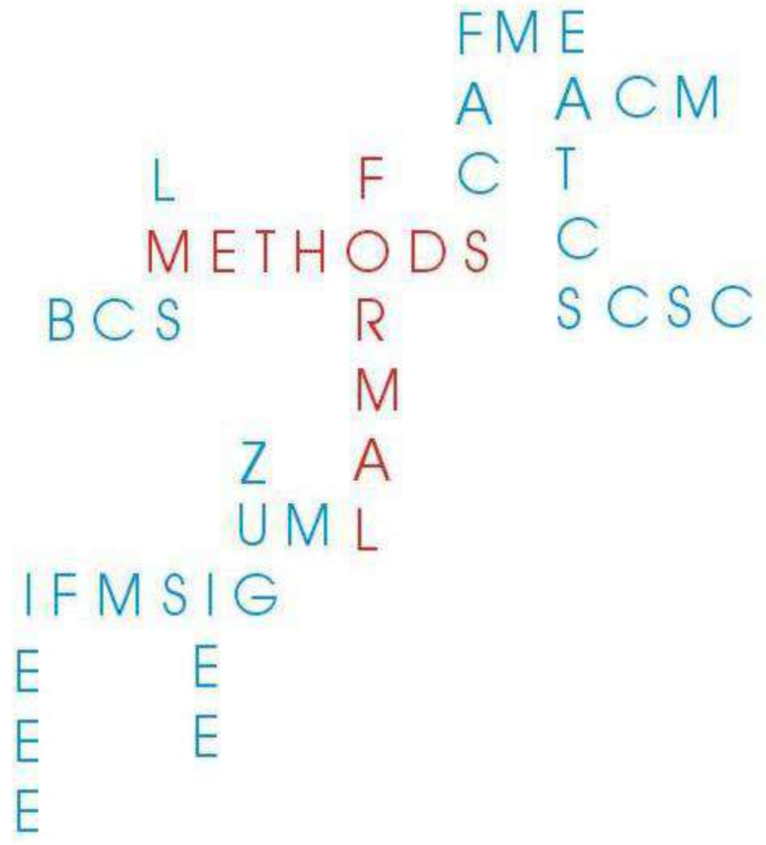
Issue 2019-1
December 2019

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 at:

https://www.bcs.org/membership/member-communities/facs-formal-aspects-of-computing-science-group/newsletters/

Back issues of FACS FACTS are available for download from:

https://www.bcs.org/membership/member-communities/facs-formal-aspects-of-computing-science-group/newsletters/back-issues-of-facs-facts/

The FACS FACTS Team

Newsletter Editors

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

Editorial Team:

Jonathan Bowen, John Cooke, Tim Denvir, Brian Monahan, Margaret West.

Contributors to this issue:

Jonathan Bowen, John Cooke, Tim Denvir, Margaret West,
Troy Astarte, Marie Farrell and Matt Luckcuck

BCS-FACS websites

BCS:	http://www.bcs-facs.org
LinkedIn:	https://www.linkedin.com/groups/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 Jonathan Bowen on jonathan.bowen@lsbu.ac.uk.

Editorial

Dear readers,

Welcome to *FACS FACTS* for 2019[1] - somewhat delayed due to unavoidable circumstances.

This issue begins with a series of reports on FACS events and other events of strong interest to FACS. First is a report on the BCS Lovelace Lecture which was given by Professor Gordon Plotkin on 4th March 2019, titled *Languages for Learning*. This is followed by reports on nine further events including the FACS AGMs in 2018 and 2019, and finally a comprehensive report from Troy Astarte on the 2019 Annual Peter Landin Semantics Seminar given by Professor David Turner titled *Some History of Functional Programming Languages*.

Next there is a feature article on *Regulating Safety and Security in Autonomous Robotic Systems* from Matt Luckcuck and Marie Farrell. That is followed by a Roving Report on the LMS Colloquium on Mathematics of Security from Margaret West and we conclude with an article by Tim Denvir on *The Temptation to Over-design*.

Finally there is a note on Future FACS Events with links and various details, together with views of the new BCS offices where they will take place: at the moment we have only one firm event to announce, but more are planned. The BCS-FACS events website will be updated with the appropriate details as and when these events are confirmed.

Digitisation Update

In March 2019 we took a collection of past FACS newsletters, reports and documents from various workshops, which we only had to hand in hard copy, to be scanned and digitised. This was carried out by Capture All, a firm that specialises in just that process. We now have these past documents available as PDF files held on various websites for safe keeping. BCS are for some reason going to jettison all older documents they hold electronically, and so we are making sure that we have our own records for posterity.

So we now have recorded as pdf files, 21 newsletters from 1990 to 1998, and 12 from 2000 to 2006. Also, there are 17 miscellaneous documents from FORTEST meetings in 2002 and 2003; 26 files, mostly of slides, from FACS Xmas Workshops from 1985-1989; and 4 background papers from a one-day seminar on Martin Löf Type Theory given by Roland Backhouse in 1987.

Tim Denvir, *FACS FACTS* co-editor
Jonathan Bowen, FACS Chair

[1] Published in January, 2020

BCS Lovelace Lecture: Languages for Learning

Professor Gordon Plotkin

University of Edinburgh

March 4th 2019

Reported by: Tim Denvir

The Lovelace Medal is awarded by the BCS each year to prominent researchers for their contributions to computer science. Gordon Plotkin won the 2018 medal and gave this Lovelace Lecture in March 2019. Gordon is particularly well-known for his work on structural operational semantics. It is encouraging to see the BCS awarding the medal to a theoretical computer scientist whose work may be considered as accessible only to a limited audience. A transcript of an interview with the speaker prior to the lecture, and a video of the lecture itself, can be found at https://www.bcs.org/content-hub/the-lovelace-lecture-2019-languages-for-learning/. The video consists of the slides and voice-over, rather than a film of the event, which is useful for anyone wanting to view the technical detail. Since this video gives the whole lecture, with slides in quality reproduction, I shall merely give a summary here.

Professor Philippa Gardner, chair of the BCS committee which awarded the medal to Gordon Plotkin, introduced the event. Philippa was a PhD student of Gordon's, and she was at pains to tell us that she left the room when the final decision was made, to allay any thoughts of bias. I realised that in the audience were many other past and present PhD students of Gordon's, and also members of his family. The latter may have made him more nervous!

Peter O'Hearn gave a more technical introduction to Gordon's work, covering at high speed Plotkin's notions of Parallel Or and Parallel Exists, which resolve the apparent disagreement between Equality in the Standard Model and Leibniz' "equality of indiscernibles", then his Structural Operational Semantics, and then Algebraic Operations and Generic Effects. Peter O'Hearn also cited Gordon Plotkin's nomination for Fellow of the Royal Society, which comprises an impressive list of achievements.

So, after some fifteen minutes of preamble we heard Gordon's lecture itself. It is worth saying that the title, *Languages for Learning*, refers to languages for writing programs which themselves learn and then evolve. These have particular relevance in AI applications. Gordon started by talking about Ada Lovelace's program for calculating Bernoulli numbers. Then in 1840 she had the idea for non-numerical computations. Turing in his 1950 paper *Can Computers Think?* alludes to the latter work of Ada Lovelace and rebuts her view that computers can only do what one instructs them to. Gordon noted that Ada Lovelace's proper title was Ada Augusta, Countess of Lovelace, and mused over why she was always called *Lady* rather than *Countess* Lovelace.

Gordon then talked about statistical programming, and started with a simple example to illustrate Bayes theorem. Thomas Bayes lived from 1702-1761, but the same principles were independently discovered by Richard Price (1723-1791) and Pierre-Louis Laplace (1749-1827) who gallantly acknowledged Bayes' work. Gordon used this example as the basis of a simple probabilistic program. Interestingly, he noted that Wikipedia lists 46 probabilistic programming languages. He then moved on to deep learning, that is, programs built from trainable functions. These use artificial neurons, loosely inspired by the architecture of the brain. In this context Gordon referred to the 1940s work of McCulloch and Pitts[2], a paper I had first come across in the 1960s when exploring the relationship between languages and automata. It rang a venerable bell for me.

Tensors can capture terms of numerous dimensions. Such terms are used for image processing and speech analysis, and are a way of formalising neural networks. Supervised learning can assist image recognition - e.g. "this is an elephant".

So vectors, allied with tensors, can model differentiable programming languages: Jacobians, which are vectors of and generalisations of Grad (∇), let us differentiate programs (I assume because programs are implementations of functions).

Finally Plotkin listed possible other work, some of which was definitely ambitious: non-smooth functions, non-differentiable functions, imprecise computation, what do statistics and differentiable programs have in common... He ended by thanking the audience and acknowledging a list of colleagues, which he did by displaying a screenful of their names and photos.

After the lecture the actual award, the Lovelace Medal, was presented to Gordon Plotkin, and Jane Hillston gave a vote of thanks.

2 McCulloch, W. S. and W. H. Pitts: 1943, 'A Logical Calculus of the Ideas Immanent in Nervous Activity', Bulletin of Mathematical Biophysics 7, 115–133.

The BCS held this Lovelace Lecture at the Royal Society in London and advertised it widely to its members. As a result the lecture room was nearly filled with about 200 people, most of them I imagine not versed in formal or theoretical aspects of computing. Apart from the speaker, I recognised only two other computer scientists there, had a drink with one of them and made the acquaintance of one of the many non-specialists during the pleasant reception afterwards.

FACS Events, 2018–2020

Jonathan Bowen (Chair of BCS-FACS)

Since the last *FACS FACTS* newsletter issued in August 2018, we have held a number of seminars and other events. This article provides an overview and record of these, both past and forthcoming. All FACS events are held at the BCS London Office unless otherwise stated.

Note that since August 2019, the BCS London Office has moved to 25 Copthall Avenue, London EC2R 7BP. The nearest tube station is Moorgate, but Bank and Liverpool Street are within walking distance as well.

Unifying Theories of Refinement

Friday 12 October 2018.

Main speaker: Professor He Jifeng**, East China Normal University, Shanghai, China.**

Other speakers: Professor Tony Hoare, Microsoft Research, Cambridge, UK & Professor Jim Woodcock, University of York, UK.

Chair: Professor Jonathan Bowen, London South Bank University, UK.

Overview:

This event celebrated the 20th anniversary of the publication of the book *Unifying Theories of Programming* by C.A.R. Hoare and He Jifeng in 1998. The main talk by Prof. He Jifeng was given in the presence of Prof. Sir Tony Hoare, who provided some introductory remarks. Prof. Jim Woodcock of the University of York provided a summary of the talk at the end. Note that 2018 was also the 40th anniversary of the FACS Specialist Group itself and the 30th anniversary of the associated Formal Aspects of Computing (FAC) journal, so this event was a triple celebration. The event was chaired by Prof. Jonathan Bowen, Chair of BCS-FACS.

Biography:

Prof. He Jifeng is a Chinese computer scientist. He graduated from the mathematics department of Fudan University in 1965. From 1965 to 1985, he was an instructor at East China Normal University. During 1980–81, he was a visiting scholar at Stanford University and the University of San Francisco in

California, United States. From 1984 to 1998, He Jifeng was a senior research fellow at the Programming Research Group in the Oxford University Computing Laboratory (now the Oxford University Department of Computer Science). He worked extensively on formal aspects of computing science. In particular, he worked with Prof. Sir Tony Hoare, latterly on *Unifying Theories of Programming* (UTP), resulting in a book of that name. Since 1986, He Jifeng has been Professor of Computer Science at East China Normal University in Shanghai. In 1996, he also became Professor of Computer Science at Shanghai Jiao Tong University. In 1998, he became a senior research fellow at the International Institute for Software Technology (UNU-IIST), United Nations University, based in Macau. He moved back to Shanghai in 2005. He Jifeng's research interests include sound methods for the specification of computer systems, communications, application and standards, and techniques for designing and implementing those specifications in software and/or hardware with high reliability. In 2005, he was elected to the Chinese Academy of Sciences. In 2013, his 70th birthday was celebrated at East China Normal University with an international three-day Festschrift in association with the International Conference on Theoretical Aspects of Computing (ICTAC).

He Jifeng and Tony Hoare

Tony Hoare

He Jifeng

Jim Woodcock

FACS networking

FACS @ 40 cakes

Coresets at the Heart of Big Data

Wednesday 17 October 2018.

Speaker: Stephane Chretien, National Physical Laboratory, Teddington, UK.

Abstract:

Big Data is pervasive nowadays due to extensive monitoring in health science and engineering. Data is now considered to be a high value resource, both for knowledge mining and as a commercial asset. Most of the work usually undertaken to extract valuable information from datasets consists of basic statistical procedures: dimension reduction, manifold learning, clustering, regression, etc. However, in the realm of Big Data most of these statistical procedures cannot be performed because of the computational burden associated with the size of the dataset and the loss of statistical significance. One recent approach to statistical analysis of Big Data is based on efficiently extracting relevant sample subsets and then performing the usual procedures on this subsample. Obviously, uniform sampling can lead to severe biases with potentially dramatic consequences. The aim of this talk is to provide an overview of the techniques recently devised for intelligent sub sampling and their connections to beautiful concepts such as modern probability and high dimensional geometry.

Verifying CSP and its Offspring

Thursday 1 November 2018.

Venue: London Mathematical Society, De Morgan House, 57-58 Russell Square, London, WC1B 4HS.

Joint event with the London Mathematical Society.

Speaker: Professor Bill Roscoe, University of Oxford, UK.

Abstract:

I like to give systems semantics with a straightforward, refinement-based verification model. This is true of CSP, Timed CSP, Occam, and new work on CSM (a vehicle for translating model-based languages such as UML). This has paid off with many successful verification projects from the transputer, many military systems, systems for creating correct embedded software, to security analysis. I touch on the expressiveness of CSP, the virtues of refinement and the challenges of scalability and accessibility by non-specialists.

Bill Roscoe introduced by Rob Hierons

Bill Roscoe delivering the 2008 FACS/LMS lecture

FACS – 2018 AGM

Monday 10 December 2018.

Agenda:

1. Apologies (Jonathan Bowen)
2. Minutes of the previous AGM (Roger Carsley/Jonathan Bowen)
3. Chairman's Report (Jonathan Bowen)
4. Subcommittees Reports (Subcommittee chairs)
5. Statement of Accounts (John Cooke)
6. Election of Officers and Committee Members (Jonathan Bowen)
7. Future events (Jonathan Bowen/Sofia Meacham *et al.*)
8. Suggested changes to the AGM
9. Archiving of FACS publications and website (Tim Denvir/Jonathan Bowen)
10. Any other business

The FACS AGM was followed by the Annual Peter Landin Semantics Seminar.

Annual Peter Landin Semantics Seminar: Algebraic Methods for Specification and Formal Development of Software

Monday 10 December 2018.

Speaker: Professor Don Sannella, University of Edinburgh, Scotland, UK.

Background:

Peter Landin (1930–2009) was a pioneer whose ideas underpin modern computing. In the 1950s and 1960s, Landin showed that programs could be defined in terms of mathematical functions, translated into functional expressions in the lambda calculus, and their meaning calculated with an abstract mathematical machine. Compiler writers and designers of modern-day programming languages alike owe much to Landin's pioneering work. Each year, a leading figure in computer science pays tribute to Landin's contribution to computing through a public seminar.

Abstract:

A software module can be modelled as a many-sorted algebra consisting of a collection of sets of data values together with functions over those sets, taking the view that the correctness of input/output behaviour is all that matters. Such a module can be specified by giving properties that the functions are required to satisfy. On this simple basis, an elegant account of formal development of verified software systems from specifications of requirements can be built, which treats modular structure in a compositional way, allowing large systems to be treated by decomposition into smaller components. The fit with systems built using the functional programming paradigm is most straightforward, but the ideas generalise smoothly to other contexts.

Don Sannella delivering the 2018 FACS Peter Landin Semantics Seminar

Ontologies for Data Provenance and Curation

Friday 15 March 2019.

Speaker: Clifford Brown, National Physical Laboratory, Teddington, UK.

Abstract:

The reproducibility issue, even if not a crisis, is still a major problem for science and engineering. Factors not originally considered relevant can be highly relevant. These factors can influence the outputs from a measurement process. In this seminar, we investigate the use of Semantic Web technologies as a basis to enhance the capture of provenance meta-data and data curation. These technologies have the potential to lead to a better understanding of reproducibility and hence resolution of problems it might be causing. We also explore the role played by logic and UML in this work.

Biography: Clifford Brown, originally trained as a physicist, has worked in the IT industry and academia for over 30 years. He is currently working in the Data Science group of the National Physical Laboratory in Teddington working on Data Curation and Provenance for Scientific research.

Composing Protocols

Wednesday 3 April 2019.

Co-sponsor: Formal Methods Europe (FME).

Speaker: Professor Farhad Arbab, CWI and Leiden University, The Netherlands.

Abstract:

Despite significant advances in concurrency theory, constructs and models for programming of concurrent applications have essentially stagnated in the past half-century. In contrast to advances in abstractions and constructs for sequential programming, no truly abstract protocol constructs have evolved to raise the level of concurrent programming. Consequently, programmers today use the same cumbersome, error-prone concurrency constructs of traditional action-centric models of concurrency to express protocols in modern software as they did 50 years ago: processes, threads, locks, semaphores, monitors, rendezvous, etc. Among other disadvantages, the unavailability of high-level protocol constructs in contemporary programming languages hampers full utilisation of the enormous potential offered by massively parallel hardware platforms in real-life applications.

In this talk, we motivate the need for interaction-centric models of concurrency, and present Reo as a premier concrete example of such alternatives. Reo offers a language that treats concurrency protocols as explicit first-class constructs, called connector circuits. More complex protocols in Reo result from composition of simpler, and eventually primitive, protocols. Treating protocols as concrete constructs yields a very expressive formal model of concurrency with highly useful software engineering properties, such as fully-compositional construction and verification, scalability, and verbatim reuse. Specifying the protocol of a concurrent system in Reo produces a connector circuit that mirrors the architecture of that system with high fidelity. Moreover, treating such specifications directly as high-level programs opens up new possibilities for compilation and optimization techniques to generate efficient executable code whose performance can meet or beat that of hand-crafted versions of those same protocols programmed in traditional models of concurrency.

Farhad Arbib delivering the 2019 FACS/FME lecture

Doctoral Symposium

Thursday 6 June 2019 (all-day event).

Speakers: Simon Thompson, BT Research, UK & Philip Davies, University of Bournemouth, UK

Co-organizers: Sofia Meacham and Jonathan Bowen, in cooperation with Marwan Elnaghi (BCS London Central Branch).

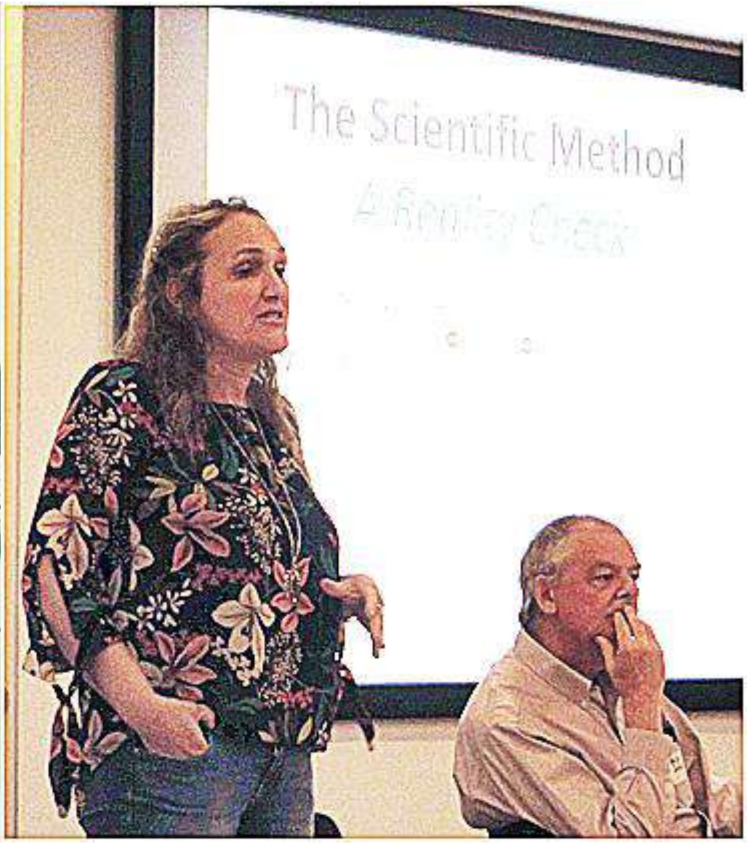
Programme

09:30	Registration and refreshments.
10:00	Dr Marwan Elnaghi, Academic Lead, BCS London Central Branch.
10:15	*Help! Can you use verification to create trust in AI systems?* Dr Simon Thompson, Head of Practice, Big Data and Customer Experience, BT Research, UK.
11:15	Break and refreshments
11:45	*The Scientific Method: A Reality Check.* Dr Philip Davies, Bournemouth University, UK
12:45	Buffet lunch.
14:00	Roundtable discussion of abstracts and research issues
15:00	Plenary discussion about key issues in research and learnings from the roundtable discussions
16:00	Close

Marwan Elnaghi (BCS London Central Branch) and Sofia Meacham

The Doctorial Symposium included two keynote lectures and a poster session for the students.

Simon Thompson



Sofia Meacham and Philip Davies

Poster session at the Doctoral Symposium

Students at the Doctoral Symposium

Help! Can you use verification to create trust in AI Systems?

Dr. Simon Thompson

Presented at the FACS/BCS Doctoral Symposium 6/6/2019

Abstract:

AI people take great delight in using mathematics to create proofs of the properties & correctness of their reasoning systems. However, it's recently dawned on everyone that the functioning of individual AI systems in particular contexts, and the functioning of the systems of systems that are being composed from various AI's as the technology breaks out into the wild, is poorly understood.

At this talk the audience was asked to consider the difference between a modern AI system and a traditional computer program, modern AI consists of an opaque model induced from a (typically) vast data set, and other artefacts, modern AI evolves – or adapts as conditions change – with human intervention or not, and acts autonomously in its environment. And modern AI is deployed in business processes that often contain other AI components with which it interacts.

The "other artefacts" that are typically stirred into modern AI systems include; the training regimes that are part of the art of creating a system, evaluation systems that guide the evolution of models, embeddings that contextualise and transform input data and parameter transfers from other pretrained (or intermediately trained models).

This complexity becomes challenging when we move beyond the artisanal use of AI and it becomes democratized and productionised. As that happens it becomes hard to identify all the models involved in an attack or a failure, or even to differentiate when it is that a model is misused unintentionally.

At BT we are developing approaches to establishing a chain of custody for our AI models that will allow us to meaningfully administer and account for our future AI estate. Huge challenges remain, and we hope that the FACS community will step up to help solve them.

How can we understand and predict the behaviour of AI systems that we can't fully analyse? What kinds of statements about limits of use and trust is it possible to make for systems of systems of this type? How do we evaluate the integrity and safety of these systems and thus help people and society avoid sudden and unexpected harm? Most importantly can we create models that allow us to understand the reaction of our community to the introduction of AI systems? This technology might enable us to avoid future catastrophes and support the adoption of AI as tools to overcome the societal challenges around us now – there is a real need for the FACS community to step forward and help!

When to Trust a Self-Driving Car...

Thursday, 21 November 2019.

Venue: London Mathematical Society, De Morgan House, 57–58 Russell Square, London, WC1B 4HS.

Joint event with the London Mathematical Society In association with the British Computer Society Formal Aspects of Computing Science (BCS-FACS), the LMS hosts an annual evening seminar on aspects of the computer science–mathematics interface. These events are free to anyone who wishes to attend and have attracted high-quality speakers.

Speaker: Professor Marta Kwiatkowska, University of Oxford, UK.

Abstract:

Computing devices support us in almost all everyday tasks, from mobile phones and online banking to wearable and implantable medical devices. We are now experimenting with self-driving cars and robots. Since embedded software at the heart of these devices must behave correctly in presence of uncertainty, probabilistic verification techniques have been developed to guarantee their safety, reliability and resource efficiency. Using illustrative examples, this lecture will give an overview of the role that probabilistic modelling and verification can play in a variety of applications, including security, medical devices, self-driving cars and DNA computing. It will also describe recent developments towards model synthesis, which aims to build these systems so that they are correct by construction. Finally, it will explore the problems of ensuring that systems that rely on learning will behave correctly, both in situations that they have seen in training, and in situations that they haven’t.

Biography:

Marta Kwiatkowska is Professor of Computing Systems and Fellow of Trinity College, University of Oxford. She is known for fundamental contributions to the theory and practice of model checking for probabilistic systems. She led the development of the PRISM model checker (www.prismmodelchecker.org), the leading software tool in the area. Probabilistic model checking has been adopted in diverse fields, including distributed computing, wireless networks, security, robotics, healthcare, systems biology, DNA computing and nanotechnology, with genuine flaws found and corrected in real-world protocols. Kwiatkowska was awarded two ERC Advanced Grants, VERIWARE and FUN2MODEL, and is a co-investigator of the EPSRC Programme Grant on Mobile Autonomy. She was honoured with the Royal Society Milner Award in 2018 and the Lovelace Medal in 2019, and is a Fellow of the Royal Society, ACM and BCS, and Member of Academia Europea.

FACS – 2019 AGM

Thursday 5 December 2019.

Agenda:

1. Apologies
2. Minutes of the previous AGM
3. Chairman's Report
4. Subcommittees Reports
5. Statement of Accounts
6. Election of Officers and Committee Members
7. Future events
8. FACS digitization and archiving
9. Any other business

The AGM was held at 4pm and was then followed by the 2019 Annual Peter Landin Semantics Seminar.

Annual Peter Landin Semantics Seminar: Some History of Functional Programming Languages

Thursday, 5th December 2019.

Speaker: Professor David Turner, University of Kent, UK.

Reported by: Troy Astarte, University of Newcastle

Abstract:

The talk will revisit a series of milestones in the emergence of lazy, higher order, polymorphically typed, purely functional programming languages from their basis in Church's lambda calculus. Starting with a brief review of the lambda K calculus (1941) we will go via LISP (1958), Algol 60, Peter Landin's ISWIM (1966), PAL (1968), SASL (1973), ML (1973), Miranda (1985) and Haskell. On the way we will review the case for lazy evaluation and how this has driven the development of increasingly efficient implementations of normal order reduction and also discuss the search for a fully adequate static type system.

Biography:

David Turner has been researching functional programming languages and their implementation since 1969. He is Emeritus Professor of Computation at the University of Kent where he has spent most of his career. David is best known as the inventor of combinator graph reduction and for designing and implementing a series of purely functional languages – SASL (1972), KRC (1981), and Miranda (1985) – that had a strong influence on the development of the field and the emergence of Haskell. He invented or coinvented some of the ideas which are now standard in functional programming including pattern matching with guards, list comprehensions, and the "list of successes" method for eliminating backtracking.

David Turner delivering the 2019 FACS Peter Landin Semantics Seminar

Report:

In a 2001 talk[3], Peter Landin joked 'I am one of those people who doesn't truly understand something unless he thinks he has invented it.'[4] Coming from anyone else, this might seem egotistical: but Landin really was an exceptional contributor to theoretical computer science—an achievement all the more impressive considering the main part of his research career spanned less than 10 years. Landin is particularly revered in the functional programming world for contributing many extremely important concepts, and explaining them in a lucid and even humorous way.

David Turner, then, had big shoes to fill on a Thursday evening in London. As luck would have it, he has commensurately big feet: one may apply α-conversion to rewrite the last sentence of the previous paragraph with 'Turner' instead of 'Landin' and the sentence remains valid. Consequently, the room in the new BCS headquarters was very nearly filled—with the eclectic mix of people one comes to expect at functional programming events—despite the best efforts of the awkward tablet-based sign-in system at the door. (I had agreed in advance to write this report on the talk, and spent the majority of the hour typing away furiously—my apologies to those around me who might have been a little distracted by the noise!)

Our host for the evening was Jonathan Bowen, whose introduction for Turner helpfully provided some legitimation for the scholar: by way of the Mathematics Genealogy Project's family trees,[5] we were shown that not only was Turner the 'nephew' of Turing, he was also the [great]*-grandson of Poisson, Lagrange, Euler, and Leibniz. Of greater direct relevance to Turner's work was his close connection to both Christopher Strachey, his DPhil supervisor at Oxford, and Dana Scott, who worked closely with Strachey around the time Turner was in Oxford too. Indeed, later in his talk, Turner explained that he had learnt combinatory logic from Scott himself, during Scott's famous[6] Michaelmas 1969 term in Oxford.

As Turner took the floor—with a joke that the two presentation screens made him feel he ought have a co-presenter—he remarked what an honour it was to

[3] Peter J. Landin. Reminiscences. In *Program Verification and Semantics: The Early Work*, June 2001. URL https://vimeo.com/8955127. A seminar held at the Science Museum, London.

[4] A quick note on quotation marks: I will use double marks ("") to indicate quotations of Turner and single marks ('') for other quotations.

[5] https://genealogy.math.ndsu.nodak.edu/id.php?id=75066

[6] If, like me, you have extensively studied the history of denotational semantics.

be speaking under the name of Landin. Turner had had the good fortune to spend a year at Queen Mary University with Landin, as well as having read a great deal of his papers at university. Turner paid particular homage to Landin's 'Next 700' paper[7] and its influence on programming language design.

With this, Turner (metaphorically) rolled up his sleeves and began the talk. He explained that he would sketch the history of functional programming languages of a particular kind: *pure, lazy,* and *non-strict*. The talk would start with lambda calculus, and run through LISP, ISWIM, SASL, KRC, NPL, ML, HOPE, and Miranda, before eventually arriving at Haskell. Turner was true to this schedule, and did indeed make it to Haskell—a language which 2/3 of the audience had used, according to a quick show of hands.[8]

On to the first topic, Alonzo Church's lambda calculus. Turner explained that this simple system—a notation with three kinds of term, and three reduction/conversion rules—was intended to provide a new way to look at the foundations of mathematics, taking the function as the basic object rather than the set. Sadly the same paradox (Russell's) which plagued set-theoretic foundations still applied; but the enduring legacy was the powerful theory of typeless pure functions. After briefly explaining some rules and theorems of the calculus, Turner pointed particularly to the α-conversion rule which says you can apply variables substitution as long as there are no unwanted name captures and bade us remember it for the upcoming discussion of LISP. Another important rule was the second Church-Rosser theorem, which gave a method for reducing a term to its normal form via delaying evaluation of arguments rather than passing them by value.

This led into Turner's next topic: lazy evaluation. This delaying of evaluation was not generally used in 1960s programming languages, which tended to use by-value calling, but in order to properly implement lambda calculus, lazy evaluation was a necessity. Although this could cause efficiency problems, Turner explained that Chris Wadsworth, a fellow Oxford DPhil student, had showed a method of graph reduction that compensated for this[9]. Turner himself then applied a similar notion with his SASL language, which reduced first to Curry's SK combinatory logic, and then used Wadsworth's graph reduction. A further efficiency step was provided much later by Simon Peyton Jones, who

[7] P. J. Landin. The next 700 programming languages. *Communications of the ACM,* 9:157–166, 1966.

[8] I rather shamefully kept my hand down; 'play with Haskell' has been on my long-term todo list for the past four years.

[9] Christopher Peter Wadsworth. Semantics and Pragmatics of the Lambda-Calculus. PhD thesis, Programming Research Group, University of Oxford, September 1971.

used larger combinators taken directly from program text, with his Spineless Tagless G-machine[10] ("Luckily not heapless too," quipped Turner, "Otherwise there would have been nothing left.").

Having introduced the important basic concepts, Turner turned to programming languages, starting with John McCarthy's LISP. This Turner called "The first[11] functional programming language with a large and serious user community."[12] LISP had a list-based structure, with basic operators *car* and *cdr,* computational completeness with conditional expressions and recursion, and a couple of different implementations. Crucially, the S-expression structure allowed the treating of programs as data and vice-versa; this was powerful, but a layer of complication above lambda calculus. Turner described this as "a first-order language made pseudo-higher-order using meta-programming". Furthermore, the implementation had a serious problem, known as the 'funarg problem': it didn't use α-conversion and so evaluations didn't always come out right. McCarthy thought at first there was a bug in the implementation, but then "must have read Church to the end" and realised it was designed wrongly.

Not until Gerald Sussman's SCHEME in 1975 did the proper static binding actually appear, argued Turner, as part of a series of LISP myths. He presented two other myths. First, no "pure" (read: solely applicative) LISP ever existed! LISP had begun with a FORTRAN-like syntax and had assignment and goto before it ever had recursion. Second, LISP was *not* based on lambda calculus. McCarthy's main inspiration was Kleene and although he got the word 'lambda' from Church he had not fully read the theory.[13] Indeed, Turner believes LISP's most important contribution was automatic garbage collection, which allowed algorithms to be written without inclusion of messy allocation concerns.

Now Turner paused the discussion of functional programming languages temporarily to introduce another important idea: static binding. ALGOL 60, unlike LISP, *had* got α-conversion correct, and Turner was impressed with the Report on the language[14], calling it a "masterpiece of precise technical writing"

[10] Jones, Simon L. Peyton. Implementing lazy functional languages on stock hardware: the Spineless Tagless G-machine. *Journal of functional programming* 2(2):127-202, 1992.

[11] Here I, having been taught by historians to make the sign of the cross upon hearing the word 'first', took a deep breath.

[12] Appropriate caveats had been put in place; I exhaled in relief.

[13] McCarthy admitted this in his paper History of LISP. In Richard L. Wexelblat, editor, *History of programming languages*, pages 173–197. Academic Press, 1981.

[14] Peter Naur, editor. Report on the algorithmic language ALGOL 60. *Numerische Mathematik*, 2(1):106–136, 1960.

despite its lack of formal semantics.[15] Editor Peter Naur's description of ALGOL's call-by-name parameter mechanism is equivalent to Church's substitution—so ALGOL 60 actually got closer to lambda calculus than LISP did! However, the mechanism caused trouble for implementers. In Brian Randell and Lawford Russell's book on ALGOL implementation[16], the authors introduced a way to handle higher-order procedure passing with a 'static chain' which jumped down the stack, connecting frames to their lexical placement. Landin then solved the problem that this didn't properly free stack variables by instead storing 'closures' on the heap—a concept that would turn out essential for functional programming.

This had been introduced in Landin's 1960s papers on programming language design, the best known of which, 'The Next 700 Programming Languages' introduced a putative language ISWIM (which stood, in typically dry Landin fashion, for 'If You See What I Mean'). The important idea here was that 'syntactic sugar' let you have 'Church without the lambda'—Landin used **let** and **where** for binding and abstraction instead. Another introduction was the use of indentation to indicate program structure, in place of all the parentheses used in LISP (Turner explained "Strachey said that LISP might as well stand for 'Lots of Irrelevant and Silly Parentheses'"). Landin had worked on a way to handle jumps in a functional context, and introduced the J operator for this purpose. Although it was never implemented directly, MIT's PAL language had all the main ideas of ISWIM, with first-class labels in place of the J operator. Joe Stoy, Strachey's right-hand man at Oxford, brought PAL back from MIT in the 60s and suggested Turner try to implement it efficiently. The attempt failed: the first-class labels were a real problem, because they allowed any otherwise closed context to be re-entered and so nothing could be garbage collected during execution.

Turner left Oxford in 1972 with his doctoral thesis still unwritten, and took up a post at St Andrew's. There he gave a course on programming language theory, inventing for the task an applicative version of ISWIM, which his colleague Tony Davie implemented using LISP. The language was named SASL, for 'St Andrews Static Language' (although Turner commented "It could have been 'St Andrews Sugared Lambda calculus'"). The language was also inspired by PAL, although

[15] Another Turner—Ray—writes that sufficiently precise technical writing frequently ends up displaying the same properties as a terse 'formal' notation (unambiguous, coherent naming, for example). Raymond Turner. The meaning of programming languages. *American Philosophical Association Newsletter on Philosophy and Computers*, 9(1):2–6, 2009.

[16] Brian Randell and Lawford J. Russell. *Algol 60 Implementation*. Academic Press, Inc., 1964.

with a number of changes, crucially multi-level pattern matching. SASL used call-by-value evaluation with dynamic typing and a purely applicative nature. There were no lambdas, with **let** used in their place, but the lambdas were present in flavour. SASL brought a few advantages over LISP (which Turner had initially considered for teaching): it was purely functional, had the correct scoping for free variables, and the multi-level pattern matching helped readability tremendously.

The dynamic typing was worth particular explanation. Turner noted that SASL and languages like it were *not* typeless—they had types, but these were checked at run time. An advantage of this approach was that you could manage lists without caring what type the components were; delaying type evaluation until runtime was the only way to manage this until Robin Milner later showed how to do it with static polymorphism.

SASL began to take off as a language beyond St Andrews, just as Turner also experimented with extensions. One such, list comprehensions, was suggested by John Darlington, and was tried out in the experimental KRC language, which allowed only one-line equations. KRC also had guards (conditional equations) instead of conditional expressions. From this experience Turner learnt the power of lazy lists. SASL became the first multi-site functional programming language which used this lazy evaluation. Turner explained that laziness supported reasoning akin to equations: you could substitute equalities without having to worry about bottom. Another reason for SASL's growing popularity was that you could avoid using exotic structures like co-routines by utilising lazy lists instead—effectively, the partnered co-routine could be treated like data. SASL also had a feature which replaced the backtracking present in PAL, although this particular idea didn't take off until Wadler had given it a name (list of successes) in 1985[17]; Turner explained this showed the power in giving a good name to a concept.

Focus now shifted away from Turner's work to Edinburgh, where important work on typing was taking place during the 1970s. Rod Burstall had introduced algebraic datatypes to ISWIM in his 1969 paper on structural induction;[18] and Darlington, Burstall's student, had developed a language called NPL.[19] This

[17] Philip Wadler. How to replace failure by a list of successes: a method for exception handling, backtracking, and pattern matching in lazy functional languages. *Conference on Functional Programming Languages and Computer Architecture*. Springer, Berlin, Heidelberg, 1985.
[18] R. M. Burstall. Proving properties of programs by structural induction. *Computer Journal*, 12:41–48, 1969. Earlier available as Experimental Programming Report, No. 17, DMIP, Edinburgh, 1968.
[19] No relation to the National Physical Laboratory, or indeed IBM's 'NPL' which became PL/I.

included multi-equation definitions over algebraic types with pattern matching, as well as set comprehension expressions. This latter was the inspiration for Turner's list comprehensions; Turner took the idea, added lazy evaluation, and decided lists were better because they could be easily converted back to sets. NPL saw an evolution of its own, into HOPE, which was higher-order, purely functional, and—a new notion—strongly, polymorphically, typed (it also lost the set expressions). This strong polymorphism had come from Milner's ML, which had been initially developed as a language for LCF[20] to express proof tactics. The robust typing system allowed type sums and type inference—as well as other clever things—while maintaining a strong type checker.

For Turner, these ideas came together with SASL to shape a language called Miranda. Taking what he felt at the time were the best features from these different backgrounds, Miranda nevertheless had one decision Turner rather regrets now: created types were indicated with an ever-increasing number of * symbols. This quickly became cumbersome when more than three types appeared in an expression. Miranda used lexical distinction between functions and variables, and Turner laments that he did not think to apply this to types as well, as Haskell later would. Using guarded equations and **where** declarations required changing the scope rules for **where**: it became the whole right-hand side of equations. Turner explained that he is a minimalist who likes only one way to do each thing, hence the absence of **let**. Miranda compiled to SK combinators and a reduction machine, and its source code was eventually released publicly: "many years later" than it should have been, Turner admitted.

This was Turner's allusion to the fact that Haskell had been created as an open-source alternative to Miranda, which had been proprietary software under Turner's control. Haskell was designed to have semantics very like Miranda's, with a few syntax changes. Guards were switched to the left-hand side of the equations, and lexical case was used to distinguish types (Turner joked "You could now see what was going on without having stars in your eyes"). Haskell incorporated almost all of Miranda, with some new features too: type classes, monadic I/O, and nested modules. Type classes, explained Turner, were very powerful and increased expressiveness—at the cost of additional complexity.

Turner then gave us an example of a short SASL program for identifying tautologies. He had believed this was not achievable in Haskell, but the solution existed using type classes. It seemed that Turner remains a little sceptical about type classes, describing them as "possibly too clever for their own good"

[20] LCF, Logic of Computable Functions, was Milner's attempt to build a proof system from Scott's domain theory.

and likened the Glasgow Haskell Compiler (ghc) to a comment made by Niklaus Wirth on PL/I: 'Like a Swiss Army Knife, it has a blade for everything, but you might cut your hand trying to get it open!"

Looking at type classes with further suspicion, Turner remarked that one of the principles of the Haskell design committee was that the new language should be based on ideas that enjoyed a wide consensus. Type classes must have been the exception, being a new introduction. Turner then explained the idea of 'coherence': a language is coherent if the computational behaviour of the term is independent of its type. For example, two empty lists added together should always result in another empty list, regardless of the types of the originals. Haskell, explained Turner, was *deliberately* not coherent—type classes enabled the changing of a term's behaviour depending on its context. This was illustrated with overloaded constants: a string formed from the numeral '1' has a different length depending on its type. A later paper by Wadler[21] did invent an alternative to overloading which was coherent, and managed this by avoiding the overloading of constants. What, speculated Turner, would the type class system look like without this?

And with that final speculation, Turner's talk was over. We had been taken on a journey through a number of various different programming languages, and the various problems and insights of these had been explored. We now all knew the chains of events leading to the creation of Turner's various major contributions to functional programming, and were left in no doubt as to the significance of them.

Then began the questions. I was lucky enough to get my hand up first, and related an anecdote told by Olivier Danvy[22]: he had been taking lunch with Landin and John Reynolds, and asked them both what they felt the evaluation mechanism was for denotational semantics. Both answered simultaneously, but differently: Landin saying "call-by-value" and Reynolds "call-by-name". What, I asked, did Turner think? He was just as emphatic as Reynolds that "by-name" was appropriate, and was furthermore sure that Strachey would have agreed, having sat through his lectures on denotational semantics—and rather surprised that Landin hadn't. Turner remarked as well that he had been very taken by denotational semantics, and that in some respects SASL was his attempt to turn the denotational meta-language into a programming language.

[21] Martin Odersky, Philip Wadler, and Martin Wehr. A second look at overloading. *FPCA*. Vol. 95. 1995.
[22] Olivier Danvy. Peter J. Landin (1930–2009). *Higher-Order and Symbolic Computation*, 22(2): 191–195, 2009.

The next question was on polymorphic strictness; was Haskell's SEQ present in Miranda? Yes, replied Turner, and furthermore so was Force, a function for evaluating a whole data structure. A follow-up question asked about Miranda's strictness of Fold-L; Turner explained he believes he followed David Bird's suggestion to put SEQ into it. The questioner replied that Haskell has non-strict Fold-L and that makes a difference. Turner's response: "Whatever was the right one, I did that one! But you can check the Miranda source and if it is the wrong one, please let me know."

Jeremy Gibbons asked whether list comprehension was first tried in KRC, which Turner confirmed. He took the idea from Darlington's NPL set expressions and made them lazy and lists. Gibbons followed up, explaining that in SETL, a language developed by Jack Schwartz at NYU, there was a set comprehension construction—in 1969. Had Darlington known about this? Turner suspected probably not; he had plenty of contact with Darlington in the early 1970s and if he had known about it, he likely would have mentioned it to Turner.

Another question asks about the relation between co-recursion and lazy evaluation. Turner confessed he was not an expert on that. If you construct a lazy list and want to ensure it always has a next element, there are rules you can use. Well-founded recursion means you need to show that there is a constant descent; but rules for lists are a harder problem. Jaco de Bakker showed one approach which involved constructing a metric for your data and showing that each co-recurse always decreases the metric. But that might be more applicable to induction.

Gibbons chimes in again, to ask whether Church encodings of datatypes work for co-datatypes. Turner says he believes so, and saw an article on it. A suggestion from the floor is that it might be Burge's book on recursive programming[23], which is discussed as a rather important book. It had parser combinators in it as early as 1975. Turner also notes that Burge, who worked for IBM, wrote the book as a way to explain to his managers what he had been working on for all those years.

The next question asks whether there are practical coherent languages which make sense. Turner replies that Miranda was very nearly coherent, and the only non-coherent things would be easily fixable. ML is as well if imperative features are ignored. Turner reveals he is considering redoing Miranda, and would make sure it was coherent if so. Gibbons asks whether full coherence would prevent

[23] William H. Burge. *Recursive programming techniques*. (1975).

equality and printing—Turner says that one might follow ML's path of using a type to represent equality. Gibbons says the behaviour of *show* is always determined by type—is there a way to implement that coherently? Turner's response is that the key to coherence is different ways of writing types. You can have overloaded functions as long as the first argument is coherent.

A question now wonders whether functional programming could be helpful or relevant in the design of large scale distributed systems, or software/hardware co-design. Turner is enthusiastic about the prospect. A follow-up asks about parallelism, which Turner says in principle is great, although the details always turn out to be hard. He recently examined a PhD on extracting concurrency from SASL, but it's tough. When VLSI came out, it was claimed that chip design was easy. An audience member notes that there is work on at the moment to translate fragments of C code into circuitry for high-performance hardware. Turner says that would be even better if the code was functional, but the audience member rejoins that the idea is to take what people are currently used to using. Turner remarks that Arvind created a company, Bluespec, which uses functional programming ideas for chip design.

Another topic of conversation about hardware is the idea of reversible computing. Essentially the idea is to keep all computation results so that they can be recalled if needed rather than recalculated; this would conserve energy. The key is that entropy is increased when computation is performed as information is lost; if it can be reversed, energy can be conserved. Is there a relation here with the notion of first-class labels? Turner says it could be related, and wonders also about a link to databases where you have systems that involve no throwing away. These systems have become more feasible as the price of storage decreases.

The very final question relates to the release of Miranda's source: what are Turner's hopes now for it? Turner explains that by releasing the code, he signals that he is giving up control! But he would like to try rewriting the compiler in Miranda and releasing that. It's something he'd been meaning to do for years.

With that, time was called, and the talk ended. Many conversations continued in the break room: the event was lively and full of curious characters all united by their deep passion for functional programming. Even for those—like me—who did not have such a strong familiarity with the paradigm, the talk was interesting and engaging; and I'm sure many others were also inspired to explore this fascinating field even more.

Regulating Safety and Security in Autonomous Robotic Systems

Matt Luckcuck and Marie Farrell

Department of Computer Science, University of Liverpool, UK

16th December 2019

Autonomous Robotics Systems are inherently safety-critical and have complex security issues to consider (for example, a security failure can lead to a safety failure). Before they are deployed, these systems often have to show evidence that they adhere to a set of regulator-defined rules for safety and security. Formal methods provide robust approaches to proving a system obeys given rules, but formalising (usually natural language) rules can prove difficult.

Regulations specifically for autonomous systems are still being developed, but the safety rules for a human operator are a good starting point when trying to show that an autonomous system is safe. For applications of autonomous systems like driverless cars and pilotless aircraft, there are clear rules for human operators, which have been formalised and used to prove that an autonomous system obeys some or all of these rules [3,4]. However, in the space and nuclear sectors applications are more likely to differ, so a set of general safety principles has been developed. This allows novel applications to be assessed for their safety, but are difficult to formalise.

Cyber security for autonomous robotic systems is a challenge that is frequently overlooked. However, ensuring that an autonomous robotic system is resilient against cyber attacks, particularly since the attacker may attempt to commandeer the robotic system and use it to cause harm. In order to begin to give guidance on implementing security measures for these systems, there must be a discussion of the current state-of-the-art with the relevant stakeholders and the development of security principles for the particular application domain.

To improve this situation, we are collaborating with regulators and the community in the space and nuclear sectors to develop guidelines for autonomous and robotic systems that are amenable to robust (formal) verification. These activities also have the benefit of bridging the gaps in knowledge within both the space or nuclear communities and academia.

1 Security Regulations in Space

On the 1st of February 2019, we held a Space Security Scoping Workshop which was jointly organised by the Universities of Liverpool (Marie Farrell and Michael Fisher) and Warwick (Matthew Bradbury and Carsten Maple). The 29 attendees were from a mix of academia and industry. The aim of this workshop was to discuss the cyber security issues related to robotic systems deployed in space in order to scope out research priorities and to develop collaborative R&D programmes on the topic of cyber security between FAIR-SPACE and industrial partners.

Discussions during the workshop revealed that the space industry is becoming more entrepreneurial, with a greater acceptance of risk for more financial gain. Current regulations and standards for space are lacking and often ignored. In particular, rules enforced by the European Space Agency (ESA) when launching satellites can be, and often are, disregarded by other organisations, and so those that do not meet the ESA's requirements may still be launched.

During this workshop, the organisers posed the following five questions to the attendees related to space security.

- What are the security issues in space?
- Are they different to the issues in autonomous ground/air vehicles?
- What will be the problems in the future?
- What are current ways of detecting/stopping attacks in these systems?
- How do environmental considerations impact on security?

From the resulting discussion, it became clear that, although some companies and organisations have a good understanding of the security issues faced by their space systems, there are others that have not seriously considered cyber security. The lack of detailed guidelines and regulations in this area is certainly a stumbling block which makes it difficult for new space companies to know exactly what is required of their systems from a cyber security perspective. There are, however, various guidelines for space systems' security published by the Consultative Committee for Space Data Systems (CCSDS), but it appears that they are no enforced and are not detailed enough, particularly for autonomous robotic missions [1].

A full report for this workshop, which describes the discussion of the above questions in detail, has been published by the FAIR-SPACE hub [2]. The report also outlines our future work in this area, which includes several academic publications describing how cyber security threat analysis techniques can be combined with formal verification and some associated case studies. We are in frequent contact with several attendees from the workshop to ensure that our research remains relevant for the space industry. Furthermore, we intend to organise a follow-up workshop in the future, as part of the FAIR-SPACE Hub[1].

[1]Future AI and Robotics for Space: `https://www.fairspacehub.org/`

2 Safety Regulations in the Nuclear Sector

In the UK, regulation of robotics for nuclear industry is more clear cut. The Office for Nuclear Regulation (ONR) is the government body responsible for checking the safety of any system operating on the 'nuclear estate'. While the ONR provide guidance for ensuring system safety, they have not yet produced any guidance specific to autonomous systems. Their guides are also more descriptive than prescriptive, which makes them difficult to formalise and use as a system specification.

To tackle this challenge, we have been running a series of workshops[2] with the ONR. The workshops aim to be an open forum for discussion between the nuclear operators and supply chain, the ONR, and academia. They explore the safety assessment process for robotic systems in the nuclear industry and examine what may change with the introduction of autonomy. The main focus of the workshops is to clarify (if not answer) the questions surrounding the verification of autonomous robotics.

We have run two workshops, both attended by a mixture of academics, nuclear operators, robotics developers, and representatives from the ONR. The first workshop introduced the safety assessment process in the UK nuclear industry and verification approaches for autonomous robotic systems, and concluded with a broad discussion session. The second workshop focussed on four case studies of proposed or operational robotic systems from nuclear operators. Two of the case studies were of laser cutting systems, the other two were for remote handling or maintenance. After their introduction, each case study was discussed, in parallel, to examine the hazards and mitigations of the current system and of the same robotic system if it were under autonomous control.

Several issues were raised during the workshops' discussion sessions about the extra considerations needed for the introduction of a robotic system. Firstly, it was thought that a robotic system could widen the environment of the system, to cover the transportation and cleaning or maintenance of the robot, and not just its usual operating facility. There was also recognition that the system failing can bring a human back into the hazardous environment. This points to the need for highly reliable (hardware *and* software) systems, especially if the system is autonomous and especially if it will be used over a long time period. Finally, there were worries about the impact of robotic and autonomous systems on the workforce, both in terms of job availability and lower safety due to complacency. Both of these issues require careful cross-disciplinary study and communication.

Crucially for the verification of autonomous robotics, the discussions in both workshops revealed that there is no standard good practice for developing robotic or autonomous systems in a way that can be robustly verified. So, to follow up these workshops, we are collaborating with the ONR to develop such guidelines for autonomous robotic systems in hazardous nuclear environments. We are currently drafting the guidelines, which will eventually be opened up to

[2]Details of the workshops are available at: `https://autonomy-and-verification-uol.github.io/events/fnrc`

wider consultation. These aim to guide developers to build their systems in ways that make robust (particularly formal) verification easier. These guidelines have the secondary aim of describing the merits and methods of forms of verification that may be new that community.

References

[1] Consultative Committee for Space Data Systems: Security Threats Against Space Missions (CCSDS 350.1-G-1). Tech. rep., Consultative Committee for Space Data Systems, Washington, DC, USA (2006), https://public.ccsds.org/Pubs/350x1g2.pdf

[2] Farrell, M., Bradbury, M., Fisher, M., Maple, C.: Workshop Report: Space Security Scoping. Tech. rep., FAIR-SPACE Hub (2019), https://www.fairspacehub.org/s/WORKSHOP-REPORT_SPACE-SECURITY-SCOPING-February-2019.pdf

[3] Rizaldi, A., Keinholz, J., Huber, M., Feldle, J., Immler, F., Althoff, M., Hilgendorf, E., Nipkow, T.: Formalising and Monitoring Traffic Rules for Autonomous Vehicles in Isabelle/HOL. In: Integr. Form. Methods. LNCS, vol. 10510, pp. 50–66. Springer (2017). doi:10.1007/978-3-319-66845-1_4, http://link.springer.com/10.1007/978-3-319-66845-1_4

[4] Webster, M., Cameron, N., Fisher, M., Jump, M.: Generating certification evidence for autonomous unmanned aircraft using model checking and simulation. J. Aerosp. Inf. Syst. **11**(5), 1–31 (may 2014). doi:10.2514/1.I010096, http://arc.aiaa.org/doi/10.2514/1.I010096

Roving Report

The LMS Colloquium on Mathematics of Security

Wednesday, 13th November 2019

Venue: London Mathematical Society, De Morgan House, 57-58 Russell Square, London, WC1B 4HS

(Briefly) reported by: Margaret West

The four speakers were:

Delaram Kahrobaei (York)
Interactions between Group Theory, Cyber Security, Artificial Intelligence, and Quantum Computation

Christophe Petit (Birmingham)
Rubik's for cryptographers: Babai's conjecture, hash functions and quantum gates

David Galindo (Birmingham)
Security Models and Designs from E-Voting to Blockchain

Alexei Lisitsa (Liverpool)
Formal Modelling of Smart Contracts Languages, Their Expressive Power and Verification

The talks were mainly concerned with the mathematical aspects of Post-Quantum cryptography, in a future world in which Quantum Computers are widely available. Quantum cryptography is the exploitation of quantum mechanical properties in the furtherance of data encryption. In contrast, Post-Quantum cryptography involves the development of Cryptographic algorithms which are secure against an attack by both a classic and a Quantum Computer via a secure quantum cryptographic channel. Quantum Computers themselves were only briefly presented by one of the speakers as an additional research area for his research group.

I was pleasantly surprised to find that the four speakers at this “all day” event presented not only on technical mathematics (primarily Group Theory, of course) but also on the societal aspects of their work. Examples of the latter included:

- Governments creating barriers to voter registration for certain sections of the population.

- Difficulties for disabled people were also mentioned in voting access and also e-banking.

Interestingly, Alexei Lisitsa's talk involved "smart contracts", one of which was the intriguingly named "Trump-Obama contract", in which Trump produces his Income tax returns when Obama produces his Birth Certificate!

All in all this was a day both interesting and technically challenging with much discussion. I hope to write this up in greater detail for a future issue of FACS FACTS.

See also
https://www.lms.ac.uk/events/lectures/lms-computer-science-colloquium

The Temptation to Over-design

Musings on Portability, Resilience and Proof

Tim Denvir

Abstract:

Over-design can be dangerous. A product that is over-designed meets its specification but has extra features or properties. In this piece I give two contrasting examples of over-design and show how they hamper portability and confuse diagnostics. Finally, I argue that constructing designs while guided by formal proofs will avoid these pitfalls: proofs aid not only correctness but portability and diagnostics too.

I met the first example of over-design when working in electronics. I was spending a gap-year before university. Semiconductors were in their infancy: they were the subject of research and development. Thermionic vacuum tubes, or valves as they were known in Britain, were used in electronics, typically for amplifying and switching analogue signals. But the earlier computers, such as the London, Cambridge and Manchester Atlas machines, also used them to process digital signals. Some readers may be too young to remember these vacuum tubes, so I shall briefly describe them.

A glass bulb, similar to a light bulb, usually with a metal base, was exhausted of any gaseous content: a vacuum persisted inside it. This allowed free electrons to travel under the influence of an electric charge without hindrance. Hence the term *Vacuum Tube*. Insulated terminals, usually in the base, connected with electrodes in the interior. The simplest vacuum tube was a diode. This, as its name suggests, had two electrodes: an anode and a cathode. The cathode would be negatively charged. A heater connected to two further terminals would be positioned close under the cathode to warm it up. This caused agitated electrons to escape from the surface of the cathode. If the anode was positively charged, it would collect these free-flowing electrons and a current of them would flow from the cathode to the anode. But because the anode was not heated, few electrons would escape from it and almost no current could flow in the opposite direction. Thus, if an alternating voltage was applied across the cathode and the anode, the diode would act as a rectifier, that is it allowed the current to flow only in one direction and converted an alternating voltage into a direct voltage and current.

A third electrode, a grid in the form of a mesh, could be inserted between the cathode and the anode. A voltage applied to the grid could control the flow of current. The resulting device was called a triode. If the static voltage on the anode was considerably higher than that on the grid, an oscillating voltage applied to the grid caused an oscillating current between cathode and anode.

This translated to an amplified oscillating voltage on the anode: one had the means of amplifying signals.

Further sophistications, in the shape of two extra grids, were able to stabilise and make the performance more linear, resulting in the pentode, a vacuum tube with five electrodes. These became the most prominent of all.

Mullard was generally considered to be the best manufacturer of valves/vacuum tubes. The different types of valves had specifications: limits to the voltages you could apply to them, degree of amplification etc. The performance of Mullard valves would usually exceed their specifications: if you subjected these products to a temporary stress beyond the specified limit, they would usually survive. They were, however, rather expensive. But they were a favourite among the professional manufacturers of quality equipment.

Another manufacturer of vacuum tubes was Brimar. Their products reliably met their specifications, but were less resilient to abuse. Exceeding their specified voltages could easily result in permanent failure, and the user circuits could not rely on performance beyond that which was defined. They were, however, much cheaper than the Mullard equivalents and were a favourite amongst amateur electronic designers, builders of home-made radios, hi-fi etc.

The trouble with the supposedly high quality Mullard tubes was that if you constructed a circuit which worked with those components, then a duplicate circuit constructed with, say, Brimar components could fail; this despite the supposedly lower quality components meeting the official specifications. In other words, this over-design on the part of the high quality components could hinder the portability of user circuits.

The second time I met this danger was when building an Algol 60 compiler. The Algol 60 Report[24] does not allow an integer actual parameter to be supplied to a procedure or function whose corresponding formal parameter expects a real number. My first design did allow this, coercing the type from integer to real when the procedure/function body was executed. I felt that this was what the user would expect, and that users were "getting more compiler for their money"! A more experienced colleague persuaded me that I should stick strictly to the language definition. If I had gone ahead, while my compiler would have accurately translated and run correct programs, it could hinder the portability of programs to other compiler implementations. Simply, a program which was incorrect in this very particular manner would run on my compiler but not on others even though those compilers were correct.

24 J. W. Backus et al., Revised Report on the Algorithmic Language Algol 60, Comm. ACM Vol.6 Issue 1 1963, also in Numerische Mathematik and the Journal of the BCS.

Are there two possible kinds of over-design? The vacuum tubes were a case of over-designed components. The Algol 60 compiler was a case of an over-designed environment. Can the example of an over-designed component occur in a software context? Yes, I think so. Suppose a program makes use of a standard library of procedures, or in an O-O context, of O-O methods. The user/programmer could discover (perhaps by experiment) unspecified features of a procedure or method, and exploit them. Then when the program is reassembled with a different library – which may nonetheless be correct according to its specification – the program will fail or behave differently. The correct library component could then falsely appear to be at fault.

Of course, if the programs are designed to be provably correct with respect to a formal language definition, and with reference to formal specifications of the components it uses, this hazard would not arise. In other words, proofs aid not only correctness but also portability and diagnostics, something which may not immediately spring to mind.

To paraphrase from the law-courts, "implement the specification, the whole specification and *nothing but* the specification".

Future FACS Events

Information on FACS events can be found on the BCS-FACS website, which can be found under www.bcs-facs.org. We welcome ideas for further FACS events. Please contact the FACS Chair, Jonathan Bowen, on jonathan.bowen@lsbu.ac.uk and the FACS Seminar Organizer, Sofia Meacham, on smeacham@bournemouth.ac.uk.

Jonathan Bowen outside the previous BCS London Office in Southampton Street

Please do remember to come to the new BCS London Office at 25 Copthall Avenue, London EC2R 7BP, for future BCS-FACS events. The nearest tube station is Moorgate, with Bank and Liverpool Street stations within walking distance too. The following are photographs of the new common area and the main meeting room. We look forward to seeing you at future meetings.

Common area at the new BCS London Office

Main meeting room at the new BCS London Office

FORTHCOMING EVENT

Privacy Assurance in Ubiquitous Systems by Typing in a Calculus of Context-aware Ambients

Thursday 27 February 2020.

Speaker: Dr Francois Siewe, De Montfort University, Leicester, UK.

Abstract:

In the early 1990s, Mark Weiser introduced ubiquitous computing (Ubicom) as a new paradigm for the next generation of distributed systems where computers disappear in the background of the user's everyday activities, making data and services readily available anytime and anywhere. From the era of one-computer-many-users (mainframes) to that of one-computer-one-user (PCs), Ubicom envisions an era of many-computers-one-user. Current realisation of this vision is the Internet of Things (IoT) which enables common objects to be enhanced with sensing, computing and communication capabilities to become smart things capable of collecting, processing and exchanging data over the Internet. How data are collected, processed, and shared, must be tightly controlled in many applications (e.g., smart homes and healthcare) to avoid unintended breaches of privacy. In this talk, Dr Francois Siewe will present the Calculus of Context-aware Ambients (CCA) used to model the behaviours of Ubicom systems, and its privacy type system that allows for the control of information flow among subsystems.

Timing and booking:

Refreshments will be available from 5.15pm. The talk will start at 6pm. Free online registration at:

https://www.eventbrite.co.uk/e/privacy-assurance-in-ubiquitous-systems-by-typing-in-a-calculus-of-context-aware-ambients-facs-registration-71624682353

Biography:

Francois Siewe received a Ph.D. degree in Computer Science from De Montfort University, UK. He obtained a B.Sc. degree in Mathematics and Computer Science, the M.Sc. degree and the Doctorat de Troisième Cycle degree in Computer Science from the University of Yaounde I, Cameroon. He is a Reader in Computer Science and Head of the Software Technology Research Laboratory (STRL) research group in the School of Computer Science and Informatics at De Montfort University (DMU), UK. Before joining DMU, he was a lecturer and visiting researcher in the Institute of Technology of Lens at the University of Artois in France. Prior to this, he was a fellow at the United Nation University/International Institute for Software Technology (UNU/IIST) in Macau in China, and a lecturer with the Department of Mathematics and Computer Science at the University of Dschang, in Cameroon. His research interests include software engineering, formal methods, cyber security, context-aware and pervasive computing, and Internet of Things (IoT). His research outputs can be found at http://www.cse.dmu.ac.uk/~fsiewe/.

Forthcoming events

Events Venue (unless otherwise specified):

NEW *BCS, The Chartered Institute for IT*
Ground Floor, 25 Copthall Avenue, London, EC2R 7BP

The nearest tube station is Moorgate, but Bank and Liverpool Street are within walking distance as well.

27th February	**Privacy Assurance in Ubiquitous Systems by Typing in a Calculus of Context-aware Ambients** Dr Francois Siewe, Reader in Computer Science, De Montfort University

Details of all forthcoming events can be found online here:

https://www.bcs.org/membership/member-communities/facs-formal-aspects-of-computing-science-group/

Please revisit this site for updates as and when further events are confirmed.

FACS Committee

BCS The Chartered Institute for IT — Formal Aspects of Computing Science Specialist Group

Jonathan Bowen
FACS Chair; BCS Liaison

John Cooke
FACS Treasurer and Publications

Roger Carsley
Minutes Secretary

Ana Cavalcanti
FME Liaison

Brijesh Dongol
Refinement Workshop Liaison

Rob Hierons
LMS Liaison

Keith Lines
Government and Standards Liaison

Sofia Meacham
Seminar Organiser

Margaret West
Inclusion Officer and BCS Women 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)
London South Bank University
Email: jonathan.bowen@lsbu.ac.uk
Web: www.bcs-facs.org

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

As well as the official BCS-FACS Specialist Group mailing list run by the BCS for FACS members, there are also two wider mailing lists on the Formal Aspects of Computer Science run by JISCmail. The main list <facs@jiscmail.ac.uk> can be used for relevant messages by any subscribers. An archive of messages is accessible under http://www.jiscmail.ac.uk/lists/facs.html, including facilities for subscribing and unsubscribing. The additional <facs-event@jiscmail.ac.uk> list is specifically for announcements of relevant events. Similarly, an archive of announcements is accessible under http://www.jiscmail.ac.uk/lists/facs-events.html with subscribe/unsubscribe options. BCS-FACS announcements are normally sent to these lists as appropriate, as well as the official BCS-FACS mailing list, to which BCS members can subscribe by officially joining FACS after logging onto the BCS website.