teaching, teaching support, text-processing, electronic mail and general administration. The VAX mainframes are connected to both the departmental and Edinburgh University Computing Service (EUCS) networks. The departmental mainframe systems have a total of over 5 Gigabytes of disc store.

Under a collaborative agreement with the Gould Corporation, the Edinburgh University School of Information Technology (comprising the Departments of Computer Science, Artificial Intelligence and Electrical Engineering) were given a Gould Powernode 9080, running UNIX, to be used primarily for teaching. This supports up to sixty-four directly connected terminals plus X25 and Ethernet connections. The department also posesses a GEC 63 which was rescued from scrap when part of the IT School collaboration came to its natural end. This is unmaintained and is run mainly by the students for trial projects. It runs Unix system V and has both ISO and Unix communications.

The Department operates a printed circuit board CAD/CAM facility based

on P-CAD software running on Hewlett-Packard Vectra PC clones. This facility produces manufacturing instructions for a BEPI solder-wrap machine for the automatic production of printed circuit boards.

Other machines in use in the department include an Orion supermini computer forming part of a SERC/industry supported research project, two further microVAXen which form parts of industrially supported research projects, a number of Apple Macintoshes which are mainly used in first year teaching, teaching support and administration and a small number of other personal computers. A Real Time Systems laboratory is equipped with several robot arms and a variety of mechanisms and microprocessor prototyping kits used in automation experiments. All departmental computing facilities provide access to a variety of special purpose peripherals including high quality laser document printers and plotters as well as line printers.

All machines (Macintoshes and PCs included) are attached to one or other segment of the Department's Ethernet. The Macs can access the Ether through Kinetics Fastpath Appletalk/Ethernet gateways. The PCs have direct PC-NFS Ethernet connections. All can fileserve to some extent from the Unix mainframes.

The Department of Computer Science and Edinburgh University Computer Services (EUCS) were jointly involved in the setting up of a microcomputer laboratory in Appleton Tower at the central university site. This comprises mostly Apple Macintosh and PC clones along with a selection of terminals, linked *via* Ednet, accessing the EUCS machines (principally a NAS mainframe running EMAS (Edinburgh Multi Access System), a Gould NP/1 running Unix and a Vax 8600 running VMS).

CS430 : Compiler Techniques

The course examines the creation of executable computer programs on typical hardware/operating system combinations. An understanding of the context, both hardware and software, within which programs execute is seen as central to the construction of compilers. The approach followed is to consider the solution of problems associated with representative languages on representative hardware. The course is practically based, and should give valuable insights into the nature of programs, as well an an understanding of compilers.

Topics include: *Language issues, Hardware architectures, Operating systems, Parsing techniques and grammars, Code generation, Portability, Global optimisation, Run-time systems, Compile-time support systems, sharing support amongst compilers, integration.*

CS431 : Distributed Systems

This course takes the knowledge of operating systems acquired from the third year module as its starting point and examines the effects of linking many processors with local area communications. The overriding criterion for evaluating such systems is the extent to which location is hiddenn rather than explicit in tasks. The emphasis is on systems consisting of general work stations and specialised service machines. In this context a number of well known problems such as the notion of time, mutual exclusion and interprocess communication are reconsidered. The client server model dominates the remainder of the course. With reference to particular implementations such as Amoeba, Mach, Sprite etc...., various services are considered, among which are naming, authentication, file and mail. Database consistency issues and transactions are briefly considered.

CS411 : VLSI Practical Design

The motivation for this course is to provide the opportunity to undertake a substantial VLSI design project and in doing so to capitalise on previous introductory courses in CS and EE. The aim is to design a single-chip microprocessor right from the design of its order-code, through the design of its internal architecture down to the detailed layout at mask level, suitable for fabrication. Regrettably, actual fabrication will probably not be available.

In principle, freedom is allowed in the design of the order-code and architecture. However, time constraints demand that it be kept as simple as possible in order to make a complete implementation feasible. A possible starting point is the "DJR Mk I", having just 8 instructions, which was implemented in nMOS some years ago. This design project will be undertaken in CMOS. The intention is to use Sun colour workstations for the practical work. In previous years, the in-house ILAP CAD tools have been used.

The course is normally undertaken in the second term but it can be brought forward to the first term by special arrangement if a complete team and supervisor can be brought together (groups of 4 or 5 students work together withh one supervisor to design their own microprocessor). This has been preferred by joint CS/EE students in the past.

Each member of a team is allocated a part of the total design to work on, for example, register design, ALU, shifter, input-output circuitry, control circuitry, system integration etc.. In most cases, these parts can be based on or derived from existing designs, in order to avoid 'reinventing the wheel' and to save time.

CS420 : Parallel Architectures

In the last ten years there have been rapid developments in the field of concurrent architecture, resulting in a plethora of concurrent systems. The primary motivation for using concurrent systems is high performance, but unfortunately few architectures approach linear speedup and none actually achieve it. The course examines a wide range of topics in concurrency including the expression of concurrency and how this relates to architecture, the principles of concurrent architecture and the structures which have evolved to meet the demands of computationally intensive applications. Examples of existing and proposed systems are presented throughout the course in order to illustrate the theory.

Topics include: *Parallelism in high-level languages, Principles of parallel architectures, Summary of SIMD systems, Message-passing MIMD architecture, Shared-store MIMD architecture, Non von Neuman architecture, Future Developments.*

MEMBERS OF STAFF

Computer Science 4

The final Honours course deals with advanced topics in Computer Science. The course provides the student with an opportunity to add to the core curriculum of the first 3 years and to study subjects from earlier years in greater depth.

Throughout the whole of the fourth year, students undertake an original project – either research or implementation – under the direction of a member of staff. Approximately 50% of the time is expected to be spent working on the project. At the end of the final year, a project report is presented as a typed and bound document to count towards the final degree awarded. Joint Honours students may also have to submit a dissertation based on previously published work.

A degree of choice is available to final year Computer Science students since, although there are eleven possible modules, the students must take six. The choice can be rather more limited for joint Honours students since modules must be taken from both streams (see Appendix for details).

CS410 : Computer Aided Design for Electronic Systems

The course follows on from the work done in CS3 on systems design and on VLSI. It is restricted to CAD for electronic systems in order to allow some depth of knowledge to be gained in a particular area of CAD rather than a superficial view of a wider range of CAD applications. A major area not covered is that of mechanical engineering.

With increasing levels of chip integration electronic systems are becoming ever more complex and difficult to design accurately. CAD aims to keep the complexity under control and to automate as many design tasks as possible. The motivation is primarily an economic one. Companies must design systems as quickly and as cheaply as possible in order to bring their products to market as soon as possible. CAD tools allow them to do that. Human designers will increasingly be restricted to making higher level architectural decisions. The true 'silicon compiler' which takes in behavioural descriptions and generates mask-level layouts of acceptable quality is only just around the corner.

Topics include: *Design styles, Mask-level layout, Sticks-level design, Logic and gate-level simulation methods, Logic minimisation, Composition systems and languages, Algorithms for wire routing, Behavioural Silicon Compilation, Algorithms for automatic Printed Circuit Board layout, Hardware description languages and their implementation.*

Practical work will involve the use of both commercial and in-house tools on Sun workstations.

CS302 : Systems Design Project

The aim of the Systems Design Project is to introduce students to the methods of coping with the problems that arise with the design and implementation of large scale computer systems. It does not preclude any other coursework that may be set in association with the lectures, though this will not include any large exercises. It is intended that students will gain experience in deciding how to:

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

For the Systems Design Project students work in groups of about eight on the design and implementation of the hardare and software of a small digital system. The majority of practical work will be carried out in the Summer term, but the groups will be formed at the start of the Spring term. Each group will establish its own management arrangements and assign responsibilities within the group for the various parts of the project. During the Spring term the groups will meet regularly in workshop sessions to discuss their design, and will investigate available systems and components in order to arrive at a specification of the system to be built. Each group will be able to "employ" a member of staff in the department as a consultant, but the amount of consultant time used must be taken into account when assessing the (hypothetical) cost of the project.

The project will be written up and presented as a typed and bound document. An audio/visual presentation will also be made and its quality will form part of the assessment of the project. Part of the assessment will be peer group assessment by members of the group as to how well each of them has performed. Each group will decide how this assessment should be made.

At present the intention is that the digital system will be a VDU terminal, and each group will decide on its own specification. Further information will be made available at the start of the project.

MEMBERS OF STAFF

Professor R.M. Burstall : Design of Pebble - a functional language with strong type checking and dependant types. Development of a proof editor using incremental evaluation for attribute grammars to give an attractive user interface. Development of the 'institutions' concept to give an abstract categorical basis for the design of specification languages.

Professor M.P. Fourman : VLSI Specification, Verification and Synthesis: Formal methods 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. Categorical and axiomatic semantics.

Professor R.N. Ibbett : *Head of Department.*
Computer and Network Architectures: the design and implementation of an integrated computing resource network based on a high-performance local area network and incorporating a variety of parallel and special purpose high-performance computers.

Professor S. Michaelson : VLSI Design Methodology, especially Silicon Compilation. Applications of computers in the Humanities e.g. Computational Stylometry, the use of formal grammars in processing humanistic data, the use of computers to improve the public face of Museums and Galleries.

Professor A.J.R.G. Milner : *Director of Laboratory for Foundations of Computer Science.*
Research interests in mathematical models of computation particularly those supporting proof methodologies using machine assistance; how to present logics to machines; models and proof theory for concurrent computation; operational semantics of high level programming constructs.

Professor G.D. Plotkin : Applications of logic in Computer Science, especially denotational and operational semantics of programming languages. Sematics of type systems, various monadic theories of computation, general proof theory and the semantics of natural language particularly type-free intensional logics for situation theory.

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.

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.

Lecturers

Mr S.O. Anderson : Use of programming logics in the specification and development of verified programs; in particular 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).

Dr I-M. Bethke : *Joint lecturer with Centre for Cognitive Science.* Type theory and theory of appllicative structures, especially lambda calculus and combinatory logic; intensional logic; partial semantics; constructive mathematics and intuitional logic; applications of logic in Linguistics.

Dr G.J. Brebner : Protocol specification and verification. Distributed computing environments. 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 A.R. Deas : *Part-time.* Systems Compilation for VLSI.

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

Dr K. Kalorkoti : Computational Complexity with special interest in algebraic complexity. This includes matrix problems and bounds on formula size of algebraic functions. Decision problems in group theory.

Mr R.A. McKenzie : Graphics algorithms and 3D, interactive modelling on parallel machines. Relation of graphics to functional programming languages. VLSI design and simulation.

CS391 : Computability and Formal Languages

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

Throughout the course, a number of concepts and techniques are introduced that have applications in many areas of Computer Science and will indeed occur – or already have occurred – in other courses. These include Turing machines, simulation methods, coding of programs, universal programs, diagonalisation arguments, problem reduction.

Topics include: *Turing machines; Primitive recursive, and partial recursive functions; The Church-Turing Thesis; Manipulation of computable functions; Decidability, partial decidability, and undecidability;* Formal Languages – *Closure Properties of Regular Sets; Pumping Lemma for context-free languages; Closure Properties of Context-free Languages.*

Practical work will consist of weekly pencil and paper exercises.

CS301 : The Operating System Major Practical

This practical gives the student substantial experience of the design and implementation of Operating System facilities. The department's (diskless) workstations are used, each of which has a 68010 processor, 6845 memory management-mechanism, at least 2Mb of memory and a VDU terminal. Each station also has an ethernet interface, but it is not used directly during this exercise. The programming language used is IMP.

In the first half of the practical, students are given a skeletal kernel. From this base they load and run three processes experimenting with scheduling policies. They then go on to implement a Unix-like pipe facility. The second half introduces the memory management mechanism. Students implement a dynamic memory allocation facility which involves allocating physical and virtual address spaces and maintaining the page entries used by the hardware.

A report is prepared on each half of the practical, the assessment is based mostly on these reports, although demonstrations of the performance levels claimed in the report may be required. Credit is given for clearly justified and specified designs as well as efficient well-described implementations in about equal measure.

CS380 : Database Systems

A database may be regarded as a model of some appliction reality. The course concentrates on this modelling process and examines the capabilities of different database systems to capture the relevant properties of the application reality. In addition, the software support required to control shared access to a database will be discussed.

Topics include: *Data Models, Data Semantics, Database Management Systems, Distributed Databases.*

Students will be required to design and construct a database.

CS390 : Analysis of Algorithms

The CS3 Analysis of Algorithms course is concerned with the application of (mostly elementary) mathematical techniques to the design and analysis of efficient algorithms. The range of techniques for algorithm design introduced in the second year Foundations course is augmented and developed. In addition, the theory of NP-completeness is used as a tool for indicating those problems which are inherently intractable.

Topics include: *Comparison problems, The classes P and NP, Algebraic problems, Graph algorithms, Introduction to Computational Geometry.*

Practical work will consist ofpencil and paper exercises and one small programming exercise.

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

Mr R.J. Pooley : Simulation methodologies, support environments for modelling and design, efficient operating system support of parallel computation, performance modelling, language design and implementation.

Dr R.N. Procter : Information Systems – design and applications, computer aided learning, Information Technology education, social and organisational impact of Information Technology, soft machines. Human-Computer Interaction.

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

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

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.

Mr P. Thanisch : Database Systems. Query evaluation methods and performance enhancement in distributed databases.

Mr B.C. Tompsett : Operating systems for distributed computing resources. Methods of teaching organisational issues in Software Engineering.

Dr N.P. Topham : Parallel Architectures: design, analysis and application thereof. Language driven architectures, novel architectures, context-flow architectures. The design of VLSI architectures for efficient parallel processing.

Dr D.J. Walker : Semantics and proof theory for concurrent computation. Application to verification of concurrent programs.

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

Demonstrators

Mr D.K. Arvind : Computer Vision: perception of 3D scenes. Machine learning, especially on parallel networks. Parallel architectures for symbolic processing. VLSI chip design aids (especially simulators) on multiprocessors.

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.

Computing Officers

Mr P. Anderson	LFCS Systems Development Manager
Mrs C. Anstruther	(Computing Support Officer) CS1 support, Advisory
Mr D.W. Baines	Gandalf contention switch support
Ms J.T. Blishen	CCS, Firefly support
Mr J.H. Butler	Service Manager
Mr K.J. Chisholm	Graphics, database support
Dr G.L. Cleland	Assistant Director of LFCS
Ms M. Findlay	LFCS support
Mr R.J. Green	ML, SUN support
Mrs L. Hamilton	(Computing Support Officer) CS1, Macintosh support
Mr A. Howitt	Hardware laboratory support
Mr D.D. Rogers	VLSI support
Dr G.D.M. Ross	APM support
Mr A.J. Scobie	VAX manager, SUN support
Ms R. Soutar	CS1, Compiler support
Mr R.W. Thonnes	BEPI support, APM operating system
Mr J.S. Turnbull	IS1, CS2 real time systems support
Dr D.A. Welch	Administrative Officer, Schools Liason

CS341 : Language Semantics and Implementation

The aim of the course is to present a unified view of programming language semantics and implementation, based upon the linked notions of structured operational semantics and abstract machines. Both declarative and imperative languages will be treated.

Topics include: *Language Semantics, Abstract Machines, Implementation.*

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

CS360 : Computer Communications

Until recently, computing and telecommunications were viewed as distinct technical entities. Now, with the advent of cheap computers and sophisticated computer networks, the two have merged to yield "information technology". The course examines the fundamental techniques used to implement the sharing of information between computers, and applies them to all levels of communication, from the transmission of bits along physical connections to the distribution of computations over many processors.

Topics include: *Information, Time, Space, Local area networking, Wide area networking, Case studies (including factory automation and electronic mail), Real world issues.*

There is a practical exercise to implement the internationally standard LAPB link control protocol, thus enabling communication with a proven implementation.

CS330 : Operating Systems

This course provides a grounding in the design and implementation of general purpose multi-tasking operating systems. The emphasis is on the concepts which lead to practical implementations. The course concentrates on co-resident systems with only a brief overview of distributed systems. Three significantly different systems; Unix, VMS and EMAS are used throughout the course to illustrate real implementations. Almost all students taking this course also do the operating systems major practical.

Topics include: *Process management, The OS Kernel, Resource Allocation, Memory Mangement, File Management, Distributed Systems.*

CS340 : Programming Methodology

Great emphasis is currently being laid on the "quality" of software, and this module aims to deal with techniques which may be used to ensure the producution of good quality software. There are two main areas of concern: programming in the small and programming in the large. Programming in the small is concerned with systematic program design methods, both formal and informal, and with ways of reasoning mathematically about program correctness. Programming in the large is concerned with the management of large pieces of software, where many individuals may each contribute a part of the whole, or where the complexity of the system is such that no one individual could possibly maintain a full grasp of the whole at any one time.

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.*

The coursework consists mostly of paper and pencil exercises, but there may be one small exercise on a Unix machine.

Secretarial Staff

Mrs H.L. Cartwright	Accounts Secretary (part-time)
Mrs C.B. Duncan	Secretary to Professor Michaelson (part-time)
Mrs J.R. Eaton	Departmental Secretary
Miss L.M. Edgar	Secretary to Professor Fourman and Dr Cleland
Mrs A.M. Fleming	Secretary to Head of Department (part-time)
Miss E.A. Kerse	Secretary to Professors Burstall & Plotkin
Miss K.A. McCall	Departmental Secretary
Mrs D.A. McKie	Secretary to Professor Milner

Technical Staff

Mr J.C. Dow	Laboratory Superintendant
Mr A.M. Duncan	General workshop
Mr M.L. Graham	APM Production
Mr D.C. Hamilton	(Chief technician) Workshop, communications
Mr G. Inkster	Junior Technician
Mr J. Johnstone	APM Production Leader
Mr P.J. Lindsay	Real Time and Special Systems
Mr I.S. Marr	APM production
Mr T.S. Wigham	General workshop

Research Workers

Mr D.M. Berry	Development of the Functional Language ML
Mr A.J. Cunningham	Sparse Project
Mr M.F. Dam	The Concurrency Workbench
Ms C.C.M. Jones	Proof Development for Formal Systems
Mr J. Lothian	Rekursiv: Object oriented programming
Mr Luo Q.Y.	Posie Project
Dr E. Moggi	Computer Assisted Reasoning
Dr C-E. Ore	Sabbatical Visitor
Mr R.A. Pollack	Computer Assisted Reasoning: Logics and Modularity
Dr N.J. Rothwell	ML Language: Modules, Concurrency and Interfacing
Dr B.U. Steffen	The Concurrency Workbench
Dr M. Tarver	Computer Assisted Reasoning
Dr P.M. Taylor	Specifications in ML
Dr M. Tofte	ML Language: Modules, Concurrency and Interfacing
Mr Zhao J.	Posie Project
Mr Zhu S.	A Formal Basis for VLSI Design

CS320 : Computer Design

This course covers the principles and practices of computer hardware design. By the end of it students will be familiar with: How more complex combinational and sequential logic circuits are designed from simpler gates and flip-flops; the various ways in which CPUs are designed, their relative advantages and disadvantages, including how the operation of the CPU is controlled by the instruction fetched, and the implementation of logical and arithmetic instructions; memory design; the design of I/O controller circuitry; the various ways in which CPU, memory and I/O are interconnected; the extent and variety of commonly available microprocessor families; how to design a basic microcomputer.

Topics include: *Logic Design, Processor Design, Memory Design, I/O Design, Buses, Microprocessors.*

Students attend a Hardware Systems Laboratory and, working in pairs, carry out five design/implementation exercises using appropriate ICs mounted on prototyping patchboards.

CS321 : Computer Architecture

Computer architects concern themselves with techniques which will improve the performance and/or reduce the cost of computer systems. Over the years changes in technology have played a major role in providing these improvements, but a whole variety of different machine architectures has been developed take advantage of these changes. This course examines a number of technological and architectural techniques and the interrelationships between them and shows how the architectures of modern supercomputers have evolved.

Topics include: *Instruction Formats, Storage Hierarchies, Pipelines, Instruction Buffering, Parallel Functional Units, Vector Processors.*

Working in groups students will produce a paper design for a high-performance supercomputer capable of running an operating system and compiled programs written in one or more high-level languages.

Computer Science 3

The third year Computer Science course provides a basic foundation for the design and implementation of computer systems. Software-oriented lecture modules concentrate on the overall design of software systems and their interaction with the underlying hardware and software; hardware courses reflect recent developments in methods of hardware implementation. Throughout the course, the theoretical foundations of Computer Science are examined.

The course consists of 10 lecture modules each of 18 lectures. Each of these modules is compulsory for single Honours Computer Science students; joint Honours students generally take half the Computer Science modules (see Appendix for details). In addition to the lecture modules there are two major practical modules which are also compulsory.

CS310 : VLSI Circuit Design

VLSI defines technologies for creating complex systems on a chip. The exploration of design possibilities, and the development of the computer aids which are essential for design implementation, make VLSI an exciting field in Computer Science. The course provides sufficient basic information about integrated devices, circuits, digital subsystems and system architecture to enable the student to span the range of abstractions from the underlying Physics to complete VLSI computer systems.

Topics include: *Structured Design, Architectures, CAD tools.*

Practical exercises will be set. These will consist of four design exercises with the fourth including simulation. An examination paper accounts for 75% of the course mark, and the practical exercises for the remaining 25%.

RESEARCH STUDENTS

Computer Science 2

The second year Computer Science course introduces the basic components of computer systems, and examines how these components constrain the design of such systems. It covers both the design of simple extensions to computer systems, and the design of small new systems. An important concern is the analysis of designs to determine their properties, and to enable evaluation against alternative designs.

There are three lectures per week throughout the year, a one-hour tutorial each week during the first and third terms, and a three-hour laboratory each week during the second term. Computer Science 1A is a pre-requisite for Computer Science 2 which, in turn, is a pre-requisite for Computer Science 3.

Syllabus for CS2A1h

The first half course examines some fundamental concepts which occur at all levels of abstraction in computer systems, and looks in detail at the processor macro-architecture of computer systems.

Topics include *Logic, Binary representations, Information theory, State machines, Editor as a state machine, Lexical analysis and tools, Regular languages and grammars, Text handling algorithms, Processor as a state machine, Instruction sets, Assembler programming, Processor structure, Computer arithmetic.*

Practical work consists of foundational exercises, the construction of a text editor in Pascal and an introduction to assembler programming.

Syllabus for CS2A2h

The second half course examines input and output in computer systems, the processing of low and high level programming languages, and the acquisition of programming style.

Topics include *Processor-I/O interface, Interrupts, Characteristics of I/O devices, Analogue I/O, Syntax analysis and tools, Context-free languages and grammars, Choice of data structures and algorithms, Verification and debugging, Performance measurement.*

Practical work consists of assembler programming of physical devices, and the construction of a non-trivial compiler in ML.

Computer Science 1B

This course is intended for science and engineering students who do not intend to pursue computer science beyond the end of first year. It may also be useful for students from other faculties wishing to learn about computing in more depth than provided by IS1.

Syllabus for CS1B1h

The aim of this half course is to give students a basic competence in programming and a general knowledge of modern computer systems. Pascal is chosen as the programming language, because it provides a good foundation for learning programming concepts and includes most features found in other procedural languages.

Topics include *Introduction, Pascal, Introduction to computer systems, Introduction to the UNIX operating system, Introduction to computer graphics.*

Syllabus for CS1B2h

The second half of CS1B starts by extending the students' knowledge of data structures.

Lisp is presented in enough detail to give students a working knowledge of the language and the course introduces students to the imporant area of programs which have some "reasoning" capability. This section is related to the increasingly important application area of Expert Systems. A brief historical perspective on a wide range of languages is presented to complete the programming part of the course.

A major example of application software is introduced and experience of its use is provided through practical exercises. A typical example is a system for algebraic reasoning, but this may vary from year to year.

Topics include *Data Structures, Comparison of Languages, Lisp, Expert Systems, Applications software.*

Practicals

Associated with CS1B1h there will be some short programming exercises, some coursework exercises for the computer systems lectures and three medium-sized practicals, at least one involving graphics.

At the end of the first 6 weeks there will be a laboratory based practical test. Students will be required to demonstrate mastery of basic skills in editing, compiling and debugging programs and using the operating system.

Associated with CS1B2h there will be one exercise in Lisp which will also build on the student's knowledge of data structures. For the application package there will be a major exercise involving the material covered.

RESEARCH STUDENTS

Student/Supervisor	Project
Simon Ambler *R.Burstall, E. Moggi, M.P. Fourman*	Linear Logics and Topos Theory
Bill Anderson *K. Kalorkoti, M.R. Jerrum*	Algorithms for Factoring Polynomials and in particular Factorisation in Quotient Rings
Jamie Andrews *D.T. Sannella*	Characterising Logic Programming Systems with Proof Systems
Jo Blishen *K. Mitchell, D.J. Walker*	Analysis of Concurrent Systems
Julian Bradfield *C.P. Stirling, G.D. Plotkin*	Algebraic Structure of Petri Nets for providing Compositional Proof Systems for Net Logics
Carolyn Brown *S.O. Anderson, D.J. Walker*	Linear Logic and Concurrency
Ken Chisholm *R.J. Pooley, R.A. McKenzie*	Implementation of an Object-Oriented Generic CAD Systems Tool using a Persistent Programming Methodology
Mads Dam *C.P. Stirling*	Relevance Logic and Compositional Reasoning about Concurrent Systems
Angus Duggan *R.J. Pooley, K. Mitchell*	Type Systems for Distributed and Systems Programming
John Elliott *R.A. McKenzie*	Digital Image Processing in Parallel with application to Medical Image Processing or Video Conferencing
Richard Eyre-Todd *R.J. Pooley, S. Michaelson*	Methods for the Efficient Support of Large Scale Software Development; The Parallelisation of Compilation; Tools and Techniques for Debugging Parallel Systems
Jordi Farres-Casals *D.T. Sannella, R. Burstall*	Algebraic Specification

Gao Bo *D.J. Rees, A.R. Deas*	Novel VLSI Architectures, particularly VLSI Array Architectures
Philippa Gardner *D.J. Walker, G.D. Plotkin*	Logics of Programs
Kees Goossens *S.O. Anderson, K. Mitchell*	Formal Semantics of Programming Lanuages; Denotational Semantics
Douglas Gurr *G.D. Plotkin*	Category Theory applied to Semantic Frameworks for Complexity
James Harland *D.T. Sannella, R. Milner*	Exploration of Intutionistic Interpretations of Logic Programming, particularly the System of Hereditary Harrop Formulae
Leslie Henderson *M.R. Jerrum, A.J. Sinclair*	Computational Complexity: The Complexity of Enumeration
Tim Hopkins *R.N. Ibbett*	Design and Evaluation of a Vector Processor Architecture which efficiently supports Vector Instructions operating on Compressed Sparse Matrix Structures
Hans Hüttel *C.P. Stirling, D.J. Walker*	Semantics of Non-Determinism and Concurrency
Kayhan Imre *R.N. Ibbett, R. Candlin*	A POSIE Performance Monitor
Andreas Knobel *G.D. Plotkin, E. Moggi*	Denotational Semantics in a Non-Classical Setting; particularly Intuitionistic Models and Topoi
Gary Law *R.N. Ibbett*	A Protocol Set based on the Triadic Network Model
Tim Lees *N.P. Topham*	Development of Architectures for the execution of Functional Languages using Graph Reduction
Luo Zhaohui *R. Burstall, P.M. Taylor*	Theory of Computation

Computer Science 1A

This course is intended for students who intend to pursue computer science as a single or joint honours course in subsequent years.

Syllabus for CS1A1h

The aim of the first half course is to give students a basic competence in designing, implementing and reasoning about programs. Through a selection of case studies and practical exercises the elements of sound program design are introduced. Pascal is the programming language used for implementations and through its use students will meet the basic concepts of a modern imperative programming language. The course emphasises a rigorous approach to programming. In particular care is taken to establish that designs have the required properties and that run-time resource requirements are considered during the design process.

Topics include *Introduction, Program Design, Imperative Programming, Reasoning about Programs.*

Syllabus for CS1A2h

CS1A2h is concerned with one of the fundamental aspects of good software engineering, the use of data abstraction. Simple examples are drawn from Pascal, which allows operations on sets or integers, for example, but hides from the programmer the way in which these operations are implemented.

The idea of data abstraction is developed further. Students are introduced to the use of data abstraction as a design technique. Data abstraction facilities in a programming language allow us to hide the complexity of implementation from the other modules of the program. ML is studied as an example of a modern applicative language which provides such facilities.

Programming examples will be drawn from sorting, searching, graph algorithms and set manipulation algorithms. These areas provide good examples of the use of data abstraction in algorithm design and the crucial role of efficient implementations of data types in the construction of good algorithms.

Material on the analysis of algorithms is developed further drawing on implementations of abstract data types as a rich source of case study material.

Topics include *Data Types, Functional Programming, Algorithms.*

Practicals

There will be a substantial amount of practical work associated with each of the half courses. Practical work will contribute 50% to the course assessment.

Arshad Mahmood *K. Mitchell, S.O. Anderson*	Algebraic Specifications particularly in relation to Concurrency
Sathiamoorthy Manoharan *N.P. Topham, R.N. Ibbett*	Parallel Computer Architectures
James McKinna *R. Burstall, G.D. Plotkin*	Application of Category Theory to Computer Science
Michael Mendler *R. Burstall, M.P. Fourman*	Logics and Models for Concurrent Systems; Specification and Verification of Hardware
Faron Moller *R. Milner*	Algebraic Characterisations of Processes (Extensional Theories for CCS)
Pawel Paczkoski *C.P. Stirling, D.J. Walker*	Compositionality in Logics for Concurrency
Randy Pollack *R. Burstall, I.A. Mason*	Computer-Assisted Proof Development, in particular Machine-Checked Mathematics using the Logical Framework and the Calculus of Constructions
Steve Proctor *G.J. Brebner, R.N. Ibbett*	A Distributed Resource Scheduler which utilises the properties of Object-Orientation to control the Allocation of Resources in a Distributed System
David Pym *G.D. Plotkin, K. Mitchell*	Proofs, Search and Computation in General Logic.
Fabio da Silva *K. Mitchell, D.T. Sannella*	Functional Programming Languages and Denotational Semantics
Tom Stiemerling *N.P. Topham, R.N. Ibbett*	A MIMD Multiprocessor with Interleaved Instruction-Streams
Sun Yong *G.D. Plotkin*	Formal Specification of Protocols
Chris Tofts *R. Milner*	Timing of Concurrent Processes and the properties of Deterministic Imperative Concurrent Languages with Shared Variables

Wang Li-Guo *M.P. Fourman, R. Burstall*	Hardware Synthesis and System Design
Tom Waring *S. Michaelson, A.R. Deas*	Automated Silicon Assembly
Amanda Welsh *R.J. Pooley, D.A. Welch*	Design of an Integrated Support Environment for Scientific Computing
Greg Wilson *R.J. Pooley, A.J. Sinclair*	Space Search and Standard Algorithms for Controlling Searches and Load Balancing
Yeung Ping *D.J. Rees, R.A. McKenzie*	Silicon Compilation and VLSI Design

UNDERGRADUATE COURSES

first and, in some cases, the second year.

The first year Computer Science course assumes no prior knowledge of computer systems and therefore forms a broad introduction to the subject. Many students have met with computers at school or through home whereas others have not. From the beginning of the course, students are encouraged to obtain practical experience using computers. In the first year, students have sole use of a network of 36 SUN workstations. In the second and subsequent years, students have the opportunity to work with a variety of other computer systems ranging from the departmental VAX or Gould multi access systems, through mini-computers to microprocessor systems. In early years, students write software for existing hardware configurations; later they may construct their own hardware from standard components and are able to design their own VLSI (Very Large Scale Integrated (circuit)) chips. Throughout the course, emphasis is laid on the concept that a computer system is a fusion of hardware and software design - each playing a fundamental role in the efficient running of the overall system. A fundamental understanding of Computer Science can be achieved from an abstract and theoretical treatment and this forms an important part of the Computer Science degree at Edinburgh.

Course Assessment

In the first three years, class examinations are usually held at the end of each term. Although these do not count directly in the final degree, some guide to the student's own progress is obtained. Good results in the first year class examinations however can earn exemption from the first year degree examination.

Written examinations ('degree' examinations) are held annually in June. Passes in the first three years (seven in all) are counted towards an Ordinary degree. For Honours, the final degree is assigned on the basis of the results of third and fourth year examinations. In all the degree examinations, course work done throughout the year is counted in the final assessment.

APPENDIX

B.Sc./B.Eng. HONOURS DEGREES IN COMPUTER SCIENCE

The Department of Computer Science at Edinburgh offers several Honours degrees, as well as providing undergraduate courses contributing to other degrees, outside of Computer Science. The Honours degrees offered are

- Computer Science (B.Sc./B.Eng.)
- Computer Science and Electronics (B.Eng.)
- Artificial Intelligence and Computer Science (B.Sc)
- Computer Science and Management Science (B.Sc)
- Computer Science and Mathematics (B.Sc.)
- Computer Science and Physics (B.Sc)
- Computer Science and Statistics (B.Sc.)

The B.Eng. degree was introduced to conform to an Engineering Council directive that all degrees leading to accreditation should be designated B.Eng. These degrees are intended for students wishing to become members of the relevant professional Engineering Institution. Computer Science degrees give exemption from the British Computer Society's membership examinations although certain of the joint degrees give only partial exemption. An Honours degree in Computer Science taken under the B.Eng. regulations includes elements of 'The Engineer in Society' – topics relevant to a practising engineer.

An Honours degree in Computer Science (single or joint Honours) lasts four years during the first two of which 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.

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

COMPUTER SCIENCE

First Year:

Computer Science 1A
2 other courses

Second Year:

Computer Science 2
2 other courses

Third Year:

Computer Science 3:-

CS310 VLSI Circuit Design
CS320 Computer Design
CS321 Computer Architecture
CS330 Operating Systems
CS340 Programming Methodology
CS341 Language Semantics & Implementation
CS360 Digital Communications
CS380 Database Systems
CS390 Analysis of Algorithms
CS391 Computability

2 Major Practicals

Fourth Year:

6 lecture modules, at least 5 from Computer Science 4:-

CS410 Computer-Aided Design
CS411 VSLI Circuit Design Practical
CS420 Parallel Architectures
CS430 Compiler Techniques
CS431 Distributed Systems
CS440 Denotational Semantics
CS450 Modelling and Simulation
CS460 Theory of Communicating Systems
CS470 Computer Graphics
CS480 Human-Computer Interaction
CS490 Computational Complexity

A Major Project

COMPUTER SCIENCE & ELECTRONICS

First Year:

Computer Science 1A
Mathematics 1
1 of:-

- Engineering 1
- Physics 1
- Electronics 1h + Computer Science 1Ah
- Electronics 1h + Physics 1h

Second Year:

Computer Science 2
Electrical Engineering 2
Mathematics 2

Third Year:

From Computer Science 3:-

- CS320 Computer Design
- CS321 Computer Architecture
- CS330 Operating Systems
- CS340 Programming Methodology
- CS360 Digital Communications

1 Major Practical

From Electronic and Electrical Engineering 3:-

- Analogue Communications
- Digital Circuits
- Microelectronic Devices
- Microelectronic Systems
- Systems Theory

Fourth Year:

6 lecture modules, at least 2 from each department

A Major Project in one discipline
A Dissertation in the other

HONOURS DEGREES IN COMPUTER SCIENCE

Technology and to ensure that all courses in Information Technology subjects reflected the important new developements taking place in the area.

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) —this now provides a computing service for the University. Computer Science at Edinburgh has an international reputation for the quality of its research. There are several very active groups working in a number of different fields — computational theory, new programming languages, integrated circuit design, graphics, communication systems and high performance computer systems for example.

The Department is situated in the James Clerk Maxwell Building (JCMB) at The Kings Buildings (the science campus at Edinburgh University), about 2 miles from the centre of the city. This building is shared with several other departments including Mathematics, Physics, Meteorology and Statistics and provides extensive accomodation including lecture theatres, machine halls, a library and several common rooms.

Departmental Computing Facilities

There is a wide range of computer systems in the Department of Computer Science. Facilities include ca. 90 SUN workstations, 64 APMs (Advanced Personal Machines), several fireflies (microvaxes with a degree of parallelism) and a VAX cluster, used as a mainframe by all staff and students. The APM was designed and built in the department and provided a service five years before any similar commercially available machine. It offers many facilities including colour graphics and, because of its modularity, can be tailor-made to suit a particular purpose. The Department of Computer Science also has a share in a Gould and two GEC mainframes with the other members of the School of Information Technology and has access through the EUCS to the University computing facilities.

ARTIFICIAL INTELLIGENCE & COMPUTER SCIENCE

First Year:

Artificial Intelligence 1 (recommended)
Computer Science 1A
Mathematics 1

Second Year:

Artificial Intelligence 2
Computer Science 2
1 other course

Third Year:

Artificial Intelligence 3:-

Knowledge Representation 1
Knowledge Representation 2
Natural Language Processing
Vision

From Computer Science 3:-

CS321 Computer Architecture
CS330 Operating Systems
CS340 Programming Methodology
CS380 Database Systems
CS390 Analysis of Algorithms
CS391 Computability

1 Major Practical

Fourth Year:

6 lecture modules, at least 2 from each department

Artificial Intelligence 4:-

Expert Systems
Mathematical Reasoning
Robotics

A Major Project in one discipline
A Dissertation in the other

COMPUTER SCIENCE & MANAGEMENT SCIENCE

First Year:

Computer Science 1
Business Studies 1
1 other course

Second Year:

Computer Science 2
Business Studies 2:-

Management Science A
1 other half course

1 other course

Third Year:

From Computer Science 3:-

7 lecture modules

or

6 lecture modules + 1 Major Practical

From Business Studies 2/3:-

Management Science B
1 other Business Studies 2/3h half course

OR

From Computer Science 3:-

8 lecture modules

2 Major Practicals

From Business Studies 2/3:-

Management Science B

Fourth Year:

6 lecture modules:-

2 from Computer Science 4
2 from Business Studies Honours
2 other modules

A Major Project relating management science and computer science
A dissertation in management science

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 enjoying 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 in 1966 along with the Edinburgh Regional Computing Centre (ERCC; now the Edinburgh Universityy Computing Service, EUCS) when the, then, Computer Unit was reviewed. It was created as a small department with six staff but has now grown to one of the largest departments in the Science faculty with six professors.

In 1966, teaching in Computer Science consisted of a postgraduate Diploma and a first year undergraduate course in the subject. This has been substantially increased over the years and the department now offers an Honours degree in Computer Science as well as joint Honours degrees with Artificial Intelligence, Electronics, Management Science, Mathematics, Physics, and Statistics. In 1980, a terminal first year course, Information Systems 1, was introduced aimed specifically at non-Computer Scientists. The department also offers a number of postgraduate courses ranging from a M.Sc. in Information Technology (run jointly with Electrical Engineering and Artificial Intelligence as the School of Information Technology) to study by research working towards the degrees of M.Phil. or Ph.D. The School of Information Technology was established to coordinate undergraduate and postgraduate teaching in Information

COMPUTER SCIENCE & MATHEMATICS

First Year:

Computer Science 1
Mathematics 1A
Applied Mathematics 1Ah
1 other half course

Second Year:

Computer Science 2
Mathematics 2A
1 other course

Third Year:

From Computer Science 3:-

CS321 Computer Architecture
CS330 Operating Systems
CS340 Programme Methodology
CS390 Analysis of Algorithms
CS391 Computability

1 Major Practical

From Mathematics 3A:-

Probability
Algebra
Mathematical Programming
Real Analysis
Set Theory

Fourth Year:

7 lecture modules including:-

2 in theoretical computer science
1 other computer science module

Mathematical Logic
2 other mathematics modules

A Major Project relating mathematics and computer science

COMPUTER SCIENCE & PHYSICS

First Year:

Computer Science 1
Physics 1A
Mathematics 1

Second Year:

Computer Science 2
Physics 2A
Mathematics 2

Third Year:

From Computer Science 3:-

CS310 VLSI Design
CS320 Computer Design
CS321 Computer Architecture
CS330 Operating Systems
1 other module

1 Major Practical

From Physics 3:-

Mathematical Methods 1
Quantum Physics 1
Statistical Physics 1
Electromagnetism or Electromagnetic Theory 1

and *normally* 1 of:-

Acoustics & Dynamics of Fluids
Nuclear Physics 1
Solid State Physics 1
Quantum Physics 2

Fourth Year:

6 lecture modules, 3 from each department

A Major Project relating computer science and physics

INTRODUCTION

COMPUTER SCIENCE & STATISTICS

First Year:

Computer Science 1A
Mathematics 1
1 other course

Second Year:

Computer Science 2
Mathematics 2
Statistics 2A

Third Year:

From Computer Science 3:-

Any 7 lecture modules

1 major Practical

Statistics 3

Fourth Year:

From Computer Science 4:-

2 lecture modules

Statistics 4:-

5 lecture modules

A Major Project relating statistics and computer science

Table of Contents

Introduction	1
Honours Degrees in Computer Science	3
Undergraduate Courses	5
Postgraduate Study	22
Research in Computer Science	24
Laboratory for Foundations of Computer Science (LFCS)	39
Departmental Computing Facilities	44
Members of Staff	46
Research Students	52
Appendix	