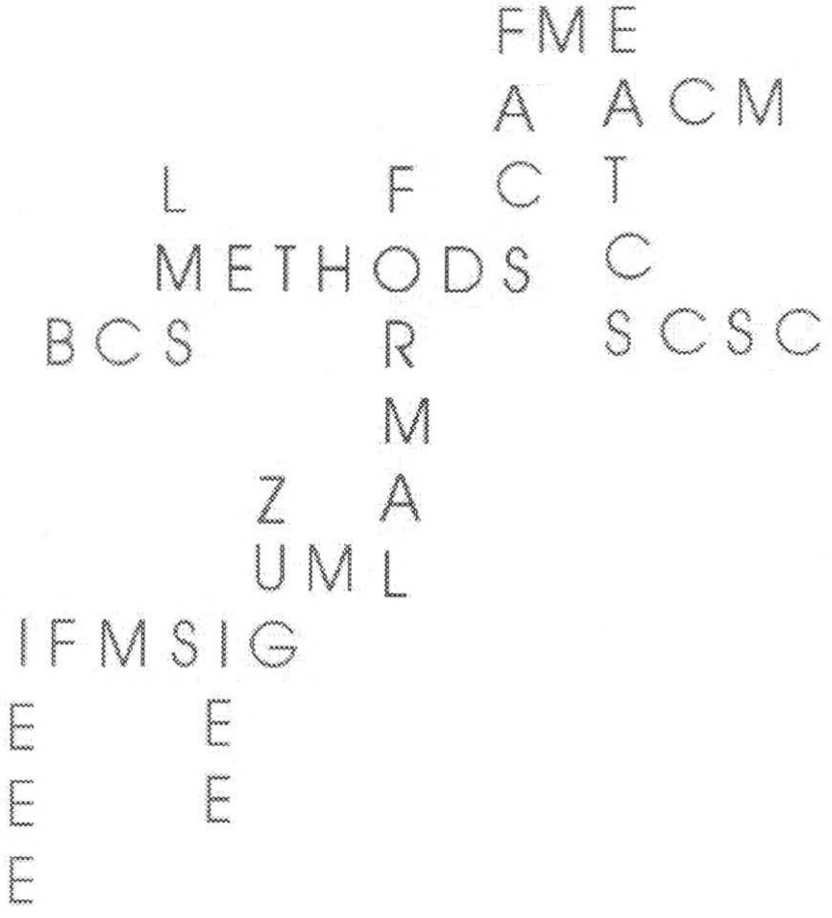
Issue 2005-1
March 2005

FACS
A
C
T
S



BCS FACS

The Newsletter of the Formal Aspects of
Computing Science (FACS) Specialist Group

ISSN 0950-1231

About *FACS FACTS*

FACS FACTS [ISSN: 0950-1231] is the newsletter of the BCS Specialist Group on Formal Aspects of Computing Science (FACS). *FACS FACTS* is distributed in electronic form to all FACS members.

As from 2005, *FACS FACTS* will be published four times a year: **March**, **June**, **September** and **December**. Submissions are always welcome. Please see the advert on page 37 for further details or visit the newsletter area of the FACS website [http://www.bcs-facs.org/newsletter].

Back issues of *FACS FACTS* are available to download from:

http://www.bcs-facs.org/newsletter/facsfactsarchive.html

The *FACS FACTS* Team

Newsletter Editor Paul Boca [editor@facsfacts.info]

Editorial Team Jonathan Bowen, Judith Carlton, John Cooke, Kevin Lano, Mike Stannett

Columnists Dines Bjørner (The Railway Domain)
Adrian Hilton (RefineNet)

Contributors to this Issue:

Rob Arthan, Dines Bjørner, Paul Boca, Jonathan Bowen, Judith Carlton, John Derrick, John Fitzgerald, Stephen Gilmore, Jane Hillston, Adrian Hilton, Roger Bishop Jones, Kevin Lano, Greg Michaelson, Teresa Numerico, Monika Seisenberger, Helen Treharne, Emil Sekerinski, Marcel Verhoef

Contents

Editorial

Paul Boca, Newsletter Editor & Jonathan Bowen, BCS-FACS Chair

Welcome to the first issue of *FACS FACTS* in 2005. We would like to begin by apologizing to Professor Steve Schneider (University of Surrey) for inadvertently omitting details of his talk from the report on *25 Years of CSP* in Issue 2004-3 of *FACS FACTS*. Steve's talk on *Verifying Security Protocols: an application of CSP* was both engaging and informative, and his participation helped make the event a success.

FACS ended 2004 with a Christmas meeting on the *Verified Software Repository*, part of the Grand Challenge 6 effort. The meeting, which was held at the new BCS London offices, was a great success and a full report can be found in this issue. A report on the *Program Verification and Semantics* event held at the Science Museum in London is included too, as are reports on other meetings.

This issue of *FACS FACTS* has attracted two technical articles. The first is written by Jane Hillston and Stephen Gilmore, both from the University of Edinburgh, on *Formal Aspects of Performance Modelling*. Jane Hillston was the recipient of the first BCS Roger Needham Award in 2004. This award, sponsored by Microsoft Research Cambridge and established in memory of the late Roger Needham of Cambridge, is for a distinguished research contribution in computer science by a UK-based researcher within ten years of their PhD. Jane won the award for her work on PEPA (Performance Evaluation Process Algebra) and compositional approaches to performance modelling. The winner of the award also has the opportunity to give a public lecture at the Royal Society, London. Jane's lecture, to a packed out audience, took place on 8 December 2004. Congratulations to Jane on her achievement.

The second technical article in this issue is provided by Rob Arthan (of Lemma One, Reading) and Roger Bishop Jones. It describes the Z proof tool, **ProofPower**, in particular, *Z in HOL in* ***ProofPower***.

This issue welcomes a new columnist too: Dines Bjørner (Technical University of Denmark, and currently on sabbatical at the University of Singapore). Dines will be reporting on *TRain: The Railway Domain*. Dines joins Adrian Hilton (Praxis High-Integrity Systems, Bath) who is reporting regularly on the UK government funded EPSRC *RefineNet* Network. A report from Adrian Hilton appears in this issue.

FACS is planning to organize some interesting events in 2005. Discussions are already underway for the 2005 Christmas Meeting; details will be announced in due course on the FACS website and mailing list. Paul Boca, Jonathan Bowen and Jawed Siddiqi are currently engaged in organizing a series of evening seminars to be held at the BCS offices in London. Further details can be found in this issue.

Finally, all BCS-FACS members are very welcome to submit to *FACS FACTS*. Please do send items for possible inclusion, whether technical or non-technical, to the newsletter editor at any time. We hope you enjoy this issue and can contribute in the future! ■

FACS Away Day

Paul Boca & Jonathan Bowen

On Saturday 23 October 2004, the FACS committee met in London (at the Union Jack Club near Waterloo Station) to discuss the future of the group. There were two main reasons for holding this meeting: to define a new mission statement, as the current one was out of date and no longer represented the group accurately, and to make progress towards drawing up a five-year plan for FACS. Lively discussions took place and our expert facilitator Jawed Siddiqi kept us on track and ensured everyone had the opportunity to put their points forward. This report summarizes the main decisions made during the eight hour meeting.

We set about formulating the mission statement with two brainstorming sessions. In the first one we individually selected the areas on which, in our individual opinions, FACS should focus. These were discussed in detail as a group and commonalities between them identified. In the second brainstorming session, each committee member put forward a list of the objectives that they felt FACS should be trying to achieve. Once again, these were discussed (in great detail) as a group. The results from both sessions were debated and, after much deliberation and redrafting, we agreed on the following new mission statement for FACS:

To promote the awareness, development and application of:

- *a mathematical basis for computer science;*
- *theories underpinning practice in computing;*
- *rigorous approaches to information processing in computer-based systems.*

The committee felt that this new Mission Statement more accurately reflected the aims of the group and encompassed a wider selection of areas than the existing statement (on the website).

Progress was made regarding drawing up a five-year plan for FACS too. The committee agreed, after a substantial debate, that in order to take the group forward, effort would need to be focused on the following:

- Encouraging our community [Paul Boca]
- Facilitating knowledge exchange between academia and industry [Judith Carlton]
- Supporting education [Mike Stannett]
- Promoting dissemination [Jonathan Bowen]

- Providing representation to government [Jawed Siddiqi]
- Evolving the discipline [John Cooke]

A committee member has been assigned to oversee each of these efforts, as noted above. Some progress has already been made on *encouraging our community*: Paul Boca, Jonathan Bowen and Jawed Siddiqi are currently engaged in organizing a series of **free evening seminars**, to be held at the BCS London offices. Four seminars will be given in the series by Dines Bjørner (Technical University of Denmark, and currently on sabbatical at the National University of Singapore), Muffy Calder (University of Glasgow), Steve Reeves (University of Waikato, New Zealand) and Jim Woodcock (University of York). Full details will be announced in due course on the BCS-FACS mailing list and website [http://www.bcs-facs.org/events/EveningSeminars]. Progress on the other efforts will be reported in future issues of *FACS FACTS*.

Ultimately FACS exists to serve the interests of its members. The committee would therefore welcome feedback on the decisions taken at the meeting; comments may be emailed to info@bcs-facs.org.uk. FACS plans to organize another away day in eighteen months time to revisit the ideas discussed at this meeting, review progress and determine what it has to do to serve the formal methods community more effectively.

Joining Other Societies and Groups

London Mathematical Society
http://www.lms.ac.uk/contact/membership.html

Safety-Critical Systems Club
http://www.safety-club.org.uk/membership.php

Formal Methods Europe
http://www.fmeurope.org/fme/member.htm

European Association for Theoretical Computer Science
http://www.eatcs.org/howtojoin.html

The 5th Symposium on Trends in Functional Programming, Munich

Greg Michaelson

For its fifth instantiation, Trends in Functional Programming (TFP) ventured beyond Scotland to enjoy the hospitality of Hans-Wolfgang Loidl and his colleagues at Ludwig Maximillian University, Munich, Germany, 22–26 November 2004 [http://www.tcs.informatik.uni-muenchen.de/~hwloidl/TFP04]. This lively Symposium was attended by 35 international participants spanning Austria, Brazil, England, Estonia, France, Germany, Hungary, Italy, Netherlands, Scotland, Spain and Turkey.

At TFP, all participants are encouraged to present papers, enabling first airings of evolving ideas and providing a critical but supportive forum for PhD students to engage with more experienced peers. At TFP'04, 27 papers were presented in two days. As typifies TFP, almost all the papers had strong explicit formal underpinnings. However, I think that the following will be of particular interest to BCS-FACS readers as they exemplify the use of formal techniques in establishing properties of programs:

- G. Hutton & J. Wright, *Calculating an Exceptional Machine*: uses transformation techniques to successively refine an abstract machine from the formal definition of a simple language.
- C. Dubois, T. Hardin, V.Viguie & D. Gouge, *FOCAL: an Environment for Developing Certified Components*: presents a language framework for constructing certified software from specification to implementation with tools that generate COQ code for automatic verification.
- N.Popov & T. Jebelean, *Verification of Functional Programs in Theorema*: discusses the development of a system for generating verification conditions for recursive programs.
- G. Grov, A. Ireland & G. Michaelson, *Model Checking HW-Hume*: considers the compilation of Hardware-Hume programs to Promela for model checking.
- L.Lensink & M. van Eekelen, *Induction and Co-Induction in Sparkle*: extends the Sparkle proof assistant for Clean with proof techniques for mutually recursive functions and data types.
- T. Altenkirch & J. Grattage: *a Functional Quantum Programming Language*; introduces the QML language for quantum computations on finite types, based on categorical semantics.

The refereed proceedings will be published later in 2005 by Intellect as "Trends in Functional Programming 5", edited by Hans-Wolfgang Loidl.

The Symposium was enlivened by excellent food, a modicum of weissbier and glühwein, and the banquet held in the Augustinerkeller.

TFP'05 will be held in Estonia and TFP'06 in Nottingham. See the TFP website [http://www.tifp.org] for further details. ■

FACS Christmas Meeting: The Verified Software Repository

Paul Boca & Jonathan Bowen

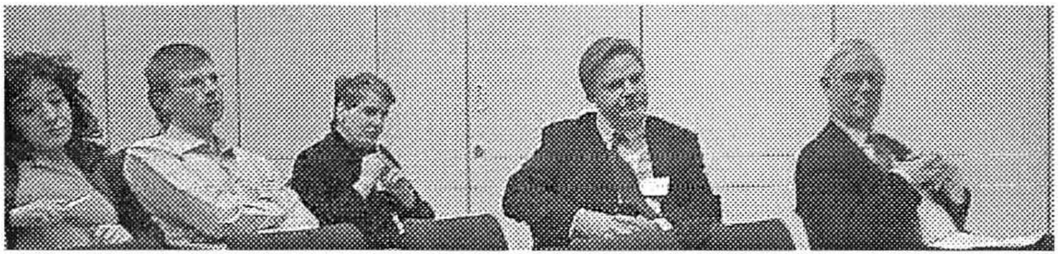
On Tuesday 21 December 2004, the BCS-FACS group held a one-day meeting on a proposed *Verified Software Repository*, to store verification tools and challenging case studies. This effort is part of the Grand Challenge 6 initiative in the area of *Dependable Systems Evolution* [http://www.fmnet.info/gc6]. Seven Grand Challenges for Computing Research, in the style of Hilbert, have been proposed [http://www.nesc.ac.uk/esi/events/Grand_Challenges], and this meeting provided a small initial step towards part of one of these. For further information, see a recently published report, edited by Tony Hoare and Robin Milner, summarizing the progress on all seven Grand Challenges in computing, available for download from the BCS website [http://www.bcs.org/NR/rdonlyres/CFC7C803-DBA74253-85E712AF5B984721/0/gcresearch.pdf].

The event was held at the excellent new British Computer Society (BCS) [http://www.bcs.org] offices in central London, near Covent Garden. Forty-three people attended the event; most were based in the UK, but some delegates had travelled from further afield (for example, from Australia, Canada, and Japan). This mix of delegates, with different backgrounds and experiences, gave rise to fruitful discussions both during and after the talks. The newly appointed BCS President, David Morriss, was able to attend for part of the day too and welcomed delegates to the new BCS venue. This was fitting, as the BCS is a major supporter of the Grand Challenge programme.

In this report, we give a summary of the six talks that made up the programme. The interested reader may download the speakers' slides from the event website [http://www.bcs-facs.org/events/xmas2004]. A selection of photographs taken at the event by Pierre-Yves Schobbens and Jonathan Bowen is available on the event website too, as are two papers on the Verified Software Repository.

Morning Session

The first session, chaired by Jonathan Bowen, began with a motivational talk by **Tony Hoare (Microsoft Research)** on the verifying compiler and the need for a verified software repository. Hoare explained that a verifying compiler is a tool that uses theorem-proving technology to prove the correctness of programs *before* they are executed. Correctness is specified with annotations such as type information, assertions and specifications, which accompany the programs to be compiled. Such a tool would reduce the number of avoidable programmer errors occurring in software systems, saving



From left to right: Teresa Numerico, Paul Boca, Fiona Polack, Colin O'Halloran, David Morriss

between 22 Million and US$60 Million per year (according to US Department Commerce Planning Report 02-03, May 2002).

The verifying compiler problem has been studied for over 30 years, with notable contributions made by Turing in the 1950s (with the introduction of assertions), Floyd in 1967 (suggesting that a verifying compiler could check assertions with a theorem prover), Dijkstra in 1968 (suggesting that assertions are written before code) and Hoare himself in 1969 (with the use of axioms). Previous attempts to make progress towards a practical verifying compiler have failed for reasons that are now well understood. Technology and theorem-proving capabilities have moved on significantly since those early days, and so Hoare believes that the verifying compiler is now achievable (this century at least!). There is still a great deal of work to be carried out and the timescale set for the project is 15 years.

Hoare explained that a first step towards producing a verified compiler was to set up a *verified software repository*. The intention is for the repository to contain a number of tools, such as program analysers, theorem provers, model checkers, and so-called *challenge codes*. The challenge codes are intended to be case studies of varying sizes, expressed in different languages, such as programs, specifications and requirements on which to run these tools. The logistics of how this repository would be set up was explained in the final talk, given by Juan Bicarregui.

Hoare stressed that the success of the project would rely on various people working together; in particular, it would require involvement of program theorists, programming tool-set builders, compiler writers and optimisers, design pattern architects, sympathetic users to test the assertions, open-source code contributors, developers of proof tools and model checkers, and teachers and students who will carry out a significant amount of work.

Colin O'Halloran (QinetiQ) was the first speaker given the opportunity to rise to the Grand Challenge laid down by Hoare. O'Halloran's experience is that most errors in software development are due to issues in the requirements or the design. It is at these stages that formal methods can be most effective. This was illustrated with the non-trivial case study of the Eurofighter Typhoon. The requirements for the Eurofighter were specified in Fortran – for historical reasons rather than by choice! – and turned into Simulink diagrams. The ClawZ tool was then used to help transform the Simulink diagrams into formal Z notation, later to be refined

Delegates during one of the sessions

to an implementation using the SPARK Ada programming language (see Rod Chapman's talk).

The correctness of the resulting Ada was verified with varying degrees of success, due to limitations of the verification tools themselves, proof classification problems, specification mismatches and in one case lack of time. The Flight Control Law was the most successful of the three different Simulink specifications (the other two were Autothrottle and Autopilot), verified with 97% of its verification conditions discharged automatically. About 80% of the verification conditions in the other two cases were automatically discharged. The likely reason for the lower percentage is that the specifications were not as mature as the specification for Flight Control Law.

Michael Butler (University of Southampton) brought the morning session to a close with a talk surveying applications using and tools for the B formal development approach. A number of such tools are available, having different functionalities and uses. Butler focused on Atelier-B (from ClearSy, France), the B-toolkit (from B-Core, UK), the BZ Testing Tool and the Southampton "home-grown" UML/B, U2B and ProB model checker.

An impressive list of applications developed using the B approach was then discussed. The driverless metro for the Paris Charles De Gaulle airport contains 180,000 lines of B (specifications and refinement), 140,000 lines of Ada and 43,000 proof obligations. All of these obligations have been proved, 98% of them automatically. Other examples were given, including work in progress at Peugeot where the specification contains 350,000 lines of B.

Butler cited the European collaborative MATISSE project (Methodologies and Technologies for Industrial Strength Systems Engineering) [http://www.matisse.qinetiq.com], which finished in 2004, as a good source of applications in B. The application domains studied in this project were railways, smartcards and healthcare.

The new EC-funded RODIN (Rigorous Open Development for Complex Systems) project [http://rodin.cs.ncl.ac.uk], which began September 2004 and will run until 2007, will have a large B element too. Some software will be formally specified using B, and some parts will be formally verified. For example, the model checker ProB will be verified in B. Butler explained that several case studies are planned: position calculation for 3G phones, engine failure management, mobile internet application, air traffic display and ambient campus. The formal developments will be made publicly available in due course. For more information on RODIN, please visit the website above, or see the article in issue 2004-3 of the *FACS FACTS* newsletter [http://www.bcs-facs.org/newsletter/facts200411.pdf]. There is a workshop on RODIN at FM'05 in July 2005, which some readers may be interested in attending. Further details can be found on the FM'05 event website [http://www.csr.ncl.ac.uk/fm05/main_workshops.php?mode=info&language=english&workshop=10].

Afternoon Session

Jim Woodcock chaired the afternoon session. The first talk by **Tom Melham (University of Oxford)** reflected on two previous theorem-proving projects: PROSPER and Forte. PROSPER [http://www.dcs.gla.ac.uk/prosper/] was an ESPRIT-IV project that developed a framework for embedded custom verification tools. PROSPER consists of a core proof engine for HOL 98 (which understands theorems and inference rules) written in ML. Custom proof engines are constructed by writing new proof procedures in ML and connecting with external plug-in components for BDDs, SMV, SAT, ACL2 and HOL libraries, and "glue code" in ML. An integration interface is provided too for communicating between the various components.

Melham explained how the PROSPER project demonstrated that the architecture worked and how it promoted the idea of componentization. There were some disadvantages, however; for example, considerable engineering overhead was required and the decision to use a functional programming language was perhaps not ideal. With hindsight, Melham suggested that it would have been better to have used standard solutions such as XML and CORBA, and separate the data scripting (which was in ML) from the control scripting (using Python).

Forte [http://www.intel.com/research/scl/fortefl.htm] is a hardware verification system developed by Intel. It consists of a lightweight theorem prover, symbolic simulation and abstraction, temporal logic model checker and a functional programming language called FL. The FL language is central to Forte and has several roles, including scripting verifications, customizing the environment, controlling model checkers, and is the term language for the higher-order logic of the theorem prover. The methodology has several advantages: it is realistic, structured, incremental, gives feedback and supports regression as well as reuse. Intel has used Forte for various verifications, including an IA32 instruction-length decoder and verification of gate-level floating point implementations against IEEE specifications. In summing up, Melham stressed that verifying industrial-scale designs was an interactive programming activity, requiring deep insight into the capabilities of the tool and design itself. The methodology was realistic, based on the way designs are carried out.

Rod Chapman (Praxis High-Integrity Systems) continued the theorem-proving theme in his talk on SPARK [http://www.praxis-his.com/sparkada/spark.asp]. SPARK is an annotated pure subset of Ada. The annotations support design by contract, including pre- and post-conditions. A verification environment, consisting of a verification condition generator and theorem prover, is available for SPARK, which can be used to prove "interesting" properties, partial correctness and the absence of runtime errors.

This tool is a commercial product, and its customers demand fast discharge of verification conditions; that is to say, minutes (such as the time taken to have lunch) rather than hours. The SPARK team at Praxis has risen to the challenge/demand of its customers, spending a considerable amount of effort on improving the proof engine for SPARK. Tactics have been added to deal with typical "patterns" arising in designs. Chapman presented some encouraging results. Four major releases of the theorem prover were applied to five example programs – the SPARK Examiner (a static-analysis tool for SPARK), SHOLIS (a safety-critical naval information system), SAWCS (marine stores management system), a ballistic simulator and US National Security Agency (NSA) Tokeniser (part of a biometrics access control system) – using the same verification condition generator running on the same computer hardware. The elapsed time together with the percentage and absolute number of verification conditions discharged were noted. In each of the examples, the elapsed verification time increased but so did the number of the verification conditions automatically discharged by the tool. SAWCS had 18704 verification conditions; 97.3% of these could be discharged with the latest version of the prover, reducing the number of verification conditions that could not be automatically discharged to just 510 (from 968).

Chapman then explained how Praxis could contribute to the verified software repository. Although it would not be possible to donate SPARK itself (as it is not open source and is unlikely to be so for the foreseeable future), or some of the case studies discussed (which were either proprietary or of a sensitive nature), Praxis could donate the Ballistic Simulator, developed by Adrian Hilton as part of his doctoral thesis. It may be possible to donate the NSA Tokeniser example too. The tokeniser is an ideal candidate challenge code because the security requirements are formally specified in Z in line with Common Criteria Evaluation Assurance Level 5 (EAL5), the core system is 100% SPARK, it is amenable to proof and several outstanding proofs remain to be carried out. Chapman urged the Grand Challenge 6 committee to approach NSA to request that the code for the tokeniser be donated to the repository.

The final talk by **Juan Bicarregui (CCLRC, Rutherford Appleton Laboratory)** focused on the logistics of setting up the proposed verified software repository, and how it would be managed and run. Bicarregui explained that a Grand Challenge 6 committee to oversee and steer the project had been set up, consisting of: Keith Bennett, Juan Bicarregui, Jonathan Bowen, Tony Hoare, Cliff Jones, John McDermid, Colin O'Halloran, Peter O'Hearn, Brian Randell, Martyn Thomas and Jim Woodcock (Chair). The committee meets at regular intervals. (The first meeting in 2005 took place on 26 January at the BCS London offices.)

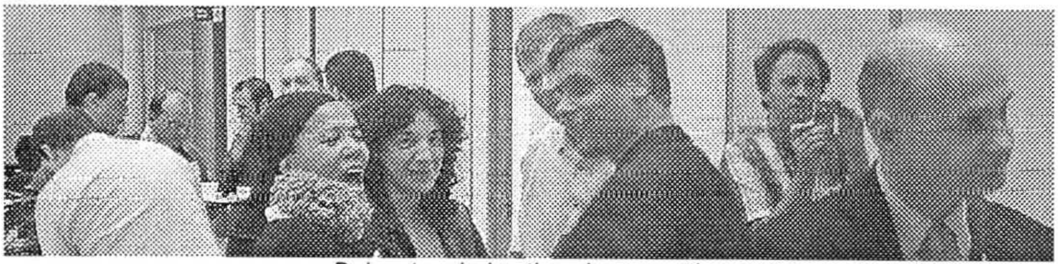
Regarding management of the repository itself, Bicarregui reminded the delegates of the vast experience CCLRC has in managing scientific repositories (at least 16, covering various disciplines). A thorough infrastructure for setting up and maintaining the repository was then proposed, involving several stages: developing the content of the repository, technology required (e.g. web-based interface for accessing the tools and making submissions), user support (e.g. helpdesk, materials, training), as well as management and dissemination.

A subcommittee would oversee the management and dissemination aspects, and would also monitor the use of resources. A forum would be set up as well, allowing tool providers, users and providers of challenge codes to give feedback on the verified software repository and its facilities.

The aim is to publicise the repository as widely as possible; ideas that will be explored include a possible newsletter, a dedicated scientific journal, and dedicated workshops/events (such as this one). So far two such workshops are planned for 2005: one at ETAPS 2005 joint conferences at Edinburgh in April [http://www.dcs.qmul.ac.uk/~ohearn/DSE-Workshop/etaps05.html], considering software verification applications for the repository, and another at FM'05, Newcastle upon Tyne in July [http://www.fmnet.info/gc6/fm05], covering tools, applications and theoretical issues. An annual Marktoberdorf-style Summer School providing training in that year's tools and challenge codes will be considered too.

To get the repository up and running, developers will be invited to contribute their tools and challenge codes. The following tools are under consideration for inclusion in the repository: the Daikon dynamic invariant checker, SLAM, the Splint static analyser, the Alloy, FDR and Spin model checkers, ACL2, HOL, ProofPower, PVS and Z/Eves theorem provers, ESC and Java optimisation framework, the B-Toolkit, JML, SPARK Ada, and the Cyclone safe dialect of the C programming language.

Following Bicarregui's talk, a lively discussion ensued, with many of the delegates in the audience participating. Several delegates were keen to receive clarification on the kinds of challenge code that could be deposited, and how the tools deposited would be used. Suggestions for possible items to be included came from the audience too, as did a list of possible outlets where the repository could be publicised. Jonathan Bowen suggested that Turing's first program could be deposited. As Turing laid the foundations for the work, this would seem a fitting tribute to the British founder of computer science. Discussions and networking continued afterwards at the festive wine (and mince pies!) reception.

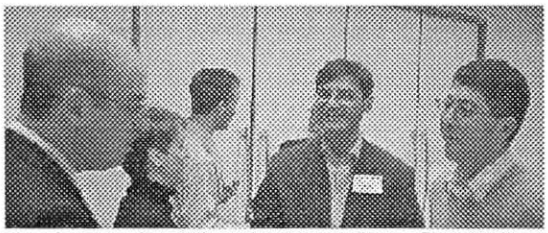


Delegates during the wine reception

Conclusion

The meeting was a success and the feedback received from the delegates and speakers was positive. All of the speakers gave informative and thought-provoking talks, and the enthusiasm and passion they felt for their work was apparent to us all. We were in the presence of some great minds, and were privileged to witness the launch of a project that could have great impact on the development of dependable systems in the future. FACS wishes the Grand Challenge 6 committee well with the Verified Software Repository project and hopes it will be able to help publicise the achievements made via reports in *FACS FACTS* and through update meetings. If you feel you are able to contribute in some way to the project, please contact Jim Woodcock on jim@cs.york.ac.uk.

The meeting has hopefully helped relaunch the FACS Christmas meeting series. It has set a high standard, which we aim to maintain and build upon if possible. Discussions on the 2005 Christmas meeting, to be held once again at the BCS London offices, have already begun. John Cooke (Loughborough University, FAC journal liaison officer for FACS) will organize this meeting, with help from Jonathan Bowen, Paul Boca and others. Further details will be announced in due course. In the meantime, please make a provisional note in your diaries for December 2005! Information will appear on the BCS-FACS website in due course [http://www.bcs-facs.org/events/xmas2005/]. Information on the work of the Grand Challenge 6 Committee on Dependable Systems Evolution, including the Verified Software Repository, is also available online [http://www.fmnet.info/gc6].



Left to right: Tom Melham, Fiona Polack, Michael Butler, Ali Abdallah, Steve Schneider

Acknowledgements

The British Computer Society (BCS) [http://www.bcs.org] provided the wonderful new facility at their offices in central London, near Covent Garden, free of charge. The EPSRC-funded Interdisciplinary Research Collaboration on Dependability [http://www.dirc.org.uk] provided financial support for the event. FACS and the organizers of the event would like to thank both the BCS and DIRC for their generosity.

Several people helped make the event a success. Juan Bicarregui (CCLRC, Rutherford Appleton Laboratory) and Jim Woodcock (University of York and Grand Challenge 6 Chair), the technical co-chairs, assembled an excellent group of speakers from academia and industry. Jonathan Bowen (London South Bank University, and FACS Chair) and Paul Boca (FACS Newsletter Editor) were the organizational co-chairs, assisted by Michelle Hammond (PhD student, London South Bank University) and Teresa Numerico (Leverhulme Research Fellow, London South Bank University).

Further information is available on the BCS-FACS website [http://www.bcs-facs.org/events/xmas2004]. ■

Formal Methods 2005

18–22 July 2005 – Newcastle upon Tyne, UK

www.csr.ncl.ac.uk/fm05

Open for e-Registration from 7 March

Full Scientific Programme 20–22 July
Invited Speakers:
Mathai Joseph (Tata Research & Development, Pune, India)
Marie-Claude Gaudel (Université de Paris-Sud, France)
Chris Johnson (University of Glasgow, UK)

Industry Day: 20 July

Co-located Conferences (18–19 July):
Calculemus 2005
Formal Aspects of Security and Trust (FAST) 2005

Workshops (18-19 July):
Grand Challenge Workshop on Dependable Systems Evolution
Web Languages and Formal Methods (WLFM 2005)
Overture – the future of VDM and VDM++
Practical Applications of Stochastic Modelling (PASM 2005)
Workshop on Rigorous Engineering of Fault Tolerant Systems (REFT 2005)
The Railway Domain (TRain 2005)

Tutorials (18–19 July):
The Spec# Programming System: an Overview
Formal Aspects of Software Architecture
Increasing Dependability of Smart Card Based Systems
Perfect Developer
SPARK
Petri-nets in Asynchronous Circuit & Systems Design
Verifying Industrial Control System Software

Organiser:
Claire Smith, tel: +44 (0) 191 222 7999, email: claire.smith@ncl.ac.uk

General Chair:
John Fitzgerald, email: john.fitzgerald@ncl.ac.uk

Letting Formal Methods in through the back door: The UML and MDA

Kevin Lano, UML Liaison Officer for FACS

The Unified Modelling Language (UML) has become the most important "semi-formal" software specification and design language, since its origin from the unification of the Booch, OMT and OOSE methods in the mid 90s. UML was adopted as a standard by the Object Modelling Group (OMG) in 1997. UML contains a wide range of notations, from class diagrams to a version of Harel Statecharts, and provides a capability for formal description of class invariants, and operation pre- and post-conditions via the "Object Constraint Language" (OCL). Z was one of the original influences on OCL, via the Syntropy method of Cook and Daniels. A major revision of UML, to version 2, is due for release in 2005, and many aspects of this revision are already available from the OMG's web site [http://www.omg.org/uml]. The revision attempts to rationalise and unify the many notations and concepts in UML, and solve some of the problems, particularly in semantics, which were present in earlier versions of the language. OCL has also been revised, with a more comprehensive type system and the removal of some peculiarities (in the original version of OCL, sets could not be elements of other sets, and so all sets were automatically "flattened" to remove such nesting).

As an example of the style of OCL specification, the constraint (on a Bag class) that the number of letters in the bag with symbol 'e' is at most 12, is:

```
bagLetters->select(symbol = 'e')->size() ≤ 12
```

In Z, this could be written as

```
# { x : bagLetters | x.symbol = 'e' } ≤ 12
```

At the same time that UML has been evolving to support more declarative and formally precise models, the need for such models has been recognised within another major OMG initiative, the "Model-Driven Architecture" (MDA). The MDA aims to raise the level at which software development takes place, from programming language code to diagrammatic models, such as those of UML. It envisages a situation where developers will focus their effort on the construction of "Platform Independent Models" (PIMs), free from implementation bias, and that from these, "Platform Specific Models" (PSMs) will be generated, possibly with a high degree of automation. PSMs describe a version of the system tailored to a particular environment, such as J2EE. From a PSM, automated code generation can take place to produce an executable version of the system for the platform.

The MDA initiative should provide a good opportunity for the formal methods community to contribute to and influence mainstream software engineering practice. The use of models as "programs" means that these models must have a clear and unambiguous semantics, and that analysis techniques, for animation or detecting errors – at the model level – are needed. One application of formal methods to UML and MDA has been in the development of model-checking techniques for UML. Translations from UML into the notations of model-checkers such as SMV, SPIN and Promela have been used for verifying UML models, and producing counter-example traces when a model fails to satisfy a specification. Other formal methods, such as B and Alloy, have also been used to verify UML specifications, avoiding the state explosion problem which limits model checking, but requiring human intervention in the proof process.

Another issue that is central to the MDA approach is the correctness of transformations on models – such transformations may be used to improve the quality or abstraction level of a PIM, or map a PIM to a PSM. A precise semantics for models is again essential in order that transformations can be certified as correct and used in tools. Some current work on transformations which uses formal techniques is available online [http://www.gvtp.org].

Apart from the OMG's website, the best source for the latest on the MDA and precise UML approaches is the Software and Systems Modelling Journal (SoSyM) [http://www.sosym.org]. ■

Formal Methods 2006

McMaster University, Hamilton, Ontario, Canada

21 – 28 August 2006

http://fm06.mcmaster.ca
Website under construction

First Announcement and Call for Satellite Events and Workshops

We are inviting proposals for satellite events and workshops. In particular, but not exclusively, we encourage proposals for a co-ordinated series of satellite events and workshops on application domains like the automotive, railway, and avionics sectors. Also, with safety and certification issues related to software becoming more prominent, we think it would be useful to have a workshop on the topic of certification. Parties interested in helping to organise related workshops should contact Tom Maibaum on tom@mcmaster.ca .

Conference Chair: Emil Sekerinski, McMaster University, emil@mcmaster.ca

Program Chairs: Jayadev Misra, University of Texas, Austin
Tobias Nipkow, TU Munich

Workshop Chair: Tom Maibaum, McMaster University
Tutorial Chair: Jin Song Dong, NUS Singapore

Poster and Tool Exhibition Chair: Marsha Chechik, University of Toronto

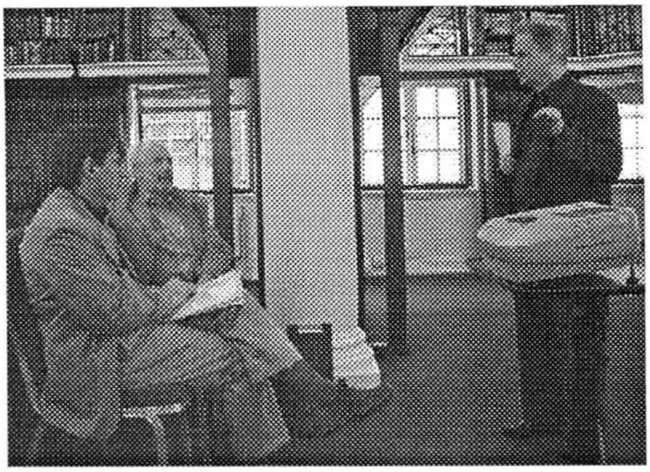
Program Verification and Semantics: Further Work, Science Museum, London

Teresa Numerico[1] & Jonathan P. Bowen

A follow-on of a 2001 meeting, in which some of the major computer scientists in the area presented their early experiences in the field.

"Semanticists should be the obstetricians of programming languages, not the coroners." – John Reynolds (2 December 2004)

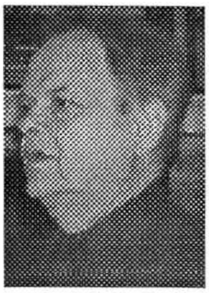
In the suitably historical location of the Fellows' Library and the adjoining Director's Suite of the London Science Museum, a meeting on "*Program Verification and Semantics: Further Work*" was held on Thursday 2 December 2004, under the joint auspices of the BCS Computer Conservation Society (CCS) [http://www.bcs.org/sg/ccs] and BCS-FACS [http://www.bcs-facs.org]. It was a follow-up of a previous meeting held in 2001, entitled "*Program Verification and Semantics: The Early Work*"[2]. The invited speakers at the 2004 event were chosen as being among the most important pioneers in formal methods; they were John Reynolds (Carnegie Mellon University, USA), Gordon Plotkin (University of Edinburgh) and Cliff Jones (University of Newcastle upon Tyne). The meeting was introduced by Roger Johnson (Birkbeck, University of London), Chair of the CCS. The meeting was split into two sessions, with Cliff Jones chairing the first half, including presentations from Reynolds and Plotkin, and Jonathan Bowen (London South Bank University, BCS-FACS Chair) chairing the second half.



Left to right: Gordon Plotkin, Cliff Jones and John Reynolds

[1] Dr. Teresa Numerico is a Visiting Research Fellow at London South Bank University, funded by the Leverhulme Trust [http://www.leverhulme.org.uk], on leave from the University of Salerno, Italy.

[2] For more details on the 2001 *Program Verification and Semantics* event, see the associated website [http://vmoc.museophile.org/pvs01].



John Reynolds of Carnegie Mellon University, Pittsburgh, USA was the first speaker [http://www.cs.cmu.edu/~jcr]. In his talk "*The discoveries of continuations*"[3] he concentrated on the invention and application of the concept of continuations, a key idea in the field of program verification and semantics[4]. He believed that continuations and related concepts were discovered and rediscovered many times in different contexts. This was due only partially to the lack of communication between scientists; instead it was largely the outcome of the different environments in which the idea could be applied. Continuations allow a wide range of program transformations that can be useful both in the interpretation of programs (e.g., through an operational semantics) and in denotational semantics, in the style of Scott and Strachey. Continuations were useful because they were an abstraction of "the rest of the program", in terms of a function or procedure, that clarified the control structure of imperative programs. Considering the multi-functionality of the concept, it is not surprising that continuations were independently rediscovered several times, and cannot be attributed to a single person.

When the Algol 60 language [http://en.wikipedia.org/wiki/ALGOL] was launched, there was a burgeoning of research on formal programming languages and their implementation. In an Algol compiler, the representation of a return address or the target of a jump had to provide both the relevant object code and a dynamic environment (i.e., a stack reference). In retrospect, according to Reynolds, this pair was a representation of a continuation. Similarly, in Peter Landin's abstract SECD (Stack Environment Control Dump) machine[5], the dump, which described the computation to be executed after the control was concluded, was a representation of a continuation.

Perhaps the earliest description of a use of continuations was given by Adriaan van Wijngaarden[6] in 1964 at the IFIP *Working Conference on Formal Language Description Languages* held in Baden bei Wein, whose proceedings were published two years later[7]. When he began to investigate the history of continuations, Reynolds was surprised that Van Wijngaarden's presentation did not seem to establish the concept within the computer science community, particularly since Dijkstra, Hoare, McCarthy, McIlroy and Strachey all participated in the recorded discussion of the presentation, and other conference attendees included Bohm, Elgot, Landin and Nivat. However, van Wijngaarden's contribution fell on fallow ground; even Strachey was unaware of

[3] The talk was based on his homonymous paper: Reynolds, J. (1993) "The discoveries of continuations", *Lisp and Symbolic Computation*, 6:233–247.

[4] For an online explanation of *continuations* with many hyperlinks, see the Wikipedia entry [http://en.wikipedia.org/wiki/Continuation].

[5] See: Landin, P. (1964) "The mechanical evaluation of expressions", *The Computer Journal* 6(4):308–320. Note that Peter Landin [http://www.dcs.qmul.ac.uk/~peterl] was a speaker at the previous *Program Verification Semantics* meeting and an attendee at this meeting. His research has had an important influence on the work described by all three speakers at this event.

[6] Wijngaarden (1916–1987) was an engineer who was also a founding father of computer science in the Netherlands. A short online biography is available [http://www.answers.com/topic/adriaan-van-wijngaarden].

[7] Steel, T. B. (1966) *Formal language description languages for computer programming*, North-Holland.

the similarities between that work and Wadsworth's later description of continuations.

It is possible that the lack of communication was due to philosophical differences between van Wijngaarden and other researchers, particularly his abhorrence of abstract syntax and his belief that proper procedures were more basic than functions. A more likely cause, however, was his failure to pinpoint continuations themselves as a significant concept. As pointed out in McIlroy's private recollection to Reynolds: "*... an idea can be understood without all its ramifications being seen, even by its originator. Since van Wijngaarden offered no practical examples of continuation passing, nor any theoretical application, save as a trick for proving one isolated and already known result, the value of continuations per se did not come through.*"

In December 1969, the Polish researcher Antoni W. Mazurkiewicz circulated a working paper entitled "*Proving algorithms by tail functions*"[8], in which he used an automaton-like concept of an algorithm, for which he proposed a semantics using a "tail function". This is an environment that maps labels into command continuations, similar to the continuation semantics of an imperative language with labels and "goto" instructions. Reynolds felt it was unclear whether this was a discovery of continuations or a precursor. However, Wadsworth has explicitly acknowledged that his later work was inspired by that of Mazurkiewicz[9].

A discovery due to F. Lockwood Morris was presented during a colloquium given at Queen Mary College London in November 1970[10] in which he gave a definitional interpreter for a call-by-value functional language with extensions for assignments and labels. Reynolds was in the audience of that lecture and from the very beginning appreciated the possibility of using different styles of definitional interpreters varying in their abstractness and circularity, making use of continuations.

Reynolds was himself involved in the spread of these ideas; during a visit to Edinburgh in December 1970, he described Morris's talk to Rod Burstall and Chris Wadsworth – and Wadsworth explained that he had been pursuing the same idea in the context of denotational semantics. Subsequently, Strachey described the idea in a seminar at the *Institut de Recherche d'informatique et d'Automatique* in May 1973[11], but Reynolds is inclined to believe that though the paper was written by Strachey the idea should be attributed to Wadsworth.

The idea of continuations was rediscovered by other researchers, such as James H. Morris. In 1971, he submitted a paper to the ACM in which he used continuations to transform Algol 60 programs. When the referee pointed out that van Wijngaarden had described a similar transformation seven years earlier, the paper was reduced to a letter to the editor, in which Morris showed

[8] The final version of the paper was printed two years later: Mazurkiewicz, A. W. (1971) "Proving algorithms by tail functions", *Information and Control*, 18(3):220–226.

[9] Wadsworth, C.P. (1992) Electronic mail to Amr A. Sabry, 24 December.

[10] Morris, F. L. (1993) "The next 700 formal language descriptions", *Lisp and Symbolic Computation*, 6(3/4):249–256. Original manuscript dated November 1970.

[11] Strachey, C. (1973) "A mathematical semantics which can deal with full jumps", in *Théorie des Algorithmes, des Langages et de la Programmation*, IRIA (INRIA), Rocquencourt, France, 175–193; and Strachey, C. & Wadsworth, C. P. (1974) *Continuations, A Mathematical Semantics for Handling Full Jumps*, Technical Monograph PRG-11, Oxford University Computing Laboratory, January.

that the transformation could be used to eliminate procedures that returned compound or higher-order values.

In 1972, at the ACM *Conference on Proving Assertions about Programs* in Las Cruces in New Mexico, Michael J. Fisher presented a paper in which he introduced continuations. This paper is notable for including the first proof about the semantics of continuations. It proved that the Continuation-Passing Style (CPS) preserves meaning. The result was obtained in a setting where λ-expressions[12] denote closures.

In 1973, in a short paper at the first Computer Science Conference, held in Columbus, Ohio, and in a longer preliminary report, S. Kamal Abdali used continuations, which he called "program remainders", to translate Algol 60 into a pure call-by-name λ-calculus, rather than the call-by-value λ-calculus used by Landin. In linking call-by-name in Algol 60 with call-by-name in the λ-calculus, this work was a precursor of the view of Algol-like languages by Oles and Reynolds in the 1980s.

Reynolds wanted to stress two concepts in his contribution: on one hand he claimed that original ideas such as continuations and related concepts are never discovered once, but they keep on being reinvented in different environments and used for various properties that they exhibit; on the other hand he argued that the relevance of continuations was also due to the fact that it was applicable both in denotational and in operational semantics contexts. According to Reynolds the two approaches are actually complementary, giving original results in the program semantics field.

The second speaker was **Gordon Plotkin**, University of Edinburgh [http://homepages.inf.ed.ac.uk/gdp], who gave a talk on "*The origins of structural operational semantics*"[13]. This was a personal reconstruction of his studies in the area and, being a major researcher in the field, also a recollection of the evolution of operational semantics itself, as well as a passionate account in favour of the use of the operational approach for all semantic descriptions of programming languages. The reconstruction was so enthusiastic that at some points during the talk it was hard to grasp all the deep and thoughtful details of the story. But the general feeling of the audience was that he was telling an account in which he had strongly contributed and indeed largely initiated, so there was a personal feel to his presentation, giving a privileged view of the research covered. The audience not only heard oral testimony, but also witnessed slides hand-written during Plotkin's influential visit to Aarhus.

As a postgraduate, Plotkin learnt λ-calculus and was impressed by the abstract SECD Machine of Peter Landin, as well as McCarthy's contributions to the field, such as the 1962 introduction of abstract syntax, a fundamental tool for all semantics of programming languages. Moreover there was the Vienna Definition Language (VDL) that was used by the Vienna School to specify a real programming language for the PL/I ("Programming Language One")

[12] See the Wikipedia entry on the *lambda calculus* for further information [http://en.wikipedia.org/wiki/Lambda_calculus].

[13] The talk was based on: Plotkin, G. D. (2004) "The origins of Structural Operational Semantics", *Journal of Logic and Algebraic Programming,* 60–61(2):3–15. See also the Wikipedia entry for *operational semantics* [http://en.wikipedia.org/wiki/Operational_semantics].

[http://en.wikipedia.org/wiki/PL/I] abstract machine. The speaker did not care for this way of doing operational semantics because it was too complex, burying the semantic ideas in technicalities.

From the end of the 1960s, denotational semantics was popular; e.g., see Mike Gordon's thesis on pure LISP, supervised by Rod Burstall from 1969[14]. The work by Dana Scott on LCF (the Logic of Computable Functions), a simply typed λ-calculus, was very influential and, in particular, Plotkin wrote a paper[15] on PCF (Programming Language for Computable Functions) in which he gave an operational semantics for typed λ-calculus with Booleans, natural numbers and recursion at all types. The idea behind PCF, according to the speaker, was that it was possible to consider the term calculus of the logic as a programming language only if one had an operational semantics. However, Scott did not completely agree with this. Obviously there were also other influences involved in the development of operational semantics, but these are some of the most relevant ones.

Plotkin also pointed out the role of teaching in developing "toy" languages that were suitable for students as well as for the progress of the research itself. In 1973, at the University of Edinburgh, he taught third year undergraduates and, instead of teaching a course on λ-calculus, he tried McCarthy-style recursive function definitions. Subsequently, in 1975, Robin Milner taught SIL (Simple Imperative Language) to students. He later worked on a structural operational semantics for concurrency, developing the Calculus of Communicating Systems (CCS). In 1979, Milner spent six months in Aarhus on a visiting lectureship that resulted in his classic CCS book[16].

Two years later Plotkin himself was invited on the same visiting lectureship, during which he developed Structural Operational Semantics (SOS). He realized at this point that researchers were then concentrating on proving the adequacy of an operational semantics with regards to the denotational semantics, but that the rule-based operational semantics was both mathematically elementary and simpler than the denotational one. So Plotkin imagined the possibility of dropping the denotational semantics, and tried to follow up this idea in his course on this subject in Aarhus[17]. Many of the linguistic ideas he employed came from denotational semantics, such as Bob Tennant's principles of abstraction, correspondence and qualification.

In his recollection of the development of SOS, Plotkin focused on two major ideas. The first was that the configurations of SOS were those of an abstract machine but without low-level detail; rules were used to avoid these complexities.

The second idea related to compositionality: denotational semantics is compositional in the sense that the meaning of a phrase in a language is given as a function of the meaning of its immediate subphrases; in contrast, with an operational semantics, the meaning of a program is, essentially, the collection

[14] Gordon, M. C. J. (1973) *Models of Pure LISP*, Ph.D. Thesis, Experimental Programming Reports: No. 31, School of Artificial Intelligence, University of Edinburgh.

[15] Plotkin, G. D. (1977) "LCF Considered as a Programming Language", *Theoretical Computer Science*, 5(3):225–255.

[16] Milner, R. (1980) *A Calculus of Communicating Systems*, LNCS, Vol. 93, Berlin: Springer-Verlag.

[17] Plotkin, G. D. (1981) *A Structural Approach to Operational Semantics*, DAIMI FN-19, Computer Science Department, Aarhus University, published in *Journal of logic and Algebraic Programming* 60–61(2):17–139.

of all the transitions it can make. This is given by the rules and is primitive recursive (structural) rather than compositional in the subphrases, hence the term "structural."

According to the speaker, looking back at what has been done so far makes it even clearer that there remains much to do, for example:

1. giving a semantics for a real (not toy) language that is complete, readable, natural, modifiable, and extendable;
2. understanding the relationship with other forms of operational semantics;
3. the creation of useful software tools based on operational semantics.

Denotational and operational semantics can be seen as counterparts with respect to offering semantics for programming languages; Plotkin was strongly convinced that the operational approach is the more useful for certain goals of formal methods and his talk tried hard to argue in favour of this position.

Peter Mosses during question time

The final speaker was **Cliff Jones**[18], University of Newcastle upon Tyne [http://www.cs.ncl.ac.uk/people/home.php?name=cliff.jones], who focused his talk on "*Language Semantics: the Vienna part of the story*". His contribution gave a wide and well-organized historical account of the different approaches of language semantics between the 1960s and the present day. He concentrated more on the "Vienna" group side of the formal languages semantics history because he spent some time in Vienna working for the IBM research group based there. As already pointed out by John Reynolds, one of the milestones of the language semantics research was the 1964 IFIP Conference at Baden bei Wien, whose proceedings were printed two years later and included the major discussions by participants.

Between 1965 and 1968, the Vienna Group faced the task of defining the very large PL/I. As well as including a multiplicity of features, PL/I was challenging because it included complex constructs such as "goto", "on units", tasking, etc. The Vienna Group called their semantics Universal Language Definition (ULD-III); however their operational semantics notation was more

[18] For some more information on the history of semantics of formal methods see: Jones, C. B. (2003) "The Early search for tractable ways of reasoning about programs", *IEEE Annals of the History of Computing*, 25(2):26–49 and Jones C. B. (2003) "Operational Semantics: concepts and their expression", *Information Processing Letters,* 88:27–32.

widely known as *Vienna Definition Language* (VDL). At that time, the semantic approach was operational. Included in the list of the influential researchers were Elgot, Landin and McCarthy with different backgrounds. The effort did achieve its objectives, and even resulted in clarifications of PL/I. But the definitions of the language tended to use "grand states" that gave great difficulty when used in proofs about VDL descriptions. These problems, together with a shift in the interests of the semantics community, pushed the Vienna Group towards a change in the paradigm, resulting in a move to what became called denotational semantics.

If we consider the operational semantics approach similar to the creation of an interpreter for the language, we can think of the denotational semantics model as the creation of a compiler for the language: but a compiler that maps to mathematical objects (functions). Strachey and Landin were the first researchers to move towards the denotational semantics perspective[19], and Dana Scott solved some of the theoretical problems of the denotational approach during the 1960s.

The Vienna Group, during the 1970s, worked on a compiler for PL/I that was developed from a denotational definition of PL/I [20] , The *Vienna Development Method* (VDM) [http://www.csr.ncl.ac.uk/vdm] did not use continuations to model various kinds of jumps (for example, "goto" statements) as advocated at the Programming Research Group in Oxford. Instead, an exit mechanism based on an earlier "functional semantics" of Algol 60 by Jones and colleagues in the IBM UK Laboratories at Hursley Park near Winchester was used.

Denotational definitions could be advantageous in the sense that they expressed more efficaciously the meaning of the syntactic constructs, but they could not completely substitute the operational definitions. In particular, they are technically inadequate to give a correct denotation of concurrency. If operational semantics could avoid the use of unneeded mechanistic features, as in the successful proposal of the structured operational semantics (SOS) made by Gordon Plotkin during his Aarhus visit in 1981, as already mentioned, the advantages of operational semantics are even clearer.

The speaker, during the 1980s, was convinced that the only correct approach to semantics was the denotational one. When he switched to an academic career in 1981 he decided to teach students the mathematically heavy denotational methods for modelling semantics of programming languages. Jones declared that later he changed his view about the best approach to presenting semantics. He is now convinced that operational semantics, particularly using the rule-oriented approach introduced by the previous speaker, Plotkin, is more useful for modelling non-determinacy and concurrent processes. According to Jones, operational semantics is very good as a "thinking tool" for creating a programming language, helping to avoid useless complexities.

[19] Landin, P. J. (1964) "The mechanical evaluation of expressions", *Computer Journal*, 6(4): 308–320 and (1965) "A correspondence between ALGOL-60 and Church's Lambda-Notation", *Communications of the ACM*, 8(2):89–101, 158–165.

[20] Bekič, H., Bjørner, D., Henhapl, W., Jones, C. B. & Lucas, P. (1974) *A formal definition of a PL/I subset*, Technical Report 25.139, IBM Laboratory, Vienna, Austria.

Left to right: Brian Monahan, Jawed Siddiqi, Peter Landin, John Reynolds and Paul Boca, at the end of the meeting

Conclusion

In summary, the meeting was well-attended by an audience of around 75 people that contributed to the success of the event with questions and anecdotes. Having a joint event between the BCS Computer Conservation Society and FACS boosted numbers considerably and ensured a good mix of people interested in the history of computing and formal methods.

The speakers tried to reconstruct the history of the field from the early stages during the 1960s to the major achievements of the 1970s and the 1980s, putting this into context with the most recent advances. The passion and wisdom of the speakers was contagious for the audience and gave a sense of the challenges faced during those early years as well as the opportunity to share with the speakers various concerns and the possibilities for the future of the field.

For further details about the meeting, including photographs and slides from the talks, see online information under the *Virtual Museum of Computing* website [http://vmoc.museophile.org/pvs04]. We hope it will be possible to hold a third instalment in the Program Verification and Semantics series in due course.

Acknowledgements

John Reynolds, Gordon Plotkin and Cliff Jones all provided very helpful feedback on their sections in this report. Paul Boca also gave useful comments on an early draft. BCS-FACS provided financial sponsorship for the event. ■

REFT 2005: Workshop on Rigorous Engineering of Fault-Tolerant Systems

RODIN

Newcastle upon Tyne, UK

19 July 2005

Deadline for submissions: 22 April

Workshop organized by the partners of FP6 IST RODIN (Rigorous Open Development Environment for Complex Systems [http://rodin.cs.ncl.ac.uk], who are aiming to build a network of researchers from a wider community to promote integration of the dependability and formal methods fields.

SCOPE

- Verification and refinement of fault tolerant systems
- Integrated approaches to developing fault tolerant systems (including integration of different formalisms as well as formal strengthening of informal notations)
- Formal foundations for error detection, error recovery, exception and fault handling
- Abstractions, styles and patterns for rigorous development of fault tolerance
- Development and application of tools supporting rigorous design of dependable systems
- Integrated platforms for developing dependable systems
- Rigorous approaches to specification and design of fault tolerance in novel computer systems
- Case studies demonstrating rigorous development of fault tolerant systems

SUBMISSION INSTRUCTIONS

Interested participants are invited to submit a 8 page position paper relevant to the topics of the workshop. The submissions must conform to the proceedings publication format (LNCS style format). Electronic submissions should be sent to Ms L. B. Talbot [L.B.Talbot@newcastle.ac.uk] as a pdf attachment. The e-mail body should include the title, the list of authors and their affiliations.

Papers will be selected on the basis of their relevance to the theme. The time slot for presenting the paper will be given depending on the potential interest of the paper for the audience.

Deadline for submissions: April 22, 2005
Notification: May 16, 2005
Final submission: June 1, 2005

All accepted papers will be included in a workshop proceedings printed as the technical report at University of Newcastle upon Tyne (UK). We are planning to publish a set of papers based on selected workshop submissions as post-proceedings in Springer.

WORKSHOP ORGANISERS

Michael Butler (University of Southampton)
Cliff Jones (University of Newcastle upon Tyne)
Alexander Romanovsky (University of Newcastle upon Tyne)
Elena Troubitsyna (Aabo Akademi)

TRain: The Railway Domain

Dines Bjørner

The aim of this *FACS FACTS* note is to convince some readers to join a loosely knit group of researchers with the aim of – together – researching and developing theories about "The Railway Domain", abbreviated 'TRain'.

TRain is essentially a "Grand Challenge" project in computing science – (also) using formal specification & verification (theorem proving, model checkers, formal specification based testing, etc.) techniques, such as along the lines of ASM, B (and event B), CafeOBJ, CASL, CSP, Duration Calculi (DC), RAISE/RSL, TLA+, VDM/VDM-SL, Z (Z/Eves, TCOZ), etc., – with these being possibly integrated with one another (to wit: RAISE with Timing + DC) or with visual/diagrammatic (formal) techniques: CTPs, MSCs, LSCs, Petri Nets, Statecharts, UML Class Diagrams, etc., etc. I refer you to the TRain website (front page pictured above) [http://www.railwaydomain.org/] for further details. These web pages are currently being revised, updated and extended.

Welcome To

Highlights

TRain Web Site Structure

Request for Membership

TRain Newsletter

TRain Brochure

You may think of yourselves as working in verification, or in formal testing, or in model checking, or in integrated formal techniques ('methods'), maybe not so much in 'railways' – or in the larger, but strongly related area of transportation systems.

Fine, excellent – but I am sure that railways pose significant challenges also in your area, as an application. Therefore I want your kind consideration. "Turn" your recent reports and papers around: And you may have a non-trivial contribution to the understanding of The Railway Domain, whether just to the domain itself, void of any reference to requirements to computing systems, or to requirements to railway systems, or to the design of such systems.

The way I see TRain is – for example – as follows, i.e., by analogy: Mechanical engineers, designing, say transmission systems, make use of the classical theory of mechanics; aircraft designers of aerodynamics, ship designers of fluid dynamics, etc., etc. So engineers make use of theories of physics. But which theory should software engineers designing software for railways, for example, base their work on? Before software can be designed its requirements must be understood, but before requirements can be finally formulated one must understand, we claim, the domain. So TRain is about understanding the railway domain.

I would like you to respond to this note, to me Dines Bjørner [bjorner@comp.nus.edu.sg] and Martin Penicka [penicka@fd.cvut.cz] who is mastering our web pages, in either of the following ways:

[NO] Sorry, Dines, It's a great idea – but I am busy otherwise. Good luck!

[YES] Yes, great idea, let's do something. Then select one or more of:

[MAILING LIST] Keep me posted on what is happing in TRain

[REPOSITORY] Please attach .pdf or .ps files of reports and papers that YOU think should go into the TRain repository – please copy the email to Martin Penicka [penicka@fd.cvut.cz].

[BIBLIOGRAPHY] Please include details (for example BibTeX entries) of reports that YOU think should be included in the Train Bibliography.

[INTERESTS] Please indicate which areas within Railways YOU are interested in:

- Allocation & Scheduling
- Interlocking
- Network Planning
- Rostering
- Signalling
- Time Tabling
- Train Control Systems

etc. ...

Please look under

- http://www.railwaydomain.org/index.php?page=docs&subpage=pre-conference
- http://www.railwaydomain.org/index.php?page=docs&subpage=pre-reports
- http://www.railwaydomain.org/index.php?page=docs&subpage=reports
- http://www.railwaydomain.org/PDF/tb.pdf

for representative documents with formal models of various facets of the railway domain.

I look forward to hearing from you. ■

Dines Bjørner will be reporting on **TRain** activities in a new regular column in *FACS FACTS*.

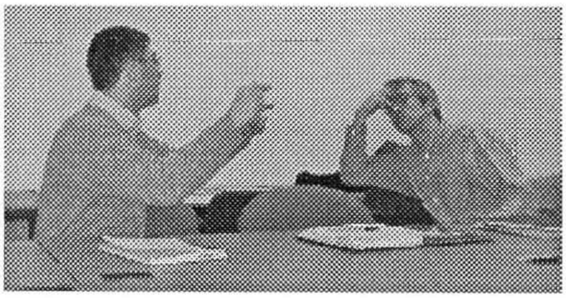
An evening seminar by Dines Bjørner is scheduled to take place on 25 July 2005. Please see the BCS-FACS website for further details of this seminar and others in the series [http://www.bcs-facs.org/events/EveningSeminars].

The RefineNet Column

Adrian Hilton

RefineNet is an EPSRC-funded collaboration of UK university research departments and companies from industry. The first network meeting of 2005 was held at Sheffield University on 10 – 11 January, and focused on the foundations of refinement.

Monday's session opened mid-afternoon. Martin Henson (University of Essex) was first up, presenting the nuZ wide-spectrum logic system. This modifies the original Z specification language by using total- rather than partial-correctness . Everything is characterized by introduction and elimination rules, and the actual language is quite small. You can derive a range of refinement rules, swapping conditional logic for a relational logic of refinement. nuZ was devised for reasoning about specifications and programs, and for deriving programs from specifications.



John Derrick (University of Sheffield), pictured on the far right with Steve Schneider, followed, examining temporal logic properties under refinement. Properties expressed in the standard temporal logics, e.g. LTL, by operators such as "always" and "eventually" are not necessarily preserved. The problem comes when the structure of the state during the operation changes between the original and refined version, and an assertion related to the original state structure. This arises when you have an odd number of negations in the assertion you define. If you have an even number of successive negations then it all works out OK. CTL has similar problems with the "there exists a path" operator, and the mu-calculus encodes LTL and CTL so you get failures accordingly.

Chris Thomson, a doctoral student at the University of Sheffield, used the last session to present his work on observing software change in software developments. The data as it stood showed no clear refinement process since the developments were continually reacting to events (e.g. requirements changes) all of the time. Can we define the out-of-sequence activities as refinement? In the ideal model yes, but in the observed studies no. There is a notion of "change" which isn't refinement and which is non-linear, unexpected, and unpredictable except late in the development cycle.

Tuesday's session opened with guest speaker Steve Dunne (University of Teesside). He aimed to convince us that there are as many varieties of refinement between abstract data types as there are between processes (e.g., the trace, failures, divergences refinements in CSP). Sure enough, he showed a hierarchy of refinements and claimed that the classical simulation ordering on ADTs just corresponds to plain trace refinement of processes. This led to some spirited technical discussion!

Ana Cavalcanti (University of York) presented on angelic nondeterminism and the Unifying Theories of Programming (UTP). The theme was examination of the varying healthiness conditions for set-based relations in UTP, establishing two-way maps between binary multirelations and set-based relations. That ended up defining refinement in the binary multirelations as implication, and subset in the binary model. Tony Simons (University of Sheffield) followed with a talk on specifying "object machines", a form of hierarchical state machine, and how to improve on the currently allowed refinements.

The meeting concluded with a business meeting, discussing the forthcoming Refinement Workshop at ZB2005 and the next few RefineNet meetings. We also discussed setting up some joint publication work. A very productive and informative couple of days overall; thanks to John Derrick for his organization.

Notes of this meeting, including PDFs of the presentations made, are available on-line [http://www.refinenet.org.uk/sheffield_notes.html]. The next RefineNet event will be a public workshop at ZB 2005 at the University of Surrey, 12th April. The meeting after that will be at Manchester University in early July, exact date to be determined. The theme will be automation and mechanisation.

The RefineNet website [http://www.refinenet.org.uk] includes a list of the current members, details of past meetings, a schedule of future meetings and contact details. Enquiries are welcomed: enquiries@refinenet.org.uk . ■

Wanted: Examples of C Code developed using Formal Methods

Dr. Sue Black and Professor Jonathan Bowen, both from London South Bank University, are considering a study of impact analysis for software developed using formal methods (compared with traditional techniques). They have a tool that can assess C code and are looking for formal methods development examples where C code was produced. This is a request for any such code that they could use as input to the tool, especially for a real industrial example if possible, although any example or pointers would be welcome. They can treat the code or any other information as confidential if necessary, although they are looking to publish the results eventually in some form, anonymised suitably if necessary.

If you can provide any help, please contact Sue Black [blackse@lsbu.ac.uk] and Jonathan Bowen [Jonathan.Bowen@lsbu.ac.uk] directly.

ZB2005

University of Surrey, Guildford, UK

12 April, 2005 - Tutorials and Refinement Workshop
13-15 April, 2005- Main Conference

Online registration now open
http://www.zb2005.org

BCS FACS AWE UniS University of Surrey

Supported by the APCB and the Z User Group

Formal Aspects of Performance Evaluation

Jane Hillston & Stephen Gilmore

1. Introduction

Computer systems need to provide services that meet, or exceed, expectations of correct function and timely response. In the context of computer software, correctness of functions is a well-studied subject. Most practising computer scientists understand very well the notions of proving implementations correct with respect to a high-level specification; developing an implementation stepwise by verified refinement steps; calculating programs by the application of algebraic laws of programming; or extracting a program from a formal constructive proof. In comparison, timely response is less well studied, and less well understood.

A very coarse notion of timely response is sometimes used in program verification, separating *partial* and *total* correctness. That is, determining whether the program will ever respond at all. A more discriminating evaluation is provided by an analysis of the asymptotic complexity of the program, describing its run-time as a function of the size of the input. Such an analysis differentiates algorithms with linear, polynomial or exponential running times but this is still far removed from a typical users' perception of program efficiency, even if only because it elides constant factors from expressions for run-times.

A different form of quantitative analysis seeks to evaluate systems with respect to performance metrics such as throughput, response time and resource utilisation. Such performance evaluation has sound mathematical foundations, dating back to Erlang's pioneering work on telephone exchanges at the beginning of the last century. However, the size and complexity of many modern systems make it infeasible to work directly with the mathematical model (typically a Continuous Time Markov Chain (CTMC)).

A variety of system description languages, equipped with varying degrees of formality, have been introduced to ease the task of generating the mathematical model. Such languages must incorporate the quantitative information needed to carry out the performance analysis as well as a mapping to a CTMC. PEPA (Performance Evaluation Process Algebra) is a process algebra designed for this purpose.

2. The PEPA project

The PEPA project [http://www.dcs.ed.ac.uk/pepa/] started in Edinburgh in 1991. It was motivated by problems encountered when carrying out performance analysis of large computer and communication systems, based on numerical analysis of CTMC. In the early nineties the most common intermediate system description languages were *queuing networks* and *stochastic Petri nets (SPN)*. Queuing networks, whilst very powerful when applicable, have limited expressiveness and lack formal interpretation. SPN models have formal interpretation but do not have the explicit structure found in queuing networks, which greatly eases model construction. Process algebras offer both compositionality and formal interpretation, but from a performance perspective they lack the quantified timing information essential to derive performance estimates. Therefore PEPA associates an exponentially distributed delay with each activity of the process algebra, thus making it suitable for generating a CTMC.

The PEPA project has sought to investigate and exploit the interplay between process algebra and the underlying CTMC. Equivalence relations have played a fundamental role within this work. PEPA has been equipped with a number of different equivalence relations, which have been shown to be useful for a variety of purposes [11]. The most significant is *strong equivalence*, sometimes termed *Markovian bisimulation*, a bisimulation in the style of the probabilistic bisimulation defined by Larsen and Skou. This has been shown to form a *lumpable* partition on the underlying CTMC, meaning that if a new stochastic process is formed by associating one state with each partition, then it will be a CTMC. This makes the equivalence relation a suitable basis for model simplification based on aggregation (taking a quotient based on equivalence). Further work has sought, with some success, ways in which the compositional structure can be exploited for decomposed analysis of the underlying CTMC [12].

3. Applications

It is unwise practice to develop theory without checking it against real applications. PEPA has been applied to study the performance characteristics of a number of computer and communication systems. Initial examples focused on well-known standard performance evaluation abstractions such as *multi-server multi-queue* systems [11] and various queuing systems [15]. However, over time, more realistic case studies emerged, both from the PEPA group and from others. For example:

- In [5] the performance impact of fault-tolerant protocols within a distributed system framework is evaluated.
- A team from the PRiSM Laboratory at the University of Versailles considered a problem of dimensioning in a cellular telephone network. They used a PEPA model to study the impact on call blocking and dropping of allocating bandwidth resources between micro and macro-cell level [8].
- In an investigation of ways in which to ease the development of parallel database systems, the STEADY group at the Heriot-Watt University proposed the use of *performance estimators*. PEPA was used to verify the output of the performance estimators for a number of particular configurations and therefore improve confidence in the approach [7].
- In recent work a group at the PRiSM Laboratory of the University of Versailles are working on a novel active rule-based approach to active networks (networks in which intermediate nodes supplement routing of data with some computation). A PEPA model has been used to study the impact of the "*active*" traffic on the non-active cross-traffic in terms of loss rate and latency within an active switch [13].

4. Tool support

Case studies of the size and complexity described in the previous section are only possible if the modelling process has adequate support. The PEPA formalism is fortunate to be supported in a number of different tools offering a variety of different analysis techniques.

The PEPA Workbench: This tool provides a parser which can apply the operational semantics to derive the derivation graph capturing all possible evolutions of the model. It can render this as the infinitesimal generator matrix of a CTMC in formats suitable for both internal numerical solvers (biconjugate gradient algorithm or successive over-relation) and external numerical computing platforms such as Maple and Matlab. In addition it includes facilities to automatically derive some performance measures such as throughput, and a one-step debugger which can show the evolution of a model one activity at a time [9,4].

The Imperial PEPA Compiler (IPC): The recently developed IPC tool incorporates an alternative parser for PEPA models in such a way as to provide a bridge to alternative analysis tools, *Dnamaca* and *Hydra*, developed at Imperial College by Knottenbelt and his group [14,1]. Analysis of models by Hydra allows the computation of *passage-time quantiles* which detail the probability of passing through the system evolution from a start state to an end state (or set of starting states to a set of end states). This includes measures such as the probability that a 10-node cluster should be able to process 3000 database transactions in less than 6 seconds.

The PEPAroni simulation engine: Simulation has proved to be a useful alternative to numerical analysis of the underlying CTMC in two cases. Firstly, if the size of the model is prohibitively large for numerical analysis simulation can be used, although issues of run length can arise. Secondly, in the context of extending the expressiveness of PEPA with general distributions [2] numerical solution is no longer exact in most cases. Principally motivated by this second case, Clark implemented a simulator for PEPA models, called the PEPAroni simulator.

The PRISM model checker: This tool was developed by Kwiatkowska's group at the University of Birmingham. It supports discrete time Markov chains and Markov decision processes as well as CTMCs. The standard input to PRISM is a model described in a simple reactive modules language. PEPA was integrated into the tool via a compiler which translates PEPA models into this language. Integration into PRISM enables model checking of the CTMC underlying a PEPA model against properties expressed in Continuous Stochastic Logic (CSL) [15]. It also provides access to the efficient numerical solutions of PRISM based on MTBDDs [10] and sparse matrix representation.

The Möbius modelling platform: The Möbius modelling framework [3] was developed at the University of Illinois Urbana-Champaign. It is both a multi-formalism and multi-paradigm modelling tool, i.e. it aims to offer the user a choice of model description techniques and solution methods. Moreover it is designed to allow a model to be composed of submodels which may be expressed in different formalisms. Integrating PEPA into Möbius offered opportunities to explore the possibilities of interaction between modelling formalisms [6].

5. Conclusions

Markovian process algebras such as PEPA provide an expressive high-level language with enough structure to support the description of the large Markov chains needed for performance analysis of modern computer systems. They are amenable to formal analysis using both the logical and algebraic methods of classical process algebras and the analytical and numerical methods of performance models. All of the charms of more familiar process algebras are still here: the languages are small and defined formally by structured operational semantics; natural equivalences exist to relate processes which an external observer could not distinguish; and complicated processes may be put together as the composition of simpler parts.

If these formal approaches to performance evaluation such as PEPA bring about an increase in our capability to express more sophisticated designs then model simplification and approximate solution techniques must surely be destined to play increasingly important roles. One of the most promising aspects of the work on stochastic process algebras is the prospect that these techniques may be better understood by more users, leading to them being applied to ensure timely responses from a greater number of computer systems.

References

[1] J.T. Bradley, N.J. Dingle, S.T. Gilmore, and W.J. Knottenbelt. Extracting passage times from PEPA models with the HYDRA tool: A case study. In S. Jarvis, editor, *Proc. of 19th annual UK Performance Engineering Workshop*, pages 79–90, University of Warwick, July 2003.

[2] G. Clark. *Techniques for the Construction and Analysis of Algebraic Performance Models*. PhD thesis, The University of Edinburgh, 2000.

[3] G. Clark, T. Courtney, D. Daly, D. Deavours, S. Derisavi, J. M. Doyle, W. H. Sanders, and P. Webster. The Möbius modeling tool. In *Proc. of 9th Int. Workshop on Petri Nets and Performance Models*, pages 241–250, Aachen, Germany, September 2001.

[4] G. Clark, S. Gilmore, and J. Hillston. The PEPA performance modelling tools. In J. Hillston, editor, *Proc. of 7th Workshop on Process Algebra and Performance Modelling*, Zaragosa, Spain, September 1999. University of Zaragosa Press.

[5] G. Clark, S. Gilmore, J. Hillston, and M. Ribaudo. Exploiting modal logic to express performance measures. In *Computer Performance Evaluation: Modelling Techniques and Tools, Proc. of 11th Int. Conf.*, number 1786 in LNCS, pages 211– 227, Schaumburg, Illinois, USA, March 2000. Springer-Verlag.

[6] G. Clark and W.H. Sanders. Implementing a stochastic process algebra within the Möbius modeling framework. In *Proc. of 1st PAPM-PROBMIV Workshop*, volume 2165 of LNCS, pages 200–215, Aachen, Germany, September 2001. Springer-Verlag.

[7] E. W. Dempster, N. T. Tomov, J. Lü, C. S. Pua, M. H. Williams, A. Burger, H. Taylor, and P. Broughton. Verifying a performance estimator for parallel DBMSs. In *Proceedings of EuroPar (EuroPar'98)*, September 1998.

[8] J.M. Forneau, L. Kloul, and F. Valois. Performance modelling of hierarchical cellular networks using PEPA. *Performance Evaluation*, 50(2–3):83–99, 2002.

[9] S. Gilmore and J. Hillston. The PEPA Workbench: A Tool to Support a Process Algebra-based Approach to Performance Modelling. In *Proc. of 7th Intl. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation*, number 794 in LNCS, pages 353–368, Vienna, May 1994. Springer-Verlag.

[10] H. Hermanns, J. Meyer-Kayser, and M. Siegle. Multi-terminal binary decision diagrams to represent and analyse continuous time markov chains. In *Proc. of 3rd Intl. Workshop on the Numerical Solution of Markov Chains*, pages 188–207, 1999.

[11] J. Hillston. *A Compositional Approach to Performance Modelling.* Cambridge University Press, 1996.

[12] J. Hillston. *FMPA Lecture Notes*, chapter Exploiting Structure in Solution: Decomposing Composed Models. Springer-Verlag, 2001.

[13] J. Hillston, L. Kloul, and A. Mokhtari. Active nodes performance analysis using PEPA. In *Proc. of 19th UK Performance Engineering Workshop*, pages 244–256, 2003.

[14] W.J. Knottenbelt. Generalised Markovian analysis of timed transition systems. Master's thesis, University of Cape Town, 1996.

[15] M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic symbolic model checker. In *Proc. of 12th Int. Conf. on Modelling Tools and Techniques for Computer and Communication System Performance Evaluation*, number 2324 in LNCS, pages 200–204, London, UK, April 2002. Springer-Verlag.

[16] N. Thomas and J. Hillston. Using Markovian process algebra to specify interactions in queueing systems. Technical Report ECS-LFCS-97-373, LFCS, The University of Edinburgh, 1997. ■

FACS FACTS Issue 2005-2

Call For Submissions

Deadline **13 May 2005**

We welcome contributions for the next issue of *FACS FACTS*, in particular:

- Letters to the Editor
- Conference reports
- Reports on funded projects and initiatives
- Calls for papers
- Workshop announcements
- Seminar announcements
- Formal methods websites of interest
- Abstracts of PhD theses in the formal methods area
- Formal methods anecdotes
- Formal methods activities around the world
- Formal methods success stories
- News from formal methods-related organizations
- Experiences of using formal methods tools
- Novel applications of formal methods
- Technical articles
- Tutorials
- Book announcements
- Book reviews
- Adverts for upcoming conferences
- Job adverts
- Puzzles and light-hearted items

Please send your submissions (in Microsoft Word, LaTeX or plain text) to Paul Boca [editor@facsfacts.info], the Newsletter Editor, by **13 May 2005**.

If you would like to be an official *FACS FACTS* reporter or a guest columnist, please contact the Editor.

Formal Techniques Industry Association – ForTIA

Subgroup of Formal Methods Europe

INDUSTRY DAY 2005

PRELIMINARY ANNOUNCEMENT

Wednesday 20 July 2005

University of Newcastle-Upon-Tyne (UK)

Formal Methods Europe [www.fmeurope.org] is an independent association whose aim is to stimulate the use of, and research on, formal methods for software development. FME has been notably successful in bringing together innovators and practitioners in precise mathematical methods for software development, industrial users as well as researchers.

A separate subgroup of FME was recently established, called ForTIA (Formal Techniques Industrial Association), which focuses on dissemination of research results and promotion of formal techniques in industry and also to feed lessons learnt from industry back towards academia. ForTIA was founded in 2003 and is currently supported by some 30 companies (for more details check out www.fortia.org). One of the key activities of ForTIA is to organize so-called Industry Days, small events targeted at industrial practitioners rather than academics. It is a forum to share experiences on the use of formal techniques in an industrial setting. We invite you to participate in this event.

The theme for I-Day 2005 is: "*Formal Methods Going Mainstream – Costs, Benefits and Experiences*". A very provocative title, which was selected on purpose; we invite you to challenge it! Potential sub-topics will be:

- Formal techniques successfully applied to ensure quality of product
- Formal techniques successfully applied to improve efficiency of development
- Failure of formal approaches, especially why they did not work
- Impact of formal approaches on the development process
  (managerial and other less-tangible aspects)
- How do I select the right tool for the job?
  Which (formal) method to use for what kind of problem?

The day will consist of a number of invited talks – we are in the process of completing the program. It will be organized as a parallel track to the main FM'05 symposium. The provisional program will be announced as soon as possible. If you have any questions or suggestions for I-Day at FM'05, please contact the organizing committee:

Bernard Schätz [schaetz@in.tm.de]
Marcel Verhoef [Marcel.Verhoef@chess.nl]

Details on the I-Day program and registration will be published on the FM'05 conference website [http://www.csr.ncl.ac.uk/fm05] as soon as it becomes available.

Z in Hol in ProofPower

Rob Arthan & Roger Bishop Jones

1. Introduction

The development of tools to assist with mathematical activities is a subject with a long and fascinating history. Since the earliest days of electronic computation, researchers have endeavoured to produce mechanised tools to assist with the development of formal mathematical theories. The possibility of using formal mathematics to improve the rigour of hardware and software systems design has spurred this research. The result has been the development during the past 20 years or so of practical systems to support rigorous formal mathematics and its applications.

ProofPower is one such system for mechanised formal mathematics. It is a tool supporting formal specification in Higher Order Logic (HOL). It also supports, by semantic embedding [6] and other techniques, various other languages, including the Z specification language. **ProofPower** is a proof tool following the LCF paradigm [8]. It was originally conceived as a re-engineering of the Cambridge HOL system. Cambridge HOL was developed by Mike Gordon *et al.* [5, 7] for research into hardware verification and was based on Cambridge LCF, itself derived (mainly by Larry Paulson [10]) from Edinburgh LCF.

As well as syntax- and type-checking specifications, and managing a hierarchy of theories in which these specifications are stored, **ProofPower** provides the user with a high level functional programming language (Standard ML [11]) for constructing (and checking, on-the-fly) proofs in its core logic. The HOL logic is a polymorphic variant (following Robin Milner [15]) of the elegant and simple reformulation by Alonzo Church [4] of the *Theory of Types* [16] which Russell devised for use in Principia Mathematica [1]. HOL is well-suited not just to hardware verification but to almost any practical problem domain in which mathematics has a part to play.

The combination of this logical strength in the object language, and the power of the metalanguage, has resulted in a global user community for the Cambridge HOL system and its descendants (including **ProofPower**). This community is applying the tools to a range of applications which was never at first imagined.

ProofPower combined from its earliest days these important elements of the LCF/HOL line of research with the somewhat different subculture of formal methods originating in Oxford and embodied in the Z specification language [18, 9]. Many of the distinctive features of **ProofPower** arise from creative friction between these two formal methods ideologies. Hard experience of the use of formal methods in industry added further elements. The end result is a tool which, while bearing the marks of its intellectual history, is unique in its capabilities.

This article gives an overview of some aspects of the design of **ProofPower** which have not previously been widely reported. In particular, we discuss our original motivation for using HOL to underpin work in Z, the impact this had on our re-engineering of an HOL support tool and our use of a semantic embedding in HOL to support Z. In addition to any historical interest it may have, we hope the article will stimulate further interest amongst formal methods practitioners in practical tools for formal mathematics.

2. Beginnings

The original motivation for developing **ProofPower** was in information security. During the 1980s, the US government had set out guidelines for the development of secure systems which included the use of mathematical methods in their design and implementation [12]. The UK was the first country to follow the US with its own national certification standards, which were later superseded by harmonised European standards. In 1985 International Computers Ltd. (ICL) established a Formal Methods Unit within its Defence Technology Centre to build a capability to meet the evolving UK standards, which at the highest levels required systems to be formally modelled and for critical properties of the models to be formally proved.

Z had been identified by UK government as the preferred formal specification language for work on secure systems and so familiarisation with Z was an early priority, as was gaining experience in the proof technologies which seemed most promising for the kind of work expected. Experience was gained with two proof tools, NQTHM (otherwise known as the Boyer-Moore prover) and Mike Gordon's Cambridge HOL system, both of which had also been used at RSRE, Malvern for the formal treatment of digital hardware.

Of these two the Cambridge HOL system seemed to offer the best prospect of success in reasoning about specifications in Z, in default of any proof tool for Z itself. The main factors in this assessment were:

> **the HOL language**: this is much closer in logical expressiveness to Z than was the restricted first order logic of NQTHM.
>
> **the LCF paradigm**: this gives the user a powerful programming language for programming proofs and adapting and extending the system.

To reason formally about systems specified in Z, the method adopted at ICL prior to **ProofPower** was to transcribe the specifications into HOL keeping the semantic correspondence (even at a low level) as close as possible. Proofs were then undertaken in HOL. Success stories with this technique included the formal aspects (specifications and proofs) of a communications device. The end product was certified at the highest level and was for many years the only system to have achieved this level of certification.

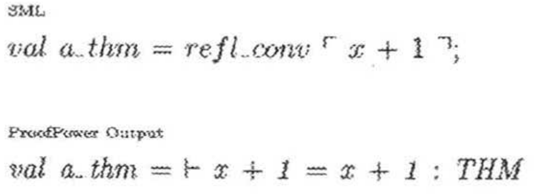
Cambridge HOL provided no special facilities for document preparation, expecting ASCII specification and proof scripts to be treated just like computer programs. Z on the other hand had a culture of presenting specifications embedded in a natural language document using mathematical symbols and special graphical layouts. We implemented a Z-like notation as a presentation layer for Cambridge HOL with a custom font and a couple of UNIX filters providing the GUI and with `troff` (and later LaTeX) providing typeset documents.

3. Re-engineering HOL

In 1990, ICL began a programme of research in collaboration with Program Validation Ltd. and the Universities of Cambridge and Kent. While proof support for Z was high on our list of priorities, we believed that the best route to that goal was via the logically simpler HOL language. We therefore concentrated at first on a re-engineering of the HOL system to address some of the problems that we had encountered in high-assurance applications and to try to ease the path towards a proof tool for Z. The resulting re-engineered HOL system became the basis for the **ProofPower** tools and we discuss some of the salient features of the re-engineering below.

3.1 The Logical Kernel

ProofPower is based on the "LCF paradigm", in which theorems are proved by computing values of type *THM*. This type is an abstract data type in a strongly typed functional programming language - in our case Standard ML [11]. The constructor functions of the type *THM* implement the primitive inference rules of the logic, suitably parameterised to make them into functional relations. For example, there is a rule *refl_conv* implementing reflexivity of equality, which we can see in the computation of a value of type *THM* in the example below. This shows the Standard ML (SML) input and corresponding output from an interactive session with **ProofPower**:



SML

```
val a_thm = refl_conv ⌜ x + 1 ⌝;
```

ProofPower Output

```
val a_thm = ⊢ x + 1 = x + 1 : THM
```

The great advantage of the LCF approach is that proof procedures of arbitrary power and complexity can be programmed freely. If a value of type *THM* is calculated, it can only have been obtained by composing primitive inference rules and therefore must be a theorem of the logic. An error in a complex proof procedure can result in no theorem at all being computed or in the wrong theorem being computed, but it cannot result in an unsound theorem. Thus the abstract data type provides a *logical kernel* in which is concentrated all the code that is critical to the soundness of the tool.

Our implementation of the logical kernel is underpinned by a formal specification of the HOL language, semantics and deductive system [2]. This supports a formal specification of the critical properties that the logical kernel must satisfy to be sound. The implementation of the logical kernel was then carried out against a formal design articulating the mechanisms used to meet the critical properties.

3.2 The Theory Hierarchy

For theorem proving in the context of large specifications some way of structuring the logical context is desirable, and in LCF-like tools this is done through a hierarchy of "theories", which in this context are a kind of hybrid between the logician's notion of a theory and the computer scientist's notion of a module. Each theory provides a collection of zero or more axioms and definitions together with a record of the theorems that the user has chosen to prove and save in the theory. Theories are organized in a hierarchy allowing modular access.

The Cambridge HOL system and its precursors offered very limited facilities for navigation in the theory hierarchy or for making changes. Modifications to axioms or definitions would generally require rebuilding theories from modified scripts in a new session. **ProofPower** was designed to offer more flexibility both in navigation and in modifying the theory hierarchy.

3.3 Document Preparation

It is the Z style of literate formal specification which determined that **ProofPower** is oriented around processing of formal texts extracted from LaTeX documents. The document processing facilities are steered by tags which introduce the different kinds of formal material which may be included in a document. The same document may be processed in different ways for different purposes, e.g., to convert it into LaTeX to be typeset or to extract a formal specification for type-checking. We have found this "document-centric" way of working extremely convenient and found it surprising that so many users of proof tools developed in academia continue to carry out their work with ASCII text that has to be manually transcribed into mathematical notation when a high-quality document is needed.

3.4 Multiple Object Languages

Terms in the HOL language are represented using an abstract data type *TERM* whose constructor functions correspond to the primitive abstract syntax constructors of the language. Like other LCF-style systems, ProofPower ML supports quotations allowing object terms to be entered and parsed using a convenient concrete syntax. The pretty printer for ML values then formats object language terms as concrete syntax for display.

The **ProofPower** syntax uses an extended character set for mathematical symbols (which are also available for forming ML names). These features may be seen in the following example in which a function application term is calculated using quotations for the function (Sin) and its operand ($m * \pi$). The "Quine corners" (⌜ and ⌝) here are the symbols that delimit the quotations. The pretty printer responds by displaying the term using the concrete syntax that we could have used to enter it in the first place:

SML

```
val a_tm = mk_app(⌜ Sin ⌝, ⌜ m * π ⌝);
```

ProofPower Output

```
val a_tm = ⌜Sin (m * π)⌝ : TERM
```

In Cambridge HOL, the quotation facilities were restricted to HOL terms, types and to embedded ML expressions (called anti-quotations). This was generalised in **ProofPower** to a system supporting general mixed language parsing and pretty-printing. This was primarily intended for Z, but has been used experimentally for various other purposes. Multiple languages, e.g., HOL and Z, can be mixed together in a single term quotation, fragments of HOL being included inside Z or vice-versa.

The system automatically maintains an association of HOL constants and object languages. The pretty printer uses this to decide how to process different parts of a polyglot term. Mixed language quotations are not normally encountered in working with Z specifications and proofs since the proof facilities and other tools are generally smart enough to keep everything in Z. However mixed language quotations are very useful both for programming proof procedures and, as we shall see in this article, for peering under the covers to see how an object language like Z has been implemented.

3.5 The Subgoal Package

As we have described it above, the LCF style implements forward proof: the inference rules are functions that compute new theorems from old. A goal-oriented approach working backwards from the conjecture one wishes to prove is usually much more natural and productive. In common with other systems in the LCF family, **ProofPower** provides a "subgoal package" supporting this way of working, following ideas of Paulson.

The user starts the search for a proof by passing to the subgoal package the desired conjecture. At each step in the proof search which follows the subgoal package presents to the user a current goal and the user then nominates a "tactic" to be applied to this goal. If the tactic is successful, it responds with some subgoals, i.e., a possibly empty list of new goals. If the list is not empty, the subgoal package presents the first goal in the list as the new current goal. If the list is empty, the tactic has completed the proof of the current goal and the subgoal package will move on to the next goal if any remain.

For example, consider the goal:

$$1/2 \in \{y \mid 1/4 < y\} \cap \{x \mid x\hat{}3 < x\hat{}2\}$$

This kind of goal is amenable to a widely used tactic called *strip_tac* whose (configurable) approach is to do the logically obvious and uncontroversial thing. In this case, applying *strip_tac* once would reduce the problem to the goal:

$$1/2 \in \{y \mid 1/4 < y\} \land 1/2 \in \{x \mid x\hat{}3 < x\hat{}2\}$$

Repeated applications of *strip_ tac* might produce the two subgoals:

$$1/4 < 1/2$$
$$(1/2)\hat{}3 < (1/2)\hat{}2$$

These subgoals can readily be proved e.g., using a standard rewriting tactic.

Behind the scenes, the tactics provide the subgoal package with the inference rules it needs to calculate the theorems corresponding to the goals. A library of combinators (called *tacticals*) for creating new tactics from old, e.g., by composition, iteration and alternation, is provided. These give a powerful language for conducting proof searches.

Tactics are not infallible, not only may they fail to offer a step in the desired proof at all, they may make a promise which they later fail to keep. I.e., the inference rule the tactic provides may fail to do the job when the subgoals are all discharged. It is very inconvenient to discover an error in a tactic only at the point of apparent completion of a large proof attempt. The subgoal package in **ProofPower** was designed to be immune to this problem. Instead of remembering a tree of subgoals and proof functions the state of the subgoal package is coded up as a theorem in which the assumptions are (codings of) the outstanding subgoals, and the conclusion is a coding of the target goal (a constant is used which mimics the semantics of the sequent turnstile). The construction of this subgoal package state theorem involves invocation of the proof function at the same time as the tactic is invoked so that its failure is detected immediately.

3.6 Proof Contexts

"Proof contexts" are a general mechanism for making the proof support sensitive to a context in which reasoning is being conducted. The "context" referred to here might be a particular theory, say a theory of arithmetic, or a library of theories, say the **ProofPower-HOL** library of Z-like operators on sets, functions and lists, or the support for a complete embedded language and its libraries such as Z and the Z toolkit.

A proof context is essentially a package of parameters to the various proof tools; selection of proof context can have radical effects upon the behaviour of the proof facilities. For example, in the example of the previous section it is the proof context that caused a goal of the form $x \in A \cap B$ to be reduced to the conjunction $x \in A \land x \in B$. In fact, the proof of the example goal

$$1/2 \in \{y \mid 1/4 < y\} \cap \{x \mid x\hat{}3 < x\hat{}2\}$$

is completed in one fully automatic step by the standard rewriting tactic in the right proof context.

The proof context idea combined with careful design of the basic tactics has proved very successful. For example, *strip_tac* is derived from what appeared to be an attempt in the early development of LCF to provide a tactic which knows the basic natural deduction rules for the predicate calculus and automatically applies the rule relevant to the current goal, provided that this can be done without extra information (such as an existential witness). If this idea is thought through it becomes apparent that a *strip_tac* can be implemented which is complete for the propositional calculus (i.e., repeated application of the tactic will prove any propositional tautology). By parameterisation via the proof context, *strip_tac* serves as a decision procedure for the propositional fragment of many useful theories and as a valuable general purpose tool in every problem domain, including embedded languages such as Z.

4. Z in HOL

In essence, Z is just a polymorphic typed set theory, and, as such, it is logically equal in strength to polymorphic simple type theory, i.e., HOL. So in principle, at least, one can interpret Z in HOL. The challenge is to devise an interpretation which works well in practise, i.e. which can be implemented in a proof tool to yield effective support for proof in Z.

The kind of interpretation we have in mind has become known as a *semantic embedding*. A semantic embedding of one language into another treats the first language as if it were an alternative syntax for certain expressions in the second. The idea is that some capability in respect of the second language is thereby transferred to the first. In this case the primary capability of interest is proof construction and checking.

The advantages of providing proof support via an embedding are substantial. One important advantage is that by this means proof capability can be realised where the semantics are known but the proof rules are not. Soundness of inference is guaranteed by the soundness of the proof system for the target language, provided that the embedding is semantically correct. A second advantage is that sharing between the languages of that most precious and costly commodity, theorems, is maximised. Thus a theory of integer or real numbers developed for HOL will be substantially re-usable in Z, together with domain specific proof automation such as a linear arithmetic prover.

Both Z and HOL are typed languages. The semantic embedding we use respects an injection of the type system of Z into that of HOL. This ensures that the embedding preserves many pleasant properties of the Z language. We will begin our discussion of the embedding by describing this injection.

4.1 The Type Injection

The Z type system is built from ground types (called *given sets*) using the power set type constructor, and a labelled product type constructor called the *schema* type constructor. Unusually for a typed language, Z has no syntax for writing down a type. One works round this by writing down a set-valued expression for the set that comprises the universe of the type. A signature (i.e., a list of variable declarations) in square brackets is used to express the set of values in a schema. Thus, for example, $[x : \mathbb{Z}; y : \mathbb{P}\mathbb{Z}]$ can be used to denote the schema type with components labelled x and y ranging over integers and sets of integers respectively.

The main difficulty in mapping the Z type system into that of HOL is caused by the schema type constructor, since its parameter is a finite map from component names to component types, whereas type constructors in HOL take a finite sequence of types. To solve this problem, we use an infinite family of type constructors in HOL to represent the schema type constructor. The names in the signature of the schema are encoded into the names of each of the HOL type constructors in a canonical order. The types associated with the names in the signature are then passed as parameters to the HOL type constructor in the corresponding order.

Z has a number of built-in operators on values of schema type. These operators cannot be assigned a type in the Z type system, not even a polymorphic or generic type. They act as if they have a family of types indexed by the signatures of the operands. These operators are represented in HOL as infinite families of HOL constants. In the implementation, the definitions of the labelled product types that represent schema types and of the constants that represent schema operators are introduced automatically when the Z constructs that need them are translated into HOL.

In addition to its schema types, the Z concrete syntax also has unlabelled product types, called *tuple types*. Borrowing a trick from Standard ML, the types of n-tuples in Z can, at least in principle, be treated as schema types with component names given by the natural numbers $1,2,\ldots,n$. (The implementation actually uses a somewhat more optimal representation.)

The power set constructor is easily represented in HOL, sets being represented by boolean-valued functions. The types of given sets in Z are represented by nullary HOL type constructors. The predefined given sets such as the integers are represented by corresponding HOL types for which the relevant theory has been developed.

So far we have discussed monomorphic types in Z. Z also has *generic* values, whose types are therefore *generic* types. Polymorphic values in HOL are values whose types contain type variables. These values can be instantiated by substituting types for the type variables. Generic values in Z are values parameterised by sets, not merely values polymorphic over type variables. Generic values are represented in HOL as polymorphic functions. For example, the generic value $X \rightsquigarrow Y$ defined in the Z toolkit has generic type $\mathbb{P}\mathbb{P}(X \times Y)$ with generic formal parameters X and Y. The HOL representation of the generic value is the (HOL) function that maps a pair of sets (X, Y) to the set of all (Z) functions from X to Y. The HOL representation of the generic type is then just the HOL type of that function.

4.2 Mapping Predicates and Expressions

The terms "predicate" and "expression" in Z correspond in the usual terminology of first order logic to "formula" and "term" respectively.

In the **ProofPower** embedding of Z into HOL, predicates are represented by HOL terms of type $BOOL$ and expressions are represented by terms whose type is determined from the type of the Z expression by the type injection described in the previous section.

Generally speaking, each constructor in the abstract syntax of Z is represented by a HOL constant or family of constants. Each so-called *semantic constant* must correctly capture the semantics of the relevant constructor. Where appropriate, e.g., for equality, a predefined HOL constant with the appropriate semantics is used (which helps with reuse of proof machinery). As discussed in the previous section, families of semantic constants are used for Z constructs such as schema operators, which are too general to be represented by an individual polymorphic constant.

In essence, the semantic constants comprise a many-sorted algebra over HOL terms isomorphic to the algebra that comprises the abstract syntax of Z. The semantic embedding can then be thought of as a homomorphism defined by primitive recursion over the abstract syntax of Z.

However, there are important features of the Z notation which complicate the semantic embedding. The following sections discuss some of the more significant problems and our solutions to them.

4.3 Undefinedness in Z in HOL

ProofPower adopts the simplest treatment of 'undefinedness' in Z which is consistent with the semantics in the Z standard [9]. For uniformity, we fix on definite description as the primitive source of undefinedness. We define it as one might in pure first order set theory by giving a *loose* definition of definite description, in which the definite description operator always returns a value whether or not the description is uniquely satisfied, but the value returned is not known (i.e., not required to be the same in all models) if the description is not uniquely satisfied. This approach avoids the need to introduce a special value for undefined terms and bypasses all the complications to the logic which such values involve.

Function application is the other Z construct that can introduce undefinedness and this is defined using definite description. The defining properties of the semantic constants $Z'\mu$ and $Z'App$ that represent definite description and function application respectively are as follows:

$$\forall\ x \bullet\ \$"Z'\mu"\ \{x\} = x$$

$$\forall\ f\ x \bullet\ Z'App\ f\ x = \$"Z'\mu"\ \{y | \ulcorner_{z}(x,\ y)\urcorner \in f\}$$

Here the special syntax $\$"Z'\mu"$ is used to allow the character string $Z'\mu$, which does not conform to the HOL rules for identifiers, to be used as the name of an HOL constant. In general, the names of the semantic constants in the Z embedding follow a naming scheme designed to avoid clashes with the user's HOL and Z name spaces.

Note that these defining properties are theorems expressed in the HOL language. The second theorem contains a nested Z quotation, since, as we have mentioned above, Z tuples are represented as a special kind of schema type and not as HOL pairs.

4.4 Hidden Variables

The denotation of a schema expression used as an expression in Z is a set of labelled tuples (called "bindings" in the Z terminology). For example, the schema expression: $[x,\ y : \mathbb{Z}\ |\ x\ <\ y]$ denotes the set of all bindings such as $(x \mathrel{\widehat{=}} 2,\ y \mathrel{\widehat{=}} 3)$ with components labelled x and y, the former being smaller than the latter. In the schema expression: $[x,\ y : \mathbb{Z} \mid x < y]$ the signature variables x and y are not free variables, just as they are not free variables in the set comprehension: $\{x,\ y : \mathbb{Z} \mid x < y\}$.

An unusual linguistic feature in Z comes from the ability to use a schema as an abbreviation for the predicate it embodies. When a schema is used in this way, the effect is exactly that of a predicate in which the names in the signature of the schema appear as free variables. Thus $[x,\ y : \mathbb{Z} \mid x < y]$ as a predicate is equivalent to the predicate $x < y$ in which x and y appear free.

Now all of the above applies to arbitrary schema-valued expressions, and in particular, to what are traditionally called *schema references*, i.e., the names of schema-valued constants. Thus the names in the signature, which become the free variables of the predicate, will not, in general, appear in the schema-valued expression itself: they are hidden away in the invisible type of the expression. These variables must be made explicit when the schema-as-predicate is represented in HOL. Schema-valued expressions may also be used as declarations in Z, and again the representation in HOL must make explicit what is being declared.

All occurrences of hidden variables in Z predicates or declarations are reducible to occurrences of what Z calls θ-expressions: $\theta[x,\ y : \mathbb{Z}]$ being a linguistic abbreviation for the binding: $(x \mathrel{\widehat{=}} x,\ y \mathrel{\widehat{=}} y)$ in which each component name is associated with the expression formed by the free variable of the same name. A schema-valued expression used as a predicate can then be treated as an ellipsis for the assertion that a θ-expression is a member of the schema, i.e., we can treat S used as a predicate as the assertion that $\theta S \in S$. A similar approach may be used for declarations.

The problem of hidden variables in general therefore reduces to the problem of hidden variables in θ-expressions. This is dealt with by using knowledge of the type of the θ-expression to represent it as an expression which is semantically identical with the binding that the θ-expression abbreviates but marked with a semantic constant which causes the pretty printer to preserve the concrete syntax.

4.5 Variable-binding Constructs

It is unfortunate that Z terminology uses the term "binding" for its labelled tuples, since the use of the term in its normal logical sense is unavoidable. So the reader is warned that the term "binding" will be used in two senses. In this article we will say "variable-binding" for the normal logical use of 'binding' i.e. for what a quantifier (or other variable-binding construct) does to the variable(s) that it declares and plain "binding" for an element of a Z schema type.

Rather than just one variable, the variable-binding constructs in Z all admit an arbitrary signature, including set constraints and possibly schema-as-declarations. These must all be represented in HOL, in which there is only one variable-binding construct (λ-abstraction) which binds a single variable subject only to a type constraint.

The HOL representation of any variable-binding construct in Z must include a nested HOL λ-abstraction to bind each of the names which are bound explicitly or implicitly by the Z construct. In general, the representation of the Z construct will then comprise a suitable semantic constant which will take the λ-abstraction as one of its operands. In the body of this λ-abstraction will appear the translation of all the Z which is in the scope of the Z variable-bindings (since this is the only way that the HOL variables representing the Z variables can become bound in Z).

The λ-expression construct in Z provides an informative example. We will now work through the representation of this construct in more detail. We do this using **ProofPower** itself to interrogate the representation of a Z λ-expression into HOL. To begin our investigations, we first enter an example of a Z λ-expression, and give it an ML name as shown below:

SML

```
val zle = ⓩλ x:ℕ; y:ℤ | x > 1 • y * x⌝;
```

ProofPower output

```
val zle = ⓩλ x : ℕ; y : ℤ | x > 1 • y * x⌝ : TERM
```

The output just tells us that the ML compiler has given *zle* the value we entered. The pretty printer for Z recognises this value as a Z term and so prints it back in Z. This is, of course, just what we want when using **ProofPower** to reason in Z, but not so helpful for us here, since we want to look under the covers.

Before delving deeper, we should remind ourselves of the intended meaning of the Z λ-expression. The expression comprises three parts: (i) a declaration (here $x : \mathbb{N} ; y : \mathbb{Z}$), (ii), a predicate (here $x > 1$) and, (iii), an expression (here $y * x$). The λ-expression denotes a function. The declaration and the predicate together determine the domain of the function, which in this case is a set of pairs of integers of which the first must be greater than 1. The value of the function at a point in the domain is given by the value taken by the expression at that point.

One excellent way of exploring the meaning of a Z construct in a proof tool is by carrying out proofs! Forward proof is particularly useful for this kind of purpose. The proof support for Z in **ProofPower** includes a large number of forward proof rules for operating on specific Z constructs and, in particular, for expressing "derived" constructs in terms of more primitive ones. These are primarily intended for people programming new proof procedures, but they are also useful for exploratory purposes and in interactive proof work. Here, the proof rule *z_ λ _conv* automatically proves for us a theorem which explicates our λ-expression as equal to the set-comprehension that was implicit in our informal description of the meaning of the expression given above:

SML

```
z_λ_conv ⓩλ x:ℕ; y:ℤ | x > 1 • y * x⌝;
```

ProofPower output

```
val it =
   ⊢ (λ x : ℕ; y : ℤ | x > 1 • y * x)
       = {x : ℕ; y : ℤ | x > 1 • ((x, y), y * x)} : THM
```

In general, a λ-expression in Z has three constituents (some of these may be omitted in the concrete syntax, in which case the language defines default values for them). The three constituents together define the function denoted by the λ-expression and are as follows:

(d) a declaration part introducing variables to be used in specifying the values of the function.

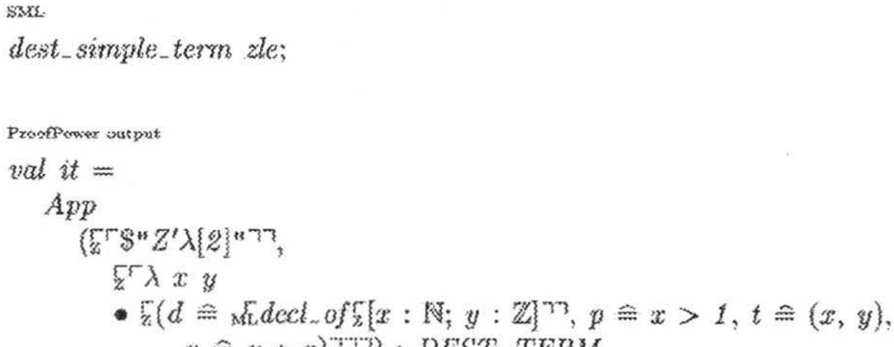
(p) a predicate giving constraints on the domain of the function (additional to any constraints implicit in the sets appearing in the declaration: e.g., in the example above, x is required to be both greater than 1 and a member of N).

(b) an expression giving the value of the function.

These constituents are combined as a Z binding into a single value in the body of a HOL λ-abstraction. The binding also includes a component giving a pro forma, called the *characteristic tuple* in Z parlance, for a parameter to the function denoted by the Z λ-abstraction. The component names are the letters used above in listing the explicit constituents, together with the letter "t" for the characteristic tuple.

The semantic constant maps the HOL λ-abstraction into a set of ordered pairs giving the graph of the required function (as required by the Z conventions for representing functions). An ordered pair *p* will be a member of this set iff. there exists an assignment of values to the bound variables which, when supplied to the function, gives a binding whose b component is the second element of p, whose t component is the first element of p and whose d and p components are *true*.

To see how this works, let us dismantle our example Z λ-expression. To do this we look at the translated Z expression in terms of the primitive HOL term constructors. The function *dest_simple_term* is convenient for this:



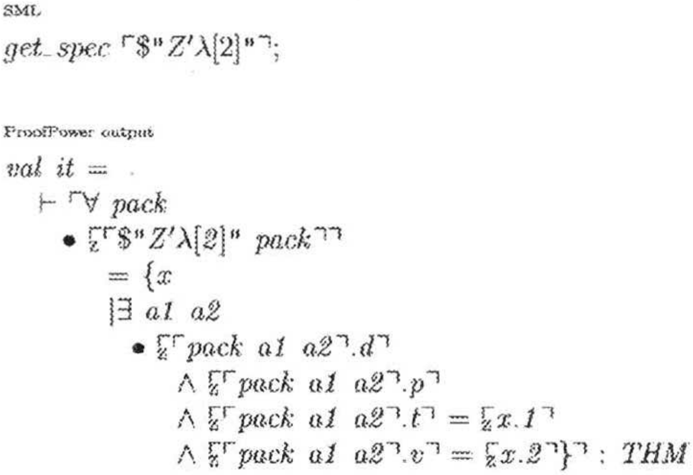
```
SML
dest_simple_term zle;


ProofPower output
val it =
   App
      (ⓩ⌜$"Z'λ[2]"⌝⌝,
        ⓩ⌜λ x y
         • ⓩ(d ≙ ⓜⓩdecl_ofⓩ[x : ℕ; y : 𝕫]⌝⌝, p ≙ x > 1, t ≙ (x, y),
              v ≙ y * x)⌝⌝⌝) : DEST_TERM
```

The constructor *App* here indicates that the term is a function application. The arguments of the constructor give the function and the argument to which it was applied. The pretty printer starts out trying to print these term arguments in Z but drops into HOL when it finds a subterm which does not appear to be in the image of the Z embedding, so inside the Z metaquotes we find some nested HOL metaquotes.

The function in the function application is the semantic constant which captures the semantics of a λ-expression over a signature with two names in it. The constant is called "Z'λ[2]" (as already remarked, a name like this sometimes has to be wrapped in quotes and paid a dollar when used as an identifier in HOL concrete syntax).

The parameter in the function application is the HOL λ-abstraction described above. In its body, we see a Z binding display. The binding display is nearly all Z, except for one component, which is the declaration part of the λ-expression. A declaration is not syntactically valid as an expression in Z, and so cannot appear in a binding display in Z. Furthermore, the Z parser will only accept in Z metaquotes a Z predicate (formula) or a Z expression. So the pretty printer uses a trick here to display the declaration part. It uses an ML expression to extract the declaration part from a horizontal schema expression which contains the required declaration.

To see how the representation of the Z λ-expression works semantically we need to look at the definition of the semantic constant called "Z'λ[2]" . We can retrieve the definition for this constant using the following ML command:



SML

```
get_spec ⌜$"Z'λ[2]"⌝;
```

ProofPower output

```
val it =
   ⊢ ⌜∀ pack
      • ⓩ⌜$"Z'λ[2]" pack⌝⌝
         = {x
         |∃ a1 a2
            • ⓩ⌜pack a1 a2⌝.d⌝
               ∧ ⓩ⌜pack a1 a2⌝.p⌝
               ∧ ⓩ⌜pack a1 a2⌝.t⌝ = ⓩx.1⌝
               ∧ ⓩ⌜pack a1 a2⌝.v⌝ = ⓩx.2⌝}⌝ : THM
```

This is a definition in HOL including some nested fragments of Z. It tells us that the value of a Z λ-expression described by the "package" of constituents pack. This package is actually a function (the HOL λ-abstraction described above whose arguments (*a1* and *a2*) correspond to the variables declared in the Z λ-expression. The value in HOL that represents the Z λ-expression is the set comprising precisely those ordered pairs *x* such that:

- the predicate implicit in the declaration of the Z λ-expression is true
- the predicate part of the Z λ-expression is true
- the 'tuple' expression is equal to the first element of *x*
- the body of the λ-expression is equal to the second element of *x*

which is just what the value of the Z λ-expression should be.

4.6 The Z Mathematical Toolkit

The Z Mathematical Toolkit gives an immediate and fruitful source of early challenge problems to test the effectiveness of the proof tools. An exposition of the **ProofPower** treatment of the Toolkit is given in [3]. For the present article, we just mention a highlight or two.

The Z Toolkit follows "plain old-fashioned mathematics" in modelling lists, which Z calls "sequences", as functions, a list of n elements being a function whose domain is the integer interval $1..n$. The definition of the set $seq\ X$ of sequences of elements of a set X is given as:

$$seq\ X = \{f : \mathbb{N} \mathrel{⇻} X \mid dom\ f = 1 \mathbin{..} \#\ f\}$$

Here $\mathbb{N} \mathrel{⇻} X$ denotes the set of finite partial functions from $\mathbb{N}$ to X and $\#\ f$ denotes the size of the finite partial function f. This is evidently not at all like a computer scientist's view of lists as defined by an induction principle. It is entirely equivalent to something like:

$$seq\ X = \bigcup \{n : \mathbb{N} \bullet 1 \mathbin{..} n \rightarrow X\}$$

which makes no mention of finiteness or sizes of finite sets and is much easier to work with. However, the Z Toolkit says what it says and to get started on reasoning about sequences, you need to sort out the basics of finiteness and the size function. You can view this as an irritation or as a challenge. Accepting the challenge means you have to work through some of the most basic mathematics there is. Done in the right frame of mind, this can be quite entertaining.

Again, Z follows plain old mathematics in defining finite sets to be the images of integer intervals. It is indispensable to prove the finite set induction principle (if $p(a)$ holds when a is the empty set, and if $p(a \cup \{\ x\ \})$ holds provided $p(a)$ holds, then $p(a)$ holds for all finite a). From which follow elementary combinatorial principles such as:

$$\begin{array}{l} \forall\ a, b : (\mathbb{F}\ \_) \bullet \\ \quad a \cup b \in (\mathbb{F}\ \_) \land \#\ (a \cup b) + \#\ (a \cap b) = \#\ a + \#\ b \end{array}$$

$$\forall\ a : (\mathbb{F}\ \_) \bullet \#\ a = 0 \Leftrightarrow a = \{\}$$

Whence the following gem:

$$\forall\ u : \mathbb{F}\ (\mathbb{F}\ \_) \mid \#\ (\bigcup u) > \#\ u \bullet \exists\ a : u \bullet \#\ a > 1$$

which is the pigeon-hole principle (u being the finite set of finite sets representing the sets of letters grouped by pigeon-hole). The proof is by finite set induction on u. Its realisation in **ProofPower** takes 18 tactic applications, generating about 3,400 primitive inferences and executing in about 90 milliseconds on a commodity desktop PC.

5 Subsequent Work

The support for Z via the semantic embedding approach has proved remarkably successful. A very important test of the power and extensibility of the technique was the development of the Compliance Tool (or "DAZ") component of the **ProofPower** suite, a verification system for Ada programs using Z as a specification language. The approach is based on ideas of Chris Sennett [17].

The Compliance Tool was designed and implemented against a specification written in Z by the Defence Research Agency (now QinetiQ) [13]. The specification was subsequently ported into **ProofPower-Z** and continues to be maintained as the tool evolves.

The Compliance Tool has been extensively used by QinetiQ for the verification of avionics control systems. This usage has fed back into further development of **ProofPower** to improve its performance and ease of use and to add new features, e.g., support for reasoning about the real numbers (to support Ada real data types). In conjunction with ClawZ, a tool that generates **ProofPower-Z** specifications from Simulink specifications [14], very large problems are being tackled with considerable success.

6 Further Information

ProofPower is one of several very successful systems whose origins can be traced back to Mike Gordon's original Cambridge HOL. Others include HOL 4, HOL Light and Isabelle-HOL. Many concerns dealt with in the design of **ProofPower** have also been addressed in one way or another in these systems, though the embedding of Z described here is unique to **ProofPower**. The HOL 4 web site [http://hol.sf.net/] is a good starting point for information on these systems.

ProofPower is an open source system supported and developed by Lemma1 Ltd. The software, documentation and further information may be obtained from the Lemma 1 web site [http://www.lemma-one.com/].

References

[1] A.N.Whitehead and B.Russell. *Principia Mathematica.* Cambridge University Press, 1910. 3 vols.

[2] R.D. Arthan. HOL *Formalised: Language and Overview.* Lemma 1 Ltd., http://www.lemma-one.com. DS/FMU/SPC001.

[3] R.D. Arthan. Analysis of Compiled Code: a Prototype Formal Model. In Jonathan P. Bowen, Steve Dunne, Andy Galloway, and Steve King, editors, *Proceedings of ZB 2000, LNCS 1878.* Springer-Verlag, 2000.

[4] Alonzo Church. A Formulation of the Simple Theory of Types. *Journal of Symbolic Logic*, 5:56-, 1940.

[5] Michael J.C. Gordon. HOL:A Proof Generating System for Higher-Order Logic. In G. Birtwistle and P. A. Subrahmanyam, editors, *VLSI Specification, Verification and Synthesis*. Kluwer, 1987.

[6] Michael J.C. Gordon. Mechanising Programming Logics in Higher Order Logic. In G. Birtwistle and P. A. Subrahmanyam, editors, *Proceedings of the 1988 Banff Conference on Hardware Verification*. Springer-Verlag, 1988.

[7] Michael J.C. Gordon and Tom F. Melham, editors. *Introduction to HOL*. Cambridge University Press, 1993.

[8] Michael J.C. Gordon, Arthur J. Milner, and Christopher P. Wadsworth. *Edinburgh LCF. Lecture Notes in Computer Science. Vol. 78*. Springer-Verlag, 1979.

[9] International Standards Organisation. *Information Technology - Z Formal Specification Notation – Syntax, Type System and Semantics*. ISO/IEC 13568:2002.

[10] L.Paulson. *Logic and Computation: Interactive Proof with Cambridge LCF*. Cambridge University Press, 1987. Cambridge Tracts in Theoretical Computer Science 2.

[11] Robin Milner, Mads Tofte, Robert Harper, and David MacQueen. *The Definition of Standard ML (Revised)*. MIT Press, 1997.

[12] Dept. of Defense. *Department of Defense Trusted Computer System Evaluation Criteria*, 1985. DoD 5200.28-STD.

[13] C O'Halloran, R. Arthan, and D. King. Using a Formal Specification Contractually. *Formal Aspects of Computing*, 9(4), 1997.

[14] C O'Halloran and A. Smith. Verification of Picture-Generated Code. *Proceedings of the 14th IEEE International Conference on Automated Software Engineering*, 1999.

[15] R.Milner. A Theory of Type Polymorphism in Programming. *Journal of Computer and System Sciences*, 17:348-375, 1978.

[16] B. Russell. Mathematical Logic as based on the Theory of Types. *American Journal of Mathematics*, 30:222 - 262, 1908.

[17] C. T. Sennett. Demonstrating the Compliance of Ada Programs with Z Specifications. In R.Shaw, editor, *5th Refinement Workshop*, Workshops in Computing. Springer-Verlag/BCS-FACS, 1992.

[18] J.M. Spivey. *The Z Notation: A Reference Manual*. Prentice-Hall, 1989. ■

BCS-FACS Evening Seminar Series

FM@Waikato
Professor Steve Reeves
[See abstract on page 64]
27 April 2005

Domain Engineering
Professor Dines Bjørner
25 July 2005

Formal Methods Meets Biochemical Pathways
Professor Muffy Calder
21 September 2005

[*Professor Jim Woodcock's seminar to be arranged*]

Venue:
BCS London Offices
5 Southampton Street
London WC2E 7HA

Time:
Seminars start at **5.45pm**
Refreshments served from **5.15pm**

Admission is **FREE** and open to FACS members and non-members. If you would like to attend, you need to pre-register by emailing Paul Boca [Paul.Boca@virgin.net] your name and affiliation. For further details, please visit the website http://www.bcs-facs.org/events/EveningSeminars .

The FACS seminar series is organized by Paul Boca, Jonathan Bowen and Jawed Siddiqi

FORTEST Network: Final meeting

Jonathan Bowen

On Thursday 10 March 2005, the EPSRC FORTEST Network held its final meeting at Cumberland Lodge [http://www.cumberlandlodge.ac.uk], a wonderful 17th Century country house in Windsor Great Park, now used for workshops and other meetings. The following talks were delivered to around twenty attendees:

Cumberland Lodge

1. Paul Baker (Motorola), *The UML2.0 Testing Profile*
2. John Clark (Univ. of York) and Rob Hierons (Brunel Univ.), *Semantic Mutation Testing*
3. Ian Craggs (IBM), *Results From Model-Based Testing an IBM Product*
4. Mike Poppleton (Univ. of Southampton), *Product Line Engineering: UML, B, Refinement and Testing*
5. Tony Cowling (Univ. of Sheffield), *Reflections on the Impossible Test Frame*
6. Tony Simons (Univ. of Sheffield), *Refinement and Regeneration: or the Failure of Regression Testing*
7. David King (Motorola), *A UML 2.0 Testing Environment*
8. Yuan Zhan (Univ. of York), *Automatic Test Set Generation and Management Framework*
9. Yongyan Zheng (Univ. of Surrey), *Using Dependency Analysis to reduce the State Space Explosion for State-based Testing*

Mike Poppleton demonstrates that formal methods and testing is not a dry subject!

The slides from the talks will be available on the FORTEST website [http://www.fortest.org.uk] for those that would like further information. Overall, this was a very enjoyable finale to the Network, which has been running since 2001 but officially finishes in April. However, future meetings are planned, some of them hopefully to be supported by FACS. Further information will be issued in due course. ■

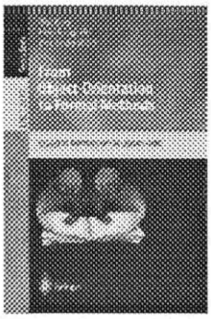
CALCO Young Researchers Workshop (CALCO-jnr 2005)
University of Wales Swansea

2 September 2005

Submission (2-page abstract): 30 April 2005
http://www.cs.swan.ac.uk/calco-jnr/

Book Review

Jonathan Bowen



From Object-Orientation to Formal Methods: Essays in Memory of Ole-Johan Dahl

By Olaf Owe, Stein Krogdahl & Tom Lyche (Eds.)
Springer, Lecture Notes in Computer Science, Volume 2635
X+389pp. £37.00 / 48.00 €.
ISBN 3-540-21366-X
Published March 2004

Ole-Johan Dahl [http://www.ifi.uio.no/~olejohan], born in 1931, is widely accepted as Norway's most famous computer scientist. Together with Kristen Nygaard, second only to Dahl in the country, he produced the initial ideas for object-oriented (OO) programming in the 1960s at the Norwegian Computing Center (NCC) as part of the Simula I (1961–1965) and Simula 67 (1965–1968) simulation programming languages [http://en.wikipedia.org/wiki/Simula]. They were the first to develop the concepts of class, subclass (allowing implicit information hiding), inheritance, dynamic object creation, etc., all important aspects of the OO paradigm. An "object" is a self-contained component (with a data structure and associated procedures or "methods") in a software system. These are combined to form a complete system. The object-oriented approach is now pervasive in modern software development, including widely used imperative programming languages such as Java and C++.

Dahl became a full professor at the University of Oslo in 1968 and was a gifted teacher as well as researcher. Here he worked on *Hierarchical Program Structures*, probably his most influential publication, which appeared co-authored with C. A. R. Hoare in the influential book *Structured Programming* of 1972 by Dahl, E. W. Dijkstra and Hoare, perhaps the best-known academic book concerning software in the 1970s. He was in the good company of two other giants in the field of provably correct software development.

As his career progressed, Dahl became increasingly interested in the use of formal methods, to rigorously reason about object-orientation for example. Like all good computer scientists, his expertise ranged from the practical application of ideas to their formal mathematical underpinning to ensure the validity of the approach.

The original idea of the book under review was that it would be a celebratory gift to Dahl in his retirement, but unfortunately 2002 was a sad year for Norwegian computing since both Dahl and Nygaard died within weeks of each other. His *Structured Programming* co-author, Edsger Dijkstra, also died in the same year so could not contribute to the book. However, this "Festschrift" volume is still a fitting posthumous tribute to Dahl by close colleagues and highly respected computer scientists from around the world.

The book starts with a good six-page overview biography of Dahl by the book's editors, followed by a comprehensive bibliography. The first full chapter is appropriately by Dahl himself (previously published in an almost identical form in 2002). It gives a historical introduction to the development of the Simula programming languages and the start of object orientation, including an assessment of the cultural impact on computing.

As well as in Europe, US computer scientists keenly took up OO ideas and it is here that much of the more recent practical development has taken place. Simula 67 was a simulation language and as such was directly useful for companies like Intel (inventors of the microprocessor) in the design of VLSI chips. Leading US research centres such as MIT and Xerox PARC also adapted OO concepts for use in languages like CLU and Lisp.

Nowadays, the tool-supported Unified Modeling Language (UML) is very widely used in industry for OO system design and development. The Common Object Request Broker Architecture (CORBA) is utilized by many companies for interfacing OO systems. C++, Eiffel and Java are important OO programming languages in current use. More specifically, the Microsoft Component Object Model (COM) is an important aspect of programming languages such as C#. Much practical software development is undertaken in an object-oriented manner and a university computer science degree course would find it extremely hard to justify ignoring the approach.

Rather than covering all the chapters, many of which are largely academic and specialist in nature, it is perhaps more interesting to pick a small number of highlights, especially where the authors had a close affinity with Dahl, as demonstrated by the comments in the opening or closing remarks in their chapter.

Tony Hoare's chapter covers the Verifying Compiler, a computer science "Grand Challenge" in the style of the mathematician Hilbert, where a program compiler could check that Boolean assertions in the code being compiled are always true as part of the compilation process. Currently in practical situations, such assertions can only be checked later when the program is actually run, as part of the testing process for example. Hoare hopes that such a compiler could be created some time this century.

Hoare also remarks on a presentation that Dahl made at a symposium in Oxford during 1999 to mark his own retirement from Oxford University. There was a power cut in the middle of the talk and, with only dim emergency lighting, no visual aids were possible. However, typically and with great aplomb, Dahl said he thought that he could continue and did so in the darkness to a rapt audience. Many lesser speakers would have floundered and delayed until the restoration of full power, which was not reinstated for some time in this case.

Eugene Kindler of Ostrava University in the Czech Republic provides an interesting account of the influence of Simula behind the Iron Curtain, where the very idea of computer simulation was regarded with suspicion by central bureaucrats. Computers were generally considered only useful as an auxiliary aid for mathematics, so his research interests were considered as somewhat subversive.

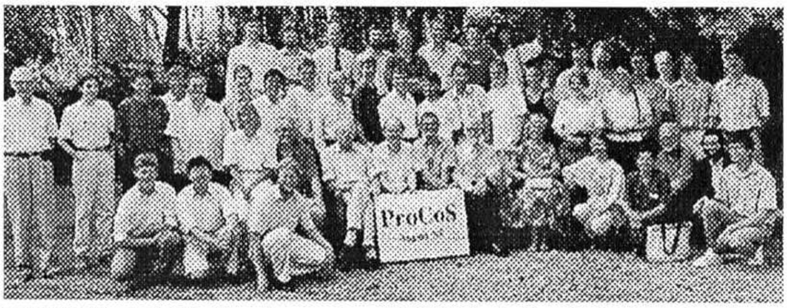
Donald Knuth at Stanford University, California, is one of the leading computer scientists in the world, with a great interest in algorithms, as illustrated by his chapter. He visited the University of Oslo (1972–1973) and Dahl visited Stanford (1977–1978), leading to fruitful collaboration on the science of programming, not to mention four-hand piano playing due to shared musical interests too!

Other very well-known contributors include Dines Bjørner, Manfred Broy, Joseph Goguen, Hans Langmaack, José Meseguer, Bertrand Meyer and Michel Sintzoff. Meyer (the original developer of the Eiffel OO language) covers object-oriented event-driven software design, including a comparison of the approach with Microsoft's .NET for connecting web services.

Dahl confesses to a degree of luck in his success with regard to the influence of his OO work, especially considering his geographically peripheral location. No doubt object-orientation would have emerged in some form as a paradigm, but as history has actually proceeded, he and Nygaard can be rightfully considered as the progenitors of the OO approach.

Despite his academic standing (and physical height), Dahl was a delightfully modest and unassuming person, with a very thoughtful but approachable manner ("a gentleman of the old school" with "good taste and elegance" as one dedication puts it). He was also a gifted classical pianist, often together with his wife on the violin. Another contributor describes Dahl as "a genial scientist, an excellent musician and a very good person." He will be missed by all that knew him.

This book will be of interest to any computer scientist who met Dahl or knows him by reputation. Anyone concerned with formal methods research will also find this a useful set of papers; all the contributors are of international calibre and many are very well-known computer scientists. Any object-oriented programmer who does not know the provenance of the approach they are using could do well to dip into this book, especially the initial three contributions. All academic libraries covering software engineering would benefit from a copy. It is sad that Dahl did not live to see the book's publication, but I think he would have approved of the content.



ESPRIT ProCoS Working Group, August 1995, including O.-J. Dahl, far left, and chapter authors H. Langmaack and C. A. R. Hoare, centre, left and right of the sign

Acknowledgement

A version of this book review originally appeared in the *Times Higher Education Supplement* [http://www.thes.co.uk/search/story.aspx?story_id=2019317]:

Pioneer who Prepared the Ground for Road to Java. The Times Higher Education Supplement, **1677**:34, 4 February 2005. Review of *From Object-Orientation to Formal Methods: Essays in Memory of Ole-Johan Dahl*, Olaf Owe, Stein Krogdahl and Tom Lyche (eds.), Berlin: Springer-Verlag, Lecture Notes in Computer Science, Volume 2635, 2004

It is reproduced here with permission. ■

Workshop and Conference Announcements

March 2005

BCTCS 2005: 21st British Colloquium for Theoretical Computer Science
22 – 24 March
Nottingham, UK
http://www.cs.nott.ac.uk/~gmh/bctcs05.html

April 2005

ETAPS 2005: The European Joint Conferences on Theory and Practice of Software
2–10 April
Edinburgh, UK
http://www.etaps05.inf.ed.ac.uk

ESOP 2005: The European Symposium on Programming
2–10 April
Edinburgh, UK
http://www.cs.tau.ac.il/~msagiv/esop05.html

REFINE 2005: Refinement Workshop – BCS FACS – EPSRC RefineNet (co-located with ZB 2005)
Guildford, UK
12 April
http://www.refinenet.org.uk/workshop_program.html

ZB2005: 4th International Conference of B and Z Users
Guildford, UK
13–15 April
http://www.zb2005.org

SFM 05: 5th International School on Formal Methods for the Design of Computer, Communication and Software Systems: Mobile Computing
Italy
26–30 April
http://www.sti.uniurb.it/events/sfm05moby

May 2005

AFL 05: 11th International Conference On Automata and Formal Languages
Hungary
17–20 May
http://www.inf.u-szeged.hu/afl05

June 2005

FMOODS 2005: 7th th IFIP International conference on Formal Methods for Open Object-based Distributed Systems
Athens, Greece
15–17 June
http://www.informatik.uni-kiel.de/~fmoods05

July 2005

Summer School on Generative and Transformation Techniques in Software Engineering
Braga, Portugal
4 – 8 July
http://www.di.uminho.pt/GTTSE2005

FATES 2005: 5th International Conference on Formal Approaches to Testing of Software
11 July
Submission: 4 April
http://research.microsoft.com/conferences/fates2005/

FM 2005
Newcastle, UK
18–22 July
http://www.csr.ncl.ac.uk/fm05

August 2005

TPHOLs 2005: 18th International Conference on Theorem Proving in Higher Order Logics
Oxford, UK
22–25 August
Submission: 27 May
http://web.comlab.ox.ac.uk/oucl/conferences/TPHOLs2005

SPIN 2005: 12th International SPIN Workshop on Model Checking of Software
San Francisco, USA
22–24 August
Submission: 1 April and 8 April
http://cm.bell-labs.com/cm/cs/what/spin2005

CONCUR 2005: International Conference on Concurrency Theory
San Francisco, USA
23–26 August
Submission: 28 March and 1 April
http://www.soe.ucsc.edu/concur05

September 2005

CALCO 2005: First Conference on Algebra and Coalgebra in Computer Science
Wales, UK
3 – 6 September
http://www.cs.swan.ac.uk/calco

SEFM 2005: 3rd IEEE International Conference on Software Engineering and Formal Methods
Koblenz, Germany
7 – 9 September
Submission: 18 March and 1 April
http://sefm2005.uni-koblenz.de

SAS'05: The 12th International Static Analysis Symposium
London
7 – 9 September
Submission: 11 April
http://www.doc.ic.ac.uk/~clh/sas05.htm

TFP 2005: 6th Symposium on Trends in Functional Programming
Tallin, Estonia
23 – 24 September
http://www.tifp.org/tfp2005

CPA2005: Communicating Process Architecture
The Netherlands
18 – 21 September
http://www.wotug.org/cpa2005/index.shtml

October 2005

FORTE 2005: 25th IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems
Taipei, Taiwan
2 – 5 October
Submission: 10 April
http://cc.ee.ntu.edu.tw/~forte05

CHARME 2005: 13th Advanced Research Working Conference on Correct Hardware Design and Verification Methods
Saarbrücken, Germany
6 – 9 October
Submission: 29 March
http://www.charme2005.com

ICTAC 05: International Colloquium on Theoretical Aspects of Computing
Hanoi, Vietnam
17 – 21 October
Submission: 25 May
http://www.iist.unu.edu/ictac05

November 2005

ICFEM 2005: Seventh International Conference on Formal Engineering Methods
Manchester, UK
1 – 4 November
Submission: 20 May
http://www.cs.man.ac.uk/icfem05

IFM 2005: 5th International Conference on Integrated Formal Methods
Eindhoven, The Netherlands
29 November – 2 December
Submission : 18 May
http://www.win.tue.nl/ifm

December 2005

FACS 2005 Christmas Meeting
London
December (date to be arranged)
http://www.bcs-facs.org/events/xmas2005/

Coming Soon in FACS FACTS....

Formal Methods Conference Reports

TRain Column

Report on FACS AGM

Formal Methods tools

Article on Functional Programming in Hardware Design

Article on Separation Logic

FACS Evening Seminars

And More...

BCS-FACS AGM

27 April 2005

2.30pm start
BCS London Offices
5 Southampton Street
London

Refreshments served from 2pm

Following the AGM there will be a talk by Professor Steve Reeves (University of Waikato, New Zealand), entitled: *FM@Waikato*. The talk, which is **free of charge**, will start at 5.45pm. Refreshments will be available from 5.15pm.

If you would like to attend the AGM and/or the talk, please contact Paul Boca [Paul.Boca@virgin.net] by 21 April.

FM@Waikato
Professor Steve Reeves, University of Waikato

This talk seeks to do two things: first, to give a brief overview of work going on at, I'm almost certain, the FM lab furthest from the BCS in London - it would take you about 30 hours of continuous travelling to get there!; second, to look in rather more detail at a few aspects of the work going on in the Waikato lab. These will include μ-Charts, simplified and completely formalised version of Statecharts or UML statecharts; and work in the area of putting together processes and states into "components" and ideas concerning the refinement of atomic actions.

Jobs

University of Surrey
School of Electronics and Physical Sciences

Exciting Opportunities in Computing at Surrey

Professor/Reader in Computing (Ref: 4784)
Lecturer in Computing (Ref: 4789)

(See http://www.open.mis.surrey.ac.uk/misweb/vacancy/home.htm for further details)

Attractive salaries and relocation package

The University of Surrey offers high-calibre teaching, a world-class research base, a thriving postgraduate community and a high quality of life in a beautiful campus setting. At the same time we have a strong eye for innovation and enterprise and are at the forefront of developments in teaching and research.

We now seek both a Professor/Reader and a Lecturer B (£27,989 - £35,883 per annum) to support the Department of Computing's continued growth. Emerging important research areas include formal methods, computer security, and development of dependable systems, and applicants in these areas are particularly encouraged to apply. Applications will also be considered for other research areas including software systems, neural and biologically inspired computing, and knowledge management. The Department is research-led with 64 PhD students and is attracting growing research support from the UK Research Councils, the EU-IST, and industry. Major IT, telecommunication, and defence organisations are sponsoring research in the Department which has two major research Centres: the Centre for Software Systems and the Centre for Knowledge Management.

The Department of Computing is part of the School of Electronics and Physical Science which has an excellent track record in computing and communications engineering with 5** rated Research Centres in computer vision and mobile communications respectively.

Applicants for the Professor/Reader position should have a record of sustained research funding together with proven management and leadership qualities, demonstrating achievements in scholarship and research at a national and international level. For the Lectureship a relevant PhD is desirable with some experience of teaching and a growing research profile.

Informal enquiries may be made to Prof Steve Schneider, Head of the Department of Computing (s.schneider@surrey.ac.uk , tel: 01483 689637).

For an application pack for the Professor/Reader post, please contact Mrs Linda Allen, Central HR Department, University of Surrey, Guildford GU2 7XH, email l.f.allen@surrey.ac.uk. If you are interested in the Lectureship please contact Karen Gooday, School HR Manager, SEPS, email k.gooday@surrey.ac.uk , or download application forms from www.surrey.ac.uk "Employment Opportunities". Completed applications for the Professor/Reader should be submitted to Mrs Allen, and to Ms Gooday for the Lectureship by 4 April 2005.

The University is committed to an Equal Opportunities Policy

Opening for Chair in Formal Methods

Declarative Systems and Software Engineering Research Group
School of Electronics and Computer Science
University of Southampton

Application deadline: 6 May 2005.

Applications are invited for a Chair in Formal Methods at the School of Electronics and Computer Science at the University of Southampton. The School is one of the most successful and largest of its kind in the UK. The School consistently achieves the highest ratings for both its research and teaching (5* in both Computer Science and Electrical and Electronic Engineering in RAE 2001, and now rated 6* by HEFCE).

The new chair will be in the Declarative Systems and Software Engineering (DSSE) Research Group [www.dsse.ecs.soton.ac.uk] which has major established research programmes in formal methods, software engineering and GRID middleware. Our formal methods research includes leading work on refinement techniques, model checking, algebraic methods and method integration. The group has strong collaboration with various industrial organisations on the practical application of formal methods.

This new appointment will extend and enhance our formal methods expertise through personal and collaborative research. We seek applicants with a track record of research fully commensurate with the School's 6* ratings and proven qualities of academic research leadership. We are especially keen to make an appointment in automated verification and/or model checking, but applications from outstanding individuals with other related research expertise are also very welcome.

Information about the School can be found at www.ecs.soton.ac.uk. Informal enquiries may be made to the Head of the DSSE Research Group, Professor Peter Henderson [ph@ecs.soton.ac.uk]; or to Professor Michael Butler [mjb@ecs.soton.ac.uk]

Salary will be on the professorial scale.

Further details from:
http://www.jobs.soton.ac.uk/adminweb/jsp/jobs/sJobview.jsp?function=View&id=04P0496 or http://www.ecs.soton.ac.uk/~mjb/

Application deadline: 6 May 2005.

Paid-up FACS Members receive the following benefits:

- substantial discount on the *Formal Aspects of Computing* journal subscription fee
- discounts at FACS events (when available)
- 25% discount on Springer titles
- 20% discount on the Requirements Engineering journal subscription fee

If you would like to become a FACS member – or renew your membership – please complete the application form on **Page 70** and return it to the Membership Secretary with the appropriate fee.

FACS Committee

Jonathan Bowen
FACS Chair
ZUG Liaison

Jawed Siddiqi
Treasurer

Roger Carsley
Secretary

Paul Boca
Membership Sec.
Newsletter Editor

John Cooke
FAC Journal
Liaison

Ali Abdallah
Events Coordinator

John Fitzgerald
FME Liaison
SCSC Liaison

Margaret West
BCS Liaison

Mike Stannett
Webmaster
LMS Liaison

Judith Carlton
Industrial Liaison

Kevin Lano
UML Liaison

Rick Thomas
LMS Liaison

FACS is always interested to hear from its members and keen to recruit additional Committee members. Presently we have vacancies for officers to handle publicity and help with fund raising, and to liaise with other specialist groups such as the Requirements Engineering group and the European Association for Theoretical Computer Science (EATCS). If you are interested in helping the Committee, please contact the FACS Chair, Professor Jonathan Bowen, at the contact points below:

BCS FACS
c/o Prof. Jonathan Bowen (Chair)
London South Bank University
Faculty of BCIM
Borough Road
London SE1 0AA
United Kingdom

T +44 (0)20 7815 7462
F +44 (0)20 7815 7793
E info@bcs-facs.org.uk
W www.bcs-facs.org

You can also contact the other Committee members via this email address.

Please feel free to discuss any ideas you have for FACS or voice any opinions openly on the FACS mailing list [FACS@jiscmail.ac.uk]. You can also use this list to pose questions and to make contact with other members working in your area. Note: only FACS members can post to the list; archives are accessible to everyone at http://www.jiscmail.ac.uk/lists/facs.html .

Announcement from Escher Technologies Ltd

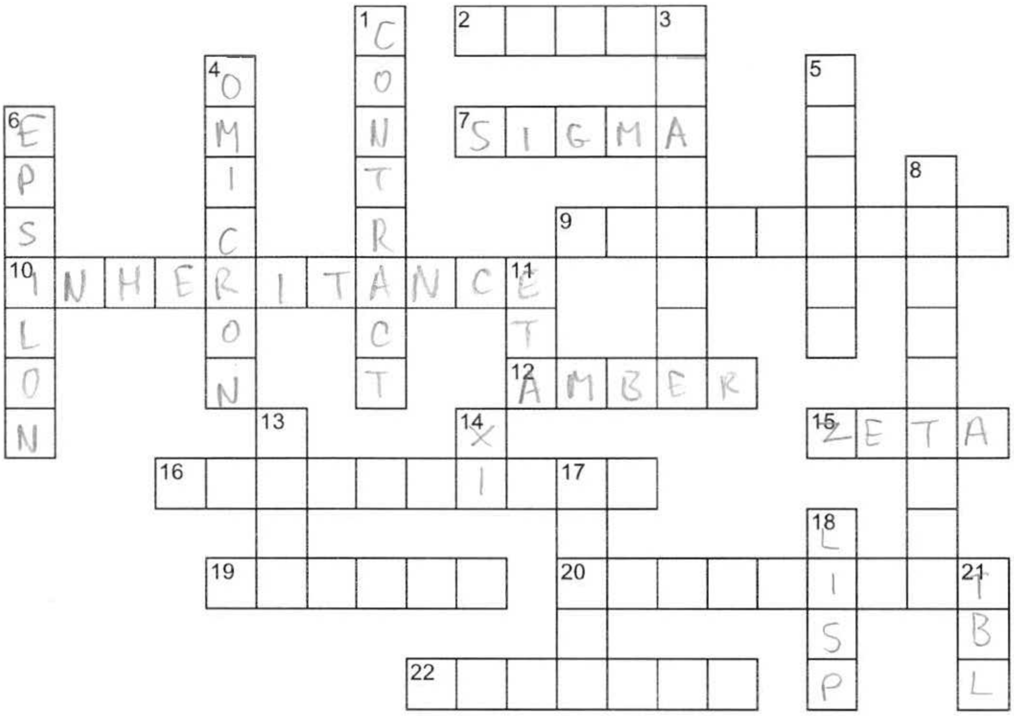
The Educational Edition of the formal methods software development tool known commercially as ***Perfect Developer*** is to be made **FREELY AVAILABLE to universities from August 2005**. It always has been free for individual student projects; from August, it will be free for classroom teaching as well.

Perfect Developer Version 3 was released in December 2004. There will be a tutorial on Perfect Developer at FM05, in July.

For more information on Perfect Developer, please see Escher's website [http://www.eschertech.com] and also the article in Issue 2004-3 of *FACS FACTS*.

Formal Methods Coffee Time

Judith Carlton



Down

1 sort of bridge
3 lady of the higher order?
4 Greek letter that sounds like a unit of resistance but is not used as the symbol for it.
5 programming language named for him – but not in French
6 fifth letter of the Greek alphabet.
8 a journey, literally
11 seventh letter of the Greek alphabet.
13 Greek letter used to indicate a very small quantity.
14 not an old Greek letter?
17 motto
18 possible speech impediment
21 initials of "Mr WWW"

Across

2 a net or dish
7 Greek letter used for summation.
9 kind of logic
10 could be taxing for children
12 in UK, may be red next
15 Greek letter for Z
16 methodical
19 significant Fortran developer
20 a constraint
22 get some spin on with this!

BCS FACS

FACS membership application/renewal (2005)

Title (Prof/Dr/Mr/Ms) _____ **First name** ____________ **Last name**____________

Email address (required for options * below)______________________________

BCS membership No. (or sister society name + membership number)

____

Address ____

____

____

Postcode ______________ **Country** ___________________________

I would like to take out **membership to FACS** at the following rate:

- □ £15 (Previous member of BCS-FACS now retired, unwaged or a student)
- □ £15 (Member of BCS or sister society with web/email access)*
- □ £30 (Non-member or member of BCS or sister society without web/email access)

In addition I would like to subscribe to **Volume 17 of the FAC journal** at the following rate:

- □ £46

For electronic only journal subscription*, please tick here □. No further discount given.

The total amount payable to BCS-FACS in **pounds sterling** is **£ 15 / 30 / 61 / 76**
(delete as appropriate). I am paying by:

- □ Cheque made payable to **BCS-FACS** (**in pounds sterling**)
- □ Credit card via PayPal (instructions can be found on the BCS-FACS website)
- □ Direct transfer (in **pounds sterling**) to:

Bank: Lloyds TSB Bank, Langham Place, London
Sort Code: 30-94-87
Account Number: 00173977
Title of Account: BCS-FACS

If a receipt is required, please tick here □ and **enclose** a stamped self-addressed envelope.

Please send completed forms to:

Dr Paul P Boca
PO BOX 32173
LONDON N4 4YP
UK

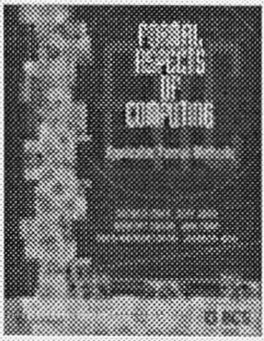
For FACS use only

Received by FACS	Date:	Initials:
Sent to Springer	Date:	Initials
Actioned by Springer	Date:	Initials

And Finally

Solution to crossword on page 69

							1 C		2 P	E	T	R	3 I						
				4 O			O						S			5 P			
6 E				M			N		7 S	I	G	M	A			A			
P				I			T						B			S		8 I	
S				C			R				9 P	R	E	D	I	C	A	T	E
10 I	N	H	E	R	I	T	A	N	C	11 E			L			A		E	
L				O			C			T			L			L		R	
O				N			T			12 A	M	B	E	R				A	
N					13 I				14 N							15 Z	E	T	A
			16 P	R	O	C	E	D	U	R	17 A	L						I	
					T						X					18 L		O	
				19 B	A	C	K	U	S		20 I	N	V	A	R	I	A	N	21 T
											O					S			B
								22 P	R	O	M	E	L	A		P			L



News from Springer

The *Formal Aspects of Computing* (FAC) journal has been accepted in ISI's Sciences Citation Index Expanded, CompuMath Citation Index and Current Contents/Engineering and Technology. Coverage will begin this year.

FACS members are entitled to a substantial discount on the FAC journal subscription fee. For further details on how to subscribe to the current volume of the journal, please see the membership form on **page 70** or visit the FACS website [http://www.bcs-facs.org] .