A Mathematical Theory of Computation?

Simone Martini

Dipartimento di Informatica – Scienza e Ingegneria
Alma mater studiorum • Università di Bologna
and
INRIA FoCUS – Sophia / Bologna

Lille, February 1, 2017

Reflect and trace the interaction of mathematical logic
and programming (languages),
identifying some of the driving forces of this process.

Previous episodes: Types
HaPOC 2015, Pisa: from 1955 to 1970 (circa)
Cie 2016, Paris: from 1965 to 1975 (circa)

Why types?

Modern programming languages:

- control flow specification: small fraction
- abstraction mechanisms to model application domains.

- Types are a crucial building block of these abstractions

- And they are a mathematical logic concept, aren't they?

Why types?

Modern programming languages:

- control flow specification: small fraction
- abstraction mechanisms to model application domains.

- Types are a crucial building block of these abstractions
- And they are a mathematical logic concept, aren't they?

We today conflate:

- Types as an implementation (representation) issue
- Types as an abstraction mechanism
- Types as a classification mechanism (from mathematical logic)

The quest for a "Mathematical Theory of Computation"

How does mathematical logic fit into this theory?

And for what purposes?

The quest for a "Mathematical Theory of Computation"

How does mathematical logic fit into this theory?

And for what purposes?

Prehistory

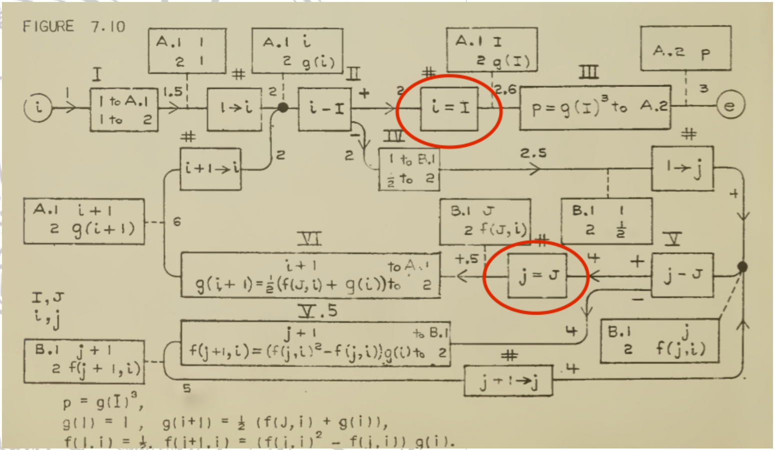
1947

Goldstine and von Neumann

[...] coding [...] has to be viewed as a logical problem and one that represents a new branch of formal logics.

Hermann Goldstine and John von Neumann
Planning and Coding of problems for an Electronic Computing Instrument
Report on the mathematical and logical aspects of an electronic computing instrument,
Part II, Volume 1-3, April 1947. Institute of Advanced Studies.

Boxes in flow diagrams



Goldstine and von Neumann, 2

Boxes in flow diagrams

- operation boxes
- substitution boxes
- assertion boxes

The contents of an assertion box are one or more relations.

An assertion box [. . .] indicates only that certain relations are automatically fulfilled whenever [the control reaches that point]

Free and bound variables, etc.

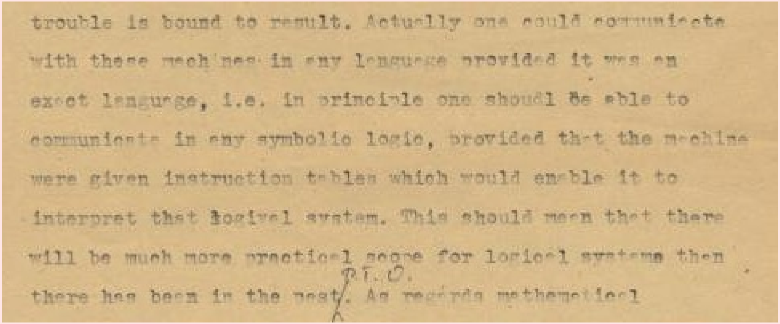
Goldstine and von Neumann

Logic as the discipline to prove assertions

Lecture on Automatic Computing Engine

London Mathematical Soc., 20 Feb 1947. Typewritten notes, in Turing Archive, AMT/C/32

High-level languages


trouble is bound to result. Actually one could communicate
with these machines in any language provided it was an
exact language, i.e. in principle one shoudl be able to
communicate in any symbolic logic, provided that the machine
were given instruction tables which would enable it to
interpret that logical system. This should mean that there
will be much more practical scope for logical systems than
P.T.O.
there has been in the past. As regards mathematical

Turing

Lecture on Automatic Computing Engine

London Mathematical Soc., 20 Feb 1947. Typewritten notes, in Turing Archive, AMT/C/32

High-level languages

In principle one should be able to communicate [with these machines] in any symbolic logic [. . .].

This would mean that there will be much more practical scope for logical systems than there has been in the past.

Turing

Logic as the discipline of formal languages

A bright future, for both

Goldstine and von Neumann:

A logical problem [. . .] that represents a new branch of formal logics.

Turing:

There will be much more practical scope for logical systems.

Turing, again: 1949

The programmer should make assertions about the various states that the machine can reach.

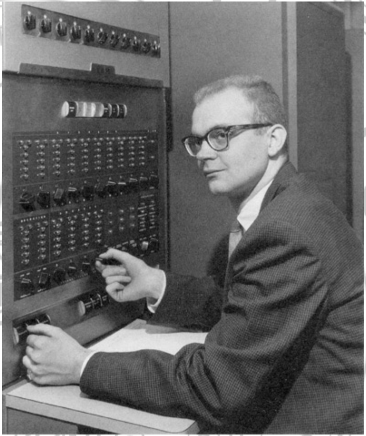
The checker has to verify that [these assertions] *agree with the claims that are made for the routine as a whole.*

Finally the checker has to verify that the process comes to an end.

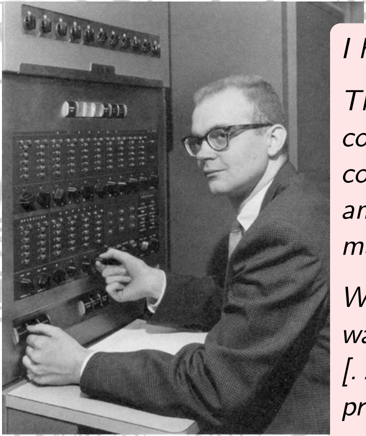
A.M. Turing. Checking a large routine. Paper read on 24 June 1949 at the inaugural conference of the EDSAC computer at the Mathematical Laboratory, Cambridge.
Discussed by Morris and Jones, Annals of the History of Computing, Vol. 6, Apr. 1984.

Programming in the fifties (and later. . .) was a different story. . .

Knuth's recollection, circa 1962



Knuth's recollection, circa 1962



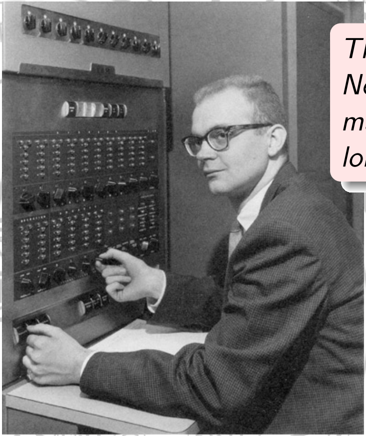
I had never heard of "computer science"

The accepted methodology for program construction was [. . .]: People would write code and make test runs, then find bugs and make patches, then find more bugs and make more patches, and so on

We never realized that there might be a way to construct a rigorous proof of validity [. . .] even though I was doing nothing but proofs when I was in a classroom

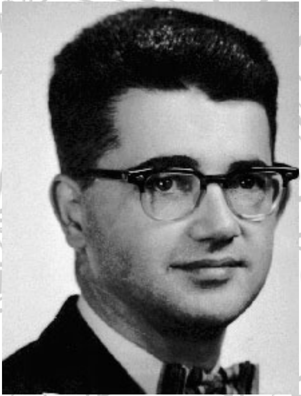
[D.K. Knuth, Robert W. Floyd, in memoriam. ACM SIGACT News 2003]

Knuth's recollection, circa 1962



The early treatises of Goldstine and von Neumann, which provided a glimpse of mathematical program development, had long been forgotten.

A Mathematical Theory of Computation

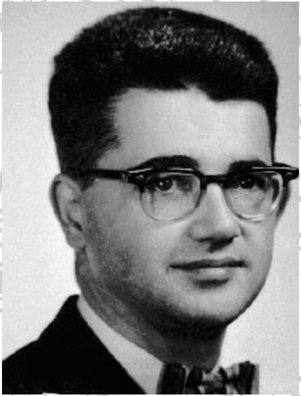


It is reasonable to hope that the relationship between computation and mathematical logic will be as fruitful in the next century as that between analysis and physics in the last.

John McCarthy, MIT 1961; Stanford 1963

From the conclusion of the final version of the paper (1963): A Basis for a Mathematical Theory of Computation. 1961: the Western Joint Computer Conference; 1962: IBM symposium in Blaricum, Netherlands; 1963: in *Computer Programming and Formal Systems*, North Holland.

A Mathematical Theory of Computation

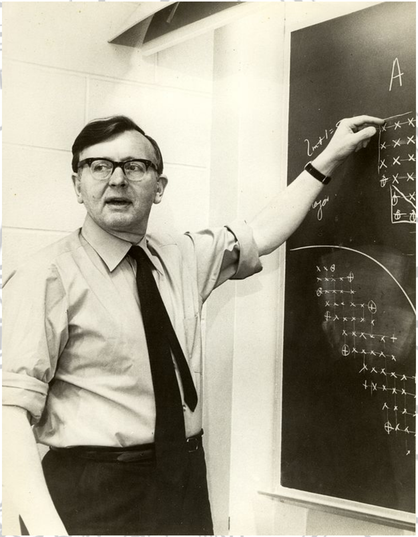


It is reasonable to hope that the relationship between computation and mathematical logic will be as fruitful in the next century as that between analysis and physics in the last.

John McCarthy, MIT 1961; Stanford 1963

From the conclusion of the final version of the paper (1963): A Basis for a Mathematical Theory of Computation. 1961: the Western Joint Computer Conference; 1962: IBM symposium in Blaricum, Netherlands; 1963: in *Computer Programming and Formal Systems*, North Holland.

Which matematics for computing?



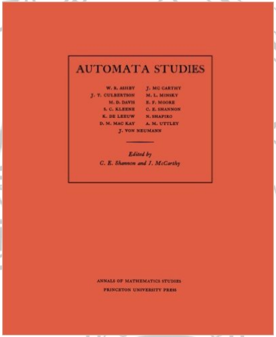
Numerical analysis

Roundoff errors in matrix computation: $Ax = b$

- Turing
- Goldstine & von Neumann: solve $A'Ax = A'b$, for A' transpose of A

Jim Wilkinson (Turing Aw. 1970): backward error analysis

Which matematics for computing?



Automata theory

McCulloch and Pitts (1943)

Kleene ("regular events"), Nerode, Myhill, Shepherdson

Automata Studies, Shannon and McCarthy (eds) [Davis, Kleene, Minsky, Moore, etc.] Princeton Univ Press, 1956

Rabin and Scott. Finite Automata and their decision problems. IBM J. 1959

A basis for a Mathematical Theory of Computation

Expected practical Results:

1. To develop a universal programming language

   "Universal" = machine independent and general

2. To define a theory of the equivalence of computation processes

   Define equivalence-preserving transformations: optimization, compilation, etc.

A basis for a Mathematical Theory of Computation

Expected practical Results:

3. To represent algorithms by symbolic expressions in such a way that significant changes in the behavior represented by the algorithms are represented by simple changes in the symbolic expressions.

   Learning algorithms, whose modifiable behavior depends on the value of certain registers.

A basis for a Mathematical Theory of Computation

Expected practical Results:

4. To represent computers as well as computations in a formalism that permits a treatment of the relation between a computation and the computer that carries out the computation.

5. To give a quantitative theory of computation. There might be a quantitative measure of the size of a computation analogous to Shannon's measure of information.

We hope that the reader will not be angry about the contrast between the great expectations of a mathematical theory of computation and the meager results presented in this paper.

Contents

- a class of recursively computable functions
- based on arbitrary domains of data and operations on them
- with conditional expressions

- functionals
- a general theory of datatypes
- recursion induction to prove equivalences

Computation and Mathematical Logic, 1-2/4

There is no single relationship between logic and computation which dominates the others.

1. Morphological parallels
   the importance of this relationship has been exaggerated, because as soon as one goes into what the sentences mean the parallelism disappears
2. Equivalent classes of problems
   reduction between problems to show undecidability
   Some of this world is of potential interest for computation even though the generation of new unsolvable classes of problems does not in itself seem to be of great interest for computation.

Computation and Mathematical Logic, 3

There is no single relationship between logic and computation which dominates the others.

3. Proof procedures and proof checking procedures:

   Instead of trying out computer programs on test cases until they are debugged, one should prove that they have the desired properties.

   Work on a mildly more general concept of formal system:

$$\mathit{check}(\mathit{statement}, \mathit{proof})$$

Computation and Mathematical Logic, 3

There is no single relationship between logic and computation which dominates the others.

3. Proof procedures and proof checking procedures:

   It should be remembered that the formal systems so far developed by logicians have heretofore quite properly had as their objective that it should be convenient to prove metatheorems about the systems rather than that it be convenient to prove theorems in the systems.

Computation and Mathematical Logic, 4

There is no single relationship between logic and computation which dominates the others.

4. Use of formal systems by computer programs:
   Mathematical linguists are making a serious mistake in their almost exclusive concentration on the syntax and, even more specially, the grammar of natural languages. It is even more important to develop a mathematical understanding and a formalization of the kinds of information conveyed in natural language.

   The main problem in realizing the *Advice Taker* has been devising suitable formal languages covering the subject matter about which we want the program to think.

No explicit program correctness?

Towards a Mathematical Science of Computation, IFIP 1962

One of the first attempts towards an epistemology of computing

1. What are the entities with which the science of computation deals?
   data, procedures, programs, semantics etc.
2. What kinds of facts about these entities would we like to derive?
3. What are the basic assumptions from which we should start?

No explicit program correctness?

Towards a Mathematical Science of Computation, IFIP 1962

One of the first attempts towards an epistemology of computing

1. What are the entities with which the science of computation deals?
   data, procedures, programs, semantics etc.
2. What kinds of facts about these entities would we like to derive?
3. What are the basic assumptions from which we should start?

For what purpose?

1. To define programming languages
   At present, programming languages are constructed in a very unsystematic way. [...] A better understanding of the structure of computations and of data spaces will make it easier to see what features are really desirable.
2. To eliminate debugging.
   Instead of debugging a program, one should prove that it meets its specifications, and this proof should be checked by a computer program. For this to be possible, formal systems are required in which it is easy to write proofs.

Contents

1. Recursion induction to prove properties of Algol programs
2. Abstract syntax of programming languages
3. Semantics: the meaning of program is defined by its effect on the state vector.

R. Floyd

An adequate basis for formal definitions of the meanings of programs [. . .] in such a way that a rigorous standard is established for proofs about computer programs

Based on ideas of Perlis and Gorn

That semantics of a programming language may be defined independently of all processors [. . .] appear[s] to be new,

although McCarthy has done similar work for programming languages based on evaluation of recursive functions.

Robert W. Floyd. Assigning meaning to programs. Mathematical Aspects of Computer Science, AMS 1967.

R. Floyd

An adequate basis for formal definitions of the meanings of programs [. . .] in such a way that a rigorous standard is established for proofs about computer programs

Based on ideas of Perlis and Gorn

That semantics of a programming language may be defined independently of all processors [. . .] appear[s] to be new,

although McCarthy has done similar work for programming languages based on evaluation of recursive functions.

Robert W. Floyd. Assigning meaning to programs. Mathematical Aspects of Computer Science, AMS 1967.

Mathematical Aspects of CS

C.A.R. Hoare

> *Computer programming is an exact science in that all the properties of a program and all the consequences of executing it in any given environment can, in principle, be found out from the text of the program itself by means of purely deductive reasoning.*
>
> *Deductive reasoning involves the application of valid rules of inference to sets of valid axioms. It is therefore desirable and interesting to elucidate the axioms and rules of inference which underlie our reasoning about computer programs.*
>
> *C. A. R. Hoare. An Axiomatic Basis for Computer Programming. CACM 12(10), 1969.*

Hoare's triples

$\{P\}\ C\ \{Q\}$: partial correctness

$$\frac{}{\{P[E/x]\}\ x := E\ \{P\}}$$

$$\frac{\{P\}\ C_1\ \{Q\} \quad \{Q\}\ C_2\ \{R]\}}{\{P\}\ C_1; C_2\ \{R\}}$$

$$\frac{\{I \land B\}\ C\ \{I\}}{\{I\}\ \text{while}\ B\ \text{do}\ C\ \{I \land \neg B\}}$$

Examples

$$\{x > 0\}\ x := x * 2\ \{x > -2\}$$

```
x:=10;
A:=0;
while x>0 do        {INV ≡ x+A = 10}
   A := A+1;
   x := x-1;
```

Computer programming is an exact science. . .

Resistances

Most scientists thought that using a computer was simply programming — that it didn't involve any deep scientific thought and that anyone could learn to program. So why have a degree? They thought computers were vocational vs. scientific in nature.

[Conte, Computerworld magazines, 1999]

Computer Science Dpts

1962 Purdue University (West Lafayette, IN): first dpt of CS; Samuel D. Conte (Perlis: 1951-1956@computation center)

1965 Stanford University (Palo Alto, CA); George Forsythe (Herriot, McCarthy, Feigenbaum, Wirth, Knuth(later)) Since 1961 it was a "division" of Math Dpt.

1965 Carnegie Mellon University (Pittsburg, PA); Alan J. Perlis (Allen, Simon)

1965 First PhD *given by a CS Dpt*: Richard Wexelblat @ University of Pennsylvania (ENIAC!)

1971 Yale (New Haven, CT); Perlis

Reflections

- A mathematical theory is the entrance ticket to science
- Successes: eg, deterministic parsing: LL, LR etc.
- Numerical analysis, formal languages, complexity theory, algorithms, . . .
- But only mathematical logic seems to be dreamed as the mathematics of computing

The grand view

Structural engineering

- mathematical physics laws
- empirical knowledge

to understand, predict, and calculate the stability, strength and rigidity of structures for buildings.

McCarthy:

the relationship between computation and mathematical logic will be as fruitful as that between analysis and physics.

C.A.R. Hoare

When the correctness of a program, its compiler, and the hardware of the computer have all been established with mathematical certainty, it will be possible to place great reliance on the results of the program, and predict their properties with a confidence limited only by the reliability of the electronics.

C. A. R. Hoare. An Axiomatic Basis for Computer Programming. CACM 12(10), 1969.

Hierarchy of machines

- All levels are of the same (abstract) nature
- All levels could be subject (at least conceptually) to the same analysis.
- A formally proved chain of compilers:
  a proof that a model of the hight level program satisfies a condition,
  transfers to a proof that a model of the low level program satisfies a certain condition (automatically obtained from the other)
- No concrete, iron, workmanship is involved.

E. Dijkstra

> *In the relation between mathematics and computing science, the latter has been for many years at the receiving end, and I have often asked myself if, when, and how computing would ever be able to repay its debt.*
>
> *Edsger. W. Dijkstra. On a cultural gap. Mathematical Intelligencer, 1986.*

The analogy with structural engineering is all that is claimed.

Not more.

[There are] *theoretical limitations of program verification. But they're are just the limitations implicit in any applied mathematics.*

Jon Barwise. Mathematical Proofs of Computer System Correctness, Notices of the AMS; 1989.

Linear Logic, 1987

Theoretical Computer Science 50 (1987) 1–102
North-Holland 1

LINEAR LOGIC*

Jean-Yves GIRARD

Équipe de Logique Mathématique, UA 753 du CNRS, UER de Mathématiques, Université de Paris VII, 75251 Paris, France

Communicated by M. Nivat
Received October 1986

A la mémoire de Jean van Heijenoort

Linear Logic, 1987

One of the main outputs of linear logic seems to be in computer science:

(i) [. . .] LL will help us to improve the efficiency of programs;
(ii) LL is the first attemp to solve the problem of parallelism *at the logical level*
(iii) [. . .] databases; [. . .] automatic reasoning
(iv) [. . .] logic programming

Linear Logic, 1987

For CS, logic is the only way to rationalize bricolage.

In some sense, logic plays the same role as the one played by geometry w.r.t physics: the geometrical frame imposes certain conservation results [...]. The symmetries of logic presumably express deep conservation of information.

Back to bricolage. . . ?

deep learning,
internet of things,
big data,
cyber-physical systems,
big networks,
. . .