Report of The Seventh British Colloquium on Theoretical Computer Science

University of Liverpool, March 26-28 1991
(Sponsors: ICL Ltd and IBM UK)

As has become traditional, BCTCS was again held this year in Holy Week of Easter. Penny Lane, made famous by the lyrics of the Beatles, provided participants with their first approach to the comforts of Derby and Rathbone Halls. This University of Liverpool location, set about by parkland and greenery, proved to be a happy and relaxed choice for the conference. From here excursions to the famous cathedrals, Mersey ferry, Albert docks, galleries and other Liverpudlian delights (including some excellent oriental restaurants) were possible in conference breaks. At the conference dinner, John Tucker's speech of thanks to the hard working local committee of Paul Dunne (Chair), Pete Higgins, Nicos Malvris and Derek Yates echoed everyone's appreciation.

The colloquium was admirably supported by nearly forty technical presentations of wide diversity. In particular the invited talks of eight distinguished guests (Henk Barendregt, Alan Bundy, Martin Dyer, John Hughes, Cliff Jones, Colm Ó Dúnlaing, Carl Sturtivant and Brigitte Vallee) provided strength and balance to the programme. Abstracts of all the talks may be found below.

The annual general meeting of the colloquium confirmed that the 8th British Colloquium for Theoretical Computer Science will be held in the University of Newcastle in March 1992. Dr Iain Stewart will be leading local organisation and he may be contacted through the Department of Computer Science at Newcastle. The colloquium was generously sponsored this year by both ICL Ltd and IBM UK.

Alan Gibbons

Abstracts
Invited Talks

Feasible Full Formalisation

Henk Barendregt
Catholic University of Nijmegen

Formalising reasoning started with Aristotle. Frege completed the work by providing a complete system of logic. Russell and Whitehead provided the first formalised pieces of mathematics. However an intuitive proof becomes unreasonably long. Using formalisation in lambda-terms and types a feasible formalisation is possible.

Automatic Guidance of Program Synthesis Proofs

Alan Bundy
University of Edinburgh

The 'proofs as programs' paradigm can be used to synthesise programs from their specifications. Assume that *spec*(*inputs*, *output*) is a logical specification of the required relationship between the inputs and outputs of the program. The program can be extracted from a constructive proof of the formula: $\forall inputs.\ \exists output.\ spec(inputs, output)$. In order to synthesise recursive or iterative programs, inductive proofs are required. The problem of guiding the search for inductive synthesis proofs will be investigated. The more machine assistance that can be provided for this task, the less human skill will be required to use the technique and the more widely it will be adopted.

Heuristics, adapted from the work of Boyer and Moore, have been implemented as tactics, and used to guide an inductive proof checker, Oyster. These tactics have been partially specified in a meta-logic, and the plan formation program, Clam, has been used to reason with these specifications and form plans. These plans are then executed by running their

associated tactics and, hence, performing an Oyster proof. Searching in the planning space is considerably cheaper than searching directly in Oyster's search space — indeed, it is even cheaper than plan execution.

Volume and related Computational Problems

Martin Dyer
University of Leeds

Integration in many dimensions is known to be a hard problem. It is provably difficult to perform even approximately for the 'simplest' integration problem, volume computation. By contrast, Dyer, Frieze and Kannan (1989) gave a randomized algorithm for approximately computing the volume of a convex set. This, and subsequent improvements by Lovàsz-Simonovits and Karzanov-Khachyan will be described. We will also discuss some more recent work on applications and related problems, in particular the 'discrete volume' problem.

Naturality, Polymorphism and Compile-time Analysis

John Hughes
Glasgow University

Interference Resumed

C.B. Jones
University of Manchester

The rely/guarantee approach set out to extend operation decomposition methods for sequential programs to cover concurrent shared-variable systems. The essential step was to recognise that interference has to be specified in order to achieve a notion of compositionality. Ketil Stoelen's thesis has addressed the main shortcomings of my earlier work. This talk will describe what the author sees as the next steps towards a development method for concurrent programs. In particular a case will be made for basing assertions on Resumptions, predicates of two states will be used, the beginnings of a collection of temporal operators aimed at natural program correctness arguments will be indicated. A list of recognised problems will also be discussed.

Computational Problems in Geometry

Colm Ó Dúnlaing
Trinity College, Dublin

Some issues in algebraic complexity

Carl Sturtivant

Algorithms for Integer Factorization

Brigitte Vallee
Université de Caen

Factoring integers is a very old problem, but in the seventies, two new facts were discovered: the introduction of complexity theory in the factoring problem, and the discovery that factoring has applications in cryptology.
Factorization is considered to be easier than NP-complete problems and there is only historical evidence that it is an intrinsically hard problem. In the RSA cryptosystem, the difficulty of factorization is applied, and it is a negative application.
We first describe the RSA cryptosystem, and show the cryptographic applications of the difficulty of the factoring problem. Then we present the main methods used in integer factoring: random squares methods, and we explain the difficulties that one meets in proving upper bounds on the running time of these algorithms, because one has to use certain unproven heuristic assumptions.
We conclude by describing a new method due to Pollard, Adleman and Lenstra which is called the number sieve algorithm. This method seems to be faster than the previous ones.

Contributed Talks

Compositional Inheritance

Jim Armstrong & Jim Howse
Brighton Polytechnic

A key design decision in object-oriented programming is whether a given program component (class) should inherit the attributes of another component or include it in its structure. This is a source of much debate in the object oriented community. Most guidelines are heuristic and are really appeals to experience. However the matter can be clarified using formal models and methods —

1. By using algebraic specification techniques to analyse inheritance relations.
2. By understanding that inheritance in object-oriented languages is actually a form of composition, and not necessarily a means of modelling IS_A relations.
3. By understanding that inheritance relations like IS_A can be captured by hierarchies of components related by composition, using a technique called *compositional inheritance.*

The algebraic techniques allow the values of objects to be denoted by terms which can then be compared with terms denoting values of related objects. 2) involves understanding the notion of the *visibility* of a given attribute. 3) involves the simple use of procedure calls, overloading and export and import statements. These three points will be illustrated by a simple design example.
(Stein, 1987) develops a model of the relation between delegation hierarchies and inheritance hierarchies. This model can be extended to map inheritance hierarchies into composition-based hierarchies. The basis of the model has already been established in (Armstrong, 1990). It will be suggested that this formal model provides a more convenient basis for making design decisions than hard-won experience.
Armstrong, J.P. *Inheritance, concept, mechanism and research*, Inf. Tech. Research Inst., Tech. report, Brighton Poly. 1990
Stein, 1987 *Delegation is inheritance*, Proc. 1987 Conf. on Object-oriented Prog. Lang., Systems and Applications

Polymorphic Dataflow Type Inference for Term Graph Rewriting Systems

R. Banach
University of Manchester

Term graphs are objects that locally look like terms, but globally resemble general directed graphs. They have been a vehicle for the implementation of functional languages for many years, but their use as a model of computation in its own right is of independent interest. The question arises as to whether the notions of typing and of type inference have any place in the more general term graph world. This is a non-trivial matter since the typing and type inference familiar from the term world are based on structural induction, which breaks down in the term graph world. It turns out that a more local notion of typing, satisfying a weaker invariant under rewriting, can be developed for graphs. Structural induction is replaced by dataflow analysis of the TGRS. This in turn enables type inference and polymorphism, using unification in the conventional Hindley-Milner way, to be introduced.

Formal Mapping of Specifications to Parallel Architectures

Naima Brown and Dominique Mery
CRIN Université de Nancy and CNRS

Nowadays, new parallel architectures are being proposed to the computer science community. Among well-known examples, like the transputer based architectures, such the T-node, or the connection machine. The goal of these machines is to offer the programmer a more powerful tool than the classical one. Computations need to be concurrently executed, but

the main question is to define what a concurrent computation is, and how to master the development of implemented solutions. A local experiment in our laboratory has consisted of developing an OCCAM program on a T-node machine. This experiment has been completed after three months of full-time work. The main result is that the T-node machine has a very poor environment; that is why a methodology based environment is required. Hence, the UNITY framework has been chosen as a starting point for the development of parallel programs, and we have mainly analyzed the transformation of UNITY-like programs into the OCCAM programming language. The mapping is not only a simple translation from one language to another one, but it is based on a *soundness* criterion as a proof preservation. .PP The UNITY framework allows the development of concurrent programs from a temporal logic-based specification language. Although it is a very powerful and attractive theory, there are some problems that can not be directly solved in UNITY. The first problem is the absence of a clear, and formal *statement* of transformation (or refinement) techniques. The second is that no formal transformation (or refinement) *theory* is well-defined. Moreover, the mapping in UNITY is quickly and superficially sketched. Yet, the careful study of problem classes with respect to the refinement notion and a restatement of UNITY logic have led us to explore the mapping problem in an algebraic way. We have used algebraic techniques to modelize the mapping. Yet, we have enriched the UNITY programming language by algebraic data types. The goal is to use data in an *algebraic way*, and to improve the *invariant* part of UNITY using invariants of data type. .PP The UNITY specification framework is detailed and the proof system is described. Specifications are statements of temporal logic. Formal techniques will be introduced to ensure the correctness of specifications with respect to the explicit, or implicit, underlying program. The development framework uses the specification language and the proof system to state the soundness of transformation rules. The principle for deriving, an OCCAM program from a UNITY one, is to associate any UNITY variable to a specific OCCAM process that manages the variable. The transformation is deadlock-free and is expressed in an algebraic style. The properties as *invariants* become *comments* in the OCCAM framework. Our transformation system leads to an executable OCCAM program. The main result is the *soundness* proof of the mapping: any property of the initial specification (UNITY) is preserved by the transformation into the implementation specification (OCCAM).

Uniformity of Shared Memory

Andrew Chin
Oxford University

Consider a multiprocessing system where the network topology has been hidden from the programmer. The decision whether to implement hashing then depends on (1) the latency to the shared memory and (2) the locality of memory accesses in the algorithm. We study the complexity of hashing in the Block PRAM model, thereby relating this decision directly to algorithmic issues. In particular, we show that there are universal families of high-performance hash functions having optimal locality.

Monotonic Reasoning about Non-monotonic Functions — or, when finding fixed points you can do (almost) anything

Alan Dix
University of York

Static analysis, especially of functional programs, often results in semantic equations over finite domains. The most complicated part of their practical solution being the solution of recursive function equations, or equivalently finding fixed points of defining functionals.

find $f : A \to B$ **st** $f = F$, i.e. $f = fixF$

Algorithms (e.g. frontiers algorithm) usually ensure that all intermediate functions are monotonic, and so the proofs of correctness (but not necessarily the implementation!) are straightforward. It may, however, be advantageous to use algorithms (such as pending analysis) where intermediate functions are not monotonic. In this case, it will not usually be the case that (even where F is defined) that: $f \leq g \Rightarrow Ff \leq Fg$ Without this proofs become

almost impossible. Happily it turns out that the functionals F of interest satisfy a stronger property *pseudo-monotonicity* whereby the above holds when either of the functions f or g is monotonic, not necessarily both. Using this we can begin to reason about non-monotonic functions using 'monotonic' arguments. Not only do proofs of existing algorithms become possible but one realises that one has enormous flexibility in calculating fixed points. You can do almost anything.

Integers in Ideals of $Z[x]$, Groebner bases and a Problem of M. Newman

P. Fitzpatrick & G. Norton
University of Bristol

Let u, v be relatively prime polynomials in $E = Z[x]$. We consider the problem of constructively determining the least positive integer $\nu(u, v)$ representable in the form $fu + gv$ for some $f, g \in E$ together with suitable multipliers f, g. if the leading coefficients of u and v are relatively prime, we show that the problem can be solved using the extended polynomial remainder sequence algorithm of [P. Fitzpatrick and G. Norton 'Linear recurrence relations and an extended subresultant algorithm', LNCS, 388, 232-243 (1989)]. In the general case we present a solution based on a reduced Gröbner basis for the ideal $< u, v >$ of E generated by u, v. Our investigation also leads to a complete classification of reduced Gröbner bases of ideals in E.

Proof and program transformation in constructive theory

D. Galmiche
CRIN-INRIA Lorraine

For many years important works in computer science have been devoted to the study of program construction with (and in) logical theories or frameworks. We can mention important techniques based on theorem proving and program transformation. In this paper the aim is to study the notion of proof and program transformation in *programming with proofs* framework. Programming with proofs in logical frameworks consists in extracting programs from constructive proofs following a schema in three principal steps: specification, proof construction and finally program extraction. But we know that, in such a framework, some programs obtained from proofs are not always efficient or some programs can not be derived from proofs and consequently the relationship between programs and proofs has to be studied.
Knowing that a proof contains more information than a program, it seems interesting to study here the notion of program transformation through proof transformations. Here, we investigate the transformation of proofs and programs in *programming with proofs* frameworks through techniques of generalization by abstraction with a view to deriving better programs.

Keywords: logical frameworks, intuitionistic logic, programming with proofs, program transformation, generalization.

Embedding Balanced Binary Trees in the Mesh

Alan Gibbons & Mike Paterson
University of Warwick

We describe how to embed balanced binary trees in the mesh such that each mesh node is associated with two tree nodes and such that each tree edge maps to a disjoint directed path in the mesh (each mesh edge $\{u, v\}$ being considered as two directed edges (u, v) and (v, u)). Such an embedding improves (for example) extant running times for parallel algorithms employing the balanced binary tree on the mesh.

Parallel Algorithms for Logic Simulation

Paul E. Dunne, C.J.J. Gittings & P.H. Leng
University of Liverpool

A Visual Verification of MergeSort

Chris Holt
University of Newcastle-upon-Tyne

Visual languages have been used to describe dataflow graphs, and the interrelationships of objects in large software engineering projects. They can also be applied to the specification and verification of programs, with the effect on programming "style" being a greater preference for relations connected by retracts. This is illustrated using MergeSort as an example of a simple, functional algorithm.

Timed process algebra $\neq$ Time $\times$ process algebra

Alan Jeffrey
Oxford University

A process algebra is one way of modelling parallel computation. Until very recently, most process algebras had no notion of time (other than 'eventually' or 'forever') but recently a number of timed algebras have materialized, notably the timed variant of CSP, CCS, and ACP. These models, although developed independently, have many similarities, which I would like to discuss.
Untimed process algebras are now reasonably understood, and the concept of a time domain is familiar from field like temporal logic. However, the combination of the two has many unexpected side-effects, such as the possible presence of processes which can stop time, non-determinism becoming may-testable and the problem of instantaneous behaviour.
This talk will consist of a short survey of the filed, and a discussion of the interaction between time and process algebra.

Parallel Program Schemas

Stephen G. Matthews
University of Warwick

The talk presents a schema based programming model with an interleaved state transition semantics designed for discussing language independent features of concurrent programs. Combining ideas from both flowcharts and temporal logic is not new, however, our 'intensional' notion of 'interpretation' for schemas is. We will show how Parallel Program Schemas are equally at home in modelling monitors as they are with modelling demand-driven data flow.

Infinite Synchronous Concurrent Algorithms — A Case Study — The Stack

Brian McConnell
University College of Swansea

We will introduce the theory of infinite synchronous concurrent algorithms, a model of parallel deterministic computation with infinite parallelism. The formulation and applications of the theory are introduced. We will then present a case study of an infinite synchronous concurrent algorithm. We study an idealised hardware stack that processes infinite streams of data and commands and show how this is correct with respect to an (higher order) algebraic specification. This work is joint with D. Gibby and J.V. Tucker.

The Algebra of Database Transactions: Monoids, Quantales, Faithful Actions

N.D.N. Measor
University of Leicester

A database can be specified in terms of a set of transactions from which it may be generated. We present the transaction set abstractly as a monoid given by generators and relations. The effects of the transactions can then be defined as an action of the monoid on the set

of states of the database, or, equivalently, as a functor from the monoid to the category of sets. The image of the monoid under this functor can be though of as a model for the abstract monoid of transactions; the adequacy of the presentation of the monoid rests on whether the monoid action is faithful, this being a kind of completeness question.
We can give the set of transactions a quantale structure by introducing joins into the monoid. The action of the transaction set is then represented by a mapping from the abstract quantale into the set of endomorphisms on a complete join semi-lattice. Each member of this semi-lattice is a set of possible states of the database, and the morphisms may therefore be viewed as transition relations from one set of possible states to another. Thus the approach lends itself to extensions based upon non-deterministic transactions and allows us to model uncertainty in a database.

Proof based Developments of Concurrent Programs

Dominique Mery
CNRS-CRIN-URA-262
54506 Vandoeuvre-les-Nancy, France

We study the development of concurrent programs from specifications according to the formal correctness of programs with respect to specification. A formal system manipulates development formulae that express a relationship between programs, specifications and proofs. An underlying logic is chosen to prove the correctness of a given program with respect to its specification: this system is based on UNITY logic and temporal proof systems and it takes a contextual information into account. The development rules have been built by generalizing manipulations on a specific case study. Our programming language consists of constructions such as sequential, parallel and non-deterministic computation but uses these concepts at different levels. The soundness of our rules is proved according to the derivability of the specification from the operational specification of the concurrent program. An operational specification is a set of transition formulae interpreting the program in an abstract way. A case study is developed in our system. The relationship with other frameworks is sketched and especially the predicate transformer semantics.

Working with Formal Languages in the Study of the Computational Properties of Graph-theoretic Document Models

G. Staniford & Paul E.S. Dunne
University of Liverpool

Document models may be considered from several standpoints; the static representation, of document structure and content, is one such standpoint and the dynamic representation in which we model the changes that occur to a document during the course of its production is another. A general document representation (GDR) is presented leading to the definition of a universe (Univ) that describes the totality of GDRs. Some GDRs in Univ are not computable; the notion of a feasible GDR is introduced and four specific examples are defined and discussed, paying particular attention to their underlying graphical structure. Moving from the static to the dynamic we define graph modification systems using the graph grammar approach and introduce recent work that has been proposed for use in developing practical systems. We believe that some restrictions, on the nature of document structure and hence on the freedom of the generative graph grammar rules, are essential if systems are to be implemented using current artificial intelligence techniques that are consistent and maintainable. Such restrictions are outlined and this leads to a discussion of future directions for the work currently in hand; conclusions are drawn about the achievements to date and a summary of the main points is presented to conclude the talk.

An introduction to post-Newtonian and non-Turing computation

Mike Stannett
Dept of Computer Science, Sheffield University

In our paper [1], we described a machine model with arguably "super- Turing" computational power. In this talk, we develop this theme further. We will survey four styles of computational model, and discuss the potential for each style to give rise to non-Turing behaviour. The classes of model we consider are

1. Newtonian models – Turing machines, analog models, CCS
2. Post-Newtonian models – Quantum computing, relativistic considerations
3. Restrictional models – models derived by restricting logical expressiveness
4. Algebraic Field Models – what properties should "computable numbers" possess in any model of computation?

We will also consider the relationships between the various styles of model.
This work is partially supported by the British Technology Group. [1] Stannett M. 1990 X-machines and the halting problem: building a super-Turing machine. Formal Aspects of Computing vol. 2 pp 331-341.

The Categorical Unification Algorithm Revisited

John G. Stell
University of Keele

Substitutions may be represented as morphisms in a category, where the objects are finite sets of variable symbols. The categorical unification algorithm provides a computation of coequalizers in this category. We show that this is not sufficient to justify the algorithm as a computation of most general unifiers. A clarification of the relationship between most general unifiers and coequalizers is thus called for, and we provide this. It turns out that the algorithm, as implemented, is correct; only its justification requires attention. We go on to show how the relationship between coequalizers and most general unifiers can be generalized to the case of order-sorted unification, and to consider to what extent this allows an order sorted generalization of the categorical unification algorithm.

On the Capture of Complexity Classes using Logic

Iain A. Stewart
University of Newcastle-upon-Tyne

We partially solve a conjecture of Gurevich and show that if $NP \cap co-NP$ is captured by a logic that is closed under disjunction then $NP = co-NP$. We also obtain similar results for other complexity classes.

A Method for the Development of Totally Correct Shared-State Parallel Programs

Ketil Stoelen
University of Manchester

A syntax-directed formal system for the development of totally correct programs with respect to an (unfair) shared-state parallel programming language will be presented. The programming language is basically a while-language extended with parallel- and await-constructs. The system is called LSP (Logic of Specified Programs) and can be seen as an extension of Cliff Jones' rely/guarantee method. His approach is strengthened in two respects:

- Specifications are extended with a wait-condition to allow for the development of programs whose correctness depends on synchronisation. The wait-condition can be given two alternative interpretations.
  - It can be though of as a commitment to the implementation. In this case the wait-condition is supposed to characterise the states in which the implementation may become blocked. The implementation is not allowed to become blocked

inside the body of an await-statement.

- It can also be interpreted as an assumption about the environment. In this case the implementation is assumed to be released whenever it becomes blocked in a state which satisfies the wait-condition.

- Auxiliary variables are introduced to increase the expressiveness. They are used in two different ways.
  - To strengthen a specification to eliminate undesirable implementations. In this case auxiliary variables are used as a specification tool; they are employed to characterise a program that has not yet been implemented.
  - To strengthen a specification to make it possible to prove that a certain program satisfies a a particular specification. here auxiliary variables are used as a verification tool, since they are employed to show that a given algorithm satisfies a specific property.

Although it is possible to define history related variables in LSP, the auxiliary variables may be of any type, and it is up to the user to define the auxiliary structure he prefers. Moreover, the auxiliary structure is only a part of the logic. This means that auxiliary variables do not have to be implemented as if they were ordinary programming variables. LSP has been proved to be sound and relatively complete with respect to an operational semantics.

Examples of semicomputable sets of real and complex numbers

J.V. Tucker
University College of Swansea

Computable and semicomputable sets over a many-sorted algebra will be discussed in terms of a general theory of computation and specification for abstract data types. Basic results, such as Engeler's Lemma, will be applied to algebras of real and complex numbers. Some recent results of Blum, Shub and Smale will be derived.
This work is joint with J.I. Zucker (McMaster)

Copy+Refusal Testing

Irek Ulidowski
Imperial College

Observational equivalence can be characterized by a testing equivalence induced by traces, refusals, delay, copying and global testing (Abramsky 87). We argue that global testing is unrealistic in the sense that it makes some unobservable aspects of process behaviour observable. The question arises what the strongest testing equivalence is which does not involve global testing. We present $copy + refusal$ equivalence, $\sim_{CR}$, discussed by Phillips (86,87) and Bloom and Meyer (90) as the prime candidate. For a derived transition system (actions τ abstracted away) with divergence we define a *refusal simulation* equivalence $\sim_{SR}$, a bisimulation-like relation which generalises *2/3-bisimulation* of Larsen and Skou (88) and *ready simulation* of Bloom, Istrail and Meyer (88). Analogously to Abramsky's work we show that, for image finite processes, *refusal simulation* equivalence coincides with $copy + refusal$ equivalence.
Further evidence to our choice of testing equivalence is gained by considering structural operational semantics (SOS) approach of Plotkin. We introduce a format of structured transition rules with negative premises, called Observational SOS. The OSOS is a subset of the GSOS format of Bloom, Istrail and Meyer and of the $ntyft/ntyxt$ format of Groote and Vaandrager (88,89). However, we argue that the OSOS rules are the most general rules which realistically describe the observational behaviour of processes without having a *global testing* character. We show that $\sim_{RS}$ is a *congruence* for OSOS languages. We define a trace congruence for the OSOS contexts, $\equiv_{OSOS}$, and prove that $\sim_{RS}$ refines $\equiv_{OSOS}$. Finally, we

prove that, for image finite processes, *copy + refusal* equivalence can be characterized by both *refusal simulation* equivalence and OSOS trace congruence.

Denotational Semantics for Jackson System Development

W.L. Yeung & P. Smith
Sunderland Polytechnic *G. Topping*
Staffordshire Polytechnic

Communicating Sequential Processes (CSP) was initially expounded as a programming notation for expressing a class of concurrent/distributed algorithms which are elegant and can be directly implemented on multiprocessor configurations. The paradigm of CSP involves sequential processes running concurrently and communicating with each other through message passing only. The influence of CSP on the Jackson System Development (JSD) method is acknowledged by Michael Jackson himself in his book. While a mathematical theory has been developed for CSP, JSD remains as an informal method.
This research attempts to provide a formal basis for the study of the JSD method by defining formal semantics to underpin the JSD notation based on the mathematical theory of CSP. The formal semantics are defined in denotational style which offers a high degree of modularity in the definition.
The benefits of this research include a better understanding of JSD, a theoretical basis for the standardization of the method, and a rigorous basis for the development of support tools.

A Data Model for Describing Tourist Information Requirements

Zhenhua Duan, George Row & Abdullah Hashim
University of Ulster

The Ulysses Information Self-Service Units (ISSU) is a public information system. It gives the tourist access to a central database of information on accommodation, attractions, travels, events, etc. The ISSU is being developed by the University of Ulster Magee College as part of the Ulysses International project.
In this paper, BNF is used to specify the syntax of a kind of data model for tourist information requirements. Based on this model we present an algebraic specification for the ISSU software system so that the verification and the implementation may be carried out in a rigorous way. Our motivation has also been to investigate techniques for developing a realistic software system using algebraic specification theory.

A bound of efficiency achievable by folding/unfolding transformations

Hong Zhu
Brunel University

The power of Burstall and Darlington's folding/unfolding system of program transformations is discussed. The following necessary condition of transformability is proved.
Theorem 1: If a recursive function $f = E(f)$ can be transformed to $g = E'(g)$ then there is a constant $K \geq 1$ such that for all $n \geq 0$ $E'_n(\perp) \leq E^{Kn}(\perp)$.
The well-known partial correctness and incompleteness of the system are corollaries of the theorem. Moreover,
Theorem 2: If the condition of Theorem 1 does not hold, there are no 'eureka' which can help the transformation of $f = E(f)$ to $g = E'(g)$.
based on these results, the efficiency achievable by transformations is discussed. The notion of inherent complexity of recursive programs is introduced.
Definition: Let $f = E(f)$. The inherent complexity of f w.r.t. $|.|$ is the function $CH_f : N \rightarrow N$. $CH_f(n) = max\{d(x) : |x| = n$ where $d(x) = min\{t : x \in Dom(E^t(\perp))\}$ and $|x|$ is the size of x.
It is proved that
Theorem 3: The order of inherent complexity of recursive programs cannot be improved by folding/unfolding transformations.
Since in any reasonable computation model, the time and space complexities of a recursive program are greater than or equal to the inherent complexity, it is a bound of efficiency

achievable. According to the result, linear search algorithm cannot be transformed to binary search, and quick sorting cannot be obtained from exchange sorting.

Shrinkage of de Morgan Formulae under restriction

Uri Zwick & Mike Paterson
University of Warwick

It is shown that a random restriction leaving only a fraction ϵ of the input variables unassigned reduces the expected de Morgan formula size of the induced function by at least a factor of $\epsilon^{\frac{5-\sqrt{5}}{2}} \simeq \epsilon^{1.63}$. A de Morgan formula is a formula over the basis $\{\wedge, \vee, \neg\}$
This improves a long standing result by Subbtovskaya and a recent improvement to $\epsilon^{\frac{21-\sqrt{73}}{8}} \simeq \epsilon^{1.55}$ by Nisan and Impagliazzo.
The new exponent yields an increased lower bound of $\Omega(n^{\frac{7-\sqrt{5}}{2}-o(1)})$ for the de Morgan formula size of a function defined by Andreev. This is the largest lower bound known for a function in NP.

Report on "Combinatorial Pattern Matching" school, London, 17-19 April 1991.

The CPM school was held at Royal Holloway and Bedford New College last April. The main theme of the meeting is all that concerns strings with particular emphasis on combinatorial or algorithmic aspects. Applications to the treatment of images, the compression of data, the analysis of molecular sequences or the processing of natural languages also fall into the scope of the school.

There were around thirty participants. Although the programme was rather dense, it was light enough to allow informal discussions between each others. And this is also one of the aims of these meetings to put altogether "stringologists" and (advanced) students interested in the domain.

Three mini-courses have been delivered by A. Apostolico, Z. Galil and myself. They presented some of the most recent works on strings. Several participants have also presented their own research. And the overall has been an homogeneous and very interesting meeting. Unfortunately, we had to regret that A. Ehrenfeucht has been prevented to come.

The organisation has been especially efficient, thanks to the local committee composed of A Davies, P. Hoare, C. Iliopoulos ans A. Johnstone. The RHBN College is well equipped to receive congresses (Pub, shops, bank,... on the campus) and a place that deserves a visit for its green and flowery park.

This CPM school is the second of its kind after the one held in Paris in July 1990 (see Bulletin 42 of October 1990). All participants agreed to meet next year probably in US around Easter. I wish the next CPM as successful as the present school.

Maxime Crochemore.