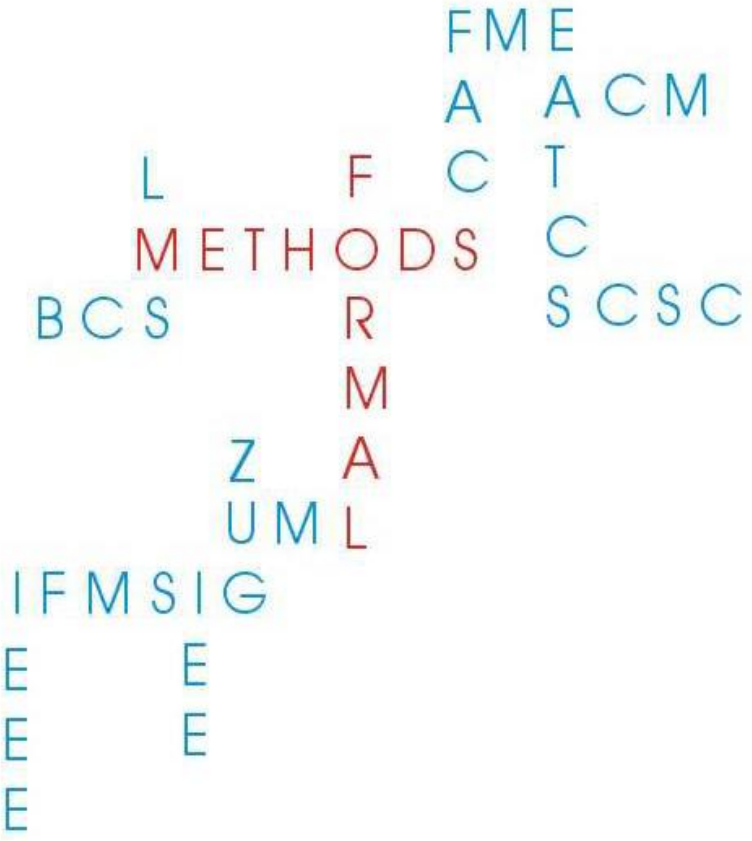
Issue 2016-2
December 2016

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

Troy Astarte, Jonathan Bowen, Muffy Calder, Tim Denvir, Michael Fisher, Sofia Meacham, Greg Michaelson, Simon Thompson, 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@gmail.com.

Editorial

Welcome to FACS FACTS issue 2016-2 and, as I am writing this at the very end of 2016, a happy 2017 to all our readers.

Without any specific intention to orchestrate it on our part, there is something of an Alan Turing theme running through this issue of the newsletter. A rather unusual letter to the editor claims to recall a conversation that the correspondent's great-uncle had with Turing in 1936, in which the latter recounts a theory of the harmonica, remarkably reminiscent of Turing's machine. Another article by one of your editors suggests that what we have left, after eliminating Turing's computable numbers and the rest of the numbers we can define, are akin to dark matter in the universe of the reals. Thirdly on the Turing theme, we have an announcement of a book, *The Turing Guide*, co-authored by Jonathan Bowen, chair of FACS, with Jack Copeland, Robin Wilson and Mark Sprevak. A detailed and positive review of this book has appeared in *New Scientist*, see https://www.newscientist.com/article/mg23331072-700-the-turing-guide-last-words-on-an-enigmatic-codebreaker/, describing it as "pretty much the last word on the subject".

We start, however, with the Chair's Report on the past year, presented at the FACS AGM on 12th December 2016. Then a number of reports on FACS and other events held throughout the year: Troy Astarte reports on a talk given by Joe Stoy, *Christopher Strachey, Pioneer of FACS*. Margaret West reports on Ada Lovelace's 200th Birthday Celebration at Oxford, December 9th - 10th 2015. Margaret West also reports on a series of talks given by Dana Scott around the UK, with a more detailed summary of his delivery of the Löb Lecture at Leeds on 18th May 2016, *Why Mathematical Proof?*.

Jonathan Bowen reports on the joint FACS-LMS seminar given by Muffy Calder on *Probabilistic formal analysis of software usage styles in the wild*, held at the London Mathematical Society, de Morgan House, London. While Muffy Calder is well known in the field of computer science, possibly less well known is that she has recently finished a spell as Chief Scientific Advisor to the Scottish Government. It is heartening to see a computer scientist in such a rôle.

Sofia Meacham and Jonathan Bowen report on the annual BCS-FACS Peter Landin Semantics seminar, on *Building Trustworthy Refactoring Tools*, given by Professor Simon Thompson of the University of Kent. Jonathan Bowen reports on the Strachey Centenary conference in Oxford, 18-19 November 2016, with photos. Michael Fisher from the University of Liverpool announces a network on verification and validation of autonomous systems.

Events that we plan to hold during 2017 include:

- A joint FME/BCS-FACS seminar by Prof. Dr. Reiner Hähnle, TU Darmstadt, Germany, who will speak on "The KeY Formal Verification Tool" on 4th May at the BCS, London. See http://www.bcs.org/content/ConWebDoc/57115.
- The regular Refinement Workshop in June
- The annual joint seminar with the LMS in November
- The FACS AGM and annual Peter Landin Semantics Seminar in December.

Also, arrangements for evening seminars are already well in hand for February, March, April, September and October.

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 the BCS for making these available to us. We look forward to seeing you there!

Tim Denvir

BCS-FACS 2016 AGM

Chair's Report

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

Monday, 12th December 2016

Prof. Jonathan P. Bowen
London South Bank University

First let me thank the FACS treasurer Prof. Jawed Siddiqi and the FACS secretary Paul Boca, for acting at the group's executive officers during 2016, as well as the rest of the FACS committee. Most business during the year is undertaken using email and the AGM is an opportunity to discuss future plans with the FACS committee and others interested in FACS activities.

During 9-10 March 2015, last year, we held a major two-day international ProCoS Workshop on *Provably Correct Systems* at the BCS London office with sponsorship from LERO – the Irish Software Research Centre. This was well attended by delegates and speakers from around the world, including Prof. Sir Tony Hoare, Prof. Dines Bjørner (from Denmark), and others who were members of or influenced by the ESPRIT ProCoS projects of the early 1990s, around 25 years ago. The event has resulted in a post-proceedings "*Provably Correct Systems*" (ISBN 978-3319486277), to be published in the Springer NASA Monographs in Systems and Software Engineering series in early 2017, edited by Prof. Mike Hinchey (University of Limerick, Ireland, and FACS committee member), me, and Prof. Dr Ernst-Rüdiger Olderog (University of Oldenburg, Germany).

FACS has held a number of its traditional evening seminars during 2016. On 17 May 2016, Jan Tretmans, Senior Research Fellow of TNO – Embedded Systems Innovation, Eindhoven, and of Radboud University, Nijmegen, in The

Netherlands, spoke on "Model-Based Testing: There is Nothing More Practical than a Good Theory". Thank you to FACS committee member Prof. Rob Hierons for suggesting the speaker and chairing this event. On 29 September 2016, Prof. Ana Cavalcanti, of the University of York and the new chair of FME, asked "Can robots ever be safe?", considering the software engineering involved with robots.

On 3 November 2016, the annual joint event with the London Mathematical Society (LMS) at De Morgan House in central London was organized again by FACS committee member and LMS liaison officer John Cooke. Prof. Muffy Calder of the University of Glasgow spoke on "Probabilistic formal analysis of software usage styles in the wild". I have written a separate report for the *FACS FACTS* newsletter, with corrections and improvements by Muffy!

Most recently, on 15 November 2016, Joe Stoy of Bluespec Inc., USA, formerly at the Oxford University Computing Laboratory's Programming Research Group, gave a delightfully reminiscent talk on his Oxford colleague "Christopher Strachey – Pioneer of FACS", who was also a colleague of Alan Turing, on the day before the centenary of Strachey's birthday. Troy Astarte, working with Prof. Cliff Jones at Newcastle University and investigating the Strachey Archive in the Bodleian Library at Oxford, has provided a beautifully written and very apt report of the talk for the *FACS FACTS* newsletter.

I give a special thank you to FACS secretary Paul Boca for yet again organizing the Annual Peter Landin Semantics Seminar later today. This is to be delivered by Prof. Simon Thompson of the University of Kent on "Building Trustworthy Refactoring Tools".

BCS-FACS depends on members proposing events, especially evening seminars. Currently 2017 is relatively wide open for possible FACS events and I would encourage FACS members to make suggestions and offer help in organizing meetings. We are entirely dependent on members volunteering in this regard, although there is good support from the BCS with an effectively free venue at the very centrally located BCS London office for FACS meetings. Meetings elsewhere in the United Kingdom can also be supported f there is local interest in supporting such events, perhaps in association with a BCS

Branch group for example. I as chair can also offer support and advice in organizing a meeting if you have not done one before. It is a good learning experience and you get a free dinner with the speaker for your efforts and travel expenses if you chair the meeting as well. We try to have a maximum of one meeting per month (January to June and September to October, since we normally have the joint LMS event in November organized by John Cooke and the Landin Seminar in December organized by Paul Boca). I look forward to hearing your ideas and suggestions, especially if you can volunteer to organize or even give an evening seminar in 2017.

We do have a FACS evening seminar planned for 4 May 2017, in association with Formal Methods Europe (FME), to be delivered by Prof. Dr Reiner Hähnle of TU Darmstadt, Germany, on "The KeY Formal Verification Tool". FME will sponsor the air travel and will also hold a board meeting and their AGM at the BCS London office before the talk. Thanks go to FACS committee member Prof. John Fitzgerald for being the FME liaison officer as Chair of FME over the years. Recently, Prof. Ana Cavalcanti has become Chair of FME and has agreed to replace John as the FME liaison officer on the FACS committee. We thank John for his sterling efforts for FACS, FME, and formal methods in general for many years. Through Ana, we aim to continue the long association of FACS and FME.

I would also like to thank FACS committee members Tim Denvir and Brian Monahan for their work on co-editing the *FACS FACTS* newsletter. I know from experience what a mammoth effort this is for little or no reward, but it is very worthwhile to have it as a continuing record of FACS activities and interests. Volunteers to write reports on talks, trip reports, book reviews, short technical submissions, or anything of potential interest to FACS members are greatly appreciated at any time. Submissions of photographs (with captions, humorous or otherwise) are also encouraged.

I hope you enjoy the rest of the day. Happy Christmas to you all and I look forward to seeing you again in 2017, hopefully at a FACS event.

Jonathan Bowen
Chair, BCS-FACS

LETTER TO THE EDITOR

Dear Editor

As an academic Computer Scientist, I occasionally receive missives from members of the general public claiming to have cracked seemingly imponderable problems, for example how to achieve hyper-computability. In such cases, I feel it is my civic duty to offer them some gentle but firm refutation. Very rarely, however, I receive suggestions with which I can find no significant flaw, despite their seeming eccentricity.

Thus, as I approach retirement, I feel honour bound to bring the following correspondence to wider attention:

Dear Sir

Having received short shrift from numerous IT historians, and browsed your fascinating book on computability, I would appreciate your thoughts on the enclosed fragment of my late Great Uncle's memoir. If I understand it correctly, it appears to shed new light on the early contributions that Allan[sic] *Turing made to Computer Studies.*

Thank you for your consideration.

Yours etc …

… Dining at High Table was largely a bore. Dons are such self centred creatures, with little practical understanding of how the world works.

A notable exception was the mathematician Dr Alan Turing, whom I met on my last visit to Cambridge in November 1936. As I recall, Dr Turing, who was seated

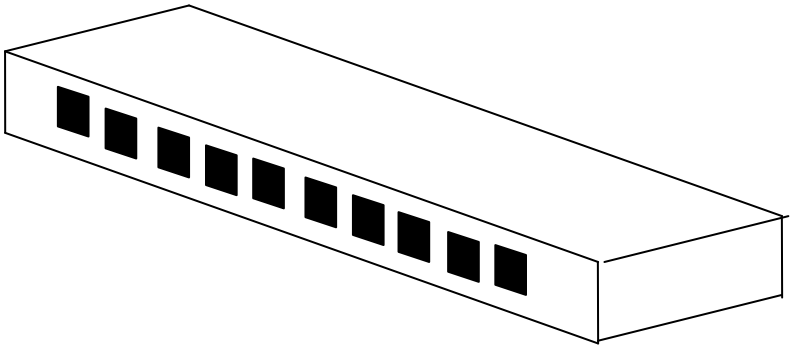
immediately to my right, was silent for much of the meal. We were finishing the main course when some Socialist minded fellow, or should I say Fellow, began to maunder on about the Jarrow Marches and how they were led by a harmonica ensemble playing popular tunes. Foolishly, I remarked on what a poor substitute a harmonica ensemble was for, say, a works brass band.

At this, Dr Turing became quite animated. He opined that, on the contrary, the harmonica was an eminently sensible choice for men walking any distance, and that, in any case, it was fascinating instrument in its own right. He had been tormented by harmonica players in the dorm at his prep school, and had sought relief by analysing its many curious aspects.

Recalling a diabolical craze for the kazoo at my crammer, I expressed sympathy and asked Dr Turing to expostulate further.

Dr Turing told me that harmonicas all share the same basic characteristics. The reeds are laid out beneath a row of holes, such that one hole is above the reeds for two notes. One's mouth is positioned over some hole, and one's tongue, or lips, are shaped to isolate it from its neighbours. Then, one note is played by sucking and the other by blowing.

Dr Turing next took off his napkin, and borrowing my fountain pen, drew the following diagram:



He observed that he had drawn a C harmonica, but that the principle was the same for any key. Starting with one's mouth over the hole corresponding to the base note, in this case for the leftmost C, a scale is played as:

blow suck blow suck blow suck suck blow

I was puzzled by the irregularity in sucking and blowing but Dr Turing assured me that this was of no consequence.

He further explained that, to be more precise, the direction in which the harmonica is moved across the lips should also be indicated:

blow suck left blow suck left blow suck left suck blow.

Thus, one might right down a tune as series of rules of the form:

(breath, direction),

where "breath" may be "blow" or "suck" or "pause", and "direction" may be "left" or "right" or "rest". In fact,

Dr Turing used some Germanic script, but I am now unable to reproduce it.

Engaged by Dr Turing's whimsy, I hazarded that the first bar of "Twinkle Twinkle Little Star" might be written down as:

(blow, rest) (blow, left)
(blow, rest) (blow, rest)
(suck, rest) (suck, rest)
(blow, rest) (pause, right)

Dr Turing commended me on my acumen. I then quizzed him as to what practical purpose this might serve, as the rules seemed far less general than stave notation, and of little use other than for teaching beginners.

Dr Turing replied that he had long speculated about constructing an automatic machine to play the harmonica. The instrument might be mounted on a ratchet, driven by a motor, that passed it over the nozzle of a bellows. The rules could be punched as patterns on cards, and read by a mechanism like that for a street organ.

I applauded Dr Turing's vision but was puzzled as to why this was of any interest to a mathematician. Dr Turing patiently explained that he believed it possible to devise some sort of calculus that could tell whether or not his machine could play an arbitrary tune, just by looking at the rules.

He pointed out that the machine could only move the harmonica left or right by one hole for each note, so it could only play tunes that were composed of notes that were at most one hole apart. Otherwise, there would be unacceptable pauses in between the notes sounding.

Furthermore, my rendering of "Twinkle Twinkle Little Star" was inaccurate. The sequence should be:

(blow, rest) (blow, left) (pause, left)
(blow, rest) (blow, rest)
(suck, rest) (suck, rest)
(blow, rest) (pause, right)

The third note was actually two holes away from the second, so this was an example which the machine could not play.

I expressed my admiration for this bravura display of the higher mathematics. However, Dr Turing said that that the matter was somewhat more complicated than at first appearance. We know that the machine is unable to play this tune because we already know what it should sound like, so we can tell that the pause should not be there. But arbitrary tunes may have arbitrary pauses, so the calculus needs to capture some notion of what the tune should sound like. Then it might be possible to demonstrate that the rules for the machine corresponded exactly to the tune.

Fortified by the College's excellent claret, and taking a wild punt, I suggested that perhaps the stave notation might be a good starting point. Dr Turing concurred, and expressed a wish that arithmetic might be so easy to mechanise. Surely, I riposted, arithmetic could be done by any fool with a pencil and squared paper.

At this point, pudding was served and the conversation turned to other matters.

Shortly thereafter, I received my first posting from the Colonial Office, to Waziristan, as I shall next relate, and never encountered Dr Turing again. If only I had kept the napkin.

I have held my peace about this extraordinary reminiscence for quite some time. Of course, I am well aware that Turing's path breaking *entscheidungsproblem* paper went to press much earlier in 1936, and that he left for Princeton that September. These salient facts lead me to suspect that a prank is being played on me by some playful colleague.

Nonetheless, there is a ring of naïve veracity to my correspondent's Great Uncle's recollections. If any of your readers has any corroborative evidence for these curious assertions, then I would be delighted to hear from them.

Yours sincerely

<signature unreadable>

Invisible Numbers: Turing's Dark Matter?

Tim Denvir

Eight of us were having tea after a sociable day's walk along the Fife Coastal path. Someone observed that four of us were mathematicians; the others were in various occupations, a physiotherapist, a social scientist, a primary school teacher. The conversation briefly, but inevitably veered to mathematics. “I never understood those invisible numbers”, said Anna. Amid good-humoured chuckles we said, “You probably mean *imaginary* numbers”. No doubt we have all been in conversations like this from time to time, but I recalled this one a little later when thinking about Turing's computable numbers. We have real, rational, computable and transcendental numbers, and imaginary and complex, including algebraic, numbers, not to mention integers and their complex counterparts, Gaussian integers.

The recent centenary of Alan Turing's birth has propelled him into the public, as well as the specialist eye for the last few years. Most computer scientists, and surely all of FACS FACTS readers (!) will know of his canonical 1936 paper, *On Computable Numbers, with an Application to the Entscheidungsproblem*, but I suspect that only a minority may have read it.

In his introduction Turing writes: “The ‘computable’ numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means. Although the subject of this paper is ostensibly the computable numbers, it is almost equally easy to define and investigate computable functions of an integral variable or a real or computable variable, computable predicates, and so forth. The fundamental problems involved are, however, the same in each case... According to my definition, a number is computable if its decimal can be written down by a machine.” Computable functions are relevant to formal semantics of programming languages because we can soon get into difficulties of we assume that the functions expressible in programming languages can be modelled by the full gamut of mathematical functions. While that particular paper of Turing's does not go into the connection between computable numbers and functions in any great detail, I think it is reasonably easy to see it intuitively: a computer program which is designed to generate the

decimal expansion of a number, $\sqrt{2}$ say, is tantamount to a function whose argument is internalised. Computable numbers are of interest at more than the extreme theory end of the spectrum spanning the theory and practice of computer science.

In his 1936 paper, in order to define computable numbers, Turing first defines the *Computing Machine*, which we know now as the Turing machine. He does this by progressively elaborating notations for the configurations (states) of the machine and the symbols written on its tape, which effectively constitutes its store. He then defines the *Universal Computing Machine* (Universal Turing Machine), which can input a codification of the configuration of a computing machine and act accordingly. This is computationally equivalent to a conventional computer which can execute a program stored within it. The detail of the exposition and examples of computing machines over the first twelve pages in the paper is painful in its intensity and I do not pretend to have followed it all down to the last symbol. But it is all very reminiscent of certain lectures in mathematical logic which I attended many years ago in my last undergraduate year. Turing also acknowledges that Alonzo Church in a previous paper had defined “Effectively Calculable” numbers, though in a very different way.

In section 8 of the paper, Turing shows that the set of computable numbers is enumerable (or countable). This seems almost obvious, since the computable numbers are those precisely which can be generated by a computing machine, and those computing machines in turn are representable by finite sequences of symbols, which comprise a countably infinite set. Nonetheless, Turing goes to some further intensely detailed pages to prove the matter. In section 9 he attempts to show that computable numbers include numbers which a human computer with pen and paper etc. can compute by normal calculating processes. I must confess that at first for a moment I thought that Turing was anthropomorphising his machine by referring to it as “he”, and “his state of mind”, but of course in 1936 there were no computers as we know them; a computer was a *person* who carried out calculations. So Turing is comparing the computations which his computing machine can perform with those that a human “computer” can.

In section 10 he shows, and in some cases proves, that various other numbers and functions are computable:

A computable function of a computable function is computable;

Any function of an integral variable defined recursively in terms of computable functions is computable;

Although a bounded sequence of computable numbers does not necessarily have a computable limit, one can devise a definition of computable convergence where the limit of a computably convergent series is computable;

From the above, it is shown that numbers expressible as the sum of a suitable series are computable, such as π and *e*.

Further, all the real algebraic numbers are computable (an algebraic number is any, possibly complex, number which is the root of a polynomial in one variable with rational (or equivalently, integer) coefficients);

We might term any number which can be defined as the value of some formula as a *definable* number. Of course algebraic numbers are definable, but so are plenty of others: trigonometric functions (of definable numbers) and sums of some series and limits of other parametrised formulae for example. Many of these are computable. Turing showed that the computable numbers are a countable set, and I would claim that the real definable numbers are also countable.

Any formula is finite in length and expressed in some finite alphabet. The set of finite sequences of characters from a finite alphabet is a countable set. (You can express the characters as a fixed-length sequence of 0-1 bits and then each formula is a unique binary integer, although not every integer translates into a meaningful formula). However, you might retort that there is an unlimited number of possible notations with unwritten conventions in which we might define such numbers and functions; the whole gamut of these as yet to be imagined notations may not be countable! I would reply that, yes, we cannot predict how many such notations might be imagined in the future, but, even if they are boundless, there are surely not more than a countable infinity of these possible notations; and the union of a countably infinite set of countably

infinite sets is still a countable set. A bit of a hand-waving proof perhaps, but I hope convincing enough.

Thus, within the real numbers, the rationals, the computable numbers, the definable numbers, are all countable subsets. That leaves the rest, numbers that we cannot define at all, that we have no means of identifying. Since the reals are uncountable, that means these undefinable numbers are also uncountable, for they are what remains after we have removed all those other countable subsets.

I was explaining this to a friend who said, “Aren't they the transcendental numbers?” No, the transcendental numbers are simply those which are not algebraic; they include π and *e* for example. Thus the transcendentals include some definable, indeed some computable numbers. The undefinable numbers are a subset of the transcendentals. So these undefinable numbers, which we *cannot define or identify in any way*, infinitely outnumber all the rest. They put me in mind of dark matter, which cosmologists deduce permeates the universe, but which no-one has ever seen or found. For this reason, and to acknowledge Anna's accidental nomenclature, I would like to call them *invisible numbers*.

Finally, I feel I must say a word about the *Entscheidungsproblem*. German for “decision problem”, this was posed by David Hilbert in 1928. Crudely put, it asks if an algorithm can be devised which, given some axioms and a statement, can determine whether the statement can be deduced from the axioms using the rules of formal logic. Alonzo Church and Alan Turing independently, and at about the same time in 1936, proved that this was not possible. Church produced his solution shortly before Turing, which Turing acknowledged in his *Entscheidungsproblem* paper. Church's solution relied on reformulating the problem in his λ-calculus. Many FACS FACTS readers will notice an intuitive similarity between Hilbert's problem and Gödel's incompleteness theorem, and I read that both Church and Turing were influenced by Gödel's earlier work. Hilbert's *Entscheidungsproblem* was the third of three problems posed by him at a conference in 1928, and these were a continuation of his “programme” of 23 problems which he initially posed in 1900.

Oh, and “finally, finally”, it seems generally accepted that a machine constructed following a von Neumann architecture is computationally

equivalent to a Turing machine, that is, they can both perform the same computations. So “computable” in all the foregoing can be taken to mean computable by modern machines.

Alan Turing was just 24 when he published his *Entscheidungsproblem* paper.

References

Turing, A.M. (1936), "On Computable Numbers, with an Application to the Entscheidungsproblem", Proceedings of the London Mathematical Society, Series 2.

David Hilbert and Wilhelm Ackermann (1928). Grundzüge der theoretischen Logik (Principles of Mathematical Logic). Springer-Verlag, ISBN 0-8218-2024-9.

Alonzo Church, "A note on the Entscheidungsproblem", Journal of Symbolic Logic, 1 (1936), pp 40-41.

Christopher Strachey, Pioneer of FACS

Joseph E. Stoy
Bluespec

Venue: BCS, Southampton Street, London

Tuesday 15th November 2016

Reported by Troy Kaighin Astarte
Newcastle University

The first thing I encountered upon being introduced to Joe Stoy prior to the talk was his collection of wonderful props. In quick succession, I was shown a photo of 45 Banbury Road, the erstwhile location of the Programming Research Group in Oxford, where Strachey worked in the final decade of his life with Joe as his right-hand man; a box of glass 'magic lantern' slides of the output of Strachey's famous draughts playing program; two books on programming language theory and semantics which Joe indicated had been his introduction to the topic; and, best of all, a copy of a timeline, hand drawn by Strachey, of the membership of the PRG, given to Joe as 'memorabilia' when he left Oxford in 2001. These were of great interest to me, as they related directly to my research, and clearly drew the attention of many of the other guests as well.

The second thing I noticed about Joe (the reader will forgive the familiarity; I feel I got to know Joe quite well during the succeeding week) was the effusive and charming manner with which he conducted himself. He chatted easily with me and many of the other guests as we awaited, with growing excitement, his talk.

Once it began, Joe's talk swept along at pace. He skilfully weaved one tale after another into a coherent narrative, and painted for us a picture of Christopher Strachey, a man who Joe clearly admired deeply. His pure white hair bouncing as he gesticulated, Joe would stop himself mid-anecdote to tell us with a twinkle in his eye another story; but the coherence of his talk didn't suffer for

this and I left with a deeper understanding of both Strachey the scientist and Strachey the man.

Joe began by describing Strachey's characteristic flamboyant and didactic style (I couldn't help but wonder how much of Joe's lecturing style was learnt from Strachey), relaying the story of how a five-year-old Christopher was found explaining to his nanny the meaning of a one in five gradient, and took us through a roughly chronological journey through the man's life.

A strong thread was Strachey the originator and innovator: Strachey programmed Canada's first computer with one of the most technical engineering calculations in the St. Lawrence Seaway project; Strachey wrote the largest program written at the time for the Ferranti Mark I at Manchester with his draughts program (and displayed the results using the CRT which had previously only been used for memory); Strachey was among the first to present work on time-sharing, though in the multi-programming sense in contrast to McCarthy's view of a multi-user system.

Another theme was Strachey's interest in both the theory and practice of computing: Joe would explain the work being undertaken at the PRG at a few points in time, and separate these into theoretical and practical—but he would carefully point out the links between the two, and the fact that most people appeared on both lists. This was also supported by the quotation from Strachey:

"*It has long been my personal view that the separation of practical and theoretical work is artificial and injurious. Much of the practical work done in computing, both in software and in hardware design, is unsound and clumsy because the people who do it have not any clear understanding of the fundamental design principles of their work. Most of the abstract mathematical and theoretical work is sterile because it has no point of contact with real computing. One of the central aims of the Programming Research Group as a teaching and research group has been to set up an atmosphere in which this separation cannot happen.*"

This also led to Joe explaining a few times that Strachey was not, despite his interest in the application of mathematics to computing, a mathematician. This was illustrated by a letter sent to the CACM by Strachey about the halting problem which contained a hidden but important error, as well as the confident way Strachey used lambda notation to model programming languages well before a formal model was constructed by Dana Scott.

A theme from which Joe clearly (albeit quietly) derived some pleasure was the PRG as a location for the origins of many important concepts in computing. The obvious one is Christopher's darling programming language CPL, a subset of which was implemented by Martin Richards as BCPL, which was used by Thomson and Ritchie to write their earliest versions of UNIX (they subsequently developed a smaller language which used the same ideas and called it B; the next iteration was called C), but Joe also told us about the link from David Turner's attempt to implement Christopher's Pedagogic Algorithmic Language (PAL), which led to SASL, KRC, and ultimately Miranda, a language which inspired the freer Haskell. Back on the UNIX theme, Doug McIlroy visited the PRG for one year (to learn of denotational semantics direct from the source, as he later wrote), and during that time came up with the concept of pipes, although the syntax was different.

A final theme that came through about Strachey was his sense of humour, and love of literary allusion. This was illustrated beautifully when Joe showed the first page of Strachey's (1973, but published only in 1997) paper 'The Varieties of Programming Language', which came "with apologies to Professor William James, Miss Stella Gibbons and the late Herr Baedeker." This, Joe explained, was a reference to James' paper 'The Varieties of Religious Experience', singular, which explained the singular 'Language'; and with the air of one who knew a great punchline was coming, he touched a button and a little less than half of the prose of the first page of the paper lit up yellow. "The yellow parts," explained Joe, "are Strachey. The rest are James." The room rumbled with laughter. The other apologies were in reference to the two asterisks which preceded the first paragraph, as Gibbons had marked sections of her novel *Cold Comfort Farm* with a number of asterisks conformant with their level of purple prose, and Baedeker had used a system of asterisks in his guide books to

indicate how worth visiting a particular point of interest was. Two asterisks meant “rather purple” and “worth a detour” respectively.

Altogether Joe’s talk was enjoyable, informative, and amusing. I cannot have been only person in the audience who was surprised to notice when Jonathan Bowen stood up to call the end that nearly one hundred minutes had elapsed. Throughout the talk, Joe’s style had engaged and absorbed us all: clearly he is a man used to holding and working an audience. It is a testament to Joe’s skill that although I saw him deliver a subset of the same talk less than a week later at the Strachey 100 centenary event in Oxford, it was just as fun and interesting the second time around.

Joe Stoy is the author of *Denotation Semantics: The Scott-Strachey Approach to Programming Language Theory*, published by MIT Press in 1977.

Joe Stoy with a slide of Christopher Strachey during the talk.
(Photograph by Jonathan Bowen)

Ada Lovelace's 200th Birthday Celebration at Oxford

December 9th - 10th 2015

Reported by Margaret West

On Thursday 10th December 2015, Ada Lovelace, the first programmer, would have been 200 years old. I attended a Symposium In honour of this event at Oxford University which took place on 9th and 10th December. The symposium which was interdisciplinary included lectures from scholars from the Humanities as well as from Computer Scientists. On the evening of December 9th there was a Dinner at Balliol College where the Earl of Lytton proposed a toast to Ada Lovelace (his Great Great Grandmother) and on December 10th a cake was shared by all.

An account of the Symposium written by Ursula Martin (the organiser) is here:

http://blogs.bodleian.ox.ac.uk/adalovelace/

I see that many messages of appreciation were received with which I most thoroughly concur for this was a most interesting and thought provoking event. The lectures from the Symposium are now online in the Oxford Podcast Series:

http://podcasts.ox.ac.uk/series/ada-lovelace-symposium-celebrating-200-years-computer-visionary

NB: Doron Swade's talk is currently unavailable, pending resolution of a permissions issue

Photographs from the event are available here:

https://www.flickr.com/photos/computerscienceoxford/sets/72157662371814411

Follow-up articles and other comments through *twitter* can be tracked at the hashtag *#LovelaceOxford*

https://twitter.com/search?f=tweets&vertical=default&q=%23lovelaceoxford&src=typd

Prof. Dana Scott – Talks in the UK

Reported by Margaret West

Professor Dana S. Scott (Prof. Emeritus, Carnegie Mellon University and Visiting Scholar in Mathematics, UC Berkeley) gave a number of talks during his visit to the UK during 2016. Professor Scott is a Turing Award-winner and recipient of many other international awards. He is a distinguished mathematical logician with a long career who has made fundamental contributions to set theory, model theory and the theory of computation. In particular Scott worked with Christopher Strachey at Oxford University on providing a mathematical foundation for the semantics of programming languages: the Scott-Strachey approach to Denotational Semantics.

Talks given in the UK included:

- Leeds University, 17th May: Logic Seminar "Types and Type-Free Lambda Calculus"
- Leeds University, 18th May: Löb Lecture "Why Mathematical Proof".
- University of Cambridge, 20th May: "Why Mathematical Proof?"
- Imperial College, London, 26th May: "Stochastic Lambda-Calculus"
- British Computer Society, London, 26th May: "Lambda Calculus: Then & Now"
- University College London, 27th May: "Types and Type-free Lambda Calculus "
- Queen Mary University of London, London, 1st June: Joint Maths Colloquium/EECS Distinguished Seminar "Why Mathematical Proof?"

Further details can be obtained:

https://www.maths.leeds.ac.uk/home/news/lob-lecture.html

http://talks.cam.ac.uk/talk/index/65188

https://verificationinstitute.org/2016/05/talks-by-dana-scott-acm-a-m-turing-award-1976-thursday-26-may-friday-27th-may-and-wednesday-1-june/

The following is a summary of the Löb Lecture at Leeds on 18th May 2016:

Why Mathematical Proof?

The lecture was introduced by Professor Stanley Wainer - who explained that the talk had been originally arranged with the help of Professor Barry Cooper. However Professor Cooper had sadly died while arrangements were still being made. Tribute was paid to Barry Cooper by Stanley Wainer and - later - by the speaker.

The talk was is in honour of Martin Hugo Löb (1921-2006), the founder of the Leeds Logic Group - a refugee from Nazi Germany who arrived in the UK just before World War 2. In 1940 he was deported to Australia as an "enemy alien" - where he was taught mathematics in the internment camp by other internees. He was allowed to return to the UK in 1943 where he continued his studies - eventually becoming a research student with Reuben Goodstein at the University of Leicester. After he had gained his PhD he was appointed as a lecturer at the University of Leeds where he developed the mathematical logic group. He is best known for formulating Löb's theorem in 1955. He became Professor of Mathematical Logic at Leeds in 1967 where he remained until the early 1970s when he became professor at the University of Amsterdam.

Professor Wainer then introduced the distinguished speaker.

Professor Scott remarked in his introduction that during his first visit to Leeds he had met Professor Löb. He commenced the "entertainment", as he termed his talk, with a timeline for Geometry commencing with Thales (in 600BC) and Euclid (300BC). The timeline finished with modern geometry - including Eliptic and Fractal Geometries.

It is notable that Euclid authored the most successful text book ever produced. The speaker questioned why Euclidean Geometry was so successful and thought that it was because our naive feeling is Euclidean and also there is a connection between (visual) intuition and proof.

He presented the most common proof of Pythagoras' Theorem and pointed out that "auxiliary lines" or "constructions" have to be added to enable some proofs. The three-dimensional extension of Pythagoras viz "Eulers Brick" was then discussed relating the relationship of the diagonals with the dimensions of the

brick (a, b, c), where:

$a^2 + b^2 = d^2$
$a^2 + c^2 = e^2$
$c^2 + b^2 = f^2$
$a^2 + b^2 + c^2 = g^2$

Given the above, is it possible to have all digital a, b, c, d, e, f, g? This general result is still unknown. It was however shown in 1719 that the smallest solution for all except g digital is (a = 117, b = 240, c = 44) and exhaustive computer searches indicate that if such a brick exists there is no solution smaller than values of the order of 10^10.

Professor Scott recommended a series of books: "Proofs Without Words" – featuring diagrams which enable the reader to see why a theorem might be true. The diagrams (or pictures) also help the reader to intuit a proof. The speaker went on to discuss the proofs of irrationality of the square roots of 2,3, 5 by the use of diagrams. The speaker wondered if we really needed proofs.

Padua (Langlands, 1937) remarked that logic is not in a particularly fortunate position:

> "On the one hand, philosophers prefer to speak of logic *without using it* while on the other hand mathematicians prefer to use it *without speaking of it* – and even without desiring *to hear it spoken of.*"

David Gale thought "mathematics was about ideas which explain and thus enable us to understand" and that appreciating mathematics means "learning to recognise and appreciate **beautiful things.**"

The speaker returned to the discussion on the use of "proof by diagram" by a further example, a tiling problem, filling a hexagonal "box" with a set of rhombi. There are three possible orientations of the rhombi and if these are coloured it can be seen that there are equal numbers of each orientation. (However Djikstra rejected this as a form of proof and produced a more rigorous one.)

The speaker spoke of the beauty of mathematics and went on to quote Galileo – who said that the book of the Universe is written in the mathematical language

comprising triangles, circles and other geometrical figures - without which it is not possible to comprehend it. Sir Michael Atiyah has commented (in Nature, December 2005) on the remarkable use of string theory in Physics in explaining the Universe and that the theory has a number of applications in areas which are far removed from mathematics. Atiyah further states: "To many this indicates that string theory must be on the right track. ... Time will tell."

After further presentation of results in some interesting areas of mathematics including the stereographic projection of the globe and knot theory Professor Scott asked the question "Is mathematics discovered or invented?" This was discussed by Paul Ernest (1996) who remarked that the "absolutist" view (shared by Roger Penrose among others) sees mathematical truths as "discovered" by the mathematician and then established by proof. The remarkable thing is that mathematics provides a surprisingly useful framework for modelling the Universe and the feeling is that it must be woven into the fabric of the world.

The speaker went on to discuss Clifford algebras which can be thought of as a possible generalisation of complex numbers and quaternions. The theory of these algebras has important applications in geometry, computer graphics and theoretic physics.

Towards the end of the talk the speaker asked "When does a Proof become a PROOF" and the answer is when it has been socially accepted as such. The ideal is unchanged since Euclid where proof is obtained by a series of deductions from a series of proven assertions.

Scott further suggested that at this time of computer based reasoning it is a good time for seminars and discussion groups on proofs and logic. He quoted from Wolfram's blog as to what the mathematician Ramanujan would have done if the Mathematica tool had been available when he was engaged with his experiments. He thought Ramanujan would have enjoyed experimenting with the tool and that a lesson should be learned. In other words it is good for mathematicians to be adventurous and experiment - even if the broader context is not understood at the time.

Questions included several on computer-assisted versions of proofs and their implications. Professor Scott thought that at the least they helped in re-organising and simplifying proofs.

BCS-FACS/LMS Evening Seminar
Joint event with the London Mathematical Society

Probabilistic formal analysis of software usage styles in the wild

Prof. Muffy Calder
(University of Glasgow)

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

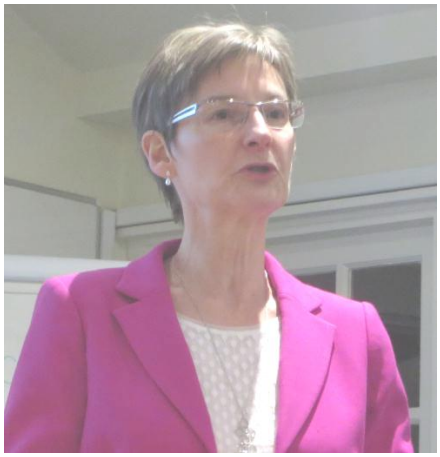
Thursday 3rd November 2016, 6:00pm

Reported by Jonathan Bowen

Abstract: *Discrete mathematics and logics are used to analyse the intended behaviour of software systems. Statistical methods are used to analyse the logged data from instrumented systems. So what happens when we instrument software: can we bring the two techniques together to analyse how people actually use software?*

But users are difficult – they adopt different styles at different times! What characterises usage style, of a user and of populations of users, how should we characterise the different styles, how do characterisations evolve over an individual user trace, and/or over a number of sessions over days and months, and how do characteristics of usage inform evaluation for redesign and future design? Can we formalise these concepts and construct effective procedures?

Professor Calder outlined a novel mathematical/computational approach that aims to answer all these questions. The approach is based on discrete space stochastic models, statistical inference of those models, and stochastic temporal logics and model checking for investigating hypotheses about use, all applied to longitudinal sets of logged usage data. The approach is the result of a five-year collaboration between software developers, statisticians, HCI, and formal methods experts. She will illustrate by way of a mobile app that is used by tens of thousands of users worldwide; a new version of the app, based on the analysis and evaluation, has just been deployed. This is formal analysis in the wild!



(Photograph by Jonathan Bowen)

Professor Muffy Calder of the University of Glasgow gave a talk to members of the BCS and LMS at De Morgan House on the evening 3rd November 2016 in the annual LMS/BCS-FACS evening seminar, organized and chaired by John Cooke, the liaison officer between the LMS and BCS-FACS Specialist Group.

The talk was based on collaborative work with Oana Andrei, Matthew Chalmers, Alistair Morrison, and Mattias Rost. It covered statistical methods, as used to analyse logged data from instrumented systems (e.g., for smart cities, etc.), and discrete mathematics, specifically temporal logics, as employed to analyse the intended behaviour of software at design time (i.e., formal methods). But what if software applications are already implemented? Can statistical and formal methods be used to analyse how users actually navigate applications? The work was motivated by the need to evaluate and redesign user-intensive apps in supporting the user's style of interaction, based on actual usage. It was noted that this is a very dynamic situation, with different users having different styles at different times, and not static (e.g., the user's age, location, gender, etc., all play their role).

The talk considered the problems of what characterises usage, how to model usage style in a population of users, how to identify different styles of usage, and how these styles evolve. Characterisation of usage was modelled with user traces, consisting of sequences of actions. Activity patterns can model the usage style in a population. There are different types of model, all probabilistic Markov models. Specifically, discrete time Markov chains (DTMC) were used in

the modelling. Different treatments of inferred variables in the model permit various questions to be considered.

Usage style in a population can be modelled where each trace is an "admixture" of a number of activity patterns (DTMCs) shared within the population of users. Admixture models derive from genetic analysis of populations where individuals have mixed ancestry: each individual inherits a fraction of his/her genome from ancestors in a population. Here, each trace does not have one fixed trait, but has an (inferred) probability distribution over the different usage styles.

Different styles can be identified by hypothesising temporal logic properties. Styles evolve over days, weeks, or months. The approach is novel in that it does not use design-time analysis of the functional behaviour of what a user could do, but rather analyses actual usage after deployment of what the users actually did. It is a scientific approach to studying an artefact that has been engineered. Here the software system usage is an object of study (after performing a scientific "experiment") and temporal logic is used as the means of performing this study. The work presented was in the context of applications (apps) on mobile phones, but is appropriate for any system with user interaction.

Analysis is in five steps. First, events on a user's phone are sent as a batch of logged timestamped events to the developer's server. Next, the raw logged data is cleaned and prepared. User traces are based on selected state abstractions. The session data is segmented (e.g., by day) and a transition-occurrence matrix is computed from each trace in each data set. Each user trace is characterised as an admixture of K activity patterns through a process of inference.

Consider K discrete-time Markov chains (or activity patterns). $\Phi_k[i,j]$ is the probability of moving from one state i to another state j while in Φ_k. For each user trace, there is a weight vector $(\Theta_1, \ldots, \Theta_K)$ where Θ_k is the probability of using the k^{th} activity pattern. Inferences are drawn from the user traces, and the goal is a model that explains that data set, it is not for prediction. An

expectation-maximization (EM) algorithm is run to learn the activity patterns and their probability distribution.

Questions are asked about the patterns using probabilistic temporal properties with rewards, based on Probabilistic Computation Tree Logic (PCTL). For example, what is the probability of reaching a state for the first time within a certain number of steps? Or what is the expected number of steps to reach a state from another state? Is it more likely to change activity pattern after visiting a given state? And so on. Such questions can be formulated and posed in the formal logic. The answers provided are quantitative and proved using the PRISM model checking tool. Tractability depends on the high-level states of the application, not the size of user trace data. The results can be discussed with the software app developers to evaluate styles as well as aiding redesign and future designs. If results are unexpected, further properties can be considered.

The AppTracker app was used as case study. It provides “personal informatics”, recording the opening and closing of apps on a smartphone, as well as the locking and unlocking of the device, running in the background. It provides charts and statistics about the usage of the device and has had over 35,000 downloads.

For the AppTracker app, there are 15 high-level states. Values of K between 2 and 5 were considered, with seven intervals between 0 and 90 days. For example, for K=2, there are two styles based on overall and in-depth activities. The session length indicated short glancing interactions by users. It was hypothesised that styles follow the main menu; however, this was quickly disproved by considering the results for K=3. One novel outcome was how activity patterns can inform the development of glancing widget extensions.

As a result of the analysis, and discussions with the developers, the high level menu structure, and underlying functionality (including glancing) of AppTracker was changed and a new release issued in May 2016. This release was instrumented and user data logged – analysis on this new data set is in progress. Stay tuned to hear whether the new design better supports user styles of interaction.

The speaker concluded by noting the contributions. User populations are characterised by inferred temporal behaviours rather than static user attributes. In particular, inference of Markov models of usage patterns from logged user sessions is possible and activity patterns can be characterised by probabilistic temporal properties using model checking. Analysis of a real mobile app informs developers about actual use and helps with future redesign. The process of redesign/implementation, logging, and analysis can be repeated as necessary.

For further BCS-FACS information on the talk, including a copy of the slides, see:

http://www.bcs.org/content/ConWebDoc/56315

For further LMS information on the talk, including details of previous LMS/BCS-FACS talks since 2008, see:

https://www.lms.ac.uk/events/lectures/lms-bcs-facs-evening-seminars

Acknowledgement: Thank you to Muffy Calder for checking, correcting, and augmenting the original draft of this report.

BCS-FACS Annual Peter Landin Semantics Seminar

Building Trustworthy Refactoring Tools

Prof. Simon Thompson
(University of Kent)

Venue: BCS London office

Monday 12th December 2016, 6:00pm

Reported by Sofia Meacham and Jonathan Bowen

Abstract: *Refactorings are program transformations that are intended to change the way that a program works without changing what it does. Refactoring is used to make programs more readable, easier to maintain and extend, or to improve their efficiency. These changes can be complex and wide-ranging, and so tools have been built to automate these transformations.*

Because refactoring involves changing program source code, someone who uses a refactoring tool needs to be able to trust that the tool will not break their code. In this talk I'll explore what is meant by "preserving meaning" in practice, and how we provide various levels of assurance for refactorings, ranging from testing to full, machine assisted, verification. While the context is tools for functional programming languages like Haskell, Erlang and OCaml, the conclusions apply more widely, for instance to object-oriented languages.

(Photograph of Simon Thompson during the talk, by Jonathan Bowen)

This talk had as a main theme refactoring tools and their trustworthiness and how this relates to Peter Landin's functional programming influence. Presented by Simon Thomson, University of Kent, the research was supported by UK EPSRC and the European Commission. Throughout the talk, an attempt to address the following important question from the coder's point of view was made: *"Why should I trust your refactoring tool on my code?"*

From the start, the speaker addressed the following: *What does refactoring mean?* It means changing how a program works without changing what it does.

Why Refactor? Refactoring is undertaken to extend and reuse code (e.g., function calls), increase comprehension, and counteract software decay.

How to Refactor? There are two ways to refactor: firstly, by hand, using an editor which is a flexible but error-prone approach, but is infeasible for large programs; secondly, using tools, which is scalable to large programs, integrated with tests and macros, handling transformation and analysis.

Recent literature is not very encouraging with regard to refactoring by tools. Specifically, the following are stated: *"up to 90% of refactorings are done by hand and some 40% of refactorings performed using tools are done in batches. Automated refactoring is highly unlikely to replace the 'human in the loop'."*

At the University of Kent, the *Wrangler* refactoring tool is being used extensively. *Wrangler* is an interactive refactoring tool for Erlang (a functional programming language for concurrent distributed systems) where simple aspects are automated and decision support tools are provided otherwise. It is embedded in common IDEs (Integrated Development Environments) such as Emacs and Eclipse.

An important question to be answered is: *Shall we trust refactoring tools or do we need to provide verification for the refactoring process?* There is literature that supports the view that refactoring tools are trustworthy enough as well as

literature supporting the position that trust must be earned. Both approaches are supported by reasonable arguments.

In looking at what coders might require, the speaker identified a range of views. Some – the most pragmatic, perhaps – would be happy *if the refactoring tool would hit 95% of the cases … and they had to fix the last 5% by hand (and the compiler) … beats doing them all by hand.* On the other hand, some not only require that the code does the same thing, but also that it will have the same layout as before, including getting the layout right for any new code that is produced.

Between these two positions, and relating to Peter Landin's work on semantics, is the view that refactorings should preserve meaning. The speaker then argued that this idea itself needed clarification: does just the meaning of the main program need to be preserved, or more of the structure; does meaning extend to test suites, *makefiles*, and so on?

If we assume fixed context and scope, assurance of meaning preservation consists of testing and verification of instances of the refactoring – that is the result of applying the refactoring to a particular program – and of the refactoring itself, i.e. every possible instance of it. Four combinations are relevant: testing instances, testing the refactoring, verifying instances, and verifying the refactoring. For the testing approach, regression tests represent the state of the art, but the speaker showed that randomly test data – including random programs – can provide effective testing. For the verification approach, SMT (Satisfiability Modulo Theories) solvers have been shown in principle to work for some cases of instance verification, whereas proof assistants, such as Isabelle and HOL are the direction proposed by the speaker for verifying refactorings in general.

Future plans include a trustworthy refactoring project. The overall goal of this project is to investigate the design and construction of trustworthy refactoring

tools. It uses CakeML (a substantial subset of ML with a semantics specified using higher-order logic) for fully formally verified refactorings of a certified language and compiler as well as the use of SMT solving for high-assurance refactoring. For more information, visit:

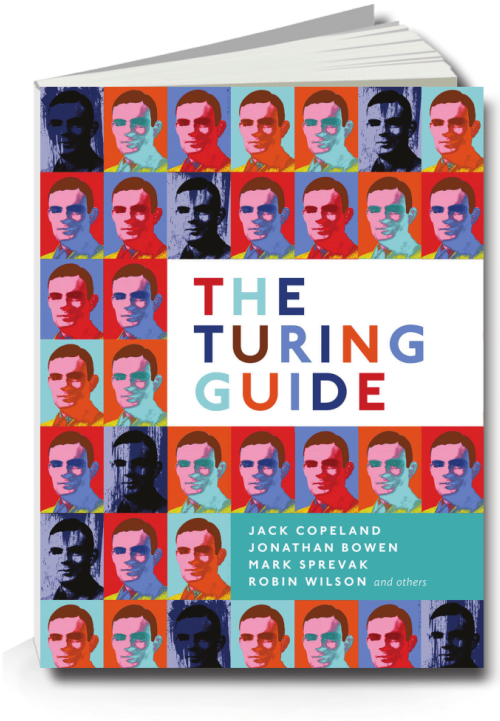
https://www.cs.kent.ac.uk/projects/trustworthy-refactoring/Trustworthy_Refactoring/Home.html

---

Acknowledgement: Thank you to Simon Thompson for checking, correcting, and augmenting the original draft of this report.

OXFORD
UNIVERSITY PRESS

The Turing Guide



Jack Copeland, Jonathan Bowen, Mark Sprevak, and **Robin Wilson**

- A complete guide to one of the greatest scientists of the 20th century
- Cover aspects of Turing's life and the wide range of his intellectual activities
- Aimed at a wide readership
- This carefully edited resource written by a star-studded list of contributors
- Around 100 illustrations

This carefully edited resource brings together contributions from some of the world's leading experts on Alan Turing to create a comprehensive guide that will serve as a useful resource for researchers in the area as well as the increasingly interested general reader.

"The Turing Guide *is just as its title suggests, a remarkably broad-ranging compendium of Alan Turing's lifetime contributions. Credible and comprehensive, it is a rewarding exploration of a man, who in his life was appropriately revered and unfairly reviled."*

- Vint Cerf, American Internet pioneer

"The Turing Guide *provides a superb collection of articles written from numerous different perspectives, of the life, times, profound ideas, and enormous heritage of Alan Turing and those around him. We find, here, numerous accounts, both personal and historical, of this great and eccentric man, whose life was both tragic and triumphantly influential."*

- Sir Roger Penrose, University of Oxford

JANUARY 2017 | 544 PAGES
PAPERBACK | 978-0-19-874783-3
£19.99 | $29.95
HARDBACK | 978-0-19-874782-6
£75.00 | $115.00

Ordering Details

ONLINE www.oup.com/academic/mathematics

BY TELEPHONE +44 (0) 1536 452640

POSTAGE & DELIVERY

For more information about postage charges and delivery times visit www.oup.com/academic/help/shipping/.

The specifications in this leaflet, including without limitation price, format, extent, number of illustrations, and month of publication, were as accurate as possible at the time it went to press.

UK Network on the Verification and Validation of Autonomous Systems

Prof. Michael Fisher
(Department of Computer Science, University of Liverpool)

Autonomous Systems. By "autonomy" we mean the ability of a system to make its own decisions about what to do and when to do it, without needing human intervention. So far, most of the systems deployed, such as robot vacuum cleaners, aircraft autopilots and automated parking systems in your car, are just pre-programmed to adapt to environmental stimuli. However, we can expect that many household/business/industrial systems will become increasingly autonomous.

There are many situations where humans cannot (due to large distances, danger, or very fast moving entities) or choose not to (due to the mundane, dirty, or repetitive nature of tasks) control these systems directly. Obvious examples are space vehicles that must operate in distant environments or unmanned air vehicles that must fly safely in crowded skies. Such vehicles must move, navigate, avoid dangers, and safely land/dock, usually without intervention from a human operator. Similar vehicles are soon to be deployed in many, less exotic, areas: environmental monitoring; surveillance; (freight) transport; etc. Examples of systems carrying out tasks that humans choose not to undertake include autonomous cleanup systems, robotic assistants (at both home and work), and health-care robots. Here, the main focus is often on interaction and cooperation, either with other autonomous systems (perhaps in swarms) or with humans.

This future may be exciting, but the thought of truly autonomous systems can also be uncertain and unappealing, not only to members of the public, but to engineers and to regulators. Even partial autonomy invokes these reactions. How can we be sure such autonomous systems are safe? When they can act autonomously, how can we be sure they will do what we require? How can we be sure they are legal, usable, and unthreatening?

Network Focus. Clearly with such advanced technologies, we must invoke techniques for analysis that provide much higher confidence than usual. Consequently, the Verification and Validation (V&V) processes used for traditional systems must be enhanced to provide increased confidence in the next wave of autonomous systems. Although scientists in the UK have made some key advances, there has been no organisation to focus on what still needs to be done and how the different approaches might be combined.

EPSRC has now funded a UK Network on the Verificaiton and Validation of Autonomous Systems[1] to bring together researchers working on the novel V&V techniques required for autonomous systems. It is important to note that many issues remain unchanged as we move towards autonomy. For example, the materials used in the construction of autonomous systems might well be identical to those used in human-controlled systems. Consequently, we are concerned with the V&V of the new aspects that come to the fore in dealing with autonomy. Primarily, these centre on the autonomous control, decision-making, adaptation, and even learning, that the system might undertake when replacing a human controller, driver or operator.

This step change in the way V&V is carried out is both complex and inter-disciplinary — it clearly re-quires expertise from Engineering, on the predictability and resilience of control, from Electronics, on the reliability of sensors and communication, and from Computer Science, on formal methods, software engineering and software testing. However, especially where human interaction is involved, the collaboration with experts from Psychology, Law, and Sociology is important: through social robotics, human-machine interaction, legality and liability, etc. The Network also involves these legal and societal areas to provide a more comprehensive view of the potential for autonomous systems.

Impact. While this is a specifically academic network, it clearly has importance and relevance to industrial and regulatory contexts. To give an indication of this breadth, we can at least expect that new verification and validation techniques will be needed for human-robot teamwork, both in work and home contexts,

[1] http://vavas.org

Safe (and road-worthy) driverless cars, autonomous robotics in nuclear/chemical/biological processes, certification of unmanned air vehicles, autonomous ocean surface monitoring and exploration, and Robotic diagnosis, rehabilitation, or surgery.

Formal Methods. The aim of Verification is to ensure that our system matches its requirements. These requirements may be informal, in which case it is hard to assess if, or how, our system does indeed cor-respond to them, or the requirements may be explicitly formal. The formal variety is often given in a clear, precise language with unambiguous semantics. Formal Verification takes this further, not only having precise formal requirements in a mathematical form, but carrying out a comprehensive mathematical analysis of the system to 'prove' whether it corresponds to the formal specification of these requirements. Formal verification is particularly used for systems that are safety, business, or mission critical, and where errors can have severe consequences. While formal verification, via model checking, is widely used especially for the analysis of critical systems, its use in autonomous software is relatively recent [1, 4], while application to the verification of practical autonomous systems is still at an early stage [3, 2].

Network Implementation. The Network is open to any academic and its primary aim is to stimulate, coordinate, promote, and disseminate research on the verification and validation of autonomous systems.

The Network is funded by EPSRC for 3 years from 1st September, 2015. It has a web-site, vavas.org, and currently over 70 academic members. It mainly funds events (typically workshops) to stimulate different research and exploitation themes and activities. For example, we have so far organised:

Sep. 2015: workshop on "Agent Verification" in Liverpool;

Feb. 2016: workshop on "Legal/Regulatory Aspects and V&V" in London;

Jul. 2016: workshop on "Industrial Perspectives on the V&V of Autonomous Systems" in Sheffield; and

Nov. 2016: workshop on "Verification and Validation of Autonomous Road Vehicles" in London.

Details of all these, and presentations provided within them, are available at vavas.org

The Network also promotes education and dissemination and so supported the

> "Winter School on Verification of Mobile and Autonomous Robots" at York in Dec. 2015.

All the above activities are not purely academic. We are keen to engage with stake-holders not only from industry, but across other academic disciplines, and involving public/policy-makers.

Summary. Robots, driverless cars, unmanned air vehicles, etc, can all be built now. Yet the main barriers holding back the widespread use of autonomous robotics can be seen as societal: what should the legal framework be for such systems; how can the public come to trust these systems; how can we ensure they are safe; and how do we know such a system will make the decisions *we* would expect of it? Increasingly, the key problem is not just to construct an autonomous system or robot, but to construct its software in such a way that it is (certifiably) safe, reliable, and trustworthy. All these problems surely require strong V&V techniques, including formal methods. Constructing autonomous systems without behaviour guarantees can lead to serious outcomes, and may consequently hold back the adoption of truly autonomous systems.

To join the Network, see http://vavas.org

References:

[1] R. Bordini, M. Fisher, W. Visser, and M. Wooldridge. Verifying Multi-Agent Programs by Model Checking. J. Autonomous Agents and Multi-Agent Systems, 12(2):239-256, 2006.

[2] L. Dennis, M. Fisher, N. Lincoln, A. Lisitsa, and S. Veres. Practical Verification of Decision-Making in Agent-Based Autonomous Systems. Automated Software Engineering 23(3):305-359, 2016.

[3] M. Fisher, L. Dennis, and M. Webster. Verifying Autonomous Systems. Comm. ACM, 56(9):84-93, 2013.

[4] F. Raimondi and A. Lomuscio. Automatic Verification of Multi-agent Systems by Model Checking via Ordered Binary Decision Diagrams. J. Applied Logic, 5(2):235-251, 2007.

Author. Michael Fisher is a Professor of Computer Science and Director of the multi-disciplinary Centre for Autonomous Systems Technology at the University of Liverpool[2]. His research concerns formal verification for the certification, safety, ethics, and reliability of autonomous systems. He is a Fellow of both the BCS and the IET, is on the editorial boards of both the J. Applied Logic and Annals of Mathematics and Artificial Intelligence, is a corner editor for the J. Logic & Computation, and is a member of the British Standards Institution committee on Robots and Robotic Devices.

[2] www.liv.ac.uk/cast

Photographs of FACS evening seminars, 2016

Jonathan Bowen

Jan Tretmans, 17 May 2016

Ana Cavalcanti, 29 September 2016

Muffy Calder, with John Cooke,
3 November 2016

Joe Stoy, 15 November 2016

Strachey 100 Centenary Conference

Photographs of Strachey 100

Department of Computer Science, University of Oxford
18-19 November 2016

Jonathan Bowen
Chair, BCS-FACS Specialist Group

Christopher Strachey (1916-1975) was a pioneering computer scientist and the founder of the Programming Research Group, now part of the Department of Computer Science at Oxford University. Although Strachey was keenly interested in the practical aspects of computing, it is in the theoretical side that he most indelibly left his mark, notably by creating with Dana Scott the denotational (or as he called it, 'mathematical') approach to defining the semantics of programming languages. Strachey also spent time writing complex programs and puzzles for various computers, such as a draughts playing program for the Alan Turing's Pilot ACE in 1951. He developed some fundamental concepts of machine-independent operating systems, including an early suggestion for time-sharing, and was a prime mover in the influential CPL programming language. Strachey came from a notable family of intellectuals and artists, perhaps most famous for Christopher's uncle Lytton, a writer and member of the Bloomsbury group.

The occasion of a hundred years since Christopher Strachey's birth on 16 November 1916, was marked three days after his birthday, with a symposium of invited speakers. The morning looked back at Strachey's life and works from a historical and technical perspective and the afternoon concerned continuing research themes in computer science inspired by Strachey, at Oxford and elsewhere. There was also be a display of related archival material at the Weston Library, part of the Bodleian Library, Oxford University's main library, on the afternoon before the conference and a dinner was held at Hertford College during the evening. The following is a selection of photographs taken during the event.

The above is adapted from the Strachey 100 website under
https://www.cs.ox.ac.uk/strachey100/

The event was organized by Cliff Jones and Troy Astarte (Newcastle University) and Samson Abramsky, Bernard Sufrin, Alex Kavvos, and Karen Barnes (Oxford University).

Note: Peter Landin (1930–2009), after whom the BCS-FACS Annual Peter Landin Semantics Seminar is named, was Strachey's assistant from 1960 to 1964, when Strachey was an independent computer consultant in London.

Friday, 18 November 2016

Strachey papers, Weston Library,
Bodleian Library, University of Oxford

Peter Mosses, Joe Stoy, Cliff Jones, Samson Abramsky, and Martin Campbell-Kelly

Samson Abramsky and Joe Stoy

Saturday, 19 November 2016

Department of Computer Science, University of Oxford

Morning: Historical Talks

Martin Cambell-Kelly, Cliff Jones, Samson Abramsky, and Troy Astarte

Chair: Cliff Jones

Michael Wooldridge: Introduction

Michael Wooldridge (Head of Department, Department of Computer Science, University of Oxford)

Martin Campbell-Kelly: "Strachey: the Bloomsbury Years"

Martin Campbell-Kelly (historian of computing)

Joe Stoy: "Strachey and the Oxford Programming Research Group"

Joe Stoy (Oxford colleague of Strachey)

Peter Mosses: "SIS, a semantics implementation system"

Peter Mosses (doctoral student of Strachey)

Martin Richards: "Strachey and the development of CPL"

Martin Richards
(Cambridge colleague of Strachey)

Robert Milne: "Semantic relationships: reducing the separation between practice and theory"

Robert Milne (colleague of Strachey)

Break

Panel: Roger Penrose, David Hartley, Michael Jackson. Chair: Bernard Sufrin

Bernard Sufrin (chair)

Penrose, Hartley, Sufrin, and Jackson

Bernard Sufrin (chair) and Michael Jackson
(taught by Strachey at Harrow School)

Roger Penrose (Strachey family friend)
and David Hartley
(Cambridge colleague of Strachey)

Jill and Tony Hoare (who took over as head
of the Programming Research Group
at Oxford from Strachey in 1975)
with others in the audience

Afternoon: Forward-Looking Session

Philip Wadler, Samson Abramsky, Alex Kavvos,
Troy Astarte, and Jane Hillston

Chair: Samson Abramsky

Dana Scott (address read by Joe Stoy)

Hongseok Yang: "Probabilistic Programming"

Jane Hillston: "A modelling language approach to defining mathematical structures via semantics"

Uday Reddy: "Parametric Polymorphism and models of storage"

Philip Wadler: "Christopher Strachey, First-Class Citizen"

David Turner (doctoral student of Strachey) and others in the audience

Jeremy Gibbons: "What are types for?"

Robert Milne, Joe Stoy, Samson Abramsky, and Christopher Wadsworth (doctoral student of Strachey)

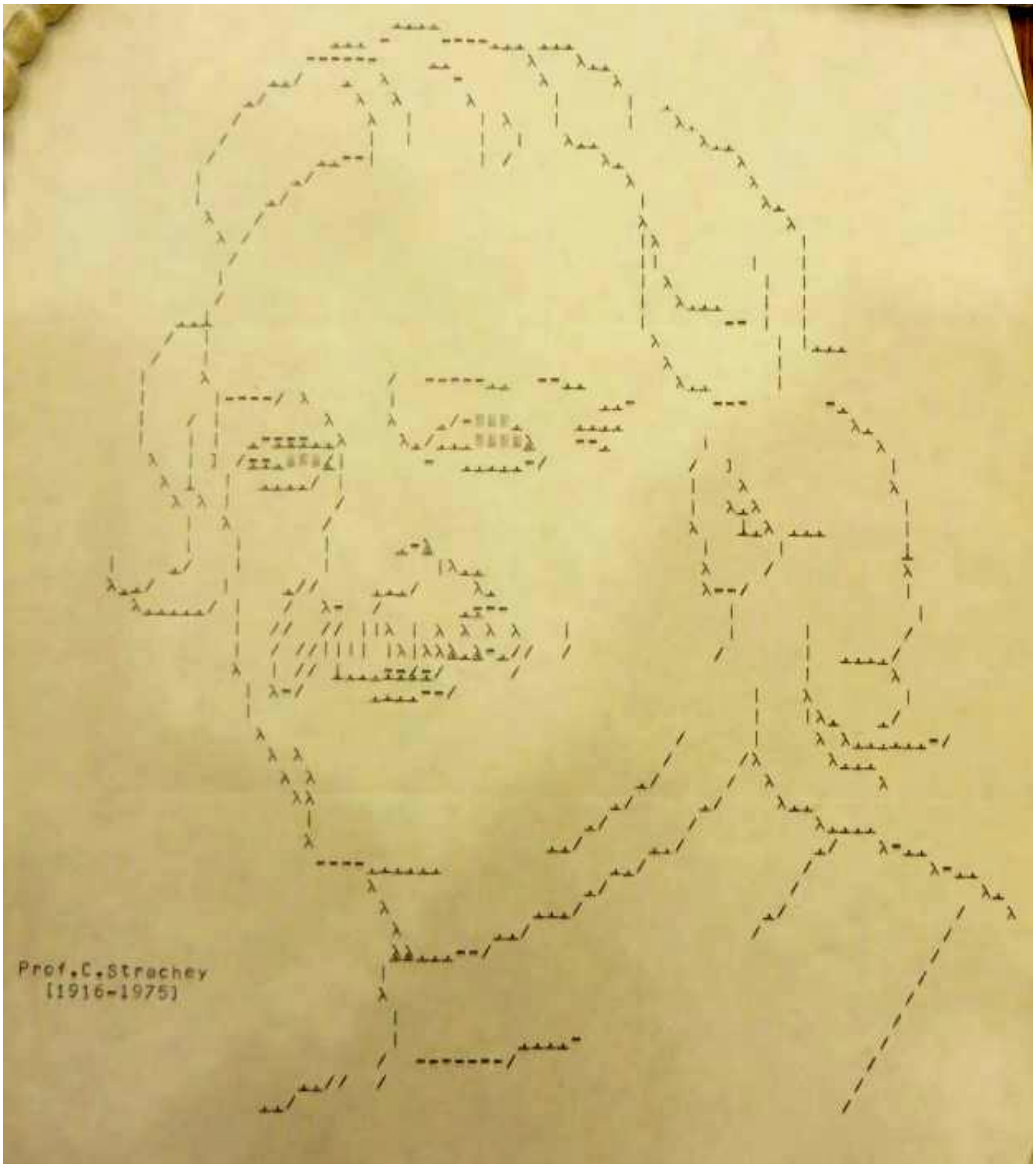
End

For further *Strachey 100* information, see: https://www.cs.ox.ac.uk/strachey100/

For information on Strachey himself, see: https://en.wikipedia.org/wiki/Christopher_Strachey

For Strachey's doctoral students, see: http://www.genealogy.ams.org/id.php?id=75007

See also:
Campbell-Kelly, M. (1985) "Christopher Strachey, 1916–1975: A Biographical Note". *IEEE Annals of the History of Computing,* 7(1):19–42. DOI: 10.1109/MAHC.1985.10001


Prof.C.Strachey
[1916-1975]

FACS Committee

Formal Aspects of Computing Science Specialist Group

Jonathan Bowen
FACS Chair; BCS Liaison

Jawed Siddiqi
FACS Treasurer

Paul Boca
FACS Secretary

Roger Carsley
Minutes Secretary

John Cooke
LMS Liaison

Ana Cavalcanti
FME Liaison

Margaret West
BCS Women Liaison

Rob Hierons
Chair, Testing Subgroup

John Derrick
Chair, Refinement Subgroup

Eerke Boiten
Chair, Cyber Security Subgroup

Sofia Meacham
Meetings Coordinator

Mike Hinchey
International Coordinator

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.

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.