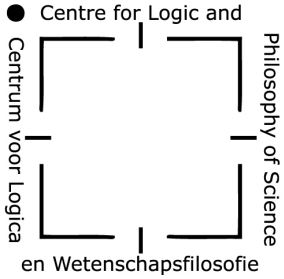
HAPOC11

History and Philosophy of Computing

07–10 November 2011
Ghent, Belgium

Het Pand

Organised by
Centre for Logic and Philosophy of Science
Ghent University



Conference Aims and Scope

Calculemus! Let us calculate! These are Leibniz' famous words. He would probably not have imagined that about three centuries later almost anybody would actually rely on computations in his/her everyday life.

Especially since the development of the so-called personal computer in the '80s, computing has penetrated the several levels of our society, going from scientific research to the organization of our social lives. Despite their widespread application, the computing sciences remain hidden behind layers of so-called user-friendly interfaces and require specialized knowledge.

The number of researchers working in fields related to computing is growing rapidly in many different directions. As Mahoney once stated, "the computer is not one thing but many different things, and the same holds true of computing". As a consequence, the computing sciences collect the most diverse complex of experts: philosophers, logicians, historians, mathematicians, computer scientists, programmers, engineers. The number of involved subjects grows accordingly: from the foundational issues to their applications; from the philosophical questions to problems of realizability and design of specifications; from the theoretical studies of computational barriers to the relevance of machines for educational purposes.

Given the significance of computing for modern society, the relevance of its history and philosophy can hardly be overestimated. Both the history and philosophy of computing only started to develop as real disciplines in the '80s and '90s of the previous century, with the foundation of journals (e.g. the IEEE Annals on the History of Computing, Minds and Machines and the like) and associations (SIGCIS, CAP, . . .), and the organization of conferences and workshops on a regular basis. A historical awareness of the evolution of computing not only helps to clarify the complex structure of the computing sciences, but it also provides an insight in what computing was, is and maybe could be in the future. Philosophy, on the other hand, helps to tackle some of the fundamental problems of computing, going from the limits of the "mathematicizing power of homo sapiens" to the design of feasible and concrete models of inter-

active processes. The aim of this conference is to bring together these two streams: we are strongly convinced that an interplay between the researchers with an interest in the history and philosophy of computing can crucially add to the maturity of the field.

Chairs

Liesbeth De Mol (Ghent)
Giuseppe Primiero (Ghent)

Organising Committee

Liesbeth De Mol (Ghent)
Giuseppe Primiero (Ghent)
Dagmar Provijn (Ghent)
Jean Paul Van Bendegem (Brussels)

Scientific and Programme Committee

Gerard Alberts (Amsterdam)
Sergei Artemov (New York)
Martin Campbell-Kelly (Warwick)
Leo Corry (Tel Aviv)
Liesbeth De Mol (Ghent)
Marc Denecker (Leuven)
Amnon Eden (Essex)
Luciano Floridi (Oxford & Hertfordshire)
Reinhard Kahle (Lisbon)
Benedikt Loewe (Amsterdam)
Joke Meheus (Ghent)
Erik Myin (Antwerp)
Sara Negri (Helsinki)
Valeria de Paiva (Cupertino)
Giuseppe Primiero (Ghent)
Dagmar Provijn (Ghent)
Sonja Smets (Groningen)
Göran Sundholm (Leiden)
Carlo Toffalori (Camerino)
Jean Paul Van Bendegem (Brussels)
Maarten Van Dyck (Ghent)
Bart Van Kerkhove (Hasselt)
Erik Weber (Ghent)

Organization and Support

HAPOC11 is supported by

Faculty of Arts and Philosophy, Ghent University

Fonds Wetenschappelijk Onderzoek - Vlaanderen (FWO)
http://www.fwo.be/

The International Association for Computing and Philosophy (IACAP)
http://www.iacap.org/

The Association for Symbolic Logic (ASL)
http://www.aslonline.org/

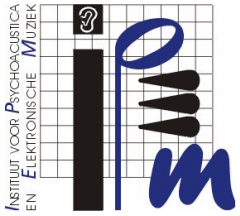
Computability in Europe Association (CiE)
http://www.computability.org.uk/

Associazione Italiana di Logica e Applicazioni (AILA)
http://www.ailalogica.it/

The Belgian Society for Logic and Philosophy of Science
http://www.bslps.be/

Centre for History of Science - Ghent University
http://www.sarton.ugent.be/

Institute for Psychoacoustic and Electronic Music - Ghent University (IPEM)
http://www.ipem.ugent.be/



Schedule

MONDAY 07 NOVEMBER

08:30 – 09:30 Registration and Coffee
09:30 – 10:00 Opening: Liesbeth de Mol & Giuseppe Primiero

10:00 – 11:00 Invited Lecture
William Aspray – *Three topics in the History of Computing*
chair: Gerard Alberts

11:00 – 11:30 Coffee break

Contributed papers
chair: Maarten Bullynck
11:30 – 12:00 Valery Shilov & Vladimir Kitov – *Mechanical brain in the XIX century: Logical machines of Alfred Smee*
12:00 – 12:30 Izabela Bondecka-Krzykowska – *First calculating machines in Poland*
12:30 – 13:00 Marie D'Udekem-Gevers – *A long history of automation : from its origins to the computer*

13:00 – 14:30 Lunch

Contributed papers
chair: Teresa Numerico
14:30 – 15:00 Julian Wilson – *The Max Newman Collection of Alan Turing's Offprints: a bibliographical enquiry.*
15:00 – 15:30 Guido Gherardi – *Alan Turing and the foundations of computable analysis*
15:30 – 16:00 Ofra Rechter & Eli Dresner – *From Symbol to 'Symbol': Turing, Hilbert and the Quasi-concreteness of Signs*

16:00 – 16:30 Coffee Break

Contributed papers
chair: Erik Myin
16:30 – 17:00 Rogier De Langhe – *Simulation and theory choice*
17:00 – 17:30 Philip Nickel – *Artificial Testimony*

17:30 – 18:00 Coffee Break

18:30 – 19:30 Invited Lecture
Stephen Wolfram – *Making the World Computable*
chair: Martin Davis

19:30 – 20:30 Drinks

TUESDAY 08 NOVEMBER

09:00 – 10:00 Invited Lecture
Martin Davis – *Universality is Ubiquitous*
chair: Liesbeth De Mol

10:00 – 10:30 Coffee Break

Contributed papers
chair: Marie D'Udekem-Gevers
10:30 – 11:00 Sten Henriksson – *A History of the Stack*
11:00 – 11:30 Helena Durnova – *Language for algorithms, or algorithmic language?*
11:30 – 12:00 Maarten Bullynck – *Computation/Communication. A parallel glance on the transformations of the computer.*

12:00 – 13:30 Lunch

13:30 – 14:30 Invited Lecture
Fairouz Kamareddine – *From the Foundation of Mathematics to the Birth of Computation*
chair: Raymond Turner

14:30 – 15:00 Coffee Break

Contributed papers
chair: Jean-Paul van Bendegem
15:00 – 15:30 Sam Sanders – *Computing the Infinite*
15:30 – 16:00 Anthony Moore & Kevin Kirby – *Reimagining Time in Computing: Reservoirs and Aural Arithmetics*
16:00 – 16:30 Duilio D'Alfonso – *Kolmogorov Complexity and Information Theory: The meaning of being minimal*

16:30 – 17:00 Coffee Break

Contributed papers
chair: Dagmar Provijn
17:00 – 17:30 Teresa Numerico – *The computer between Computationalism and Cybernetics: the crucial role of Turing and von Neumann, and why they were ignored*
17:30 – 18:00 John Geske – *From the Church-Turing Thesis to the Triumph of the Von-Neumann Architecture: The Serialization of Philosophic Thought in Computer Science*

19:00 – 20:00 Lecture/Performance
Co-organised with IPEM

WEDNESDAY 09 NOVEMBER

09:00 – 10:00 Invited Lecture
Sybille Krämer – *Mathematizing power, formalization and the diagrammatical mind, or: What does Computation mean?*
chair: Benedikt Löwe

10:00 – 10:30 Coffee Break

Contributed papers
chair: Maarten van Dyck
10:30 – 11:00 Wolfgang Brand – *Models, Experiments and Computing: A Historical Case Study of the Design of the Membrane Roof of the Munich Olympic Stadium using the first Supercomputers*
11:00 – 11:30 Giuditta Parolini – *Making and Remaking the Statistical Tables for Biological, Agricultural and Medical Research*
11:30 – 12:00 Christopher Belanger – *Chaos, Prediction, and Computation: Rehabilitating Laplacean Determinism*

12:00 – 13:30 Lunch

13:30 – 14:30 Invited Lecture
Giovanni Sambin – *Computability without Turing Machines*
chair: Fairouz Kamareddine

14:30 – 15:00 Coffee Break

Contributed papers
chair: Liesbeth de Mol
15:00 – 15:30 Pierre Mounier-Kuhn – *Computer science in France: a controversial emergence*
15:30 – 16:00 Viola Schiaffonati and Mario Verdicchio – *Is Computer Science Made Scientific by its Experiments?*
16:00 – 16:30 Raffaele Mascella – *Programming languages as a revealing enterprise in computer science*

16:30 – 17:00 Coffee Break

Contributed papers
chair: Peter Verdée
17:00 – 17:30 Francisco Hernández-Quiroz – *Logics of programs as a fuelling force for semantics*
17:30 – 18:00 Walter Dean – *On models of computation and the analysis of feasibility*

19:00 – 22:00 Conference Dinner

THURSDAY 10 NOVEMBER

10:00 – 11:00 Invited Lecture
Raymond Turner – *Towards a Philosophy of Computing Science*
chair: Giovanni Sambin

11:00 – 11:30 Coffee Break

Contributed papers
chair: Giuseppe Primiero
11:30 – 12:00 Joscha Bach – *No Room for the Mind: Enactivism in Artificial Intelligence*
12:00 – 12:30 Giovanni Camardi – *Computation, Information and Computer Simulations*
12:30 – 13:00 Federico Gobbo & Marco Benini – *From Computing Machineries to Cloud Computing: The Minimal Levels of Abstraction of Inforgs through History*

13:00 – 13:30 Closing

List of Invited Talks

Three Topics in the History of Computing

William Aspray
School of Information, University of Texas, Austin, US
bill@ischool.utexas.edu

This paper addresses three historiographic issues related to computing. The first topic concerns the pre-history and post-history of the development of the concept of computability and of recursive function theory. This section examines little studied antecedents in mathematics and philosophy to the development of mathematical constructivity, and advocates the application to the study of theoretical computer science an historical approach concerning scholarly communities and intellectual agendas first proposed by the late Michael Mahoney. The second topic concerns the concepts of information domain, information business, and information society. This section reintroduces a simple analytical mapping device introduced in the 1980s but largely forgotten today, and it suggests how historians of computing could profit from being more familiar with related work by historians of libraries and museums as well as by critical theorists. The third topic concerns the use of history in the study of everyday information seeking behavior. It suggests how historians of computing can broaden and enrich their work through a critical examination of the role of the Internet in everyday life, informed by work of information studies scholars, sociologists, and phenomenologists.

Universality is Ubiquitous

Martin Davis
Courant Institute, NYU
Visiting Scholar, UC Berkeley
martin@eipye.com

The work of Turing, Post, Church, Gödel, and Kleene during the 1930s fundamentally altered our notion of the nature of computation. I will discuss this in terms of the theoretical underpinnings of the development of all-purpose computers and of modern computer science. I will go on to speculate about the role of computation in the human mind and in biological evolution.

From the foundation of mathematics to the birth of computation

Fairouz Kamareddine
School of Mathematical and Computer Sciences, Heriot-Watt University, Edinburgh, UK
fairouz@macs.hw.ac.uk

Mathematics is old, Logic is old, but in some sense, many basic ideas of the foundation of computer science are old too. For example, that proof checking (or type checking) is decidable but proof construction (or type inference) is not, was hinted at by Aristotle. In this talk, I go through some of the developments in mathematics and logic which influenced the creation of some computer science ideas in the 20th century. In particular, I discuss how the need for more precision and formality in the 18th century, led to the development of logic in the 19th century, to the work of Frege and the discovery of Russell's paradox. I then discuss the use of type theory (a concept that was already implicit in Euclid's Geometry 325 B.C.) by Russell to avoid the paradox and explain the development and the influence of types in computation.

Mathematicizing power, formalization, and the diagrammatical mind or: What does 'computation' mean?

Sybille Krämer
Institut für Philosophie, Freie Universität Berlin, Germany
sybkram@zedat.fu-berlin.de

In order to understand 'computation', we must clarify three of its essential aspects:

(i) In computation we do not operate directly with numbers or quantities, but rather with symbols or signs, and we do so in a manner dependent upon rules which do not themselves refer to the meaning of those signs. This is the guiding idea behind 'formalization', first developed by Gottfried Wilhelm Leibniz: Mind can be performed without interpretation and consciousness.

(ii) This formalization, which is crucial to computation, is a modality of visualization, an 'imaging procedure' grounded in the interaction of hand and eye. Algebraic operations with characters depend upon perception just as much as geometric operations with figures. Mathematics is impossible without intuition, and therefore also impossible without the use of per-

ceptible forms of graphism such as signs and figures. This insight can be found in Plato, Descartes and Wittgenstein.

(iii) Computation is an activity of the 'diagrammatical mind'. Diagrammatical graphism emerges through the interaction of surface, line, and point. Notations, graphs, diagrams, and maps belong to the realm of diagrammatics. By 'diagrammatics' we mean: by aid of spatial relations non-spatial epistemic connections are not only depicted but explored, generated and even constituted. These ideas are developed in the work of Lambert, Kant and Peirce.

The concept of 'computation' can be characterized by the 'locus', where (i) symbolism, (ii) visualization and (iii) diagrammatics intersect.

Computability without Turing Machines

Giovanni Sambin
Dipartimento di Matematica Pura e Applicata, Universitá di Padova, Italy
sambin@math.unipd.it

Mathematics is born by abstraction from reality. In a dynamic view, the choice of a specific foundation of mathematics amounts to a decision of what kind of information is considered relevant in the process of abstraction. In this sense, all of mathematics, and constructive mathematics especially, is linked with information science.

I will report on a 25-year-long experience in developing constructive (by which I mean both intuitionistic and predicative) topology. I will show in particular in which precise sense it can be conceived as including an abstract theory of computation, without Turing machines and codings. A crucial role is played by our choice for a minimalist foundation (introduced by Milly Maietti and myself), whose main novelty is the presence of two different levels of abstraction (one for computations and one for mathematics) and of their interaction.

I will also argue that the study of interactions between different levels, or logical types, in general could be the road leading us to computability beyond Turing machines.

Towards a Philosophy of Computing Science

Raymond Turner
School of Computer Science and Electronic Engineering, University of Essex, UK
turnr@essex.ac.uk

The Philosophy of Computer Science is concerned with philosophical issues that arise from reflection upon the nature and practice of the academic discipline of Computer Science. In this talk we shall consider a group of interrelated questions that emanate from the theoretical end of the subject. More explicitly:

I. How is a programming language determined? What is the appropriate role of formal semantics?

II. Are programming languages mathematical objects? What is the ontological status of programs? Are they abstract objects?

III. What is the logical substance of specification? Do specifications act like definitions in mathematics, or is something else going on?

IV. Does the notion of correctness relative to a specification depend upon the nature of the artifact? More specifically, is it different for abstract and concrete artifacts?

V. What is the role of types in computer science?

VI. What is abstraction in computer science? How is it related to abstraction in mathematics?

VII. Is computational thinking just a restricted form of mathematical thinking? While it may be the case that some of these questions, when unpacked, are neither substantial nor even well formulated, finding this out is part of the philosophical task.

Making the World Computable

Stephen Wolfram
Wolfram Research
s.wolfram@wolfram.com

Evening lecture through video conferencing.

Special Event

On Tuesday, November 8 an evening lecture followed by two demonstrations organised in collaboration with IPEM.

Computing embodied experiences with Music

Prof. dr. Marc Leman, Pieter-Jan Maes, Luc Nijs
IPEM, Institute for Psychoacoustics and Electronic Music, Ghent University
`http://www.ipem.ugent.be/`

Recent brain research provides evidence that music modifies the brain on the basis of a coupled action-perception (or hearing-doing) system. The work of my team so far been pushing the frontiers of musical action-perception research by linking the basic concepts of 'musical gesture' and 'music mediators' to computational environments. We focus on two types of music mediators:

1. the human body as the natural mediator, and
2. technology as the artificial mediator, where the latter is seen as an extension of the human body.

Mediators allow the human mind to construct new types of realities in which new meaning formation is possible. Using computational tools, we aim at extending these realities. This approach will be illustrated by means of two demonstrators. A first demonstrator is the Music Paint Machine. This is a system that translates sound and movement of a musician into visuals. The second demonstrator is the Conduction Master. This is a system that hooks into the action-perception loop such that the loop can be extended and modified. The audience can test the demonstrators.

Abstracts of Contributed Talks

No Room for the Mind: Enactivism in Artificial Intelligence

Joscha Bach
Berlin School of Mind and Brain
Humboldt-University of Berlin
joscha.bach@gmail.com

The philosophical tenet of enactivism has recently risen in popularity; it has even gained considerable traction within cognitive science in general, and Artificial Intelligence in particular (e.g., [13]). It is often associated with notions of an extended mind [5] and embodied cognition; however, in its radical form, it goes beyond these concepts. At enactivism's core lies the notion of an inextricable connection between mind and environment. The various paradigms in psychology, cognitive science or artificial intelligence (including the extended mind) each posit different interfaces between mind and world (respectively provided by nerve endings, senses, sensory-motor interaction, subcortical brain structures, qualia, and so on) and consequently arrive at slightly different (but not necessarily incompatible) conceptualizations of what constitutes a mind, mental activity, and interaction with the world. Radical enactivism is different: it eschews such an interface, and instead suggests that minds supervene over the interaction of an agent's body with its environment. It denies the role of internal representations (knowledge, thoughts, dreams and so on) and instead relocates cognition from the brain's information processing into the world.

Enactivism rejects the idea that the mind is reducible to a formal (i.e., ultimately computational) model, which puts it closer to phenomenalist traditions in philosophy than to the more widespread "mind as machine" paradigm of cognitive science [4]. Enactivism's growing traction in cognitive science can only be explained by a coincidence: Computational modeling of the mind has experienced a change in focus during the last 15 years, from classical artificial intelligence (AI) architectures towards robotics, reflecting a growing research interest into the effects of situatedness, interactivity and affordances [9] on cognition. The proponents of embodied systems hope to overcome the problem of the old generation of expert systems (see, for instance, [2, 7]) by replacing manually entered, rule-based knowledge with situated, autonomous exploration of the world, driven by the artificial agent itself. This alone does not entail a paradigmatic shift in the philosophy of AI. Embodied agents may be simply proposed as an engineering

solution to the problem of knowledge acquisition, while the mind is still considered to be supervenient over the computer's (or brain's) information processing. Embodied cognition may easily be taken one step further, by integrating the tool-use and immediate environment in a model of agency. This notion of extended mind may posit a new research paradigm, but it does not present any challenge to the idea of AI itself. Enactivism, however, is a much more radical idea than extending agency and intentionality beyond an agent's skin. Moderate proponents of an extended mind will grant that the bulk of cognition takes place within the agent itself, and involves the representations formed about the external and internal environment of the agent (even if the agent may 'outsource' a part of the processing, and of the representations to its surroundings). According to enactivism, the mind does not supervene over the activity of its substrate (i.e., the nervous system, the body and perhaps a slice of the world), but over the interaction between body and physical reality. Many researchers in cognitive robotics fall prey to a misunderstanding: they take enactivism's account of embodied cognition to be a philosophical manifesto for exploring autonomous agents in richly structured environments [1, 12], and require physical scenarios instead of simulations because of the well-known difficulties of adding the required depth to simulated worlds. Unfortunately, this is not the case, because AI and enactivism are fundamentally incompatible. At the heart of this incompatibility lies an epistemological problem: Robots and situated AI architectures interact with their environment through an opaque interface that makes it impossible for the agent to discern the 'reality' behind the data patterns that are constitutive to that interaction. No robot can know whether its physical environment has been replaced a simulation, or its simulation by another simulation, as long as the structural properties of the data at its interface are the same. To an AI researcher, the world is simply a provider of richly structured information, but to an enactivist, it is a directly accessible reality, beyond information. According to enactivism, no AI researcher can ever hope to build an information processing system (i.e., a robot or an AI architecture) with a true mind, because minds require something that reaches beyond any information processing, can not be expressed by formal theories and computational models, will never be subjugated by algorithms and data structures. Minds require the direct touch with the essence of reality itself, and according to enactivism, the physical reality cannot be reduced to information processing [3, 6]. There seem to be important insights taught by enactivism to robotics, especially that it is possible for a system to possess skills without representing them informationally. These 'embodied' skills, as demonstrated for instance in passive walkers [11], seem to demonstrate a paradigm that extends into higher level cognition, without ever losing sight of the problem of grounding [10]. However, these skills are only superficially more complex than for, for instance, the 'knowledge' that a stone needs to fall in a ballistic arc. It does not exhibit features of higher-level control, and it cannot do so, because this would require internal states. Cognition is not an elaboration of a mechanical reflex.

As I will discuss in more detail, enactivist embodiment does not capture cognitive processing – instead, it diminishes cognition to a mindless embodimentalism. I am going to argue that a detailed response to enactivism appears to be worthwhile, because it will force us to clarify and reflect upon the epistemological assumptions behind AI and the implied relationship between reality and computation.

References

[1] Barsalou, L. W., (2008). 'Grounded Cognition.' *Annual Review of Psychology*, 59(6), 617–645.

[2] Barsalou, L. W., (1999). 'Perceptual Symbol Systems' *Behavioral and Brain Sciences*, vol. 22, 577–660.

[3] Bateson, G., (1972). 'Steps to an Ecology of Mind: Collected Essays in Anthropology, Psychiatry, Evolution, and Epistemology' University Of Chicago Press.

[4] Boden, M. (2006).'Mind as Machine: A History of Cognitive Science', Oxford University Press.

[5] Clark, A., (2008).'Supersizing the Mind: Embodiment, Action, and Cognitive Extension', Oxford University Press.

[6] Di Paolo, E. A. (2010). 'Introductory chapter to Enaction: Toward a New Paradigm for Cognitive Science', by Stewart, J., Di Paolo, E. A., Gapenne, O. (eds.). MIT Press, Cambridge, MA, vii–xvii.

[7] Dreyfus, H. (1979). 'What Computers can't do'. New York, MIT Press.

[8] Floridi, L. (2011). 'The Philosophy of Information', Oxford University Press.

[9] Gibson, J. J. (1977), 'The theory of affordances' In R. E. Shaw & J. Bransford (Eds.), *Perceiving, Acting, and Knowing*. Hillsdale, NJ: Lawrence Erlbaum Associates.

[10] Harnad, S. (1990). 'The symbol grounding problem' *Physica D*, vol. 42, 335–346.

[11] McGeer, T. (April 1990). 'Passive dynamic walking.' *International Journal of Robotics Research* April 1990.

[12] Pecher, D., Zwaan, R.a. (2005). 'Grounding Cognition: The Role of Perception and Action in Memory, Language and Thinking.' Cambridge University Press.

[13] Pfeifer, R., Bongard, J. C. (2006). 'How the Body Shapes the Way We Think. A New View of Intelligence.' Cambridge: MIT Press 2006.

First calculating machines in Poland

Izabela Bondecka–Krzykowska
Adam Mickiewicz University
Department of Mathematics and Computer Science
Poznań, Poland
izab@amu.edu.pl

The history of mechanical computation is long and interesting, but very often this story is limited to inventors from Western Europe: Schickard, Pascal, Leibniz and their successors. The aim of this paper is to present the calculating machines built in Poland (Eastern Europe) which were known in contemporary Europe. The machines were constructed in 19^{th} century by three inventors: Abraham Stern, Chaim Słonimski and Izrael Staffel. All of them spent most of their life in Warsaw (the present capital of Poland) which at that time was part of the Russian Empire.

Abraham Stern was as a splendid inventor. He presented his calculating machines a couple of times at the meetings of the Royal Warsaw Society of the Friends of Science (predecessor of the Polish Academy of Science). His first machine for four arithmetic operations only was presented in December 1812, the second machine for extracting square roots in January 1817 and finally the combined machine for four operations and square roots in April 1818. Stern's machines were highly valued but they were never manufactured, maybe because of its intricate mechanism which resulted in the high costs of production. But in 1844 a well-known Swedish producer of arythmometers Willgodt T. Odhner familiarized with Stern's inventions and utilized it later in his own constructions which were mass-produced.

Chaim Słonimski was a talented inventor. Among many Słonimski's inventions calculating machines were worth noting. He invented and produced two calculating machines, one for addition and subtraction, and the other one for multiplication. The most interesting is the other one – a simple device, whose construction was based on a theorem in number theory. This theorem, named after its inventor, enabled Słonimski to arrange the table of numbers, which was the base of construction for the calculating machine. Słonimski's machine was a box of the size: $40cm \times 33cm \times 5cm$. It was rather simple to use it. The multiplicand was set on the lowermost row of apertures with handles mounted on the cover. After operation the products of all ranks were displayed in rows of apertures $4^{th} - -11^{th}$: in the 4^{th} row there was the product of multiplication by 2, the 5^{th} row by 3, the 6^{th} row by 4 etc. Thanks to the usage of the mentioned theorem Słonimski's machine had a very simple construction and was cheap. At that time existed only a few calculating machines which were based on such a

good theoretical background. Unfortunately the machine did not survive to our times.

Another machine which did not survive to the present day is an invention of clockmaker Abraham Staffel. He designed and built in 1845 a calculating machine for four basic arithmetical operations, exponentiation and extracting square roots. The modern (in those days) construction of the machine enabled performing not only simple calculation but also calculating more complicated expressions like this:

$$\frac{a+b+c-d-e+(g\times h)-m^2}{n}$$

The machine of Abraham Staffel was presented at least at three exhibitions. At the exhibition in Warsaw in 1845 Staffel received silver medal for his invention. In 1846 the machine was presented to Russian Academy of Sciences in St. Petersburg. Two famous mathematicians, V. Bunyakovski and B. Jacobi, gave it a very positive opinion and Staffel was awarded a Demidov prize (amounting of 1500 rubles). The machine was also presented at The Great Exhibition in London in 1851 in one group with arithmometer of Xavier Thomas de Colmar. Two machines were awarded: Staffel's and Colmar's.

At the end of his life Staffel handed over his invention to the Russian Academy of Sciences. After the collapsing of the tsarism the collection of Academy broke down. Probably the Staffel's machine was destroyed then and did not survive to our times.

Stern's, Słonimski's and Staffel's inventions were not the oldest constructions of such type in Europe, but were equal to calculating machines produced in Western Europe in those days. The experts assessed Polish constructions as very good and rewarded them many times during the international exhibitions. So the "eastern thread" of the history of mechanical computing in Europe is worth talking about.

References

[1] Apokin I.A., (2002), 'J. Jakobson's Machine', in *Computing in Russia*, G. Trogemann, W. Ernst, A.Y. Nitussov (Eds.), Vieweg, Stuttgart, 27-29.

[2] Apokin I. A., (2002), 'The Slonimski Theorem and Its Implementation in Simple Multiplication Devices', in *Computing in Russia*, G. Trogemann, W. Ernst, A.Y. Nitussov (Eds.), Vieweg, 2002, 29-31.

[3] Crelle A.L., (1844), 'Démonstration d'un théoréme de Mr. Slonimsky sur le nombres, avec une application de ce théoréme au calcul de chiffres', *Journal für die reine und angewandte Mathematik*, Vol. 28, 184-190.

[4] Detlefsen M., (1976), 'Polnische Rechenmaschinenerfinder des 19. Jahrhunderts', *Wissenschaft und Fortschritt*, Vol. 26, No. 2, 86-90.

[5] Knight H., (1847), *Multiplication Tablets Derived from a Theorem of S. Slonimski*, Birmingham.

[6] 'New Calculating Machine', *Scientific American*, August 23, 1851, Vol. 6, No. 49, 392.

[7] 'Selig Slonimski und sein Recheninstrument', *Illustrierte Zeitung*, 1845 Vol. 5, No. 110, 90-92.

[8] Slonimsky Ch.Z., (1846), 'Allgemeine Bemerkungen über Rechenmaschinen und Prospectus eines neu erfundenen Rechen-Instrumente', *Journal für die reine und angewandte Mathematik*, Vol. 30, 215-229.

[9] 'Staffel's Calculating Machine', *The Illustrated London News*, Exhibition Supplement, Vol. 19, No. 518, September 30, 1851.

Models, Experiments and Computing
A Historical Case Study of the Design of the Membrane Roof of the Munich Olympic Stadium using the first Supercomputers

Wolfgang Brand
University of Stuttgart
Abteilung Geschichte der Naturwissenschaften und Technik
wolfgang.brand@studenten.ims.uni-stuttgart.de

For centuries, the design and planning process in architecture was based on building models and conducting experiments with them. Lacking computing devices and limited capabilities of the early computing machinery available, prevented widespread use of numerical computations and simulations in architecture and civil engineering. The advent of the electronic high speed computer during the late 1940s and early 1950s allowed architects and civil engineers for the first time to study their designs using numerical methods.

The design of the membrane roof of the stadium for the 1972 Olympic Games in Munich was one of the first major projects in which numerical simulations played a crucial role. The architect Frei Otto - head of the Institute for Lightweight Structures at the University of Stuttgart - had been working and experimenting for many years with lightweight tensile and membrane structures, space frames and their structural efficiency. Otto's design ideas where transformed by Gnter

Behnisch and Jörg Schlaich into a piece of landmark membrane architecture of glass and steel.

This case study will focus on the transition period where the architects started to abandon their physical models for evaluating constructional aspects in favor of computational models. The Munich Olympic Stadium's tent-shaped roof acts as the showcase on how numerical computations and modeling were established as a part of the architectural design process.

Machines, such as Control Data's CDC 6600, provided sufficient computing power to civil engineers and architects to build large-scale, lightweight and naturally shaped structures, such as the membrane roof of the Munich Olympic Stadium. Model building and experimentation as key elements of the design process where augmented and even substituted by numerical computations and simulations. The availability of substantial amounts of computing power for non-military purposes at university laboratories opened the gates to designs using novel shapes and materials.

However, simply transferring the physical model into a computer model wasn't sufficient. The Munich Olympic Stadium's roof is also an example of new players coming to the game, who contributed the knowledge from their fields to the design process. The design of the roof required the deployment of methods from geodesy to determine the optimal shape and inner structure of the construction.

Other theoretical advances in numerical mathematics and algorithms, such as the Finite Elements Method (FEM), co-invented by John Argyris at Stuttgart University during the early 1960s, formed another pillar on which progress in structural design rested. Architects, engineers and mathematicians were members of a heterogeneous group who were able to tackle such ambitious new projects. The genesis and networking of this group is studied and illustrated.

After computers started to make their ways into science and engineering, there was also a lively and controversial debate on the relations between experiment and simulation in science, engineering and architecture. One central question was - and is - whether and how numerical results reflect reality and how they might be verified and validated. This study illustrates the developments in this transitional period of the 1960s and early 1970s regarding this topic, too.

After powerful numerical simulation tools, i.e. computers, became available, there were two groups discriminated by their attitude towards this new approach: Those who preferred the traditional model building and experiment over computation and those who readily took up the new capabilities to implement their dreams of novel architectural structures. This can be regarded as a kind of "revolution" in architecture and civil engineering. Thomas S. Kuhn offered a theory of "scientific revolutions" in predominantly theoretical fields such as

physics. It is investigated how these developments can be regarded as reflecting Kuhn's ideas of revolutionary changes and what (if any) philosophical lessons can be learnt not only for this case study but also for the rise of computational science.

The contribution is mainly based on materials never used before in a historic study. A major part of the material is based on oral interviews with those who participated in these developments. First-hand accounts are given on deploying high performance computers to construct these large lightweight structures.

This study is part of the author's PhD thesis project on the history of high performance computing and the contributions of the Stuttgart and Karlsruhe areas at the Department for the History of the Natural Sciences and Technology at the University of Stuttgart.

References

[1] Jaeger, Falk (2005), Ingenieurporträt: Frei Otto: Architekt, Konstrukteur und Visionär, Förderer der Leichtbauweise. db deutsche bauzeitung, (139, 6), 72–77.

[2] Thomas S. Kuhn (1970), *The structure of scientific revolutions*, 2nd edition, University of Chicago Press, Chicago.

[3] Marios C. Phocas (2005), John Argyris and his decisive contribution in the development of light-weight structures: Form follows force. Proceedings of the 5th GRACM International Congress on Computational Mechanics (GRACM05), Vol. 1, Limassol, 29 June 1 July, Kantzilaris Publications, Nicosia, 2005.

Computation/Communication
A parallel glance on the transformations of the computer

Maarten Bullynck
Department Mathematiques et Histoire de Sciences
Université Paris 8, France
maarten.bullynck@kuttaka.org

One of the most important developments in the history of the computer is its near transformation from an instrument of computation in its early days into a medium of communication today. Although this transformation, the outcome of many independent and complex processes, is historically welldocumented, many

of the more epistemological aspects of this transformation are still badly understood though they silently underlie defining structural aspects of our everyday experiences with a computer [3]. As everyday experiences unroll in rather small, human dimensions, the interlocking of those experiences with the computer must, at least initially, be studied locally, proceeding in small steps and with attention for detail [4]. Of course, such initial investigation can only be suggestive of further research to get at details of this transformation of the computer. In this case, I will look at a specific fragment of the history of computing, the development of two different brands of parallel computing in the years 1964 -1975.

From an epistemological point of view, perhaps the most interesting chapters in the history of parallel computing are those dealing with “non-trivial” task parallelism. This means, neither the bit, operator or data parallelism that can be built into an essentially still sequentially operating processor, nor the idea of doing n times the same task (trivial parallelism). Instances of this kind of “non-trivial” task parallelism have occurred throughout the history of computing. They can be found in some machines or systems for scientific computation (such as ENIAC, ILLIAC IV, cluster or grid computing), or in computing systems with asynchronous communications (such as time sharing systems, and later on, networks of computers).

This talk wants to contrast parallel computing on the ILLIAC IV with the contemporary development of concurrent programming (both 1964-1975). Although in both cases problems in parallel computing are tackled, they are embedded in different technical settings. The ILLIAC IV was a computer system consisting of a central unit controlling 64 parallel units especially designed for scientific computation and simulation [2]. Concurrent programming was an answer to a set of problems occurring in the design of operating systems for time sharing systems, viz., how to house several users and their programs on one central computer without conflicts [1]. As a consequence, parallel computing on the ILLIAC IV deals mainly with the organisation and computation of numbers and proceeds via synchronous computing cycles, while concurrent programming deals with entities of a different kind altogether, such as programs, processes, users and their asynchronous communications.

Given these different settings and foci of interest, it is obvious that both developed different sets of important problems and questions that would serve as a kind of “test battery” for parallel computing. The relevant techniques and concepts that ultimately evolved from these researches have to be seen against these sets of problems. Said differently, the concepts and techniques developed correlate directly with the kind of problems the computer systems will have to deal with. In fact, looking at the sets of problems studied in both contexts, there is but one problem that both approaches to parallel computing have in common,

viz. the sieve of Eratosthenes. This problem will serve in our talk as paradigmatic example.

Our interest in contrasting ILLIAC IV with concurrent computing, apart from the historiographic value of how sets of problems, concepts and techniques correlate with specific technological setting and research communities [5], lies in the following. Although the computer was originally conceived mainly as a calculator, as a machine dealing with numbers, it now has become increasingly a machine of communications.[1] In our analysis, we look at how one problem (parallel computing) gets "read" in these two contexts of computing. Once as a problem of computational organisation, once as a problem in the organisation of communications. Two salient points of this case-study analysis should be mentioned. First, the role of time, in particular the difference synchronous/asynchronous, is fundamentally different on the ILLIAC IV and in time-sharing. Second, the need for differentiation of types and for a hierarchy of entities is common and often necessary within a concurrent time-sharing system dealing with communications, whereas in the ILLIAC IV computing system, the leveling of data and command types is more frequent.

The outcome of this analysis will permit us to return to the original question of how our understanding of computing alters when shifting from computation to communication, feeding the micro results back into broader questions. The micro-structuring of parallel computing uncovered by the analysis of ILLIAC IV and time-sharing allows to step up one level and to try to partially reconstruct how and to what extent this micro-structuring may contribute to the shaping of our macro-experience with the computer. By making sense of the so-called petty details of computing, one can take the computer part in man-computer interaction seriously and look at how the structures of human experience link up with the patterns of specific computing organisations.

References

[1] Per Brinch Hansen (Ed.) *The Origin of Concurrent Programming. From Semaphores to Remote Procedure Calls*. 2002.

[2] R. Michael Hord. *The ILLIAC IV. The first supercomputer*. 1982.

[3] F.A. Kittler. "Hardware, das unbekannte Wesen." in: Sybille Krämer (Editor). *Medien Computer Realität*. Wirklichkeitsvorstellungen und Neue Medien, p. 119 - 132. 1998.

[1] The ideas that a calculating computer was an information machine and could become a communication machine date back to the 1940s and 1950s (the time of information theory and cybernetics). The actual realisation of those ideas, however, only started in the 1960s and 1970s.

[4] Michael Lynch. “Ethnométhodologie et pratique scientifique: la pertinence du détail”, *Cahiers de recherche sociologique*, 5 (2), 45–62. 1987.

[5] M.S. Mahoney. *Histories of computing*. 2011.

Computation, Information and Computer Models

Giovanni Camardi
University of Catania, Italy
giovanni.camardi@fastwebnet.it

Computation theory and information theory interact productively in computational practice and are treated as highly consilient theories in the philosophy of computing. There are deep-rooted logical intersections between them, as Shannon has shown in 1938 and 1956, before and after his 1948 major paper. However, they are distinct historical entities and their differences, as well as their affinities, are to be carefully identified. Unfortunately, both in scientific practice and historical reconstructions, computational and information-theoretical analyses have often been conflated into one undifferentiated object. Furthermore, the role of Shannon’s theory has been overshadowed ([1]).

Since a few years, a number of philosophers are committed to bring information theory back at the center of philosophical arena. They advocate a ‘semantic’ or ‘strongly semantic’ version of information theory that is much different from Shannon’s alethically neutral theory ([4]), and confirm the conflation of information and computation ([3]).

I disagree with the semantic version of information theory and the conflation practice even more so. I intend to show that Shannon’s information theory plays a central and autonomous role in the development of a research area that is of major interest, today: computational modeling. I will argue that information theory has set the ground for computational models to encompass a statistical apparatus as a fundamental instrument of their construction and updating. And this does not have anything to do with the theory of computation. My argument is as follows: computational models are sophisticated tools, designed for encoding scientific hypotheses in such a way as to make them reliably testable. Mathematical statistics provides sophisticated procedures for testing hypotheses. Shannon’s mathematical theory of communication is based precisely on a *computational* relation between encoding procedures and mathematical statistics. Indeed, Shannon’s primary claim is that a discrete source of information – i.e. a set of encoded data – can be represented by a stochastic process, and viceversa ([8, 40]). Thus, the theory of information grants us a sound epistemic framework to deal with the fundamental components of a computational model.

It “provides a unification of known results, and leads to natural generalizations and the derivation of new results.” ([6, VII])

In order to explain in more detail the impact of Shannon’s concept of information on computer models, I will now compare it to Fisher’s concept of information. As a premise, I will examine briefly the concept of ‘type’.

Let us consider – apart from other important characters – a computational model as “a mathematical model constructed from data types using operations and relations that are computable relative to those types” ([10]). On the one hand, computational data types are sets of instructions (‘type constructors’) for a logical variable to occur in a programming language. On the other hand, information-theoretical ‘types’ are “sequences that have the same empirical [probability] distribution” ([2, 347]). These definitions are not mutually homogeneous and suggest no straightforward convergency, at an operational level. The computational type is supposed to put constraints on the information-theoretic type, while the latter should provide measurements to be applied in reverse to the former. These measurements must have been previously derived from an estimation of experimental data, then corrected and re-fitted into the computational structure (the data types) of the model ([7]). Winsberg ([11]) describes such a fitting process as a sequence of “back-and-forth, trial-and-error piecemal adjustments” of statistical parameters. A not forthright process, as I said. The focal point of statistical evaluation is Fisher’s Information Concept, defined as the measure of adequacy, or the estimation of the variance, of a statistical parameter. This is an illuminating notion, much helpful in understanding the mathematical basis for statistical investigations to be reliable. But Fisher was dealing only with numerical issues. His concept of information was limited to error control and could not admit type recoding operations as a consequence of statistical divergences. Quite to the contrary, Shannon’s information can do this. So, if we carry out a careful analysis of Shannon’s notion of entropy and compare it to Fisher’s information, we have to conclude that entropy is a much more powerful concept than parameters estimation. I submit that information theory provides a more penetrating control on hypotheses and models than statistical validation, because it extends our ability to update models beyond the level of numerical computation, up to the level of encoding schemata constituting the logical structure of a model. Information theory can manipulate encoding procedures precisely via statistical analyses.

Thus, the coding process can be properly divided into two packages, “the coder and the model. The model, or statistical processor, passes information about the statistical nature of the source text to the coder, which uses then this information to encode the source text efficiently.” ([5, 181])

As a conclusion, I will consider an example from metereology, a typical object of computational models ([9]). I will show that ‘post-statistical’ computational

processing of ‘parameterization schemes’ is performed using the resources of information theory.

References

[1] Aspray, W., (1985), ‘The Scientific Conceptualization of Information: a Survey’, *Annals of the History of Computing*, 7, 117-140.

[2] Cover, T. and J. Thomas, (2006), *Elements of Information Theory*, Wiley, New York.

[3] Floridi, L., (ed.), (2003) *The Blackwell Guide to Philosophy of Computing and Information*, Oxford University Press.

[4] Floridi, L., (2011) *The Philosophy of Information*, Oxford University Press.

[5] Henkerson, D., Harris, G. and P. D. Johnson, (2003), *Introduction to Information Theory and Data Compression*, Chapman Hall /CRC, Boca Raton.

[6] Kullback, S., (1968), *Information Theory and Statistics*, Dover Publications, Mineola, New York.

[7] Mayo, D. and A. Spanos, (2010), *Error and Inference*, Cambridge University Press.

[8] Shannon, C., Weaver, W., (1998), *The Mathematical Theory of Communication*, University of Illinois Press, Urbana.

[9] Stensrud, D., (2009), *Parameterization Schemes: Keys to Understanding Numerical Weather Prediction Models*, Cambridge University Press, 2009.

[10] Turner, R. (2009), *Computable Models*, Springer, London.

[11] Winsberg, M., (2010), *Science in the Age of Computer Simulation*, University of Chicago Press.

Kolmogorov Complexity and Information Theory: the meaning of being minimal

Duilio D’Alfonso
Università della Calabria, Cosenza, Italy
duilio.dalfonso@tin.it

I intend to illustrate the meaning of minimal programs for tractable problems. The concept of minimal program is the crucial concept in the Kolmogorov algorithmic theory of complexity. Algorithmic information theory (Kolmogorov Complexity), applied in computability theory, has been conceived to tackle with the

lower bounds of tractability's problems. Roughly speaking, the gist of the argument is as follows. If you cannot write a minimal program that is significantly shorter than a given set of data you want to reproduce, such set lies, in a sense, beyond the (lower) threshold of tractable complexity. In other words, the analysis almost always revolves around the question: what happens if (and what does it mean that) a bit-string, representing the values of the function to be computed, is not compressible?

In this paper I propose to consider the complementary question: what happens if you can compress a string representing a problem, i. e. if you can write a minimal program significantly shorter than the set to be reproduced? At a first glance, the answer is quite obvious: the problem is tractable. But this true and obvious answer may be the starting point of some consideration.

In a perspective inspired by Shannon Information Theory, shortening, in computation, means capturing regularities, eliminating noise and redundancy, identifying relevant parameters for determining the output of the computation.

A program p shorter than another program q reduces the entropy involved in the computation: shortening means reducing information needed to compute the output. This point can be expressed mathematically by defining a "gain function", depending on the entropy of a program. Essentially, the gain function can be defined as the difference between the entropy of the problem and the entropy of a program resolving that problem. So defined, the gain function explicates the strong tie between the length of a program and the optimization of the computation: minimal programs maximize the gain function.

After having introduced the gain function in general, I'll proceed with some illustrative example of application, taken from those areas of Artificial Intelligence, such as machine learning, where shortening as optimization is of crucial relevance, not only for computational reasons, but also for more general theoretical reasons (especially when the aim is to model human capabilities).

References

[1] M. Li and P. Vitànyi, *An introduction to Kolmogorov complexity and its applications*, Springer-Verlag, Second Edition, 1997.

[2] D. Hammer, A. Romashchenko, A. Shen and N. Vereshchagin, Inequalities for Shannon Entropy and Kolmogorov Complexity. *Journal of Computer and System Sciences*, **60**(2): 442-464, 2000.

[3] C.E. Shannon, The mathematical theory of communication. *Bell System Tech. J.*, **27**:379-423, 1948.

On models of computation and the analysis of feasibility

Walter Dean
Department of Philosophy
Warwick University, United Kingdom
w.h.dean@warwick.ac.uk

The purpose of this paper is to examine whether it is possible to refine the criteria traditionally used to analyze effective computability so as to obtain a mathematical characterization of the class of computational models adequate for the development of complexity theory. In the broad sense, a model of computation $\mathfrak{M}$ may simply be taken to to be a pair consisting of a class of mathematical structures $\mathbf{M}$ (which, following tradition, I will refer to as *machines*) and a definition $App_{\mathfrak{M}}$ which specifies what it means to *apply* a machine $M \in \mathfrak{M}$ to an input of the appropriate sort. If the machines of $\mathfrak{M}$ are understood as computing (possibly partial) functions of type $X \to Y$, the result of *applying* $M \in \mathbf{M}$ *to* $x \in X$ – symbolically $App_{\mathfrak{M}}(M, x)$ – will be an element $y \in Y$.

What appears to ground our willingness to regard a given structure $\mathfrak{M} = \langle \mathbf{M}, App_{\mathfrak{M}} \rangle$ as a model of computation is that it is possible to construct $M \in \mathbf{M}$ which we are willing to accept as *procedural models* of at least certain informally specified algorithms (e.g. Euclid's algorithm or long division) – i.e. mathematical representations which represent the mode of operation of such procedures relative relative to the definition of $App_{\mathfrak{M}}$. Most readers will be familiar with the range of formalisms which are traditionally classified in this way – e.g. variants of the Turing machine model, lambda calculi, systems of recursive equations, grammatical production systems, and variants of the Cook-Rechkow RAM model. Matters of family resemblance aside, however, standard references (e.g. [8]) do not present these models as instances of a more general mathematical definition. It is thus reasonable to ask whether it is possible to identify the features in virtue of which we judge them to fall under the heuristic characterization of a model of computation.

If this question is approached from the standpoint of computability theory, it is possible to replace it with a more familiar one: what mathematical conditions must be imposed on $\mathfrak{M}$ in order to ensure that the class of functions $\mathcal{F}_{\mathfrak{M}} = \{\lambda \vec{x}.App_{\mathfrak{M}}(M, \vec{x}) \mid M \in \mathbf{M}\}$ coincides (possibly under an effective encoding of X and Y into $\vec{\mathbb{N}}$) with the class of partial recursive functions PartRec? Since for most familiar models $\mathfrak{M}$, it is easy to see that PartRec $\subseteq \mathfrak{F}_{\mathfrak{M}}$, this question may be answered by specifying conditions sufficient to ensure that $\lambda \vec{x}.App_{\mathfrak{M}}(M, \vec{x})$ is partial recursive for all $M \in \mathbf{M}$. In the case where $\mathfrak{M}$ is an *iterative* model of computation – i.e. such that the $M \in \mathbf{M}$ have the form $\langle St, \sigma \rangle$ where St is a class of computational state and σ is a transition function of type $St \to St$ and

$App_{\mathfrak{M}}$ is defined in terms of the iteration of σ – this question has been examined in detailed. In particular, Gandy [3], Sieg [9] and Dershowitz and Gurevich [2] have all argue that conditions of following form ought to be sufficient to ensure the inclusion $\mathfrak{F}_{\mathfrak{M}} \subseteq$ PartRec:

(B_0) Boundedness: The value of $\sigma(s)$ must depend only on a bounded region surrounding the current loci of computation (of which there may only be finitely many).

(L_0) Locality: For all $s \in St$, if $\sigma(s) = s'$, then s' can only differ from s in a finite region surrounding the computational loci.

Familiar models like the single tape Turing machine satisfy these properties relative to the natural interpretations of "computational loci" (i.e. head position), "region" (i.e. scanned tape cell), and "depend" (i.e. determination of the value of the transition function δ by the current state $q \in Q$ and the currently scanned symbol). However, making sense of (B_0) and (L_0) with respect to an arbitrary model of computation $\mathfrak{M}$ appears to require that we find analogous spatio-temporal and causal interpretations of the features of its machines.

It is generally straightforward to find such interpretations of the models mentioned above and thereby show precisely that they satisfy Gandy's set theoretic analyses of (B_0) and (L_0). However, it is also possible to show that these conditions are satisfied by models which are not generally considered admissible for the development of complexity theory. In this context, we must define for each model $\mathfrak{M}$ definitions of time and space complexity $time_M(x)$ and $space_M(x)$ which are respectively intended to analyze the number of basic computational steps performed and the number of storage or memory locations accessed by $M \in \mathbf{M}$ during its operation on input x. The term *reasonable* (cf., e.g., [8]) is conventionally adopted in this context to describe models for which these measures are in at least rough accord with the practical exigencies we face in everyday computation. If $\mathbf{M}$ contains a machine M which efficiently solves a mathematical problem which is known (or expected) to be genuinely infeasible in practice (i.e. to lack an efficient algorithm) – then $\mathbf{M}$ will typically not be regarded as reasonable. For instance, although many familiar models like the single- and multi-tape Turing machine and sequential RAM model with unit time addition and subtraction are regarded as reasonable, both the Parallel RAM (PRAM) model and sequential variant allowing for unit time multiplication (MBRAM) are regarded generally as unreasonable. For instance, both can be shown to contain machines M which solve **NP**-complete problems (e.g. SAT or the Traveling Salesman Problem) in with running time $time_M(x)$ polynomially related to $|x|$.

Since it is also widely accepted that polynomial time decidability is an extensionally adequate analysis of feasible decidability, the existence of such machines has traditionally been taken to demonstrate that PRAM and MBRAM are not models

relative to which feasibility can be accurately measured. However, neither model allows for the computation of functions outside PartRec and both may be shown to satisfy Gandy's formalization of (B_0) and (L_0).[1] Since this analysis is explicitly aimed at encompassing effective models of parallel computation, this result is hardly surprising. However, it does point to a natural refinement of the question posed above – i.e. is it possible to formulate mathematical conditions which pick out the class of models which are regarded as reasonable in practice?

The traditional approach of complexity theory has been to attack this problem extrinsically – e.g. by first observing that the complexity of a problem relative to the Turing machine model provides an accurate gauge of its practical computational difficulty and then proposing that the class of reasonable models should be equated with those whose members may be efficiently simulated by Turing machines. But it is also possible to ask whether the class of reasonable models can be characterized in a more intrinsic way.[2] I will address this question in the following way: 1) I will first attempt to provide a general characterization of what is meant by an admissible definition of time or space complexity for an arbitrary model of computation $\mathfrak{M}$; 2) on the basis of this characterization, I will propose a means of refining Gandy's original formulations of (B_0) and (L_0) by proposing that the operations in terms of which a reasonable model is given must be definable in a weak set theory (e.g. a subsystem of $\mathsf{ZF} - \mathsf{Inf}$ as studied in [5]).

References

[1] S. Bellantoni and S. Cook. A new recursion-theoretic characterization of the polytime functions. *Computational complexity*, 2(2):97–110, 1992.

[2] N. Dershowitz and Y. Gurevich. A natural axiomatization of computability and proof of Church's Thesis. *The Bulletin of Symbolic Logic*, 14(3):299–350, 2008.

[3] R. Gandy. Church's thesis and principles for mechanisms. In H. K. J. Barwise and K. Kunen, editors, *The Kleene Symposium*, volume 101, pages 123–148. North Holland, 1980.

[1] It is also possible to directly construct a Gandy machine (i.e. structure $M = \langle St, \sigma \rangle$ satisfying condition I-IV of [3]) which solves SAT in polynomial time – see [6].

[2] A related project is to provide an intrinsic characterization of complexity classes such as **P** and **NP** in a manner which is independent of specific models of computation or time bounds. Such analyses have been proposed from the perspective of descriptive complexity theory (cf., e.g., [4]) and proof theory (cf., e.g.,[1]) and can be taken to provide evidence of the robustness of these classes and hence (indirectly) for the equation of a problem's feasibility with its membership in **P**. The approach advocated here is complementary in the sense that it seeks to provide justification for the use of the standard models as metrics of intrinsic complexity once this equation is accepted.

[4] Immerman, N. *Descriptive complexity*, Springer Verlag, 1999.

[5] R. Kaye and T. L. Wong. On Interpretations of Arithmetic and Set Theory. *Notre Dame Journal of Formal Logic*, 48(4):497–510, 2007.

[6] A. Obtulowicz. Gandy-paun-rozenberg machines. *Romanian journal of information science and technology*, 13(2):181–196, 2010.

[7] W. Sieg. Church without dogma: Axioms for computability. *New Computational Paradigms*, pages 139–152, 2008.

[8] P. van Emde Boas. Machine models and simulations. In J. Van Leeuwen, editor, *Handbook of theoretical computer science (vol. A): algorithms and complexity*. MIT Press, Cambridge, MA, 1990.

Simulation and theory choice

Rogier de Langhe
Helsinki Collegium for Advanced Studies, Finland
rogierdelanghe@gmail.com

Science presents us with multiple accounts of similar phenomena. As such the problem of theory choice is a central problem in philosophy of science. Usually accounts of theory choice take the form of a simple utility calculus which is then maximized. The challenge is then to find the right utility function, viz. that function which weighs epistemic values against each other. The assumption of the existence of such a utility function is equivalent to assuming the existence of a Scientific Method. Since the demise of logical empiricism, the notion of a unique scientific method has lost its appeal. However, it remains unclear how theory choice can be rational without a shared utility function which ultimately allows to rationally compare alternatives. For example Thomas Kuhn denied the existence of such a common utility function[1] (every such utility function is at least partially relative to a paradigm) but he was at a loss how in such a situation scientific rationality can be saved.

> "Even those who have followed me this far will want to know how a value-based enterprise of the sort I have described can develop as a science does, repeatedly producing powerful new techniques for prediction and control. To that question, unfortunately, I have no answer at all [...] The lacuna is one I feel acutely" ([2, 332-33])

[1] Kuhn [1, 94]: "the choice [between paradigms] is not and cannot be determined merely by the evaluative procedures characteristic of normal science, for these depend in part upon a particular paradigm, and that paradigm is at issue".

The formulation of the problem of theory choice in philosophy of science has long been committed to the assumption of a Scientific Method because it was thought that abandoning this assumption would lead straight to irrationality. Indeed many of the harshest criticism on Kuhn was aimed exactly at this point ([1, 198-99]). Now that better formalism and understanding is being attained on the dynamics of complex adaptive systems which typically have multiple, not necesarily optimal equilibria, this turns out to be a non sequitur. It is indeed possible to behave rationally in a situation with no general utility function. This only follows if rationality is equated with the existence of a unique, optimal solution. Indeed, probably most problems humans are faced with are exactly of this nature. Scientists and philosophers alike should not limit their domain to those aspects of the world which fit their preferred model of (unique and optimal) rationality, but should rather find out how they can expand their domain of application to model more complex and realistic situations.

The aim of my paper is to demonstrate that computer simulation allows to rationally deal with the problem of theory choice in the absence of a scientific method. My account will consist of a notion of scientific rationality based on Herbert Simon's notion of 'satisficing.[2] I will then describe the problem of theory choice under multiple scientific methods in analogy to the multi-armed bandit problem.[3] A multi-armed bandit problem consists of a series of jackpots each with different, previously unknown payoffs. An agent needs to make a rational decision about what lever to pull. The characteristics of this problem show remarkable similarity to the problem of theory choice, most importantly the need for a rational choice in the absence of a shared utility function. As a consequence this problem does not have an analytical solution, but still it is possible to develop satisficing strategies (cf. Herbert Simon) which guarantee a satisfactory outcome every time the game is played. To develop this analogy and to explore different strategies, simulation is an essential tool. This framework will allow me to draw a number of epistemological consequences. This will

References

[1] Kuhn, T. (1970). *The Structure of Scientific Revolutions*. 2nd ed. Chicago, Chicago University Press.

[2] Kuhn, T. (1977). *The Essential Tension*. Chicago and London: University of Chicago Press.

[3] Mayo-Wilson, C., Zollman, K. & Danks, D. (unpublished manuscript), 'Wisdom of the Crowds vs. Groupthink: Learning in Groups and in Isolation',

[2] The term was introduced by Simon in his [4].

[3] See [3], for an application to collective rationality. As early as in [5] this analogy was used to model choice between research projects in a science foundation.

retrieved May 15th from http://repository.cmu.edu/cgi/viewcontent.cgi?article=1100&context=philosophy.

[4] Simon, H. (1956), 'Rational Choice and the Structure of the Environment', *Psychological Review*, 63(2), pp. 129-138

[5] Weitzman, M. (1979), 'Optimal Search for the Best Alternative', *Econometrica*, 47(3), pp. 641-54

From Symbol to 'Symbol' Turing, Hilbert and the Quasi-concreteness of Signs

Eli Dresner
Department of Philosophy
Tel Aviv University
dresner@post.tau.ac.il
Ofra Rechter
Department of Philosophy
Tel Aviv University
rechter@post.tau.ac.il

In the first part of the paper we note the similarities between Turing's conception of symbols in § 9 of his 1936 paper and Hilbert's conception of signs in the early 1920's. We then argue that an affinity between the respective appeals to signs and symbols is instrumental to appreciating the nature of an important contrast between them, to which we devote in the second part of the paper. Turing's interest in the nature of symbols is focused on their suitability for a particular characterization problem (or cluster of such problems). To be sure, Turing is characterizing an intuitively familiar domain of mathematical objects and not a domain of the truths of a theory, so the question of establishing acceptability, engaging Hilbert, is moot. But this might obscure the fact that Hilbert too is engaged in a characterization problem for which the symbol structure of sign strings is exploited. (Interestingly, this can be obfuscated by failure to notice distinguishable types of view about Hilbert's formalism: its metaphysical association with contemporary nominalism as opposed to an emphasis on symbol structures' appropriateness for solving certain characterization problems.) Hilbert aims to capture the truths of a rich theory like ZF as the theorem of its axioms by showing that the characterization of the class of truths of the theory – as well as the theory's acceptability – depend on simple combinatorial properties of the theory considered as a symbolic system. The role of signs in showing the acceptability of the theory is distinguishable from their role in Hilbert's goal of capturing the class of the truths of the theory, which is, of course, equally a part of Hilbert's formalism, and appears to be as much a characterization problem as Turing's.

In Part II it is shown that Turing's formal treatment of symbols (as part of his formally defined computing machines) involves an important divergence from Hilbert: Drawing on the distinction between quasi-concrete and pure abstract objects as found in the work of Charles Parsons we articulate the shift from the notion of symbol in §9 of the paper. The symbols described in §9 are, like Hilbert's signs, are what Parsons calls quasi concrete, but the notion of symbol in Turing's formal model described in the mathematical sections of the paper no longer assumes their quasi concreteness. The point is both historical and conceptual. We offer indirect arguments for this central claim (from the mathematical abstractness of the machines), and go on to argue for it directly. This aspect of Turing's work consists in yet another significant (and hitherto not sufficiently acknowledged) novelty of his 1936 paper.

We distinguish this aspect of Turing's work from claims implicit in Sieg's analysis and from consequences of the well-known observation that Turing Machines can be axiomatically characterized. In Sieg's account the status of symbols or its development is not raised. But we find supporting considerations in Sieg's systematic work (especially [7, 8, 9]). We consider the relation between our analysis and Sieg's account.

Our analysis invites the conjecture that the novelty of Turing's view might be further clarified if it is shown to permit a theory of strings that does for the symbol structures something comparable to the axiomatic characterization of Turing Machines, indeed that on Turing's view the theory of strings itself can be axiomatically characterized. Deploying Parsons' apparatus we introduce the question of the logical relation between the non-quasi concreteness of symbols, their pure abstractness, and the idea of the axiomatization of the theory of strings itself.

We conclude with possible objections to which Turing is vulnerable even by the weaker commitment to the symbols' non-quasi concreteness, and pose two questions for further research: can Turing be criticized for the departure from the quasi-concreteness of symbols, and if so, how generally the force of the criticism might extend beyond the constrict of a Hilbertian point of view.

References

[1] Bernays, P. 1998. Reply to the Note by Mr. Aloys Müller, 'On Numbers as Signs'. In P. Mancosu. Ed., *From Brouwer to Hilbert, The debate on the Foundation of mathematics in the 1920s*, Oxford: oxford University press, 223-226.

[2] Hilbert, D. 1996. The New Grounding of Mathematics. First Report. In Ewald, W., ed., *From Kant to Hilbert: A source Book in the Foundations*

of Mathematics, Oxford: Oxford University Press, 1115-1134.

[3] Hodges, A. 1983. *Alan Turing: The Enigma*, New York: Simon and Schuster.

[4] Hodges, A. 2007. Did Church and Turing Have a Thesis about Machines? In A. Olsewski, and J. Wolenski (eds.), *Church's Thesis after 70 Years*, Frankfurt: Ontos Verlag, 242-252.

[5] Parsons, C. 1983. *Mathematics in Philosophy* Ithaca: Cornell University Press.

[6] Parsons, C. 2008. *Mathematical Thought and Its Objects*, Cambridge: Cambridge University Press.

[7] Sieg, W. 1994. Mechanical Procedures and Mathematical Experience, in George, A. (ed.), *Mathematics and Mind*, Oxford: Oxford University Press, pp. 71-114.

[8] Sieg, W. 2002. Calculations by man and machine: conceptual analysis, in W. Sieg, R. Sommer and C. Talcott (eds.), *Reflections on the Foundations of Mathematics*, Natick, MA.: Association for Symbolic Logic, pp. 390-409.

[9] Sieg, W. 2008. Church without Dogma: Axioms for Computability, in S.B. Cooper, B. Löwe, and A. Sorbi, (eds.), *New Computational Paradigms Changing conceptions of what is computable*, New York: Springer, pp. 139-152.

[10] Tarski, A. 1934. The concept of truth in formalized languages, reprinted in A. Tarski, *Logic, Semantics Metamathematics*, Oxford: Clarendon Press.

[11] Turing, A. 1936-7. On computable numbers, with an application to the Entscheudungsproblem, *Proceeding of London Mathematical Society* 42 (2), 230-265.

[12] Turing, A. 1939. Systems of Logic based on Ordinals, *Proceedings of the London Mathematical Society*, 45 (2), 161-228.

A long history of automation: from its origins to the computer

Marie d'Udekem-Gevers
Faculté d'informatique, Namur University, Belgium
marie.gevers@fundp.ac.be

The talk begins by proposing an operational and generalisable definition of the concept of automaton and describes the two principles that allow automation to function, namely, regulation and programming. These two principles are not incompatible: they both function, for example, in a mechanical clock or in a computer.

It is made clear that the programming takes place, on the one hand, by determining in advance the sequence of operations (or actions) that have to be effected by a machine and, on the other, the recording of this sequence on a material support that serves as a memory. The opportunity is taken to explicitly make an enlightening distinction between the program such as it is conceived by man and the program such as it is understood by the machine. Then, a new detailed classification of programs (such as they are understood by the machine) is suggested. According to this typology, a program is said to be interior (to the machine) if it is fixed. But this description is inadequate: it is indispensable to say whether the program is 'distributed' (that is, there is no centralizing of the sequencing) or whether it is 'centralized' (in the opposite case). A program is said to be exterior (to the machine) if it is modifiable (this implying a centralization of the sequencing) manually. Finally, a program is said to be recorded in the (central) memory, in the last possible case. One final fundamental detail: a machine can have several levels of programming. Mention will also be made of the underlying technologies of automation (mechanical, electromechanical and electronic) and their evolution over time.

While systematically applying this programming typology and detecting the cases of regulation, I will review the technical developments of mechanical tools having recourse to an automation that was employed until the beginning of the 19^{th} century in different domains : the measuring of time and the parts of clocks, music (programmed organs), entertainment and teaching, weaving.

Famous calculating machines are subsequently analyzed and compared by applying the typology defined above. The Pascaline, a mechanical adding machine constructed by Blaise Pascal, can in fact be defined by its interior distributed program which allows the carry mechanism to be automated (thanks to a series of control levers each situated between two gears). As for the machines envisaged by Charles Babbage, they have several programming levels. The Difference Engine partly constructed in Babbage's time has two: it incorporates, in addition to the distributed interior program (analogous to that of the Pascaline but better), a centralized interior program (concretized by an irremovable gear) dictating the execution of a certain sequence of additions which is always the same. And as it was subsequently envisaged by Babbage, the Analytical Engine has three programming levels. This machine testifies, it is true, to a revolutionary ambition and ingenuity: it is a question of automating the execution of any sequence of operations by basing it on an exterior program (in the form of punched cards), itself implemented by an interior centralized program, a kind of 'microprogram' (concretized by a cylinder with studs) which will, in turn, direct the sequencing of the carriage mechanism realized by an interior distributed program (bringing into play sophisticated levers situated between the gears piled up to form cylinders). Moreover, the Analytical Engine had to be capable of making a decision on the basis of a result that it had obtained, in other words: its exterior program

had to allow ‘conditional branching’, or in yet other words: the machine had to be ‘regulated’.

In the form of a partial synthesis, I will then, in diagrammatic form, propose a new attempt at the ‘phylogenesis’ of the programming of mechanical tools from the 12^{th} to the 19^{th} century in Europe, covering the different domains that were tackled until then. Reference will be made to the mechanographic (electromechanical) machines of the end of the 19^{th} century, to show that they are not automata. Mention will subsequently be made of large calculating machines (electromechanical and electronic) of the 1940s, their programs being either interior or exterior (via strips of perforated paper or panels of electrical connections, for example).

I shall finish by focusing on the computer. It will be seen that the underlying technology is necessarily electronic, to comply with temporal requirements. The fundamental features of the computer will be defined, it being emphasized that they are identical to those of the Analytical Engine with one exception: the highest level program is here recorded in the central memory (instead of remaining on the outside), which implies the understanding by the automaton of a ‘machine language’. The path that has led to this new option will be explained and one immediate consequence highlighted: henceforth the machine will be capable of itself modifying its own program.

The extent of this consequence was not immediately perceived at the time. However, it turned out to be decisive: henceforth a programmed automaton could help man not only to calculate but also to write its programs! It was now indeed possible to invent and use high level languages (that is to say, near to natural language or mathematical language) without knowing the specific material characteristics of the computer employed to produce a program which would in turn be provided as a given to another program, called a compiler. And this compiler would as a result produce a program in machine language. The concept of ‘software’ emerged and was clearly distinguishable from that of ‘hardware’.

Language for algorithms, or algorithmic language?

Helena Durnova
Masaryk University, Brno, the Czech Republic
helena.durnova@mail.muni.cz

Can an algorithm formulated in a clumsy way and without the use of a higher level programming language be as efficient as those formulated in one? Such is the case of the minimum spanning tree algorithm formulated in 1926: this solution, dubbed obscure three decades later, proved to be one of the most efficient

approaches to the minimum spanning tree problem [9]. Moreover, its author did not even call it an *algorithm*, but simply a *solution*. Such solutions, nowadays usually called algorithms, appeared and continue to appear outside the context of computers and programming languages, with the famous Euclidean algorithm hardly being the oldest. The term itself derives from the name of the Arabian learned man al-Khwarizmi, but its specific and widely spread-out usage to denote certain approach is rather recent: algorithms devised for problems in discrete mathematics as late as mid-1950s were called procedures, constructions, or simply solutions to a problem. As a specific way of solving problems, algorithms have attracted the attention of philosophers who ask about the limits of such approach [7].

The term algorithm, synonymous with reckoning since the 12^{th} century, was given a new meaning by Leibniz. For Leibniz, algorithm denoted above all the straightforward rules used in algebra. Together with such strict mechanical rules, Leibniz's universal characters would allow mechanization of reasoning [6]. The search for accurate ways of expressing algorithms as well as of adequate ways for talking about their qualities can still be found in papers devoted to algorithmic solutions to problems, e. g. in discrete optimization, in the mid-1950s. Both the formulation and the verbal evaluation of the algorithm were inherent to the mathematical environment and related to the justification of the solution as a new mathematical result. Although judging the quality of mathematical results is not formalized, the originality of the discovery as well as its simplicity and elegance of the discovery are among the evaluation criteria [1]. Indeed, the authors themselves justified their result by having arrived to a more comprehensible, faster, or simply better solution [3].

Formal analysis of algorithms, in terms of their consumption of time and space started in the mid-1960s [8] with carefully phrased notes, which were called, for example, a *philosophical digression* [4]. While considering whether 'efficient' or 'good' is a better word to be used for evaluating algorithms, Jack Edmonds also makes a clear distinction between a conceptual description of an algorithm on the one hand and a particular formalized algorithm on the other.

Older and newer algorithmic solutions can only be compared through understanding the language in which these solutions were written. Matthias Dörries [2] and Ladislav Kvasz [5] have shown the crucial role of language in communicating respectively, science in general and mathematics in particular. With computers, the issue of language comes to the fore not only on the level of feasibility of machine translation, but also on an almost intrinsic level: in the 1950s, many people in computing (e.g. John von Neumann and Heinz Rutishauser) thought that trained mathematicians should have no problem when translating their problems into a language understandable by computers. However, as translating of the language of mathematics into machine language was found to be

extremely laborious, this task was left to the compilers for various programming languages.

Of the new programming languages, one, namely ALGOL 60, was chosen by the Association for Computing Machinery as the publication language of the algorithms section of the *Communications of ACM*. The example was followed in the mid-1960s by the algorithms section of the Czechoslovak journal *Aplikace matematiky*. The algorithmic language, abbreviated as ALGOL, thus became a major language for exchanging algorithms in an easily comparable way by formalizing expressions for loops and branches in the notation of algorithms.

Translating algorithms formulated prior to the widespread use of programming languages need not be a direct process, as the older, non-standardised, formulations might be only conceptual descriptions of the algorithm rather than descriptions leading to mechanical execution of the procedure. Colloquial language formulations also usually did not call for the analysis of the algorithm in terms of time and space consumption. What was valued more was the insight and the elegance of the solution. The paper will attempt to show what kind of mathematical culture stimulated the formulation of exact algorithms while their authors could not even think of trying the algorithms out on a computer.

References

[1] Čulík, K., 'O postavení matematiky. O aplikované matematice', *Aplikace matematiky*, vol. 10 (1965), No. 6, 504–508.

[2] Dörries, M., 'Modern science and the spirit of language and philology', keynote lecture at the conference *Language as a Scientific Tool*, Vienna, Austria, 29 November 2010.

[3] Durnová, H., *A History of Discrete Optimization*, Ph.D. dissertation, Masaryk University, Brno, Czech Republic, 2001.

[4] Edmonds, J., 'Paths, Trees, and Flowers', *Canadian Journal of Mathematics*, vol. 17 (1965), 449-467.

[5] Kvasz, L., *Patterns of Change. Linguistic Innovations in the development of classical mathematics*. Birkhäuser: Basel, Boston, Berlin, 2008.

[6] Mahoney, M.S, 'Calculation - Thinking - Computational Thinking: Seventeenth-Century Perspectives on Computational Science', in Folkerts, Menso; Seising, Rudolf (Hg.): *Form, Zahl, Ordnung. Studien zur Wissenschafts- und Technikgeschichte. Ivo Schneider zum 65. Geburtstag*, Stuttgart: Franz Steiner Verlag, 2003.

[7] Parrochia, D., 'Algorithmics and the limits of complexity', *Science in context*, vol. 9, 1996, no. 1, pp. 39-56.

[8] Shallit, J., 'Origins of the Analysis of the Euclidean algorithm', *Historia Mathematica*, vol. 21 (1994), 401-419.

[9] Šišma, P., *Teorie graføu*, Praha: Prometheus, 1997, p.50.

From the Church-Turing Thesis to the Triumph of the Von-Neumann Architecture: The Serialization of Philosophic Thought in Computer Science

John G. Geske
Kettering University, USA
jgeske@kettering.edu

Computer Science, more than any other discipline of inquiry, has been driven by technological development, implementation trade-offs, and market forces. As a result, the philosophical study of Computer Science has been molded by the dominance of hardware development. It is our hypothesis, developed in this paper, that the early adoption of the stored-program computer, the "fetch-execute-store" paradigm of computing, and the central processing unit as the dominant architecture has driven the study of foundational issues to such an extent that we tend to focus on a serial-centric philosophic view of computing in ways that can hinder alternative views of computability and information theory.

In the study of Hilbert's Entscheidungproblem and the concept of effectively computable, three dominant models emerged: the lambda-calculus of Church; Recursive Functions as developed by Kleene and Rosser; the Turing Machine as developed by Turing (with a similar, particularly Orwellian equivalent concept developed by Post). In addition, and somewhat later, Post and Thue developed the concept of term-rewriting systems (or "phrase-structured grammars"). That each of these competing models were found to be equivalent led to the formulation of the Church-Turing Thesis that these models captured the notion of effective computability, and that anything (effectively) computable was computable by a Turing Machine. While equivalent in expressability, each model approached the notion of "computable" in distinct ways that influenced the development of philosophical questions on computability. Nevertheless, the Turing Machine model has become preeminent. The mathematical simplicity of the Turing Model certainly played a role in this, but the attraction the Turing Machine can be partly attributed to the fact that it is the one model that specifically alludes to the mechanistic nature of the Hilbert's idea of "effective," and computer science is all about "computing machines."

In parallel to the development of the philosophical idea of effective computability was the physical realization of computing engines. The architecture of these machines were not directly influenced by the Turing Machine model, but in dealing with electro-physical realities, they did result in the same serialization of

computing processes that was the hallmark of the Turing's development of effective computability. For a multitude of reasons the "stored program machine" that uses a central processing unit and a single, separate memory to store programs and data – what has been coined the "von Neumann architecture" – became the principle architectural model of all successive machines. The inherent serial nature of Turing Machines and physically realized computing machines became the driving force in the development of computer science. The ensuing decades brought about a flurry of alternative machine architectures to challenge the emerging dominance of the von Neumann architecture and deal with what quickly became known as the "von Neumann bottleneck." Ultimately, the market juggernaut of IBM, Control Data, Univac, Digital Equipment predominated, research moved away from alternative views of computing, settled on the von Neumann architecture as a given, and the serialization of computation – the fetch (from memory)-compute(on the processor)-store (into memory) – became a fait accompli.

The serialization of computing, and the very mechanistic nature of it, became the de facto standard (perhaps subconsciously) that influenced much of the foundational work in computer science. The confluence of theory and practice worked to strengthen the view that this was the correct way to view computation. Alternative formulations of computable models, alternative formulations of Church-Turing, always return to the fact that they can "efficiently simulate any realistic model of computation," and so in essence, why bother to think in terms outside the constraints laid out by the "von-Neumann machine?" We present two such consequences of how this thought-process has impacted the study of computable phenomena – one theoretical in nature, one much more practical:

The Complexity of Problems vs. Problems of Complexity. The study of algorithms (processes) and the complexity classes that characterize them has been developed exclusively on the Turing Machine model of computation. [The very elegant work by Ron Fagin on Finite-Spectra being an exception.] Alternative models such as the Random-Access-Machine that more closely resembled the physical realities of modern computing were developed, and a variant of the Church-Turing Thesis, the Invariant Thesis, emerged that 'reasonable' machines can simulate each other within a polynomial-bounded overhead in time and a constant factor overhead in space." Since the time complexity classes of interest are all closed under polynomial composition, the Turing model has become the de facto model for categorization of a problems complexity. Yet, it can be shown that problems of intrinsic complexity (developed on Turing machine models) cannot be adequately captured by the very same complexity classes developed for these models [1]. These results would seem to indicate that alternative

models of capturing complexity are needed. (It should be noted here that the finite-spectra models also fail to capture this complexity.)

Processes vs. Processors: The phenomena of over-control of physical processes. Rolf Pfeifer has investigated how in robotics, artificial intelligence and neuroscience there has been a focus on the study of the control of the neural system itself, and on symbolic or connectionist representations. He has demonstrated that complex processes in nature cannot be the result of a central processor/control concept – the von Neumann/serialization concept of computing and that surprisingly complex behaviors can be achieved with little control and representation [2]. To better understand and mimic even simple (seemingly intelligent) processes we must break away from the linkage of process and a central processor/memory control of these processes.

References

[1] John G. Geske (2006), On the Hardness of Computable Sets, European Conference on Computing and Philosophy, ECAP06.

[2] Rolf Pfeifer, Robots, Embodiment Intelligence, and Philosophy, European Conference on Computing and Philosophy, ECAP10.

Alan Turing and the foundations of computable analysis

Guido Gherardi
Department of Philosophy
University of Bologna, Italy
guido.gherardi@unibo.it

In my talk I deal with Turing's original work on real number computability.

The relevance of Alan Turing's contribute to the rigorous treatment of the intuitive notion of 'effective calculability' is well-known. Nevertheless, the fundamental results obtained by him concerning the computable functions of an integer variable have obscured his important achievements in computability theory for real numbers and real functions. In particular, this is the case with his famous paper '*On computable numbers with an application to the 'Entscheidungsproblem''* [4], where the notions of a computable real number and of a computable real function play a crucial role. This is clear from the title itself: since all natural (and rational) numbers are trivially computable, it is manifest that Turing's interest pertained to real numbers in a peculiar way. The characterization of the

computable real numbers within the indistinct background of the generic real numbers was in fact a primary motivation for the introduction of the machines named after him as 'Turing machines'. These machines were in fact originally conceived to calculate the infinitely long binary expansions of the computable real numbers.

In the same paper, Turing introduced an early definition of a computable real function and provided some important preliminary results on the corresponding theory.

The year after, Turing published a short note, '*On computable numbers with an application to the 'Entscheidungsproblem'. A correction*' [5], in which he changed the representation of real numbers. In fact, after the elaboration of [4], he realized soon that the decimal expansion representation is not suitable for computability theory, and hence he suggested a new representation inspired by Brouwer's treatment of real numbers *via* 'overlapping intervals'. In doing so, Turing gave an example of what, in contemporary computable analysis, is now called an 'admissible representation'. This very short but deep addition to [4] contains important ideas which anticipate some insights achieved by the modern theory of representations as developed by Klaus Weihrauch almost 50 years later. In spite of its importance, this note has been underestimated for a long time, probably because it could not be adequately appreciated before the systematic foundation of computable analysis was completed. [4, 5] provide then a foundation for the so called 'Type-2 Theory of Effectivity' (TTE), an approach to computable analysis based on the use of Turing machines to transform digital sequences of infinite lengths seen as encodings of mathematical objects in metric and topological spaces [7].

Some years later, in 1948, Turing wrote a paper on the solution of linear equation systems [6] in which he followed a remarkably different approach to real number computability. This paper has been seen as providing justification for the 'realRAM machine' model [1, 3], which, in contrast to TTE, deals with real numbers as atomic objects [2]. In particular, [6] contains two fundamental notions maintained in this paradigm. The first one is the evaluation of computation complexity as the number of discrete steps performed during the execution of algorithms. The second one is the notion of a 'condition number', which is used to estimate the impact of round-off errors arising in computations performed by electronic devices where real numbers are in fact substituted by their rational approximations.

This 'double-face' approach to real number computability corresponds to a fundamental dichotomy still present in mathematics: the development of theoretical computer science and numerical analysis as two radically separated subjects.

In my talk I show how both TTE and the realRAM machine model have actually

their foundations in Turing's work, and, viceversa, how the technical tools developed by these two incompatible paradigms allow a systematic interpretation of Turing's pioneering results in the subject.

References

[1] Leonore Blum: Computing over the reals: where Turing meets Newton. *Notices of the American Mathematical Society* 51(9): 1024–1034. 2004

[2] Leonore Blum, Felipe Cucker, Michael Schub, Stephen Smale: *Complexity and real computation*. Springer. 1998

[3] Felipe Cucker: Real computations with fake numbers. *Journal of Complexity* 18: 104–134. 2002

[4] Alan M. Turing: On computable numbers, with an application to the 'Entscheidungsproblem'. *Proceedings of the London Mathematical Society* 42(2): 230–265. 1936

[5] Alan M. Turing: On computable numbers, with an application to the 'Entscheidungsproblem'. A correction. *Proceedings of the London Mathematical Society* 43(2): 544–546. 1937

[6] Alan M. Turing: Rounding–off errors in matrix processes. *The Quarterly Journal of Mechanics and Applied Mathematics* 1(1): 287–308. 1948

[7] Klaus Weihrauch: *Computable analysis*. Springer. 2000

From Computing Machineries to Cloud Computing:
The Minimal Levels of Abstraction of Inforgs through History

Federico Gobbo
CRII – Research Center "Informatica Interattiva"
University of Insubria, Varese, Italy
federico.gobbo@uninsubria.it
Marco Benini
DICOM – Department of Computer Science and Communication
University of Insubria, Varese, Italy
marco.benini@uninsubria.it

The history of computing can be divided in two main periods: the ancient era and the modern era. Since ancient times, humankind succeeded to build methods and tools in order to help in calculation; in particular, in various parts of the world, in completely independent ways, different civilizations such as Roman and Chinese invented the abacus, which was still used by the Russian in 1957

for the necessary calculations to put Sputnik in space [8, 147]. But the tool, i.e., the abacus, was not enough: various methods of representing numbers from 1 until 9,999 with the only help of the fingers were necessary to exploit the possibilities of the abacus, as reported by Leonardo Fibonacci da Pisa in his *Liber Abaci* (1202–1228)—'fingers' in Latin is *digita*, from which our use of 'digit' to indicate numerals derives [1].

Calculating machines were considered auxiliary tools for computation in the simple, non-abstract, sense—i.e., the user puts numbers in to have numbers out, and their meaning is in the eyes of the user himself—even when modern science and mathematics in parallel grew: Schickard's calculating clock (1623), as well as Pascal's *Pascaline* (1642) as well as Leibniz's Step Reckoner (1671–1673) are some notable examples. Their aim was to hide the calculation process to their users, as the idea of calculation as a tedious, low operation of the mind, good for slaves, not for men, as Napier first wrote in 1614 in publishing for the first time logarithms [9, 161] and Leibniz reprised at the end of the century [11, 22]. This divorce between intelligence and calculation, as put by [3], was also the philosophical basis of Babbage's Analytical Engine—his dream was the mechanical calculation and printing of all tables of ephemerides [2].

The modern era of computing was born in 1936, when Church, Post and Turing put the foundations of general-purpose machines, while in 1941 Zuse built the first Turing-complete machine in the world [12]. Unlike ancient times, modern computers were conceived to manipulate *symbols* in form of numbers: as Newell effectively recalled, "I've never used a computer to do any numerical processing in my life" [11, 129]. It is worth noticing that [14] still wrote explicitely 'computing machinery' to refer to machines, not human beings, when he proposed his famous test for Artificial Intelligence—term introduced at MIT by McCarthy in 1959 [10]. The idea behind modern computers is completely different from ancient times: calculation can represent—in digital form—intelligent behaviour or even mind *per se*. In other words, there is an *epistemological* level of abstraction in considering numbers as symbols, i.e., something that stands for something *aliquid stat pro aliquo*. In other words, the symbolization of numbers put in mechanical computation—which eventually constitutes software—is a collection of levels of abstraction (LoA), as "0's and 1's as such have no causal powers at all because they do not even exist except in the eyes of the beholder" [13, 30]. In fact, ontological forms of levellism—i.e., where LoA effectively exist, not only in the eyes of the observer—are hardly tenable if we analyse the generation of information after the Fourth Revolution [6, 37], especially if we do adopt a philosophical monism, i.e., that syntax is not intrinsic to physics [13].

From the advent of general-purpose, Turing-complete machines, the relation between operators, programmers and users with computers, i.e., human-computer systems, or rather interconnected informational organisms or *inforgs*, in Floridi's terms, can be seen in terms of levels of abstraction (LoA), and henceforth analysed with the method of levels of abstraction [7, 5].

In this paper an analysis of LoA throughout history of modern computing is proposed, in order to find the minimal number of LoA needed to explain the epistemology of inforgs—from early modern general-purpose operators of computing machineries until the final users of so-called 'cloud computing'.

This epistemological levellism uses Category Theory as the methodological reference, treating information as functions, i.e., a domain intensionally mapped into a codomain where the inner structure is preserved, instead of Cartesian products. The key idea which is developed in the paper is that the level A abstracts over the level B when it is possible to find a map from A to B which preserves the structure of A. We claim that the notion of 'structure' is suitably coded by categories, in the mathematical sense, while the notion of 'map preserving structure' corresponds to the notion of functor. Finally, a comparison with the method of LoA by [5] is proposed, in order to find a categorial treatment of interconnected informational organisms.

References

[1] Boncompagni, B. (1854) , *Intorno ad alcune opere di Leonardo Pisano*, Tipografia delle belle arti, Roma. `http://gallica.bnf.fr/ark:/12148/bpt6k99398c`

[2] Campbell-Kelly, M., ed. (1994) , *Passages from the life of a philosopher: Work by Charles Babbage written c. 1812*, IEEE Press.

[3] Daston, L. (1994) , 'Enlightenment calculations', *Critical Inquiry* (21), 182–202.

[4] Donovan, J. J. (1974) , *Operating Systems*, McGraw-Hill.

[5] Floridi, L. (2008) , 'The method of levels of abstraction', *Minds Mach.* **18**, 303–329.

[6] Floridi, L. (2009) , Philosophical conception of information, *in* G. Sommaruga, ed., 'Formal Theories of Information: From Shannon to Semantic Information Theory and General Concepts of Information', Springer, Frankfurt Am Main, pp. 13–151.

[7] Floridi, L. (2010) , *Information: A Very Short Introduction*, Oxford University Press, Oxford.

[8] Guedj, D. (2005) , *Numbers: the universal language*, Thames Hudson.

[9] Knuth, D. E. (1973) , *The art of Computer Programming: Seminumerical Algorithms*, Vol. 2, Addison Wesley, Reading (Mass.).

[10] Levy, S., ed. (1994) , *Hackers: Heroes of the Computer Revolution*, Anchor Press.

[11] McCorduck, P. (1979) , *Machines who think*, Freeman, San Francisco.

[12] Rojas, R. (1998) , 'How to make Zuse's z3 a universal computer', *IEEE Annals of the History of Computing* **20**(3), 51–54.

[13] Searle, J. R. (1990) , 'Is the brain a digital computer?', *Proceedings and Addresses of the American Philosophical Association* **64**(3), 21–37.

[14] Turing, A. M. (1950) , 'Computing machinery and intelligence', *Mind* **59**, 433–460.

A history of the stack

Sten Henriksson
Computer Science Department
Lund University, Lund, Sweden
Sten.Henriksson@cs.lth.se

The stack has a special place in the emergence of theoretical computer science, as argued by Michael Mahoney, the pioneer of the history of the theory of computing: "Between 1955 and 1970, a new agenda formed around the theory of automata and formal languages, which increasingly came to be viewed as foundational for the field as a whole" [9]. Interest arose in "devices with more generative power than finite automata, and more special structure than Turing machines" [3] The push-down automaton is such a device.

1. What is a stack? The definition and function of a stack is shortly described. Also the adaption of the word *stack* is discussed. In the OED, there are many stacks, characterized by words like pile or heap but without prescriptions of how items are added or deleted. *Stack* was chosen by E.W. Dijkstra – before that *pushdown store*, *LIFO list* and *cellar storage* were used.

2. A reverse history. We define the push-down automaton and its relation to languages, citing "... the theory of push-down automata is, in fact, essentially another version of the theory of context-free grammar" [4]. The PDA was introduced by Newell in 1959 [11].

The PDA is more capable than a finite automaton, but weaker than a TM. However, an automaton with two stacks can simulate a TM. To complicate there is also the *stack automaton* which is more capable than a PDA but does not reach the level of a TM.

The property of the PDA to accept context-free languages made it useful in the theory and practice of programming languages. Natural language processing in human speakers and listeners also involves parsing, which led Victor H. Yngve to propose that a speaker's short-term memory could be modelled by a stack with limited depth but this was convincingly rejected by Miller and Chomsky [10].

The computer science stack was introduced by E.W. Dijkstra [5]. Dijkstra was a pioneer of compiler construction with the first Algol 60 compiler. Writing about its history, Dijkstra describes his contribution [6]:

> During my 1959 summer holiday in Paterswolde I had given my first thoughts to the question how to implement recursion: in the early months of 1960 we discovered how, in combination with that, to do justice to the scope rules of Algol 60. The definition of Algol makes extensive use of recursive productions. A run-time system for a language like Algol allowing unrestricted procedure calls has to contain some kind of a stack mechanism. If a subroutine/procedure/ method calls itself recursively it is necessary to store return addresses and local variables in such a way that they are not overwritten when the next call of the subroutine is executed. Correspondingly what has been stored must be made available on return from the previous deeper level.

The term recursive was spread from the area of decidability theory. It seems to have returned to computing from this direction. In a session on the history of Algol 60 John Backus was asked where the metalanguage in the definition of Algol 60 came from and whether it was influenced by linguists like Chomsky. John Backus answered that he had attended a class of Martin Davis where the work of Emil Post and the notion of productions was presented. Backus's metalanguage was further developed and simplified by Peter Naur in the classic Algol 60 report [1].

As for the procedure calls, Donald Knuth gives several historic references [8]:

> In 1947 A.M. Turing developed a stack, called Reversion Storage,

> for use in subroutine linkage. No doubt simple uses of stacks kept in sequential memory locations were common in computer programming from the earliest days, since a stack is such a simple and natural concept. The programming of stacks in linked form appeared first in IPL, as stated above; the name "stack" stems from IPL terminology (although "pushdown list" was the more official IPL wording) and it was also independently introduced in Europe by E.W. Dijkstra.

In a 1947 paper Harry D. Huskey describes a stack for storing return addresses of subroutines, referring to a group headed by A.M. Turing [7]. Huskey was for a couple of months a visitor to Dijkstra's group in Amsterdam at the time of Dijkstra's use of a stack in 1959. This may be an indication that the work of 1947 influenced Dijkstra.

With list processing, the use of recursion was spread. Data structures defined recursively could be handled, now that recursion became available in general programming languages. The ideas of structured programming, transparency of code and proofs of correctness also favored recursive definitions.

3. Algebraic expressions and tax avoidance. In a different context, the stack was invented independently by F. L. Bauer and K. Samelson [2] working on the fundamentals of mathematical notation. Their work led to hardware implementations of stacks in the computers KDF9 and B5000.

According to Knuth, the stack principle was first used in calculations of inventory value in business during the 1930-ies.

4. The implicit stack. But of course the stack principle is older. The situation where we have to postpone one action because another one has to be completed is a basic human experience, and can thus be expected to show up in folk literature. I will illustrate by a cumulative tale.

When told, each incident of the tale is put on top of a stack and the current stack content is retold after each new addition. In the last paragraph, the stack is emptied and the tale is brought to its conclusion.

Such tales were spread over Africa and India before Western colonization. So this is where the search for the origin of the stack ends. The stack is a part of human culture. We may conclude that the stack is indeed "a simple and natural concept" and we should not be surprised that it has found its way into computation.

Acknowledgements. A draft version of this paper was presented at a SIGCIS Works in Progress Workshop at the SHOT Conference, Pittsburgh 2009. I thank the participants in the discussion for their valuable comments.

References

[1] Backus, J.W, (1962), *Revised report on the algorithmic language ALGOL 60*, ed. Peter Naur, Copenhagen.

[2] Bauer, F.L. (1990), 'The Cellar Principle of State Transition and Storage Allocation', *Annals of the History of Computing*. Vol 12, No.1, Springer International.

[3] Chomsky, N. (1959), 'On Certain Formal Properties of Grammars', *Information and Control* 2, 139.

[4] Chomsky, N. and Miller, G.A. (1963), 'Introduction to the Formal Analysis of Natural Languages', *Handbook of Mathematical Psychology*, Vol II, p.340, John Wiley and Sons.

[5] Dijkstra, E.W. (1960), *Numerische Math. II*, p. 312.

[6] Dijkstra, E.W. (1980), 'A programmer's early memories', in *A history of computing in the twentieth century: a collection of essays*, edited by N. Metropolis, New York, Academic Press.

[7] Huskey, H.D. (1949), 'Semiautomatic instruction on the Zephyr', *Proceedings of a second symposium on Large-scale digital calculating machinery*, pp. 83-90. Cambridge Mass, Harvard.

[8] Knuth, D. (1968), *The Art of computer programming, vol I, Fundamental algorithms*, Addison-Wesley, New York.

[9] Mahoney, M.S (2004), 'What Was the Question? The Origins of the Theory of Computation', in *Using History to Teach Computer Science and Related Disciplines. Selected Papers from a Workshop*, Atsushi Akera and William Aspray (eds.). Washington, D.C.: Computer Research Association, pp. 225-232.

[10] Miller, G.A. and Chomsky, N. (1963), 'Finitary Models of Language Users', in *Handbook of Mathematical Psychology* Vol II, p.419, John Wiley and Sons.

[11] Newell, A., Shaw, J.C., and Simon, H.A. (1959), 'Report on a general problem-solving program in Information Processing', *Proc. International Conference*, UNESCO Paris.

Logics of programs as a fuelling force for semantics

Francisco Hernández-Quiroz
Facultad de Ciencias
Universidad Nacional Autónoma de México, México
fhq@ciencias.unam.mx

While it cannot be said that programming language semantics is a real jungle, it is fair to call it a diverse ecosystem. Axiomatic, operational and denotational semantics (of various kinds) populate this realm. This variety demands an explanation, and it will probably not be simple.

Nevertheless, some reasons for this diversity have been advanced [10]. The traditional divide between formalism and Platonism in mathematics has been put forward as one of the sources of the different semantic approaches [9]. Roughly speaking, denotational semantics is said to lean towards Platonism, while operational semantics towards formalism, both with some caveats.

Nevertheless, these are not the only reasons for diversity. Another major force behind the flourishing of semantic approaches in the 70 was the search for techniques for improving software reliability which fuelled the development of axiomatic semantics (already in the scene since the end of the 60 [3, 5]). Axiomatic semantics defines the meaning of programs by means of an external language (as did initial attempts at operational semantics), but this aim is not reached through translation, and thus axiomatic semantics is set on very different grounds.

Axiomatic semantics does not look necessarily for explanations of programs' meanings, but for formal tools to verify software. Verification means hear a formal proof that a program (written in an ideal or in a "real" computer language) meets the requirements it was designed for. So, the most common product of an axiomatic semantics is a "logic of programs" for conducting such proofs. A related approach is formal specification and derivation of programs, which also relied on a particular formal semantics.

The so-called "software crisis" of the mid 70 was a strong stimulus for this effort. It was expected that formal tools would help to produce more reliable software which in its turn would help overcome or at least soften the software crisis.

Formal methods for verifying programs experienced a boom from this period (Dijkstra's discipline [2], Gries's Science of Computer Programming [4]).

Axiomatic semantics are themselves quite diverse and this diversity can be sometimes a major weakness. There are different systems based on the language they target or the logical language they are built upon. They also differ on the prop-

erties of programs they can prove. As a consequence, is not so easy to evade the accusation of being ad hoc systems.

There have been efforts to eliminate this ad hoc appearance by grounding logics of programs to a more abstract foundation. An obvious candidate for this foundational role is denotational semantics. One of the most ambitious attempts is Abramsky's domain theory in logical form [1], which intended to derive logics of programs directly from denotational semantics. In its aim to unify two (kinds of) semantic models, it has some parallel with the search for fully abstract semantics (unifying operational and denotational semantics) [6, 7].

Domain theory in logical form is based on the idea that "given a denotational description of a computational situation in our meta-language, we can turn the handle to obtain a logic for that situation", as Abramsky himself explains [1, p. 1]. This is done by using Stone's duality [8]. The denotational side of the semantics takes away the ad hoc flavour of the resulting logic and hence brings together the best of two worlds (at least in theory).

In a way analogous to full abstraction's unification of two semantic paradigms (the denotational one and the operational one), domain theory in logical form shows that the opposition between the very practical, software engineering-based approach of axiomatic semantics is not totally unrelated from the extremely abstract style of denotational semantics and that the entities populating both universes have not so obvious (though not less real) connections.

References

[1] S. Abramsky, Domain Theory In Logical Form, *Annals of Pure and Applied Logic*, 1991, 51, pp. 1–77.

[2] E. Dijkstra, *A Discipline of Programming*, Prentice Hall, 1976.

[3] R.W. Floyd, Assigning meanings to programs, *Proceedings of the American Mathematical Society Symposia on Applied Mathematics*, 1967, vol. 19, pp. 19–31.

[4] D. Gries, *The Science of Programming* Springer Verlag, 1981.

[5] C.A.R. Hoare, An axiomatic basis for computer programming, *Communications of the ACM*, 1969, 12(10), pp. 576–580,583.

[6] R. Milner, Fully abstract models of typed lambda-calculi, *Theoretical Computer Science*, 1977, 4, pp. 1–22.

[7] G.D. Plotkin, LCF considered as a programming language, *Theoretical Computer Science*, 1977, 5, pp. 223-255.

[8] M.H. Stone, The theory of representations for Boolean algebras, *Transactions of the American Mathematical Society*, 1936, pp. 37–111.

[9] Raymond Turner, Understanding Programming Languages, *Minds & Machines*, 2007, 17, pp. 203–216.

[10] Graham White, The Philosophy of Computer Languages, in L. Floridi (ed.), *The Blackwell Guide to the Philosophy of Computing and Information*, Blackwell, 2004, pp. 237–248.

Programming languages as a revealing enterprise in computer science

Raffaele Mascella
University of Teramo, Italy
rmascella@unite.it

Programming languages has been for long undervalued by computer scientists and historicians of the discipline as a characterizing asset. Anyway, secondary literature starting from Mahoney's historical papers, has given an account of programming mostly in terms of its autonomous status inside the computing discipline, of the multiple features in programmers' work, and of software evolution, with the "software crisis" as the key moment. Nowadays many programming concepts seem evident, but top researchers and programmers had to face serious difficulties because nothing was obvious a priori. At the beginnings difficulties were mainly in developing concepts and techniques without inherent preceding work, later on other difficulties arose by the attempt of adopting engineering methods. However, programming languages have been considered a creative and somewhat artistic enterprise, but its scientific features seems greater in many facets, relating basically to logical concepts and reasoning dynamics, and to mathematics, as Dijkstra continuously promoted. Following some pioneering ideas, and with an insight from the general history of technology, the first aim of the paper is on the relation between the artistic and the scientific part, and on the contributions attributable to them.

The history of mathematics shows that the process of development of models for solving problems has been quite long, with constantly improving notations and increasing number of concrete and abstract problems which has been solved. But until the emergence of computer science discipline, notations were intended just for taking a picture of the situations, thus considering mathematics, and logic as well, also the highly developed ones, as static processes. The dynamical part of performing operations, calculating functions, executing solving algorithms was leaved to the ability of the human mind. Instead, as Mahoney [8] suggested, software *"is constructed with action in mind"*. In this sense, one purpose of the paper is to show that one of the main concerns of computer languages is properly epistemological, for its normative decision about how the dynamic mathematical (and logical) processes should be performed, how they have to be

described, and under which ontologies. For example, in the 1950s it was decisive the idea that computers could be reasoned about as symbol manipulators, rather than number crunchers.

The languages built through various analyses, have had wide differentiations (detectable also by a syntactical investigation) due also to the little attention that programmers have usually given to the details of distinct but seemingly similar attempts. But, nonetheless, they have come to share many core ideas, such as recursion, the use of statements with the same meanings, the use of structured data and procedures, and the same different levels of abstraction. On a more general overview, some models have been taken by other sciences, mainly logics, mathematics and engineering, while other models built in here have been used outside the computing discipline, for example the flowchart diagram descriptive model nowadays used almost everywhere, or the model of the cognitive mind borrowed by neuroscientists.

Lastly, I will focus briefly on the fact that history of programming languages is constellated by events which pushes to regard it, as indeed we do with science and technology, as a socially influenced scientific and technological progress. Its evolution, since 1950s, has been influenced by some top researchers, which in some cases have had the habit of regarding themselves and behaving as members of a priesthood, with mysterious skills and knowledge too complex for ordinary people. This situation has had its counterpart in the general low level of education of incoming programmers, whose question of professionalization have been investigated by Ensmenger and others [5], [6]. But also in this set of practitioner creativity and imagination in applying efficiently the tools provided has played a fundamental role, while plans of making languages accessible to a wider population have been often regarded with hostility and derision.

From one side, as we know, a constructionist approach has had a great role in programming evolution, invited by the technological counterpart of programming (programming techniques involved mechanical work as well); from the other side, its process in the establishment of new tools has lived under a seemingly continuous revolutionary process. When new ideas were proposed, usually the acceptance was not provided by their concrete usefulness, or by the elegance of the new systems, or by other factors relating to an objective evaluation of their qualities, but instead heavily connected to the energy and influence of proponents and their companies.

Social factors are probably the only feature that we can consider in analogy with the general idea of scientific revolutions, in which the historical-epistemological analysis shows the presence of knowledge paradigms within science. Some scientists have tried to bring together programming paradigms (sequential, structured, object oriented, and so on) to Kuhnian ones, influenced by the similarity

of terms, but I think it would be more correct to speak of simple evolution of research programs, in line with Lakatos' analysis, and it would be fairer to speak simply of "programming styles" (as indeed some actually do). In my opinion, the basic paradigm of programming has not changed over the past 60 years, as it is strongly linked to the type of computation on which it is based, namely the type of physics, and therefore technology, on which the mechanisms of calculations are performed. And computing technology, as we know, is still based on Newtonian physics.

References

[1] Backus J. (1978), 'Can Programming be liberated from the von Neumann style? A functional style and its Algebra of programs', *CACM* 21, n. 8, pp. 613-641.

[2] Bergin T. J., Gibson, R. G. (1996), *History of Programming Languages*, eds., ACM Press, New York.

[3] Dijkstra E. W. (1982), 'On the fact that the Atlantic Ocean has two Sides', in *Selected Writing on Computer Science: A personal Perspective*, eds., Springer-Verlag, New York, pp. 268-276.

[4] Dijkstra E. W. (1972), 'The humble programmer', *CACM* 15, n. 10, pp. 859-866.

[5] Ensmenger N. L. (2001), 'The 'Question of Professionalism' in the Computer Fields', *Annals of the History of Computing*, vol. 23, n. 4, pp. 56-74.

[6] Ensmenger N. L., Aspray W. (2002), 'Software as Labor Process', *History of computing: software issues: International Conference on the history of computing*, ICHC 2000, Springer-Verlag, New York, pp. 139-165.

[7] Floyd R. W. (1978), 'The Paradigms of Programming', *CACM* 22, n. 88, pp. 455-460.

[8] Mahoney M. S. (1988), 'The History of Computing in the History of Technology', *Annals of the History of Computing*, vol. 10, pp. 113-125.

[9] Mahoney M. S. (1990), 'The roots of Software Engineering', *CWI Quarterly* 3, n. 4, pp. 325-334.

[10] Mahoney M. S. (2002), 'Software as science: science as software', *History of computing: software issues: International Conference on the history of computing*, ICHC 2000, Springer-Verlag, New York, pp. 25-48.

[11] Metropolis N., Howlett J, and Rota G.C. (1980), *A History of Computing in the Twentieth Century*, eds., Academic Press, New York.

[12] Sammet J. E. (1969), *Programming Languages: History and Fundamentals*, Prentice-Hall, Englewood Cliffs, New Jersey.

[13] Weinberg G. M. (1971), *Psychology of Computer Programming*, Computer Science Series, Van Nostrand Reinhold, New York.

[14] Wexelblat, R. L. (1978), *History of Programming Languages: Proceedings of the History of Programming Language Conference*, eds., ACM Monograph Series, Academic Press, New York.

Reimagining Time in Computing: Reservoirs and Aural Arithmetics

Anthony Moore
Academy of Media Arts, Cologne, Germany
moore@khm.de
Kevin Kirby
Northern Kentucky University, USA
kirby@nku.edu

Digital and analog computation have traditionally been framed within simple models of discrete and continuous time. In this paper we consider a richer view of time and show how this leads to an extended view of computing.

We motivate this by discussing time in sound and hearing, 'aural arithmetic' [5] – how time and arithmetic may have been conjoined in pre-literacy periods of both early Greek and non-European histories, and why, with the invention of signs and alphabets, time becomes anathema for arithmetic, as if the latter needed to be purged of it (the Western hegemony of the eye, the shift away from orality and the power of geometry to freeze concepts). We continue our tale of time in computing with the two Turing Machines.

A machine *executes a computation in time*. When we move to formal description in the theory of automata, we refer to sequences of configurations. These could be steps in a grammatical derivation, or a sequence of Turing machine configurations (state, tape content, and head position). These sequences are sets indexed by the natural numbers ($\mathbb{N}$). These whole numbers are our time indices. This is discrete time.

Turning to the 'other' Turing machine, the reaction-diffusion network that Turing used as a proof of principle for a morphogenesis model [8], this can similarly be viewed as executing a computation in time [3]. We again have a sequence of configurations – concentrations of morphogens, their smooth changes unfolding along a time axis. Thus these configurations are a set indexed by the set of real numbers ($\mathbb{R}$). These real numbers are our time indices. This is continuous time.

Now turn away from formalism and look at physical devices or organisms interacting with an environment – transforming, representing, interpreting, comput-

ing. If we can study these systems and come up with a notion of states, we shall then want to track them through time – i.e. index these states by $\mathbb{N}$ or $\mathbb{R}$. But must we be limited in this way?

We could 'break' time by three increasingly radical moves. First, we could move from simple, discrete or continuous time to non-standard topologies of the 'time line', such as the hyperreal number system ${}^*\mathbb{R}$ (which includes infinitesimals and infinite numbers), or chose other non-metric topologies. Second, we could turn from the set-theoretic formulation of time as consisting of a set of points ('points in time') to deal with fusions or other non-set theoretic notions of aggregation [6]. Third, most radically, we could question the applicability of the notion of state.

Examine the use of the word 'state'. A state is a 'snapshot' that maximally determines future behavior. The notion of state itself is thus bound up with our conception of time. When describing a complex configuration of interacting objects in the world we cannot help but use the word 'system', and identifying *material* with a *system* comes with a commitment to the existence of certain states. Further, the assignment of a configuration of objects in the physical world to states is an act of interpretation involving an interplay of convention with physical constraint. This leaves room for a kind of skepticism in which a 'hermeneutic demon' sets up unconventional state assignments and seems to change our computation without changing anything physical [4].

We suggest that one can pull back from the notion of *states* (indexed by a set that represents *time*) and move to a broader, more biological re-imagining of computation. The clue comes from the aural arithmetic idea we opened with. Superficially, it seems like sound is a mere continuous signal in time. But the 'Fourier tradeoff' of time versus frequency (mathematically equivalent to the uncertainty principle in quantum mechanics) is suggestive here. It surprises many people who are unfamiliar with the theory of sound that, although it makes sense to have a 'middle C' note played for the duration of a minute or a second, it makes no sense, *even in principle*, to have a 'middle C' note that sounds only for a picosecond. The superposition of waves is the root of this tradeoff.

We relate this back to computing by reframing a recent class of alternative computing models, now called *reservoir computation* [2];[7]. In reservoir computing models, ripples of inputs create waves of context which reverberate in a *reservoir*, which could be anything from a recurrent neural network to Turing's morphogenesis machine. In a subsequent stage of computation, a more conventional device (a symbolic computer program, a single-layer perceptron) functions as a teachable hermeneutic demon, sorting out the ripples in the reservoir to define meaningful states. This presents interesting tradeoffs between programmability and adaptability, cf. [1]. We suggest that this is a deep idea in computer science:

reservoir-based computing breaks us constructively out of the Turing model to a biocomputing-based approach to computing-in-life.

References

[1] Conrad, M. (1988), 'The price of programmability'. In *The Universal Turing Machine: A Half-Century Survey*, R. Herken, Ed., Oxford University Press, 285–307.

[2] Kirby, K.G. (1991). 'Context dynamics in neural sequential learning'. *Proceedings of the Florida Artificial Intelligence Research Symposium* (FLAIRS 1991), 66–70.

[3] K. Kirby and M. Conrad. (1986). 'Intraneuronal dynamics as a substrate for evolutionary learning'. *Physica D*, Vol. 22, 150–175

[4] K. Kirby (2009). 'Putnamizing the liquid state'. *North American Conference on Computing and Philosophy* (NACAP 2009).

[5] Moore, A. (2006) 'Aromatic Vapours Tuning and Measurement in Greek and Chinese Antiquity'. In *Media before Media: Übertragung, Störung, Speicherung bis 1700*, F. Kittler and A. Ofak, Eds., Wilhelm Fink Verlag München.

[6] Potter, M. (2004). *Set Theory and its Philosophy*. Oxford: Oxford University Press.

[7] Schrauwen, B., Verstrete, D., & Campenhout, J.V. (2007). 'An overview of reservoir computing: theory, applications, and implementations'. *Proceedings of the European Symposium on Artificial Neural Networks* ESANN 2007, 471–482.

[8] Turing, A.M. (1952). 'The chemical basis of morphogenesis'. *Philosophical Transactions of the Royal Society*, B237, 37–72.

Computer science in France: A controversial emergence

Pierre Mounier-Kuhn
CNRS & Université Paris-Sorbonne, France
mounier@msh-paris.fr

How was computing constructed and recognized as a science? The case study of France offers an example of an average, mid-size scientific scene, which may be therefore representative of the emergence process of informatics in other countries, or at least provide an element for international comparisons. As 'computing' was not a defined category until about the late 1960s, at least in the

French University, one has to study computer-related activities and representations at different scales, by changing focus, from the trajectories of individuals, of project teams or local laboratories, to the national or even international level of learned societies, through the decision-making committees or bureaucracies within science policy agencies. My main sources are the archives of these persons and organizations, a collection of some 100 interviews, and the relevant published literature (see part 3 of [1]).

One particularity of the French case is that, contrary to most other advanced countries, no academic or state laboratory succeeded in building a computer during what we may call the 'pioneer era', before 1960. This has been described already in other publications, so I will not analyze it further here, but it is important to remember that the post-war computer effort in France started with a series of failures, which for a while made this new field even less attractive to academic talents, and delayed practical experience with stored-program machines.

From applied mathematics to *informatique*. In the 1950s, French academic pioneers of computing devoted their main efforts to develop numerical analysis and to assert its legitimacy, against the dominant pure mathematics, with support from non-academic allies such as the defense and the electrical industry. After 1955, stored-program computers were acquired from the industry as tools for this low-status, struggling sub-discipline. A first step to define computing, what it was and even more importantly what it was not, was perhaps the strong rejection of cybernetics, which had been a useful brainstorming exercise for a short while, but became considered a topic for journalists and woolgathering babblers.

Toward 1960, new, non-numerical applications of computers were investigated, such as language translation or information retrieval, then artificial intelligence, while the quest for better programming methods led to R&D on languages (particularly Algol), on compilers, then on operating systems. These various research programs broadened the computing field, as in turn they called for diverse branches of mathematics, in algebra and in logic. In the same move, the computer, its software and the structure of the information they handled, became attractive topics for scholars from different fields, from formal linguistics to graph or information theory. In other words, computer scientists broadened the scope of their scientific interests, while computing attracted a growing population of researchers with various intellectual agenda.

At this stage, computing began to gain autonomy from applied mathematics, as a few militants endeavoured to promote it as a new discipline. While they could not hope yet to impose it at the core of the academic system, they followed what we may consider a 'peripheral strategy'. Beside expanding computer laboratories

and chairs in a handful of universities, they created learned societies (*Association franaise de calcul et de traitement de l'information*, 1962); grouping French computer experts from all professional backgrounds, these societies published journals and organized annual conferences, discussed about standards and good practices, and participated in international organizations, thus giving computing many characteristic features of a 'normal science'. From 1963 they also used a governmental agency dedicated to technology policy, DGRST, where they created a computing committee to fund their research projects and to spread their views throughout the administration. Beyond their scientific arguments, their most convincing assets were the growing need for computing power and the massive demand for trained computer engineers in the nation's economy, which required that higher education invested in this field.

Tensions. At the level of practice, computer users and researchers (who began to call themselves *informaticiens* in the mid-1960s) were caught in a tension between two sets of problems:

- The immense, fascinating potential of applications and experiments which they perceived in computers, but whose practical achievement demanded vast R&D efforts in programming, in numerical analysis and in software methods.
- The material defects of first generations computers – poor reliability, minuscule memory size which limited programming capabilities, etc. – problems which were the business of the manufacturers' sales and technical agents, and were only practical constraints and motives of frustration, void of any intellectual interest, for academic *informaticiens*.

Another tension regarded the management of computing facilities, in which diverging interests tended to oppose computer scientists and users from other disciplines. Users simply needed a reliable, cheap and fast computing service. Yet computer scientists wanted to keep control on the machines, giving priority to their research and teaching tasks over the common users, whom they encouraged to learn Fortran or Algol to write their own programs and to run them on the computer in a 'self-service' mode.

In fact, computer scientists were caught in a contradiction. They had benefitted from the resources of their computing center – note that computer science was a rare case of a research field which had laboratories long before it was recognized as a discipline: computer science burgeoned in a corner of the computing center, and needed its resources to blossom. Yet they took great care not to identify themselves with the computing machine, in order to assert the theoretical nature of their investigations, as opposed to the technical nature of hardware and programs. If they wanted to promote computing as a science, they were

bound to accept the separation of research labs and computer centers. Such separation was progressively imposed by scientific authorities.

Enters mathematical logic. It was only in the mid-1960s that mathematical logic, particularly the foundational works on computability published 30 years earlier, appeared commonly in the references and bibliographies of publications by French computer scientists. Why such a late 'discovery', instead of a linear process which would have proceeded from mathematical theory to computing technology – from the abstract Turing machine to material computers? Three interrelated explanations can be offered:

1. logic had been eclipsed from the French mathematical scene since Jacques Herbrand's premature death in 1931, and nearly banned from mathematics by the Bourbaki group;
2. until the early 1960s, computer experts were focused on solving technical problems or developing numerical analysis, so that mathematical logic made little sense to them;
3. however in the mid-1960s, in a rapidly evolving context, they felt the need to better understand what they were doing, to cope with the growing complexity of computer systems, to formalize what they were teaching in order to elaborate informatics curricula and to assert the scientific 'nature' of computing. It was only under these constraints that the foundational works on computability published in the 1930s made sense to them.

The institutionalization of computer science was a gradual, controversial endeavour, marked by visible milestones: the founding of the first computer departments (1964), the creation of master's diploma in *informatique* (1966), the setting up of specific committees for computer science at national level in the university system (1972) and at the CNRS (1975).

From then on, computer science was institutionally stabilized for a quarter of a century. It took then about as much time to be recognized at the Académie des Sciences and at the Collège de France, the two highest institutions of French science, where elections rest exclusively on scientific values, not on socio–economic pressure. After 2000, computer science conquered wider academic recognition; with some 800 full professors and 2.000 associate professors in 2010, it became the major discipline in French higher education by the number of faculty staff.

References

[1] Mounier-Kuhn, P. (2010). *L'Informatique en France, de la seconde guerre mondiale au Plan Calcul. L'émergence d'une science*, Presses de l'Universitè Paris-Sorbonne.

Artificial Testimony

Philip Nickel
Technische Universiteit Eindhoven, the Netherlands
p.j.nickel@tue.nl

Artificial testimony consists of assertions, offered to a hearer as a source for belief, but created by a technological artifact and having as a whole no single human source. In this paper I consider the implications of artificial testimony for philosophical theories of testimony and its justification. Here are a few examples of artificial testimony:

- An on-board automobile navigation system tells you that you have arrived at your destination;
- A robot reports its location in your house and its upcoming movements [5];
- A monitoring system gives a physician a morning update about the condition of an infant in a pediatric intensive care unit [7].

These utterances are remarkable because they are more than just human utterances delivered via technology. They have an original linguistic content generated by an artifact.

I begin by sketching an account of linguistic agency that is broad enough to accommodate computer speech. A *speech actant* is the authorial source of a linguistic message, where this is neutral between human speakers and other original sources of linguistic messages. Some computers such as those mentioned above have the power to create or assemble somewhat original messages adapted to new contexts. The things they say on particular occasions are not fully mechanically foreseeable by their designers, manufacturers or owners. (Note that they might be 'conversationally' predictable, in something like the way that human speech is sometimes predictable. But this is not a sign of failure to manipulate language authorially.) As technologies have become more capable of integrating observations of reality, complex symbolic representations (e.g., maps, databases, or images), make word choices, and assemble grammatical sentences, they have become more autonomous and spontaneous in producing spoken and written messages [4]. Background machine intelligence enhances these capacities. Such technologies are linguistic actants, capable of authoring artificial speech.

I then go on to explain why these phenomena raise problems for what I call intentionalist accounts of the epistemology of testimony. One of the goals of an epistemology of testimony is to explain why one can gain knowledge or warranted belief by relying on the assertions of others. Philosophical accounts of

testimony are often framed in terms of the relations between persons, or as depending on mental attitudes like beliefs or intentions that are normally thought only to be possessed by persons. For example, Paul Faulkner holds that beliefs about speakers' intentions (e.g., about their sincerity, or on a later view, their responsiveness to affective trust) are central to one's warrant for accepting their testimony [1, 587]; [2, 899]. Jennifer Lackey discusses a family of "Belief Views of Testimony" which she attributes to a score of prominent contemporary epistemologists [3]. On belief views of testimony the hearer's warrant for testimony depends on the warrant the speaker has for the belief expressed by that testimony, as expressed for example in Alvin Plantinga's claim that "a belief on the part of the testifiee has warrant only if that belief has warrant for the testifier" [6, 86], quoted in [3, 77]. If we allow that artificial testimony is testimony, this appears to fit poorly with intentionalist accounts. Since artificial linguistic actants do not have beliefs, on these belief-based views it is impossible to have a justified testimonial belief on the basis of artificial speech.

I discuss several possible responses to these problems. The first is to cast the intentional net wider, extending the conception of *whose intentional states count* for the sake of establishing testimonial warrant. Specifically, an account of testimonial warrant can make reference to the intentional mental states of the designers and/or deployers of artificial linguistic actants, rather than those of the linguistic actant itself. This strategy could be applied to belief and allied cognitive states, or to intentions and other conative states. The second response focuses instead on the reliability of the utterances themselves, or the system that produces them. However, reliabilist accounts of testimony must be able to explain what counts as a relevant item to be evaluated for reliability. This poses special problems in the case of artificial testimony. In the end, I will argue for a view that combines an intention-based condition with a proper function condition implying reliability.

In the concluding section I discuss the importance of the design of "articulate machines" in relation to the warrant we have for believing them, and I consider an argument from Bernard Williams that the capacity for sincerity and insincerity (not possessed by most computing machines, it would seem) is of fundamental importance to how we think about the reports of such machines.

References

[1] Faulkner, P, (2000). 'The Social Character of Testimonial Knowledge', *The Journal of Philosophy* 97, 11, pp. 581?601.

[2] Faulkner, P, (2007). 'On Telling and Trusting', *Mind* 116 (2007): 875?902.

[3] Lackey, J. (2006). ‘Learning from Words’, *Philosophy and Phenomenological Research* 73, 1: 77?101.

[4] Nass, C., Brave, S. (2005). *Wired for Speech: How Voice Activates and Advances the Human-Computer Relationship*, MIT Press.

[5] MuDiS, ‘Multimodal Dialogue System’, Robotics and Embedded Systems group, TU Munich, `http://www6.in.tum.de/Main/ResearchMudis`. Accessed 20 April 2011.

[6] Plantinga, A. (1993). *Warrant and Proper Function*, Oxford UP.

[7] Portet, F., Reiter, E., Gatt, A., Hunter, J., Sripada, S., Freer, Y. & Sykes, C. (2009). ‘Automatic generation of Textual Summaries from Neonatal Intensive Care Data’, *Artificial Intelligence*, Volume 173 Issue 7-8, Pages 789-816.

[8] Williams, B. (1973). ‘Deciding to Believe’, in *Problems of the Self*, Cambridge: 136–151.

The computer between Computationalism and Cybernetics: the crucial role of Turing and von Neumann, and why they were ignored

Teresa Numerico
University of Rome 3, Italy
`teris@mclink.it`

It is difficult to underestimate the role of computability theory in the birth of computer science. The project of the computer as a stored-program electronic machine is strictly intertwined with the definition of the concept of algorithm, via an identification of it with all the processes that can be computed by a Turing Machine. These results were used (almost ten years later) as the theoretical basis for the logical structure of the computer (see von Neumann *First Draft* in [2], Turing’s *Proposed electronic calculator* in [1]).

The central role of Turing Machines as a general characterization of a mechanized process (a process that can be performed by the Machine) offered a frame and a clear explanation of what can be calculated and what cannot and relied on the complex concept of emulation, that later evolved in the broader concept of simulation.

The problem, however, is to establish the real nature of that simulation, the types of procedures that can be represented by the Turing Machine’s style devices; at what conditions those machines can emulate or better simulate intelligent behaviors and finally which is the nature of a so-called ‘intelligent procedure’. Is it related exclusively with the manipulation of symbols according to well-established

rules? Is it related to self-reproducing automata, devices capable of replicate themselves according to the management of complex organization layers? Is it linked with interactive capabilities of the machine to communicate with the human operator? Is it represented by the possibility of educate a network of unintelligent nodes, according to weight evaluation of the effect of positive or negative stimuli?

It is peculiar that both Turing and von Neumann were fascinated with the anticipation of different kinds of machines more related to cybernetic relational approach to mechanical devices, than to the computationalism centrality of the first years of development of computer science.

Turing was attracted by the English Cybernetics group called the Ratio Club ([4]) and later developed an intense interest for mathematical biology, known at the time as morphogenesis, while von Neumann was involved in the *Macy's Conferences* and was particularly interested in self-reproducing automata. These automata should have been capable of creating a complete copy of themselves with the help of the complexity of their inner structure.

As far as simulation was concerned, they showed a critical attitude about the interests in the computational device whose logic structure was a reproduction of the rules invented within the logical paradigm during the Thirties of last century, while were attracted by other paradigms of artificiality, better characterized by an integration of natural/artificial models.

Turing's and von Neumann's approaches were in tune with the Cybernetics vision of machine and their close relationship with natural organisms. At the beginnings the supporters of the different perspectives met the same conferences (see for example the *Macy's conferences* 1946-1953, see [5], the Dartmouth famous 1956 *Conference on Artificial Intelligence*, see [6] and the *NLP Mechanization of Thought Conference* held in 1958, see [3]), while the divergent visions were later stigmatized by the establishment of the so-called scientific agenda about which were the major objectives, and the best methods to reach them. However Turing and von Neumann belonged by default to the AI group, because of their contributions to the theoretical basis of computer science, their later more critical contributions (see [7], [8], [9]) were almost ignored, until recently, by the scientific community. At the core of the dispute between AI and Cybernetics, according to me, there were not only the different metaphors of the computer as a simulator respectively of the mind or of the brain – as the conflict was normally described by historians – but a dissimilar perception about the relationships of the device with logic and of the eventual organic integration between the machine and the human operator. The computational AI supporters considered language as a tool for representing information, while Cyberneticians considered language as a tool for communication and as the basis for the human-machine

integration. Viewed in this perspective the center of the discrepancy was concerned with the role of simulation in machines. According to Cyberneticians mechanical devices and organisms were similar with respect to communication and control practices, while AI scientists such as John McCarthy, Marvin Minsky, Herbert Simon etc. wanted to simulate the intellectual capabilities of the human mind by a symbolic representation of knowledge and its organization inside the machine. They thought in terms of automation, while Wiener and the other members of the Cybernetics group (including Turing and von Neumann, though with different twists) were concerned with feedback and other interactive mechanisms, both in natural organisms and automata that allowed a better integration of the different agents of the communication processes.

References

[1] Copeland J. (2004) (Ed.) 'Proposed electronic calculator by Alan Turing' in *Alan Turing's Automatic Computing Engine*, Oxford University Press, Oxford; http://www.alanturing.net/proposed_electronic_calculator/.

[2] Godfrey M. D. (1993) 'The First Draft Report on the EDVAC' by John von Neumann, *IEEE Annals of the History of Computing*, Vol. 15, No. 4, pp.27-75, http://qss.stanford.edu/~godfrey/vonNeumann/vnedvac.pdf.

[3] Kline R. (2010) 'Cybernetics, Automata Studies, and the Dartmouth Conference on Artificial Intelligence,' *IEEE Annals of the History of Computing*, 26 May. 2010. IEEE computer Society Digital Library. IEEE Computer Society, http://doi.ieeecomputersociety.org/10.1109/MAHC.2010.44.

[4] Holland O., Husbands P. (2011) 'The origins of British cybernetics: the Ratio Club', *Kybernetes*, Vol. 40 Iss: 1/2, pp.110 - 123.

[5] Pias C. (2003) (Ed.) *Cybernetics. The Macy Conferences 1946-1953 - Transactions*, Diaphanes, Zurich- Berlin.

[6] Shannon C.E., McCarthy J. (1956) *Automata Studies*, Princeton University Press, Princeton (MASS.).

[7] Turing A. M. (1948), 'Intelligent Machinery Report', National Physics Laboratory, in B. Meltzer D. Michie (Eds.), *Machine intelligence*, 5, Edinburgh Univ. Press, 1969: 3-23; reprint in [1, 410-432].

[8] von Neumann J. (1948/1961) 'General and logical Theory of automata', Hixon Symposium, reprinted in Taub A.H. (ed) *Collected Works*, Pergamon Press, Oxford 1963, Vol.V: 288-328.

[9] von Neumann J. (1958) *The computer and the brain*, Yale Univ. Press, New Haven.

Making and remaking the Statistical Tables for Biological, Agricultural and Medical Research

Giuditta Parolini
CIS – International Centre for the History of Universities and Science
Department of Philosophy, University of Bologna, Italy
giuditta.parolini2@unibo.it

Table making deserves a place of its own in the history of computing and information technologies. As pointed out by Campbell-Kelly et al. [1]], the historian's challenge is not to present the tables – mathematical, statistical or astronomical – as a static artefact, but to make sense of them as an instrument shaped by its makers and able to shape the communities of its users.

The *Statistical Tables for Biological, Agricultural and Medical Research* [2], published for the first time in 1938 and co-authored by the statisticians Ronald Fisher and Frank Yates, represent an interesting case study in this perspective. Throughout the following forty years these tables were used as a computing tool for the application of analysis of variance and experimental design, the statistical methods developed by Fisher during the 1920s, while chief statistician at the agricultural research station of Rothamsted.

I want to argue that the *Statistical Tables* were not a value-free collection of numbers, but that they were planned and computed as an instrument for the dissemination of Fisher's statistical methods in biology, agriculture and medicine. In so doing they embodied a form of power and authority and in this sense I will call them a political artefact, after Langdon Winner [3].

The publication of a book of statistical tables was not a novel event in Britain. Since 1914 the Biometric Laboratory headed by Karl Pearson at University College London had issued a collection of tables for statisticians and biometricians [4]. But the *Statistical Tables* made a clean break with this tradition focused on correlation and curve fitting. In Fisher's and Yates's book the balance is shifted towards significance testing – at the beginning of the collection there are the tables of Student's, chi-square and z distribution used for making tests of significance – and the randomised design of experiments is presented as integral part of a statistical approach offering *ad hoc* instruments, such as Latin squares and random numbers, in order to apply it.

Fisher himself, head of the Galton Laboratory at University College London during the 1930s, led in person the making of the *Statistical Tables* along with Frank Yates, at first his assistant at Rothamsted and from 1933 his successor as head of the local statistics department. The first edition of the book was a slim volume of 90 pages with 34 tables preceded by a lengthy introduction that explained how

the tables had been computed and how they should be used. From 1938 up to the early 1960s a new edition of the book appeared more or less every five years.

The collection had a self-consistent structure. Ideally all the tables needed for the application of analysis of variance and experimental design to agriculture, biology and medicine were included. Since the first edition the book was sold at a modest price, affordable even for users like agronomists, biologists and physicians that devoted only a small amount of their research budget to tools for scientific calculations. Moreover, the tables were planned and computed in such a way that a research worker supplied only with a slide rule or a poor desk calculator could work through his or her data using the tables.

Langdon Winner has pointed out that artefacts can have politics, i.e. they can have embedded in their design forms of power and authority not explicitly declared. In evaluating technologies he suggests it is necessary to go beyond their immediate use and examine "whether a given device might have been designed and built in such a way that it produces a set of consequences logically and temporally *prior* to any of its professed uses" [3]. The artefacts that Winner has in mind are made of concrete and steel, but computing instruments like statistical tables should not be exempted from this scrutiny. They cannot be considered neutral just because they are the outcome of mathematical knowledge and number crunching. Quite the opposite, they should be carefully cross-examined because tacit aims and goals are not as evident in a collection of numbers as they may be in a material artefact.

The many research workers and statisticians who bought the *Statistical Tables* as a computing tool were sold at the same time a peculiar vision of statistics. The collection of tables had been planned as a complement to Fisher's main textbooks on statistics, *Statistical Methods for Research Workers and the Design of Experiments*, and the choice to tabulate the probabilities in the tables only for selected values contributed to spread the 5 per cent threshold for statistical significance that Fisher had recommended in *Statistical Methods*, linking further statistical computing and statistical theory.

Moreover, unlike Karl Pearson, Fisher and Yates adopted a liberal policy in relation to their computing effort. Statisticians and research workers that approached them and their publisher, Oliver and Boyd, with requests for reprinting tables – often the ones of the chi-square and Student's distribution – were usually authorised subject to a proper acknowledgement. Endless is thus the number of publications in which tribute was paid to Fisher and Yates and in which their tables were replicated strengthening further the authority of the original collection.

I claim that the careful planning of the *Statistical Tables* and the liberal idea of copyright endorsed by their authors made the book not only a successful comput-

ing tool, but also a political technology in which a commanding vision of statistics was embedded. In order to give evidence for my argument I will examine the structure of the collection of tables, its making throughout six editions, and its reception among research workers and statisticians who provided advice to agronomists, biologists and physicians.

References

[1] Campbell-Kelly, M., Croarken, M., Flood, R. and Robson, E. (2003), *The History of Mathematical Tables: From Sumer to Spreadsheets*. Oxford: Oxford University Press.

[2] Fisher, R. A. and Yates, F. (1938), *Statistical Tables for Biological, Agricultural and Medical Research*. Edinburgh: Oliver & Boyd.

[3] Winner, L. (1980), 'Do Artifacts Have Politics?', *Daedalus* 109 (1): 121-136. Quotation p. 125.

[4] Pearson, K. (1914), *Tables for Statisticians and Biometricians*. Cambridge: Cambridge University Press.

Computing the Infinite

Sam Sanders
Department of Mathematics, University of Ghent, Belgium
Mathematical Institute, Tohoku University, Sendai, Japan
sasander@cage.ugent.be

Computability and Arithmetic. Intuitively, a subset of $\mathbb{N}$ is *computable* if there is a Turing machine (finite procedure, idealized computer, algorithm,...) that fully describes this set. An example of a *non-computable* set is the *halting problem*. The latter procedure just determines whether a given Turing machine halts for a given input. See [7] for an introduction.

Emil Post's celebrated theorem ([7, Theorem 2.2]) connects first-order logic and computable sets. We consider the fourth item.

Theorem 1 For $n \geq 0$, we have

$$A \in \Delta_{n+1} \Leftrightarrow A \leq_T \emptyset^{(n)}.$$

Thus, a set A is computable if and only if A is Δ_1. Moreover, a set A is Δ_2 if and only if A can be computed on a Turing machine with a finite number of queries

to the halting problem. The *limit lemma* ([7, Theorem 3.3]) provides another formulation of Δ_2 sets.

Theorem 2 A set satisfies $A \leq_T \emptyset'$ if and only if A is *limit computable*, i.e. there is a computable sequence f_m s.t. $A = \{n \in \mathbb{N} : \lim_m f_m(n) = 1\}$.

Computability and Nonstandard Analysis For an introduction to Nonstandard Analysis, we refer to [2]. For our purposes, it suffices to know that, using techniques from Logic, the set $\mathbb{N}$ can be extended with extra elements that are larger than all $n \in \mathbb{N}$. The resulting set is called ${}^*\mathbb{N}$ and any number $\omega \in {}^*\mathbb{N} \setminus \mathbb{N}$ is called *infinite*.

In contrast, the original numbers $n \in \mathbb{N}$ are called *finite*. When working with $\mathbb{Q}$ instead of $\mathbb{N}$, the inverse of an infinite number is called an *infinitesimal*. It is straightforward to extend the domain and image of a function $f : \mathbb{N} \to \mathbb{N}$ to ${}^*\mathbb{N}$. We use the same symbol f to denote this extended function.

The following *transfer principle* plays an important role in Nonstandard Analysis.

Principle 3 (Π_1-transfer) For all $\varphi \in \Delta_0$, we have

$$(\forall n_1, \ldots, n_k \in \mathbb{N})\varphi(n) \to (\forall n_1, \ldots, n_k \in {}^*\mathbb{N})\varphi(n)$$

Note that Π_1-transfer is equivalent to several theorems of ordinary Mathematics (See [4, Theorem 2] and Theorem 9 below).

Definition 4 (ω-invariance) A function $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ is ω-invariant if

$$(\forall n \in \mathbb{N})(\forall \omega, \omega' \in {}^*\mathbb{N} \setminus \mathbb{N})[f(n, \omega) = f(n, \omega')].$$

A set $A \subset \mathbb{N}$ is ω-invariant if $A = \{n \in \mathbb{N} : f(n, \omega) = 1\}$ for an ω-invariant f. It can be shown that the value of an ω-invariant function is finite for $n \in \mathbb{N}$ and can be computed by a finite procedure. In particular, we have the following theorem. We refer to [6, 4, 5] for proofs.

Theorem 5 A set A is Δ_1 if and only if it is ω-invariant.

Thus, the classical notion of computability is captured exactly by ω-invariance. Furthermore, we have the following theorem.

Theorem 6 (Hyperlimit lemma) A set A is in Δ_2 if and only if A is *hyperlimit computable*, i.e. there is a computable sequence f_m such that, for all $\omega \in {}^*\mathbb{N} \setminus \mathbb{N}$,

$$A = \{n \in \mathbb{N} : f_\omega(n) = 1\}.$$

Corollary 7 The hyperlimit lemma is equivalent to Π_1-transfer

In light of Theorem 2, Π_1-transfer is closely related to the halting problem. As a matter of fact, Π_1-transfer provides an ω-invariant procedure to determine the truth of Σ_1-formulas.

Another connection between Π_1-transfer and the halting problem is given by the following, to be compared to Theorem 1.

Theorem 8 The Π_1-transfer principle is equivalent to the statement '*For every A in Δ_2, there is an ω-invariant set B such that $A = B$*'.

Thus, a Δ_2 set becomes computable if we have access to Π_1-transfer, and vice versa. This perfectly mirrors the situation in Theorem 1 for $\emptyset'$.

As an intermediate conclusion, we observe that Π_1-transfer and $\emptyset'$ exhibit the same (non)computable behavior. This close connection becomes even more interesting when we consider the following results from [4].

Theorem 9 The following[2] are equivalent to Π_1-transfer.

1. An ε-δ-continuous function is integrable over $[0, 1]$.
2. An ε-δ-continuous function attains a maximum over $[0, 1]$.
3. For bounded ε-δ-continuous f, there is a solution to $y' = f(x, y)$ on $[0, 1]$.

Thus, we observe that basic operations on ε-δ continuous functions, like integration, are *non*-computable. However, things change when we introduce the following notion of continuity.

Definition 10 A function f is *nonstandard* continuous on $[0, 1]$ if

$$(\forall x, y \in [0, 1])(x \approx y \to f(x) \approx f(y)).$$

Theorem 11 There is an ω-invariant procedure for the following operations.

1. Integrating a nonstandard continuous function over $[0, 1]$.
2. Finding the maximum of a nonstandard cont. function over $[0, 1]$.
3. Finding a solution to $y' = f(x, y)$ on $[0, 1]$ for bounded nonstand. cont. f.

[2] In [4], notions like 'integral', 'maximum', etc. are defined up to an infinitesimal. Exploring this topic further is beyond the scope of the current paper.

Thus, Nonstandard Analysis provides a 'more computable' framework for calculus and physics. Similar results are available for Bishop's 'constructive' analysis ([1, 6]).

Acknowledgment This research is supported by a grant from the John Templeton Foundation for the project *Philosophical Frontiers in Reverse Mathematics*. The opinions expressed here do not necessarily reflect the views of the John Templeton Foundation.

References

[1] Bishop, E., Bridges, D.S., *Constructive analysis*, Grundlehren der Mathematischen Wissenschaften, volume 279, Springer, Berlin, 1985.

[2] Goldblatt, R. *Lectures on the hyperreals*, Graduate Texts in Mathematics, volume 188, Springer-Verlag, New York, 1998.

[3] Robinson, A., *Non-standard analysis*, North-Holland Publishing Co., Amsterdam, 1966.

[4] Sanders, S., 'ERNA and Friedman's Reverse Mathematics', *J. of Symbolic Logic*, 2011, volume 76.

[5] Sanders, S., 'Reverse mathematics and non-standard analysis; a treasure trove for the philosophy of science', in Mitsuhiro Okado (ed.), *Proceedings of the Ontology and Analytic Metaphysics meeting, Keio University Press*, 2011.

[6] Sanders, S., 'A tale of three Reverse Mathematics', 2011, Submitted.

[7] Soare, Robert I., *Recursively enumerable sets and degrees*, Perspectives in Mathematical Logic, Springer-Verlag, Berlin, 1987.

Is Computer Science Made Scientific by its Experiments?

Viola Schiaffonati
Dipartimento di Elettronica e Informazione
Politecnico di Milano, Italy
schiaffo@elet.polimi.it
Mario Verdicchio
Dipartimento di Ingegneria dell'Informazione
Università degli Studi di Bergamo, Italy
mario.verdicchio@unibg.it

Computer-aided design of bridges and integrated circuits, the long established mapping of computable functions onto recursive functions, and the fundamental

role of computer simulations in the verification of models and theories in physics, chemistry, and biology are examples on how ubiquitous the concepts and the instruments of computer science have become in the fields of engineering, mathematics, and natural sciences. Despite such spread and versatility in scientific applications, the scientific status of computer science is still in question.

Computer science has been considered as the science of computers and related phenomena [7], thus supporting the conception of computer science as *artificial science* [9], namely an empirical study of the phenomena connected to computers viewed as artifacts. Contrary to such empirical interpretation, computer science has elsewhere been conceived as the study of theoretical notions, such as information and algorithms [2]. These concepts play a central role also in the work of Peter Denning, who advocates the scientific character of the discipline by shifting the focus from computers to computation and considering the latter as a domain distinguished from and with equal status as physics, society, and biology [3]. As these domains are the subject of physical, social, and life sciences, respectively, so is computation the topic of a specific science, namely, computing. Although agreeing with the author in considering computation as fundamental in his endeavors, we adopt an orthogonal point of view, searching for scientific legitimation not in the content of the discipline, but in its methodology.

If one of the defining criteria of scientific activity is experimental method, according to the empirical sciences' tradition, firstly we need to determine which of the applications of computer science can be legitimately viewed as experimental activity. Undoubtedly, in such analysis we need to take into account the debate on the status and the role of scientific experiments that has been going on in philosophy of science [8]. Then, we intend to understand the contribution of these experiments to the empirical nature of computer science, by using them as means to determine the position of this discipline with respect to the ones traditionally recognized as experimental.

There are two ways to interpret the activities of computer scientists as experiments:

- Programs (e.g. the above-mentioned computer simulations) are used as experiments to provide new data about physical systems, like atoms or galaxies, that are difficult or even impossible to investigate with direct observation [4];
- Iterated computer-based procedures can be used to arbitrate between competing models or hypotheses that are not dealing with a physical or social phenomenon, but with entities that are intrinsically related to computer science, like algorithms and programs.

We consider the former case as out of our scope: it provides an instrumental

view of computer science as *infra-science*, according to which the discipline provides new and better instruments for experimentation in the existing sciences. From this perspective, a computer simulation of a galaxy is an up-to-date experiment in astrophysics, but provides no support in analyzing the scientific status of computer science.

Instead, we focus on the latter activities, aimed at working with topics within the scope of computer science, meant as the discipline of automatic computation and information processing. Although in computer science, and computer engineering, the activities called experiments share some common features, there are also impressive differences that make it difficult to have a common definition of experiment. If we consider algorithms, a certain level of rigor in conducting experiments can be evidenced [5], as some principles of experimental methodology, such as *comparison*, *reproducibility*, and *repeatability*, are concretely declined. For instance, in software engineering and software testing in particular, pieces of code are empirically probed following standardized procedures that run them on a set of preselected input values and compare the results with the expected outcome, to search for conditions under which the examined items do not function properly [6]. Still, there exist other subfields of computer science, like autonomous mobile robotics, in which experimentation has not yet reached a comparable level of maturity. Despite the recognized importance of experimental approaches, these ideas have not yet become really part of the current practice, due to the weak awareness of experiments as fundamental elements in the development of a robotic system [1].

The examples above show a certain degree of heterogeneity when speaking of experiments in computer science. Rather than a single experimental method, computer science is characterized by different methodologies for each subfield, where experiments are performed not to confirm a general theory, but to test whether a given system works appropriately. Our working plan is to analyze the production of artifacts in the form of algorithms and programs, and focus on the experimental side of this process, searching for common features in the several subfields of the discipline. We aim at shedding some light on the possibility, despite their significant differences, to individuate general experimental principles in the subfields of computer science in the direction of a more rigorous approach typical of mature scientific disciplines.

Acknowledgement. This work was partially supported by MIUR in the framework of the PRIN Gatecom project.

References

[1] Amigoni, F., Reggiani, M., Schiaffonati, V. (2009). 'An insightful comparison between experiments in mobile robotics and in science', *Autonomous Robots*, Springer, 27(4), pp. 313-325.

[2] Colburn, T. (2004) 'Methodology of Computer Science', in Floridi, L. (ed.), *Philosophy of Computing and Information*, Floridi, L. (ed.), Blackwell, pp. 318-326.

[3] Denning, P. (2009). 'Computing: the Fourth Great Domain of Science', *Communications of the ACM*, 52 (9), pp. 27-29.

[4] Humphreys, P. (2004). *Extending Ourselves. Computational Science, Empiricism, and Scientific Method*, Oxford University Press.

[5] Johnson, D. (2002). 'A Theoretician's Guide to the Experimental Analysis of Algorithms', in Goldwasser, M.H., Johnson, D.S., McGeoch, C.C. (eds.), *Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges*, American Mathematical Society, Providence, pp. 215-250.

[6] Meyers, G. (1979). *The Art of Software Testing*, Wiley & Sons.

[7] Newell, A., Perlis, A., Simon, H. (1967). 'Computer Science', *Science*, 157 (3795), pp. 1373-1374.

[8] Radder, H. (2003). *The Philosophy of Scientific Experimentation*, University of Pittsburgh Press.

[9] Simon, H. (1969). *The Sciences of the Artificial*, MIT Press.

Mechanical brain in the XIX century: Logical machines of Alfred Smee

Valery V. Shilov
MATI - Russian State Technology University
shilov@mati.ru
Vladimir V. Kitov
Institute of History of Natural Sciences and Technics Russian Academy of Science
vladimir.kitov@mail.ru

In the middle of the XIX century the reputed English scientist Alfred Smee (1818-1877) developed the theory, which he called electrobiology. Its purpose was to study the effect of electrical phenomena on the functioning of the human body. Smee was primarily interested in the connection of electrical stimulation of the nervous system and brain work. Later his research in this area became

more philosophical than natural-scientific, and he published the book *Process of Thought Adapted to Words and Language* [1], in which in particular he proposed a plan to build an artificial system of mental conclusions modeling inner mechanisms of human brains. Though Smee based the knowledge of these mechanisms (at the time virtually unknown) on his own ideas of what they might be. Later this small treatise with a few additions was included into the monograph *The Mind of Man: Being a Natural System of Mental Philosophy* [2].

According to Smee, the mental image showing concepts or objects of the external world arose in the minds of humans due to electrical effects on the elements of the nervous system (nerve fibers), and each image was represented by some combination of fibers. This combination was stored, and later when the same object was being recognized, it was retrieved from memory. According to William Hamilton (1778-1856), proposition expressed a comparison of concepts or objects, so to make judgments means to recognize the relations of agreement or disagreement between two concepts, two separate subjects or a concept and a separate subject, comparing them with each other. Smee tried to go further: "... it occurred to me that mechanical contrivances might be formed which should obey similar laws [*laws of thought* – (note of the authors)], and give those results which some may have considered only obtainable by the operation of the mind itself" [2, 39]. To find relations between concepts (i.e. actually for simulating the operation of brains), he proposed to use two original mechanical logical machines and described them.

Relational machine was intended for presentation and comparison of concepts based on the so called *geometric* series: "In order to induce a general law from specific instances, and deduce the application of a law to a particular case by means of mechanical contrivances, we must take advantage of the geometrical arrangement of words formerly described, and denote each word by a cipher, and lastly then arrange them in such a manner that each cipher may bear its proper relation to every other cipher" [2, 40], see Fig.1.

Unfortunately, his explanations are very concise and unclear. In general, one can understand that Smee conceived his device as a collector of knowledge, with capability to add new facts, concepts, etc. Based on rules derived from the laws of thought, its internal state changed, taking into account and showing all the relations between new and previously presented concepts. If this collector of knowledge were universal and all-inclusive, then, according to Smee, "it is thus apparent that this mechanism gives an analogous representation of the natural process of thought, as a human contrivance can well be expected to afford" [2, 44].

Of course, Smee knew that it was impossible to implement his idea in full scale: "Supposing that the machine could be made sufficiently extensive for all prac-

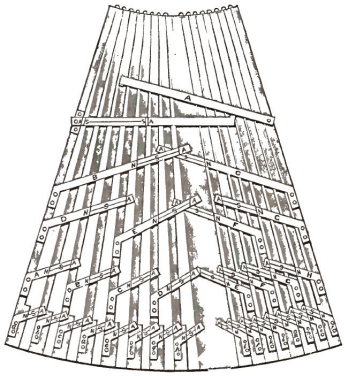


Figure 1: Relational Machine.

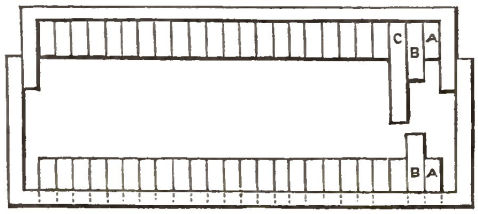


Figure 2: Differential machine.

tical purposes, yet the labour of employing it would be so great, that persons would soon rely upon the abilities which it has pleased Providence to give to them, and not seek assistance from extraneous sources" [2, 45].

The second machine, proposed by Smee – *Differential machine*, was intended to compare two concepts. The machine was a two-piece rectangular frame. One of the parts had a slightly shorter length, and therefore could be slided into the second part. Along the long sides of the frame rectangular bars of different lengths were laid, representing the properties of some concepts. Bar of one unit height corresponded to the presence of property, and a bar of 2 units height – its absence. At the top of the other frame for the presentation of other terms bars of height of 4 units were used, meaning the absence of property (see Fig.2).

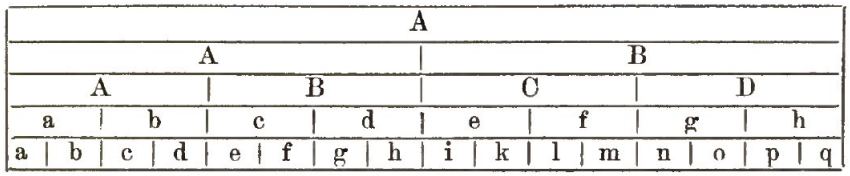


Figure 3: Relational slate.

For comparison of concepts Smee used four degrees of coincidence. If you moved the two parts of the frame and tilted her, the bars representing the relevant properties became connected. If their total height was 2 $(1 + 1)$, then this property was common to both concepts, if equal to 3 $(1+2$ or $2+1)$, then we could conclude that the coincidence of the properties was likely to occur, if 4 $(2 + 2)$ the conclusion that the coincidence of the properties was possible. If the total height was 5 $(1 + 4)$, then we concluded the denial of the coincidence, etc. Smee supposed that such device could be used in many cases. For example, he cites examples of comparisons of testimonies and decided that "the mechanical judge" in the civilized world would produce a sensation.

It is also worth mentioning *Relational slate* – the "didactic logical tool", first described in Smee's book in 1875 which was intended for use in the study of logic. It looks like a box with multiple compartments or just appropriately lined table (Fig. 3). Depending on the dependencies of analyzed concepts, they were placed in one or another compartment, after which a conclusion was deduced. Smee explained that bars, representing concepts with the same meaning, were placed in one compartment, bars, corresponding to concepts that were parts of other concepts, were put under respective compartment, etc.

Books of Smee gained some popularity among psychologists and physiologists, but were not recognized by mathematicians and logicians. So, Stanley Jevons (1835-1882) learned the ideas of Smee only after invention of his own logical machines. Jevons said: "So far as I can ascertain from the obscure descriptions and imperfect drawings given by Mr. Smee, his *Relational Machine* is a kind of

Mechanical Dictionary, so constructed that if one word be proposed its relations to all other words will be mechanically exhibited. The *Differential Machine* was to be employed for comparing ideas and ascertaining their agreement and difference. It might be roughly likened to a patent lock, the opening of which proves the agreement of the tumblers and the key" [3, 517].

Perhaps the unclear descriptions of Smee's logical devices mentioned by Jevons prevented their further investigation. For example, Martin Gardner in his classic work *Logic Machines and Diagrams* only mentioned them, calling the book of Smee "strange" [4, 144]. So the paper presented is the first attempt to describe and to analyze Smee's machines in the context of the history of logical machines. We also want show the evident similarity of Smee's devices with "intelligent machines" of the Russian inventor Semen Korsakov (1832) and the "logical machine" of William Hamilton (c1840).

References

[1] Smee A., *The Process of Thought Adapted to Words and Language, Together With a Description of the Relational and Differential Machines*. L.: Longman, Brown, Green, and Longman, 1851. xvi+77 p.

[2] Smee A. *The Mind of Man: Being a Natural System of Mental Philosophy*. L.: George Bell and Sons, 1875. xx+262 p.

[3] Jevons W. S., 'On the Mechanical Performance of Logical Inference', *Philosophical Transactions of Royal Society*. Vol. 160. 1870. pp.497-517.

[4] Gardner M., *Logic Machines and Diagrams*. N.-Y., Toronto, L.: McGraw Hill Book Co, 1958. 158 pp.

The Max Newman Collection of Alan Turing's Offprints: a bibliographical enquiry.

Julian Wilson
Associate Director Christie's
jwilson@christies.com

The present paper considers the Max Newman collection of Alan Turing's offprints (now housed at Bletchley Park) from a bibliographical perspective, and asks several questions: what are they? what is their significance? and why are they so rare? In the course of answering these questions, this paper will demonstrate the bibliographic importance of Turing's offprints, describing the nature

of their priority over Turing's published work in periodicals and journals, and show why they should be considered as primary source material in the history of mathematics and computer science. This paper will also discuss the importance of the provenance of the collection.

What is an offprint?

The Oxford English Dictionary gives the definition: "A separately produced copy of an article, etc., which originally appeared as a part of a larger publication." Bibliographically speaking, this is inaccurate, and has led to much confusion. A more accurate definition might be: "A separately produced copy of an article, etc., which appears as a part of a larger publication; particularly those articles which are printed as proofs for the author to correct before final printing of the larger publication." Offprints in this latter sense have priority over the main publication, and can have important ramifications for historians of science concerned with establishing the order of publication.

Offprints of this type may often be identified from "extracts" or later separate re-printings of individual articles by such features as textual variance, different pagination, changes to drophead-titles, separately printed wrappers, or an individual binding, which in the case of nearly all of Turing's offprints, means stapling.

In the case of "On Computable Numbers", the offprint has the same pagination as the final publication of the article that appeared in Proceedings of the London Mathematical Society, ser. 2, vol. 42. London: November 12th 1936. However, it has a separately printed half-title, and separately printed olive-green wrappers, with the whole stapled.

What is their significance?

Although Turing had submitted his typescript for "On Computable Numbers" to Max Newman in April 1936, Alzono Church had pre-empted his claim to a solution of the Entscheidungsproblem in the paper "A note on the Entscheidungsproblem", published in Journal of Symbolic Logic on 15th April 1936. Nonetheless, Newman, impressed by the sheer originality of Turing's work, persuaded Turing to publish. Turing took the opportunity to add an appendix to his paper with reference to the work of Church and Kleene which led to a delay of several months - the appendix is dated 28 August. The offprint was printed in either September or October 1936, with the final paper being ready for publication on 12th November 1936, with the whole volume becoming available in January 1937.

In October 1936, Emil Post submitted a paper to Church for publication in the Journal of Symbolic Logic which approached unsolvable problems from the perspective of a “mindless worker” receiving instruction notes. Because his offprint was already printed, Turing pre-empted Post.

In the case of Max Newman’s copy of “On Computable Numbers”, it was also a chance for Newman, the mentor, to proof and introduce corrections before finally submitting to press. Thus Max Newman’s own copy of Turing’s "On Computable Numbers" not only provides the historian with a "first draft" with textual variance, but allows the historian to see how Newman intervenes in the publication history and determine his input. In this sense, the offprint becomes primary source material for the historian of mathematics and computer science.

Why are they so rare?

Turing’s offprints are extremely rare in institutional holdings and in commerce, with only Bletchley Park, the Turing Archive at Kings College, Cambridge, and a private collection having significant holdings. Over the past 30 years there have been a reasonably large number of opportunities to acquire offprints published by other eminent 20th-century scientists, such as Albert Einstein, Alexander Fleming, and James Watson and Francis Crick. This paper proposes four reasons why Turing’s offprints, by comparison, are extremely rare:

1. Offprints are printed in very small batches. A print run for an offprint is usually 20-25 copies only.
2. They are ephemeral. Designed as proofing tools only, these offprints were not encased in stout bindings, nor printed on superior paper to ensure a degree of longevity.
3. Turing’s offprints were not widely dispersed. "On Computable Numbers" was published when he was an unknown postgraduate, and whose work had been pre-empted by Church. Consequently the reaction from the mathematical world was muted.
4. Turing’s personal and professional life was chaotic. The Second World War, and his work at Bletchley Park, interrupted what should have been a routine academic career path which would have provided more opportunities to spread the word of his achievements. The Official Secrets Act prevented many people outside of Turing’s close-knit circle of personal friends (who tended to be colleagues) of being aware of the practical application of his theoretical mathematics. And anyway, it was not in Turing’s character to press offprints of his work onto those incapable of understanding it. In the wake of his tragic death, any caches of offprints that he might have retained were probably swept away.

Conclusion

The Max Newman Collection of Alan Turing’s offprints allows historians of mathematics and computing science to establish priority of publication, investigate patterns of dispersal, and see how Newman was involved in the publication process of "On Computable Numbers". The collection owes its survival to being in the hands of one of Turing’s closest colleagues, friends and supporters.

Conference and Dinner Location

The conference will take place in *Zaal Rector Vermeylen* at *Het Pand*,

Het Pand
Onderbergen 1
9000 Gent
tel. 09 264 83 05

an old Dominican monastery located in the heart of the city on the banks of the river Leie, near the medieval port with the guildhalls as its remnants.

How to reach *Het Pand*:

- From the railway station *Gent Sint-Pieters*: Take tram 1 at the railway station (every 6 minutes) direction 'Centrum' (follow the directions for "Lijn 1 Centrum"). You should get off at the stop 'Korenmarkt' (9th stop) and cross the river Leie via the bridge 'Sint-Michielshelling'. Turn around the Sint Michaels church. *Het Pand* is situated next to the church.
- By car: Entering Ghent from the E40 or the E17, follow the parking-route to P7, 'Sint Michiels Parking'. This parking is only 50 m away from *Het Pand*.

The Dinner will take place at *Het Pand* as well on Wednesday, 9th of November starting at 19.00h.

Internet Access at the Conference Location

Make a wireless connection with "UGentGuest". If you have set up to request an IP address automatically, you will receive an IP address starting with 193.190.8x.

Now you are connected, but not yet authenticated. You should start a web-browser and you will be redirected to a logon screen. Enter the following user-name and password:

Username: XXXXX

Password: XXXXX

After correct authentication you can use the Internet connection.

Your connection to this wireless LAN is not encrypted To protect your personal data, please use encrypted connections like https, imaps, ssh etc. or a VPN client.

You’re not allowed to pass on the login information to others.