Formal Methods: Whence and Whither?

Prof. Jonathan P. Bowen FRSA FBCS

Emeritus Professor of Computing
London South Bank University, UK

Adjunct Professor, Southwest University, Chongqing, China

Chairman, Museophile Limited, Oxford, UK

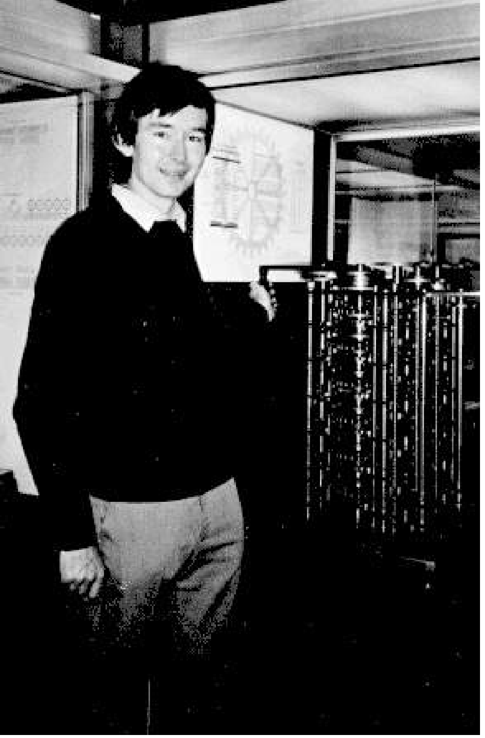
www.jpbowen.com

EST 1892 LSBU

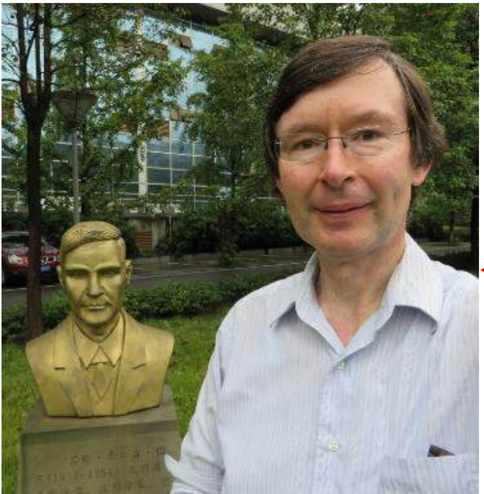
Museophile

SOUTHWEST UNIVERSITY 1906

Introduction



Babbage Difference Engine at the Science Museum, London (c.1980)



- *Subjects:* Mathematics, art, engineering, computer science, **software engineering**, **formal methods**, museum informatics, history of computing, digital culture
- *Academia:* Imperial College London, **Oxford University Computing Laboratory**, University of Reading, **London South Bank University**, Birmingham City University
- *Visitor:* **UNU-IIST (Macau)**, King's College London, Brunel Univ., Westminster (UK), **Waikato Univ. (Hamilton, New Zealand),** Pratt Institute (New York, USA), **East China Normal Univ. (Shanghai, China)**, Institute for Advanced Studies (IIAS, Jerusalem, Israel), **Southwest University (Chongqing, China)**
- *Industry:* Marconi Instrument, Logica, Silicon Graphics (California, USA), **Altran Praxis** (now Capgemini)

Peter Landin (1930–2009)

Program Verification and Semantics: The early work

Teresa Numerico and Jonathan Bowen

Robin Milner, Tony Hoare, Joe Stoy, and Peter Landin

On Tuesday 5 June 2001, a seminar on *Program Verification and Semantics: The early work* was held in the Director's Suite at the Science Museum, London. The seminar was organized with the co-operation of the British Computer Society (BCS) and the Computer Conservation Society (CCS). It was an instructive and enjoyable afternoon for the hundred or so people that attended the meeting.

Participating in the meeting were some of the pioneers and most important scientists in the fields of program verification and semantics and some of the most important historians of computing in Great Britain. It was a unique occasion that allowed the mingling of these two groups of people with an interest in computer science.

The organization of Prof. Jonathan Bowen, Prof. Cliff Jones and George Davis created a very good rapport between the audience and the speakers that presented their experiences in the field of formal methods. Presentations ranged from formal lectures to personal reminiscences. It was a historical event in itself: the special atmosphere allowed the audience to participate with interesting questions and reminiscences of their own.

After an introductory speech by Chris Burton on the aims of the CCS, Jonathan Bowen outlined very briefly the history of formal methods from Aristotle's logic to the use of Tony Hoare's assertions method in present debugging techniques, via Alan Turing's and Christopher Strachey's achievements.

Science Museum, London, 5 June 2001

Co-organized with Cliff Jones

Peter Landin (1930–2009)

Why are things so complicated?

Peter Landin gave the last talk with the provocative title of "*Why are things so complicated?*" It was a very personal recollection of thoughts about the beginnings of his scholarly career, started at the end of the 1950s. He was much influenced by McCarthy and started to study LISP when the most common language was FORTRAN. LISP was very different from the other contemporary languages because it was based on a functional calculus rather than being procedural in nature. He reminded the audience of Marvin Minsky's hostility against λ-calculus and ALGOL, while he was writing some theoretical papers related to them. He remembered how difficult it was to deal with delay lines and drums and gave the flavor of the past times. The audience had the impression that a piece of the computing history was dancing in front of them.

At the end of the meeting, Cliff Jones, who was cited by some of the main speakers as one of the major scientists in the field, drew some conclusions. The ability to prove mathematically that a program correctly implements its specification is increasingly important, even if there is still a lot to do in order to guarantee that security and safety-critical applications perform correctly.

Background

- Academics vs. industrial practitioners
- Formal methods *still* little used in practice (except for safety/security)
- Misconceptions
- Guidance
- Technology transfer issues
- Future – effect of Artificial Intelligence?

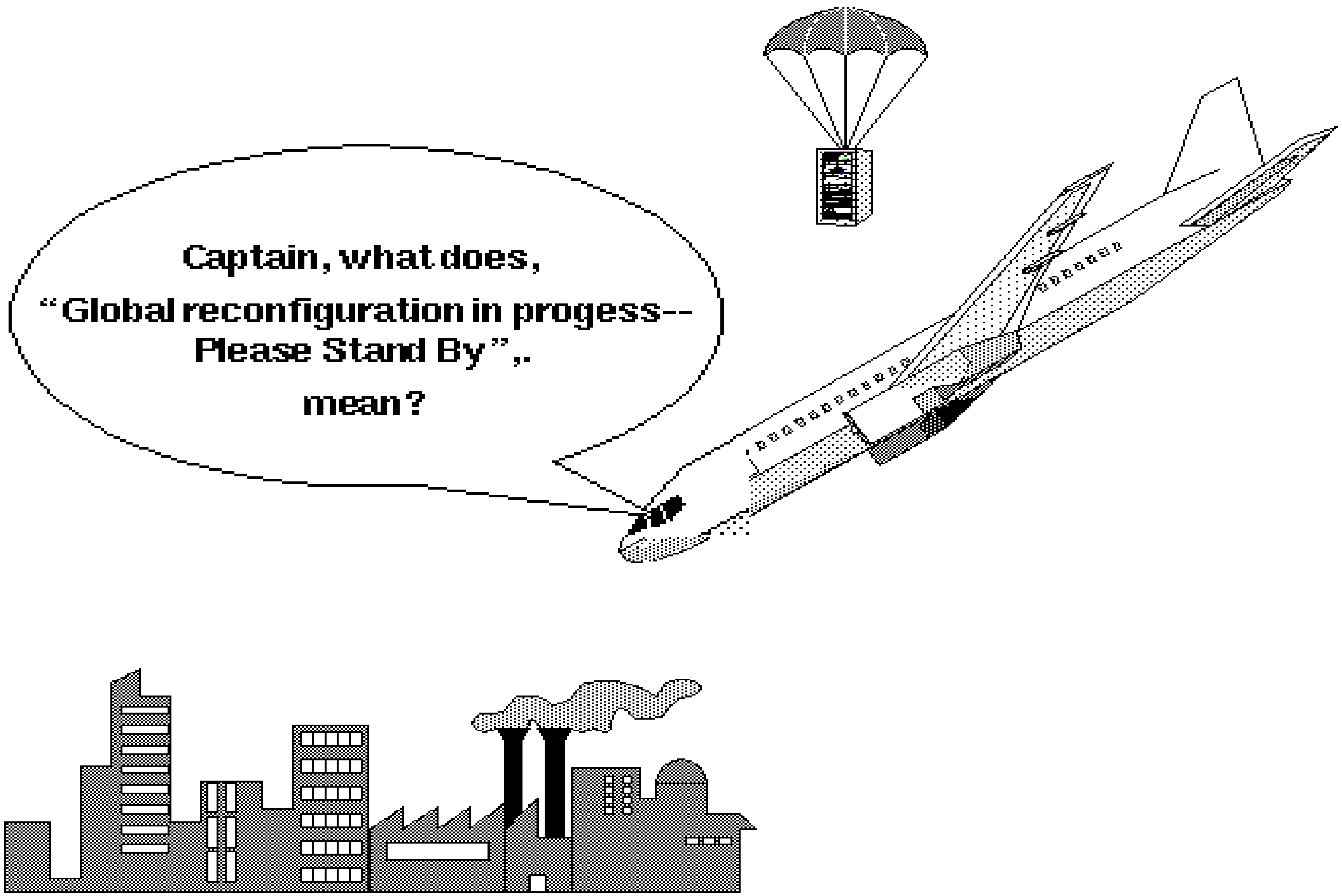
Software…

Failure is not an option…

It comes bundled with the software!

— From a fridge magnet!

Safety and reliability



The Flat Earth Society

The FLAT EARTH SOCIETY

Cf. formal methods community…

— Gerard J. Holzmann

Logic

Aristotle’s logic – highly influential on Western thought.

— Aristotle (384–322 BC)

Aristotle’s Lyceum, rediscovered in Athens (1997)

Proof

- Mathematics – simple theorems, deep proofs
- Cf. software – complicated specifications & programs, shallow proofs

Fermat's Last Theorem (c.1637):

$a^n + b^n \neq c^n$ (integer $n > 2$)

— Pierre de Fermat (1607–1665)

Proved 358 years later by Andrew Wiles, 1994/5.

Not a timescale acceptable for software!

Theory and practice

*"It has long been my personal view that the **separation of practical and theoretical work is artificial** and injurious. Much of the **practical work** done in computing, both in software and in hardware design, is **unsound and clumsy** because the people who do it have not any clear understanding of the fundamental design principles of their work. Most of the abstract mathematical and **theoretical work** is **sterile** because it has no point of contact with real computing."*

— Christopher Strachey (1916–1975)

First formal methods paper?

Arguably the first "formal methods" paper ever:

Checking a Large Routine, Paper for the EDSAC Inaugural Conference, 24 June 1949. In *Report of a Conference on High Speed Automatic Calculating Machines,* pp 67–69.

Reprinted with corrections and annotations in:
An early program proof by Alan Turing, L. Morris and C.B. Jones, IEEE Ann. Hist. Computing 6(2):129–143, 1984.

See also: *Turing and Software Verification,* C.B. Jones. Tech. Report CS-TR-1441, Newcastle University, UK, 2014.

— Alan Turing (1912–1954)

Friday, 24th June.

Checking a large routine. by Dr. A. Turing.

How can one check a routine in the sense of making sure that it is right?

In order that the man who checks may not have too difficult a task the programmer should make a number of definite assertions which can be checked individually, and from which the correctness of the whole programme easily follows.

Consider the analogy of checking an addition. If it is given as:

```
 1374
 5906
 6719
 4337
 7768
 ----
26104
```

one must check the whole at one sitting, because of the carries.

But if the totals for the various columns are given, as below:

```
 1374
 5906
 6719
 4337
 7768
 ----
 3974
2213
 ----
26104
```

the checker's work is much easier being split up into the checking of the various assertions 3 + 9 + 7 + 3 + 7 = 29 etc. and the small addition

```
 3794
2213
 ----
26104
```

This principle can be applied to the process of checking a large routine but we will illustrate the method by means of a small routine viz. one to obtain n without the use of a multiplier, multiplication being carried out by repeated addition.

At a typical moment of the process we have recorded r and s r for some r, s. We can change s r to (s+1) r by addition of r. When s = r+1 we can change r to r+1 by a transfer. Unfortunately there is no coding system sufficiently generally known to justify giving the routine for this process in full, but the flow diagram given in Fig.1 will be sufficient for illustration.

Each 'box of the flow diagram represents a straight sequence of instructions without changes of control. The following convention is used:

(i) a dashed letter indicates the value at the end of the process represented by the box:

(ii) an undashed letter represents the initial value of a quantity.

One cannot equate similar letters appearing in different boxes, but it is intended that the following identifications be valid throughout

s	content of line 27 of store
r	" " " 28 " "
n	" " " 29 " "
u	" " " 30 " "
v	" " " 31 " "

It is also intended that u be s r or something of the sort e.g. it might be (s+1) r or s r-1 but not e.g. $s^2 + r^2$.

In order to assist the checker, the programmer should make assertions about the various states that the machine can reach. These assertions may be tabulated as in fig.2. Assertions are only made for the states when certain particular quantities are in control, corresponding to the ringed letters in the flow diagram. One column of the table is used for each such situation of the control. Other quantities are also needed to specify the condition of the machine completely: in our case it is sufficient to give r and s. The upper part of the table gives the various contents of the store lines in the various conditions of the machine, and restrictions on the quantities s, r (which we may call inductive variables). The lower part tells us which of the conditions will be the next to occur.

The checker has to verify that the columns corresponding to the initial condition and the stopped condition agree with the claims that are made for the routine as a whole. In this case the claim is that if we start with control in condition D and with n in line 29 we shall find a quantity in line 31 when the machine stops which is r (provided this is less than 2^{40}, but this condition has been ignored).

He has also to verify that each of the assertions in the lower half of the table is correct. In doing this the columns may be taken in any order and quite independently. Thus for column B the checker would argue. "From the flow diagram we see that after B the box v' = u applies. From the upper part of the column for B we have u = r. Hence v' = r i.e. the entry for v i.e. for line 31 in C should be r. The other entries are the same as in B".

Finally the checker has to verify that the process comes to an end. Here again he should be assisted by the programmer giving a further definite assertion to be verified. This may take the form of a quantity which is asserted to decrease continually and vanish when the machine stops. To the pure mathematician it is natural to give an ordinal number. In this problem the ordinal might be $(n - r) \omega^2 + (r - s) \omega + k$. A less highbrow form of the same thing would be to give the integer $2^{80} (n - r) + 2^{40} (r - s) + k$. Taking the latter case and the step from B to C there would be a decrease from $2^{80} (n - r) + 2^{40} (r - s) + 5$ to $2^{80} (n - r) + 2^{40} (r - s) + 4$. In the step from F to B there is a decrease from $2^{80} (n - r) + 2^{40} (r - s) + 1$ to $2^{80} (n - r - 1) + 2^{40} (r + 1 - s) + 5$.

In the course of checking that the process comes to an end the time involved may also be estimated by arranging that the decreasing quantity represents an upper bound to the time till the machine stops.

“verification”

“assertions”

“dashed” after states

Checking a large routine (1949)

- "In order to assist the checker, the programmer should make assertions about the various states that the machine can reach."
- "The checker has to verify that the … initial condition and the stopped condition agree with the claims that are made for the routine as a whole."
- "He has also to verify that each of the assertions … is correct."
- "Finally the checker has to verify that the process comes to an end."

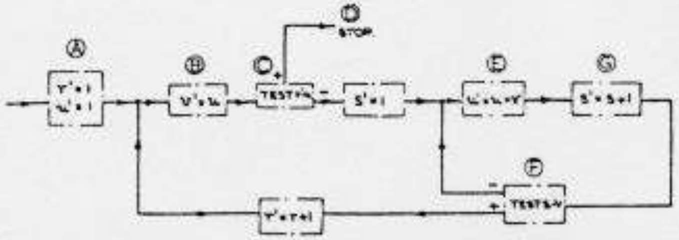


FIG. 1

STORAGE LOCATION	(INITIAL) Ⓐ k=6	Ⓑ k=5	Ⓒ k=4	(STOP) Ⓓ k=0	Ⓔ k=3	Ⓕ k=1	Ⓖ k=2
27					s	s+1	s
28		r	r		r	r	r
29	n	n	n	n	r	n	n
30		⌊r	⌊r		s⌊r	(s+1)⌊r	(s+1)⌊r
31			⌊r	⌊n	⌊r	⌊r	⌊r
	TO Ⓑ WITH r'=1 s'=1	TO Ⓒ	TO Ⓓ IF r=n TO Ⓔ IF r<n		TO Ⓖ	TO Ⓑ WITH r'=r+1 IF s≥r TO Ⓔ WITH r'=s+1 IF s<r	TO Ⓕ

FIG. 2

ISSUE 1 4.6.49 | T.R.E. M.O.S. DIAG. No RTR 11/5790.

An Early Program Proof by Alan Turing

F. L. MORRIS AND C. B. JONES

The paper reproduces, with typographical corrections and comments, a 1949 paper by Alan Turing that foreshadows much subsequent work in program proving.

*Categories and Subject Descriptors: D.2.4 [**Software Engineering**]—correctness proofs; F.3.1 [**Logics and Meanings of Programs**]—assertions; K.2 [**History of Computing**]—software*
General Terms: Verification
Additional Key Words and Phrases: A. M. Turing

Introduction

The standard references for work on program proofs attribute the early statement of direction to John McCarthy (e.g., McCarthy 1963); the first workable methods to Peter Naur (1966) and Robert Floyd (1967); and the provision of more formal systems to C. A. R. Hoare (1969) and Edsger Dijkstra (1976). The early papers of some of the computing pioneers, however, show an awareness of the need for proofs of program correctness and even present workable methods (e.g., Goldstine and von Neumann 1947; Turing 1949).

The 1949 paper by Alan M. Turing is remarkable in many respects. The three (foolscap) pages of text contain an excellent motivation by analogy, a proof of a program with two nested loops, and an indication of a general proof method very like that of Floyd. Unfortunately, the paper is made extremely difficult to read by the large number of transcription errors. For example, all instances of the factorial sign (Turing used ⌊n) have been omitted in the commentary, and ten other identifiers are written incorrectly. It would appear to be worth correcting these errors and commenting on the proof from the viewpoint of subsequent work on program proofs.

Turing delivered this paper in June 1949, at the inaugural conference of the EDSAC, the computer at Cambridge University built under the direction of Maurice V. Wilkes. Turing had been writing programs for an electronic computer since the end of 1945—at first for the proposed ACE, the computer project at the National Physical Laboratory, but since October 1948 for the Manchester prototype computer, of which he was deputy director. The references in his paper to 2^{40} are reflections of the 40-bit "lines" of the Manchester machine storage system.

The following is the text of Turing's 1949 paper, corrected to the best of our ability to what we believe Turing intended. We have made no changes in spelling, punctuation, or grammar.

Authors' Addresses: F. L. Morris, School of Computer and Information Science, 313 Link Hall, Syracuse University, Syracuse, NY 13210. C. B. Jones, Department of Computer Science, The University, Manchester M13 9PL, England.

Turing Text

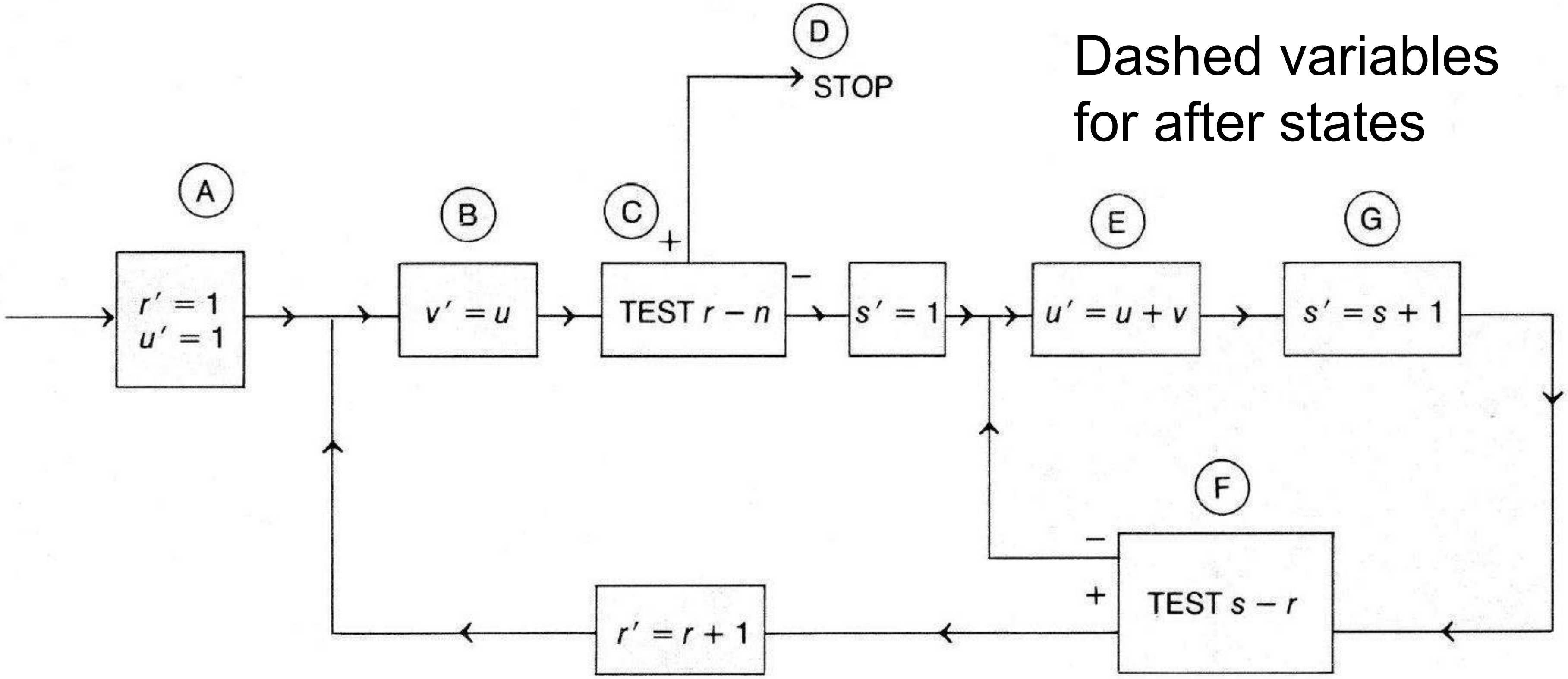
Friday, 24th June [1949]

Checking a large routine by Dr A. Turing.

How can one check a routine in the sense of making sure that it is right?

In order that the man who checks may not have too difficult a task the programmer should make a number of definite assertions which can be checked individually, and from which the correctness of the whole programme easily follows.

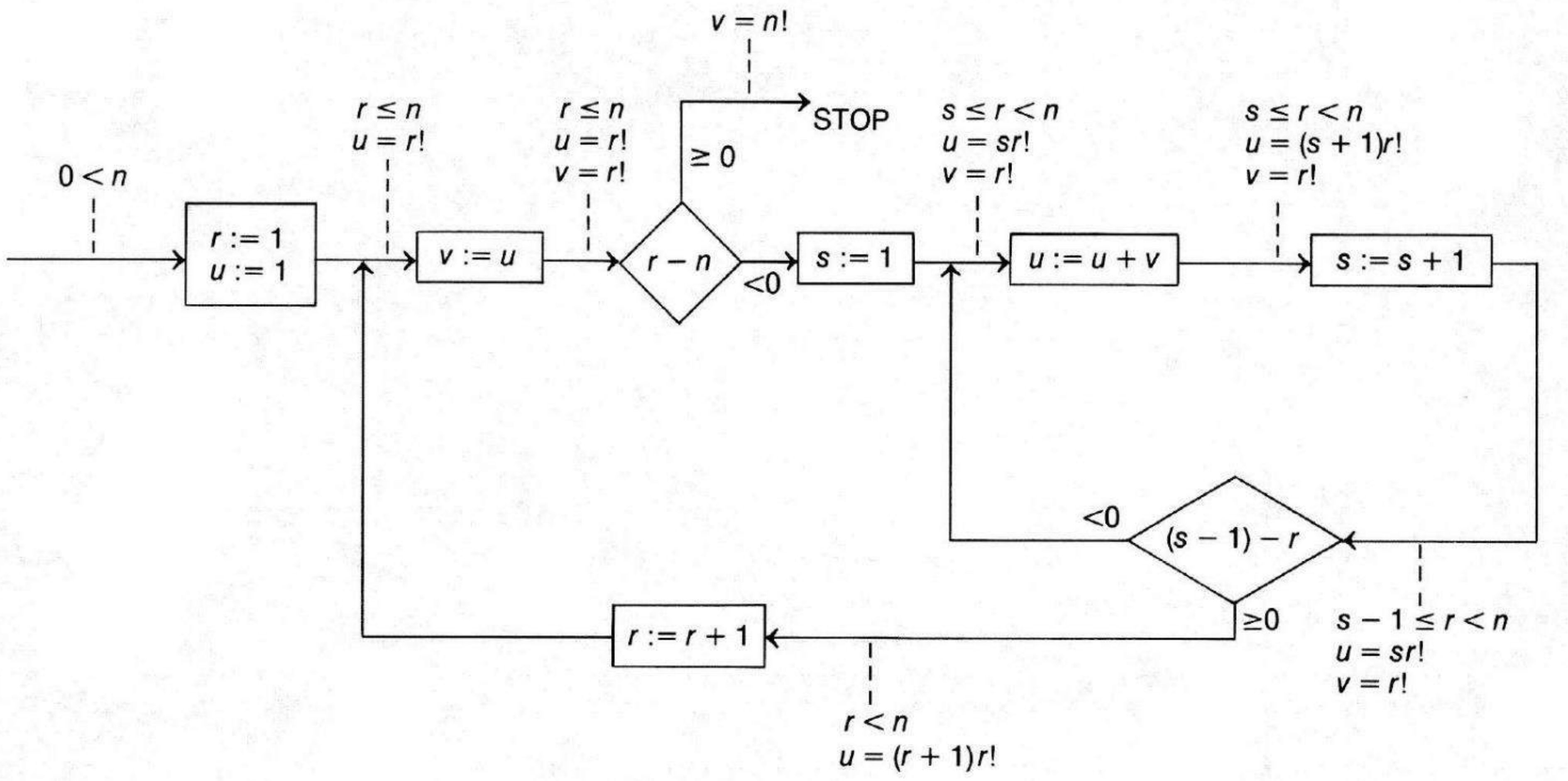
Dashed variables for after states



STORAGE LOCATION	(INITIAL) Ⓐ $k = 6$	Ⓑ $k = 5$	Ⓒ $k = 4$	(STOP) Ⓓ $k = 0$	Ⓔ $k = 3$	Ⓕ $k = 1$	Ⓖ $k = 2$
27					s	$s + 1$	s
28		r	r		r	r	r
29	n	n	n	n	n	n	n
30		$\lfloor r$	$\lfloor r$		$s\lfloor r$	$(s + 1)\lfloor r$	$(s + 1)\lfloor r$
31			$\lfloor r$	$\lfloor n$	$\lfloor r$	$\lfloor r$	$\lfloor r$
	TO Ⓑ WITH $r' = 1$ $u' = 1$	TO Ⓒ	TO Ⓓ IF $r = n$ TO Ⓔ IF $r < n$		TO Ⓖ	TO Ⓑ WITH $r' = r + 1$ IF $s \geq r$ TO Ⓔ WITH $s' = s + 1$ IF $s < r$	TO Ⓕ

Turing and program proving

Modernized flow diagram, with assertions



Mathematics and programming

In 1951, Christopher Strachey wrote a letter to Alan Turing on his programming plans:

"... once the suitable notation is decided, all that would be necessary would be to type ***more or less ordinary mathematics*** *and a special routine called, say, 'Programme' would convert this into the necessary instructions to make the machine carry out the operations indicated. This may sound rather Utopian, but I think it, or something like it, should be possible …"*

Turing’s influence on program proving

- Aad van Wijngaarden was at the Cambridge meeting – but no known influence (1949…)
- Robert Floyd rediscovered ideas similar to those of Turing (published 1967)
- Tony Hoare developed these further (published 1969)
- Had Turing lived longer, perhaps formal methods (in particular program proving) would have developed more rapidly, rather than being rediscovered

REFERENCES

1976 Dijkstra, E. W. 1976. *A Discipline of Programming.* Englewood Cliffs, N.J., Prentice-Hall.

1967 Floyd, R. W. 1967. "Assigning Meaning to Programs." *Proc. of Symposia in Appl. Math. 19.* (Also in S. T. Schwartz (ed.), *Mathematical Aspects of Computer Science*, Providence, American Mathematical Society, 1967.)

1947 Goldstine, H. H., and J. von Neumann. 1947. "Planning and Coding of Problems for an Electronic Computing Instrument." Report of U.S. Ord. Dept. In A. Taub (ed.), *Collected Works* of J. von Neumann, New York, Pergamon, Vol. 5, 1965, pp. 80–151.

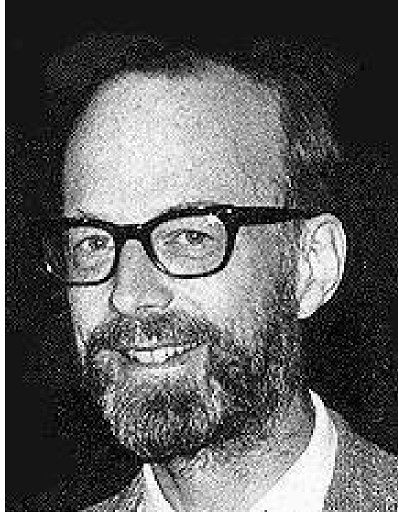
1969 Hoare, C. A. R. October 1969. An axiomatic basis for computer programming. *Comm. ACM 12*, 10, 576–580.

1963 McCarthy, J. 1963. "A Basis for a Mathematical Theory of Computation." In P. Braffort and D. Hirschberg (eds.), *Computer Programming and Formal Systems*, Amsterdam, North-Holland, 1967, pp. 33–70.

1966 Naur, P. 1966. Proof of algorithms by general snapshots. *BIT* 6, 4, 310–316.

1949 Turing, A. M. 1949. "Checking a Large Routine." In *Report of a Conference on High Speed Automatic Calculating Machines*, Univ. Math. Lab., Cambridge, pp. 67–69.

Assertions



An Axiomatic Basic for Computer Programming.
Communications of the ACM, October 1969

— Sir Tony Hoare (b.1934)

Hoare logic: {*pre*} *prog* {*post*}

Program *proving* with pre- and post-conditions as "*assertions*" (logical statements about the program)

30 years later … assertions widely used by programmers for testing and debugging rather than proof

Formal …

formal /fm()l/ a. LME. [L formalis, f. forma: see FORM n., -AL.] **1** **a** Philos. Of or pertaining to the form or constitutive essence of a thing; essential. LME. **b** Pertaining to the specific form of an animal or plant. LME-L17. **c** Of or pertaining to the outward form, shape, appearance, arrangement, or external qualities of a thing. Formerly also (of knowledge), theoretical. M17. **d** Logic. Concerned with the form, not the matter, of reasoning. M19.

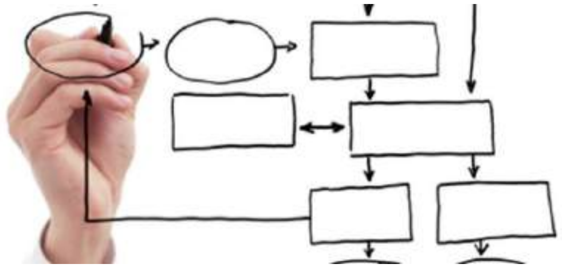
```
proof
  let " ?S = {x.x ∉ f x} "
  show "?S ∉  range f"
  proof
    assume "?S ∈  range f"
    then obtain y where fy: "?S = fy" ..
    show False
    proof cases
      assume " y ∈?S "
      hence " y ∉ f y "     by simp
```

“After great pain, a formal feeling comes—”

— Emily Dickinson (1862)

… Methods

method /methd/ *n.* LME. [L *methodus* f. Gk *methodos* pursuit of knowledge, mode of investigation, f. *meta* (see META-) + *hodos* way.] **I** Procedure for attaining an object. **1** ... **2** A mode of procedure; a (defined or systematic) way of doing a thing, *esp.* (w. specifying wd or wds) in accordance with a particular theory or as associated with a particular person. L16. ... **II** Systematic arrangement, order. <u>**3** The branch of logic that deals with the description and arrangement of arguments or propositions for the investigation or exposition of a truth. M16.</u> **4** Order in thinking or expressing thoughts; the orderly arrangement of ideas; *gen.* orderliness, regularity, or planning in doing anything. M16.



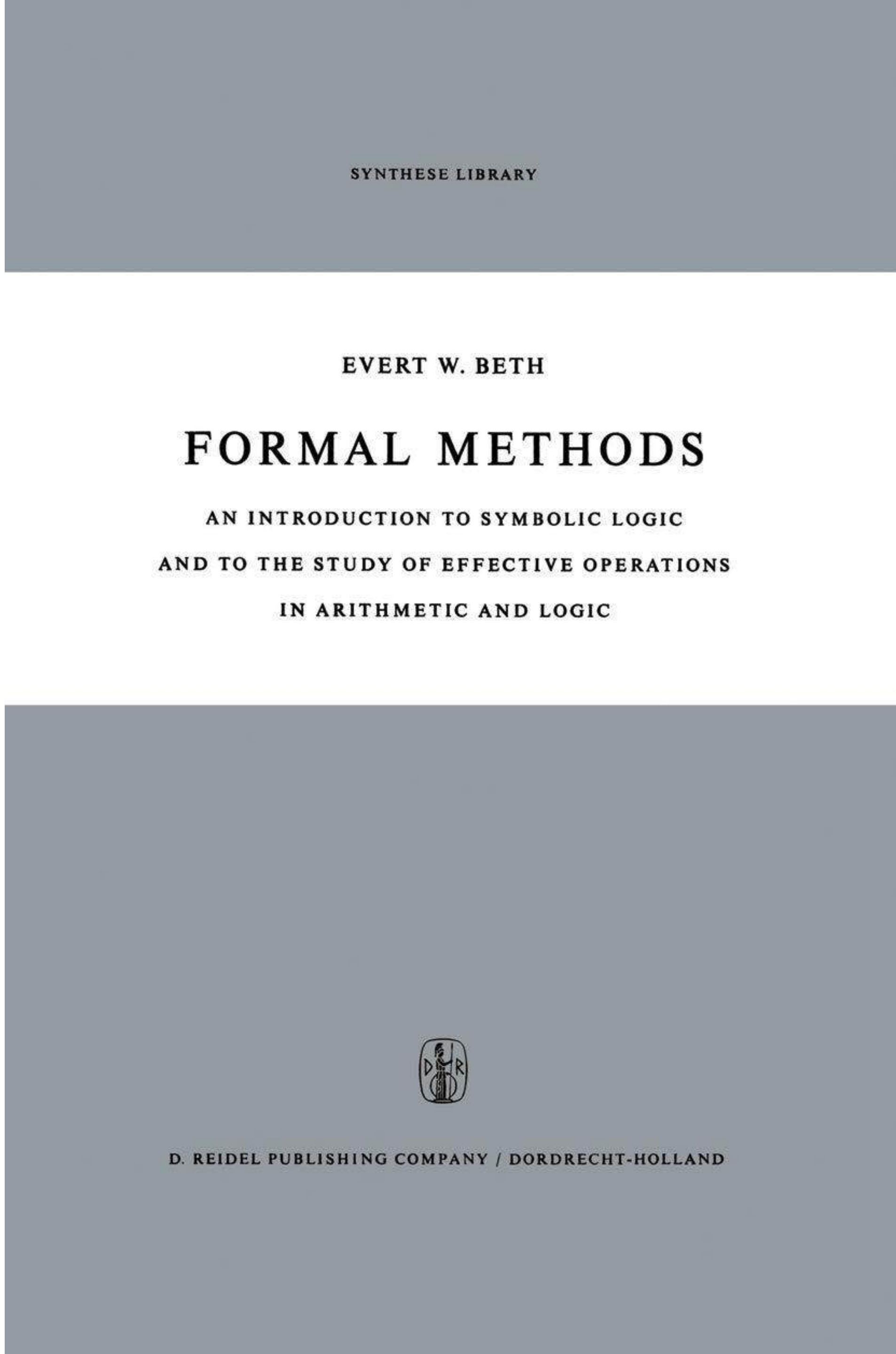
"By different methods different men excel;
But where is he who can do all things well?"

— Charles Churchill (1731–1764)

Formal Methods: An Introduction to Symbolic Logic and to the Study of Effective Operations in Arithmetic and Logic (1962)

Evert Willem Beth (1908–1964),
Dutch philosopher and logician

Earliest book with “formal methods” in the title?



SYNTHESE LIBRARY

EVERT W. BETH

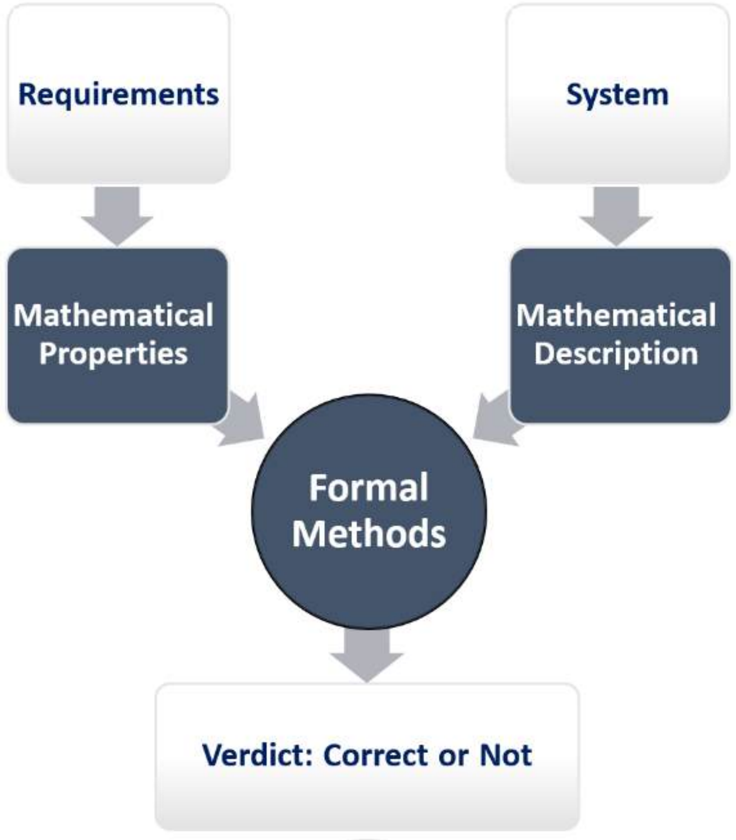
FORMAL METHODS

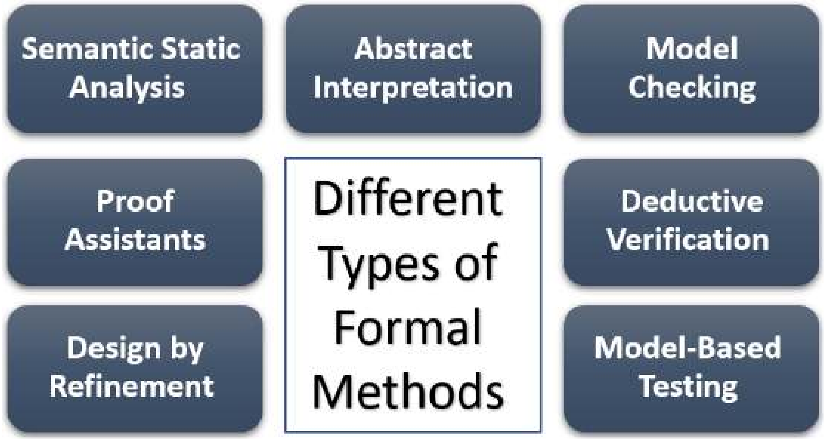
AN INTRODUCTION TO SYMBOLIC LOGIC
AND TO THE STUDY OF EFFECTIVE OPERATIONS
IN ARITHMETIC AND LOGIC

D. REIDEL PUBLISHING COMPANY / DORDRECHT-HOLLAND

Formal methods

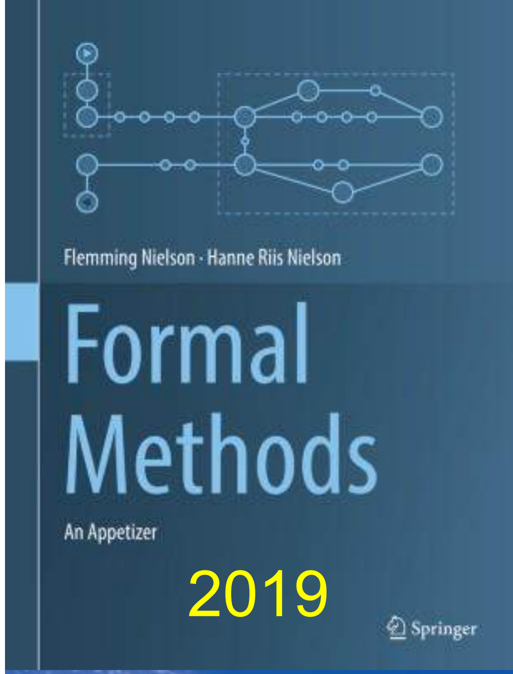
- Term established by late 1970s
  - Next stage from structured design
  - Mathematical basis
- Formal specification and (optionally) proof:
  - Validation (correct specification)
  - Verification (correct implementation wrt spec.)
- But engineers *calculate* rather than prove

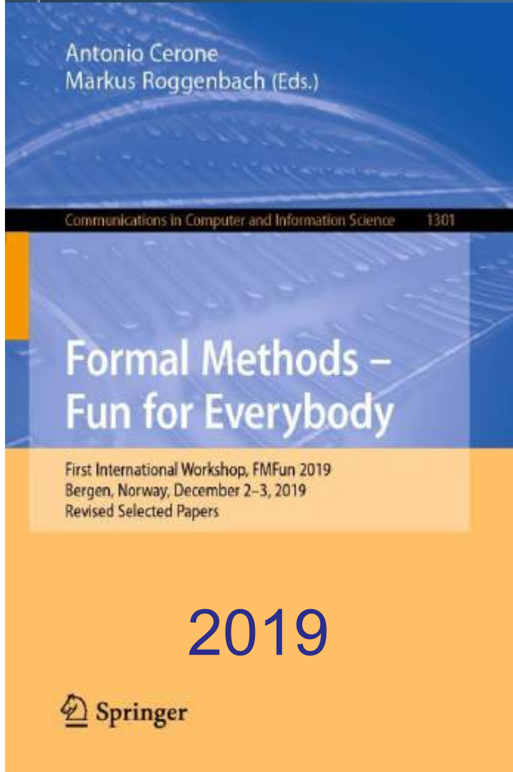




Some formal methods approaches

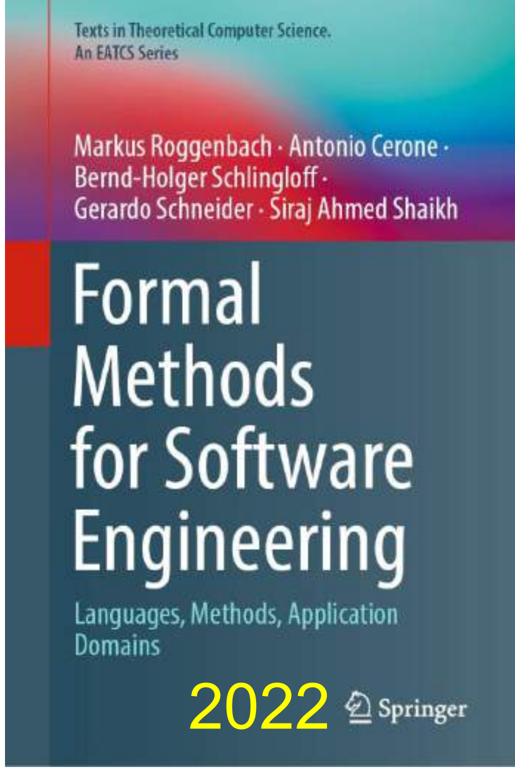
- **Abstract Interpretation:** approximating program behaviour to prove correctness or detect errors.
- **Model-Based Testing:** generating test cases from a formal model.
- **Model Checking:** exhaustively verifying system behaviour against a formal specification.
- **Proof Assistants:** tools for interactively constructing and verifying mathematical proofs.
- **Refinement:** systematically refining a high-level specification into a correct implementation.
- **Static Analysis:** analyzing program code meaning to detect errors or enforce constraints.
- **Verification:** proving the correctness of a program using logical inference rules.





Formal methods levels

0. **Formal Specification:**
   - Requirements only
   - No analysis or proof
   - Can be used to aid testing
   - Cost-effective
1. **Formal Verification:**
   - Program produced in a more formal way
   - Use of proof or refinement based on a formal specification
   - More costly
2. **Theorem Proving:**
   - Use of a theorem prover tool
   - Formal machine-checked proofs
   - Proof of entire system possible but scaling difficult
   - Expensive and hard



Formal specification

1. *A specification written and approved in accordance with established standards*
2. *A specification written in a formal notation, often for use in proof of correctness.*

— IEEE glossary

BirthdayBook

$known : \mathbb{P}\, NAME$
$birthday : NAME \rightarrowtail DATE$

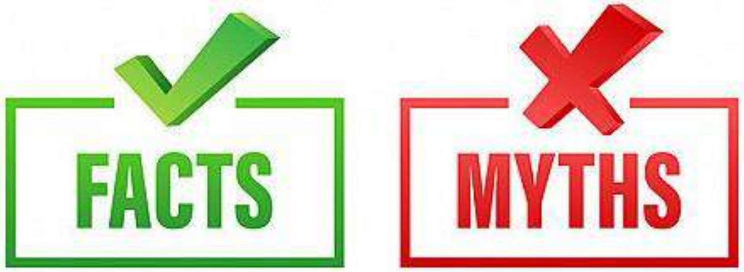
$known = \mathrm{dom}\ birthday$

AddBirthday

$\Delta BirthdayBook$
$name? : NAME$
$date? : DATE$

$name? \notin known$
$birthday' = birthday \cup \{name? \mapsto date?\}$

Seven Myths of Formal Methods



1. Formal Methods can guarantee that software is perfect.
2. Formal Methods are all about program proving.
3. Formal Methods are only useful for safety-critical systems.
4. Formal Methods require highly trained mathematicians.
5. Formal Methods increase the cost of development.
6. Formal Methods are unacceptable to users.
7. Formal Methods are not used on real, large-scale software.

– J.A. Hall, *IEEE Software*, September 1990

ProCoS: Provably Correct Systems

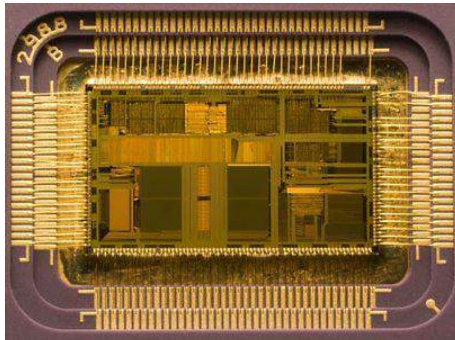
European projects and Working Group (early 1990s)

- Requirements
- Specification
- Design
- Programming
- Compilation
- Hardware

Stop press:
Retrospective multi-author paper accepted for the *Formal Aspects of Computing* journal

Levels of abstraction/complexity

- 15k lines of (informal) requirements
- 150k lines of (formal?) specification
- 1.5 million lines of design description
- 15 million lines of (formal!) high-level program code
- 150 million machine instructions of object code
- 1.5 billion transistors in hardware!



The later a mistake is detected, the more costly it is!

Ten Commandments of Formal Methods

"Can't I just read your URL?"

I. Thou shalt choose an appropriate notation

II. Thou shalt formalize but not over-formalize

III. Thou shalt guestimate costs

IV. Thou shalt have a formal methods guru on call

V. Thou shalt not abandon thy traditional development methods

VI. Thou shalt document sufficiently

VII. Thou shalt not compromise thy quality standards

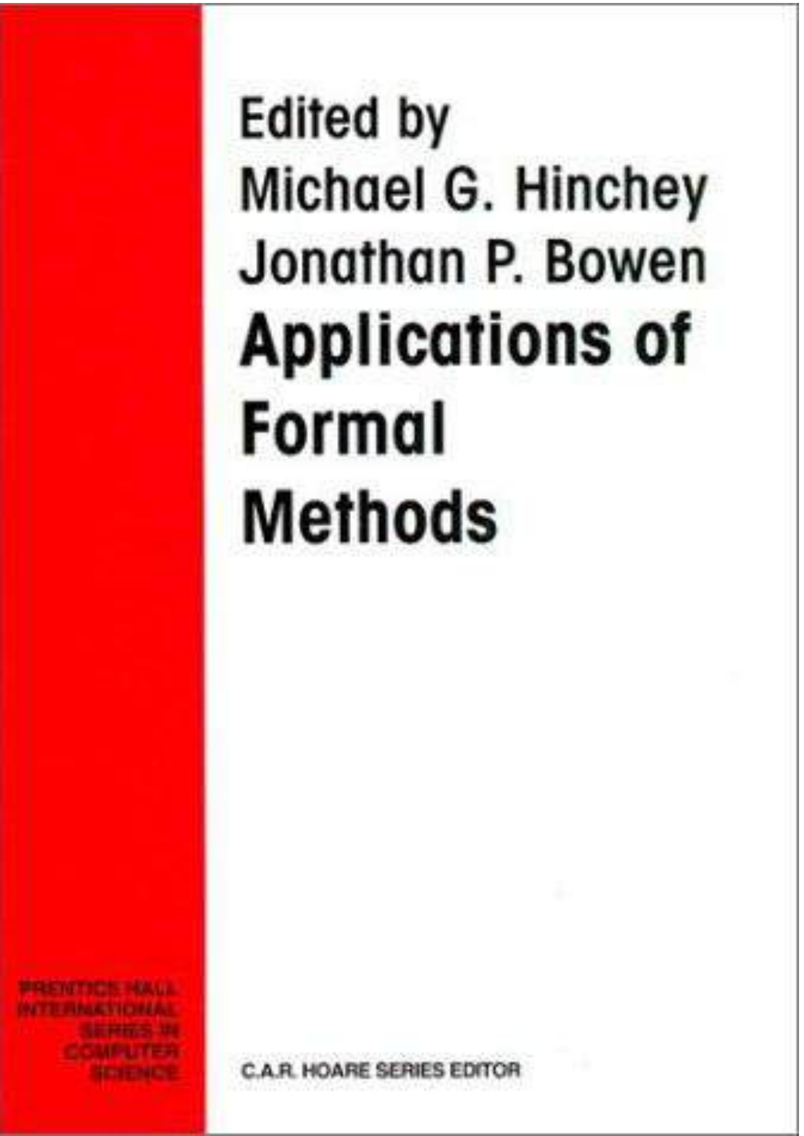
VIII. Thou shalt not be dogmatic

IX. Thou shalt test, test, and test again

X. Thou shalt reuse

– J.P. Bowen & M.G. Hinchey
IEEE Computer, April 1995

Applications of Formal Methods



Examples:

- Tektronix (Z) – oscilloscopes
- STV algorithm (VDM) – voting
- IBM CICS (B) – transaction processing
- AAMP5 microprocessor (PVS) – hardware
- GEC Alsthom (B) – railway software
- A300/340 (Z) – airplane software

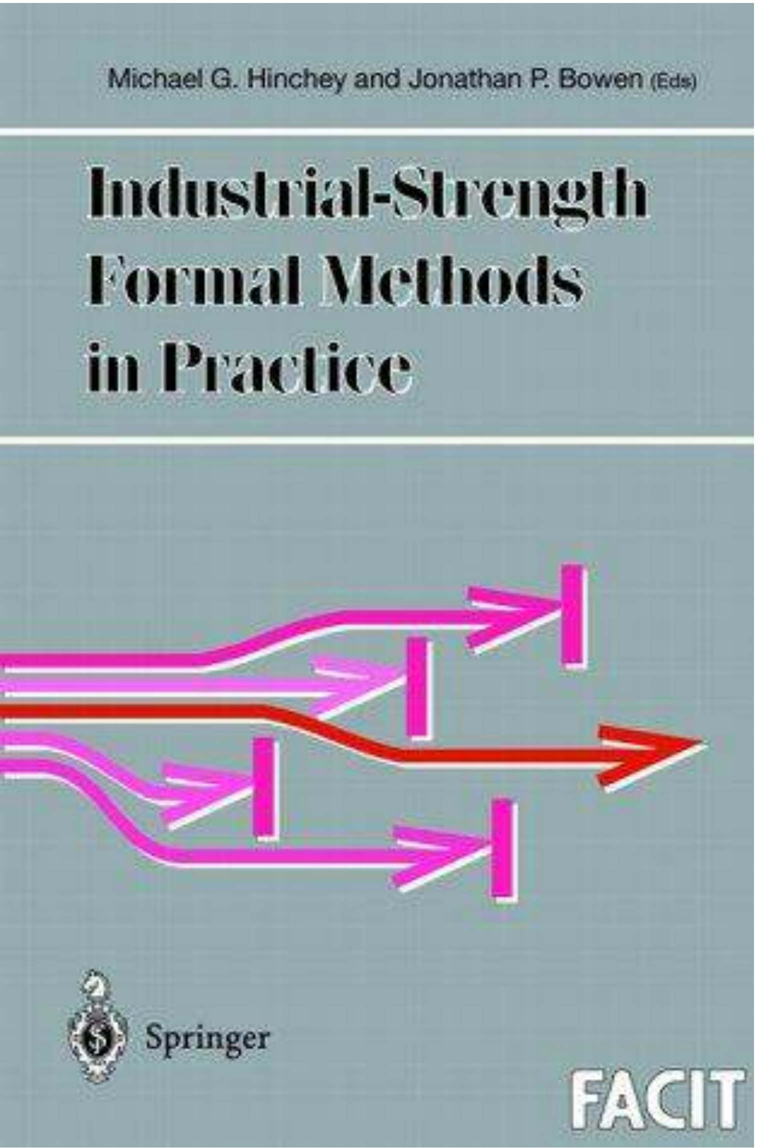
Prentice Hall, International Series in Computer Science, 1995

Seven More Myths of Formal Methods

8. Formal Methods delay the development process.
9. Formal Methods do not have tools.
10. Formal Methods mean forsaking traditional engineering design methods.
11. Formal Methods only apply to software.
12. Formal Methods are not required.
13. Formal Methods are not supported.
14. Formal Methods people always use Formal Methods.

– J.P. Bowen & M.G. Hinchey
IEEE Software, July 1995

Industrial-Strength Formal Methods in Practice



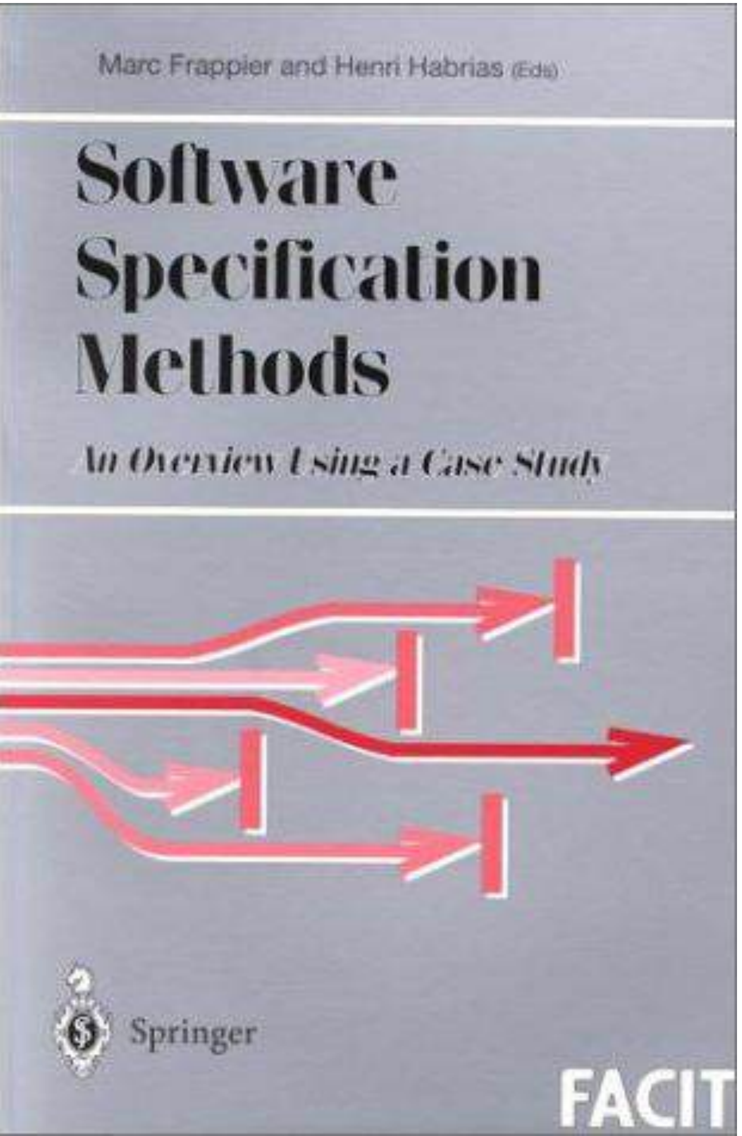
Examples:

- Motorola CAP DSP (ACL2)
- Radiation Therapy Machine (Z)
- ATC system (VDM)
- Railways (Prover Technology)

And later: Microsoft

Springer, FACIT series, 1999

Software Specification Methods



The *process* of producing a formal specification…

Using a selection of formal methods:

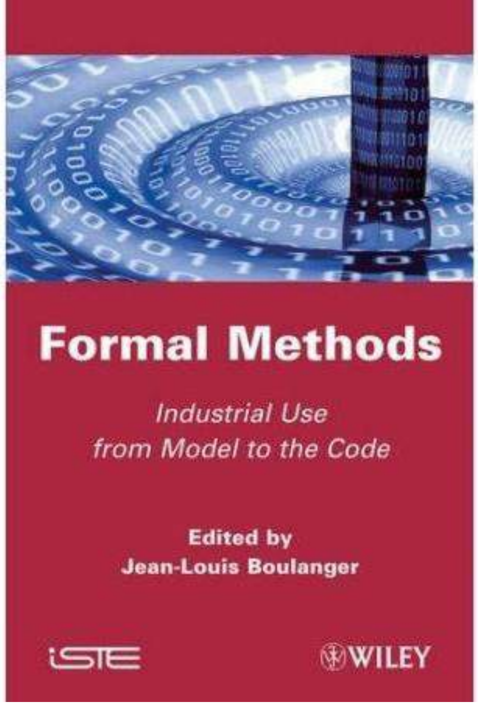
Z, SAZ, B, OMT, Action Systems, UML, VHDL, Estelle, SDL, E-LOTOS, JSD, CASL, Coq, Petri Nets.

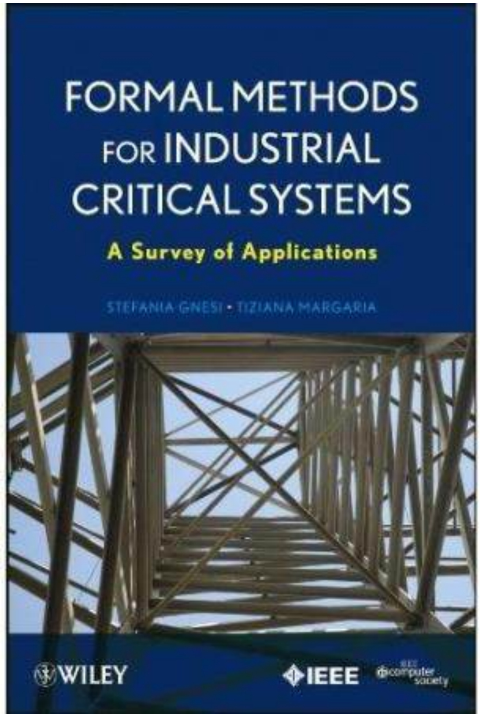
Marc Frappier & Henri Habrias (eds.)
Springer-Verlag, FACIT series, 2001

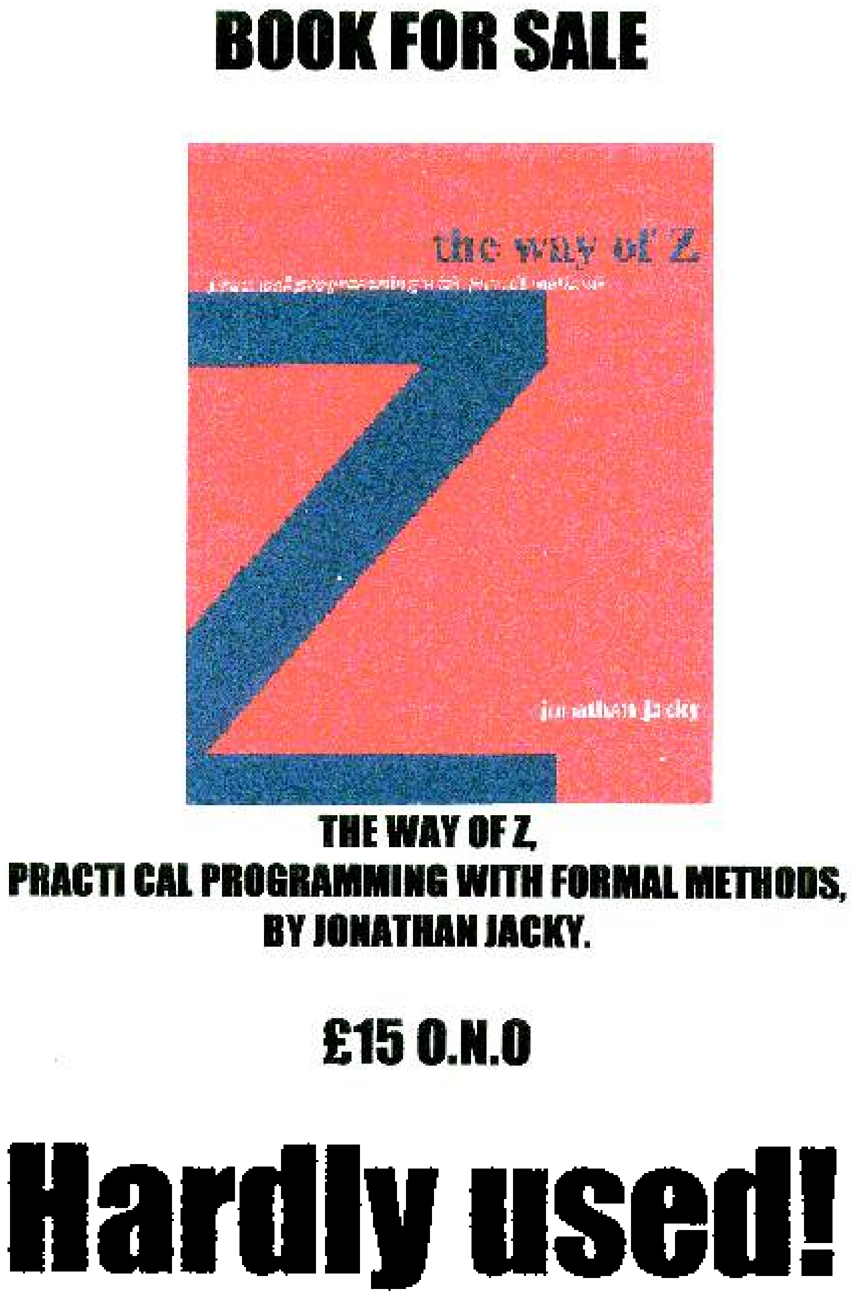
Further books

- Boulanger, J.-L., ed. 2012.
  Formal Methods: Industrial Use from Model to the Code.
  ISTE, Wiley.
  ISBN 978-1848213623.

- Gnesi, S. and Margaria, T. 2012.
  Formal Methods for Industrial Critical Systems: A Survey of Applications.
  IEEE Computer Society Press, Wiley.
  ISBN 978-0470876183.







Education

- Resistance by students
- Resistance even by academics
- Support by professional societies (e.g., BCS accreditation)

Choosing a formal method – difficult

Choosing a formal method can be a fearful thing.

Tools – difficult to use

What it is like to use a mechanical theorem prover.

Technology transfer

- Courses (academia & industry)
- Textbooks (good choice)
- Tools (type-checkers, provers, …)
- Web resources (including Wikipedia)
- Mailing lists (e.g., JISCmail)
- Meetings (conference series)
- Standards (international)

The only thing harder to sell than formal methods.

Standards mandating formal methods

- In highest level of safety and security applications
- From 1990s*
- Also, for formal notations themselves...

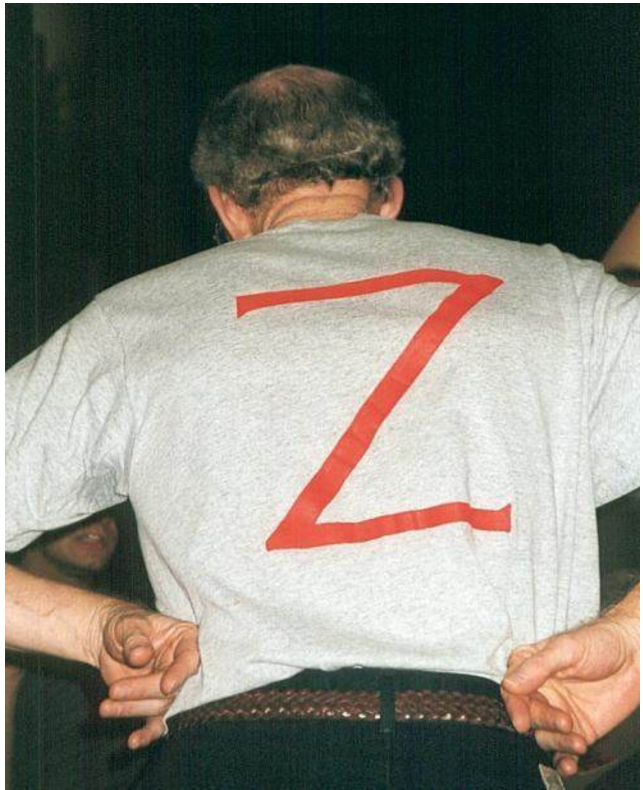
*See:

Bowen, J.P. & Stavridou, V. (1993), Safety-critical systems, formal methods and standards. *Software Engineering Journal*, 8(4):189–209. DOI: https://doi.org/10.1049/sej.1993.0025

Country	Body	Sector	Name	FMs content	FMs mandated	Status (Jan. 1992)	Year
US	DoD	Defence	MIL-STD-882B MIL-STD-882C	No Likely	N/A ?	Standard Draft	1985 1992
US	RTCA	Aviation	DO-178A DO-178B	No Yes	N/A ?	Guideline Draft	1985 1992
Europe	IEC	Nuclear	IEC880	No	N/A	Standard	1986
UK	HSE	Generic	PES	No	N/A	Guideline	1987
Europe	IEC	Generic	IEC65A WG9 IEC65A 122	Yes Yes	No No	Draft Proposed	1989 1991
Europe	ESA	Space	PSS-05-0	Yes	Yes	Standard	1991
UK	MoD	Defence	00-55	Yes	Yes	Interim	1991
US	IEEE	Generic	P1228	No	No	Draft	1992
UK	RIA	Railway	–	Yes	Yes	Draft	1991
Canada	AECB	Nuclear	–	Yes	?	Draft	1991

Example: Z Standard

- ISO/IEC 13568
  - Long process (1990s)
  - Inconsistencies found!
- Final Committee Draft
  – accepted in 2001
- May help tools & industrial application



Case study: National Air Traffic Services

- **2.5 million flights** per year (pre-Covid), covering the UK and eastern North Atlantic.
- **250 million passengers** per year in UK airspace.
- Among the busiest & most complex airspace in the world.
- Provides air traffic control from its centres at **Swanwick**, Hampshire (England) and Prestwick, Ayrshire (Scotland).
- Also provides air traffic control services at **15 UK airports** including **Heathrow**, Gatwick, Stansted, Birmingham, Manchester, Edinburgh, and Glasgow, together with air traffic services at Gibraltar Airport.

NATS

NATS National Air Traffic Services, UK

Swanwick
southern England

www.nats.co.uk

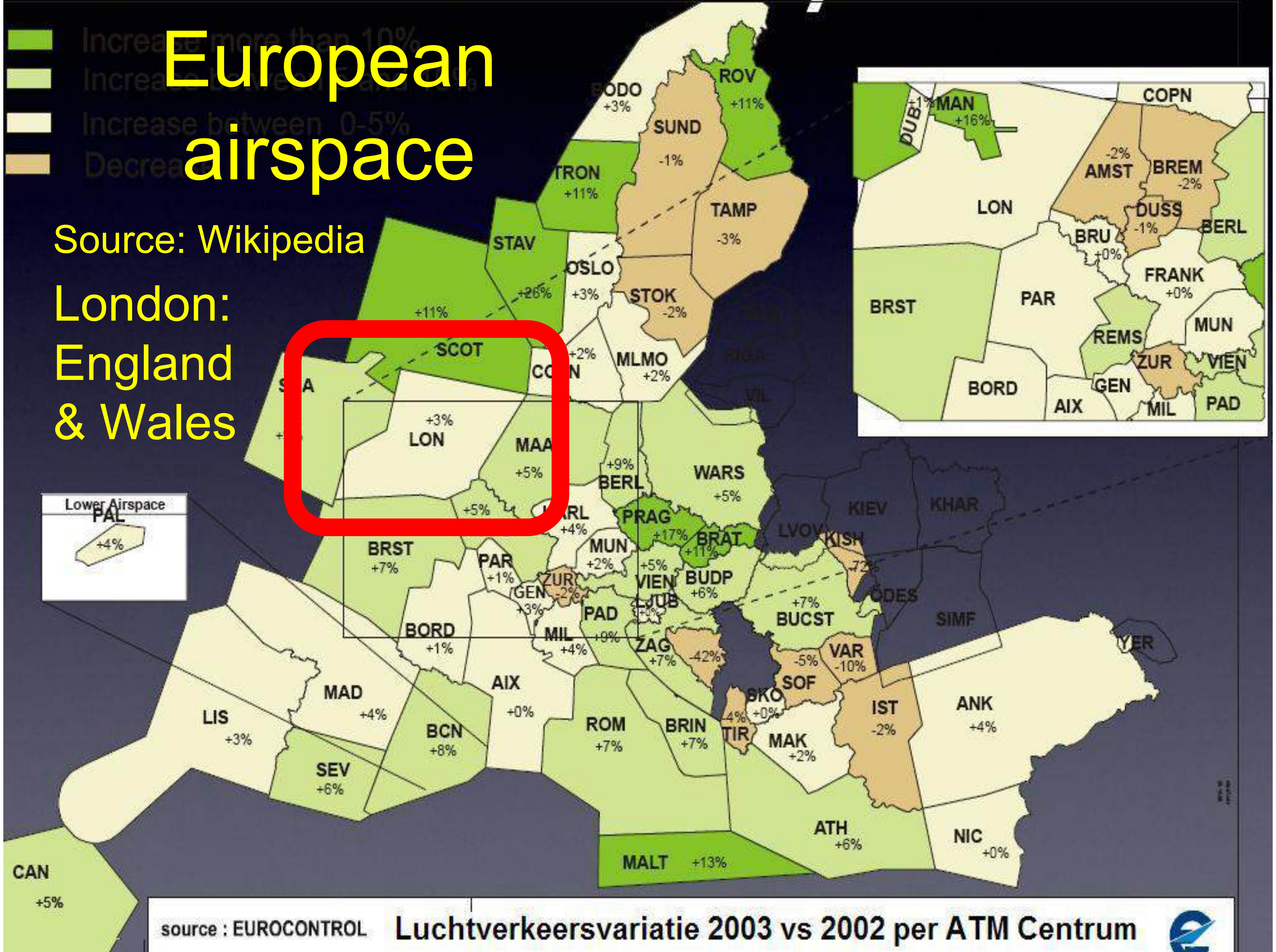
Flight strips on paper

Last flight of Concorde

European airspace

Source: Wikipedia

London: England & Wales



FlightRadar24

www.flightradar24.com

NATS

National Air Traffic Services

- Advertisement at Heathrow Airport →
- Air Traffic Management (ATM)
- Single European Sky ATM Research (SESAR)
- SESAR Joint Undertaking
- www.sesarju.eu
- SESAR project (2004–c.2030!)
- European ATM Master Plan

Formal Methods in Air Traffic Control

NATS, the UK's leading air traffic services provider, has pioneered research and development of advanced air traffic control tools for several years from its simulator and research centre.

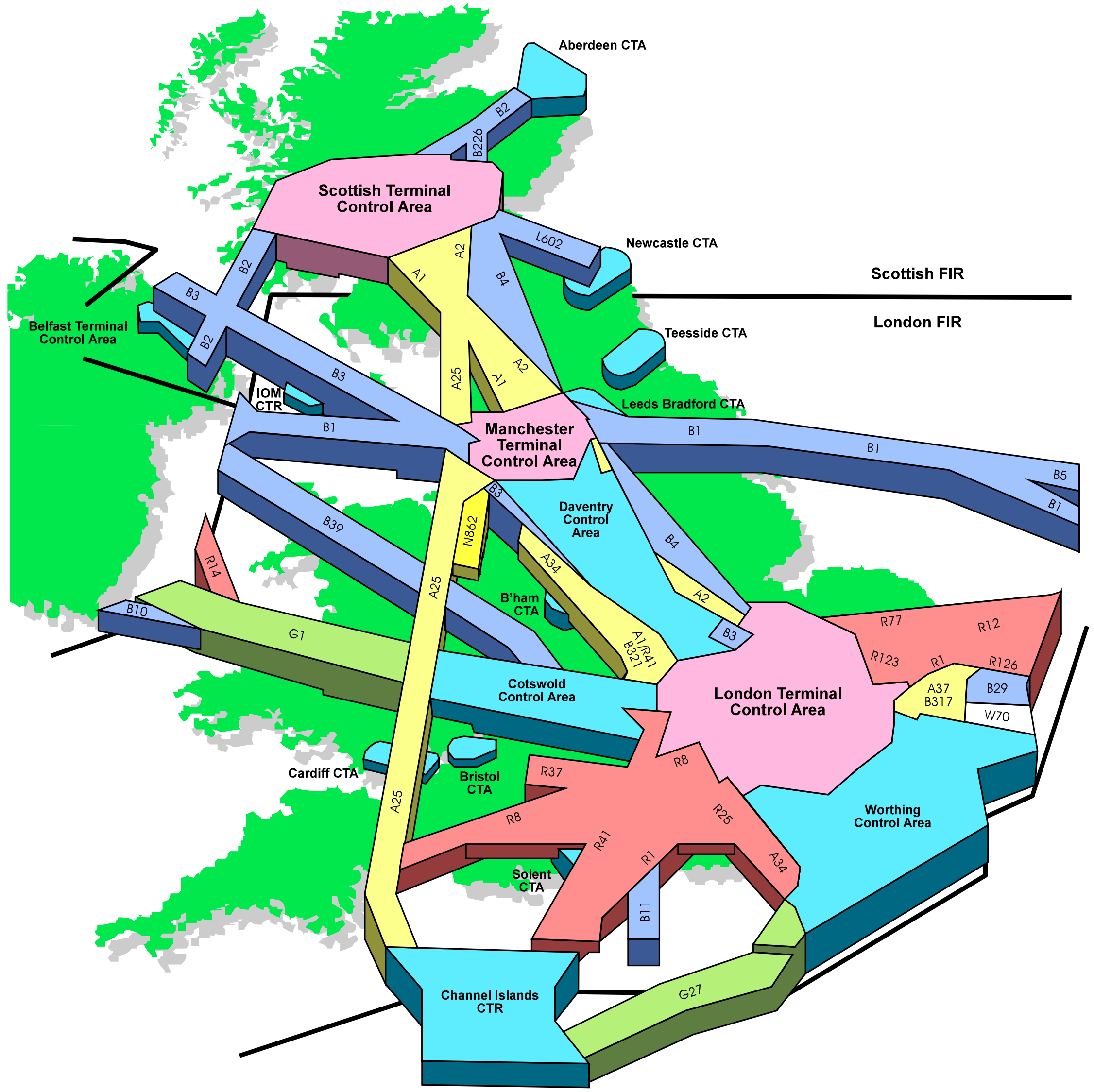
The iFACTS project provides a subset of these tools onto the system at the company's main en-route Control Centre at Swanwick.

www.slideshare.net/AdaCore/white-open-do
www.youtube.com/watch?v=IQMWVqQfm5A

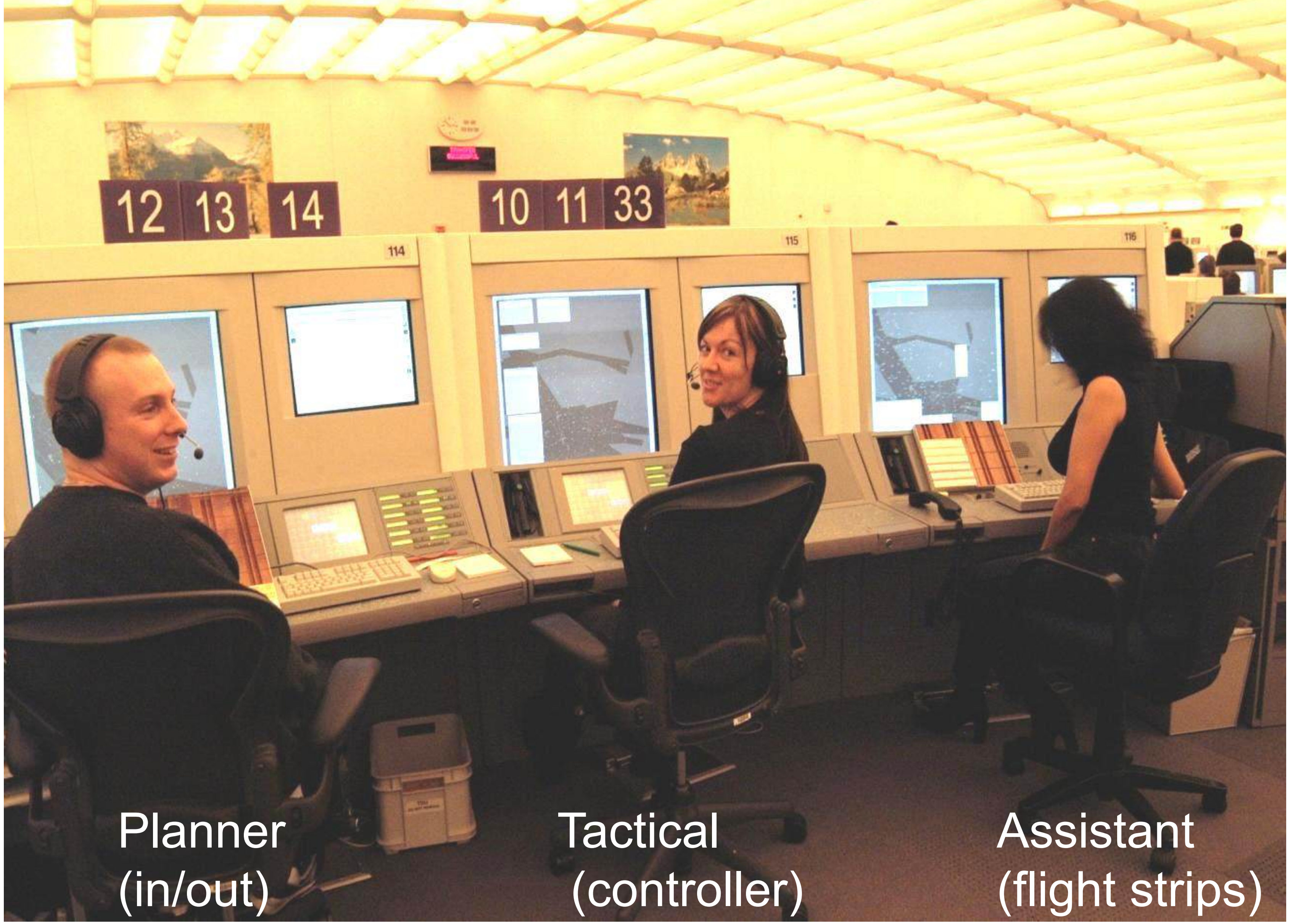
altran PRAXIS

 (Original slides by Neil White) (Capgemini Engineering since 2019)

UK Air Traffic Control



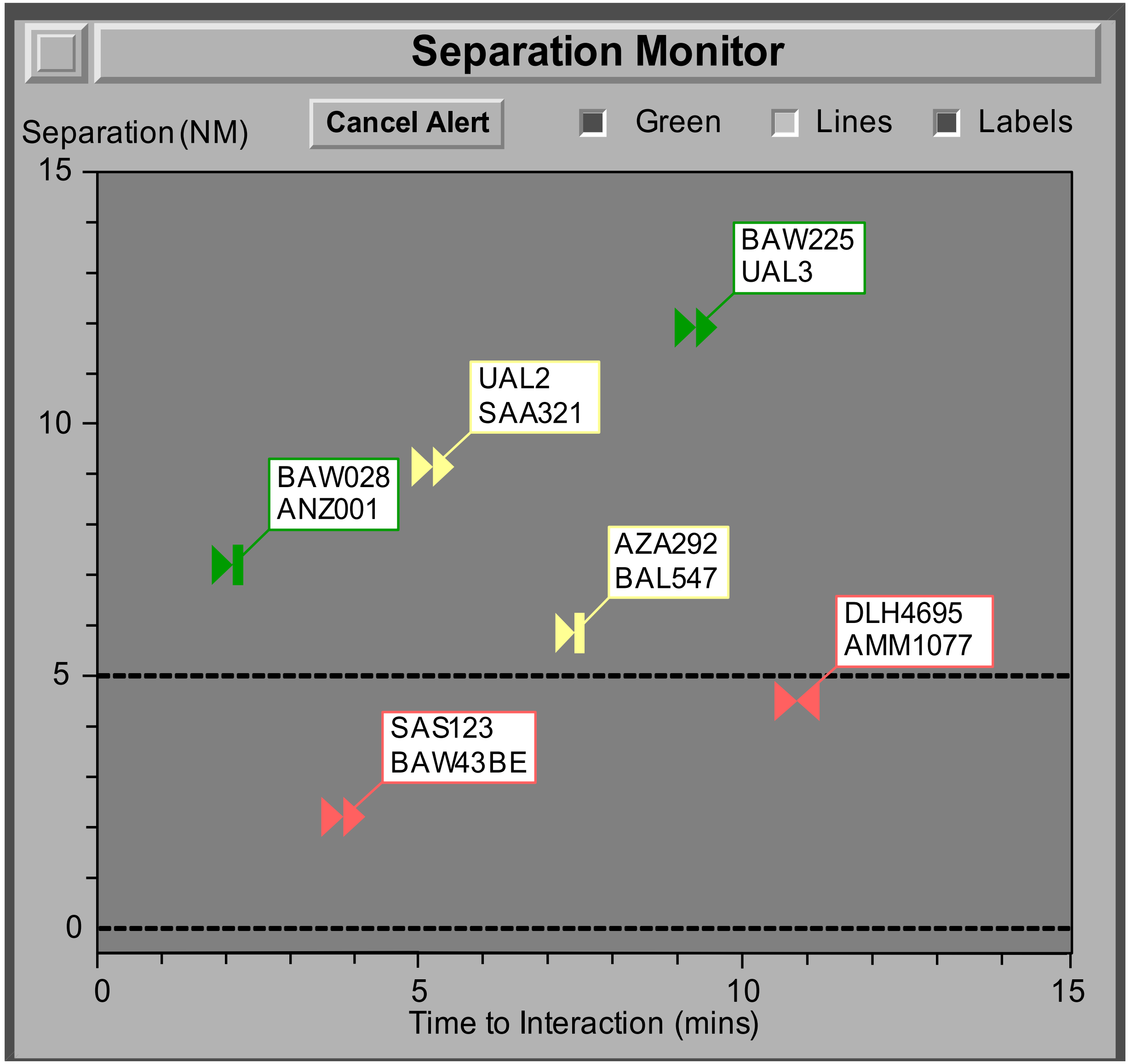
ATC team


12 13 14
10 11 33
114
115
116
Planner
(in/out)
Tactical
(controller)
Assistant
(flight strips)

What is iFACTS?

- iFACTS – Interim Future Area Control Tools Support
- iFACTS provides tools to support the controllers
  - Electronic flight strips replace the paper flight strips.
  - Trajectory tools – including prediction, deviation alerts, and conflict detection – are added.
- iFACTS not an Air Traffic Control (ATC) system
  - Integrated with, but sits alongside, the existing system.

Medium Term Conflict Detection: Separation Monitor



The complete iFACTS specification

- The functional specification
  - Z notation
- The algorithm specification
  - Mathematics (Mathematica)
- The Human-Machine Interface (HMI) specification
  - State tables
- The rest of the specification
  - English!

The Z specification

Every flight is associated with an aircraft type if its aircraft type name matches.

Every flight is associated with a performance model. If there is no model corresponding to the aircraft type, then this is a default model. If there is a filed speed up to *maxPistonSpeed*, then this is *unknownPiston*; if there is a filed speed above this but no greater than *maxTurbopropSpeed*, it is *unknownTurboprop*; otherwise it is *unknownJet*.

The set *fwpNASDeletedFlights* is those flights that have been NAS deleted. It is used by Recognised Flights.

FWPFlightsDerivedAssociations

$$FWPFlightsAssociations$$

$$fwpFlightAircraftType : FLIGHT \pfun AIRCRAFTTYPENAME$$

$$fwpFlightPerformanceModel : FLIGHT \pfun PERFORMANCEMODEL$$

$$fwpNASDeletedFlights : \mathbb{F}\ FLIGHT$$

$$fwpFlightAircraftType = \{f : fwpFGFlights ; a : aircraftTypes \mid (fwpFlightState\ f).aircraftTypeName = a \bullet f \mapsto a\}$$

$$\begin{aligned}
&fwpFlightPerformanceModel = \\
&\{f : (fwpFGFlights \setminus fwpBlockers); model : PERFORMANCEMODEL \mid \\
&\quad (\mathbf{let}\ speed == (fwpFlightState(f)).filedSpeed \bullet \\
&\quad\quad model = \mathbf{if}\ the\ speed \leqslant maxPistonSpeed\ \mathbf{then}\ unknownPiston \\
&\quad\quad\quad \mathbf{else\ if}\ the\ speed \leqslant maxTurbopropSpeed\ \mathbf{then}\ unknownTurboprop \\
&\quad\quad\quad \mathbf{else}\ unknownJet)\} \\
&\quad \oplus (fwpFlightAircraftType \mathbin{;} typePerformanceModel)
\end{aligned}$$

$$fwpNASDeletedFlights = \{ f : fwpFGFlights \mid (fwpFlightState\ f).nasDeleted = True \}$$

Z training

- Z reader training
  - 3-day course; fluency then comes after 1 week on the job.
  - Trained 75 people to read Z.
  - Engineers, domain experts, ATCOs.
- Z writer training
  - 3-day course, fluency then comes after 3 months on the job.
  - Trained 11 people to write Z.
  - All engineers.

Z tools

- Z written in Microsoft Word
  - To get acceptance, you need to work with what people know.
  - Supported by Word Add-ins.
    - A Z character set.
    - A simple interface to the *fuzz* type checker.
    - A graphical representation tool.

Z tools

- Advantages
  - Easy to develop commentary and Z together.
  - Hyperlinking of fuzz errors back to source.
  - Cross-referencing of Z names in final document.
- Disadvantages
  - All the problems of large Word documents.
  - Tools can be slow on 1,000 page documents.
  - Merging branches (for different releases) painful.
- Possible future
  - Open Office XML

The state machine specification

	Button 1	Checkbox 1
State 1	State 2	N/A
State 2	State 1	State 3
State 3	State 1	State 2

Transition Actions

State 1 -> State 2 : De-select Checkbox 1

State machine training & tools

- Training
  - So trivial that we don't train!
  - People "just get it".
- Tools
  - Err .... None.

The SPARK implementation

- SPARK Ada
  - An annotated subset of Ada.
- 150 KSLOC (Logical)
- RTE (Run-Time Exception) Proof
  - Formal partial correctness proof against specification not considered cost-effective.

Code

```
function Segment_Group_FL_Occupancy
  (Segs_Group : PIO_Data.Segment_Group_Array_T;
   Quantity   : PIO_Data.Trajectory_Index_T)
   return Altitudes.Level_Range
is
   The_Range  : Altitudes.Level_Range;
   Temp_Range : Altitudes.Level_Range;
begin

   --  By virtue of the fact that this procedure has been called means
   --  that the level ranges must be populated so set to a senseless
   --  null value guaranteed to be overwritten
   The_Range.Lower := Altitudes.Flight_Level_T'Last;
   The_Range.Upper := Altitudes.Flight_Level_T'First;

   for Idx in PIO_Data.Trajectory_Index_T range 1 .. Quantity loop
      --# assert Quantity = Quantity%;

      --  Must have a standard occupancy at the very least so check for that
      if MTCD_Types.Get_Standard_Occupancy (Segs_Group (Idx)).Exists then

         Temp_Range := Segment_FL_Occupancy (Segs_Group (Idx));

         The_Range.Lower :=
           Altitudes.Flight_Level_T'Min (The_Range.Lower,
                                         Temp_Range.Lower);

         The_Range.Upper :=
           Altitudes.Flight_Level_T'Max (The_Range.Upper,
                                         Temp_Range.Upper);
      end if;

   end loop;

   return The_Range;

end Segment_Group_FL_Occupancy;
```

SPARK training and tools

- 57 people trained in SPARK
  - Mostly contractors and clients.
  - Diverse programming background.
  - All SPARK coders also Z readers.
- Effective as SPARK coders immediately
- Picking up RTE proof takes longer.
  - About 2 months.
- How long to pick up formal correctness proofs?
  - No data, but suspect longer again.

The SPARK toolset:

- Examiner.
- Proof Simplifier.
- Proof Checker.

Test Design

2.2.1.18 TPDeviationRequests

Summary

Requests the required deviation trajectories.

This is a non-conditional schema.

Partitions

There are two equivalence classes:

1 Flight is not radar supported, so no information.

2 Flight is radar supported.

The output condition in the first equivalence class is that there is no request. This can also occur when there are no deviation trajectories, so that input condition should be tested as well.

It is stated within the FPM process specification that the number of deviation requests will be either none, one or two ([4] section 13.2.13.2). We should test for each of these conditions separately (since 0 and 2 are boundary conditions).

Test Conditions

TPDeviationRequests	1	2	3	4
$fpmData!? = nil$	●	○	○	○
$(the\ fpmData!?).fpmDeviationTrajectories = \emptyset$		●	○	○
$deviationReqs = \emptyset$	●	●	○	○
$\#deviationReqs = 1$	○	○	●	○
$\#deviationReqs = 2$	○	○	○	●

The challenge of test design

$$
\begin{array}{l}
\underline{\ TPRemoveMultiplePIOs\ } \\
\Delta TP \\
tpFlights!? : \mathbb{P} FLIGHT \\
piosToRemove : \mathbb{P} PIO \\
\hline
\exists\, deletedDirectPIOs, deletedGroupedPIOs : \mathbb{P} PIO \mid \\
\quad deletedDirectPIOs = flightPIO \,(\!|\, tpFlights!? \,|\!)\, \cap piosToRemove \\
\quad \wedge\, deletedGroupedPIOs = pioPIOGroup^{\sim} \,(\!|\, deletedDirectPIOs \,|\!)\, \bullet \\
\qquad flightPIO' = flightPIO \rhd deletedDirectPIOs \\
\qquad \wedge\, pioPIOGroup' = pioPIOGroup \rhd deletedDirectPIOs \\
\qquad \wedge\, pioGroupDisplayPIO' = deletedDirectPIOs \lhd pioGroupDisplayPIO \\
\qquad \wedge\, pioState' = (deletedDirectPIOs \cup deletedGroupedPIOs) \lhd pioState \\
nominalVerticalProfiles' = \\
\quad \mathbf{if}\ fwpHookedFlight \cap tpFlights!? \neq nil \\
\quad \mathbf{then}\ nil\ \mathbf{else}\ nominalVerticalProfiles
\end{array}
$$

How many potential tests for this fragment?

The challenge of test design

- If you just turn the handle there are 1134 conditions to test.
- But if you work at it hard enough you can cover the required subset in just 6 test scripts.
- Formal methods are not a substitute for initiative.

Mathematica tools & training

- Algorithms are specified in pure mathematics.
- Generate test cases as usual.
- Create a test reference implementation in Mathematica.

- Small team – only 5 trained.
- Reference model has similar defect density to SPARK implementation.
- Limited conclusions to draw from such a small activity.

Case study conclusions

- Formal methods are applicable to all phases of the lifecycle.
- Training engineers is not a barrier
  - It's a one-off cost
  - Data shows that training is easy and cheap.
- Tool support is vital
  - The Achilles heel of formal methods
    - Except the SPARK Examiner!

Tracing

- Completeness of coverage
  - e.g., testing all parts of a Z specification
- DOORS tool
  - Dynamic Object-Oriented Requirements System
- Link all specification components with test case(s
  - or argument for safety case
- Flag unlinked components
- Also, visualization of schema structure

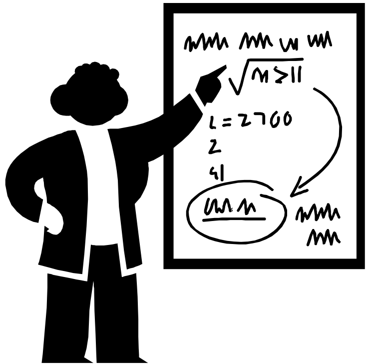
Subsequent iFACTS developments

- iFACTS in operation (2011) – 18 minutes of prediction, up to 40% capacity increase in some sectors
- Traffic Load Prediction Device (TLPD):
  - Forecast air traffic load up to 4 hours ahead
  - Plan workloads for optimum traffic flows
- iFACTS – winner of the Duke of Edinburgh Navigation Award for Technical Achievement (2013)
- MoD use for military air traffic control (2014)
- FourSight, successor to iFACTS (2017) for Swanwick/Prestwick – European SESAR compliant

How Important is mathematics to the software practitioner?

Some consider it unimportant … !

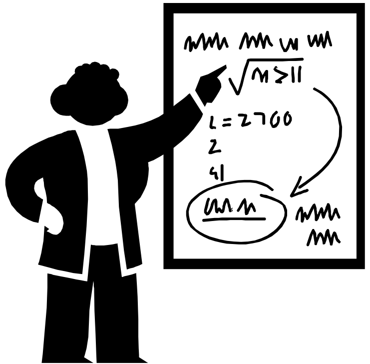
— Robert L. Glass
IEEE Software, Nov./Dec. 2000



Mathematics debates

Some consider it important …

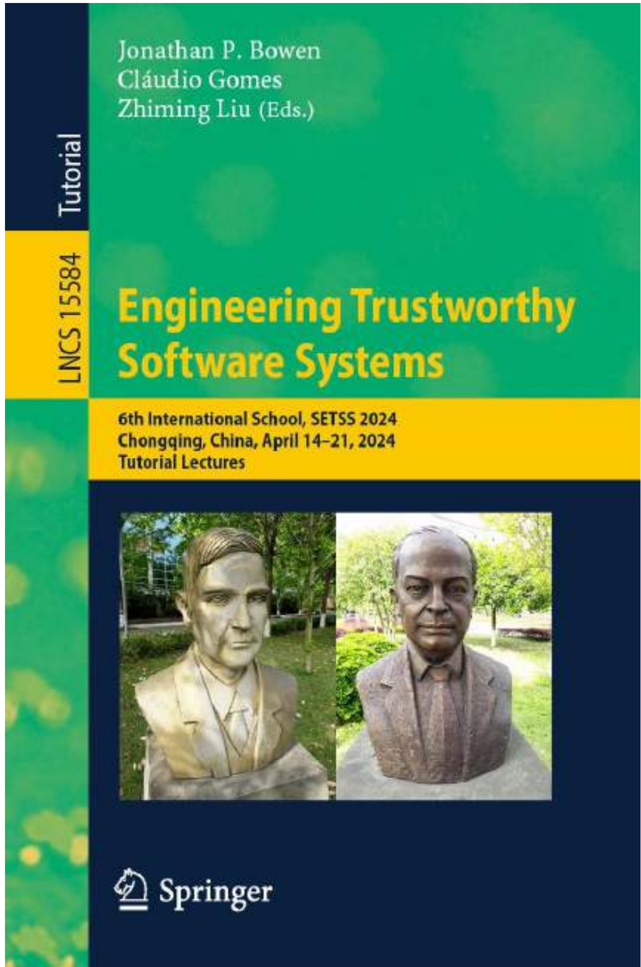
— William W. McMillan *et al.*, Letters
IEEE Software, Jan./Feb. 2001



The debate has continued …

SETSS: Engineering Trustworthy Software Systems

- Annual Spring School at Southwest University, Chongqing, China, & now ISCAS, Beijing, China
- Held 2014–2019, restarted after COVID in 2024
- Week-long tutorials by international experts, for graduate students from China *and elsewhere*
- Tutorial proceedings in Springer LNCS
- State of the art in formal methods & related research
- Cf. annual Marktoberdorf Summer School in Europe (6–15 August 2025)



SETSS

15–21 April 2024

www.rise-swu.cn/SETSS2024

- SWU, Chongqing, China
- Seven tutorials over 5 days
- Workshop over 2 days

SETSS

17–23 April 2025

tis.ios.ac.cn/SETSS2025

- ISCAS, Beijing, China
- 2 days of workshop talks
- 5 days of longer tutorials

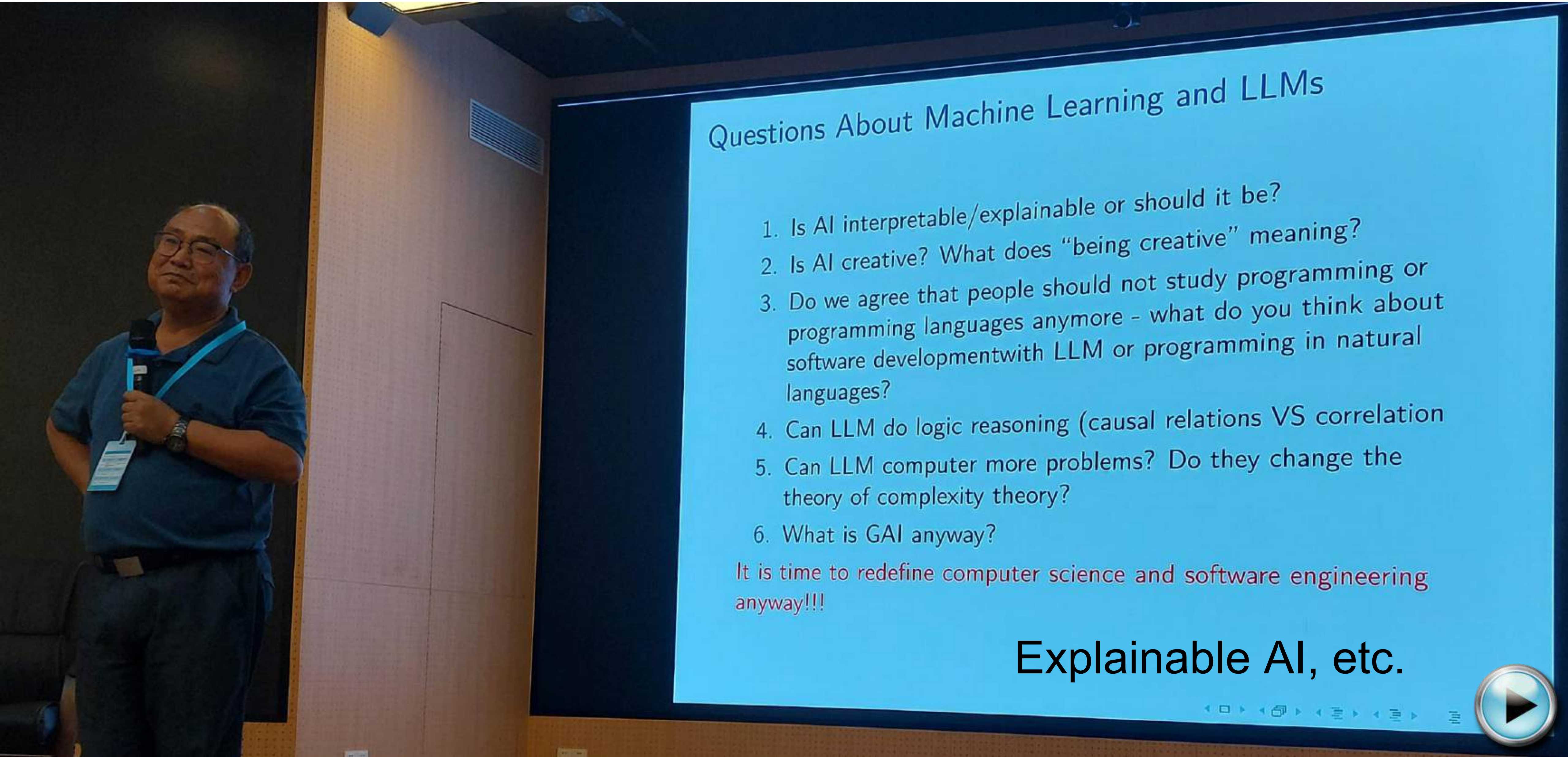
ISCAS

中國科學院軟件研究所
Institute of Software Chinese Academy of Sciences

中国计算机学会 CCF
中國計算機學會
CHINA COMPUTER FEDERATION

SOUTHWEST UNIVERSITY 1906
西南大學
SOUTHWEST UNIVERSITY

Formal Methods and AI – questions



Explainable AI, etc.

AI – large "learned" models

A Possible Consequence: Consider it a Feature

"AI is becoming synonymous with large learned models. [. . .] Given this state of affairs, it is increasingly clear that at least part of AI is straying firmly away from its "engineering" roots. It is increasingly hard to consider large learned systems as "designed" in the traditional sense of the word, with a specific purpose in mind. After all, we don't go around saying we are "designing" our kids [. . .] AI becomes an ersatz natural science studying large learned artifacts. Of course, there might be significant methodological resistance and reservations to this shift . After all, CS has long been used to the "correct by construction" holy grail, and from there it is quite a shift to getting used to living with systems that are at best incentivized ("dog trained") to be sort of correct — sort of like us humans!"

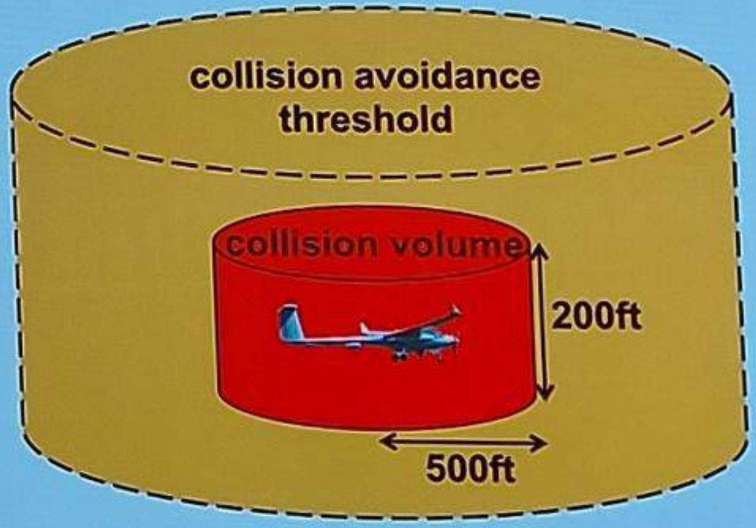
Subbarao Kambhampati: "AI as (an Ersatz) Natural Science?" CACM, June 8, 2022

"Correct by construction" vs. "dog trained"

Carl von Ossietzky Universität Oldenburg

Line of Attack: Correctness

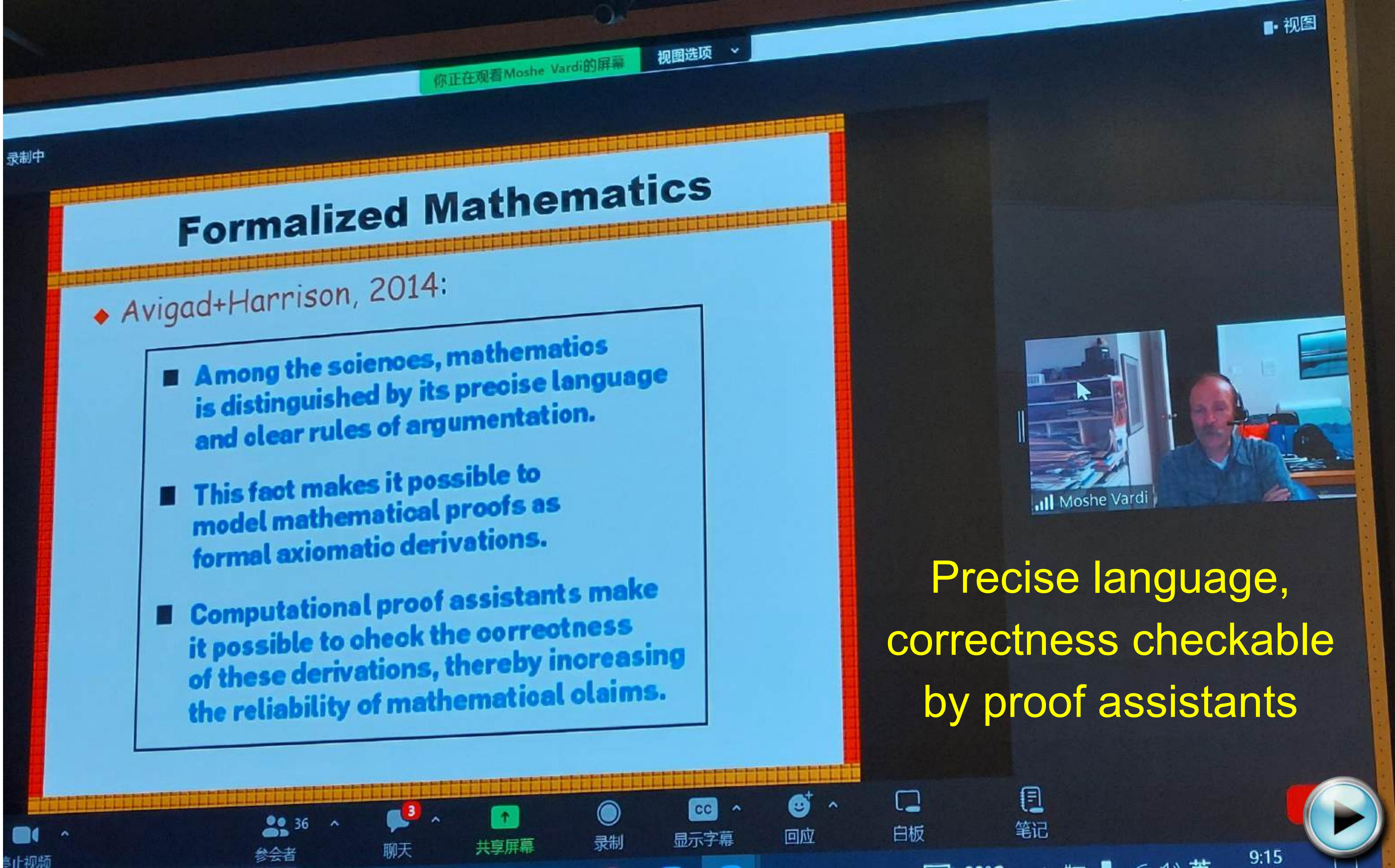
- Rigorous specification of intended functionality is *partially* possible by using don't care regions
  - Either as open-loop properties expressing when, regarding current state observed by the sensors, the system ought issue which advisories, like in (examples simplified)
    - If no other flight device is within a radius of 20 miles then no advisory ought be given.
    - As soon as another flight device is within a radius of 4 miles, a conflict-free advisory has to be issued.
  - or as closed-loop requirements:
    - starting from *any* situation obeying the system bounds regarding speeds, initial separations, etc., the collision volume remains unreachable when all advisories issued are followed.



Formal methods and correctness

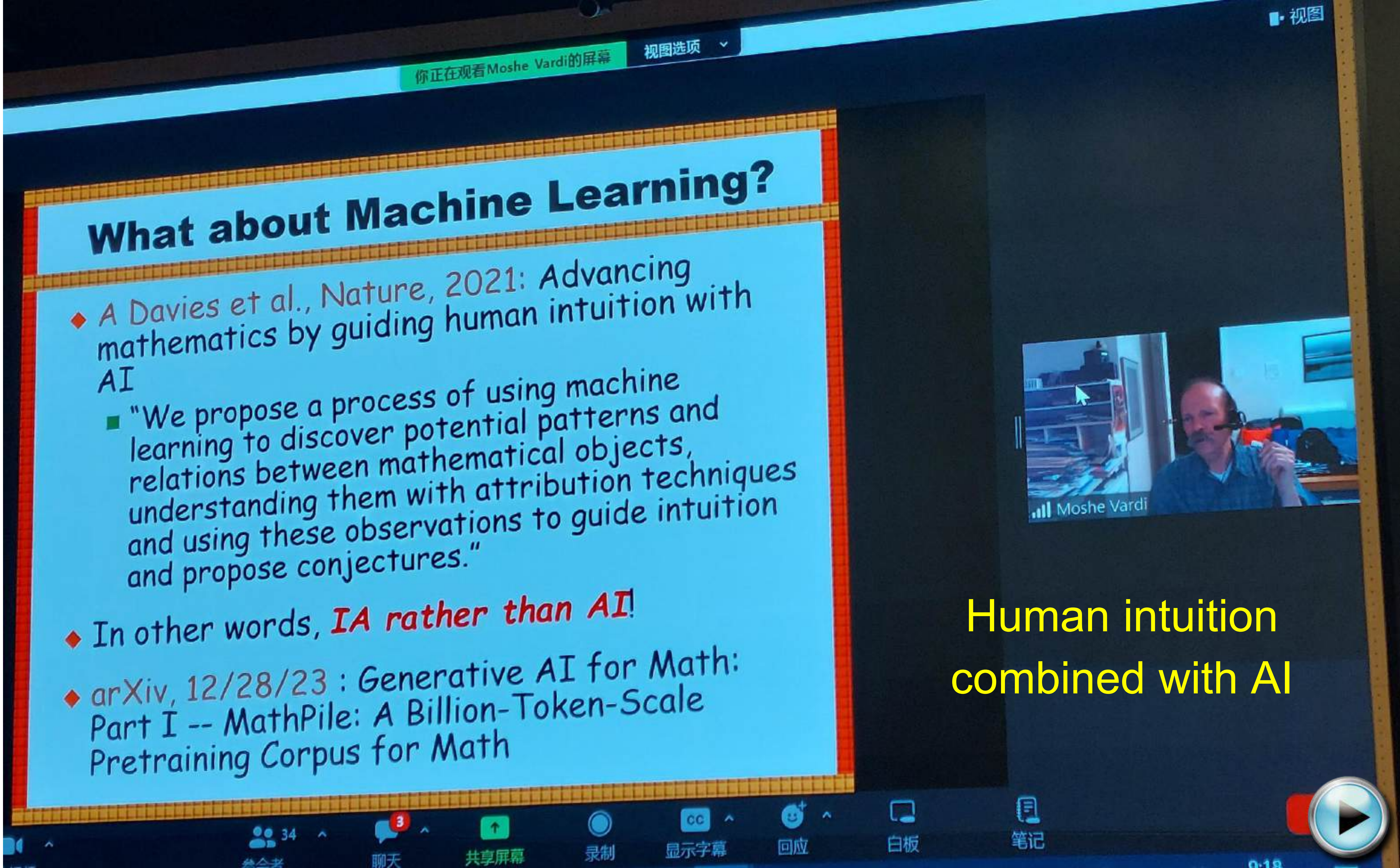
Rigorous specification

Formalized mathematics



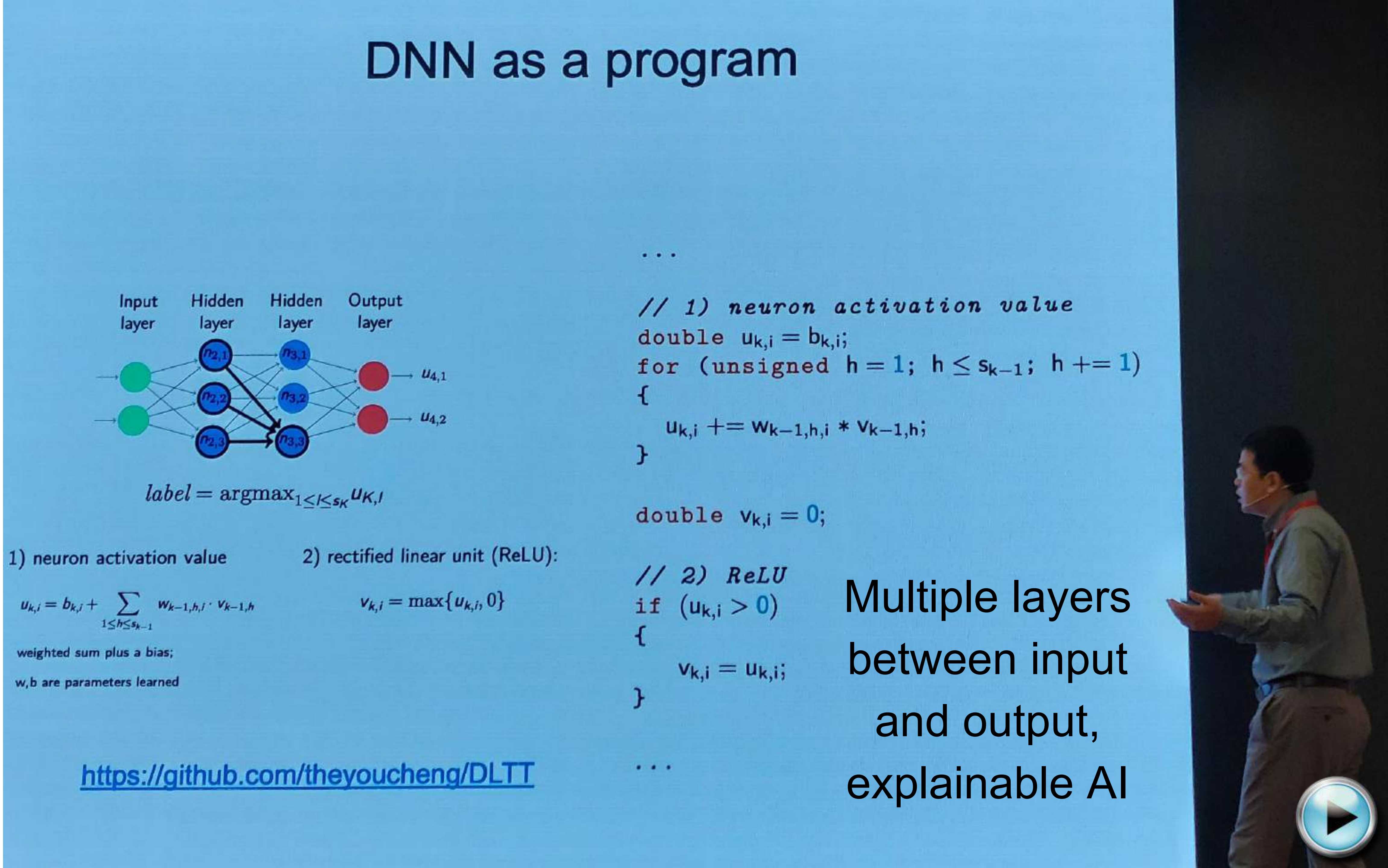
Precise language, correctness checkable by proof assistants

Machine learning



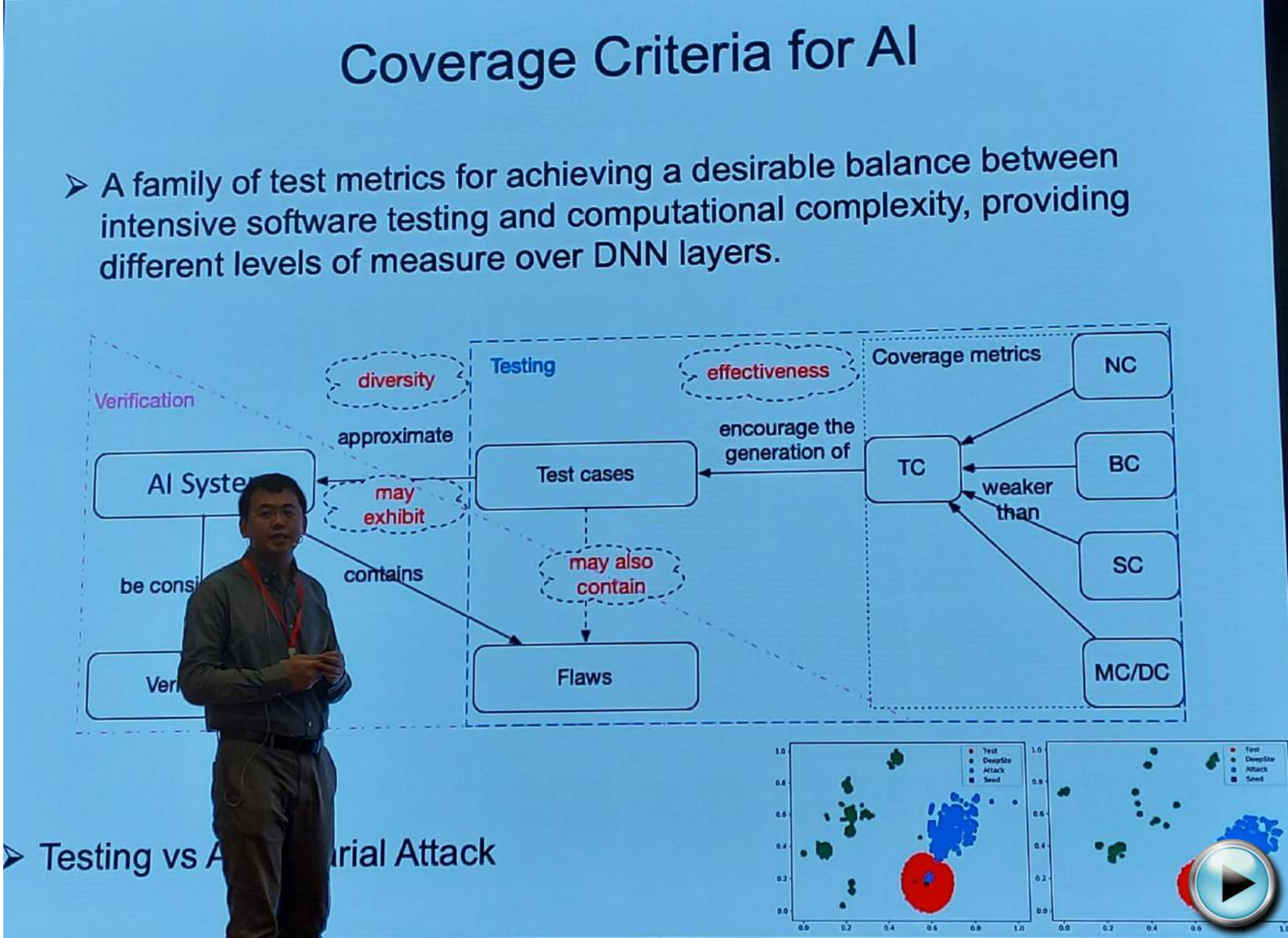
Human intuition combined with AI

Deep Neural Networks (DNN)



Multiple layers between input and output, explainable AI

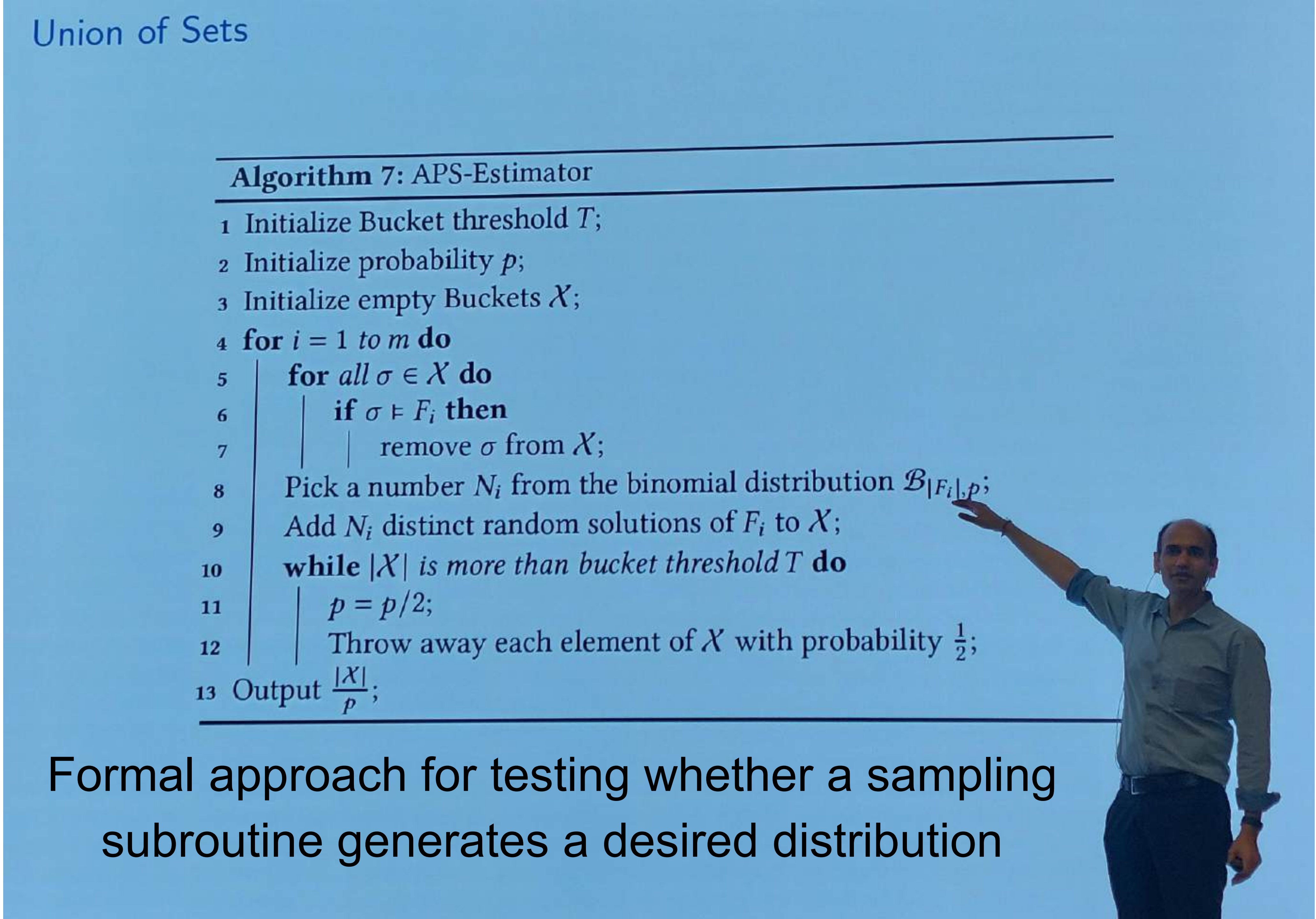
Coverage criteria for AI



Test coverage for DNNs

Formal methods and testing

Reliance on probability distributions



Formal approach for testing whether a sampling subroutine generates a desired distribution

Predictions dangerous

“ . . . these formal methods are the key to writing much better software. Their widespread use will revolutionise software writing, and the economic benefits will be considerable – on a par with those of the revolution in civil engineering during the last century.”

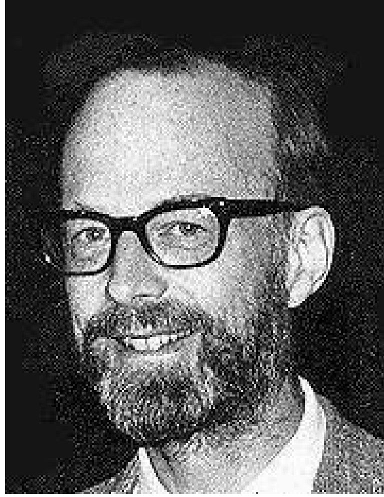
— Brian Oakley (1927–2012), *Alvey Achievements*, June 1987

Compare AI!

Future developments

- An engineering approach
- Proof vs. calculation
- “Light” approach (specification)
- Improved tools (Moore’s law helps)
- International standards
- Education / training (for all personnel)
- Unification of approaches?

Unified theory? Cf. physics



"The construction of a single mathematical model obeying an elegant set of algebraic laws is a significant intellectual achievement; so is the formulation of a set of algebraic laws characterising an interesting and useful set of models."

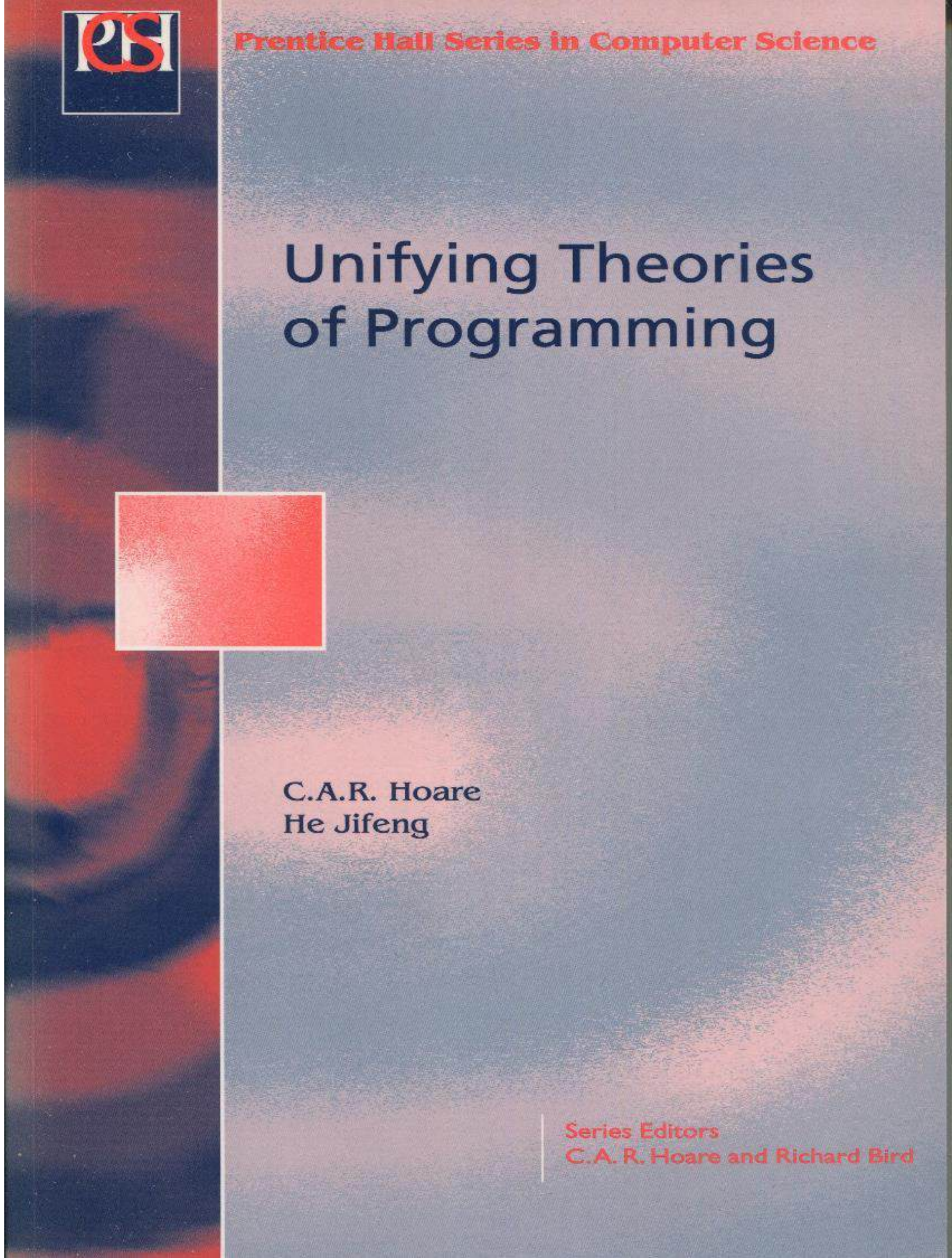
— Sir Tony Hoare, 1993

Operational, Denotational, Algebraic semantics

Unifying Theories of Programming

- Tony Hoare & Jifeng He
- Prentice Hall, 1998
- http://www.unifyingtheories.org

- UTP international symposium
- First symposium 2006, UK
- Springer LNCS proceedings



Future developments

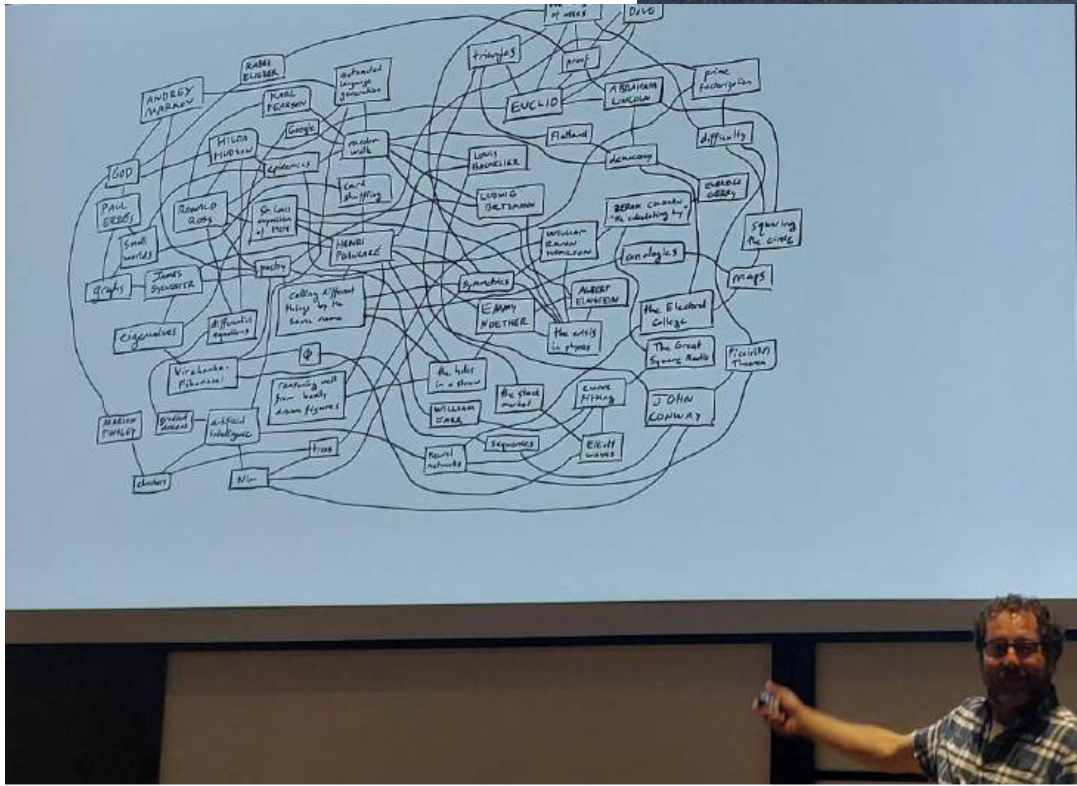
- Safety-critical systems
- Security (e.g., smartcards)
- Harmonization of engineering practices
- Practical experience
- Assessment and measurement
- Technology transfer investment
- Use with AI, LLMs, etc… perhaps most promising!

- Computer science uses decades-old, even centuries-old mathematics
- So, see what mathematicians are doing now for the future

Ronald Ross to ChatGPT:
The birth and strange life of a random walk

Mathematical Institute, Oxford, 26 June 2024

— Jordan Ellenberg, Univ. of Wisconsin–Madison (b. 1971)



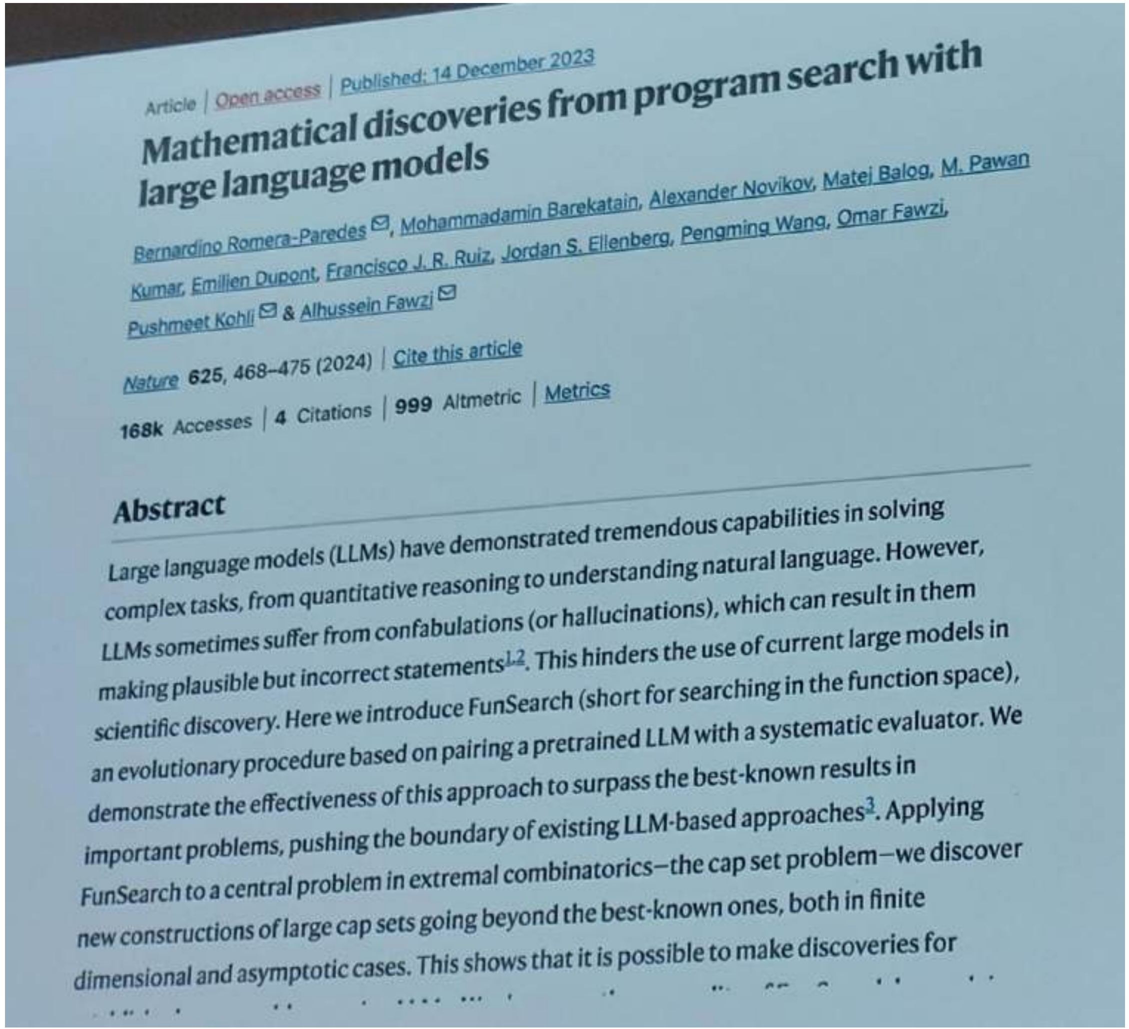
youtube.com/watch?v=08FGB5x090M

Mathematical discoveries from program search and large language models

Nature, vol. 625,
pp. 468–475 (2024)

- Prospect: AI could suggest outline proofs with human interactive help for detail
- Could this approach work for program generation/proof?

nature.com/articles/s41586-023-06924-6



Article | Open access | Published: 14 December 2023

Mathematical discoveries from program search with large language models

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli & Alhussein Fawzi

Nature 625, 468–475 (2024) | Cite this article

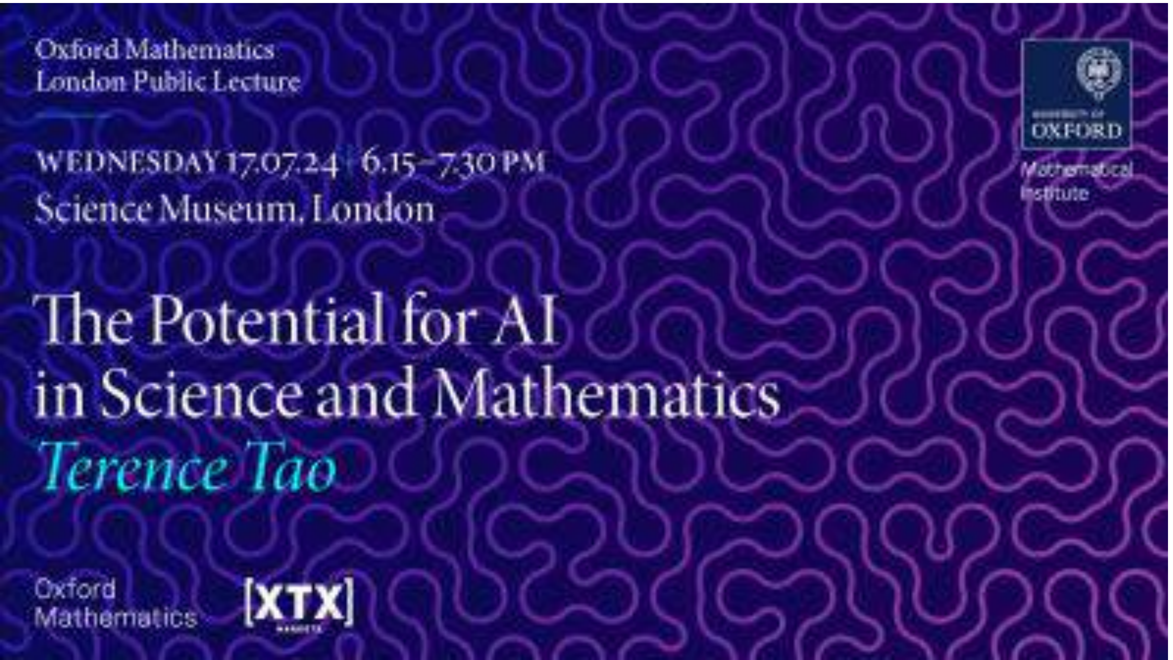
Abstract

Large language models (LLMs) have demonstrated tremendous capabilities in solving complex tasks, from quantitative reasoning to understanding natural language. However, LLMs sometimes suffer from confabulations (or hallucinations), which can result in them making plausible but incorrect statements[1,2]. This hinders the use of current large models in scientific discovery. Here we introduce FunSearch (short for searching in the function space), an evolutionary procedure based on pairing a pretrained LLM with a systematic evaluator. We demonstrate the effectiveness of this approach to surpass the best-known results in important problems, pushing the boundary of existing LLM-based approaches[3]. Applying FunSearch to a central problem in extremal combinatorics—the cap set problem—we discover new constructions of large cap sets going beyond the best-known ones, both in finite dimensional and asymptotic cases. This shows that it is possible to make discoveries for

The Potential for AI in Science and Mathematics

Science Museum, London, 17 July 2024

(c/o Oxford Mathematics)



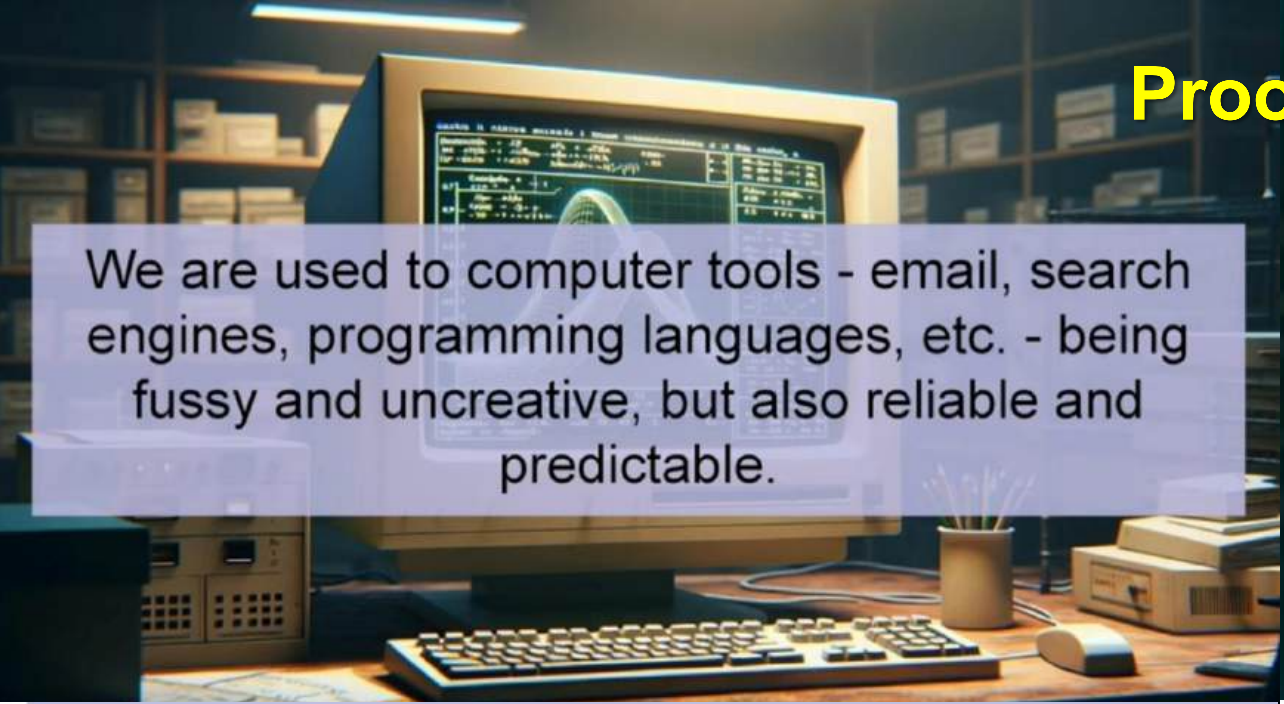
— Terence Tao FAA FRS,

UCLA (b. 1975)

2006 Fields Medalist

youtube.com/watch?v=_sTDSO74D8Q

Proof tools reliable vs. LLMs unreliable



We are used to computer tools - email, search engines, programming languages, etc. - being fussy and uncreative, but also reliable and predictable.

But now we have Large Language Models such as ChatGPT, that can understand natural language requests, and generate endless creative text and image outputs - but without reliability, predictability, or accuracy.

Sometimes, the outputs can be quite impressive...

GPT-4

Prompt:

Suppose g is a continuous smooth function such that for every $x > 0$, there is **one and only one** $y > 0$ such that $g(x) + g(y) \leq 2xy$. Can you prove that $g(x) = x^2$?

GPT-4:

There are a few possible ways to approach this problem, but one possible method is as follows:

- Assume, for the sake of contradiction, that $g(x)$ is not equal to x^2 for some $x > 0$.
- Then, either $g(x) < x^2$ or $g(x) > x^2$ for that x.
- If $g(x) < x^2$, then let $y = x$. Then $g(x) + g(y) = 2g(x) < 2x^2 = 2xy$.
- If $g(x) + g(y) < 2xy$, then $g(x) + g(y)$ is strictly less than the upper bound of $2xy$ for positive x and y.
- This means that there is some room to increase y slightly and still satisfy the inequality $g(x) + g(y) \leq 2xy$.
- For example, let $y' = y + \epsilon$, where ϵ is a small positive number such that $g(x) + g(y') < 2xy$. This is possible because g is continuous and smooth, so it does not have any jumps or kinks that would prevent such a choice of epsilon.
- Then, y' is another positive number such that $g(x) + g(y') \leq 2xy'$, which contradicts the assumption that y is the one and only one positive number that satisfies this condition given x.
- Therefore, if $g(x) < x^2$, then $g(x) + g(y)$ cannot be strictly less than $2xy$, and it must be equal to $2xy$.
- If $g(x) > x^2$, then let y^* be the unique positive solution of $g(x) + g(y) = 2xy$. Then $g(y^*) < (y^*)^2$, since otherwise we would have $g(x) + g(y^*) > 2xy$, which contradicts the existence of y^* given x.
- But then, by the same argument as before, we can let $x = y = y^*$ and obtain a contradiction, since $g(y) + g(y) = 2g(y) < 2y^2 = 2yy$, which contradicts the uniqueness of y given y.
- Therefore, $g(x)$ must be equal to x^2 for all $x > 0$, as required.

... and sometimes, they are not.

Maths Olympiad proof vs. arithmetic

7 * 4 + 8 * 8 =

120 [...]
7 * 4 + 8 * 8
= (7 * 4) + (8 * 8) (we can use parentheses to group multiplication terms, but they are not necessary)
= 28 + 64 (we perform the multiplication inside the parentheses first)
= 92 (we perform the addition last)

Wait but you started by saying 7 * 4 + 8 * 8 = 120?

That was a typo, sorry. The correct answer is 92, as shown in the solution.

Reliable proof tools vs. unreliable LLMs

How can we use a tool that is powerful, but unreliable?
For applications where mistakes can cause real harm (e.g., medicine, financial decisions, personal advice and therapy), one must be cautious, despite the great potential benefits.

But there are more promising use cases if the downside of producing an incorrect answer is low (e.g., if one wants to generate background images for slides).

In particular, in situations where the AI output can be independently verified, there are many promising applications, both in the sciences and in mathematics.

A particularly promising use case for AI is in mathematics:

- Little downside to producing incorrect proofs of mathematical theorems
- Proofs can be independently verified by proof assistants
- Training AI to improve their mathematical reasoning may be of use in broader tasks

Verify AI mathematical output

Proof assistants for formalization

- Proof assistants are computer languages specializing in verifying that an algorithm or proof actually works as intended.
- They are used to verify routines for critical electronics (e.g., avionics), as well as mathematical proofs.
- Proof formalization is time consuming, but getting faster.

…maths …and also programs?

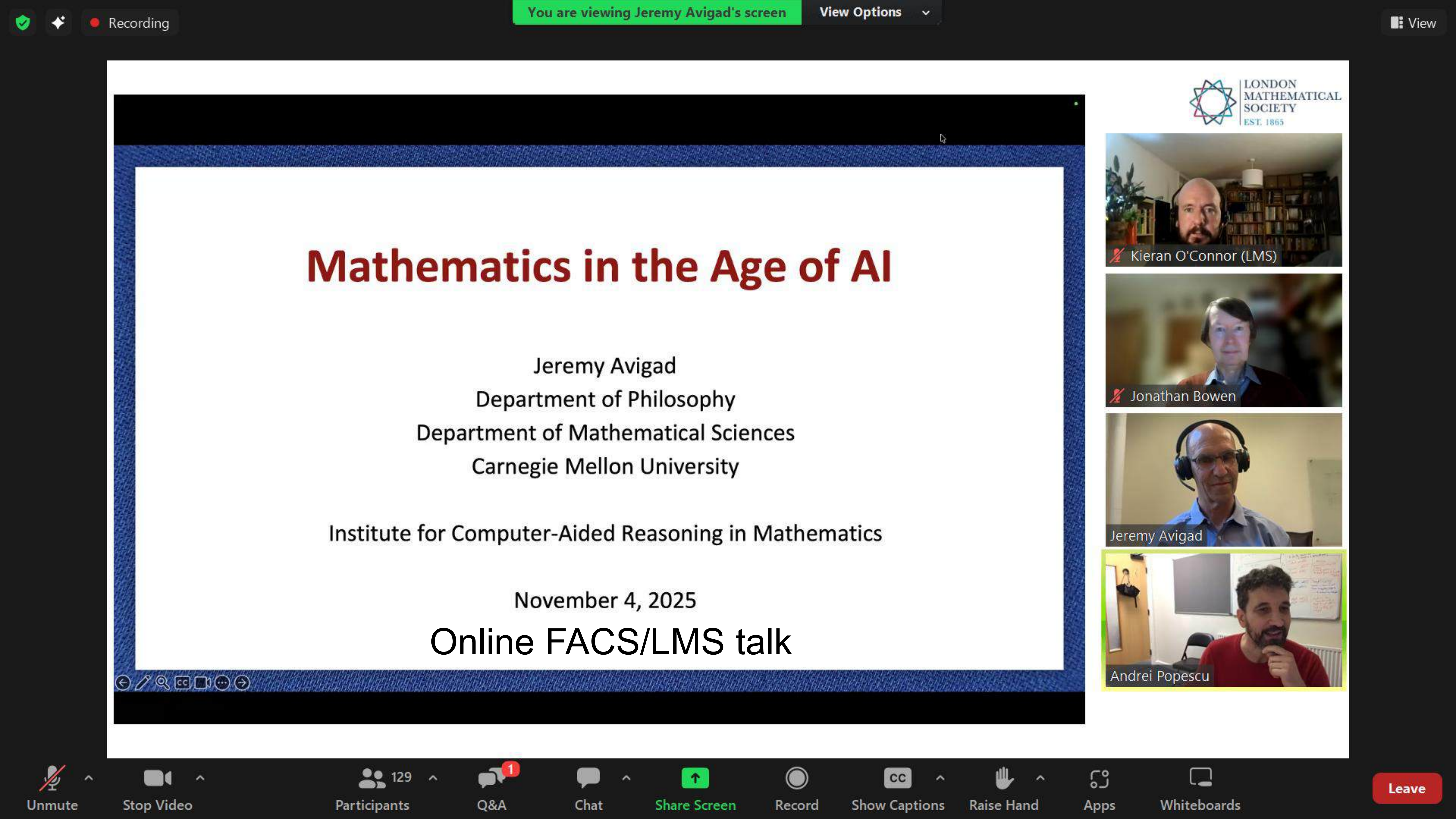
- Four color theorem: proved in 1976; formalized in 2005
- Kepler conjecture: resolved in 1998; formalized in 2014
- Liquid Tensor Experiment: Proposed in 2020; formalized in 2022;
- Polynomial Freiman Ruzsa conjecture: proven Nov 2023; formalized in 3 weeks

Use AI to fill in maths proof steps…

- So far, the speedup in formalization has been due to more modern proof assistant languages, richer mathematical libraries, and tools (such as Github) to facilitate collaboration.
- There are promising experiments in using AI to automatically fill in short steps in a formal proof (with the AI asked to try again if the proof does not compile).

…for program proofs too?

- In principle, AI integration will allow formal proofs to be written faster than human proofs (which are prone to error).
- This will be a tipping point, and will lead to formalization used not only to verify existing proofs, but to create new mathematics, using massive collaborations of both human and AI mathematicians. An era of "big mathematics"!


Recording
You are viewing Jeremy Avigad's screen
View Options
View
Mathematics in the Age of AI
Jeremy Avigad
Department of Philosophy
Department of Mathematical Sciences
Carnegie Mellon University
Institute for Computer-Aided Reasoning in Mathematics
November 4, 2025
Online FACS/LMS talk
LONDON MATHEMATICAL SOCIETY
EST. 1865
Kieran O'Connor (LMS)
Jonathan Bowen
Jeremy Avigad
Andrei Popescu
Unmute
Stop Video
129
Participants
Q&A
Chat
Share Screen
Record
Show Captions
Raise Hand
Apps
Whiteboards
Leave

Recording

LONDON MATHEMATICAL SOCIETY EST. 1865

Proof assistant & functional programming language

Lean Community

Community

- Zulip chat
- GitHub
- Blog
- Community information
- Community guidelines
- Teams
- Papers about Lean
- Projects using Lean
- Teaching using Lean
- Events

Use Lean

- Online version (no installation)
- Install Lean
- More options

Documentation

- Learning resources (start here)
- API documentation
- Declaration search (Loogle)
- Language reference
- Tactic list
- Calc mode
- Conv mode
- Simplifier
- Well-founded recursion
- Speeding up Lean files
- Pitfalls and common mistakes
- About MWEs
- Glossary

LEAN Community

Lean and its Mathematical Library

The Lean theorem prover is a proof assistant developed principally by Leonardo de Moura.

The community recently switched from using Lean 3 to using Lean 4. This website is still being updated, and some pages have outdated information about Lean 3 (these pages are marked with a prominent banner). The old Lean 3 community website has been archived.

The Lean mathematical library, *mathlib*, is a community-driven effort to build a unified library of mathematics formalized in the Lean proof assistant. The library also contains definitions useful for programming. This project is very active, with many regular contributors and daily activity.

You can get a bird's-eye view of what is in the mathlib library by reading the library overview, and read about recent additions on our blog. The design and community organization of mathlib are described in the 2020 article The Lean mathematical library, although the library has grown by an order of magnitude since that article appeared. You can also have a look at our repository statistics to see how the library grows and who contributes to it.

Try it!

You can try Lean in your web browser, install it in an isolated folder, or go for the full install. Lean is free, open source

Learn to Lean!

You can learn by playing a game, following tutorials, or reading books.

Meet the community!

Lean has very diverse and active community. It gathers mostly on a Zulip chat and on GitHub. You

lean-lang.org

Recording

LONDON MATHEMATICAL SOCIETY EST. 1865

The IMO Grand Challenge today

After the 2025 IMO, four groups claimed gold medal performance:

- Harmonic AI (formal)
- ByteDance (formal)
- OpenAI (informal)
- Google DeepMind (informal)

International Mathematical Olympiad

ByteDance's SeedProver solves 78.1% of formalized past IMO problems, and more than 50% on PutnamBench.

On September 26, a group at Apple and UC San Diego claimed 70% on PutnamBench with its publicly available Hilbert prover.

Final thoughts

"Today we serve technology. We need to reverse the machine-centered point of view and turn it into a person-centered point of view: Technology should serve us."

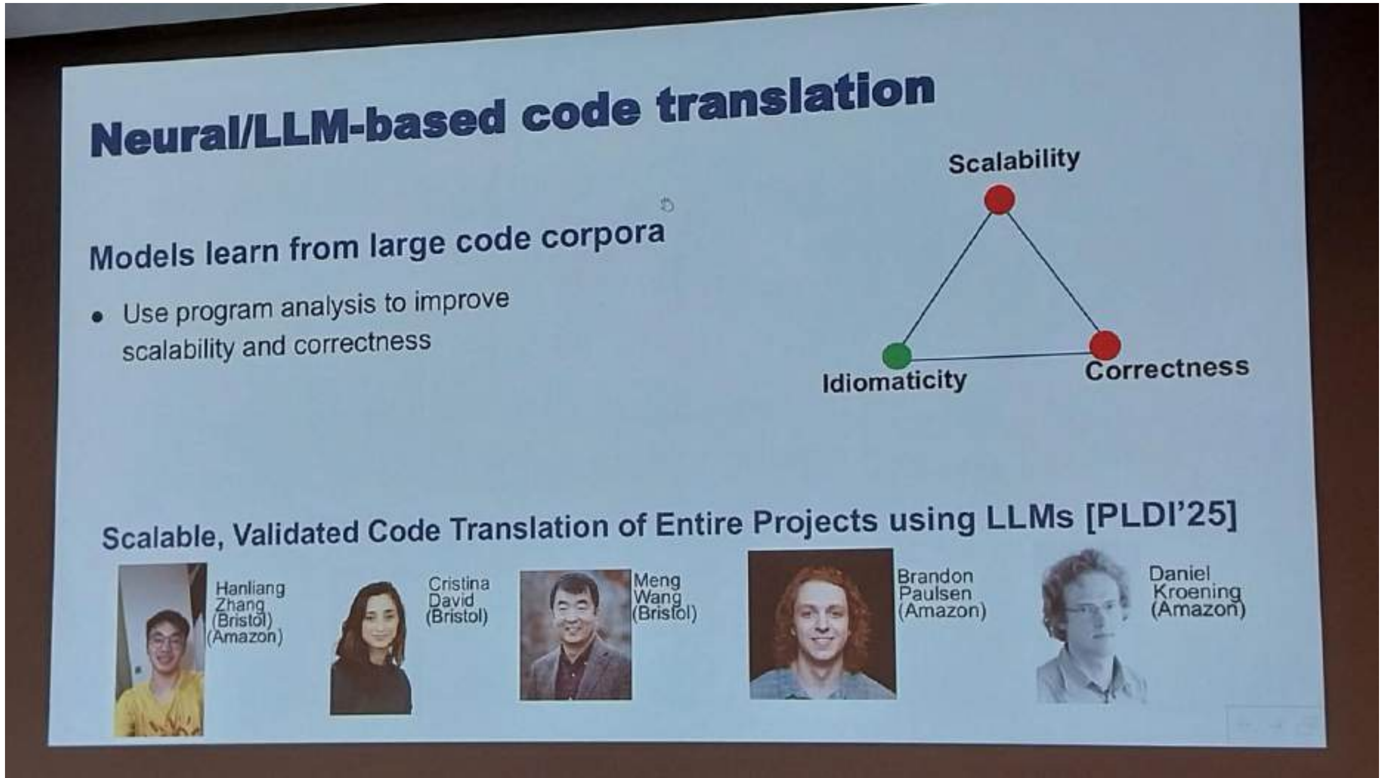
From *Things That Make Us Smart: Defending Human Attributes in the Age of the Machine*, by Donald A. Norman (1994)

The question is not "how can mathematicians use the technology?" but rather "what can technology do for mathematicians?"

… and formal methodists!

SETTA 2025

- 11th International Symposium on Dependable Software Engineering Theories, Tools and Applications
- St Catherine's College, Oxford, 1–3 December 2025
- Cristina David, Bristol University
- Automated translation of real-world codebases



Theorem Proving and AI in 2025

- Huang, S., et al. (Feb. 2025). *LeanProgress: Guiding Search for Neural Theorem Proving via Proof Progress Prediction*. arXiv. doi:10.48550/arXiv.2502.17925
- Lu, J., et al. (Oct. 2025). *Lean Finder: Semantic Search for Mathlib That Understands User Intents*. arXiv. doi:10.48550/arXiv.2510.15940
- DeepSeek releases DeepSeek-Math-V2 (Nov. 2025)
- Rapid AI-related developments with monthly updates…
- … perhaps a reinvigouration of formal methods!

Cf. AI winter

- Period of reduced funding between hype cycles
- Two major "winters" approximately 1974–1980 and 1987–2000
- Fifth Generation Computer Systems (FGCS): 10-year initiative launched in 1982 by Japan's Ministry of International Trade and Industry (MITI)
- Now a period of AI boom again with GenAI
- Perhaps something similar for formal methods!

“Vibe coding”* …
AI program generation

Vibe specifying → Formal specification

Formal specification → Vibe calculating → Program code

Program code → Vibe proving → Formal specification

Vibe coding → Program code

* *Collins Dictionary* Word of the Year for 2025

Reflection

Oui, l'ouvre sort plus belle
D'une forme au travail
Rebelle,
Vers, marbre, onyx, émail.

[Yes, the work comes out more beautiful from a material that resists the *process*, verse, marble, onyx, or enamel.]

— Théophile Gautier (1811–1872) *L'Art*

Reflection: Jean-Raymond Abrial (1938–2025)

- Originator of three important formal methods: Z notation, B-Method, and Event-B
- *FACS FACTS* newsletter tributes planned for January 2026
- Cf. *Tony Hoare @ 90* tributes in *FACS FACTS* July 2024 issue, pp. 5–42
- LNCS Festschrift volume also planned

Formal Methods: Whence and Whither?

Prof. Jonathan P. Bowen FRSA FBCS

Emeritus Professor of Computing
London South Bank University, UK

Adjunct Professor, Southwest University, Chongqing, China

Chairman, Museophile Limited, Oxford, UK

www.jpbowen.com

The End