BRICS RS-94-1 G. Winskel: Semantics, Algorithmics and Logic

BRICS

Basic Research in Computer Science

Semantics, Algorithmics and Logic

Basic Research in Computer Science

BRICS Inaugural Talk

Glynn Winskel

BRICS Report Series **RS-94-1**

ISSN 0909-0878 **February 1994**

See back inner page for a list of recent publications in the BRICS Report Series. Copies may be obtained by contacting:

BRICS
Department of Computer Science
University of Aarhus
Ny Munkegade, building 540
DK - 8000 Aarhus C
Denmark

Telephone: +45 8942 3360
Telefax: +45 8942 3255
Internet: BRICS@daimi.aau.dk

Semantics, Algorithmics and Logic

Basic Research in Computer Science

BRICS Inaugural Talk

Glynn Winskel

February 1994

Abstract

This is a transcript of a talk at the inauguration of BRICS, Basic Research in Computer Science, Centre of the Danish Research Foundation, on 2 February 1994 at the Steno museum, University of Aarhus, Denmark.

Whereas engineers draw from the Mathematics of Physics, the Mathematics of Computer Science comes largely from another tradition, that of Mathematical Logic. The relevant Mathematics forms three strands:

- Semantics of Computation
- Algorithmics
- Mathematical Logic

Roughly, these can be described as the ***What***, ***How*** and ***Why*** of Computer Science. Semantics of Computation answers the question ***what*** is Computer Science by providing mathematical models of computing systems. Algorithmics says ***how*** to do thing with computers by giving algorithms (or programs) to achieve tasks. While Mathematical Logic says ***why*** a method or system is correct by providing proof systems in which this can be established formally. Our particular concern in BRICS is to build-up strengths in Algorithmics and Logic coupled to our expertise in Semantics. In particular, Logic is a relatively undeveloped area in Denmark.

This addresses a general audience perhaps, with only a nodding acquaintance with computers. So here's a "one-line" definition of Computer Science:

> Computer Science is the management of complexity
> in the construction and analysis of systems.

Here "complex" is being used in its everyday sense to mean complicated, or huge. Many other areas share the concern of Computer Science in managing complexity. But this issue is particularly acute in the design and control of computing systems.

One way to see why is to observe that a single computer contains millions of transistors, each a physical device. In principle, we could model a computer as a physical device, as one big piece of Physics. But do so and we're lost! The key to managing this complexity is to structure our view of a computing system according to different levels of abstraction, and in this way ignore superfluous details. Here in Figure 1 is an illustration of the different levels of abstraction in a computing system.

At the lowest level of abstraction there is Physics. Ultimately a computer is built out of physical devices whose design and behaviour is understood in terms of their Physics. Just above is the Gate level in which transistors, which roughly behave as switches, are combined together in circuits to perform primitive functions. At the Hardware level these circuits are combined to form units performing arithmetical and logical operations. A computing machine, a computer, is formed by assembling the arithmetical and logical units together. At this level, the Machine level, the computer is addressed by instructions from a machine language. But this bit-moving language is far too finicky for many purposes. At the Program level, high-level programming languages allow us to write instructions using abstract mathematical concepts like sets and functions which are then compiled, or translated, down to instructions at the Machine level. Finally, the illustration shows the System level where several machines are linked together, perhaps via a satellite connection. At this level, we might be principally concerned with the pattern of communication between machines.

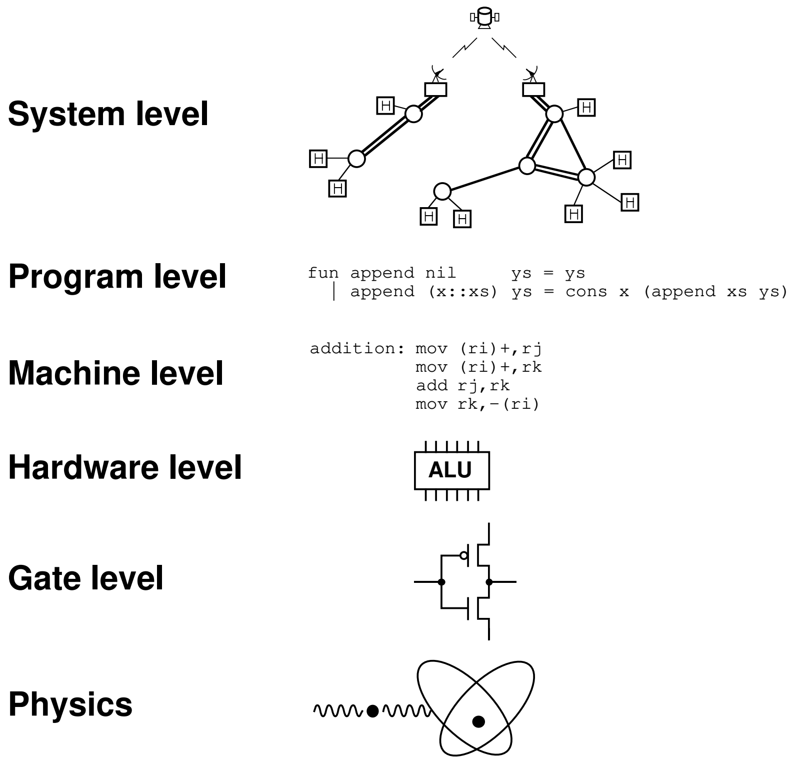

Figure 1: Levels of abstraction in a computer system.

Each level has its own basic units out of which more complicated structures are composed. At the Gate level the basic units are transistors, behaving essentially as switches, which are joined together as circuits. At the Program level the basic units are programming instructions which are juxtaposed to form a program. In implementing at a level in terms of that just below, it is sufficient to concentrate on showing the basic units are implemented correctly and that so is their means of composition. This spares us from an enormous amount of detail. It is ultimately why a computer designer doesn't need to know so much Physics! (Of course, designers are always on the look out for new technology which ultimately stems from Physics.)

Semantics of Computation provides the mathematical tools with which to model computing systems precisely. Its purpose is to give programming languages and systems a mathematical meaning which is abstract (ignoring as much detail as possible for the problem at hand) and obtained compositionally (in the sense that the meaning of a structure is obtained by composing the meanings of its parts). Its other task is to formalise the means to cross abstraction levels.

There are several different techniques in use in Semantics. One very successful approach is that of Denotational Semantics with its mathematical foundations in Domain theory. Its techniques are widely applicable from the Hardware level to certain aspects of the System level. Domain theory is founded in Topology, the Mathematics of approximation. How does this come about? In order to be sufficiently abstract the meanings of computing constructs are often taken to be infinite objects and computers can only work on their finite approximations. In the same way we compute with real numbers, often having infinite decimal expansions, in terms of their approximations up to finitely many decimal places. Semantics generally has been successfully applied to showing the correctness of compilers, programs, systems and hardware. But more than this, it has suggested new computing paradigms such as OCCAM, the programming language of the transputer, and functional programming. This is because while semantics shares with Physics the feature of providing mathematical models of reality, it differs significantly in one aspect. The reality it models is man-made. If that reality goes against the grain of Mathematics, that can sometimes mean we are modelling the wrong thing, that the system or programming language should be constructed differently.

A present limitation of Semantics is that it lacks systematic ways to go from the mathematical model to methods to reason economically about programs and systems. This is one area where Mathematical Logic is valuable.

Computer Science shares with Engineering the concern of building things. In the design of computing systems we must take account of what's feasible with respect to our computational resources, how much store we have in the computer, or how long we are able to wait. This leads Computer Scientists to consider another kind of complexity, which I'll first illustrate with an old story, one with which I'm sure you're familiar and which is drawn in Figure 2.

There was once a very rich, but rather naive sultan. A wise man had performed some service for him, and in exchange the sultan offered to grant the wise man's wish. Looking at the sultan's chessboard the wise man asked for enough rice that he could have one grain for the first square

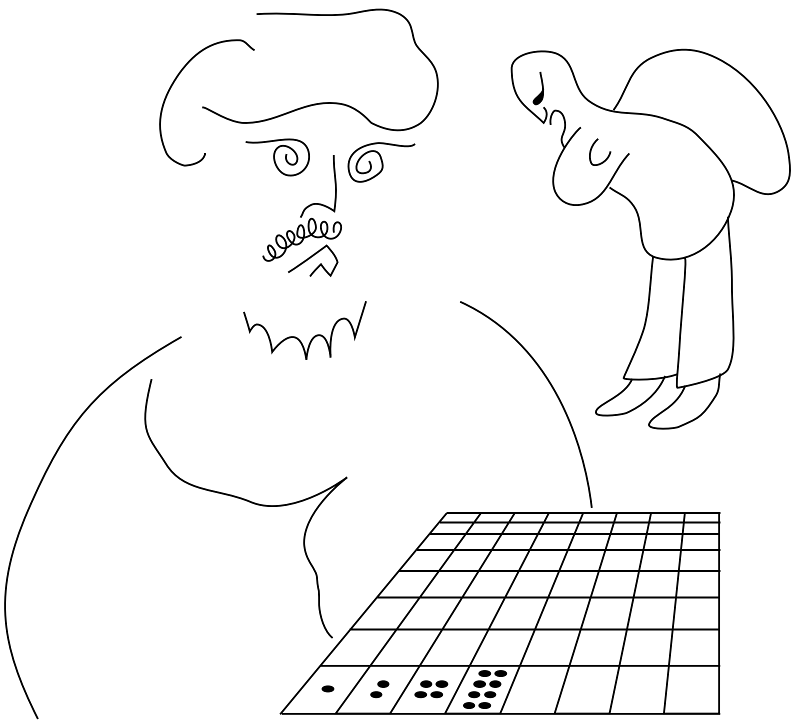

Figure 2: The naive sultan.

of the chessboard, two grains for the second, four for the third, and so on, doubling at every next square. The naive sultan agreed, rather too hastily, and sent his servant out for a sack of rice, little realising that he'd promised away millions of tons of rice!

What's this to do with Computer Science? Well, perhaps not so much. But just as the wise man's demand is more than enough to ruin the sultan, so can programs be catastrophic in their use of resources. After all, what does the wise man present the sultan with but a simple program, based on doubling in an iterative fashion. In a similar way a program can gobble up all the store in a computer, or, if it were to run to completion, take longer time than mankind will exist.

Algorithmics is concerned with computation within bounded resources. Its purpose is to

- design efficient algorithms,
- understand the relationship between different problems, how a program being a solution to one problem, reduces to a solution to

another,

- understand bounds on efficiency, and in particular when a problem does not have a feasible program as a solution.

Algorithmics employs a wide variety of techniques from traditional Mathematics, Logic and Probability theory. It has produced numerous efficient algorithms, many in day-to-day use. Even negative results, that certain problems do not have feasible programs, are important in subjects like cryptography and its applications, e.g. to data security. There it is precisely because a problem is hard that it is useful; just think of a simple combination lock — it should be hard for a thief to discover its combination!

A present limitation of Algorithmics is the inability to show certain problems do not have efficient programs as solutions. Many techniques are being tried, one approach being to reduce such questions to properties of logical systems.

Semantics and Algorithmics, like Computer Science itself, have their roots in Mathematical Logic, that branch of Mathematics formalising and studying properties of proof. Its history is sketched in Figure 3. Like so many things, the origins of Mathematical Logic can be traced back to the Ancient Greeks. Not only did they invent the central idea of mathematical proof, but they also began to consider specialised logics for reasoning, in particular beginning the study of Modal Logic, the logic of possibility and necessity. Arguably, the next great advances in Logic had to wait till the nineteenth century when there were significant discoveries in how to formalise Mathematics. This culminated in the work of mathematicians like Frege and Russell showing how to found Mathematics on a few primitive notions. Unfortunately, their work also brought forth paradoxes. Their foundations for Mathematics were not as secure as had been hoped. One reaction, that of Brouwer, was to try to find a new ontology for Mathematics, basing the meaning of mathematical statements on concrete constructions. Another reaction, that of Hilbert, was to retreat from the issue of what Mathematics meant. Instead, he hoped to side-step the thorny issues associated with the paradoxes by regarding Mathematics as taking place within a formal system, according to precisely defined rules, and in this way to reduce Mathematics, at

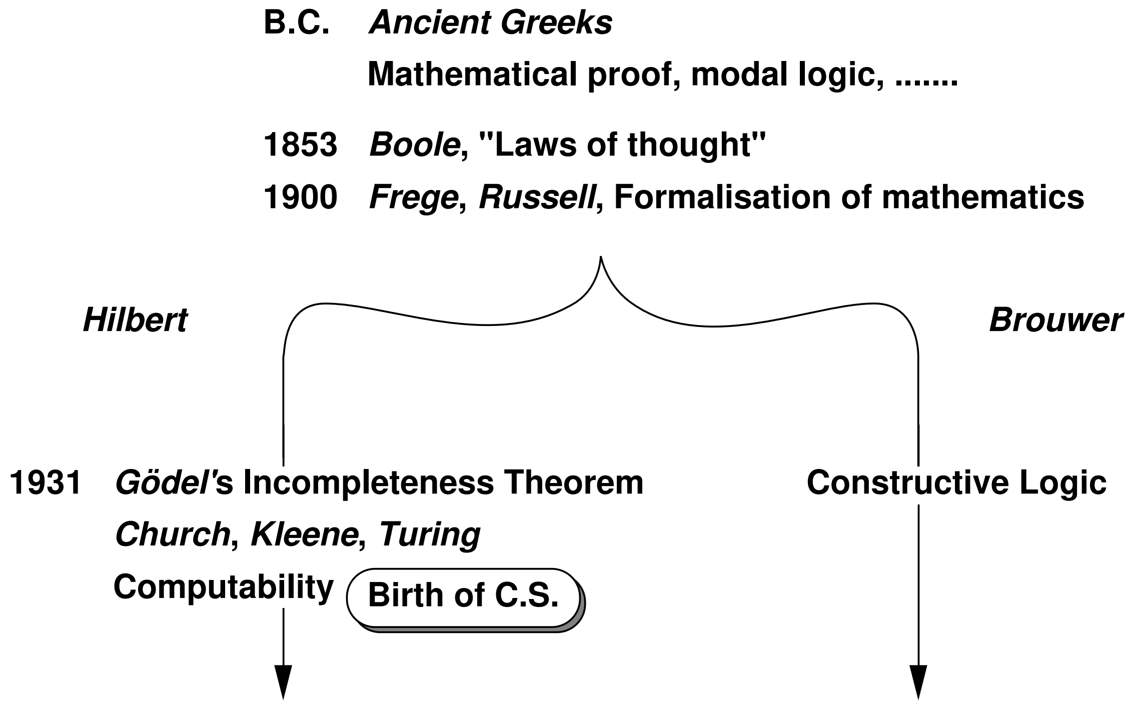

Figure 3: Mathematical logic.

least in principle, to a kind of game played with pen and paper. Hilbert's programme collapsed in 1931 when Gödel published his Incompleteness Theorem. This showed that any formal system expressive enough to handle arithmetic must leave some true arithmetical statements unprovable. In the next few years there was frantic activity to understand the concepts on which Gödel's proof rested. Gödel and others realised that the key lay in the properties of mechanically computable functions, that the Incompleteness Theorem expressed a limitation in what was mechanically computable. In pinning the ideas down, abstract computing machines, like Turing machines, were invented and their properties studied. This arguably marks the birth of Computer Science, more than a decade before the first electronic computers.

Logic has since had a guiding influence on much of Computer Science. It provides proof systems for establishing programs and systems are correct. For example, Modal Logic is important for proving safety properties in distributed systems, showing that necessarily, no matter how the system evolves, a specified bad situation can never arise. Harder to communicate, is the guiding influence Logic has had in the development of Computer Science. Logic, Algorithmics and Semantics have many techniques and methodologies in common. Even Brouwer's Constructive Logic is coming to play an increasing role in Computer Science, essentially because his constructions can be viewed as programs. The influence is far from all one-way. Computers bring Logic to life in theorem provers and logic programming; the numerous, tedious formal manipulations required by proofs within a formal system can be made actual only through the use of machine power.

I hope this has given an impression of the three areas Semantics, Algorithmics and Logic, and why they are every bit as important to computation as the technology on which computers are based. Computers require a new Mathematics. Our intention is that BRICS will play a significant role in its development and propagation. How are we to do this? Through a combination of long-term positions (1–2 years), guests, and short-term emphasis on research themes. Our first theme "Probabilistic proof" (August-September 1994) has been successful in attracting world-class researchers. It is based on the fact that, by abandoning absolute certainty for probabilistic certainty, intractable problems can come to have efficient solutions as programs.

Acknowledgments

Many thanks to Karen K. Møller for typing and Uffe H. Engberg for help with preparation of the figures.

Recent Publications in the BRICS Report Series

RS-94-1 **Glynn Winskel.** ***Semantics, Algorithmics and Logic: Basic Research in Computer Science. BRICS Inaugural Talk*****.** **February 1994, 8 pp.**

RS-94-2 **Alexander E. Andreev.** ***Complexity of Nondeterministic Functions*****.** **February 1994, 47 pp.**

RS-94-3 **Uffe H. Engberg and Glynn Winskel.** ***Linear Logic on Petri Nets*****.** **February 1994, 54 pp. Appear in:** ***Proceedings of REX '93*** **(eds. J. W. de Bakker et al.), LNCS 803, 1994.**