FACS Europe

The Newsletter of the BCS Formal Aspects of Computing Science Special Interest Group and Formal Methods Europe.

Series I Vol. 1, No. 3 Winter 1994

Contents

Editorial

Welcome to the Winter issue of FACS Europe. We had our AGM in September, the minutes appear on the next page. One of the concerns raised was the number of lapsed members of FACS. If you are one of these, treat yourself to a Christmas present by completing and sending off, before the Christmas post, the reminder sent to you by John Cooke.

As Christmas approaches, I thought it timely to include our version of A Christmas Carol! A Brief History of Formal Methods by Bernie Cohen. On reading it you will realise that Bernie has presented a very personal view which does not claim to be coherent or complete, but is thoroughly selective, hence plenty of name-dropping. Our columnists have gathered for us interesting material on Z and VDM, also included is a report of ZUM '94, and the regular recent books column.

At the last FACS Committee Meeting, there was some discussion as to how to best serve the membership. How often should the newsletter be published? Our wish is quarterly, but lack of contributions has made this difficult. Should the newsletter be electronic and available on the world-wide web? Should we stay in the present form, or should we have the topical information in electronic form and print the more substantial contributions on paper twice a year? What do you think? Let us know your views. Write or send an email to Anne Wrightson (University of Huddersfield, `scomaw@zeus.hud.ac.uk`).

At the AGM Brian Monahan relinquished his position as the editor and I am personally very grateful for his contribution particularly his input to improving the quality of presentation through the use of LaTeX, moving it from the packages of handwritten contributions stapled between two blue covers to this polished form. He will continue to serve on the committee and will continue to contribute to FACS Europe. Brian's role is being taken over by Chris Roast who joined Sheffield Hallam University after completing his doctorate at the University of York on Executing Models in Human Computer Interaction. His interests include formal methods and HCI and is currently working on the formalisation of usability requirements.

Jawed Siddiqi

Minutes of AGM of BCS FACS

Tim Denvir Chairman

The AGM of the BCS FACS was held on 14 September at the Institution of Electrical Engineers, Savoy Place, London. Attendance was low owing to a rail strike which was hampering travelling at that time, but a quorum was present: Tim Denvir, John Cooke, Roger Stone, Roger Carsley, Jawed Siddiqi.

Chairman's Report

Highlights of the past year include the 1993 Xmas Workshop on the Formal Basis of Object Oriented Systems, the Sixth Refinement Workshop on 5 - 7 January 1993, and in association with the Z User Group, the Z User Meeting In June 1994.

Stronger links have been forged with Formal Methods Europe: the newsletter is now the joint newsletter of FACS and FME; the formal methods tools database has been set up with the joint agreement of FACS and FME and is operational, available by ftp from five sites globally distributed and also via the world wide web. FACS is at present managing the FME mail list.

It was agreed to review the current free distribution of FACS Europe to the FME mail list.

Membership Secretary's Report

Benefits for members include discounted prices for attending FACS meetings, dramatically reduced rates for subscription to the Formal Aspects of Computing Journal, receipt of the, admittedly intermittent, FACS Europe newsletter, and receipt of other occasional material of interest to the formal methods community. Members can also join and renew their subscriptions to EATCS through FACS which can save currency conversion fees and be an added convenience.

Current membership is 141 fully paid up members. The number of paid up and recently lapsed members (who remain on the mailing list for a limited time) is 230.

It was agreed to maintain the membership subscription at the same rate for 1995 as in 1994.

Formal Aspects of Computing Journal

The arrangement that longer papers may be split into two parts, a synopsis published in the journal as usual and a longer version available by ftp, is now in operation. Issues 6/5, 6/6 and an extra issue 6/6A are now in hand. They total 800 pages with an additional 100 pages of extended material available by ftp. These three issues contain 7, 8 and 5 papers respectively. Several of these are "short communications".

Committee

Several members of the committee had volunteered to serve for a further year. There was some discussion of the committee structure, without any firm conclusion. Since not all committee members had notified that they were willing or otherwise to continue, it was decided to make no changes to the present committee for the moment.

Treasurer's Report

The administrative assistant, Carys Bez, had been unable to continue beyond the end of last year. There was a hope that another member of the Loughborough computer science department would take over the tasks, but this has not materialised. John Cooke and Roger Stone will continue to look for a substitute.

FACS has cash reserves of £24,000. This is £2,000 less than last year, but after taking into account commitments, which include distribution of the VIth Refinement Workshop Proceedings to attendees and programme committee members, and subscriptions to the journal and EATCS still to be forwarded, the real position is a balance of £17,000.

A Brief History of 'Formal Methods'

B. Cohen
City University

Harbingers

Formal methods are not new.

We can trace their origins back into the dawn of civilisation. The Babylonians (1800 BC) laid out their astronomical **databases** and mathematical **tables** in **first normal** cuneiform. Plato (400 BC), in the *Sophist*, gave us **structured analysis** long before we knew what to use it for. Panini, Plato's oriental contemporary, formally defined the **grammar** of Sanskrit and did it so well that he set the **standard** for a thousand years (could ISO?). Euclid defined his **geometry** using the **axiomatic** method and Diophantus formulated **effective techniques** for solving **arithmetical** problems. The Pythagoreans knew they had serious limitations, but kept them secret because the explanation violated conventional wisdom — anyone who has publicly questioned the soundness of such 'standards' as SSADM or SDL will sympathise. Al Khworizmi (800 AD) retrieved Greek techniques from the obscurity of the dark ages and, in gratitude, we gave them his name.

Newton, Leibniz, Laplace and Lagrange gave us **real analysis**, which we needed to solve the **dynamical** problems of the industrial age. The 19th century mathematicians put their **calculus** into the axiomatic framework led, as usual, by Gauss whose ***in situ* analysis** anticipated the **abstract algebra** of Galois and Abel. George Boole was putting **logic** on an algebraic footing at the same time as Lady Lovelace (Ada herself), was constructing, and **proving correct, programs** for Charles Babbage's prophetic engines. Frege formalised **propositional logic** as an axiomatic system and enunciated his **denotational imperative**, the basis of formal **semantics**.

At the turn of the century, Hilbert and Russell turned **formalisation** into a philosophical principle, which inspired Gödel, Church and Turing to formalise **computation** itself. Curry and Schönfinkel developed the **λ-calculus** and the treatment of **higher order functions** in it which is now called **currying** (mainly because it is not called *schönfinkeling*).

Although there was still little machinery to compute with, there was no shortage of things to compute. Science and engineering, business and government (particularly its military arm) were posing hard, numerical problems which were not susceptible to closed form solution in classical analysis. They demanded ever more sophisticated techniques, usually involving **convergent iterative approximation**. Practitioners of these techniques, who were known as 'computors', developed and codified them into a large body of effective, mechanically applicable methods called **numerical analysis**.

Norbert Wiener and Stafford Beer, speculating about the mechanisation of these techniques and their application to broad problems in human society, combined Turing's **automata theory** with the **General Systems Theory** of von Bertalanffy and von Förster to form the science of **cybernetics**.

George Spencer-Brown's **recursive arithmetic**, with its minimalist notation, presaged the formalisation of **non-convergent computation** and of **non-terminating, discrete systems**.

The Dawn of the Information Age

By the 1930s, when Turing, von Neumann and Zuse were contemplating the construction of electronic computers, much of the mathematical framework they needed already had a long and fruitful history. The machines that they and their successors constructed, using that framework, were enormously successful. To access their unprecedented computational power, all their users needed was the odd (and some were very odd) **programmer**, skilled in the use of the machine's impenetrable instruction set, who could cast the appropriate numerical method into a 'code' that the machine could follow.

For the first twenty years of the new age, the applications of computers were strictly limited by their enormous cost and, by today's standards, puny power. Relatively few programmers were required and their employers needed little understanding of the arcane art they practised, as long as they delivered the goods. This they undoubtedly did, although they seemed to be unable to predict with any accuracy when the task would be more than '90% complete'.

They did, however, notice that much of the work of 'coding' was repetitive and mechanical — just the job for computers themselves — and developed their own labour-saving, computational tools: assemblers, macros and, eventually, compilers for 'high level programming languages'. These were designed to enable programmers to express their 'codes' using English-like 'statements' and arithmetic-like 'expressions' involving 'meaningfully-named' variables. At last, programs could be read by their own users, although they found that legibility alone did not provide understanding and had little effect on the 90% syndrome. Attempts to rectify these linguistic inadequacies led to the proliferation of a huge variety of programming languages.

Cmdr. Grace Murray Hopper, USN, promoted her COBOL as being so close to natural language that even business users could write their own programs in it. By 1960, it was widely believed that the days of the programmer were numbered.

Verification Rules, OK?

In COBOL, the statement DIVIDE CAKE INTO THREE does not mean what it says. A correct compiler generates code which divides the value of the variable THREE by that of the variable CAKE.

There are two ways to articulate the 'meaning' of a program: as **denoting** its intended computational function and as **operating** on a computer in order to effect it. As long as programs were relatively small, their operational analysis could provide convincing evidence for their denotational adequacy. This evidence was presented either as a sample of the program's 'behaviour space', identified by judiciously selected 'test cases', or by 'walking through' a directed graph, or 'flowchart', representing the program's operational structure.

But as applications grew in size and complexity, and as high level languages diminished the intimacy between program and computer architecture, such operational arguments became less convincing. As early as 1958, AT&T had installed ESS No.1A, the first 'stored program controlled' telephone exchange, operating in 'real time' using 'interrupts'. Statisticians at Bell Labs estimated that the number of different 'paths' executable by its program exceeded 10^{400}. To test such a system, as a 'black box', to a 1% level of confidence would have required 10^{398} separate tests, which would have occupied the testers for somewhat longer than the age of the universe.

A few mathematicians realised that the **verification** of computer programs would require something more analytical than testing and flowchart walk-throughs.

Böhm and Jacopini showed that all 'imperative' programs (i.e. those executed by 'von Neumann' machines) could be represented in flowcharts containing only three constructs: sequence, selection and iteration. Floyd and Hoare showed that these constructs, together with the 'assignment' of values to variables, could be construed as **predicate transforms**. Their composition could be used in a **formal proof** that the program's operation would always **satisfy** some logical predicate ranging over the values of its input and output variables. Edsger W. Dijkstra designed a language (which did not include a GOTO statement) whose **semantics** were completely defined in this way, together with a logic of **weakest preconditions** in which to conduct the requisite **proofs of correctness**. He illustrated the technique in the pyrotechnical display of problem solving called *A Discipline of Programming*.

John Goodenough and Susan Gerhardt demonstrated, in a seminal, but sadly neglected, paper in CACM that the selection of test data for a sequential program requires the same internal analysis that would be required for its proof.

Meanwhile, a totally different tradition was being established by McCarthy, Iverson and Landin, whose **functional** languages, LISP, APL and LAMBDA, exploited the **λ-calculus**. For them, both execution and proof were to be performed by applying a set of symbolic **reduction rules** to the expression represented in a program. Programming in these languages was **declarative** rather than operational. Rather than 'following the instructions' of the programmer, the computer had to use the rules to find a solution to the equation expressed in the program's text. (Colmerauer and Warren extended this idea so that first order predicate logic, in the form of PROLOG's Horn clauses, could be used directly as a declarative programming language).

That the computer could be used to solve not only numeric problems, but symbolic ones such as mathematical proof itself, had been recognised by Herbert Simon. His General Problem Solver (GPS) was the first of a long line of **theorem provers**, leading through *Boyer-Moore,* Good's *Gypsy* and the Edinburgh *LCF* to Abrial's *B-tool.*

Alarums and Diversions

In the boom years of the 60s, little of this work filtered through into practice. Technology was moving apace. IBM dominated the burgeoning market of business applications. Mastering the quirks of the rapidly multiplying versions of OS360, COBOL and FORTRAN required so much detailed, but ephemeral, expertise that demand for programmers, far from disappearing, vastly exceeded the supply. A new craft was born, highly lucrative and free of the professional and academic restrictions imposed by other disciplines.

Despite such heavy commercial damping, a few sparks burned. APLers, vowing that they would rather die than switch, delighted in displaying mathematically elegant, highly efficient, provably correct, higher order, functional one-liners that were totally unintelligible. Ivan Falkoff, anticipating *VIPER* by twenty years, gave an APL definition of the *IBM360*, showing, incidentally, that almost half of its semantics lay in the *Channel*, its I/O port — an indicator which the telecommunications community utterly failed to notice. LISP took off in the rapidly developing world of Artificial Intelligence, where the symbolic dominated the numeric and empirical

experimentation dominated systems analysis (which might well have contributed to Lighthill's damaging report).

In 1968 and 1970, two NATO-sponsored conferences gave the world a new term: 'Software Engineering'. All the leading theoreticians and practitioners were represented and the proceedings (now sadly out of print) were widely circulated. They called for more discipline — in requirements analysis, in verification and validation, in estimation and measurement, in documentation and in management. They indicated, often with surprising accuracy, the directions that research and development, training, education and practice needed to follow.

The academic institutions had been responding as best they could. 'Computer Science' courses concentrated on the theoretical foundations which were then most applicable — largely those of language theory, for compiler writers, and the theory of computation, for algorithm designers — and on programming praxis. 'Computer Engineering' courses delved into the electronics and architectural features of the new machines but rarely ventured into the abstract world of theoretical computation. 'Pure Mathematics' courses largely ignored these developments and continued to teach the right mathematics in the wrong way. 'Applied Mathematics', and most traditional 'Engineering' courses saw computers as a passing fad, or as an occasionally useful tool, and taught only the elements of the programming craft. Teachers, after all, need as much professional development as practitioners.

Scottery

The untimely death of Christopher Strachey, Oxford Professor of Computation, scion of the Bloomsbury set and author of '**C**hristopher's **P**rogramming **L**anguage' — the progenitor of C, ended a remarkable research partnership. With Dana Scott, he had developed **denotational semantics**, a mathematical framework in which the semantics of all the imperative programming languages could be defined.

This relies on the definition of a **domain** in which the computational effect of every syntactically correct program can be represented. This is much harder to do than it seems, because all programming languages permit the expression of programs which do not terminate; they can legitimately enter 'endless loops', as craft programmers know only too well. Most also contain the GOTO statement, which Dijkstra 'considered harmful' because of its deleterious effects on verifiability.

To accommodate such pathological behaviour, Scott-Strachey semantics defines domains in the **lattice theory** anticipated by Spencer-Brown. The semantic function evaluates each syntactic construct of the language to a function on the lattice theoretic domain. It takes as additional arguments the (recursively evaluated) syntactic components of the statement(s) involved and the current state of the statement's environment. The semantics of a looping (or recursive) construct is defined as the least fixed-point of the chain of successively more defined functions generated by each iteration of the loop. Since a non-terminating loop never produces a fully defined result, its 'value' is a function that always returns 'bottom'. The semantics of the GOTO statement takes as its argument the semantics of the entire program to which it 'goes' — the so-called

'continuation'. The semantics of a program is composed from the semantics of its parts, as demanded by Frege's denotational imperative.

Denotational semantics made it possible, in principle, to determine whether any program, in any programming language whose semantics were so defined, computed the function for which it was commissioned. There were only a few problems. One was the enormous labour involved in providing all the programming languages then in use with a formal semantics. Another was the seemingly gratuitous complexity that emerged whenever the semantics of a 'real' language was analysed. Clearly, semantic cogency was not the highest priority for language designers (although it was hard to discern what was). A third was the inescapable fact that the real definition of a language's semantics, regardless of what the designer, or even the programmers' handbook said, lay deeply buried in its compiler.

Finally, there was the notational and conceptual unfamiliarity of lattice theory, domain theory and the whole paraphernalia of 'Scottery'.

It appeared that we could, at last, reason formally about our programs, but only if we found enough domain theoreticians to build the semantics, enough logicians to construct the proofs and enough compiler designers capable of understanding both well enough to write correct compilers. As Scheutz said of Babbage's first engine, 'even with all England's technical expertise, it would be impossible to advance further, as long as one followed the same plan'.

Peter Mosses tried to cut this Gordian (Plotkian?) Knot by constructing a compiler-compiler that could read denotational semantics, but the compilers it generated were too slow to be taken seriously. They could, however, be used as benchmarks against which commercial compilers could be (partially) checked, a procedure which gained credence much later in the US DoD.

Denotational Semantics was taken very seriously by computer scientists on both sides of the Atlantic, Joe Stoy being invited from Oxford to teach it in MIT. His lecture notes, published by MIT Press, constituted the main text on the subject because the sourcebook, completed after Strachey's death by his research student, Robert Milne, was considered too difficult to teach from.

Tales from the Vienna Woods

Dr. Heinz Zemanek, Director of IBM's Vienna Laboratory in the early 70s, was one of the first to recognise the wider implications of formal semantics. With astounding foresight, he sought to provide one not only to IBM's latest 'standard' programming language, PL/1, but also to the instruction set of the kind of processor to which it might be compiled and to the requirements of systems which might be implemented in it. For none of these applications did he choose to use the Scott-Strachey approach.

J. A. N. Lee's book, *Computer Semantics*, records the first attempt: the 'Vienna Definition Language (VDL), which provided a tree-structured domain in which the 'state' of a virtual processor could be represented, its operations being modelled by formally defined transformations on the tree.

In parallel, the PL/1 team, led by Hans Bekić, were developing their own modelling framework, a set-theoretic language called MetaIV (get it?). This proved to be more successful than VDL, possibly because its notation posed fewer problems to beginners. The PL/1 semantics was published and shipped off to IBM's compiler development team. History does not record their reaction, but no PL/1 complier was ever proved to comply with Vienna semantics.

Peter Lucas used MetaIV, with encouraging success, to specify a fairly complex database application. His technique came to be known as the 'Vienna Definition Method' (VDM), which Dines Bjørner and Cliff Jones, promoting it with their books and supporting it with tools developed in Copenhagen and Manchester, eventually turned into the pan-European product now known as RAISE.

VDM differs from the Scott-Strachey approach in that it is founded on sets rather than lattices. A system specification in VDM consists of a model of the system's state space whose components may take certain set-theoretic values, including powersets and mappings. The collection of components is constrained by (invariant) predicates and the system's behaviour is modelled by operations on the state, defined in terms of pre- and post-conditions.

This basis was found to be powerful and sufficient for many applications, such as Lucas's database, but presented significant problems for the specifiers of programming language semantics and of 'real time' systems.

Sets in the West

In the early 70s, Dr. Patrick Doyle, a mathematician with the Irish Life Insurance Company in Dublin, was commissioned to develop a sales commission tracking system. Not being a 'systems analyst', he tackled the problem in an unconventional way: by constructing a model of the required system in set theory. Although he believed that the model he had constructed captured all the requirements of the potential users of the system, he felt that it should be signed off as an acceptable specification before he proceeded to implement it. So he offered the appropriate authority, the Board itself, an interesting alternative: either to receive a long, rather boring and probably ambiguous English-language document, which he could derive from his model, or to follow a short course in elementary set theory which would enable the Board members to read and understand his specification in its original form. The Board took the course, read and understood the formal specification, made some suggestions for change and signed it off. Doyle turned the model into a collection of precise software module specifications which he passed to a small team of (non-mathematical) programmers, who coded and 'integrated' the modules. The system worked first time! Paddy Doyle was so far ahead of his time that he had to publish his own book, *Every Object is a System* (still available from its author), in which he presents his unique view of the rôle of mathematics in information system design, concluding that, ultimately, it is an exercise in **topological manifolds**.

At about the same time, Jean-Raymond Abrial and Steve Schuman, in the IRIA laboratory in France, were also investigating the use of set theory as a medium for system specification. They called their notation *Z* (after Zermelo and Fränkel, who had defined the well-founded set theory on which they relied). Z was taken up by the Programming Research Group at Oxford University, by then under the leadership of Strachey's successor, Tony Hoare, where it was enriched, supported by tools and applied to several real problems in industry and commerce. One of these was the CAVIAR system for administering visitors to STL Harlow, ITT's main laboratory. Abrial himself interviewed the client, Gladys, who manually maintained the records and bookings for the 12000 visitors who passed through STL each year, and constructed the (very elegant) Z specification. However, unlike Doyle, he made no attempt to instruct Gladys in the mysteries of

set theory. Instead, he 'validated' his model by deriving from it ten theorems ('emergent' properties of the model), each of which could be cast in the form of a simple, English-language statement about the system, such as: 'No two visitors shall share the same hotel room', and asked Gladys to confirm, or deny, them. Gladys gladly did so and the system was duly implemented.

Z has since become, with VDM, one of the main vehicles for formal specification.

Letters from America

Meanwhile, back on the ranch ... in the USA, the need for a mathematical formal approach to the design of computer-based systems had also been recognised by the mid '60s. Much of the research was being done in IBM laboratories — in Yorktown Heights, in San Jose, in Federal Systems Division and in the tiny Scientific Centre on MIT's campus in Cambridge, Massachusetts. The rest of IBM, however, then as now, paid little heed to these impractical mathematicians and concentrated their 'mythical man-months' on the 'real world' of OS360.

One of the most important IBM research groups called themselves 'ADJ', which is neither an acronym nor a clue to the authors' identity but an abbreviation for 'adjoint functor'. Thatcher, Wagner and Wright, together with Stanford University's Joe Goguen, published their 'Junction between Category Theory and Computer Science' in about 1975. This set of documents, together with Eric Wagner's course notes and the elegant papers published jointly by Goguen and Rod Burstall of Edinburgh University, were widely circulated and introduced many computer scientists to the mysteries of abstract algebra.

Steve Zilles, at Yorktown Heights, used algebraic techniques for the specification of what are now known popularly as 'abstract data types'. John Guttag and Barbara Liskov, of the Laboratory for Computer Science at MIT, developed these into the influential (and executable) specification language CLU.

Another algebraically-inspired group was at work just down the road from MIT at a Space Programme spin-off company called Higher Order Software. Margaret Hamilton and Saydean Zeldin had been software project leaders on NASA's Apollo Programme and NASA invited them to discover how the (relatively few) bugs in Apollo software had escaped their stringent QA procedures. Their analysis showed that a strictly **functional** approach to design would have been both effective and feasible and they developed their ideas into the first of the 'formal methods' — HOS. This toolset provides a graphical specification language, supporting the 'hierarchical', 'top-down', 'functional decomposition' of a system requirement. Strict constraints apply to the structural elements used at each level so that the leaves of the resulting tree constitute a strictly functional program, whose variables range over the abstract data types defined in the associated algebraic specification language, AXES, and whose functions are just the constructors and derived operators defined for those types. Translators were provided for both the graphical language (into more popular, structured, but less formal forms, such as HIPO, SADT, PSL/PSA and Yourdon) and the algebraic language (into various compilable programming languages) so that the results of analysis and design in HOS are presentable to the client, verifiably consistent and executable. What more could one ask for?

HOS caused considerable excitement in certain quarters: James Martin, the great guru himself, liked it so much that he not only wrote a book about it (*Application Development Without Programmers*, which ushered in the era of CASE tools) but bought the company! A lesser known development occurred in the small Computing Unit of the Royal Marsden Hospital, in Surrey, where Jo Milan was seeking a way to reduce the often repetitive labour of developing clinical and administrative applications in a MUMPS (a popular dbms) environment. He saw the potential in HOS but was unable to justify the high cost of the toolset, so he reconstructed it!

Curiously, HOS seemed to fall between two stools: it was never taken seriously by the academic fraternity, even in its home town of Boston, and industrial take-up was negligible — even with the backing of Martin and the management team he imported, who promoted it heavily at the highest levels of the computing industry, but never seemed to appreciate that, like all good engineering tools, only those who understood its theoretical foundations could use it effectively.

The other major concern, and source of funding, of formalists in the USA was security, particularly the verifiability of secure operating systems. In the late 60s, the US DoD let a contract for a military message switch known as SATIN4. This contract contained a clause, unnoticed by the prime contractor (ITT) until late in the development, that the kernel of the operating system be verifiably immune from unauthorised access. In including this clause, the client had been influenced by published research results in program verification and mechanised theorem proving. When the developers of SATIN4 realised the significance of the verification clause, they invited appropriate research groups to quote a price for verifying its kernel, which consisted of several hundred thousand lines of PL/1. The best offer they could get was a 'ballpark' estimate, from Gerry Estrin of UCLA, of $10000 per line of code! Although not a sensible measure of the effort of verification, this was nevertheless sufficient to persuade Congress to abandon the project!

The DoD subsequently published its celebrated 'Orange Book': the *Trusted Computer Base*, which defined such concepts as 'Security Policy Model' and 'Levels of Integrity' and laid down criteria for the evidence which would have to be submitted when certifying systems as being acceptably secure in different circumstances. At the highest levels, this evidence must include a mathematical proof that the code implements the security policy.

These demands from the security community generated considerable support for work in mechanised theorem provers and formal specification techniques. The best known theorem provers, *Boyer-Moore* and *Gypsy* are both now the property of Don Good's company in Austin, Texas, but similar work was also being done by Nagel in Ford Aerospace. SRI International led the work in formal specification: Peter Melliar-Smith (ex-Newcastle University) developed *HDM* and used it to define and verify PSOS, the *Provably Secure Operating System*, while Joe Goguen and Jose Meseguer, working with Rod Burstall at Edinburgh, developed *CLEAR* and its successors *OBJ*, *OBJ1* and *OBJ2*, which were later commercialised by Hewlett-Packard in Bristol under the leadership of Robin Gallimore, who had worked on them while in Manchester University.

Thinking in Parallel

The problems of **concurrency** started to plague computing as soon as external storage devices were connected and communication among computers was established. These problems fall

into two categories, both of which were already represented in classical electromechanical systems. In **control** systems, from device controllers to avionics, **feedback** and **delay** are fundamental to the system's behaviour, but classical control theory offered little succour to the software designer.

In **parallel processing**, the problem is to predict how communicating, sequential processes behave in concert, particularly when they compete for the same, scarce resources. The telecommunications industry had built up a great deal of experience and valuable theoretical foundation — from Erlang's traffic theory to Moore and Mealy's switching theory — in its development of complex signalling systems and their corresponding relay sets but, again, none of this seemed to solve the software design problems.

Of course, part of the trouble was, and still is, the reluctance of engineers in disparate disciplines to recognise the utility of each others' design principles. The advent of computer-based control, and particularly the emergence of the cheap, powerful, easily (?) reprogrammable microprocessor, coalesced the technologies of previously specialised application domains — providing 'priority interrupts', 'schedulers' and even 'general purpose operating systems' — but did not unify their theoretical foundations.

In 1960, Carl Adam Petri presented his Doctoral thesis in Mathematics to the University of Bonn. It postulated a graph-theoretic structure, the *Petri Net*, as a model for **concurrent processes**. This model is not equivalent to the 'finite state machine' of switching and computational theory. The states of the systems are not enumerated but characterised by the 'reachable' markings, which may be explored both analytically, using graph-theoretic theorems, and under simulation, by 'playing the token game'. Certain well-known pathological behaviours of composite (i.e. communicating) state machines, such as 'Mexican stand-off', 'resource starvation' and 'priority blocking', are identifiable with (the absence of) net properties, such as 'liveness', 'safeness' and 'fairness'.

Net Theory has generated a wide following, mainly in Petri's own Institute (part of GMD in Bonn), in Denmark and in France Its applications extend from the semantics of programming languages (such as *PEARL*) to the analysis of protocols and operating systems.

In the mid-60s, the problem of concurrent processes was theme of a UK Research Programme called 'Distributed Computer Systems' (DCS). This funded work both on parallel computing architectures and on the semantic foundations of concurrency. The main centres for the latter were the Universities of Oxford, where Tony Hoare developed his notation for 'Communicating Sequential Processes' (*CSP*), and Edinburgh, where Robin Milner defined his 'Calculus of Communicating Systems' (*CCS*) and its 'Synchronous' version (*SCCS*). These are **process algebras** in which the expressions denote (recursive) process definitions and their composition: conjunctive (in parallel) and disjunctive (through 'choice', which may be either deterministic or non-deterministic). Equivalence over these expressions, as defined by the axioms of the algebra, is intended to correspond to the indistinguishability, by mere observation, of the processes which they denote. This 'observational congruence' turns out to be extremely difficult to capture algebraically (which may be a clue as to why concurrent systems themselves seem to be so difficult to design). The late David Park, of Warwick University, was instrumental in defining the apropriate 'bisimulation' congruence of CCS.

The DCS Programme also supported Milner's theorem proving work at Edinburgh. This relied on the development of an (executable) meta-language, *ML*, based on the λ-calculus, in which to define reductions and transformations of the algebraic structures over which theorems were to be proved. This language has become a powerful programming system (*Standard ML*) in its own right and the original theorem provers have been developed into the *LCF* and *Concurrency Workbench.*

Both Petri Nets and the process algebras reflect the **non-determinism** inherent in the semantics of concurrency. That is, even a complete knowledge of the behaviours of each of the processes in a concurrent system is, in general, insufficient to determine the state that the system will reach after a given sequence of external events. One can, however, determine the set of states (or, equivalently, the properties of those states) which are so 'reachable'. This suggests that to reason over the behaviours of concurrent systems, one must discuss both the 'possibility' and the 'necessity' that a given predicate hold; that is, one must use a **modal** logic. In the early 70s, Zohar Manna and Amir Pnueli of the Weizmann Institute, Leslie Lamport of Stanford University and others, analysed concurrency in terms of the **temporal** modality, whose quantifiers , 'always' and 'eventually' apply to predicates over states that the system might enter, now and at some 'time' in the future.

Most people who are introduced to these theories of concurrency find it surprising that there is so little reference to the familiar, engineering notions of **time** as 'duration' and 'instant'. (The 'Timed Petri Net' and Ben Moszkowski's 'Temporal Interval Logic' present similar concepts, and even they are not entirely intuitive.) Yet Einstein and Heisenberg had pointed out the fallacies of these intuitions at the turn of the century; Ivor Catt and Chuck Seitz had identified them as the source of **the glitch**, the bane of digital systems design, in the 60s; and the more recent work on **chaos** has revealed many other serious analytical problems at the mathematical boundary between the continuous and the discrete. Yet there are still practising engineers who demand a fully analytical framework for causality and performance measurement in concurrent systems and, worse still, there are many charlatans in the 'methodology' business who are only too pleased to sell them one.

Roots

The rapid progress of electronic miniaturisation soon confronted hardware designers with levels of behavioural complexity similar to those already seen in software. In addition, the naturally occurring structures in the hardware world were fundamentally concurrent; synchronisation was both a conceptual problem and a physical one, as the Glitch investigators knew.

The formal specification of digital electronic systems, as branch of concurrency theory, was investigated by several Euro-American groups, notably the highly innovative team formed by Carver Mead and John Gray in Caltech. This included Chuck Seitz, who was aware of Net Theory at a very early stage thanks to his Glitch colleagues, Bob Shapiro and Tolly Holt. Martin Rem, one of Dijkstra's students from Eindhoven was there, as was George Milne, who turned Milner's *CCS* into *CIRCAL*, the first successful discrete circuit calculus.

Meanwhile, in Cambridge University, Mike Gordon was developing *HOL* for similar purposes, with the help of Ben Moszkowski and Avra Cohn of Stanford. This was to be used in the specification and verification of RSRE's *VIPER* chip, a huge

undertaking whose repercussions, especially in Australian Railways, put "formal methods" firmly into the legal and commercial arenas.

The Telecom Universities

By the mid 70s, the results of research programmes in algebra, logic, set theory and concurrency were sufficiently mature and well-promulgated to encourage their exploitation in industrial laboratories. The two industries which led the way were defence (mainly in the security branches whence, unfortunately, few results emerge) and telecommunications, which, for decades, had been pushing technology to its limits of complexity. Centres for 'Applied Software Research' were established, somewhat reluctantly, in the Laboratories of the major Telecoms manufacturers, and even some of their customers, the PTTs.

ITT had four such centres, the first and biggest in STL Harlow (under Tim Denvir and the author), smaller ones in BTM Antwerp (Raymond Boute) and ITTLS Madrid (Felix Vidondo); GEC had one in its Hirst Laboratory, Wembley (Peter Scharbach and Jim Woodcock) ; Plessey in its Roke Manor Lab (John Smith); and Ericsson in Stockholm (Bjarne Däcker).

The STL Centre ran an influential Symposium on 'Formal Design Methodologies' in 1979, inviting many of proponents of both 'structured' and 'formal' techniques of system specification and design to display their wares for critical review by their peers and by their prospective users. As a result of this meeting, the STL group committed itself to formal approaches under the following four main headings:

a) protocol specification, where Mike Shields, working on a CASE award through Robin Milner, showed, among other things, that no interesting system properties were decidable over specifications written in SDL, which was, at the time, the CCITT's standard protocol specification language. This fact seemed to have had no impact whatsoever on the standards committee!

b) model-based specification, in both Z, with Steve Schuman at Oxford, and VDM, with Brian Monahan at Manchester. One result was the establishment, under Mel Jackson, of the VDM Standardisation Group.

c) theorem provers, or rather their generation from equational axioms, under Will Harwood, including the development of the *NIMBUS*, *EST* and eventually *GENESIS* toolsets.

d) professional development in the requisite mathematical foundations, including internal courses, teaching each other the skills we had acquired separately, and external courses, organised through Hatfield Polytechnic, for our colleagues in ITT's operating divisions. The results of these educational ventures were unexpected. The internal ones flushed out of the woodwork many highly qualified pure mathematicians who were already in the company's employ but who were unaware that their mathematical skills had become applicable. The external ones revealed that the essential mathematical ideas were not difficult to teach to practitioners as long as they were presented in appropriate sequence and context, which, unfortunately, no mathematical textbook yet does.

This issue of professional development has been troubling the telecommunications industry for many years. BTM, Phillips, STC, GEC and BT all commissioned reports on the post-experience 'Software Engineering' curriculum. These reports are all remarkably similar to the first one, produced in 1974 by Raymond Boute (now Professor of Computer Science at Nijmegen), which called for courses in classical mathematics: control theory, queuing theory, reliability theory

etc., as well as the discrete mathematics underlying computer science. Unfortunately, they also suffered a similar fate to Boute's: shelved for lack of funds! Three years ago, BT finally commisioned London University to develop and teach an MSc for its systems engineers. Following protests from Sinclair Stockman, the number of software modules in the course was increased from one to two! *Plus ça change ...*

The Telecoms Labs also took a serious interest in hardware specification. Strachey's student, Robert Milne, developed his *LTS* at STL and Raymond Boute originally conceived his *GLASS* while still in BTM.

Anno Mirabilis — 1979

During the year following STL's FDM symposium, three significant events took place in Europe. The first was the Winter School on 'Abstract Software Specifications', held at the University of Copenhagen. It brought together most of the workers in the fields of semantics, specification languages, concurrency and theorem proving, from both sides of the Atlantic. It provided a forum in which they could present their work to each other, to research students and, significantly, to investigators from industrial laboratories. One of the objectives was to discover if Meta-IV could be improved as a semantic metalanguage without going the whole hog of introducing Scott's lattice-theoretic Domains. (There was a problem with the semantics of GOTO, which Meta-IV handled with a very curious construct called *TIXE* — or EXIT backwards!) Although the answer was largely negative, the confrontation was extremely constructive and established many contacts which would later be exploited in the ESPRIT Programme and the Z/VDM Conferences.

A similar effect was produced by the Conference on the Semantics of Concurrency held in Evian. The relationships among theories grounded in different mathematical frameworks were starting to become clearer, as were their relative advantages and disadvantages when applied to different classes of problem and in different application domains.

By contrast, the meeting of the IFIP Information Processing group in Oxford that year, presented with a mix of material similar to, and, in fact overlapping with, that of the previous month's FDM Symposium, preferred informal, structured approaches (particularly those supported by interactive, graphical tools) to the mathematically formal ones, thereby setting a trend from which the Information Processing industry is only now starting to emerge.

Modern Times

The 80s saw formal methods finally emerging from the ivory tower to which they had been confined by industrial scepticism. The reasons for this change of heart are not straightforward. John Buxton attributes it to the two most powerful human motivators: fear and greed.

The latter was encouraged by the sudden influx of funding from the great European IT Research Programmes, Alvey and ESPRIT (both of which were themselves inspired by fear of a Japanese threat). Both were heavily influenced by the industrial giants of European IT (the 'gang of 12'), whose laboratories had been experimenting with formal techniques in the 70s. Large amounts of funds were also made available by agencies which had identified real needs for these techniques, notably the US Strategic Defence Initiative ('Star Wars') and the National Security people — NSA and GCHQ — on both sides of the Atlantic.

Some members of the formalist community saw these as Wilde saw the protagonists in a fox hunt: the one uneatable, the other unspeakable.

These same Government agencies were being motivated by fear. David Parnas, who had long promoted structured programming techniques, presented his resignation from the SDI Programme Office in the form of 12 public papers which seriously questioned the certifiability of 'Star Wars' systems. His main point is that testing alone cannot produce the evidence that a complex, adaptive system will satisfy its operational requirements (full operational tests of SDI systems would have to be conducted in about 4 minutes), while formal proofs of correctness cannot even be attempted without formal specifications of both requirements and components.

The verifiability of security systems took on new significance with the almost simultaneous appearance of two opposite, and apparently dissociated, trends:

a) the discovery, and publication, of practically unbreakable cryptographic techniques, such as the Rivest-Shamir-Adelman system, based on 'trapdoor' functions in Galois fields, and the concepts of public and private keys.

b) 'computer hacking', including both unauthorised access to supposedly secure systems often, but not always, for pecuniary gain, and the spread of potentially deadly 'viruses', both of which were being perpetrated by young, otherwise respectable, computer buffs, with very peculiar friends, who could operate from undetectable bases over the world's computer networks.

Both of these provoked fear reactions in Government security agencies, the first that the public could operate computer and communications systems that they could not break, and the second that the very technology on which modern security relied was its Achilles heel. These fears have given rise to some strange policies, including the NSA's refusal to permit commercial systems to employ *RSA* (until, presumably, they can break it themselves) and the US Government's embargo on the export of the *Gypsy* theorem prover. They have also led Europe to develop its own, more stringent, version of the US DoD's 'Orange Book', called *ITSEC*, which calls for the use of formal techniques of specification and verification at the highest levels of security.

Similar fears are motivating the reformulation of draft standards by regulatory agencies responsible for Safety in various sectors, including Military (MoD 0055/0056), avionics (RTCA 186B), industrial health (the UK Health and Safety Executive's PES) and the UN's own 'top' safety standard, currently identified as 'Secretariat 95A'. All of these are, somewhat reluctantly, incorporating references to the potential rôle of 'formal methods' in the generation of evidence for certification of software in 'safety-critical' systems.

Some civil safety-critical system development projects have already anticipated these standards, most visibly in the domain of railway signalling, mainly in France and Australia, and, with less publicity, in such sensitive areas as nuclear power station control and the avionics systems of 'fly-by-wire' passenger aircraft. Formal specification techniques, such as *Z*, *VDM*, *OBJ*, *LUSTRE* and Harel's *Statemate*, have been deployed, together with Static Analysis of the generated code (using *MALPAS*, *SPADE* and *Logiscope*), to assist proof that the system is immune from 'threats' to its safety. In the UK, Rolls Royce Associates have demonstrated that a development process incorporating these can be at least as economical and productive as one which relies on the best informal techniques. Even taking into account the

huge cost of assembling the documentary evidence required by the certification authorities' standards, they have found that safety-critical systems are no more expensive to develop using formal techniques than are 'normal' industrial control systems using informal techniques.

Composition, Consistency and Agents

In the 90s, two classes of problem have presented analytical challenges that offer major opportunities to the formalist.

One is "feature interaction" in telecoms systems (a very old problem in desperate need of a theoretical base more powerful than the classical "state machine"), which Bellcore have identified as the biggest single barrier to open systems. I addressed this issue earlier this month at BNR Europe's lab in Harlow (which used to be STL). It will be explored further next month at the Second International Workshop on Feature Interaction, at the Dutch PTT's lab in Amsterdam.

Feature interaction is not, however, confined to telecoms systems. A British Ford Dealer recently wanted to promote the security features of the new Mondeo, which Ford have advertised as 'unstealable'. He challenged a car thief to get into it in front of the local press. The thief strolled around the car for a few minutes then kicked it hard on the front bumper. This immediately inflated the airbags, a safety feature which , correctly, overrode the security features and threw open all the doors. The thief stepped in, hot-wired the ignition was off in half a minute.

The second problem class is "Business Process Reengineering", where the management consultantcies are discovering just how inadequate is the analytical power of their current modelling frameworks – soft systems, dataflow, entity-relation and even object-oriented – which derive from our own software methodologies. The work of Michael Glykas, first at Cambridge and City Universities and recently with the Greek Tobacco Company, has demonstrated that a judicious combination of formal and informal notations can yield enormous gains in understanding of the complex interrelationships among the 'agents' of the modern corporation.

The key to both of these problem classes is the ability to construct large models by the composition of small ones, to verify their internal consistency and to explore the validity of their consequences, in such a way that both the composition calculus and the reasoning logic 'scale up' through composition.

Both of these classes of problem also demand a formal treatment of 'agents' – objects that negotiate contractual relationships with each other, if necessary changing their own obligations, responsibilities and 'theories of the world' in the process. We do not, as yet, have a suitable mathematical framework for manipulating, composing and reasoning about them.

Having explored most of the formal specification styles over the last fifteen years, the only one I have found that comes near to satisfying all of these criteria is the variant of Z developed by Steve Schuman and Dave Pitt at the University of Surrey. However, as this is practised by relatively few people, has no proprietary tools to support it, and is not the subject of an international standardisation movement, it has had little impact on the Formal Methods world.

Conclusion

At last, fear, greed and soundness seem to be converging.

Formal Methods are about to take their place among the Engineering Disciplines.

All we need now are the responsible, professional engineers who will use them.

Functional Programing

Philip Wadler, University of Glasgow[1]

Welcome to the first functional programming column! Functional programmers and formal methoders have much to say to each other, and this column is intended to further such communication.

Functional programming supports formal methods. A large number of proof systems and other formal methods packages have been implemented in functional languages. ML, the doyen of functional languages, takes its name from its first application as a meta language for writing theorem provers. Functional languages may also be suitable for writing executable specifications, for rapidly generating prototypes from specifications, or as the target language of a system that transforms specifications into efficiently executable code.

Conversely, formal methods support functional programming. Functional languages are well known for their amenity to mathematical reasoning, and formal methods can supply tools to support such reasoning. The application of mathematical laws to functional programs is not just a dream for the future – it underpins most of the techniques for generating efficient code from functional languages.

Contributions for future columns are invited – my postal and e-mail addresses appear on this page. I intend to interpret functional programming in the broad sense, ranging from lazy languages with absolutely no side effects, such as Haskell, to strict languages with disciplined use of side effects, such as Standard ML.

The first column deals with Erlang, a language of the latter sort. One impediment to the spread of functional programming is that most of the efforts have been academic in nature, and relatively little effort has gone into matters like programming environments that can make or break the suitability of a system for use in industry. Mike Williams of Ericsson has led the effort to develop Erlang, which is provides a convincing demonstration that functional languages can have industrial relevance. All of us interested in transfering ideas from academia to industry will be keeping an eye on Erlang to see how well it succeeds. Although it doesn't incorporate all the latest academic niceties – for instance, it lacks a type system – if successful it may pave the way for a generation of functional languages and other ivory tower innovations to stroll out of the classroom and into the software houses.

[1]Professor Philip Wadler, Department of Computing Science, University of Glasgow, Glasgow G12 8QQ, Scotland. E-mail: wadler@dcs.glasgow.ac.uk.

Functional Programming Column

Erlang – A Functional Programming Language for Developing Real-Time Products

Mike Williams, Ericcson Infocom Consultants AB[2]

1 Background

Software development is by far the largest part of the design costs for Ericsson's products. A single AXE-10 telephone exchange, for example, has on average software corresponding to 62 million lines of source code in C, C++, EriPascal and PLEX. Reducing the cost of software design, improving quality (fewer bugs) and cutting time to market are thus of paramount importance.

Using a programming technology which results in smaller and more understandable programs would improve the situation. A functional language is an obvious alternative.

We have designed the functional programming language 'Erlang' for this purpose. Using Erlang as the implementation language in several real-time system products has resulted in considerable reduction both in the size of programs and in the work required for software design.

2 Why do we need a new language?

Why not use ML or Haskell or some similar language? To answer this question we must describe the problems which need to be solved.

Massive
: The sort of system we are considering are *massive*, which means that they contain tens of millions of lines of source code.

Long Lived
: Systems must survive and evolve over a long period of time. For example, the AXE-10 system, which is still a best seller today was designed in the mid 70's. The technology we use now must be able to inter-work both with older software and with new technology which no doubt will turn up in the future.

Distributed
: Systems are made from a large number of different types of computers running different operating systems and communicating with each other by a variety of mechanisms.

Concurrent
: Thousands of different activities must be handled at the same time. These activities (processes) must be able to communicate with each other. We are not primarily interested in parallelism - splitting up a computation onto several different computers to gain speed - we are interested in handling the massive concurrency we already have!

[2]Mike Williams, Ericsson Infocom Consultants AB, Erlang Systems Division, Box 1153, S-164 22 Kista, Sweden; and Ellemtel Utvecklings AB, Box 1505, S 125 25 Älvsjö, Sweden. E-mail: mike@erix.ericsson.se.

Robust
: Failure of a part a system, due to a hardware failure or a software fault, should have as little effect on the whole system as possible. It must be possible to speedily recover from failures and go on running despite the fact that parts of the system may be out of service.

Non Stop
: Some systems cannot be stopped to make software changes, to correct bugs, to add new software or to remove unused software. It must thus be possible to do this while the system is running and to ensure the software changes are made in such a way that the services offered are disturbed as little as possible.

Real-time
: Many actions must be completed within a fixed time after receiving a stimulus. Language implementations which cause significant pauses in execution, for example for garbage collection, are thus unsuitable.

Very few languages or implementations of languages and operating systems meet the criteria above. As far as we are aware, Erlang is the only functional language designed and implemented with these criteria in mind, criteria which apply to an increasingly large class of real-time systems apart from telecommunication applications.

3 Erlang - The Language

For a detailed description of Erlang see [1]. It has been unkindly said that Erlang is a functional language without proper functions and a logic language without logical variables. More kindly (perhaps) it has been described as a sort of concurrent Lisp.

The following points give a very concise summary of Erlang:

- Dynamically typed. Single assignment variables
- All choice / selection done by pattern matching with shallow guards
- Recursion equations
- Explicit light weight processes part of the language
- Relatively free from side effects
- Asynchronous message passing between processes. Message reception is selective, a process can wait for messages which match a number of patterns and cause other messages to be queued.
- Primitives for detecting run-time errors, including errors in other processes
- Transparent cross-platform distribution. The message passing primitives are the same for messages sent between processes on the same processor and between processors. The same applies to the error detection primitives.
- Real-time garbage collection
- Modules, export and import declarations
- Dynamic code replacement - you can replace code in running systems.

- Foreign language interface

The language is eager. It is also very compact and easy to learn. Everything which was found to impair efficiency or which was difficult to implement in a distributed non homogeneous environment was excluded from the language. The language thus does not support Currying, higher order functions, lazy evaluation, ZF comprehension or deep guards.

The lack of higher order functions is alleviated by the built-in function apply, a function supplied as an argument is applied to a list of arguments.

4 Implementations

4.1 JAM Implementation of Erlang

In the JAM (Joe's Abstract Machine) implementation. Erlang [2] is compiled to code for a virtual machine, the JAM. The Erlang compiler is itself written in Erlang. The JAM is implemented by an emulator written in "C". The emulator also implements code loading, memory management, concurrency, garbage collection, distribution (at present with TCP/IP) and built-in functions.

Each Erlang module is compiled to a separate file of instructions which can be executed by the emulator. The emulator loads these files as they are needed. Modules can also be pre-loaded. Modules which are already loaded can be replaced by modified modules - even if these modules are being used.

Garbage collection is done on a per internal Erlang process basis. This means that pauses for garbage collection will be very short provided that there are no very large processes.

Supported versions of this implementation are available (at present) on

- SUN Workstations (both SunOS4 and Solaris 2)
- RS 6000 / AIX
- Interactive Systems UNIX on Intel 486
- VXWorks on Force 68040 and SPARC
- HP 9000 HP-UX

Unsupported versions are available for Intel 486 on MSDos.

This implementation is the most widespread implementation of Erlang and the one which is most suitable for developing and testing software. The code emulated by the JAM (i.e. the output of the compiler) is very compact.

4.2 VEE Implementation of Erlang

The VEE (Virding's Erlang Engine) implementation is similar to the JAM, but is based on a different virtual machine and compiler. A real-time, two space, copying garbage collector using a modified Baker scheme with Brook's optimisation is used. This gives better real-time performance than the JAM.

The VEE implementation is about twice as fast as the JAM implementation.

Supported versions of this implementation are available on

- SUN Workstations (both SunOS4 and Solaris 2)

The code emulated by the VEE (i.e. the output of the compiler) is still compact, but not as compact as in the JAM implementation.

4.3 BEAM Implementation of Erlang

In the BEAM [3] (Bogdan's Erlang Abstract Machine) implementation, Erlang is compiled (by a compiler written in Erlang) into C code. The C code is in turn compiled by the GCC, C-compiler and linked with a run-time kernel. The linker has been specially written for Erlang and allows incremental loading of code into running systems in the same way as the other implementations. Garbage collection is done in the same way as in the JAM implementation.

Supported versions of this implementation are available on:

- SUN Workstations (both SunOS4 and Solaris 2)
- VXWorks on Force 68040 and SPARC

This is the fastest implementation of Erlang, its speed is comparable with non optimised "C" code when executing sequential code. The code size is about five times larger than that of the JAM implementation.

5 Tools to Aid Erlang Programming

Apart from the compilers and run time system needed for the implementations described above, there are many tools available to the Erlang programmer. These include

Libraries:
: Trees, lists, strings, sets, code handling, error handler, error logger, expression evaluator, file system interface, global registration of processes, formatted io, mathematical functions, networking, expression parsers, process groups, load distribution, random numbers, and socket interfaces.

Interface to Foreign Languages:
: At present only an interface to C is supported. A library written in Erlang transforms Erlang terms to (and from) a binary representation. A library written in C can be used to transform *structs* in C to (and from) this binary representation.

Tools:
: Code coverage and profiling, windows based debuggers, make, pretty printer, shell, X-windows based point and shoot interface, parser generator, lexical analyser generator and Erlang mode for emacs.

Special Tools:
: Compiler for ASN.1. SDL to Erlang Compiler Interface to X-Windows.

6 Development of Erlang

Erlang was developed at Ellemtel, a jointly owned subsidiary of Ericsson and Telia (Swedish Telecom). Work with Erlang started with a series of experiments with the goal of determining how software design of large real-time systems could be made easier [4]. The main conclusion of this study was that so called declarative[3] languages resulted in the shortest and clearest programs. At that time we were very involved with Prolog and Parlog. In fact one of the earliest references to Erlang is the STRAND book [5]. Erlang has inherited syntax and data structures from Prolog.

[3]The term declarative is deliberately not defined here!

It soon became clear that many things in Prolog were unsuitable for our applications. Real-time systems, if they are to do anything useful, have side effects, for example in controlling hardware. Backtracking in Prolog must thus be severely restricted when changes have been made. As a simple example, in a telephone system we may cause a phone to ring. We cannot backtrack and cause the phone to stop ringing, someone may have heard it ringing!

Controlling hardware means that operations must be done in a predecided order. When Parlog is used, extra effort was required to ensure sequentiallity. Languages with unconstrained parallelism are too parallel.

Using logical variables would be extremely difficult to implement efficiently in a distributed system and would imply all sorts of hidden information passing between processes. Logical variables were thus excluded.

All the implementations we studied had some form of stop and copy garbage collection which were unsuitable in real-time systems.

It was also important to us that concurrency is explicit. In our applications we control a very large number of activities at the same time. For programs to be clear, it is important that each real life asynchronous activity be represented by a process in our system. It is also important that processes are light weigh. The computational effort in creating and scheduling processes and sending messages between processes should be small. The memory required by each process should be as small as possible and vary dynamically as the process expands and contacts.

Erlang was being developed and used at the same time as the application described in [6]. This application was a prototype of a new PABX (Private Automatic Branch Exchange). The designers of this prototype were mainly interested in the software architecture of their system and needed a new language which made it easy for them to perform experiments. The users working in this project developed several thousands of lines of code despite the fact that we were changing the language at the same time. These users contributed much to the design of Erlang. When we found that constructs in the language were not being used, we removed them, when we found that users were writing unclear or convoluted code we added constructs which alleviated their problems and made the code clear. This has resulted in a very small and compact language which we have found easy to use and to teach to new users.

7 Use of Erlang in Ericsson

For the first few years Erlang was mainly used to build prototypes. Ericsson has been involved in six projects in the European RACE programme where Erlang has been used to build demonstrators and prototypes. An example is BIPED [7].

Erlang has been used to build many prototypes internally in Ericsson. Notable among these are a private telephone exchange[6], an office cordless telephone system [8] and a prototype to demonstrate user mobility in telephone networks [9]. Erlang has been used to build countless other prototypes ranging from optical cross connect controllers to paging systems.

The success of these prototypes has resulted in Erlang being used as the implementation language for several products which are at present being developed. For commercial reasons the details of these products cannot be mentioned here.

The largest of these products, a complex and very specialised switching system, is being developed by a team of about 25 software engineers. So far about 300 000 lines of Erlang source code have been developed for this product, which is probably the largest and most complex functional program in the world today. It has been estimated that this would have required about three million lines of source code if this had been programmed in a conventional language - and would have required a much larger team.

It is important to note that using a language such as Erlang effects far more than the coding phase of systems development. The semantic gap between specifications and programs is much smaller than if languages such as C, C++ or PLEX are used. Testing also becomes easier, there is less code to test, the code is easy to understand and testing can be done on the symbolic "Erlang" level. The total work required to produce software, from first specifications to testing has, in some projects, been reduced by as much as an order of magnitude.

Erlang is thus spreading slowly into Ericsson products, starting with the newer less establish products. It has proved difficult to get Erlang used in older established products for which there already is a large volume of code written by an organisation which reflects a well defined methodology.

8 Use of Erlang Outside Ericsson

The JAM implementation is available free of charge, on a non commercial licensed basis to universities and similar organisations. About 80 systems have been delivered on this basis. Erlang is now being taught at several universities as far afield as Bejing, Melbourne and Stockholm.

Ericsson has decided to sell Erlang on a commercial basis. This task has been assigned to Ericsson Infocom Consultants AB. The Erlang Systems Division has been set up to support, maintain, develop and sell Erlang both within Ericsson and to external customers. A team of consultants is available to help customers with the use of Erlang.

Courses are held regularly in Sweden and abroad. These have been attended by both a large number of Ericsson employees and by people from other companies.

Enquires about obtaining Erlang, both for commercial and noncommercial use, should be address to Ericsson Infocom Consultants. AB, Erlang Systems division.

9 References

[1] Concurrent Programming in Erlang, Joe Armstrong, Mike Williams and Robert Virding, Prentice Hall, 1993.

[2] Implementing a functional language for highly parallel real time applications, Joe Armstrong, Bjarne Däcker, Mike Williams and Robert Virding, Software Engineering for Telecommunication Switching Systems and Services, March 30 - April 1, 1992, Florence.

[3] Turbo Erlang, Bogumil Hausman, International Logic Programming Symposium, October 26-29, 1993 Vancouver.

[4] Experiments with Programming Languages and Techniques for Telecommunication Applications, Bjarne Däcker, Nabiel Elshiewy, Per Hedeland, Carl Wilhelm Welin and Mike Williams, Software Engineering for Telecommunication Switching Systems, April 14-16 1986, Eindhoven.

[5] Chapter 3 Programming Telephony (Armstrong and Virding) from Strand - New Concepts in Parallel Programming Ian Forster and Stephen Taylor, Prentice Hall, 1990.

[6] New Technology for Prototyping New Services, Kerstin dling, Ericsson Review no. 2 1993

[7] Control Switching Implementation of the BIPED Demonstrator, F. Monfort Second Australian Conference on Telecommunications Software, Sydney 1993

[8] Prototyping Cordless Using Declarative Programming, Ingemar Ahlberg, John-Olof Bauner and Anders Danne, Ericsson Review no. 2 1993

[9] A Prototype Demonstrating User Mobility and Flexible Service Profiles, Jan van der Meer, Ericsson Review no 1 1994.

VDM Examples Repository

Peter Gorm Larsen
IFAD

November 15, 1994

1 Introduction

Below is a list of examples which can be freely obtained using Mosaic at the URL called `http://hermes.ifad.dk/examples/examples.html` or alternatively ftp'ed from the site `hermes.ifad.dk` in the directory pub/vdm/examples.

All the volunteers which have submitted these examples so far have also made the source text available. Thus, if you are interested in investigating the examples further you can play around with the source (they all come as a tarred file which need to be saved as a binary file and un-tarred by "tar -xvf") and if you have access to the IFAD VDM-SL Toolbox you can actually analyse the source texts further. All the postscript versions of the documents have been automatically generated from the source texts which have been submitted unless the providers have explicitly produced the postscript output. Those who have submitted plain ASCII specifications have been included in a simple skeleton . Thus, some places the line breaks could be improved.

- Specification of an ammunition control system.
  The Source file and a postscript version can be obtained.

  This specification describes the safety requirements involved in adding and removing explosives at an explosives storage site. The specification is based on United Kingdom Ministry of Defence regulations concerning safe storage of explosives, which in turn are based on UN regulations. Details of the specification may be found in:

  P. Mukherjee and V. Stavridou, "The Formal Specification of Safety Requirements for the Storage of Explosives", technical report DITC 185/91, National Physical Laboratory, 1991.

  P. Mukherjee and V. Stavridou, "The Formal Specification of Safety Requirements for Storing Explosives", Formal Aspects of Computing, 5(4):299-336, 1993.

- Railway Interlocking Systems.
  The source and postscript version of this specification will be made available by Kirsten Mark Hansen (Technical University of Denmark/ Danish State Railways (DSB)).

- Formal Semantics of Data Flow Diagrams.
  The Source file and a postscript version can be obtained.

  Data Flow Diagrams are used in Structured Analysis and are based on an abstract model for data flow transformations. This specification is a transformation from an abstract syntax representation of a data flow diagram into an abstract syntax representation of a VDM specification. Details can be found in:

P.G. Larsen, Nico Plat, and Hans Toetenel, "A Formal Semantics of Data Flow Diagrams", Formal Aspects of Computing, 1994, December.

- Specification of the Single Transferable Vote (STV) algorithm.
  The Source file and a postscript version can be obtained.

  STV is a scheme for performing elections, as advocated by the Electoral Reform Society. This specification formalizes the English language requirements and has been tested by animating the specification. Details may be found in:

  P. Mukherjee and B.A. Wichmann, "STV: A Case Study of the Use of VDM", Technical Report DITC 219/93, National Physical Laboratory, 1993.

  P. Mukherjee and B.A. Wichmann, "Single Transferable Vote, A Case Study of the Use of VDM-SL", Proc. The Mathematics of Dependable Systems, ed. C.J. Mitchell, Oxford University Press 1994.

- Specification of the MAA standard.
  The Source file and a postscript version can be obtained.

  The Message Autheticator Algorithm (MAA) standard is used in the area of data security in banking and the scope of the standard is authentication. More details can be found in:

  G.I. Parkin and G. O'Neill, "Specification of the MAA standard in VDM", In S. Prehn and W.J. Toetenel (eds), "VDM'91: Formal Software Development Methods", Springer-Verlag, October 1991.

- The Specification of a Binary Relational Database System (NDB) which will be made available by John Fitzgerald (Manchester University).

  The Non-programmer database system (NDB) is a nicely engineered binary relational database system invented by Norman Winterbottom of IBM. The formal Specification of NDB was originally undertaken by Anne Walshe, who has subsequently documented the specification and its refinement.

  NDB has been used as an example problem for *modular* specification in VDM-SL. The versions soon to be available here will include the basic "flat" specification and a structured form using the IFAD Toolbox's specification module syntax.

  Relevant publications are:

  A. Walshe, "NDB: The Formal Specification and Rigorous Design of a Single-User Database System", in C.B. Jones and R.C. Shaw (eds), "Case Studies in Systematic Software Development", Prentice Hall 1990, ISBN 0-13-116088-5.

  J.S. Fitzgerald and C.B. Jones, "Modularizing the Formal Description of a Database System", in D. Bjorner, C.A.R. Hoare and H. Langmaack (eds), VDM '90: VDM and Z – Formal Methods in Software Development, Springer-Verlag, LNCS 428, 1990.

- Denotational Semantics of the programming language NewSpeak.
  The Source file and a postscript version can be obtained.

  The programming language NewSpeak is a language designed specifically for use in safety-critical systems. It employs the notion of Orwellian programming – undesirable properties are avoided by restricting the syntax of the programming language. This is a formal semantics for the language in VDM-SL. Details of the language and its semantics:

P. Mukherjee, "A Semantics for NewSpeak in VDM-SL". In T. Denvir, M. Naftalin, M. Bertran (eds), "FME '94: Industrial Benefit of Formal Methods", Springer-Verlag, October 1994, to appear.

I.F. Currie, "NewSpeak - a reliable programming language". In C. Sennett (ed), "High-Integrity Software", Pitman 1989.

- Looseness Analysis Tool for a VDM-SL Subset.
  The Source files and a postscript version can be obtained.

  The specification language VDM-SL contains a notion of looseness. This is a specification of a looseness analysis tool which given a specification written in a subset of VDM-SL can determine which values a given VDM expression (using the definitions) may evaluate to in the different models for the specification. This illustrates how looseness (underdeterminedness) is combined with recursion in VDM-SL. Here the use of the test coverage facility of the IFAD VDM-SL Toolbox is illustrated. Details may be found in:

  P.G. Larsen, "Evaluation of Underdetermined Explicit Definitions" . In T. Denvir, M. Naftalin, M. Bertran (eds), "FME '94: Industrial Benefit of Formal Methods", Springer-Verlag, October 1994, to appear.

The VDM FAQ

Marcel Verhoef

The *vdm-forum* mailing list (announced in the Spring Issue of FACS Europe) is pleased to present a new Frequently Asked Questions file. Compiled from contributions by researchers and VDM users, the FAQ contains answers to basic technical questions and up-to-date information on the sources for the VDM Bibliography and ISO VDM-SL Standard.

The first edition will be published in September in *vdm-forum* and newsgroups including comp.specification, comp.specification.z and comp.software-eng.

The FAQ will be maintained by Marcel Verhoef (marcel@dutct05.tudelft.nl) and was compiled by John Fitzgerald (vdm-forum-request@mailbase.ac.uk). Topics currently covered include:

What is VDM?
What Modularity is provided in VDM?
Does VDM support object-orientation?
Can VDM handle concurrency?
Is VDM executable?
What is data reification?
How is VDM related to algebraic specification languages?
What is the difference between VDM and Z?
What is the relationship between VDM and RSL?
What is the relationship between VDM and the Refinement Calculus?
What is the semantics of VDM-SL?
Where do I start learning about VDM?
Where are the other VDMers?
Where can I find papers on VDM?
I heard there's a Standard for VDM ...?
Who is using VDM?
What sorts of system can VDM be used to describe?
What tools are available?
A short list of books on VDM

The FAQ will be posted to the forum on a monthly basis, starting from September 1994. LaTeX and hmtl versions will be announced.

Formal Methods Applications Database

Nico Plat

The industrial use of formal methods is limited and much of the experience gained with that industrial use is not available to the outside world. That is a pity, considering the fact that such experience can be very useful in avoiding the pitfalls one may encounter when starting to use formal methods in a 'serious' way. It is with this thought in the back of their heads that a group of people active in FME (*F*ormal *M*ethods *E*urope) have come up with the idea to start a database with examples of applications of formal methods, accessible through ftp and WWW on a public site. The intention of the database is to give concise descriptions of – either successful or unsuccessful – application of formal methods/specification languages that will allow users to assess:

- whether or not formal methods/specification languages are being used for large, industrial applications;
- what the difficulties/advantages are of applying formal methods/specification languages on a larger scale;
- for which application domains formal methods/specification languages are being used and which formal methods/specification languages are actually being used.

If you know of any applications of formal methods/specification languages worth considering for inclusion in the database (i.e. you have been involved yourself or you know someone who was involved with such an application) then it would be appreciated if you would fill in the form below and send it to the contact person.

Name of the application: Name of the application or a one-line description of it.

Developed by organisation/consortium: Main organisation(s) involved with the development of the application.

Formal method/specification language used: Self explanatory.

Tools used: Name of the tool used (if any).

Domain: A short characterisation of the application domain in which the formal method/specification language has been applied, e.g. 'medical systems', 'railway systems' or 'Air-traffic control (ATC) systems'.

Period: Period in time during which the application has been or will be developed. We think this is relevant information because as time passes, the time during which something has been developed can be used as an indication of the state of the art that has been applied.

Size: A rough indication of the 'size' of the application, in terms of lines of specification/source code of the implementation or in terms of human resources, i.e. the number of man-years involved.

Short description: A short (maximum 15 lines) description of the application and the way formal methods were applied on the application, e.g. did you apply formal proofs? Which mode of working did you use, i.e. separate specification teams/ implementation teams, did you consult a formal method 'guru', did you use formal methods/specification languages in conjunction with other software engineering techniques (e.g. Yourdon). Etc.

Conclusions: Any (subjective) observations on the application of formal methods/specification languages for your application. Do you feel that application of formal methods/specification languages for the case you describe was successful? If so/not, why/not? Etc.

Relevant publications: Adequate references to relevant publications, if any.

Contact: Name + address details (snail mail, phone, fax, e-mail) of a contact person who may be approached if a person wants to have more information about a case. Fill in 'unavailable' if you do not want to or can not be contacted for further information.

Further remarks: Any further remarks which you think may be relevant.

Leave any fields with information that you do not want to make public blank.

Please note that an electronic copy of the form can be obtained by sending an e-mail message containing only the line:

```
send vdm-forum fm-appl-db-form.txt
```

to `mailbase@mailbase.ac.uk`, and the form will be send back to you. Alternatively, you can pick it up on WWW at URL:

```
gopher://nisp.ncl.ac.uk/11/lists-u-z/vdm-forum/files
```

or by anonymous ftp from:

`ftp.ifad.dk` (130.225.136.3) (directory `pub/vdm`)

Please send your contributions to (e-mail preferred): Nico Plat, Cap Volmac, Dolderseweg 2, 3712 BP Huis ter Heide, The Netherlands. Fax: +31-3404-31174. E-mail: Nico.Plat@ACM.org.

Report on Z User Meeting (ZUM'94)

Jonathan Bowen
Oxford University
Computing Laboratory
Wolfson Building
Parks Road
Oxford OX1 3QD
Jonathan.Bowen@comlab.ox.ac.uk

Mike Hinchey
University of Cambridge
Computer Laboratory
New Museums Site
Pembroke Street
Cambridge CB2 3QG
Michael.Hinchey@cl.cam.ac.uk

ZUM'94, the 8th Z User Meeting, was held at St. John's College, University of Cambridge, during the last week of June 1994. In a break with tradition, there was no Z User Meeting in 1993. With an 18 month break between ZUM'92 and ZUM'94, the venue was greatly enhanced by the more clement British summer climate than that which had been endured during the previous Z User Meetings, in the previously regular December slot.

The meeting was organized by the Z User Group, in association with BCS-FACS, and kindly sponsored by BT, Logica Cambridge and Praxis, with support from the Commission of the European Communities through the ESPRIT **ProCoS-WG** Working Group. Jonathan Bowen (Oxford University) was the Conference Chair, Anthony Hall (Praxis) was Programme Chair, and Mike Hinchey (University of Cambridge) acted as Tutorial Chair and Local Organizer, with much support from Rosalind Barden (Logica).

With 140 delegates from 15 countries attending, the numbers were substantially increased from previous years, and the conference was expanded to fill an entire working week. Half-day and full-day tutorials on introductory Z, B and the B-method, project management issues, dependable real-time systems, and object-oriented specification in Z, were presented on Monday and Tuesday, 27th and 28th June, and proved to be very popular indeed. The main sessions were held on Wednesday and Thursday, 29th and 30th June, and were followed by a half-day session on educational issues of formal methods, held in the 12th century School of Pythagoras on the morning of July 1st. This provided an opportunity for educators and industrialists to discuss their ideas on the teaching of formal methods in general, and Z in particular. Four full papers were presented, and there were a number of poster displays.

Fortuitously, the location and timing of the meeting added a historical flavour to the event. Forty-five years previously to the week, the first ever European conference on computer science was held in Cambridge, in which the early EDSAC computer, designed and constructed by Prof. Maurice V. Wilkes and his team at the University of Cambridge, was discussed. This historic event was commemorated with a highly entertaining after-dinner speech at the ZUM'94 conference dinner by the organizer of the 1949 EDSAC meeting, Prof. Wilkes himself. The London Science Museum and the Whipple Museum of the History of Science very kindly organized a small display of some historical memorabilia, including the core memory of EDSAC-2, and a fragment of Babbage's Difference Engine.

During the main meeting, Mr. Robert Worden of Logica Cambridge gave the opening remarks, likening the system development process to "Fermenting and Distilling" and concluding with his belief in the benefits of formal methods and object-orientation to software engineering, when appropriately applied. Dr. Jim Woodcock of Oxford University presented "The Formal Specification in Z of Defence Standard 00-56", work that was performed by Formal Systems (Europe) Ltd. on behalf of the UK Ministry of Defence. Prof. David Garlan of Carnegie Mellon University described the approach taken at his university in "Integrating Formal Methods into a Professional Master of Software Engineering Program". Dr. Mike Gordon of University of Cambridge described an embedding of Z in HOL (Higher-Order Logic) as a support tool for formal development in Z. Dr. Leslie Lamport of Digital Systems Research Center, USA, described "TLZ", an extension to Z to make it akin to his own Temporal Logic of Actions (TLA), and hence suitable for use in the development of real-time and concurrent systems.

As well as these invited speakers, refereed papers on applications of formal methods, object-orientation, Z semantics, methods and concurrency were presented by speakers from the US, Australia and Europe. The published proceedings were issued at the meeting itself, an innovation which both saves overall time spent by the editors and which seemed to be appreciated by the attendees.

A session on standards was chaired by John Nicholls (Oxford University) including a status report of the proposed Z standard currently undergoing international standardization under ISO/IEC JTC1/SC22. George Cleland (University of Edinburgh) gave information on a survey undertaken for the UK government DTI (Department of Trade and Industry) to ascertain the best use of funding for the encouragement of the use of formal methods. Randolph Johnson (DoD, USA) reported on the activities of the ANSI X3J21 Technical Committee on FDTs (Formal Description Techniques) including Z and VDM, under the X3 Accredited Standards Committee on Information Processing Systems, which is overseeing the standardization of Z. It is also interested in future developments such as object-oriented extensions to Z and its possible standardization.

In addition to presentations, tools demonstrations, publishers' stands, and a poster session on Internet access to information relevant to Z, and formal methods in general, on the global on-line World-Wide Web hypermedia system were also available during the meeting. In particular, for on-line details of Z User Meetings and other Z-related information, see:

`http://www.comlab.ox.ac.uk/archive/z.html#meetings`

We were even blessed with good weather throughout the week in Cambridge.

The AGM of the Z User Group was held in conjunction with ZUM'94. At this meeting, ZUM'95, the 9th International Conference of Z Users, was announced. This will take place at University of Limerick, Ireland, 7–8 September 1995. Jonathan Bowen (Oxford University) will be the Conference Chair, Mike Hinchey (New Jersey Institute of Technology) will be the Programme Chair, and Norah Power (University of Limerick) will be the Tutorial Chair and Local Organizer. Once again the conference will be organized by the Z User Group in association with BCS-FACS, with kind sponsorship from BT, Forbairt, Praxis and University of Limerick, and support from **ProCoS-WG**. The invited speakers will be: Jean-Raymond Abrial (France), David Lorge Parnas (McMaster University, Canada), John Rushby (SRI International, USA) and Jeannette Wing (Carnegie Mellon University, USA).

Recent books column

Cliff Jones

November 25, 1994

I agreed to produce listings of books which relate to the purpose of this newsletter. Authors should send references (BibTeX format preferred) to `cbj@cs.man.ac.uk`.

Authored books: [Plü90, Dev90, And91, FB94, Mit94, Pif91, FJM94, JL92, Ban93, Ban94]

Edited book: [Bow94, Lau93]

Proceedings: [BH94, Old94]

References

[And91] James H. Andrews. *Logic Programming: Operational Semantics and Proof Theory.* Distinguished Dissertation Series. Cambridge University Press, March 1991.

[Ban93] U. Banerjee. *Loop Transformations for Restructuring Compilers: The Foundations.* Kluwer Academic Publishers, 1993.

[Ban94] U. Banerjee. *Loop Transformations for Restructuring Compilers: Loop Parallelization.* Kluwer Academic Publishers, 1994.

[BH94] J. P. Bowen and J. A. Hall, editors. *Z User Workshop, Cambridge 1994*, Workshops in Computing. Springer-Verlag, 1994.

[Bow94] J.P. Bowen, editor. *Towards Verified Systems.* Real-Time Safety Critical Systems Series. Elsevier Science Publishers, 1994.

[Dev90] Yves Deville. *Logic Programming: Systematic Program Development.* Addison-Wesley, Wokingham, England, 1990.

[FB94] R. Floyd and R. Beigel. *The Language of Machines: An Introduction to Computability and Formal Languages.* New York: Computer Science Pr., 1994. ISBN 0-7167-8266-9.

[FJM94] Loe M. G. Fiejs, Hans B. M. Jonkers, and Cornelius A. Middelburg. *Notations for Software Design.* Springer-Verlag, 1994. ISBN 3-540-19902-0.

[JL92] R. Janicki and P. E. Lauer. *Specification and Analysis of Concurrent Systems.* EATCS Monographs on Theoretical Computer Science. Springer-Verlag, 1992.

[Lau93] P. E. Lauer, editor. *Functional Programming, Concurrency, Simulation and Automated Reasoning.* Lecture Notes in Computer Science 693. Springer-Verlag, 1993.

[Mit94] Richard Mitchell. *Abstract Data Types and Modula-2.* Prentice-Hall, 1994.

[Old94] E.-R. Olderog, editor. *Programming Concepts, Methods and Calculi.* North-Holland, 1994.

[Pif91] Mike Piff. *Discrete Mathematics: an introduction for software engineers.* Cambridge University Press, 1991. ISBN 0-521-38622-5.

[Plü90] Lutz Plümer. *Termination Proofs for Logic Programs*, volume 446 of *Lecture Notes in Artificial Intelligence.* Springer-Verlag, Berlin, 1990.

FORTHCOMING EVENTS

1995

December 15–17, 1994
Fourteenth Conf. on the Foundations of Software Tech. and Theoretical Comp. Sci., Madras, India. *Contact:* R Ramanujam, FST&TCS 14, Inst. of Mathematical Sci.s, C I Campus, Taramani, Madras 600 113, India Email: jam@imsc.ernet.in

January 3–6
Parallel and Distributed Computing: Theory, Systems and Applications, Maui, Hawaii *Contact:* H El-Rewini, Dept. of Comp. Sci., Univ. of Nebraska at Omaha, Ohmaha, NE 68182, USA. Email: rewini@unomaha.edu **Or** B Shriver, HICSS-28, 17 Bethea Drive, Ossining, NY 105620, USA. Email: b.shriver@computer.org

January 23–25
ACM SIGPLAN-SIGACT Symp. on Principles of Prog. Lang., San Fransico, California. *Contact:* (Program Chair) Peter Lee, Carnegie Mellon Univ., School of Comp. Sci., 8119 Wean Hall, 5000 Forbes Avenue, Pittsburg, PA 15213 Email: petel@cs.cmu.edu

April 5–8 WIFT'95,
Workshop on Industrial-strength Formal specification Techniques, Boca Raton, Florida USA. Submissions to Program Chair **Or** appropriate Regional Coordinator; information from Organizing co-chairs PROGRAM CHAIR: Susan Gerhart, RICIS, Univ. of Houston-Clear Lake, Houston, TX 77058, USA, Tel: +1-713-283-3800 Fax: +1-713-283-3810 Email: gerhart@cl.uh.edu EUROPEAN COORDINATOR: Tom Docker, CITI Ltd, Challenge House, Sherwood Dr, Bletchley MK6 3DP, UK, Tel: +44-908-377800, Fax: +44-908-371257 Email: 100121.2156@compuserve.com ASIAN COORDINATOR: T. H. Tse, Dept. of Comp. Sci., Univ. of Hong Kong, Pokfulam Rd, Hong Kong, Tel: +852-859-2183, Fax: +852-559-8447 Email: tse@csd.hku.hk ORGANIZING CO-CHAIRS: Robert France, Maria Larrondo-Petrie, Dept of Comp. Sci. & Eng., Florida Atlantic Univ., Boca Raton, FL 33431-0991, USA, Tel: (Robert): +1-407-367-3857, Tel: (Maria): +1-407-367-3899, Fax: +1-407-367-2800, email: robert@cse.fau.edu **Or** maria@cse.fau.edu Information can also be obtained via anonymous ftp to shark.cse.fau.edu (/pub/WIFT95/Info).

July 10-14 ICALP'95
22nd Int'l. Coll. on Automata, Lang. and Prog., Szged, Hungary. *Contact:* F Gecseg, J Csirik, Attila Jozesef Univ., Dept. of Comp. Sci., Aradi Vetanuk tere 1, H-6720 Szeged, Hungary Email: icalp95@inf.jate.u-szeged.hu

July 17–21 MPC '95
Third Int'l. Conf. on the MATHEMATICS OF PROGRAM CONSTRUCTION, Kloster Irsee, Germany *Submission date:* 1 December 1994. *Contact:* Prof Dr B Moller (MPC '95), Institut fur Mathematik, Universitat Augsburg, D-86135 Augsburg, Germany Fax: +49 821 598 2200 E-mail: moeller@uni-augsburg.de

September 7–8 ZUM '95
9th Int'l. Conf. for Z Users, Limerick, Ireland. Supported by BCS FACS and the ESPRIT ProCoS-WG Working Group. Sponsored by BT and Praxis. Organized by the Z User Group. *Submission date:* 9 December 1994. *Contact:* Jonathan Bowen (Conf. Chair), Oxford Univ Computing Lab., Wolfson Building, Parks Road, Oxford OX1 3QD, UK. Tel: +44-1865-283512 (direct) 283521 (secretary) Fax: +44-1865-273839 E-mail: Jonathan.Bowen@comlab.ox.ac.uk

British Computer Society

BCS - FACS

Formal Aspects of Computing Science

Please reply to: Dr D J Cooke
Department of Computer Studies
Loughborough University of Technology
Loughborough
LE11 3TU
UK

tel: +44 (0)1509 222676
fax:+44 (0)1509 211586
email: D.J.Cooke@lut.ac.uk

16 December 1994

Dear FACS member

Enclosed you will find your copy of the latest issue of FACS Europe and a membership renewal form for 1995. The forms are usually sent out much earlier but we thought it only right to delay sending them until you received the next issue of the newsletter. Also enclosed are forms for renewing subscriptions to EATCS and to the FACS journal.

The journal goes from strength to strength. Again this year (as in 1992) there is a bonus issue - there are 7 "books" - plus 2 extra papers available via ftp. Also, at long last, we are making considerable progress in persuading authors to write more concisely and hence there are more papers - on more diverse topics - in each issue; issue 6(6) contains 8 papers. The journal price to FACS members remains as it was for 1994. Indeed, all the fees remain unchanged.

As many of you will know, we are starting to use email as a vehicle for broadcasting FACS announcements. Please help us to avoid unnecessary email traffic by ensuring that we have your correct email address; several addresses we had on file proved to be incorrect.

Yours sincerely

John Cooke

Dr D J Cooke
Membership Secretary

THE INSTITUTE OF MATHEMATICS AND ITS APPLICATIONS

2ND CONFERENCE ON THE MATHEMATICS OF DEPENDABLE SYSTEMS (MDS 95)

4-6 September 1995

University of York, England

ANNOUNCEMENT AND CALL FOR PAPERS

The construction of dependable systems, by which we mean systems providing high levels of reliability, availability, safety and/or security, is a problem of considerable concern to both providers and users of information processing systems of all types. Historically, different aspects of system dependability (e.g. reliability and security) have been studied quite independently, albeit that many of the goals are similar. For example, the notion of certifying functionality assurance levels applies equally to reliable systems and secure systems. In addition, users will often require some combination of security and fail-safe operation.

The purpose of MDS 95 is to consider the mathematical aspects of the provision of dependable systems, one goal being a comparison and possible unification of mathematical techniques for providing safe, reliable and secure systems. A number of different mathematical approaches have been taken to the overall problem, including probabilistic/statistical reasoning, formal models of safe, secure and reliable systems and logics of authentication and access control/privilege delegation. Papers on all these areas are solicited, the unifying theme being the application of mathematical techniques to the overall dependability problem. Hence papers will be particularly welcome which cross-fertilise the application domains. The conference will consider dependability for both hardware and software systems.

Programme and Proceedings: The conference will consist of three days of presentations by contributing authors. The programme will also include invited lectures by prominent researchers and practitioners in dependable systems theory and practice. Time will be made available for discussions. A digest of papers will be available to participants during the meeting and the proceedings will be published after the conference.

Invited Speakers: Monsieur P Chapront (GEC Alsthom, France), Professor J Knight (University of Virginia, USA), Professor B Littlewood (City University, UK), Professor D L Parnas (McMaster University, Canada), Professor F Piper (Royal Holloway, University of London, UK) and Dr C T Sennett (Defence Research Agency).

Submissions: Five copies of complete papers (in English) should be sent to Mrs Pamela Bye, Conference Officer, The Institute of Mathematics and its Applications, 16 Nelson

Street, Southend-on-Sea, Essex, SS1 1EF, England (Tel. +44 702 354020, Fax +44 702 354111, Email imacrh@v-e.anglia.ac.uk) by 31st March 1995. Authors will be notified of the outcome of their submission by 26th May 1995 and will be sent the required style files. Revised manuscripts will be due by 14th July 1995. Papers must not exceed 6,000 words in length (with full-page figures counted as 300 words). Each paper should include a short abstract and a list of keywords for subject classification. All papers will be refereed and the final choice will be made by the programme committee on the grounds of relevance, significance, originality, correctness and clarity. Submitted papers must not be published or be under consideration for publication elsewhere in the same or similar form.

Programme Committee: Programme Chair: V Stavridou (Royal Holloway, University of London), D Gollmann (Royal Holloway, University of London), M Ingleby (British Rail Research), J Jacob (University of York), N Jefferies (Vodafone Ltd), B Littlewood (City University), R Shaw (Lloyd's Register), B Wichmann (National Physical Laboratory).

Location: The conference will be held at the University of York, a modern campus built around a lake with excellent facilities. The campus is two kilometers from the centre of the medieval walled city of York, and 60 kilometers from the North Yorkshire coast. The city is also well placed for the Pennines and Yorkshire Wolds. York is very well connected by rail to London, Edinburgh and Manchester and the nearest airports are Manchester and Leeds/Bradford.

Guidelines for Newsletter Contributions

Contributions may be in the form of **single-sided** camera-ready copy, suitable for layout and sub-editing. They can also be sent to us using electronic media (i.e. by floppy disk (MS DOS or Mac)/e-mail/etc.), to be formatted in the house style. As a rule, we generally accept pure ASCII text or TEX/LATEX in order to avoid complications involving interchange between wordprocessing formats. We regret that we are unable to offer typesetting facilities for handwritten material.

If contributions are sent using proprietary wordprocessor/markup language formats (i.e. MicroSoft Word 5, FrameMaker), then these will be treated as though they were camera-ready copy. If we are unable to print them adequately or to otherwise convert to another more suitable form then the authors may be asked to provide paper copies of appropriate reproduction quality.

Artwork can be provided for appropriate inclusion, either using general formats (such as DVI files or Encapsulated PostScript) by sending camera-ready paper copy. Generally, line drawings and other high-contrast graphical diagrams will be acceptable.

Material must be of adequate quality for reproduction. Output from high quality printers with at least 300 DPI resolution is generally acceptable. Output from printers with lesser resolution (i.e. dot-matrix printers) tends not to reproduce very well and will not be of sufficiently good print quality. The Editorial Panel reserves the right to refuse publication for contributions which cannot be reproduced adequately.

Page definition information

If possible, contributions should be designed to fit standard A4 paper size, leaving a margin of at least one inch (1") on all sides. Camera ready copy should be sent in **single-sided** format, with page numbers written lightly **on the back**. Ideally, all fount sizes used should be no smaller than 10pt for clarity. Contributions should attempt to make adequate use of the space, filling at least 60% of each page, including the last one. Authors should note that all contributions may be sub-edited appropriately to make efficient use of space.

Deadlines

The production deadlines for the coming year are:

Spring end of February	**Summer** end of May
Autumn end of September	**Winter** end of November

Disclaimer

The views and opinions expressed within articles included in the FACS Europe newsletter are the responsibility of the authors concerned and do not necessarily represent the opinions or views of the editorial panel.

Addresses

Editors:
Dr. Jawed Siddiqi & Dr. Chris Roast
Computing Research Centre
Dept. of Computing and Management Sciences
Sheffield Hallam University
100 Napier Street
Sheffield, S11 8HD
United Kingdom

Tel: +44 742 533141
E-mail: `J.I.Siddiqi@shu.ac.uk` and `C.R.Roast@shu.ac.uk`

BCS FACS Committee 94-95

General

General enquiries about the BCS FACS group, the newsletter or its meetings can be made to:

BCS FACS
Department of Computer Studies
Loughborough University of Technology
Loughborough, Leicestershire
LE11 3TU
Tel: +44 509 222676
Fax: +44 509 211586
E-mail: FACS@lut.ac.uk

Membership fees 1994

Standard (i.e. non-BCS members) : £25
BCS members : £10

Discount subscription rates 1994

EATCS : £10
FACS Journal : £35 (6 issues, Vol. 6)

FACS Officers

Chair	Tim Denvir
Treasurer	Roger Stone
Committee Secretary	Roger Carsley
Membership Secretary	John Cooke
Newsletter Editors	Jawed Siddiqi & Chris Roast
Liaison with BCS	Margaret West
Liaison with FACS Journal	John Cooke

FACS Committee Members

Name	Affiliation	Tel:	E-mail
D. Blyth	Incord Ltd.	0202-896834	DBlyth@cix.compulink.co.uk
J. Boarder	Buckinghamshire	0494-22141	jcb@buckscol.ac.uk
R.E. Carsley	Westminster	071-911-5000x3568	roger@westminster.ac.uk
D.J. Cooke	Loughborough	0509-222676	D.J.Cooke@lut.ac.uk
B.T. Denvir	Translimina Ltd.	081-882-5853	timdenvir@cix.compulink.co.uk
S.J. Goldsack	Imperial	071-589-5111x5014	sig@doc.ic.ac.uk
A.J.J. Dick	Bull	0442-884586	J.Dick@brno.uk03.bull.co.uk
R.B. Jones	ICL Winnersh	0734-693131x6536	
R.J. Mitchell	Brighton	0273-642458	rjm4@unix.brighton.ac.uk
B.Q. Monahan	Manchester	061-275-6137	brianm@cs.man.ac.uk
M.P. Naftalin	Lloyd's Register	081-681-4040	tcsmpn@aie.lreg.co.uk
C.R.Roast	Sheffield Hallam	0742-533141	C.R.Roast@shu.ac.uk
J.I.A. Siddiqi	Sheffield Hallam	0742-533141	J.I.Siddiqi@shu.ac.uk
D. Simpson	Brighton	0273-600900x2450	ds33@unix.bton.ac.uk
R.G. Stone	Loughborough	0509-222686	R.G.Stone@lut.ac.uk
D.R. Till	City	071-477-8552	till@cs.city.ac.uk
M.M. West	Leeds	0532-335430	mmwest@scs.leeds.ac.uk
A. Wrightson	Huddersfield	0484-472758	scomaw@zeus.hud.ac.uk