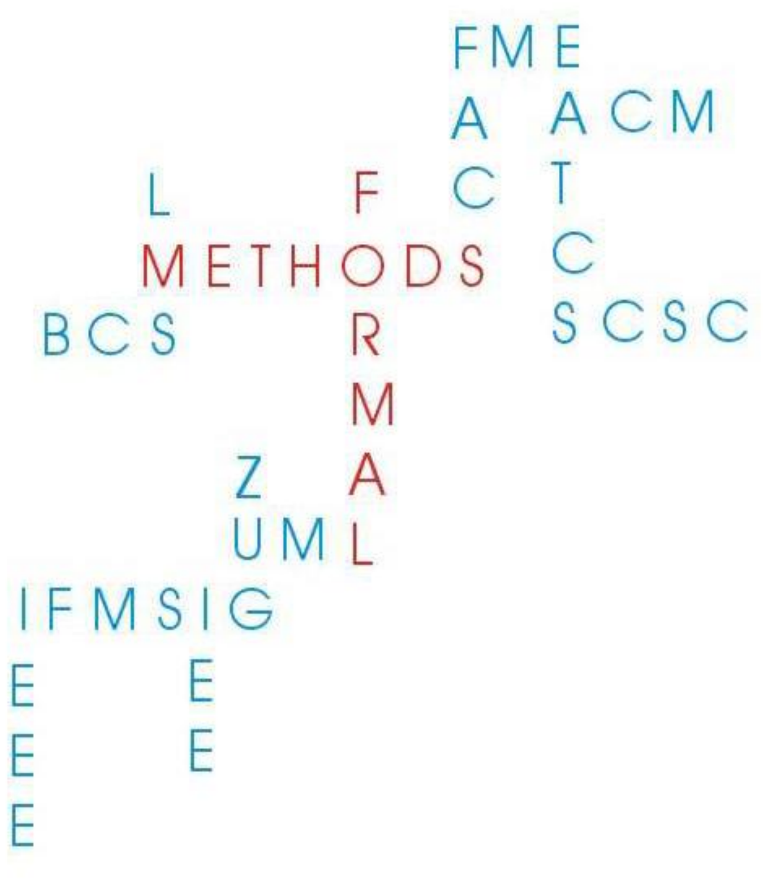
Issue 2015-1

March 2015

FACS

A C T S



bcs The Chartered Institute for IT

Formal Aspects of Computing Science Specialist Group

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.

Submissions to FACS FACTS are always welcome. Please visit the newsletter area of the BCS FACS website for further details (see http://www.bcs.org/category/12461).

Back issues of *FACS FACTS* are available for download from: http://www.bcs.org/content/conWebDoc/33135

The FACS FACTS Team

Newsletter Editors	Tim Denvir	timdenvir@bcs.org
	Brian Monahan	brianqmonahan@googlemail.com

Editorial Team Jonathan Bowen, Tim Denvir, Brian Monahan, Margaret West.

Contributors to this Issue

Jonathan Bowen, Eerke Boiten, Tim Denvir, Brian Monahan, Margaret West.

BCS-FACS websites

BCS:	http://www.bcs-facs.org
LinkedIn:	http://www.linkedin.com/groups?gid=2427579
Facebook:	http://www.facebook.com/pages/BCS-FACS/120243984688255
Wikipedia:	http://en.wikipedia.org/wiki/BCS-FACS

If you have any questions about BCS-FACS, please send these to Paul Boca <paul.boca@googlemail.com>

Editorial

Welcome to issue 2015-1 of *FACS FACTS*.

The FACS AGM was held on 8th December 2014 and was followed by the Peter Landin Annual Semantics Seminar, given by Professor Peter Mosses. An abstract and brief report of his talk can be found below.

This issue of FACS FACTS also contains a report by Margaret West on the Lovelace Lecture given by Professor Samson Abramsky, “Contextual Semantics: From Quantum Mechanics to Logic, Databases, Constraints, Complexity, and Natural Language Semantics” on 5th June 2014; and an abstract of the BCS-FACS Evening Seminar: “Decision Problems for Linear Recurrence Sequences”, a joint event with the London Mathematical Society held on: Wednesday 22nd October 2014, given by Professor Joel Ouaknine.

An article by Eerke Boiten, “It’s possible to write flaw-free software, so why don’t we?” is reproduced with permission. This article is aimed at a more general audience, but is a good example of the kind of formal methods dissemination piece that readers of FACS FACTS are encouraged to emulate.

The Forthcoming Events includes an announcement of the BCS-FACS ProCoS Workshop on 9-10 March 2015, and a detailed programme is also provided. This is now of course no longer “forthcoming”; time has, regrettably, overtaken us.

Brian Monahan has written an “opinion piece”: “The Future of High-precision Programming”, in which he relates the programming of complex systems to engineering, specification and composition of subsystems. Two book reviews follow, on Paul Butcher’s “Seven Concurrency Models in Seven Weeks” and Richard Bird’s “Thinking Functionally with Haskell”, both again by Brian Monahan.

Most FACS seminars take place in the offices of the British Computer Society in the Davidson Building, Southampton Street. These excellent facilities are conveniently situated in Central London close to Covent Garden and we would like to thank them for making these available to us.

Forthcoming Events

Forthcoming events from the Formal Aspects of Computing Science (FACS) Group are listed below:

Date	Details
9-10 March 2015	**Title:** BCS FACS - ProCoS Workshop on Provably Correct Systems **Venue:** BCS, London
22 June 2015	**Title:** BCS FACS - Refinement Workshop **Venue:** Oslo
16 September 2015	**Title:** BCS FACS - an evening Seminar with Prof. Ian Hayes **Venue:** BCS, London
3 November 2015	**Title:** Joint FACS/LMS seminar - speaker: Professor Roland Backhouse, Nottingham University **Details to follow**

(See: Forthcoming Events for up-to-date information.)

Lovelace Lecture

Contextual Semantics: From Quantum Mechanics to Logic, Databases, Constraints, Complexity, and Natural Language Semantics

5 June 2014
Imperial College, London

Professor Samson Abramsky
(University of Oxford)

Reported by: Margaret West

Introduction

The lecture commenced with a welcome by Professor Jeff Magee of Imperial College, the chair of the BCS Academy. The BCS Lovelace Medal was first presented in 1998 and is named after Ada Lovelace, a mathematician and scientist who worked with Charles Babbage. The medal is for *individuals who have made an outstanding contribution to the understanding or advancement of Computing.*

He then introduced us to the 2013 winner of the medal - Professor Samson Abramsky who is Christopher Strachey Professor of Computing and a Fellow of Wolfson College, Oxford University.

A brief resume of the academic achievements of Samson Abramsky followed which included his LiCS Test-of-Time award for his 1987 paper *Domain Theory in Logical Form* . He had played a leading role in the field of game semantics and its application to programming language semantics. In addition he had worked on the lazy lambda calculus, strictness analysis, concurrency theory, interaction categories and geometry of interaction. His recent work involves high-level methods for quantum computation and in 2007 he was awarded an EPSRC Senior Research Fellowship on Foundational Structures and Methods for Quantum Informatics and in 2013 was awarded the BCS Lovelace Medal.

The medal was presented by Professor Philippa Gardener of Imperial College who chaired the BCS Academy Awards Committee.

Professor Bill Roscoe of the University of Oxford was the next speaker and he spoke briefly about the award winner's involvement with the subject of the lecture. He said that when he first took up his post at Oxford his colleagues thought they were getting a classical Computer Scientist "so who would have thought that they were getting a theoretical quantum scientist as well". Samson Abramsky would also participate in the University's bid for EPSRC funding in the field of Quantum Technologies and Bill Roscoe remarked on the tremendous bravery with which Samson would subsequently invade the territory of the established quantum theorist.

Note: On November 26th the results of the competitive peer reviewed process were announced – and the University of Oxford was one of the Universities selected as one of the Quantum Technology Hubs, that will explore the properties of quantum mechanics and how they can be harnessed for use in technology. See http://www.epsrc.ac.uk/newsevents/news/quantumtechhubs/

Talk

The talk commenced with an acknowledgement: such an award recognizes the **research community** which makes work of this kind possible. Samson also made the point that it was pleasing that the BCS (via the award) rewards scientific discipline in its own right and not just for its applications. He next provided a brief career history, and in so doing recognised the communities which were an important part of it.

He then noted that one of the first axioms of Computer Science was *that computers might as well be made out of green cheese.* Thus device physics is immaterial once we understand the logical computer model. For many purposes this is very useful in that we do not need to worry about physics/hardware and can abstract away from it. However this now cannot be assumed when we consider the interplay between computer science and physics in, for example, cyber computation. Thus there is an exciting two way interplay between Physics and Computer Science which extends to the foundations of both, as well as to more practical matters. The talk then focussed on some non intuitive phenomena of Quantum Informatics: viz **contextuality, entanglement and non-locality** which have profound consequences for an understanding of the very nature of physical reality.

In order to illustrate, an example was provided of two agents, Alice and Bob who each have two local bit registers from which to extract information. Alice reads from a_1 or a_2 and Bob reads from b_1 or b_2, where each register from $\{ a_1, a_2, b_1, b_2 \}$ can contain a **1** or a **0** . The contents of their **read** is then combined and focussed on some Target.

These registers are loaded randomly – so it is not known beforehand what Bob or Alice might read from the registers. However if this is repeated we might extract some statistics from the outcomes. Thus a probability table (or *Bell model*) of the experiment might look like this:

A	B	(0,0)	(1,0)	(0,1)	(1,1)
a_1	b_1	1/2	0	0	1/2
a_1	b_2	3/8	1/8	1/8	3/8
a_2	b_1	3/8	1/8	1/8	3/8
a_2	b_2	1/8	3/8	3/8	1/8

A possible explanation of this in a classical sense would be a probabilistic source writing to the registers where it chooses which values to write from maybe the tossing of several coins or from sampling some probability distribution.

However there is another and much simpler way we can represent this – by replacing the probabilistic model with a **possibilistic model**. In the corresponding possibility table which follows there is just a binary distinction between the outcomes where a “1” indicates there is a possibility of that combination and a “0” indicates there is none and in addition much of the information from the first table is thrown away. The registers can now be regarded as **measurements.**

A	B	(0, 0)	(0, 1)	(1, 0)	(1, 1)
a_1	b_1	1			
a_1	b_2	0			
a_2	b_1	0			
a_2	b_2				0

Can we explain this using a classical source? Objective properties for registers are independent of our choice of which measurement to perform and these are what we would assume for such a source. However such **non-contextuality** of measurement is found to be untrue if **actual** measurements are taken in experimental micro-physics. Samson Abramsky remarked that this fact is both a challenge and an opportunity.

This is reflected in the possibility table (or *Hardy model*) for if we examine the first two elements of column one of the table plus the element in the last row and column and attempt to assign values to registers { a_1, a_2, b_1, b_2 } we see that the only possible assignment is:

$$a_1 \rightarrow 0,\ a_2 \rightarrow 0,\ b_1 \rightarrow 1,\ b_2 \rightarrow 1$$

However if we check further we see that this assignment of values to registers in the classical sense is inconsistent with the value zero in column 1, row 3 for it precludes the outcome (0, 0) for measurements (a_2, b_1). Thus Hardy models are contextual and cannot be explained by a classical source and this is known as the *Hardy Paradox*.

If we use **quantum** resources as opposed to **classical** resources this configuration can be achieved. Registers are replaced by a suitable entangled state of two *qbits* and *spin* measurement directions a_1, a_2 for Alice and b_1, b_2 for Bob where directions have a binary choice – *up* or *down.* Further, Alice and

Bob may be a distance apart. It is then possible to replicate the above table and extensive experimentation has confirmed this. In quantum mechanical terms even if particles are spatially separated measuring one has an effect on the state of the other.

Samson remarked that this proves a **strong version** of Bells Theorem, the word strong indicating that possibilities are considered as opposed to probabilities.

The speaker went on to discuss the mathematics of possibility tables and further developed this into a Contextual Geometry. He explained how non-locality and contextuality fitted into this geometrical model. Further, there was an isomorphism between these formal descriptions and basic definitions and concepts in relational database theory. Thus databases can be considered as possibility tables and a dictionary can be developed between relational databases and measurement scenarios. Examples include *attribute* which corresponds to *measurement* and *database schema* which corresponds to *measurement cover.* It seems that the phenomenon of contextuality is pervasive and once we look for it we can find it everywhere.

Further work in contextual semantics was then briefly discussed with some current developments in quantum information and foundations. This was followed by an outline of some current research in contextual semantics in *classical* computation where this was related both to constraint satisfaction and to natural language semantics. Samson Abramsky then introduced us to some of the other people involved in contextual semantics – his "comrades in arms" – and also to the Oxford University Quantum Group.

The speaker concluded with some enduring thoughts: the fact that Computer Science is a fundamental discipline among the sciences which both illuminates and interacts with them. It is important both for its modes of thought, and for its subject matter. Samson concluded by saying we should not limit ourselves, but dare to think BIG.

Q and A

This was chaired by Professor Jeff Magee of Imperial College. The session included a query and further discussion about the "strong" version of Bells theorem and there was a question about Natural Language: *Are our brains contextual agents because of our ability to interpret?*

Samson Abramsky was also asked if he drew any conclusions about the teaching of Computer Sciences given the nature of Quantum Informatics. He remarked that the development of Computer Science over the years presented a challenge to teaching as to what kind of mathematics is relevant or useful? This has now grown to include probability theory and continuous mathematics. He thought it a positive result in that people from varied backgrounds achieve a common language and he thought we should aspire in our teaching.

The vote of thanks was given by Professor Adam Brandenburger of the University of New York in which he included an appropriate quotation from *The Moonstone* (Wilkie Collins) by one of the narrators, Gabriel Betteridge:

> *"I arose the next morning with the objective subjective and the subjective objective inextricably entangled together in my mind".*

He added that the novel was a mood altering and mind bending piece of art influenced by laudanum but there was no mention of Quantum Mechanics in it. However in spite of this he believed there was no better way of summing up the lecture than that tomorrow morning we would all rise with the objective subjective and the subjective objective inextricably entangled in our minds but entangled in the most wonderful and educational manner.

He addressed Samson directly: "You are an intellectual colossus standing astride disciplines and fields. You see more than almost anybody else and you see it, write it down and even more can communicate it in such an effective and educational forum and thank you for that."

He further thanked both Imperial College and the BCS Academy for their arrangements in putting on the lecture and in particular the Society for its excellent judgment in choosing Samson as an awardee.

He ended (in the traditional manner) by announcing the 2014 Lovelace Medal winner - Professor Steve Furber of Manchester University who will give the next lecture in spring 2015

The talk is available at https://www.youtube.com/watch?v=lE0WyhSy7lg

It's possible to write flaw-free software, so why don't we?

by Eerke Boiten
(University of Kent)

If Spock would not think it illogical, it's probably good code. Alexandre Buisse, CC BY-SA

Legendary Dutch computer scientist Edsger W Dijkstra famously remarked that "testing shows the presence, not the absence of bugs". In fact the only definitive way to establish that software is correct and bug-free is through mathematics.

It has long been known that software is hard to get right. Since Friedrich L Bauer organised the very first conference on "software engineering" in 1968, computer scientists have devised methodologies to structure and guide software development. One of these, sometimes called strong software engineering or more usually formal methods, uses mathematics to ensure error-free programming.

As the economy becomes ever more computerized and entwined with the internet, flaws and bugs in software increasingly lead to economic costs from fraud and loss. But despite having heard expert evidence that echoed Dijkstra's words and emphasises the need for the correct, verified software that formal methods can achieve, the UK government seems not to have got the message.

Formal software engineering

The UK has always been big in formal methods. Two British computer scientists, Tony Hoare (Oxford 1977–, Microsoft Research 1999–) and the late Robin Milner (Edinburgh 1973–95, Cambridge 1995–2001) were given Turing Awards – the computing equivalent of the Nobel Prize – for their work in formal methods.

British computer scientist Cliff B Jones was one of the inventors of the Vienna Development Method while working for IBM in Vienna, and IBM UK and Oxford University Computing Laboratory, led by Tony Hoare, won a Queen's Award for Technological Achievement for their work to formalise IBM's CICS software. In the process they further developed the Z notation which has become one of the major formal methods.

The formal method process entails describing what the program is supposed to do using logical and mathematical notation, then using logical and mathematical proofs to verify that the program indeed does what it should. For example, the following Hoare logic formula describing a program's function shows how formal methods reduce code to something as irreducibly true or false as 1 + 1 = 2.

Taught at most UK universities since the mid-1980s, formal methods have seen considerable use by industry in safety-critical systems. Recent advances have reached a point where formal methods' capacity to check and verify code can be applied at scale with powerful automated tools.

$$\frac{P\{\mathtt{S}\}Q \quad Q\{\mathtt{T}\}R}{P\{\mathtt{S;T}\}R}$$

Hoare logic formula: if a program S started in a state satisfying P takes us to a state satisfying Q, and program T takes us from Q to R, then first doing S and then T takes us from P to R.

Government gets the message

Is there any impetus to see them used more widely, however? When the Home Affairs Committee took evidence in its E-crime enquiry in April 2013, Professor Jim Norton, former chair of the British Computer Society, told the committee:

We need better software, and we know how to write software very much better than we actually do in practice in most cases today… We do not use the formal mathematical methods that we have available, which we have had for 40 years, to produce better software.

Based on Norton's evidence, the committee put forward in recommendation 32 "that software for key infrastructure be provably secure, by using mathematical approaches to writing code."

Two months later in June, the Science and Technology Committee took evidence on the Digital by Default programme of internet-delivered public services. One invited expert was Dr Martyn Thomas, founder of Praxis, one of the most prominent companies using formal methods for safety-critical systems development. Asked how to achieve the required levels of security, he replied that:

Heroic amounts of testing won't give you a high degree of confidence that things are correct or have the properties you expect… it has to be done by analysis. That means the software has to be written in such a way that it can be analysed, and that is a big change to the way the industry currently works.

The committee sent an open letter to cabinet secretary Francis Maude in asking whether the government "was confident that software developed meets the highest engineering standards."

Trustworthy software is the answer

The government, in its response to the E-crime report in October 2013 , stated:

The government supports Home Affairs Committee recommendation 32. To this end the government has invested in the Trustworthy Software Initiative*, a public/private partnership initiative to develop guidance and information on secure and trustworthy software development.*

This sounded very hopeful. Maude's reply to the Science and Technology committee that month was not published until October 2014, but stated much the same thing.

So one might guess that the TSI had been set up specifically to address the committee's recommendation, but this turns out not to be the case. The TSI was established in 2011, in response to governmental concerns over (cyber) security. Its "initiation phase" in which it drew from academic expertise on trustworthy software ended in August 2014 with the production of a guide entitled the Trustworthy Security Framework, available as British Standards Institute standard PAS 754:2014.

This is a very valuable collection of risk-based software engineering practices for designing trustworthy software (and not, incidentally, the "agile, iterative and user-centric" practices described in the Digital by Default service manual). But so far formal methods have been given no role in this. In a keynote address at the 2012 BCS Software Quality Metrics conference, TSI director Ian Bryant gave formal methods no more than a passing mention as a "technical approach to risk management".

So the UK government has been twice advised to use mathematics and formal methods to ensure software correctness, but having twice indicated that the TSI is its vehicle for achieving this, nothing has happened. Testing times for software correctness, then – this is something that will continue for as long as it takes for Dijkstra's message to sink in.

Editors' Note

FACS readers are very much encouraged to follow this example and spread the word by writing similar articles that are aimed at a wider audience.

BCS FACS – ProCos Workshop on Provably Correct Systems

http://www.bcs.org/content/ConWebDoc/53939

Talks on YouTube

https://www.youtube.com/channel/UC_S0VLVAAomm05otUi4_onQ

Date/Time: Monday 9 March – Tuesday 10 March 2015
Venue: BCS, 1st Floor, The Davidson Building, 5 Southampton Street, London, WC2E 7HA | Map

Cost: £60.00 for BCS Members & Students; £120.00 for Non-members

Book Online

Book Online for the dinner

Co-chairs:
Prof. Jonathan Bowen, Birmingham City University, UK
Prof. Mike Hinchey, LERO, University of Limerick, Republic of Ireland
Prof. Dr Ernst-Rüdiger Olderog, Carl von Ossietzky Universität Oldenburg , Germany

The years 2014 and 2015 mark 25 years and 20 years, respectively, since the start and end of the European ESPRIT ProCoS projects on *Provably Correct Systems*, inspired by the CLInc project in the US. The ProCoS I/II projects and the associated ProCoS-US initiative ran from 1989-1995, followed by the ProCoS-WG Working Group of 25 partners. The projects aimed to perform research in the fundamental technical aspects of a development process for critical embedded systems, from the original capture of requirements all the way down to the computers and special purpose hardware on which the programs run. The projects were significant in their contributions to provably correct systems, and led directly to a better general understanding of the relationship between a range of theories, and how their combination can be used in the planning and development of critical software tasks. This event marks these 20th and 25th anniversaries of ProCoS to look back at its achievements and to identify key research that will contribute to the next generation of provably correct systems, with invited talks by leading international computer science researchers, many directly involved with the original ProCoS projects.

Sponsored by Lero (The Irish Software Research Centre)

Programme

Monday 9 March 2015 (Whence)

09.00–09.30 Registration

09.30–11.00 Session 1 (Introduction) – Chair: Prof. Dr Ernst-Rüdiger Olderog, Carl von Ossietzky Universität Oldenburg , Germany

How it all Began: As seen from Denmark – Prof. Dines Bjørner, Technical University of Denmark, Denmark

Provably Correct Systems: Whence and whither? – Prof. Jonathan P. Bowen, Birmingham City University, UK

Algebraic Proof of Consistency of Operational and Verification Semantics – Prof. Tony Hoare, Microsoft Research Cambridge, UK

11.00–11.30 Coffee/tea break

11.30–13.00 Session 2 (Hybrid systems) – Chair: Prof. Jonathan Bowen, Birmingham City University, UK

Hybrid Systems from the ProCoS Gas Burner to Highway Traffic – Prof. Anders P. Ravn, Aalborg University, Denmark

Engineering Arithmetic Constraint Solvers for Automatic Analysis of Hybrid Discrete-continuous Systems – Prof. Dr Martin Fränzle, Carl von Ossietzky Universität Oldenburg , Germany

Hybrid Relation Calculus – Prof. Jifeng He, East China Normal University, China

13.00–14.00 Lunch break

14.00-16.00 Session 3 (Reasoning, Analysis & Refinement) - Chair: Prof. Mike Hinchey, LERO, University of Limerick, Republic of Ireland

Reasoning Abstractly about Concurrency - Prof. Cliff Jones, Newcastle University, UK

From ProCoS to Space and Mind-models - Prof. Dr Bettina Buth, HAW Hamburg, Germany

Refinement Algebra and Applications - Prof. Augusto Sampaio, Universidade Federal de Pernambuco, Brazil

Space for Traffic Manoeuvres - Prof. Dr Ernst-Rüdiger Olderog, Carl von Ossietzky Universität Oldenburg , Germany

16.00-16.30 Coffee/tea break

16.30-18.30 Session 4 (Mechanization) - Chair: Prof. Dr Debora Weber-Wulff, Hochschule für Technik und Wirtschaft Berlin, Germany

Model Checking Duration Calculus: The DCVALID story - Dr Paritosh Pandya, Tata Institute of Fundamental Research, India

Automatic Verification of Infinite-state Systems - Prof. Dr Markus Müller-Olm, Westfälische Wilhelms-Universität Münster, Germany

Commercial Use of the ACL2 System - Prof. Warren Hunt, University of Austin, Texas, USA

Managing Large Terms Representing Realistic Machine States - Prof. J Strother Moore, The University of Texas at Austin, USA

18.30-20.00 Reception

Following the BCS-FACS SG "ProCoS Workshop" - you are invited to dinner at Carluccio's, Covent Garden

Book Online for the dinner

Tuesday 10 March 2015 (Whither)

9.00-10.30 Session 1 (Assertions & Testing) - Chair: Prof. Michael R. Hansen, Technical University of Denmark, Denmark

Run-time Assertion Checking of Data- and Protocol-oriented Properties of Java Programs - Prof. Frank de Boer, CWI, Netherlands

Assertions for Hardware - Prof. Wayne Luk, Imperial College London, UK

Combining Testing and Verification - Prof. Dr Heike Wehrheim, University of Paderborn, Germany

10.30-11.00 Coffee/tea break

11.00-12.30 Session 2 (Proof) – Chair: Dr Hans Rischel, Technical University of Denmark, Denmark

Proof with Event-B/Rodin - Prof. Michael Butler, University of Southampton, UK

Are We There Yet? Twenty years of industrial theorem proving with SPARK - Dr Rod Chapman, Protean Code Ltd, UK

What have we Learned about Proof? - Prof. Ursula Martin, University of Oxford, UK

12.30-13.30 Lunch break

13.30-15.00 Session 3 (Models & ATP) – Chair: Dr Huibiao Zhu

Model-checking Extended Linear Duration Invariants - Prof. Naijun Zhan, Institute of Software, Chinese Academy of Sciences, China

A Model of Cyber-physical Component Systems - Prof. Zhiming Liu, Birmingham City University, UK

Advances in Connection-based Automated Theorem Proving - Prof. Dr Wolfgang Bibel, Darmstadt University of Technology, Germany and Prof. Dr Jens Otten, Potsdam University, Germany

15.00-15.30 Coffee/tea break

15.30-16.30 Session 4 (Correctness) – Chair: Prof. Jim Woodcock, University of York, UK

Synthesis of Provably Correct Systems - Prof. Dr Bernd Finkbeiner, Saarland University, Germany

Linearizability and Correctness for Weak Memory Models - Prof. John Derrick, University of Sheffield, UK

16.30-16.35 Close

BCS-FACS Evening Seminar

Decision Problems for Linear Recurrence Sequences

Joint event with the London Mathematical Society
(held on: Wednesday 22nd October 2014)

Professor Joel Ouaknine
(University of Oxford)

Linear recurrence sequences (LRS), such as the Fibonacci numbers, permeate vast areas of mathematics and computer science. In this talk, Professor Ouaknine considers three natural decision problems for LRS, namely the Skolem Problem (does a given LRS have a zero?), the Positivity Problem (are all terms of a given LRS positive?), and the Ultimate Positivity Problem (are all but finitely many terms of a given LRS positive?). Such problems (and assorted variants) have applications in a wide array of scientific areas, such as theoretical biology (analysis of L-systems, population dynamics), economics (stability of supply-and-demand equilibria in cyclical markets, multiplier-accelerator models), software verification (termination of linear programs), probabilistic model checking (reachability and approximation in Markov chains, stochastic logics), quantum computing (threshold problems for quantum automata), discrete linear dynamical systems (reachability and invariance problems), as well as combinatorics, statistical physics, formal languages, etc.

Perhaps surprisingly, the study of decision problems for LRS involves advanced techniques from a variety of mathematical fields, including analytic and algebraic number theory, Diophantine geometry, and real algebraic geometry.

The slides from this talk can be found here; various relevant papers are:

- On termination of integer linear loops @ SODA 15
- Ultimate Positivity is decidable for simple linear recurrence sequences @ ICALP 14 *(Best Paper Award)*
- On the Positivity Problem for simple linear recurrence sequences @ ICALP 14
- Positivity problems for low-order linear recurrence sequences @ SODA 14

BCS FACS - Annual Peter Landin Semantics Seminar 2014

On correspondences between programming languages and semantic notations

(held on: Monday 8 December 2014)

Professor Peter Mosses
(Swansea University)

Abstract

50 years ago, at the IFIP Working Conference on Formal Language Description Languages, Peter Landin presented a paper on "A formal description of ALGOL 60". In it, he explained "a correspondence between certain features of current programming languages and a modified form of Church's λ-notation", and suggested using that as the basis for formal semantics. He regarded his formal description of ALGOL 60 as a "compiler" from ALGOL abstract syntax to λ-notation.

10 years later, denotational semantics was well established, and two denotational descriptions of ALGOL 60 had been produced as case studies: one in the VDM style developed at IBM-Vienna, the other in the continuations-based style adopted in Christopher Strachey's Programming Research Group at Oxford.

After recalling Landin's approach, I'll illustrate how it differs from denotational semantics, based on the ALGOL 60 descriptions. I'll also present a recently developed component-based semantics for ALGOL 60, involving its translation to an open-ended collection of so-called fundamental constructs. I'll assume familiarity with the main concepts of denotational semantics.

This seminar presented by Professor Peter Mosses was introduced by Professor Tony Clark, Middlesex University. Tony Clark writes:

> Peter Landin
>
> ———————
>
> I was Peter Landin's last PhD student 1989-1996 studying the semantics of Object-Oriented Programming Languages. After a while we settled in to regular fortnightly meetings and, despite digital evidence to the contrary, I was always aware that Peter was engaged with what was going on in Programming Language Research and generally active.
>
> As the PhD progressed, I started to detect an approach that has made a lasting impression on me. Peter would always engage with a problem by trying to identify the essential features of a concept: what *is* this thing if it is shorn of all modish adornments? Taking this approach, my PhD broadened into a series of lengthy discussions on subjects including how to embed Prolog in Lambda (without continuations), exploring the inside of environment structures during program execution, and how to draw out the machinations of a type-checker using a modification of Cuisenaire Rods.
>
> Peter is known for lambda, program transformation, algebraic foundations, SECD, continuations, and perhaps less well known for influencing VDM and Scheme. A quote from John Reynolds (*Theories of Programming Languages*, Cambridge University Press, 1998) captures my experience of Peter: "Peter Landin remarked long ago that the goal of his research was "to tell beautiful stories about computation". Since then many researchers have told many such stories. This book is a collection of my favorites in the area of languages for programming and program specification".
>
> Peter produced some seminal papers during the 1960s that I am sure we all know. When preparing for this introduction, I recalled a paper that exemplified the beautiful stories, but had great difficulty tracking it down. I eventually found it in Samson Abramsky's introduction to his contribution to the special issue of Higher Order and Symbolic

Computation in honour of Peter. It turns out to be the last paper Peter published before his interests broadened and is well worth tracking down.

In 1969, Peter is recorded as saying: For some years I have aspired to 'language-free programming'. He seemed to return to this again during the years that I studied with him.

During our meetings he would discuss a new course he was developing on first year programming. Around 2000 he sent me a copy (330 pages) of notes for a text-book that he hoped to publish. Sadly this did not happen, but it would be great if we could collectively find a way of publishing them.

Peter's final work appears to return to the notion of a basis for language-free programming. These took the form of some (unfinished as far as I know) extensive notes on what he called Calculations. (*Calculations*, Peter J. Landin, in Higher Order and Symbolic Computation, (2009) 22: 333-359).

From his notes Peter listed his motivations as including: 'being persuasive about the intuitions that guide small design choices; to explore the elementary concepts of computing without mentioning programs; to explain something full of implications regarding algebra without actually resorting to it; to present, paradoxically, a wholly textual, picture free explanation of a highly pictorial concept'. Peter sent me a version of these notes and they appear as part of the special issue honouring him.

To sum up, Peter had an enormous effect on me as I am sure he did on many people that he worked with professionally. His work continues to have influence today, and, as I have mentioned there are a few little known gems to be discovered in the archive. Although we would consider much of his work to be foundational: as he paraphrased to me, 'There are still many more tunes left to be played in C'.

Tony Clark then introduced the speaker, Peter Mosses:

Professor in Computer Science, Swansea University.

DPhil from Oxford

Basic Research in Computer Science (BRICS), Denmark.

Computer Science, Swansea.

Member of several IFIP working groups including Chair of IFIP 1.3 Foundations of System Specification 98-2003.

Editorial/Advisory Board member of several leading journals including Science of Computer Programming and Higher-Order and Symbolic Computation.

Key Contributions:

Action Semantics.

Modular approaches to Programming Languages and SOS.

Current Project: Programming Language Components: partners RHUL, City University and Microsoft, funded by EPSRC.

In his seminar, Peter Mosses dwelt on the developing history of semantics, starting in the 1960s, moving to the 1970s, and finally outlining work in the current PlanCompS project. We were at the 50th anniversary of the first IFIP TC2 Working Conference on *Formal Language Description Languages*, held in 1964 with proceedings published in 1966. There were 50 invited participants and seminal papers by Peter Landin, Christopher Strachey and many others.

In 1964-1966 Peter Landin published several significant papers on the formal description of languages:

- The mechanical evaluation of expressions
- A correspondence between ALGOL 60 and Church's lambda-notation
- A formal description of ALGOL 60
- A generalization of jumps and labels
- The next 700 programming languages

The mechanical evaluation of expressions (Computer Journal (1964) 6) employed *Applicative Expressions* (AEs) which generally have values in a given environment *E*. In 1964-65 Landin published a two-part paper in Communications of the ACM on a *Correspondence between ALGOL 60 and Church's lambda notation*. This employed *Imperative Applicative Expressions* (IAEs) and drew a correspondence between ALGOL abstract syntax and IAEs via semantic functions. Landin was an advisor on the official language definition for ALGOL 60 and in the proceedings of IFIP TC2 above published *A Formal Description of ALGOL 60*.

Peter Mosses then compared Christopher Strachey's approach to language semantics (the language in this case being CPL) with that of Landin, showing the virtues and drawbacks of each.

The 1970s saw work beginning on denotational semantics at Oxford by Dana Scott, Christopher Strachey, Peter Landin, John Reynolds and many others. Scott and Strachey's Oxford Technical Monograph PRG-6 *Towards a mathematical semantics for computer languages* paved the way. It showed the correspondence between program phrases and their *denotations* in Scott-domains (originally *lattices*, later *CPO*s). The least fixed-point operator *Y* was no longer, in Strachey;s words, "paradoxical". Peter Mosses himself published Technical Monograph PRG-12 *The mathematical semantics of ALGOL 60*, exploiting a continuations style. Then in 1974 H. Bekić, D. Bjørner, W. Henhapl, C. B. Jones, P. Lucas from the IBM Vienna Laboratory published *A formal definition of a PL/I subset* (Tech. Rep. TR 25.139, IBM Lab. Vienna, Dec. 1974). In the late 1970s W. Henhapl and C. B. Jones published *A formal definition of ALGOL 60* in "The Vienna Development Method: The Meta-Language", LNCS 61: 305-336, 1978, and Chapter 6 of "Formal Specification & Software Development", Prentice-Hall, 1982. Further reading can be found in P. D. Mosses: *VDM semantics of programming languages: combinators and monads* in Formal Aspects of Computing (2011) 23: 221-238 and C. B. Jones: Semantic descriptions library
homepages.cs.ncl.ac.uk/cliff.jones/semantics-library/

Finally Peter Mosses described a current project: *Component-based semantics*. The aim is to have reusable components corresponding to program constructs with fixed notation behaviour and algebraic properties, "specified and proved once and for all!". This is part of the PlanCompS project, which is running from 2011 to 2015.

Slides of Peter Mosses' seminar can be found at:

http://www.plancomps.org/landin-seminar-2014/

The future of high-precision programming?

by Brian Monahan

Programming is an important form of engineering – it's about creating and running systems that behave in some well-defined manner and that ideally always achieve the goals they were designed for. However, it is often not feasible or simply too costly in various ways to design and build systems that always behave perfectly under all reasonable circumstances – in which case, we instead want systems to achieve their design goals as well as possible under the prevailing circumstances, whatever that turns out to be.

All in all, we describe our systems in terms of a series of often already-known algorithms that typically only meet some of our goals and will therefore need to be skilfully *composed* together to make systems that do meet our required goals as best they can. This idea of putting various sub-systems together to somehow achieve the overall goal is essentially what programming is all about. This remains true at whatever scale one is working, be it assembly-level programming and hardware, all the way up to Big Data processing systems and beyond.

The overall complexity of all these systems working together as well as possible quickly becomes somewhat daunting – the complexity of systems and what is expected of them is only set to increase. A number of implications follow:

- In order to compose sub-systems together, it is clearly important to understand somehow what each sub-system can provide and also in what way it might fail due to resource limitations and constraints on inputs.
- Availability is an important requirement of most systems – and knowing what time/resources are needed for each operational situation is often critically important for manageability. Designs and systems should behave as predictably as possible.
- Parallel and concurrent sub-systems present all sorts of challenges – memory management, communication bandwidth, data throughput and management, resilience in the presence of failure, etc.

Many readers might now be thinking that this argument is clearly leading towards a call for improved mathematical specification techniques and the like. Far from it – the problem with such ambitious approaches in practice is that software specifications tend to get at least as complicated as the software they are describing, making it very challenging to meaningfully gain practical assurance that given programs meet some required specification – unless of course the specification and program just happen to be mostly structurally *identical* in any case!

Instead, perhaps a better approach would be to exploit exactly this observation above and provide ways to meaningfully construct our programs so that particular system properties of interest become mostly self-evident from the actual structure of the program itself, thereby making separate software specification less of a necessity. Knowing such properties would help ensure that our sub-systems become more *compositional*, and that overall, systems also become more resilient and durable, leading to improved confidence that systems will behave as they are designed and constructed to have.

Of course, specifications still have an important role to play within the realm of requirements and in clarifying what the high-level architecture might look like. Specification would still contain precise statements of important characteristics and invariant properties of the system. It's more that they need to address user-level expectations, perhaps in terms of describing the overall API and what it would provide.

With improvements like these, programming might then be able to become synonymous with the design and engineering of reliable, high-precision systems.

Book Review

Seven Concurrency Models in Seven Weeks: When Threads Unravel

Paul Butcher, Pragmatic Programmers Bookshelf, 2014

Computing and IT are complicated, detailed and intricate, with fast moving developments and new emerging ways of approaching age-old fundamental issues, such as parallel and concurrent processing at scale. Paul Butcher's recent book bravely tackles this area in the popular "Seven in Seven" series format.

I have to say I was surprised to see a title involving "Concurrency Models" in this series and more than a little intrigued to see how such a varied and complicated topic could possibly be brought usefully and pragmatically into focus. But this book largely achieves its ambitious goal of doing exactly that - presenting a broad overview of modern pragmatic approaches to parallelism and concurrency.

A nice point here is that the common confusion between parallelism and concurrency is dealt with straightaway in the first chapter by quoting Rob Pike:

> Concurrency is about dealing with lots of things at once.
> Parallelism is about doing lots of things at once.

This chapter continues to set the scene by outlining what the seven concurrency models are: *Threads and Locks, Functional Programming, The "Clojure*[1]*" Way, Actors, Communicating Sequential Processes (CSP), Data parallelism*, and finally *The Lambda Architecture*. Subsequent chapters then pragmatically deal with each of these models in turn, using appropriately common problems to provide examples, such as the deceptively simple sounding task of computing word-count over Wikipedia in its entirety. Each chapter then contains three example themes that illustrate and exercise the main features of each model, typically

[1] Clojure: A Lisp-like programming language, compiling down to JVM (see: http://clojure.org)

with the aim of pragmatically showing what works – and also what doesn't. As one would expect, each of these models works well at some scales and not so well at others – and this point is well conveyed through the kinds of examples that are tackled.

Apart from general interest in all of these models, what will particularly attract the attention of readers of this newsletter is the inclusion of topics here such as *functional programming*, *actors* and *CSP* that will hopefully confirm what many readers have probably thought were long ready for the commercial prime-time. It seems that today's commercial needs for Big Data processing and real-time analytics really are urgently presenting plenty of technology opportunities for applying ideas like these.

What is shown here is a broad overview of these concurrency models with a surprisingly useful amount of detail, given the length of the book (275 pages, incl. index). As such, it succeeds at cutting through to the essential aspects of each model in a pragmatic and down-to-earth manner via actual examples. Additionally, there is an awareness of the obvious limitation that coverage of any interesting sub-topics must necessarily be brief or even non-existent – but generally, this is covered as one would hope by giving URLs to further discussion/documentation. Overall, a well-written modern account of a difficult and demanding area that will continue being relevant for some time to come.

Reviewed by Brian Monahan

Book Review

Thinking Functionally with Haskell

Richard Bird, Cambridge, 2015

It is very refreshing to see the *essence* of a entire subject covered in a single text – but that is what Richard Bird set out to do, and is largely achieved in his recent book. As he says in the preface, the best thing in his view about functional programming is the ability to think about programs in a mathematical way. I must say that I entirely agree with this point and why I would imagine the book will be potentially of great interest to readers of this newsletter.

The functional language Haskell provides the canvas for expressing the overall message. Haskell should by now be pretty well-known to many readers – it began in the late 80's, with its most recent revision in 2010; the language itself and its compilers (e.g. GHC) have been thoroughly described and exhaustively explained in many online sources, countless blogs, and various other books. Here such material is concisely dealt with via simple examples in the first couple of chapters or so. In particular, the big strengths of Haskell such as its use of (parametric) polymorphism combined with *type classes* (e.g. Eq, Ord, Show, etc.) to provide an elegant form of overloading are all demonstrated early on.

A significant strength here is that little (if anything) is required in the way of previous programming experience to understand what is going on – it is aimed as a textbook for a general undergraduate audience and begins relatively gently. The ideas behind Haskell are introduced gradually in sufficient depth for readers to tackle with confidence the exercises given at the end of each chapter. A further nice touch is that answers are provided as well, making it useful for self-study and giving explicit reinforcement of the points made.

Mostly, the text goes straight to the heart of the matter and tackles the more basic issue of how to think about programs and solving problems effectively using the functional approach. Case study examples in later chapters, such as a Sudoku puzzle solver and an equational calculator (i.e. symbolic interpreter) nicely illustrate various ways to solve problems in the functional style.

A question that does arise with using Haskell is how to introduce its approach to *actions* and *effects* in terms of "imperative functional programming" (i.e. monads). This is wisely discussed late in the book in Chapter 10, in an entire chapter dedicated to this topic. By then, the reader will have been thoroughly exposed to the richness and power of pure functional programming and should have then understood that much can be achieved without any resort to effects. The price of delaying this treatment is that something needs to be said earlier on how to print values generally – and this is paid for with a brief section in Chapter 2 where IO is explained in terms of simple *commands*. Overall, the explanation of the monadic programming style is nicely done and is easily one of the most natural tutorial accounts I have seen.

I can wholeheartedly recommend this textbook as an excellent starting place for exploring functional programming using Haskell. Even for experienced programmers, the book serves as a reminder of how to write clearly and organize material to convey a message, namely: programs can be explained elegantly, simply and directly.

Reviewed by Brian Monahan

FACS Committee

Formal Aspects of Computing Science Specialist Group

Jonathan Bowen
Chairman
ZUG Laison

Jawed Siddiqi
FACS Treasurer

Paul Roca
FACS Secretary

Roger Carsley
Minutes Secretary

John Cooke
BCS Liaison
Publications

John Fitzgerald
FME Liaison

Margaret West
BCS Women Liaison

Rob Hierons
Chair, Formal Methods
And Testing Subgroup

John Derrick
Chair, Refinement
Subgroup

Eerke Boiten
CryptoForma Liaison

Tim Denvir
Co-Editor, FACS
FACTS

Brian Monahan
Co-Editor, FACS
FACTS

FACS is always interested to hear from its members and keen to recruit additional helpers. Presently we have vacancies for officers to help with fund raising, 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 the FACS Chair, Professor Jonathan Bowen at the contact points below:

BCS-FACS

c/o Professor Jonathan Bowen (Chair)
Birmingham City University

Email info@bcs-facs.org.uk

Web 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.