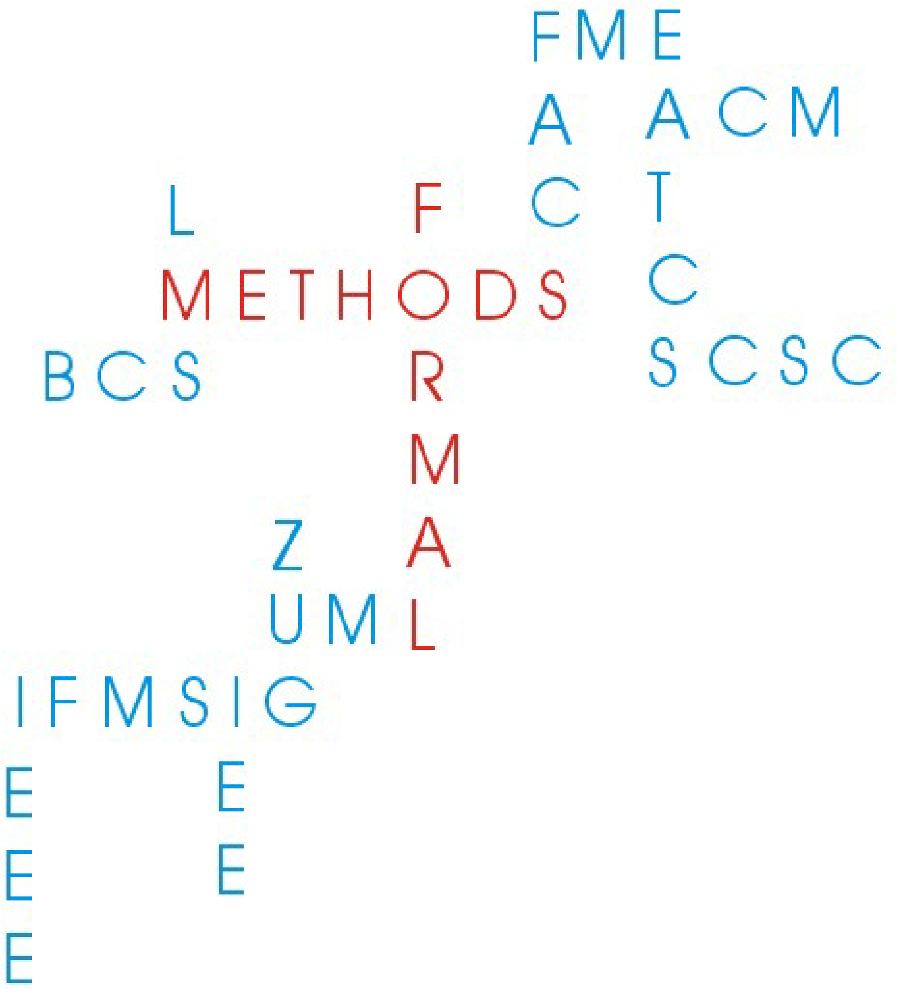
Issue 2006-1
March 2006

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.

FACS FACTS is published four times a year: **March**, **June**, **September** and **December**. Submissions are always welcome. Please see the advertisement on **page 11** 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
Columnists	Dines Bjørner (Train Domain)

Contributors to this Issue

Paul Boca, Jonathan Bowen, Michael Butler, Greg Reeve (via Steve Reeves), F.X. Reid, Victor Zemantics

Contents

The activities of FACS (i.e. **sponsoring conferences and workshops**, **offering student bursaries** and **hosting evening seminars**) are funded solely from membership subscriptions. The more paid-up FACS members we have, the more we can do. ☺

If you would like to become a FACS member – or renew your lapsed membership – please complete the membership form on **page 25** of this issue of *FACS FACTS*.

If you have any questions about FACS, please send these to Paul Boca [Paul.Boca@virgin.net]

Editorial

Paul Boca & Jonathan P. Bowen, BCS-FACS

Welcome to the first Issue of *FACS FACTS* of 2006. As usual we thank all of the contributors for their support – without them there would be no newsletter. Submissions are always welcome, so please do feel free to contact the editor, Paul Boca [Paul.Boca@virgin.net].

This is a somewhat sombre time for the editorial board of the newsletter, as we have recently learned that F. X. Reid [http://en.wikipedia.org/wiki/F._X._Reid], long-term contributor to the FACS newsletter, has unexpectedly passed away. An obituary, written by Victor Zemantics, appears on page 12. Reid authored several articles over the years, and has been responsible for "educating" many PhD students as a result. As a tribute to him, we will reprint some of his "gems", starting with an article on the semantics of the **COMEFROM** statement (see page 18).

On a happier note, we can report that the FACS Evening Seminars are still proving to be *very* popular. We began the year with a panel discussion, entitled *Formal Methods in the Last 25 Years*; more than 70 people attended. The event was organized together with Formal Methods Europe (FME), chaired by John Fitzgerald. We would like to thank FME and the Centre for Software Reliability, University of Newcastle upon Tyne for co-sponsoring the event with FACS. We are hoping to bring you a report on the event in the next issue of the newsletter, together with a reprint of the classic "Magic Roundabout" article.

Peter Mosses gave the second seminar in the series this year, entitled *Programming Language Description Languages: From Scott and Strachey to Semantics Online*. The seminar was attended by almost 40 people – an excellent turnout for a Friday evening. At the time of writing this editorial, we are organizing the third seminar in the series, to be given on 24 April 2006 by Cliff Jones, entitled *Specifying Systems that Connect to the Physical World*. We expect this to be equally successful, and once again it is an opportunity for people to hear an address from another pioneer in formal methods.

The seminar programme for the rest of the year is available on the BCS-FACS website [http://www.bcs-facs.org/events/EveningSeminars] and in the advertisement on page 14. Slides from previous seminars are available to download from this website too. We hope you will find the programme of interest and will be able to attend – the seminars are free of charge, funded from membership subscriptions (see page 25 for a membership form).

FACS held its AGM on 3 March 2006, and a report on the meeting will appear in a future issue of the newsletter. FACS welcomes two new committee members: John Derrick (University of Sheffield) to coordinate activities relating to refinement and Mark D'Inverno (University of Westminster) to coordinate model-based specification events (on topics such as B, VDM, Z, etc.). Please do liaise with John or Mark if you are interested in being involved with events in these areas. Unfortunately, FACS has lost two of its officers, Ali Abdallah and Kevin Lano. On behalf of all the members, we would like to thank them for all their efforts over the years. In particular, Ali organized two major and very successful events for FACS, FASec in 2002 on formal aspects of security and CSP 25 in 2004 to celebrate a quarter of a century of FACS and

Communicating Sequential Processes activities. In no small measure, these helped rejuvenate FACS and both proceedings appeared as Springer LNCS volumes.

We hope you will enjoy reading the current issue and again encourage you to contribute as well.

Joining Other Societies and Groups

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

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

European Association for Theoretical Computer Science
http://www.eatcs.org/organization/membership.html#how_to_join

Association for Computing Machinery
https://campus.acm.org/Public/QuickJoin/interim.cfm

IEEE Computer Society
www.computer.org/join/

The British Computer Society
www.bcs.org/bcs/join/

On the Verified-by-Construction Approach

Michael Butler, University of Southampton

Introduction

At the VSTTE (Verified Software: Theories, Tools, Experiments) conference [http://vstte.inf.ethz.ch/] held in ETH Zürich in October 2005, Tony Hoare and Jay Misra presented a vision of an international Grand Challenge to construct a program verifier. This vision appears to be having a very powerful catalysing effect on getting researchers in all manner of formal development approaches to pool resources and work towards more common goals. This is borne out by the large gathering of top researchers at the VSTTE conference and by the subsequent establishment of working groups set up to refine the challenge further. This article is my attempt at making the case for the so-called verification-by-construction approach to formal development and the contribution it can make to the challenge of verified software.

Why verification-by-construction is important

Much discussion on the need for a powerful program verifier seems to contain the following underlying assumptions:

- That a program verifier will be used mostly to verify programs
- That when verification fails it is because the program contains errors

While a powerful program verifier is a very valuable tool for programmers, it does not help them construct a verifiable program in the first place. Equally, the quality of any verification is dependent on the validity of the formal properties against which a program is checked. The verification-by-construction approach helps developers who want to construct reliable software systems by addressing the following questions:

- How do we construct properties against which to verify our software?
- How do we construct our software so that the verification will succeed?

The verification-by-construction approach is about providing design tools that help developers produce reliable software. It broadens the focus away from just being analysis of the finished *product* and addresses better the development *process*.

How can verification-by-construction be achieved?

Verification by construction can be achieved by having a formal framework in which models are constructed at multiple levels of abstraction and related by refinement[1] relations. The highest levels of abstraction are used to express the required behaviour in terms of the problem domain. The closer it is to the problem domain, the easier it is to validate against the informal requirements, i.e., ensure that it is the right specification. The lowest level of abstraction corresponds to an implementation or to a specification from which an efficient implementation can be derived automatically. Also critical in this framework are mechanisms for composing and decomposing models. Composition can be useful for building up specifications by combining models incorporating different requirements. Decomposition is important for relating system models to architectures of subsystem models and subsequent separate refinement of subsystems.

Ensuring that two models, M1 and M2, are in a refinement relation may be achieved in one of two ways:

1) Posit-and-Prove: The developer provides both M1 and M2 and uses tools to verify that M1 is refined by M2. In some cases this might be possible using a model checker. Alternatively a tool will generate proof obligations which can be verified using powerful theorem provers or possibly checked using model checkers. Typically this approach requires properties such as invariants and variants to be provided by the developers.

2) Transformational Approach: The developer provides M1 and applies a transformation that automatically constructs M2 in a way that guarantees refinement. This might result in the generation of side conditions that will need to be verified but discharging these should be a lot less effort than proving that M1 is refined by M2 in the posit-and-prove way.

One can immediately see how the transformational approach helps developers to construct software such that the verification will succeed. Unfortunately a fully transformational approach for a broad range of problems and solutions is far from being realised so that the posit-and-prove approach will rule for the foreseeable future. It might not appear immediately clear how the posit-and-prove approach helps developers to construct software for which the verification will succeed since the developer is expected to provide M2 as well as M1. This is where having multiple levels of abstraction is important. Typically there is a large abstraction gap between a good formal specification, i.e., one that is easy to validate against the requirements, and an efficient implementation. This gap means it is more difficult to be guided by the specification when constructing an implementation. By having smaller abstraction gaps between a model M1 and its intended refinement M2, it is more natural to be guided by M1 when constructing M2. Typically a refinement step incorporates a design decision about how some effect is achieved or represents an optimization of the design. With a small abstraction gap, the construction of M2 is driven by both M1 and the desired design decision or optimization. When the construction of M2 is guided by M1, then the verification that M2 refines M1 is more likely to succeed.

[1] According to dictionary.com, 'to refine' means 'to reduce to a pure state'. Ironically our use of the term has the exact opposite meaning. The term 'reify' (as used by Cliff Jones and others) is perhaps more appropriate for what we do but is far less widespread. I expect we are stuck with 'refine'.

A halfway house between transformational and posit-and-prove can be envisaged, where certain *patterns* of model and refinement can be captured and used in the construction of refinements. This is a more pragmatic idea than transformational refinement in that the pattern might not guarantee the correctness of the refinement[2]. Instead M2 would be constructed from M1 by application of a pattern and the correctness of the refinement would be proved in the usual posit-and-prove way. Ideally the pattern should provide much of the ancillary properties (e.g., invariants, tactics) required to complete the proof.

Models versus Properties

In a refinement approach one does not necessarily distinguish between properties and models. Essentially we are working with models in a modelling language and the important property to be proved of some model M2 is that it is a refinement of some other model M1. In doing this, we may need ancillary properties like invariants, variants and assertions. Good tools can help us discover these ancillary properties as part of the effort of trying to prove a refinement. So the answer to the question 'what properties should we prove of a model?' is 'those properties that allow us to show that it is a refinement of its abstraction'. For the most abstract models, the important property is that they satisfy the requirements of the problem domain. This is an informal check which can sometimes be aided by ancillary properties. Within a particular framework there may be differing strengths of refinement. A weaker notion might capture the preservation of safety behaviour, while stronger notions might capture preservation of liveness and/or fairness.

With a refinement approach the 'creative' input in a development is a collection of explicit models at different levels of abstraction. The invention of ancillary properties is dictated by the need to prove refinement between these explicit models. From an engineering perspective, I would argue that an explicit model is a fairly natural thing to have to create because one can easily get a feeling of 'completeness' of the model (at a certain level of abstraction). When creating properties rather than models I find it is more difficult to achieve that sense of 'completeness'.

In my experience, refinement is never purely top down from most to least abstract. The reason is that it is difficult to get the abstract model precisely right. One usually starts with an idealistic abstract model because that is easy to define. As refinement proceeds and more architectural and environmental details are addressed it often becomes clearer how the ideal abstract model needs to be modified to reflect reality better. Modifications to some level of abstraction will ripple up and down the refinement chain. This is not a weakness of the refinement approach per se, rather a reflection of the reality of engineering of complex systems.

It goes without saying that the refinement relation should enjoy some form of transitivity. I say 'some form of transitivity' because refinement is based on comparing some notion of what can be observed about a model and it is useful to be able to modify what can be observed at different levels of abstraction. In particular, the interface to a system is usually described

[2] A refinement M2 is correct with respect to some model M1 when M2 refines M1.

abstractly and may need to be made much more concrete at decomposition or implementation levels. In such cases, the observable behaviour is not directly comparable, but needs to be compared via some mapping and transitivity of refinement is via composition of mappings.

Other points in favour of verification-by-construction

The verification-by-construction approach encourages verification of designs and not just verification of programs. From an engineering perspective, it is possible that there is a greater payoff from verifying designs rather than programs. Does it not seem more likely that a design error would have a detrimental impact on system reliability than a programming error?

As well as supporting verification of designs and implementations, good formal modelling languages encourage a rational design process. The use of good abstractions and simple mathematical structures in modelling can lead to cleaner, more rational system architectures that are easier to understand and evolve than architectures developed using less disciplined approaches. Being able to verify a system is not enough. It is also important to be able to test, maintain and evolve it. This is facilitated by rational design.

The inclusion of annotations such as invariants and assertions in programming languages (e.g., Eiffel, Spark Ada, JML, Spec#), along with associated analysis tools, provide powerful support for programmers. However, this approach is not enough on its own as these annotations are designed to specify properties about programs but do not easily allow for reasoning about the contribution an individual program makes to the overall reliability of a system. Control systems, interactive systems and distributed systems involve multiple agents (users, environments, new programs, legacy components) all of which contribute to the reliability of a system. Individually the agents may be very complex so reasoning about compositions of agents in all their gory detail may be infeasible. Instead, there is evidence that it will be feasible to reason about complex systems through good use of abstraction, refinement and decomposition.

When verifying a program directly one is having to reason about a number of issues simultaneously; the problem to be solved, the data structures used in the solution and the algorithmic structures used in the solution. If these issues can be factored out and dealt with separately as much as possible, the proof obligations can be simplified and the reasoning made more manageable. Abstraction and refinement supports this factorisation. It is often possible to model and reason about how a strategy solves a problem in an abstract way using abstract algorithmic and data structures. This abstract solution can then be optimized by introducing more concrete algorithmic and data structures through refinement. Reasoning about these optimizing refinements no longer requires reasoning about the original problem as this will have been dealt with by the earlier refinement. By keeping the models as abstract as possible at each level, we will have simpler proof obligations to discharge. At higher levels of abstraction we focus the reasoning more on the problem domain and less on the details of the particular solution.

Further questions

Which notations should be used? My own experience is that one can go a long way with set theory and logic as used, for example, in Z, VDM and B. Dealing with reactive and distributed systems in these notations requires richer notions of refinement and decomposition, but not necessarily major extensions to the notations. In cases it is appropriate to augment set theory and logic with notations such as process algebra and temporal logic.

What type of systems should we work on in the grand challenge? I am especially interested in multi-user, distributed systems and in control systems involving an environment and believe these will provide many interesting challenges.

What about the link to programming languages? To some extent, the choice of particular programming language is not so important in the verification-by-construction approach. What matters is that a sound mapping can be made between the lower level abstractions used in verification-by-construction and the constructs of target programming languages. There is however an interesting overlap between this mapping and important research in programming language design which tries to improve programming abstractions. In particular I am thinking of:

- Declarative styles of programming
- Atomicity and transactional support for concurrent programming
- Abstractions for structured data (e.g., abstractions of XML messages, abstractions of pointer structures)

Clearly, better programming abstractions will make it easier to bridge the gap between models and programs.

The challenge

To a large extent the required theory to support verification-by-construction already exists. The challenge is to provide a powerful set of tools to support abstraction, refinement and decomposition. In achieving this, we should strive to achieve as much integration as possible and avoid silos. We should also exploit as much of the existing and future advances in theorem proving and model checking as possible, as well as advances in programming language design, program verification and automated program generation. As they evolve, the support tools should be applied to the development of interesting software-based systems. No doubt interesting theoretical advances will be identified and achieved along the way as well.

FACS FACTS Issue 2006-2

Call for Submissions

Deadline **19 May 2006**

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 **19 May 2006**.

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

Obituary: F.X. Reid

Victor Zemantics

The world of theoretical computer science was devastated last night by the announcement of the death of that great pioneer, teacher, raconteur, *bon vivant* and serial philanderer, Professor F. X. Reid. Speaking from Reid's villa in Marsascala, his physician, Dr. de Bono, told reporters:

> *'Not since the death of Jean Parisot de la Vallette have the people of Malta so mourned a resident alien. Professor Reid fought his infirmity with the implacability with which, so I'm told, he was notorious.'*

He was then observed to shed a small tear.

Tributes have been coming in from all over the world[3]. A special mass has been announced in St Peter's in Rome and a number of prominent Anglican cathedrals have already begun a somewhat unseemly wrangle over the possession of his bones. Poet's Corner has yet to put in a bid.

Details of the great man's life are not so much hard to come by as impossible to verify, or, at least, believe. Born 'in the early 1930s' in Przemsyl and christened Francis Xavier Rzyzryrd ('a bad hand at Scrabble', as he quipped later in life), he was educated privately ('Despite my name, my parents were dismissive of the Jesuits – name one Jesuit mathematician!'), before entering the University of Vienna ('at a remarkably early age'), to study philosophy. He quickly transferred to Gottingen ('Wittgenstein or somebody made a heavy pass at me.'), where he soon attracted the attention of David Hilbert, among others. At this point, the record becomes obscure. Rzyzryrd claims that his exposure to the Abstract Algebraists at Gottingen prompted him, not only to invent Universal Algebra ('a rather obvious generalization') but to see an application for it in the theory of abstract data types and algebras of processes ('My first attack on the formalization of computability preceded those of Church and Turing and was remarkably prophetic, given the later development of strongly typed procedural languages.') Unfortunately, Rzyzryrd's PhD thesis, if it ever existed, was destroyed during the firebombing of Dresden[4].

By this time, Rzyzryrd, who now styled himself Reid, was in Bletchley Park. Again, due to the top secret nature of the decryption work going on there, there are no records to back up his claim that: 'If it were not for me, the Bismark would still be afloat' nor his accusation that 'Alan Turing or somebody made a heavy pass at me'.

It was at Bletchley that Reid first encountered electronic computers. I think we can discount his claim to have written a primitive version of Pacman for the Colossus machine, but it was certainly from this period that his principal research effort originates. 'My first inkling as to how recursive procedures could be implemented came in the canteen at Station X, watching trays being stacked and unstacked; anything to take my mind from Spam fritters!') His first major

[3] Except Oxford, of course, where they affect not to have heard of him, or, indeed, anybody else.

[4] Just why all copies of the thesis were in Dresden has yet to be explained.

discovery (unattributed), the wash-rinse cycle is said to date from his time at Bletchley ('It took weeks from the laundry to come back; fortunately I had two pairs of socks').

At the conclusion of hostilities in Europe, Reid sought to join Turing in Manchester at the ACE project, but apparently Turing 'felt nervous in my company, for some reason.' Instead, Reid began a peripatetic existence, moving (and being moved on) from University to University seeking and occasionally finding academics with whom to collaborate. Karl Adam Petri, Dana Scott, Christopher Strachey, Donald Knuth, Robin Milner, Tony Hoare, Cliff Jones, David Turner and Antoni Mazurkiewicz are just some of the leading theoreticians who found an urgent need to be elsewhere when he turned up on their doorsteps.

Nevertheless, he pursued his work, publishing his celebrated 'Redundant Sock Theorem' (an application of Shannon's information theory to Laundry Science) in 1949, his study of the stochastics of mis-delivered mail (a profound influence of the development of the notion of packet-switching), in 1953 and in the period from 1961 to 1985, a series of ground-breaking papers culminating in the publication of his General Theory of Generality (a marriage of Scott's recursive domains and non-standard logic). This work is still not fully understood and, indeed, probably never will be.

Reid's acting career is much less widely known, probably for good reasons. While I am not convinced that he *did* actually stand in for William Hartnell in the first series of *Dr. Who*, his height (not to mention his somewhat precise mode of speech) would have made him ideal as a Dalek. His sequence of commercials for a well known brand of haemorrhoid ointment is now, thankfully, forgotten.

His non-scientific writings have also been unnoticed. His first novel 'Legless in Gozo', written shortly after moving to Malta, 'to avoid the Yob culture of Hampstead and such places', remains unpublished, although Reid's literary agent, Tony Bowdler, maintains that it is

> *'a masterpiece – a synthesis of Proust, J. K. Rowling and the Marquis de Sade…it is not fully understood and probably never will be.'*

A thinly disguised autobiography, it inevitably contains a variety of passages in which various eminent people make heavy passes at the narrator.

But it is to Malta that we owe a resurgence in Reid's scientific creativity. Writing always for obscure publications, such as the FACS newsletter, *FACS FACTS*, he continued to extend and elaborate his work on non-Bayseian probabilistic cube-complex automata, asynchrony theory and deadlock taxonomies. His magnum opus, the as yet unpublished *Principia Informatica*, a work not fully understood, etc., was produced during this period. There are rumours, some ugly, of a second novel and a concertino for oboe and string orchestra.

And what of Reid's legacy? This is a difficult question to answer, or even contemplate[5]. Was he, as he claimed, the embodiment of the *Zeigeist* always at the front line of the burgeoning discipline of informatics, always anticipating the

[5] At least with a straight face.

work of others and modestly declining to take the credit? Or was he merely the figment of a warped imagination? Time may tell, but I certainly won’t.

BCS-FACS Evening Seminar Programme (2006)

9 November	Professor Ursula Martin Queen Mary, University of London
4 September	Professor Peter Ryan University of Newcastle *The Computer Ate my Vote* *(starts at 6pm)*
21 June	Dr Anthony Hall Independent Consultant *Realising the Benefits of Formal Methods*
24 April	Professor Cliff Jones University of Newcastle *Specifying Systems that Connect to the Physical World*

All seminars are held at the BCS London Offices, near Covent Garden:

BCS London Offices

First Floor, The Davidson Building
5 Southampton Street
London WC2E 7HA

If you would like at attend any of these seminars, please contact Paul Boca [mailto:Paul.Boca@virgin.net]. Unless otherwise stated, all seminars start at 5.45pm, with refreshments served from 5.15pm.

See http://www.bcs-facs.org/events/EveningSeminars for further details.

Conference Announcements

The following are sponsored by BCS-FACS and/or considered of special interest to BCS-FACS members:

April 2006

BCTCS 2006 – 22nd British Colloquium for Theoretical Computer Science
4–7 April
Swansea, UK
http://www.cs.swan.ac.uk/BCTCS2006

ZUM 2006 – 16th International Z User Meeting
25 April
Columbia, USA
http://www.zuser.org/zum2006/

June 2006

WADT 2006 - 18th International Workshop on Algebraic Development
1–3 June
Submission: 15 April
La Roche en Ardenne, Belgium
http://www.info.fundp.ac.be/~pys/WADT06

DisCoTec 2006 – Distributed Computing Techniques
13–16 June
Bologna, Italy
http://www.discotec06.cs.unibo.it/satellite.htm

CiE 2006 - Computability in Europe 2006. Logical Approaches to Computational Barriers.
30 June – 5 July
Swansea, Wales
http://www.cs.swan.ac.uk/cie06

July 2006

CORDIE 06 – 1st International Symposium on Concurrency, Real-Time and Distribution in Eiffel-Like Languages
4–5 July
York, UK
http://www-users.cs.york.ac.uk/~paige/cordie06.htm

August 2006

MFCSIT 2006 – 4th Irish Conference on Mathematical Foundations of Computer Science and Information Technology
1–5 August
Submission: 17 April
Cork, Ireland
http://www.ucc.ie/info-mfcsit

FTSAS 2006 – Formal Techniques for Specification and Analysis and Security
20 – 25 August
Santiago de Chile, Chile
http://www.fing.edu.uy/inco/eventos/ftsas06/index.html

FM2006 – Formal Methods 2006
21–27 August
Submission: 24 February
Ontario, Canada
http://fm06.mcmaster.ca

SAS 06 – 13th International Static Analysis Symposium
29 – 31 August
Submission: 14 May
Seoul, Korea
http://ropas.snu.ac.kr/sas06/

September 2006

SEFM 2006 – 4th IEEE International Conference on Software Engineering and Formal Methods
11–15 September
Pune, India
http://www.iist.unu.edu/SEFM06/

JMLC 2006 – Joint Modular Languages Conference 2006
13–15 September
Submission: 7 April
Oxford, UK
http://cms.brookes.ac.uk/computing/JMLC2006

ICFP 2006 – 11th ACM SIGPLAN International Conference on Functional Programming
18–20 September
Submission: 7 April
Oregon, USA
http://icfp06.cs.uchicago.edu/

September 2006

FDL06 – Forum on specification and Design Languages
19–22 September
Submission: 10 April
Darmstadt, Germany
http://www.ecsi-association.org/ecsi/fdl/fdl06/

October 2006

ICFEM 2006 – 8th International Conference on Formal Engineering Methods
30 October – 3 November
Submission: 12 May
Macao, China
http://www.iist.unu.edu/icfem06

November 2006

FMCO 2006 – 5th International Symposium on Formal Methods for Objects and Components
7–10 November
Submission: 5 September
CWI, Amsterdam
http://fmco.liacs.nl/fmco06.html

IsoLA 2006 – 2nd International Conference on Leveraging Applications of Formal Methods, Verification and Validation
15–19 November
Submission: 2 June
Cyprus
http://sttt.cs.uni-dortmund.de/isola2006/

ICTAC 2006 – 3rd International Colloquium on Theoretical Aspects of Computing
20–24 November
Submission: 1 May 2006
Gammart/Tunis, Tunisa
http://www.iist.unu.edu/ICTAC2006

December 2006

BCS-FACS Christmas Meeting on Teaching Formal Methods
15 December
London, UK
http://www.bcs-facs.org/events/xmas2006.html

For further conference announcements, please visit the **Formal Methods Europe** (FME) website [http://www.fmeurope.org], the **EATCS** website [http://www.eatcs.org] and the **Virtual Library Formal Methods** website [http://vl.fmnet.info/meetings].

On the Formal Semantics of the COMEFROM Statement

F.X. Reid

[*Editors' note: Surprisingly, some of the early work of F. X. Reid has been almost entirely ignored and the following paper, reprinted here from the Proceedings of the Huddersfield Philosophical Society, is no exception. In his, as yet unpublished Autobiography* 'Biographia Informatica'*, which we have been privileged (or to be more accurate, badgered) to examine, Reid writes:*

> *I confess to have been disappointed by the reception of the Huddersfield paper. Its originality and importance can surely not be exaggerated extending as it did conventional notions of control flow and introducing at one blow, a combination of backtracking and non-determinism. Although developed in the spirit of the Floyd flow-diagram paradigm, it was quite capable of an extension to the more block-structured approach. However, my paper on the* ***while*** *P* ***undo*** *S* ***odnu*** *construct failed to appeal to the Zeigeist and had to be withdrawn. Such was the short-sightedness of my erstwhile colleagues.*

… and so on. We sought permission from Huddersfield to print the following extract but learned that the Society was no longer extant, as shortly after the publication of the paper it broke up following what one survivor described as 'a most un-philosophical punch-up in the Rat and Feathers*', adding darkly, 'publish the wretched thing – if you dare!'*

Our temerity thereby challenged – here it is. We skip the introduction, as we found it somewhat verbose.]

2 The Basic model

We consider a *program* in the abstract as consisting of:

1. A finite set V of *variable names*. For simplicity, we assume all variables to take values from the set $\mathbf{Z}$ of integers.
2. A finite set N of *lines* and a total order $R \subseteq N \times N$. Write n_o for the unique element of N such that $n \leq n_0$ all $n \in N$ and write n_∞ for the unique element of N such that $n_\infty \leq n$ all $n \in N$. We define $succ(n)$ to be the unique line covered by n, unless $n = n_\infty$ and we define $succ(n_\infty) = n_\infty$.
3. A labelling function $\lambda : N \rightarrow Stat$, where $Stat$ is a set of *statements.* Statements have one of the following three forms.

Skip Statements: There is only one of these taking the form $\mathbf{skip}$.

Assignment Statements: these take the form $v = E$, where $v \in V$ and E is an arithmetic expression as in the programming language FORTRAN.

Comefrom Statements: these take the form $\text{if } B \text{ comefrom } n$ where $n \in N$ and B is a Boolean expression as in FORTRAN.

[*Editors' note: In Reid's original paper, expressions were defined syntactically, using what he subsequently described as RBNF. The paper was, in fact, written before the publication of the ALGOL report. At least it is* dated *before that publication.*]

3 Operational Semantics

We define a *configuration* of the program to be a pair $(n,\varepsilon) \in N \times [V \to \mathbf{Z}_\perp]$, where $\mathbf{Z}_\perp = \mathbf{Z} \cup \{\perp\}$, where $\perp$ represents the totally undefined value. If $\sigma = (n,\varepsilon)$, then we define $n_\sigma = n$ and $\varepsilon_\sigma = \varepsilon$. Denote the set of all configurations by $Conf$.

If $\varepsilon \in [V \to \mathbf{Z}_\perp]$ and E is an arithmetic expression, then we define $E(\varepsilon)$ to be the value obtain by substituting $\varepsilon(v)$ for each variable v appearing in E. If $\varepsilon(v) = \perp$ for some v appearing in E, then $E(\varepsilon) = \perp$. We make analogous definitions for Boolean expressions.

If $\varepsilon \in [V \to \mathbf{Z}_\perp]$, $z \in \mathbf{Z}_\perp$ and $v \in V$, then we define

$$\varepsilon[v \setminus z](v') = \begin{cases} z \text{ if } v' = v \\ \varepsilon(v') \text{ otherwise} \end{cases}$$

If $\sigma, \sigma' \in Conf$, then we define $\sigma \vdash \sigma'$ as follows.

Skip. If $\lambda(n_\sigma)$ is $\mathbf{skip}$, then $n_{\sigma'} = succ(n_\sigma)$ and $\varepsilon_{\sigma'} = \varepsilon_\sigma$.

Assignment. If $\lambda(n_\sigma)$ is $v = E$, then $n_{\sigma'} = succ(n_\sigma)$ and $\varepsilon_{\sigma'} = \varepsilon_\sigma[v \setminus E(\varepsilon_\sigma)]$.

Comefrom. If $B(\varepsilon) = \mathbf{true}$, then $n_{\sigma'} = succ(n)$ and $\varepsilon_{\sigma'} \in undo(\varepsilon_\sigma, \lambda(n_\sigma))$; otherwise $n_{\sigma'} = succ(n_\sigma)$ and $\varepsilon_{\sigma'} = \varepsilon_\sigma$. Here $undo(\varepsilon_\sigma, \lambda(n)) = \{\varepsilon\}$ if $\lambda(n_\sigma)$ is not an assignment. Otherwise if $\lambda(n_\sigma)$ is $v = E$, then

$$undo(\varepsilon_\sigma, \lambda(n)) = \{\varepsilon' \in [V \to \mathbf{Z}_\perp] : E(\varepsilon') = \varepsilon_\sigma\}.$$

A *partial execution* of a program is a sequence $\sigma_1 \cdots \sigma_n$ such that σ_1 is of the form (n_0, ε) and $\sigma_i \vdash \sigma_{i+1}$, $i = 1, \cdots, n-1$. It is a *total execution* if in addition σ_n is of the form (n_∞, ε) and σ_i is not of this form $1 < i < n$.

[*Editors' note: At this point, Reid explains that the program fragment*

```
$n$        skip
$n+1$      $statement$
        ...
$n'$       if $x = 0$ comefrom $n$
```

has exactly the same effect as

```
$n+1$      $statement$
        ...
$n'$       if $x = 0$ goto $n$
```

As he makes rather a meal out of this rather obvious point, we omit it, together with his remarks on what he calls the Reid-Church-Turing Thesis*, which might cause distress to those of a nervous or choleric disposition.*]

Programs compute relations not functions. The relation computed by a program

$$F \subseteq [V \to \mathbf{Z}_\perp] \times [V \to \mathbf{Z}_\perp]$$

is defined as follows. $(\varepsilon, \varepsilon') \in F$ if and only if there exists a total execution $\sigma_1 \cdots \sigma_n$ such that $\sigma_1 = (n_0, \varepsilon)$ and $\sigma_n = (n_\infty, \varepsilon')$.

The following is an example program, in which $V = \{1,2,3,4\}$

```
1   $x = 1$
2   $y := x$
3   if $x \neq 0$ comefrom 1
4   skip
```

Applying the semantics, and representing a configuration by a triple (n, x, y), we have possible executions

$$(1, \perp, \perp) \vdash (2, 1, \perp) \vdash (3,1,1) \vdash (2, z, 1) \vdash (3, z, z) \vdash (2, 0, z) \vdash (4, 0, z).$$

Hence $((\perp, \perp), (0, z)) \in F$ for all $z \in \mathbf{Z}$. In other words, the program acts as a random number generator.

[*Editors' note: To return to 'Biographia Informatica', Reid writes:*

> *I regret not having included a section in which I demonstrated how* ***comefrom*** *programs may be used to solve what are now called Sudoku puzzles. It is a simple enough matter to Gödelise a Sudoku grid and to construct a primitive recursive predicate* done(n) *with the property that* done(n) *is true if and only if n is the Gödel number of a successfully completed puzzle. However, the chairman of the Society, a Mr. Bauls, convinced me (no easy task) that such puzzles would never enjoy any popularity and I reluctantly omitted the section in question. A pity.*

Reid finally summaries the paper at unnecessary length, and concludes with a discussion of implementation. As this involves a detailed description of Hermitian operators on Hilbert spaces, and as life is short, we have omitted it. Of course, Reid now claims to have invented quantum computing.]

BCS-FACS/FME Evening Seminar

Specifying Systems that Connect to the Physical World

Professor Cliff Jones

University of Newcastle

24 April 2006

5.45pm

We all know about developing programs from formal specifications. For "closed" systems, such methods offer a gold standard against which less formal approaches can be measured. But there is an increasing demand for "open systems" which interact with the physical world. The overall system might include sensors and actuators whose signals flow to and from some control program. The task of obtaining a specification for the control program can be more challenging than that of deriving a program from that specification. This talk argues that recording an initial specification of the behaviour of the whole system in the physical world gives a way to *derive* a specification of a control system and also to record precisely the assumptions being made about those components which sit outside the computer.

Refreshments will be served from 5.15pm

The seminar is free of charge and open to everyone. If you would like to attend, please email Paul Boca [Paul.Boca@virgin.net] your name by **19 April 2006**. Pre-registration is required, as security at the BCS Offices is tight.

Book Announcement

Modern Formal Methods and Applications
Edited by Hossam A. Gabbar
Springer

Modern Formal Methods and Applications

Gabbar, Hossam A. (Ed.)
2006, XXIII, 197 p., Hardcover
ISBN: 1-4020-4222-1

Formal methods is a robust approach for problem solving. It is based on logic and algebraic methods where problems can be formulated in a way that can help to find an appropriate solution. This book shows the basic concepts of formal methods and highlights modern modifications and enhancements to provide a more robust and efficient problem solving tool.

Applications are presented from different disciplines such as engineering where the operation of chemical plants is synthesized using formal methods. Computational biology becomes easier and systematic using formal methods. Also, hardware compilation and systems can be managed using formal methods.

This book will be helpful for both beginners and experts to get insights and experience on modern formal methods by viewing real applications from different domains.

Written for:

Undergraduate and graduate students, industrial professionals in engineering systems

Keywords:

Formal approach for biological systems
Formal approach for engineering systems
Formal methods
Language specifications
Software and hardware specifications

Paid-up FACS members are entitled to a **30%** discount on Springer titles. If you are interested in claiming this discount, please contact Springer directly on journalslondon@springer-sbm.com .

PhD Abstracts

Name	Greg Reeve
Title	A Refinement Theory for μ-Charts
Supervisor	Prof. Steve Reeves
Institute	University of Waikato
Examiners	Prof. Martin Henson & Prof. Jonathan Bowen
Awarded	December 2005
URL	http://www.cs.waikato.ac.nz/pubs/2005/pdfs/reeve-thesis.pdf
Keywords	State charts, μ-Charts, Z, Logic, Refinement

The language μ-Charts is one of many Statechart-like languages, a family of visual languages that are used for designing reactive systems. We introduce a logic for reasoning about and constructing refinements for μ-charts. The logic itself is interesting and important because it allows reasoning about μ-charts in terms of partial relations rather than the more traditional traces approach. The method of derivation of the logic is also worthy of report. A Z-based model for the language μ-Charts is constructed and the existing logic and refinement calculus of Z is used as the basis for the logic of μ-Charts. As well as describing the logic we introduce some of the ways such a logic can be used to reason about properties of μ-Charts and the refinement of abstract specifications into concrete realisations of reactive systems.

A refinement theory for Statechart-like languages is an important contribution because it allows us to formally investigate and reason about properties of the object language μ-Charts. In particular, we can conjecture and prove general properties required of the object language. This allows us to contrast possible language design decisions and comment on their consequences with respect to the design of Statechart-like languages.

This thesis gives a comprehensive description of the μ-Charts language and details the development of a partial relations based logic and refinement calculus for the language. The logic and refinement calculus are presented as natural deduction style proof rules that allow us to give formal proofs of language properties and provide the basis for a formal program development framework. The notion of refinement that is encoded by the refinement rules is also extensively investigated.

Preliminary Announcement

BCS-FACS Christmas Meeting on

Teaching Formal Methods – Practice and Experience

15 December 2005

BCS London Offices

First Floor, The Davidson Building
5 Southampton Street
London WC2E 7HA

http://cms.brookes.ac.uk/tfm2006/
http://www.bcs-facs.org/events/xmas2006.html

This workshop will give teachers of formal methods an opportunity to discuss their experiences in this area, to share successes and failures, to identify issues in teaching formal methods and discuss how they might be addressed. Topics to be covered include:

- How to motivate the study of formal methods;
- Techniques for teaching formal methods;
- Handling students with limited mathematical backgrounds;
- Linking formal methods and software development;
- Tools for teaching formal methods (including demonstrations);
- How to assess formal methods.

For more information, and registration details (discounts for BCS-FACS members), please visit either of the web pages above, or e-mail Professor David Duce [daduce@brookes.ac.uk].

BCS FACS

FACS membership application/renewal (2006)

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)

ALL MEMBERS WILL RECEIVE FREE ELECTRONIC ACCESS TO THE *FORMAL ASPECTS OF COMPUTING* JOURNAL UNTIL THE END OF DECEMBER 2006

I would like to subscribe to **Volume 18 of the FAC journal** (paper copy) at the following rate:

- ❑ £48

The total amount payable to BCS-FACS in **pounds sterling** is **£ 15 / 30 / 63 / 78** (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:

FACS Committee

Jonathan Bowen
FACS Chair
ZUG Liaison

Jawed Siddiqi
Treasurer

Paul Boca
Secretary and
Newsletter Editor

Executive Officers

Roger Carsley
Minutes
Secretary

John Cooke
FAC Journal
Liaison

John Fitzgerald
FME Liaison
SCSC Liaison

Judith Carlton
Industrial Liaison

Margaret West
BCS Liaison

Rick Thomas
LMS Liaison

Mark D'Inverno
Model-based
specification

John Derrick
Refinement

Rob Hierons
Formal methods
and testing

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

BCS FACS
c/o Professor 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 officers 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.

Coming Soon in FACS FACTS….

Details of upcoming FACS Evening Seminars

TRain Column **Conference reports**

Book Reviews **Report on GC6 activities**

Report on History of Formal Methods Panel

And More…