

50 Years in Formal World: Mainstream Computing's Mirror Universe

Fall 2023

Introduction

Hello!

- Excited!
 - Invited by professor-friends
- Random talk
 - Half history, half tech
- Not on your exam
 - Fun? Maybe interesting?



Who am I?

- Industrial programmer
 - Non academic
- Rust initiator
 - Long ago
- Unrelated to Tony Hoare
 - Photos!



Turing Award Winner
Tony Hoare
Accepting His
Turing Award
1980



Me and my dad,
who is not
Tony Hoare
1982

**Let's begin with
an interesting quote**

Edsger Dijkstra



Finally, in order to drive home the message that this introductory programming course is primarily a course in formal mathematics, we see to it that the programming language in question has not been implemented on campus so that students are protected from the temptation to test their programs.

Introduction

???

???? protected from
testing programs ????

Mirror universe

- Makes sense
 - In "formal world"
- Today's topic!



Formal World runs parallel to mainstream

- Branched from mainstream
 - 50 years ago
- Adjacent
 - Interacts!
- Weird
 - Challenging
 - Science fiction



Formal world's fundamental premise

- 100% correctness
 - Zero bugs
 - Don't run, don't test: prove
 - Full state-space
- Via formal logic

Major formal world sites

- France: INRIA, ENS, Paris 7, I'X
- UK: Edinburgh, Oxford, Cambridge, Manchester
- Sweden: Chalmers, Stockholm, KTH
- Netherlands: CWI, TU Eindhoven
- Denmark: DTU, Aalborg
- Germany: MPI Saarland, TU Munich
- Switzerland: ETH Zurich, EPFL
- Israel: Technion, Tel Aviv
- USA: Cornell, CMU, Stanford/SRI, Austin, Iowa, Berkeley, MIT; Intel, IBM, MSR



This talk

- History of split
 - Context, development
- Situation today
 - Tools and techniques
- Informed Choice
 - Cornell
 - Faculty
 - Your future



<https://flickr.com/photos/hazael/2783985259> - CC BY-NC 2.0

Caveats

- Outsider
- Biased and wrong
- Many slides
 - No time for Q&A
 - Some fairly dense
 - Download later
 - Not on exam



Part 1

Origins

Part 1.1

The world of computing
just before the split

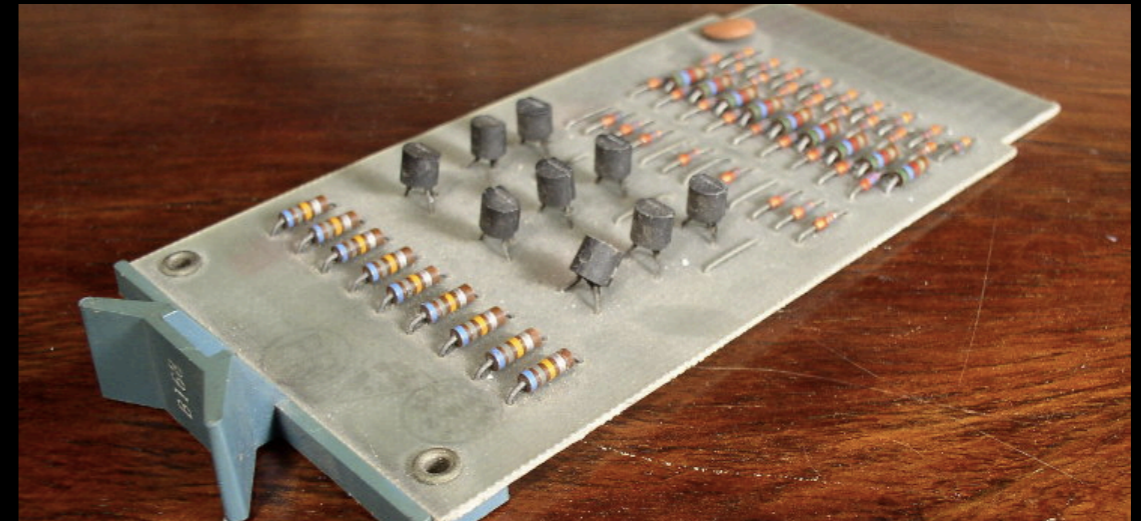
50 years ago: 1973

- Computing ~25 years old
- US and Europe
- Well established
 - Recognizable!
- Not so long ago
 - "Less than one career"
 - Robert Constable professor
 - Dexter Kozen undergrad

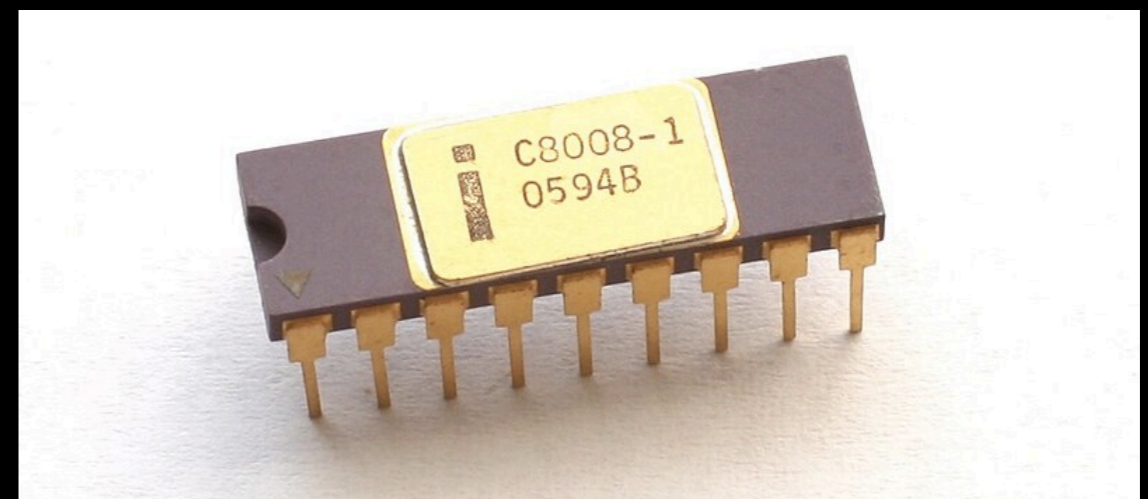


1973 Hardware

- Discrete transistors!
- New: integrated circuits
 - 8008, ancestor of x86-64
 - Too puny
 - Moore's law year one



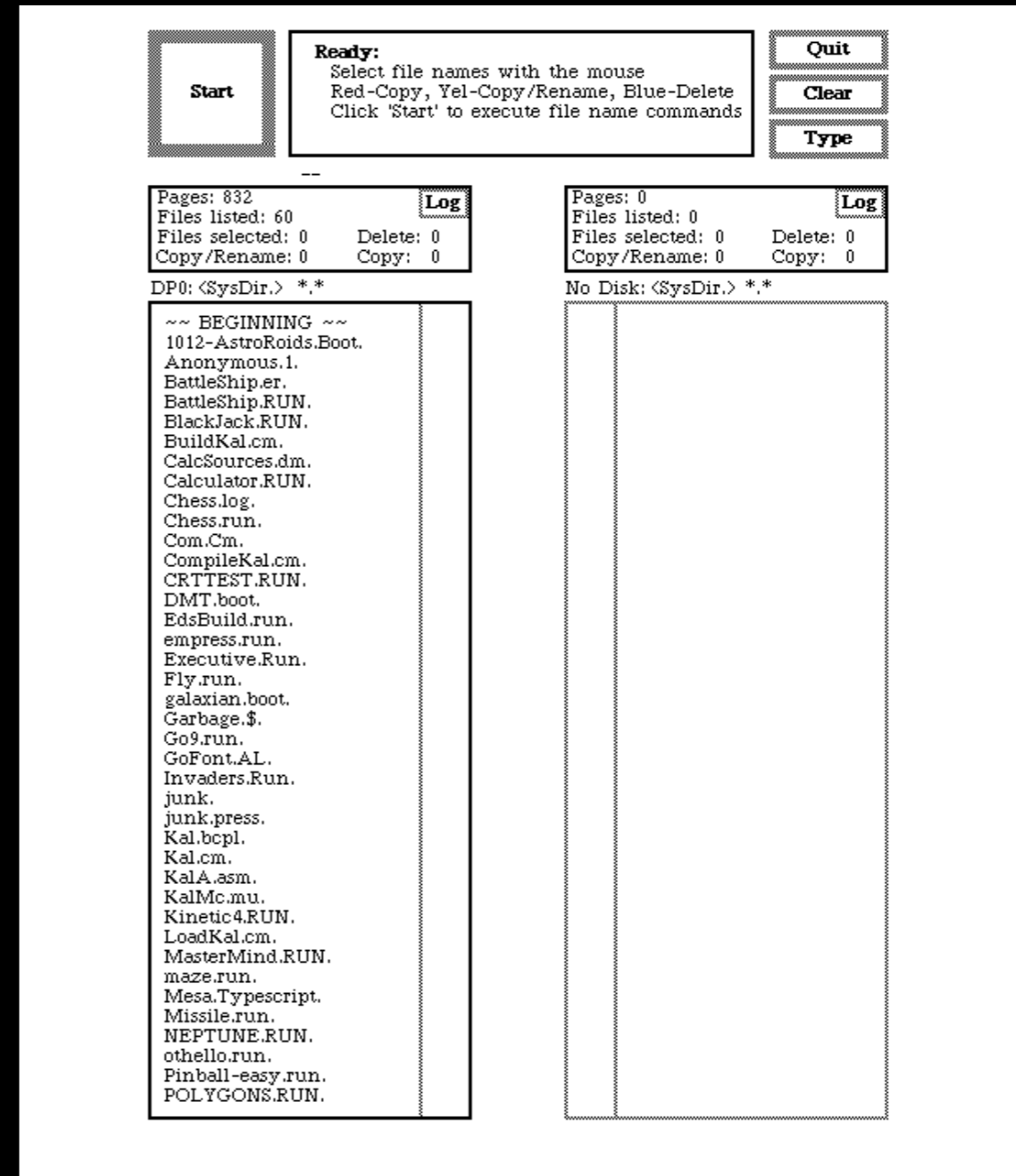
https://commons.wikimedia.org/wiki/File:KA10_mod_end.jpg - CC BY-SA 2.5



https://commons.wikimedia.org/wiki/File:KL_Intel_C8008-1.jpg - CC BY-SA 4.0

1973 Software

- Unix, grep
- Version control systems
- Relational databases
- Hypertext, early GUIs
- Video games



1973 Languages

- C
- Smalltalk
- ML
- CLU
- Prolog
- *20 years* of other HLLs:
FORTRAN, COBOL, ALGOL,
LISP, Simula, PL/I, etc.

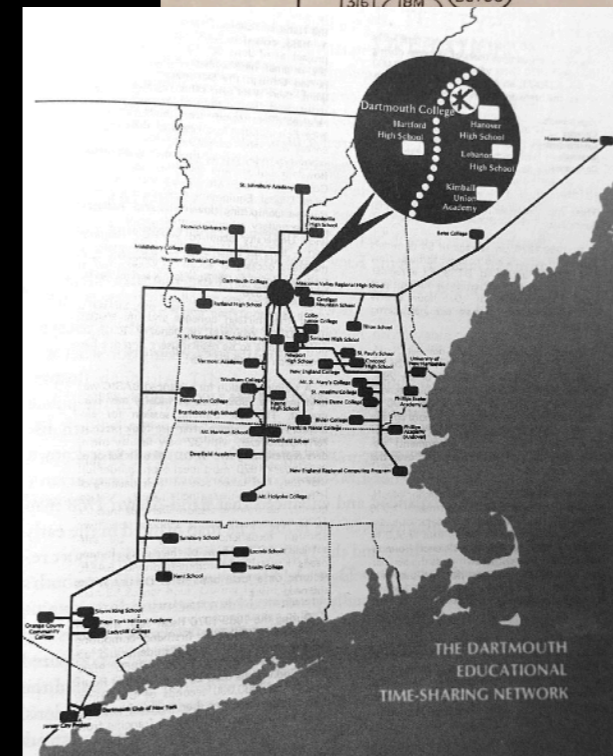
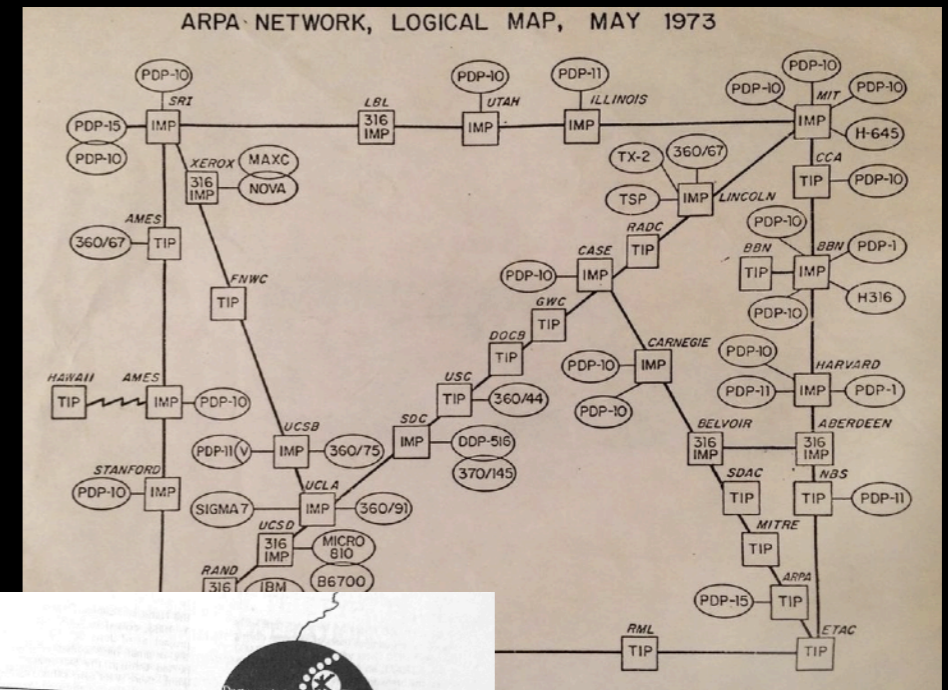


THE
C
PROGRAMMING
LANGUAGE

Brian W. Kernighan • Dennis M. Ritchie

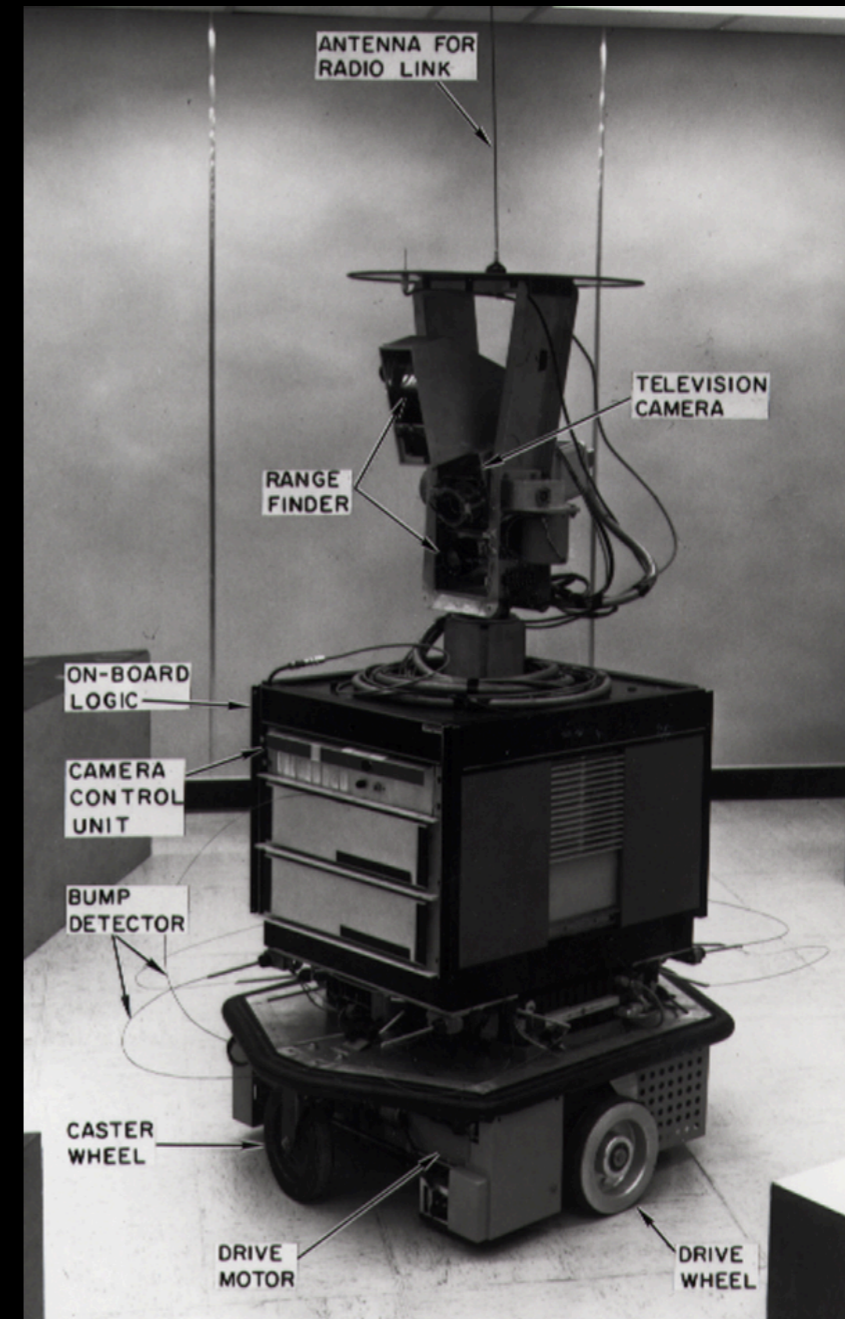
1973 Networking

- Proto-internets: ARPAnet, CYCLADES
- Ethernet
- "Timesharing" (DTSS, PLATO)
- Online culture, games, chat



1973 AI

- 17 years old!
- Big university labs
- Symbolic, deductive logic
- Neural networks sidelined
- Another story



1973 Government

- "Military-industrial complex"
 - Space race
 - Missile defence
 - Vietnam counter-insurgency
 - AI, "human augmentation"
- Declining spending
 - Stirring neoliberalism



1973 Business

- Century-old regulated monopolies:

- AT&T: "the phone company"

- 82% of all phones, \$22bn revenue, >1m employees



- IBM: "the computer company"

- 70% of all computers, \$8bn revenue, >250k employees



1973 Mainstream computing: Institutional

- Computers require institutions
- Programming quite serious
- 2000 lb machine, \$100,000
 - Add 8K of RAM for \$50,000!

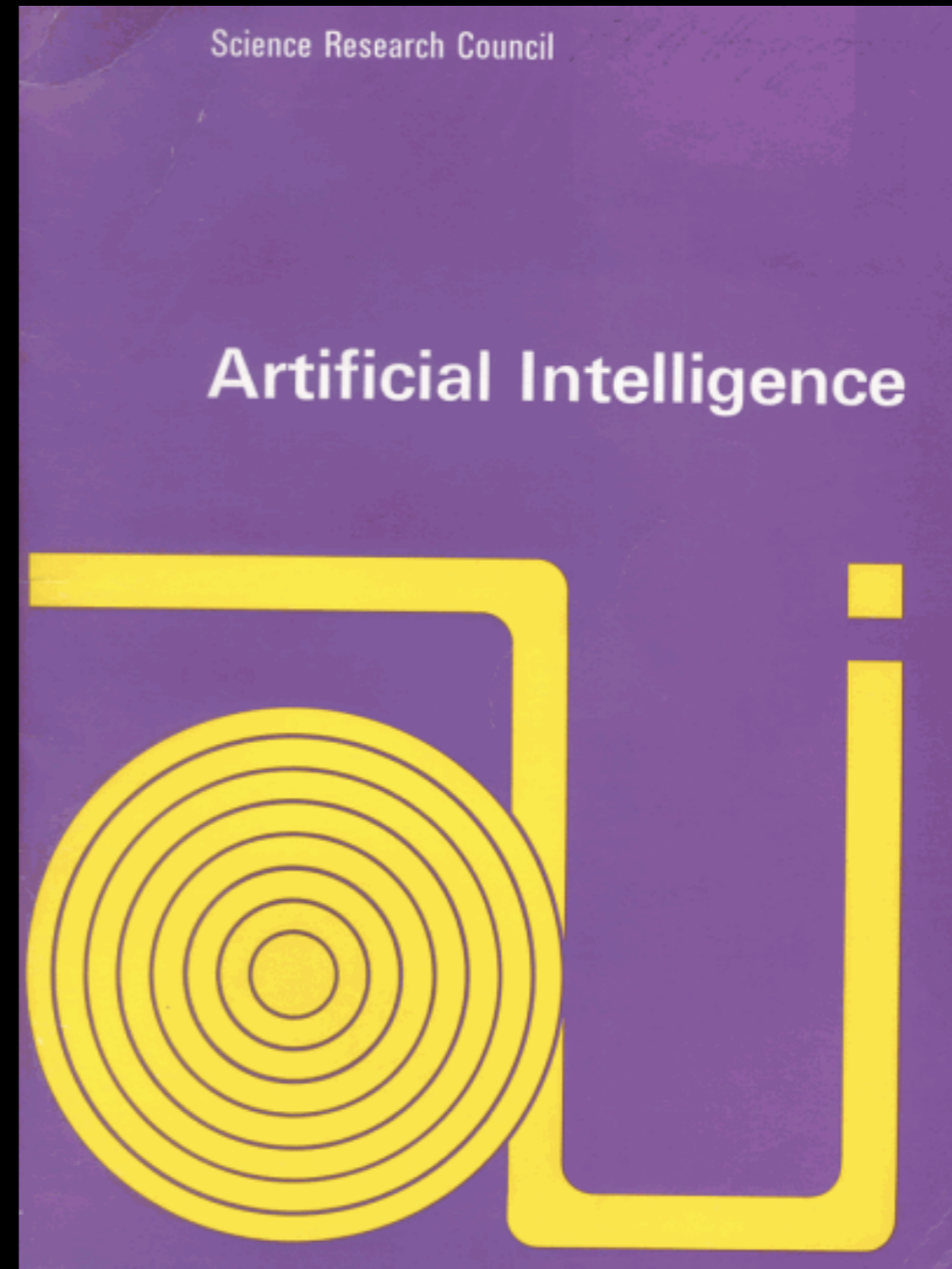


https://commons.wikimedia.org/wiki/File:PDP-10_1090.jpg CC-BY-SA 3.0

Part 1.2
Causes and emergence
of the split

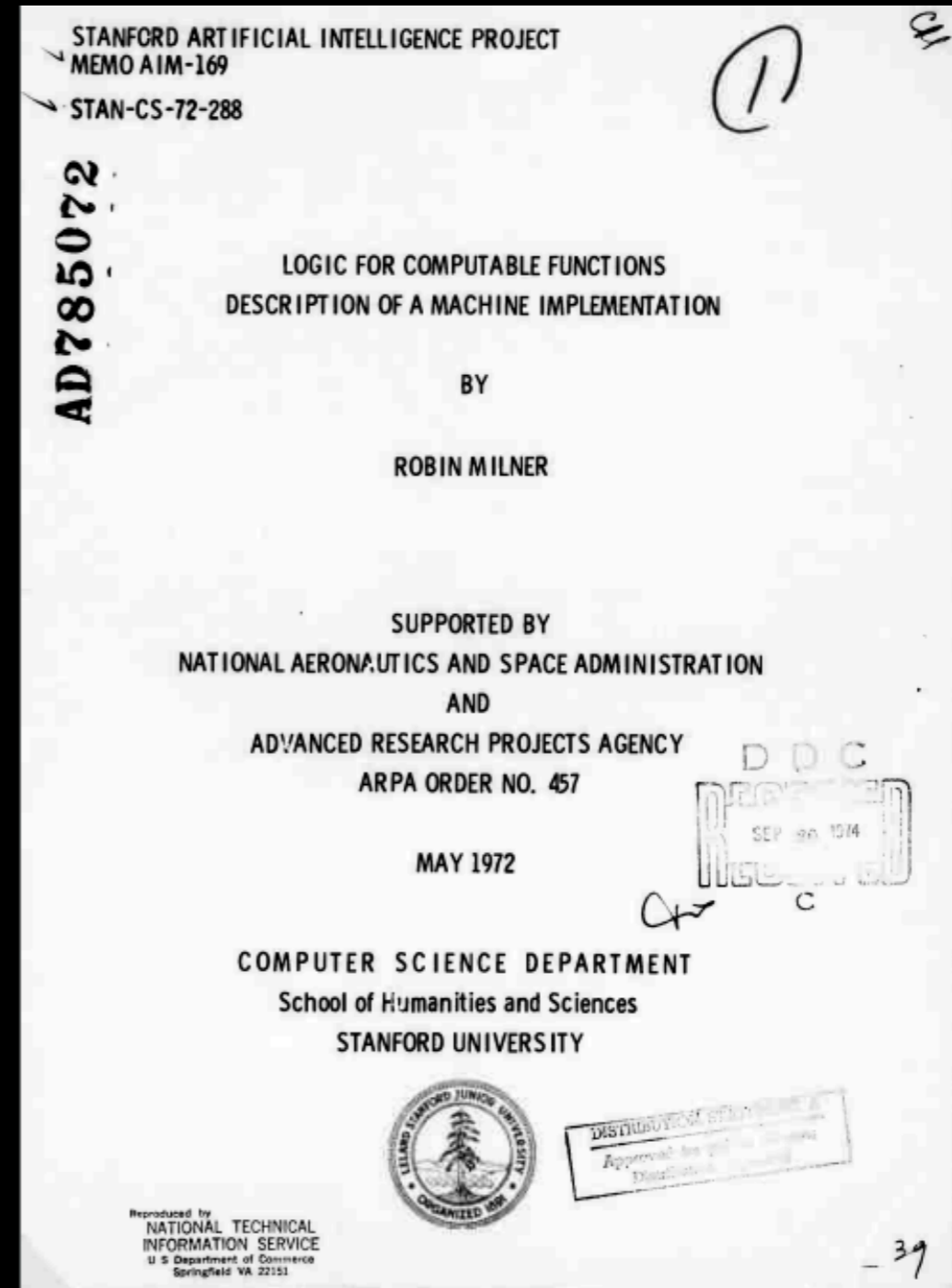
Logic-for-AI project running into trouble

- Disappointments
 - Many failed projects
 - NP-complete class found
- 1973-4: funding cut
 - "AI winter"
 - Idle logic talent



Logic of *Programming* making better headway

- Strachey, Scott, Kahn, Berry:
 - Logical semantics of PLs
- Boyer, Moore, Plotkin, Milner:
 - *mechanized* FP logics
 - ML born here, in LCF system
- 1973 Dijkstra predicate-transformer semantics:
 - *mechanized* imperative logic



Logic *as* Types blossoming

- 1967 de Bruijn & 1969 Howard extend Curry's link: natural deduction is lambda calculus

theorem \leftrightarrow type

proof \leftrightarrow program

- 1972 Girard & 1974 Reynolds: 2nd order logic is typed polymorphic lambda calculus ("System F")
- 1973 Martin-Löf dependent type theory, math foundations

AN INTUITIONISTIC THEORY OF TYPES:
PREDICATIVE PART

Per MARTIN-LÖF

University of Stockholm, Stockholm, Sweden

The theory of types with which we shall be concerned is intended to be a full scale system for formalizing intuitionistic mathematics as developed, for example, in the book by Bishop[1]. The language of the theory is richer than the languages of traditional intuitionistic systems in permitting proofs to appear as parts of propositions so that the propositions of the theory can express properties of proofs (and not only individuals, like in first order predicate logic). This makes it possible to strengthen the axioms for existence, disjunction, absurdity and identity. In the case of existence, this possibility seems first to have been indicated by Howard[10], whose proposed axioms are special cases of the existential elimination rule of the present theory. Furthermore, there are axioms for universes (in the sense of category theory) which link the generation of objects and types and play somewhat the same role for the present theory as does the replacement axiom for Zermelo–Fraenkel set theory. They also make the theory adequate for the formalization of certain constructions in category theory, like the construction of the category of all small categories.

An earlier, not yet conclusive, attempt at formulating a theory of this kind was made by Scott[19]. Also related, although less closely, are the type and logic free theories of constructions of Kreisel[12, 13] and Goodman[8].

In its first version, the present theory was based on the strongly impredicative axiom that there is a type of all types whatsoever, in symbols, $V \in V$, which is at the same time a type and an object of that type. This axiom had to be abandoned, however, after it had been shown

Frustration building software, talk of "Software Engineering"

- 1968 NATO "Software Crisis" conference
- Software lousy
 - Late
 - Expensive
 - Buggy
 - Hard to maintain
- Need "Software Engineering"

SOFTWARE ENGINEERING

Report on a conference sponsored by the
NATO SCIENCE COMMITTEE
Garmisch, Germany, 7th to 11th October 1968

Chairman: Professor Dr. F. L. Bauer
Co-chairmen: Professor L. Bolliet, Dr. H. J. Helms

Editors: Peter Naur and Brian Randell

Algol 68 Rift

- IFIP 2.1 Algol committee
- Dissenting Minority Report:
new languages aren't enough to solve the software crisis
- Break in community: IFIP 2.3
- "Programming methodology",
proofs and correctness
 - later "formal methods"

Now the language itself, which should be judged, among other things, as a language, in which to compose programs. Considered as such, a programming language implies a conception of the programmer's task. We recognise that over the last decade the processing power of commonly available machines has grown tremendously and that society has increased its ambition in their application in proportion to this growth. As a result the programmer of today and tomorrow, finding his task in the field of tension between available equipment and desired applications, finds himself faced with tasks of completely different and still growing scope and scale. More than ever it will be required from an adequate programming tool that it assists, by structure, the programmer in the most difficult aspects of his job, viz. in the reliable creation of sophisticated programs. In this respect we fail to see how the language proposed here is a significant step forward: on the contrary, we feel that its implicit view of the programmer's task is very much the same as, say, ten years ago. This forces upon us the conclusion that, regarded as a programming tool, the language must be regarded as obsolete.

The present minority report has been written by us because if we had not done so, we would have forsaken our professional responsibility towards the computing community. We therefore propose that this Report on the Algorithmic Language ALGOL 68 should not be published under IFIP sponsorship. If it is so published, we recommend that this "minority report" be included in the publication.

Signed by: DIJKSTRA, DUNCAN, GARWICK, HOARE, RANDELL,
SEEGMUELLER, TURSKI, WOODGER.

Formation of IFIP 2.3

- Leading lights of 1970s academic CS
- Several Cornell folks!
- Invite-only, no goal, sharing ideas
- Kept meeting for 50 years!

- | | | | | |
|--------------------------------|--------------------------------|----------------------------------|-------------------------------|---|
| • Jean-Raymond Abrial Emeritus | • Edsger W. Dijkstra (d. 2002) | • Shriram Krishnamurthi Emeritus | • Susan Owicki Emeritus | • Natarajan Shankar |
| • Ralph-Johan Back | • Sophia Drossopoulou | • Butler W. Lampson Emeritus | • David Lorge Parnas Emeritus | • Michel Sintzoff (d. 2010) |
| • Dines Bjørner Emeritus | • David Gries Emeritus | • Gary T. Leavens | • Benjamin C. Pierce Emeritus | • Jan L. A. van de Snepscheut (d. 1994) |
| • Per Brinch Hansen (d. 2007) | • John Guttag Emeritus | • Doug McIlroy Emeritus | • George Radin (d. 2013) | • Christopher Strachey (d. 1975) |
| • Manfred Broy | • Eric C. R. Hehner Emeritus | • George H. Mealy (d. 2010) | • Brian Randell Emeritus | • Warren Teitelman Emeritus |
| • Rod Burstall Emeritus | • Tony Hoare | • Bertrand Meyer | • John C. Reynolds (d. 2013) | • Emina Torlak |
| • Michael Butler | • Jim Horning (d. 2013) | • Jayadev Misra | • Douglas T. Ross (d. 2007) | • Jim Woodcock |
| • William R. Cook (d. 2022) | • Daniel Jackson | • Carroll Morgan | • Fred B. Schneider Emeritus | • Niklaus Wirth Emeritus |
| • Patrick Cousot | • Michael Jackson | • Peter Naur (d. 2016) | | • Mike Woodger Emeritus |
| • Ole-Johan Dahl (d. 2002) | • Cliff Jones | • Greg Nelson (d. 2015) | | • Pamela Zave |

Part 2

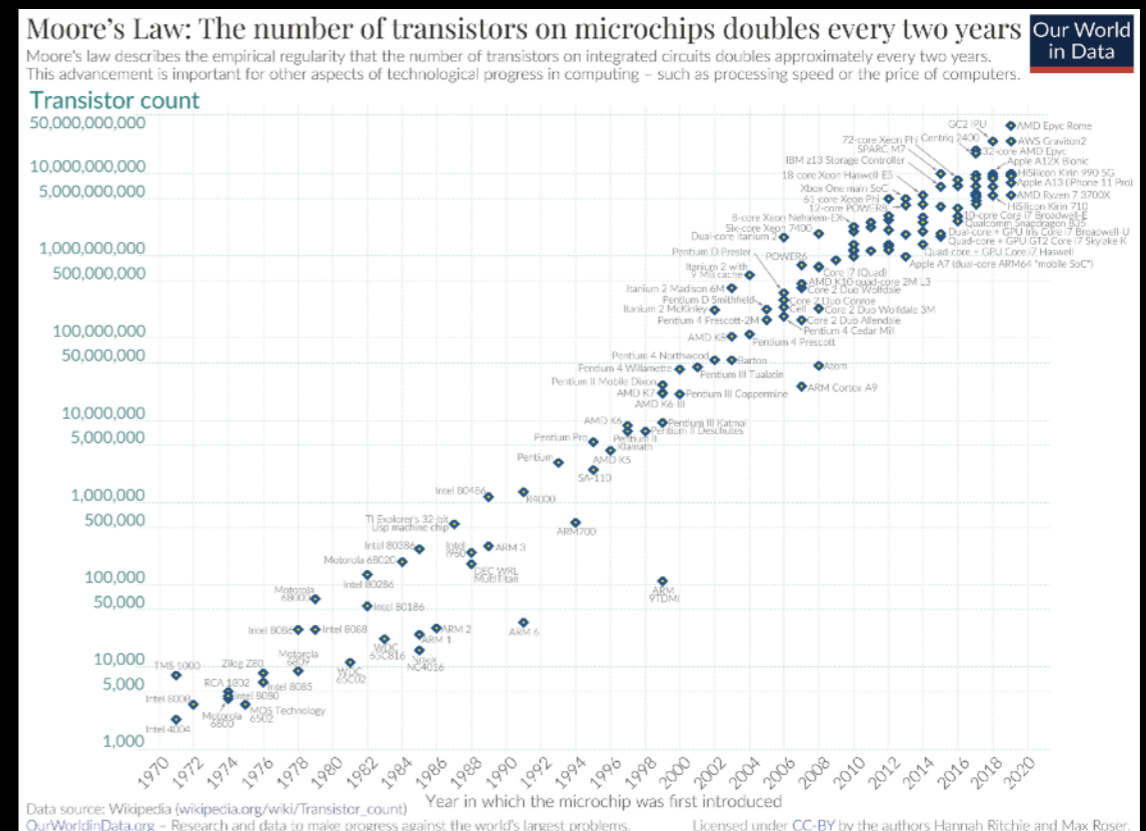
Divergence

Part 2.1

**50 years of changes
in the world of computing**

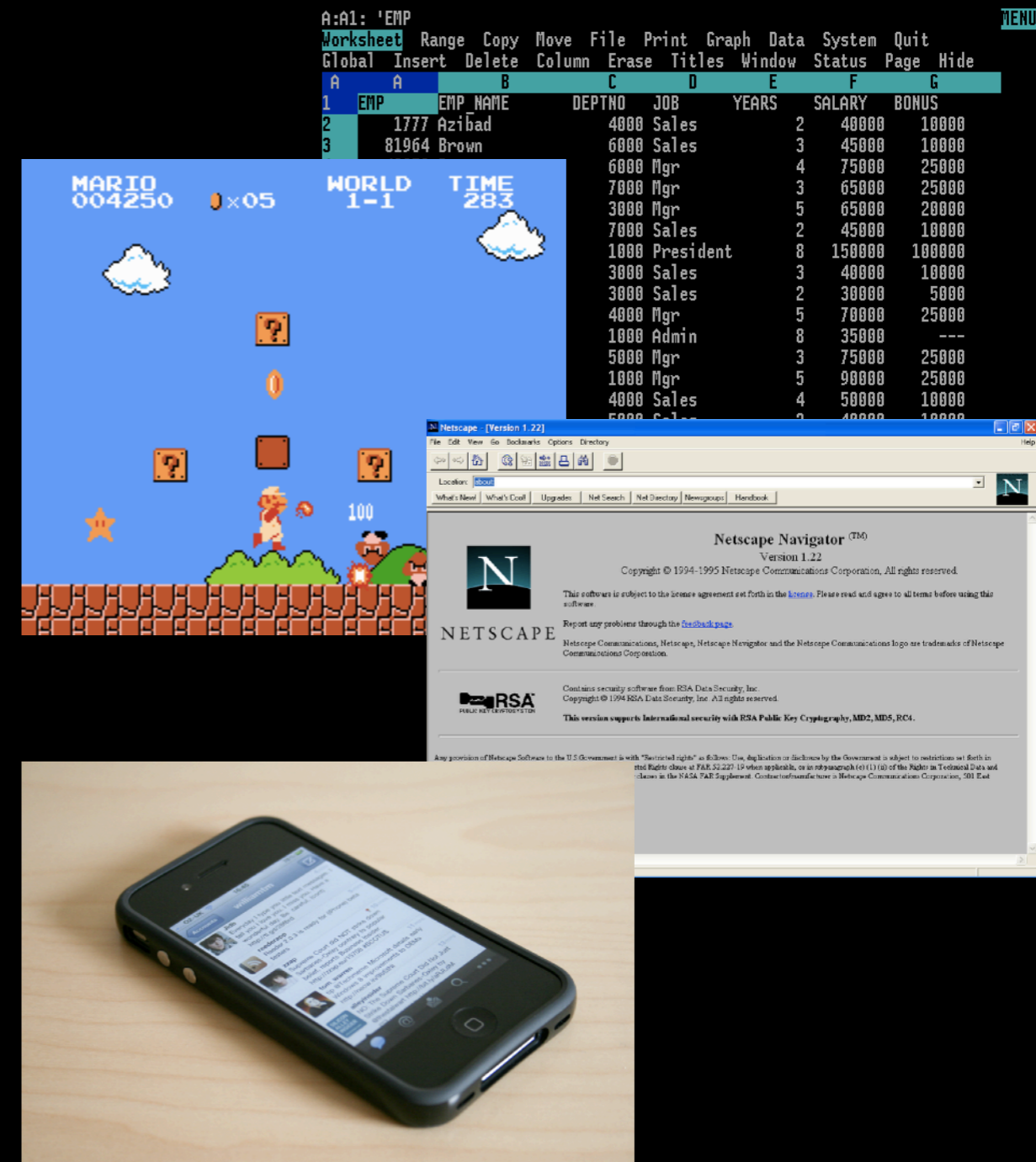
Hardware Changes

- Moore's law
 - 1973's 8008: 3k transistors
 - 2023's M2: 20b transistors
 - By hand up to ~200k!
- Cheap, plentiful computers



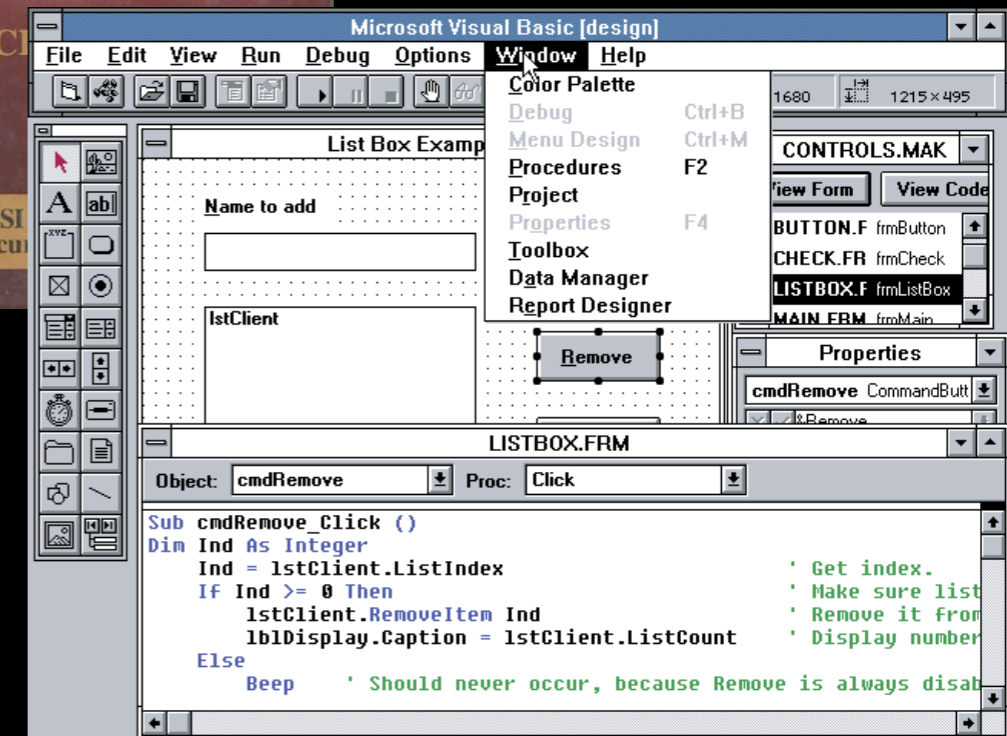
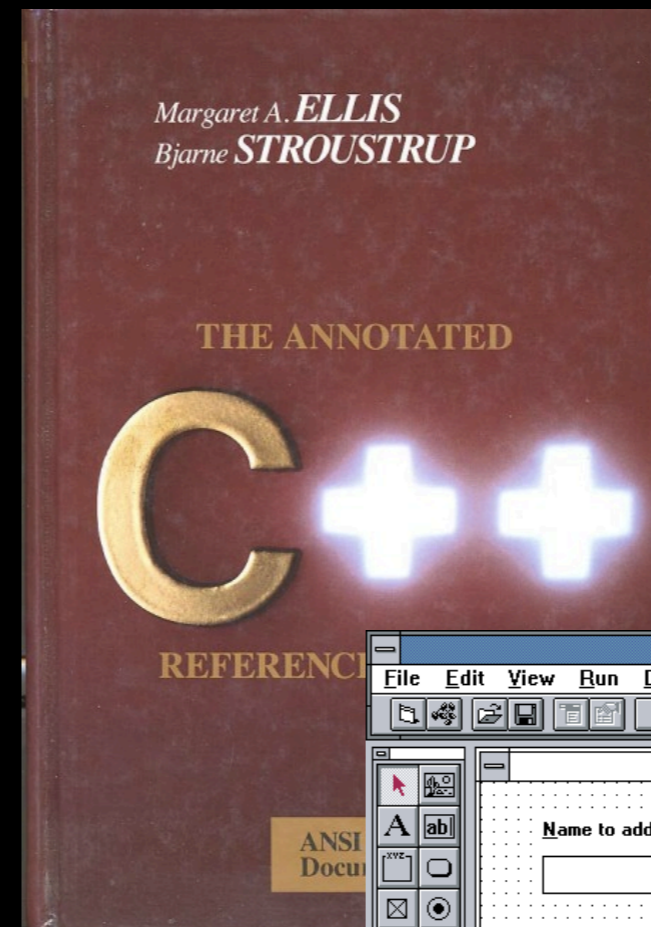
Software Changes

- Mass market of software
- For every occasion
 - Work
 - Leisure
 - Learning
 - Communicating
- Massive oversupply



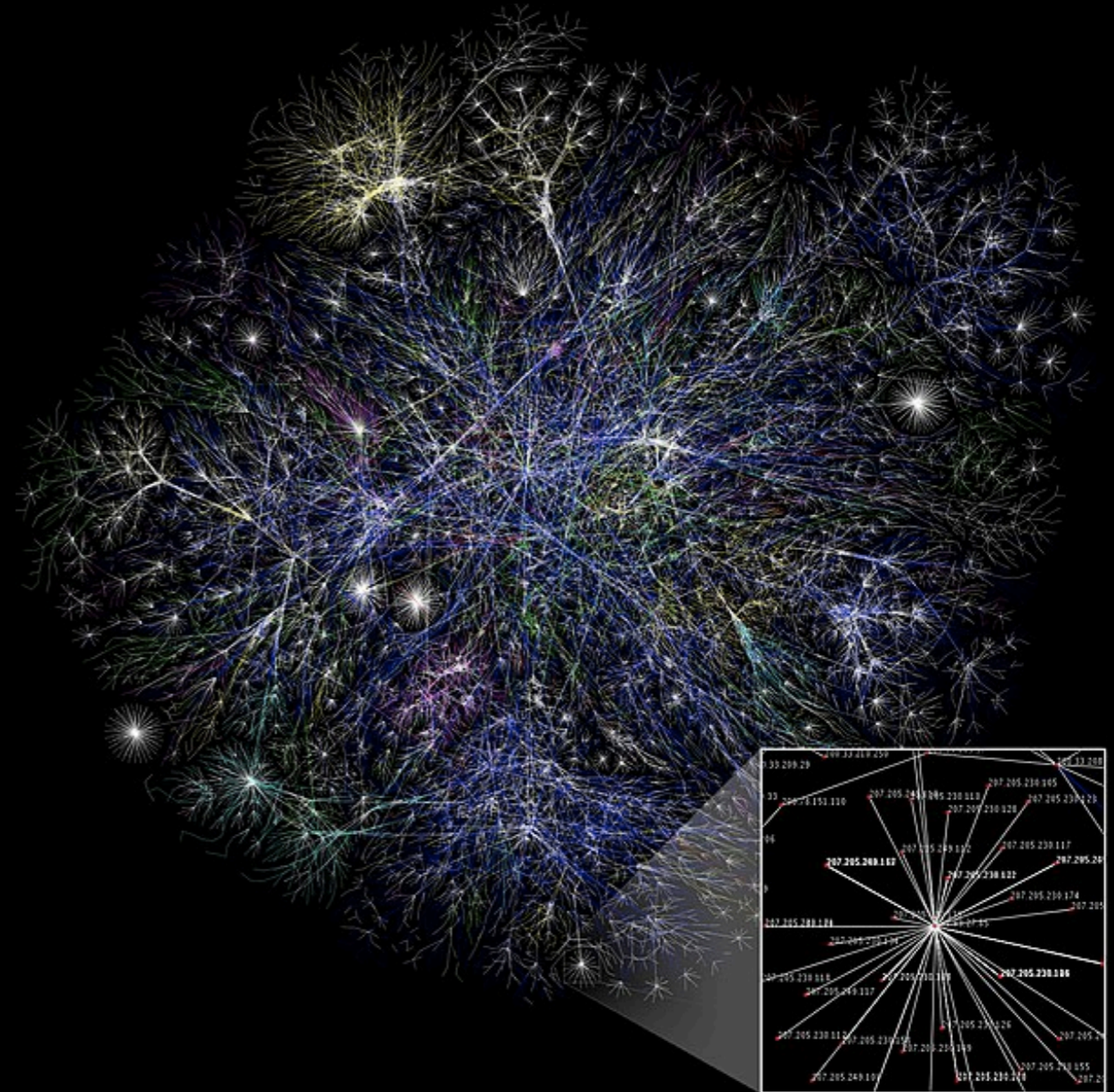
Language Changes

- Ousterhout dichotomy:
"systems" vs. "scripting"
- Systems: C/C++
 - Speed above all
- Scripting: VB, Python, JS
 - Ease of use above all
- Neither prioritize correctness



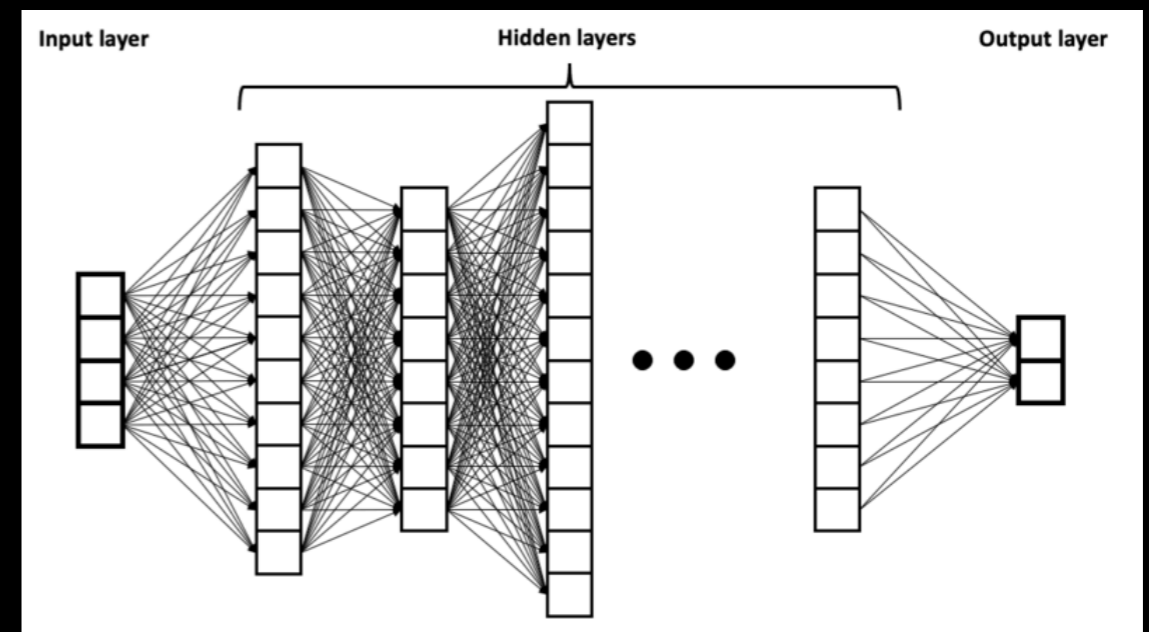
Networking Changes

- BBSs, private systems (CompuServe, AOL)
- CCITT/ITU (telcos) "OSI" data network: not internet!
- Internet wins
- Nice briefly, then nasty
- 24/7 attacks, organized crime
- "Cyber" now a *military* term



AI Changes

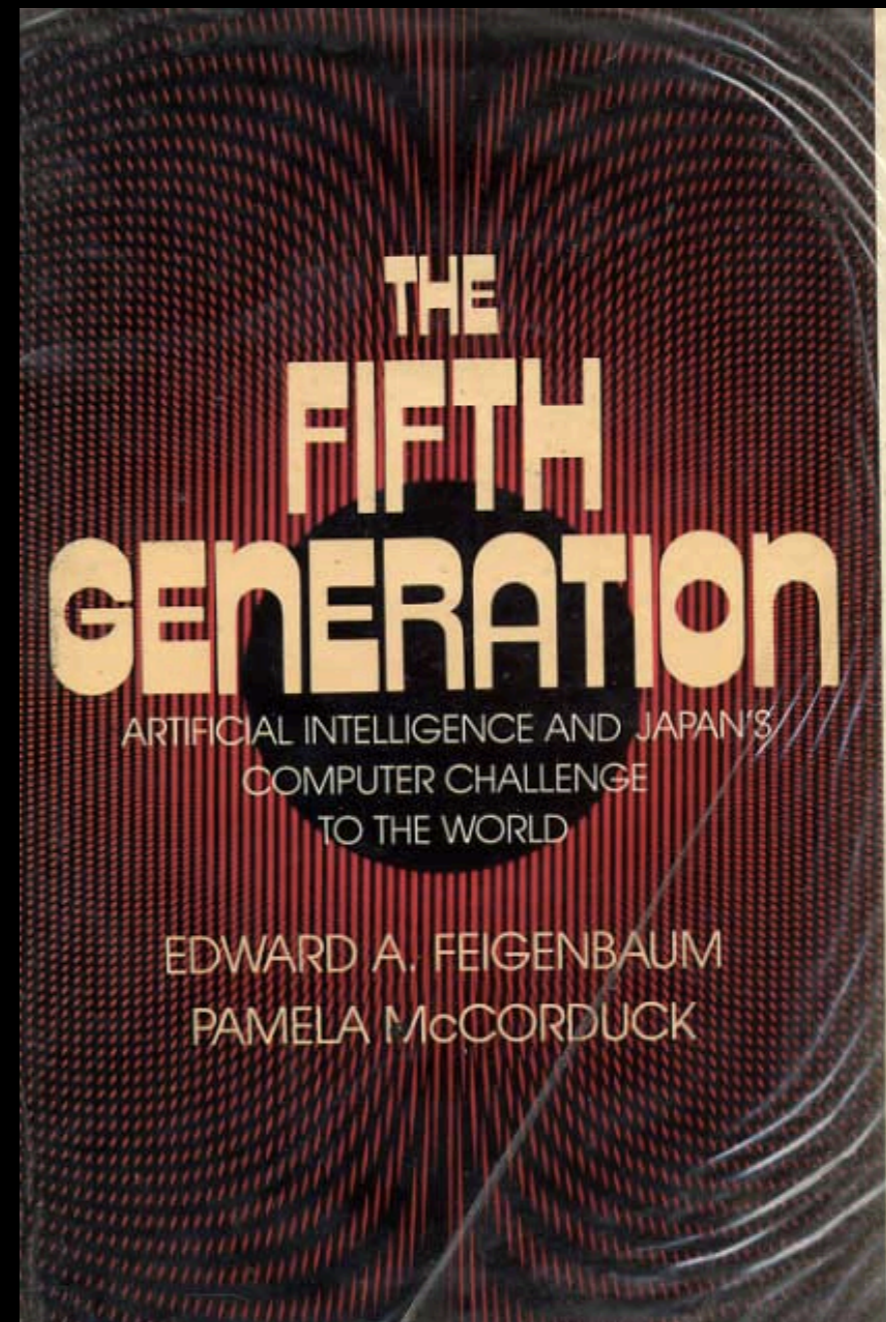
- Waves of hype, despair
 - Neural nets win
- Unclear relevance to formal
 - Maybe safety critical?
 - Maybe hard to verify?
 - Maybe helps verify?
- Won't discuss more, no time



https://commons.wikimedia.org/wiki/File:Example_of_a_deep_neural_network.png CC-BY-SA 4.0

Government Changes

- Neoliberalism: deregulation, deindustrialization, IP focus
- More Military-industrial complex: Cold War, Internet, War on Terror
- Western panic over Japan
 - MITI VLSI, "5th gen computer"
 - DARPA VLSI, VHDL projects
 - US hardware-design firms
 - Europe: Alvey, ESPRIT



Business Changes

- 70s antitrust
- AT&T: no computers, breakup
 - Ubiquitous UNIX
- IBM: unbundle software, clones
 - Ubiquitous DOS
- Broad, cheap computer market
- 80s antitrust: keep prices low!

MAJOR BRAND LIQUIDATION



Retail Value \$5000
Save up to 60%

ACP PRICE NOW ONLY!
\$1985⁰⁰

*Color 14" RGB Included
10 Mb Hard Disk*

ACP is proud to make this one-time special offer for a complete computer system that is 110% compatible to IBM™. This is by far the most significant bargain that we at ACP have offered in our 10 year history. This system was successfully designed and manufactured to exceed IBM™'s PC in terms of quality, expansion modularity and capability, aesthetic appearance, and performance. The system design utilizes the latest in state-of-the-art technology including:

- VLSI - Large Scale Integration Circuit Design
- Ergonomic CRT Design with Tilt Screen
- Professional Molded Packaging and Design
- High Quality 100 Watt Switching Supply
- Complete Integrated System
- Microsoft Compatible Mouse Function

The system is not a Taiwan or Korean knock-off. Each component is specifically designed and specified to meet the highest performance and reliability standards in the industry. It represents the best that Japanese craftsmen have to offer and you will be equally proud to own one of your own. ACP has a limited quantity of these systems in several different configurations. IBM™ PC-DOS™ v1.1/2.1, MS-DOS™ v2.11 and Concurrent v3.1 compatible. We have found no known incompatibility with any IBM™ PC application. Our technical staff has 8.5 Megabytes of various MS-DOS software packages installed including Lotus 1-2-3 and Flight Simulator. Each system comes complete with a 90 day warranty.

ACP Base System Consists of:

- (1) 360K DD/DS Floppy Disk Drive
- Mouse with Software
- 256K Memory Expandable to 640K on the Motherboard
- Deluxe Keyboard with LEDs
- Serial Port and Parallel Port
- Color or Monochrome Controller
- 4.77MHz, 8088 CPU
- 100 Watt Switching Supply w/Fan
- Three Expansion Slots
- Optional 6 Slot Expansion Chassis with Power Supply (add \$399)

	SYSTEM CONFIGURATION	Est IBM List*	Your Price
SYSTEM A	Base System (see left) PC with 360K Floppy, Keyboard & Mouse.	\$2100.00	\$995.00
SYSTEM B	Base System (see left) plus Add'l 360K Floppy Drive	\$2295.00	\$1099.00
SYSTEM C	Base System plus 12" Green Monitor with Detachable Tilt/Swivel Base.	\$2575.00	\$1399.00
SYSTEM D	Base System plus 12" Color Monitor with Detachable Tilt/Swivel Base.	\$2995.00	\$1699.00
SYSTEM E	Base System plus Ctr Monitor, 10Mb Hard Disk and Boot Diagnostics.	\$5000.00	\$1985.00
SYSTEM F	Base System plus 80 Col. x 25 Line LCD Screen	N/A	\$1299.00

Base System A (as above) **\$995.00**

*Assumes required add-in boards to provide same capacity. IBM PC is a trademark of IBM Corp.

Mainstream computing changes: becoming personal

- Personal computers
- Everyone gets one
- Software made by and for individuals
- Mass market much larger than "institutional era"



Unfortunately, personal
(mainstream) software
was not very formal

Part 2.2

50 years of rejection

Formal world made several wrong bets

- Wrong assumptions about software verification:
 - Plausible
 - Tractable
 - Tolerable
 - Necessary
- Wrong at least for mainstream



Verification: implausible

- Software exploratory
 - No spec
 - "Try stuff and iterate"
- Personal computers, casual, cheap, mass programming
 - Self-taught coders
 - No training or interest in formal logic, proof



Verification: intractable

- Hard: mutable aliased state
 - Mainstream PLs full of it!
 - Worse with threads!
- Hard: big programs
 - Mainstream programs huge
 - GUIs, feature creep



Verification: intolerable

- Two software markets
 - "Software as overhead": platforms, enterprise
 - "Software as commodity": applications, games
- Neither values quality: minimize cost
 - Verification = 10-100x cost!



Verification: unnecessary

- 1976 Copyright law: software is "work of literature"
 - No warranty, no liability
 - User learned helplessness
- Software engineering
 - High level PLs, types
 - Tests
 - Version control
 - Continuous field updates



Case study: Ada

- US military solution to crisis
 - Not even "formal methods", just "high reliability"
- Little uptake: expensive tools, complex, rigid, niche market
- Mandated 1991, repealed 1997
- "Off-the-shelf" C/C++ won

Reference Manual for the
ADA[®]
Programming Language

ANSI/MIL-STD-1815A-1983

United States Department of Defense

Approved February 17, 1983
American National Standards Institute, Inc.



Springer Science+Business Media, LLC

ADA[®] is a registered trademark of the U.S. Government, ADA Joint Program Office

Ok that sounds bad!
Is *anything* verified?

Yes!

Part 2.3

50 years of adoption

Formal world has some areas of successful adoption

- Hardware verification
- Safety-critical systems
- Telecoms & networking
- Operating systems & drivers
- Cryptographic algorithms
- Programming languages



How?

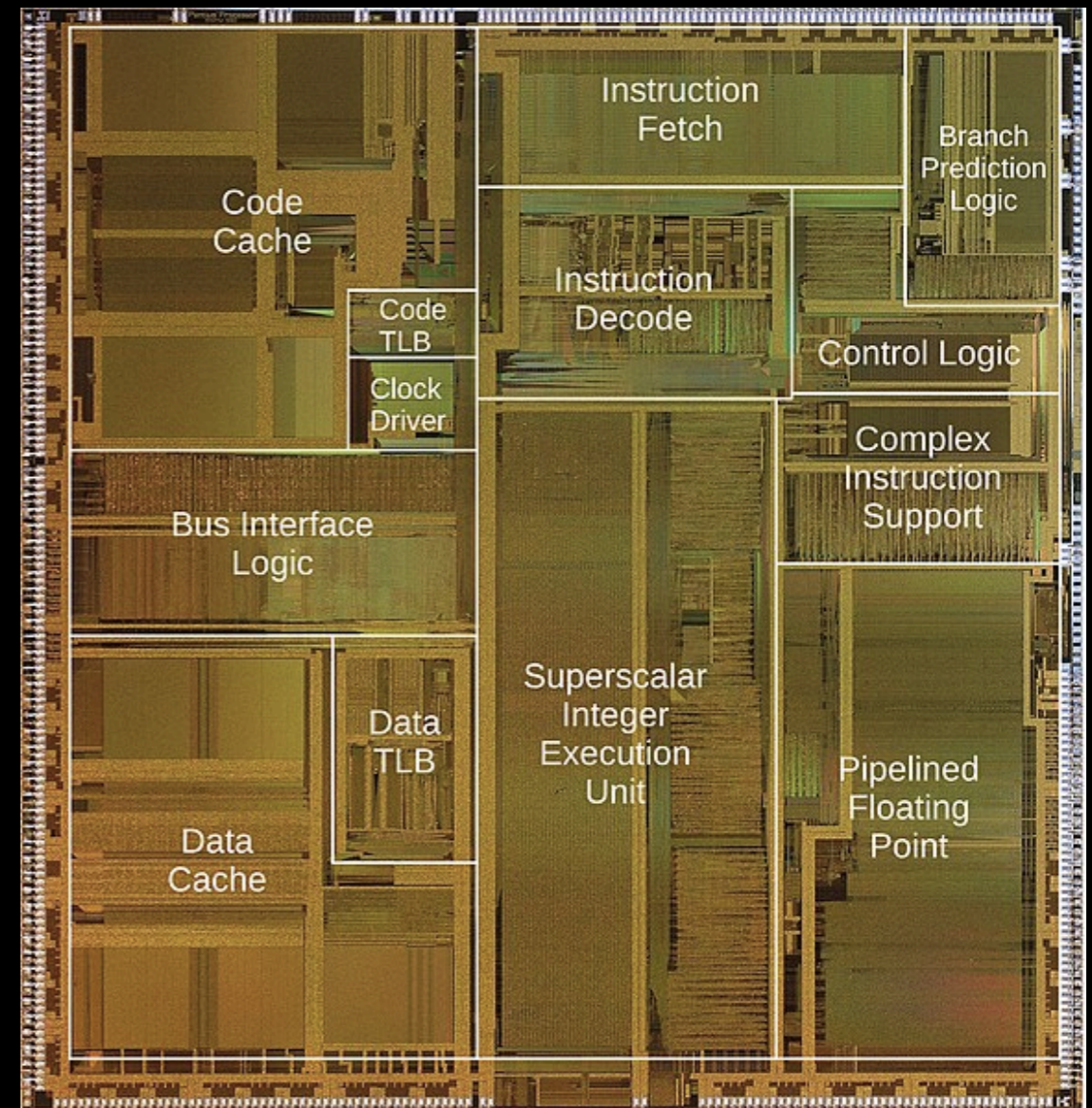
Verification assumptions were not *always* wrong

- Recall 4 assumptions:
 - Plausible
 - Tractable
 - Tolerable
 - Necessary
- *Sometimes* they hold



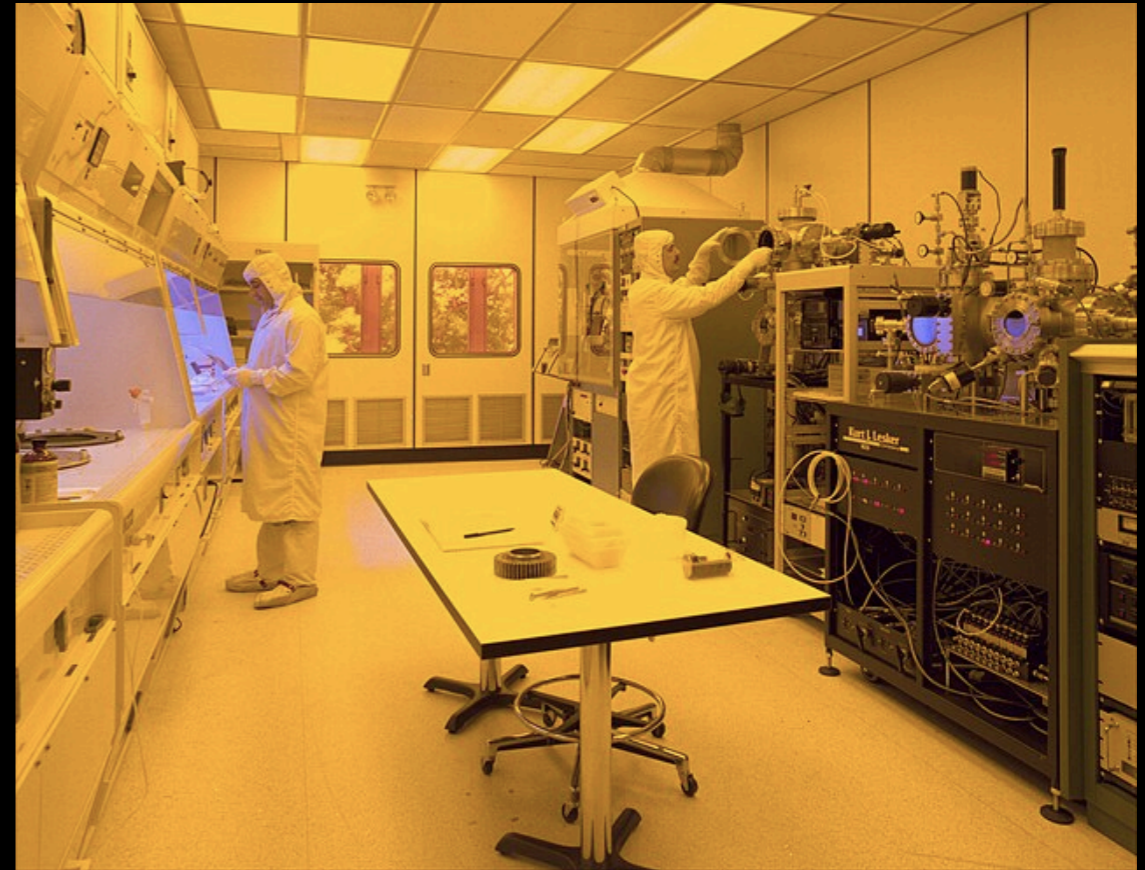
Verification: tractable

- Small systems
 - Inherently small
 - Or modular
 - Or small approximations
- Finite-state
 - Or finite approximations



Verification: tolerable

- Quality matters sometimes
 - Mistakes very expensive
- Big testing budgets already
 - Formal may be cheaper!



https://commons.wikimedia.org/wiki/File:Clean_room.jpg

Verification: necessary

- Regulations
 - Or warranties
- Governments may dictate
 - Industrial policy
 - National security
 - Public safety



Part 3

Results

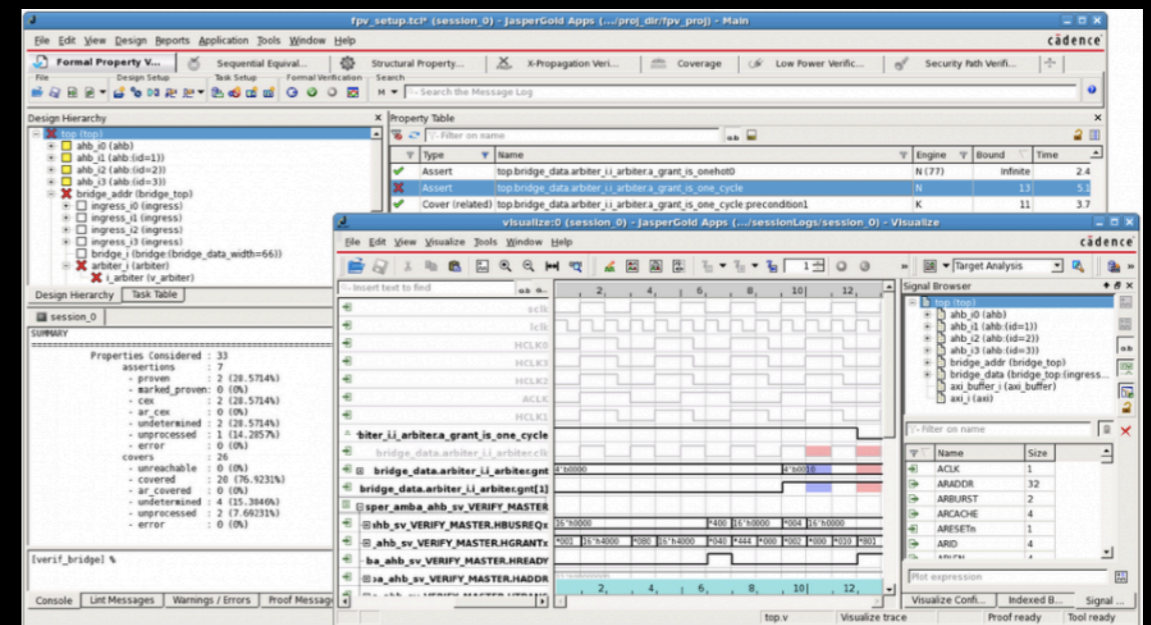
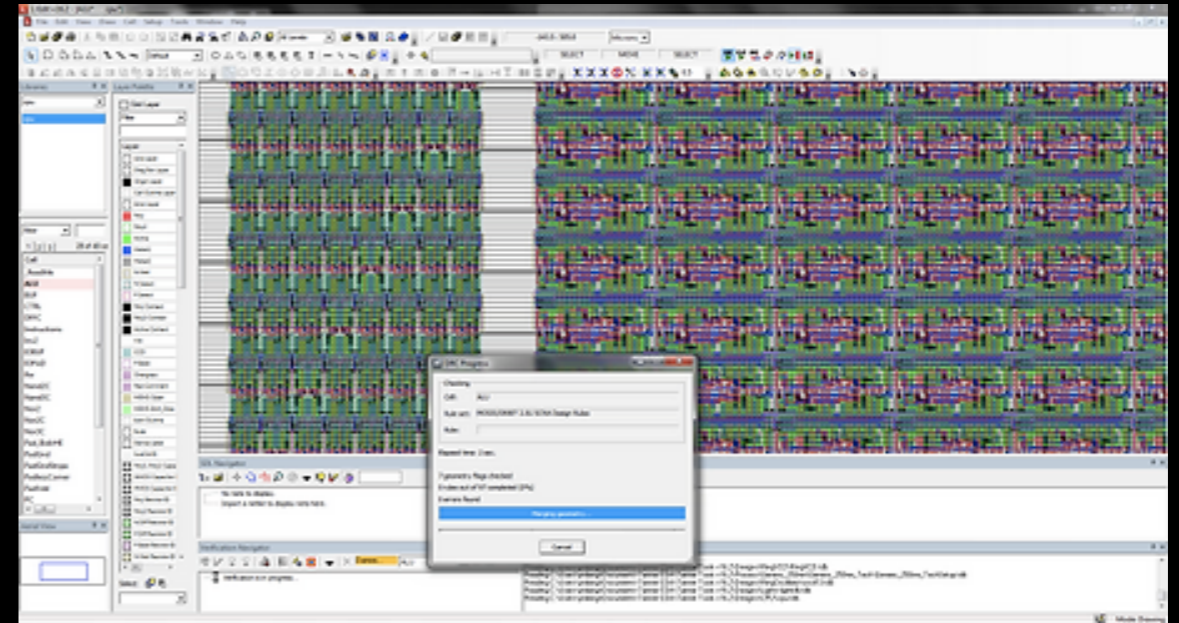
Part 3.1

50 years of proofs

Hardware

("EDA" electronic design automation)

- IBM, Intel, AMD, Arm, NXP, Qualcomm, Apple, STMicro, NVIDIA, Sun, HP
- Many tools
 - Academic (Murphi, NuSMV)
 - Industrial (Synopsys, Cadence)
- Hardware description languages (Verilog, VHDL)
- Spec languages (IBM Sugar, Motorola CBV, Intel ForSpec, VHDL Assertions, SystemVerilog SVA, Synopsys OVA)



Safety-critical

- Flight control: Airbus, Boeing, Dassault, Rockwell, Honeywell
 - Also NASA & ESA
- Metro rail control: Alstom, Thales, Siemens (Singapore, Paris, Sao Paulo, Ankara, Hong Kong)
- Flood dam control: Rotterdam
- Nuclear power plant control: France, China, Korea, USA
- Many tools: B, SCADE, PVS, Astrée, Polyspace, NuSMV



<https://flickr.com/photos/hugokernel/5027492067> - CC BY-NC-SA 2.0



[https://commons.wikimedia.org/wiki/File:Effet_SACEM_en_gare_d%27Auber_\(RER_A\)_par_Cramos.jpg](https://commons.wikimedia.org/wiki/File:Effet_SACEM_en_gare_d%27Auber_(RER_A)_par_Cramos.jpg)
CC BY-SA 3.0

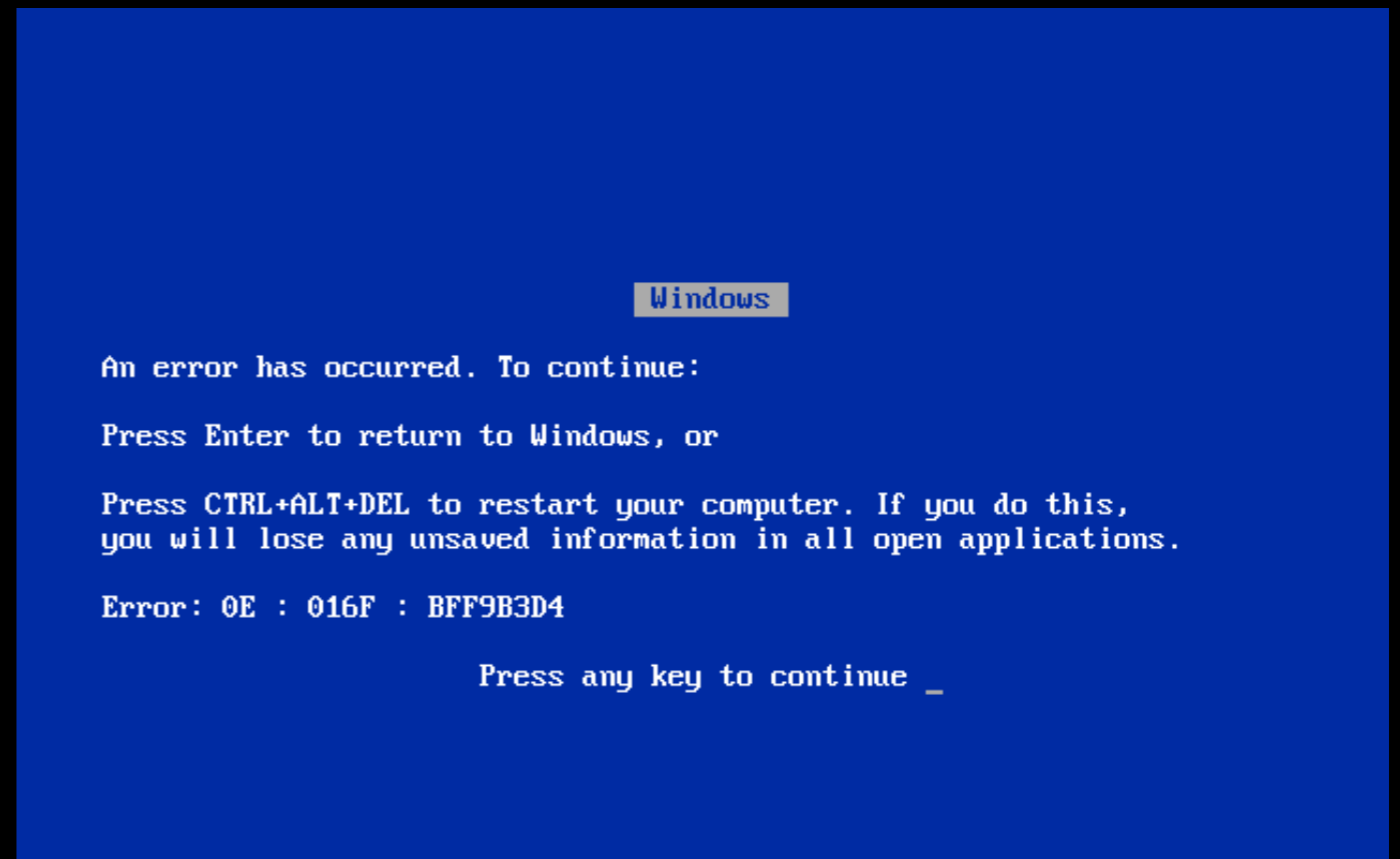
Telecoms & Networking

- OSI protocols in LOTOS
- AT&T phone net
 - 5ESS switch, CDMA BSS
 - CCITT SDL, SPIN, VeriSoft
- New SDN projects
 - OpenFlow, P4, Frenetic/
NetKAT (Cornell!)



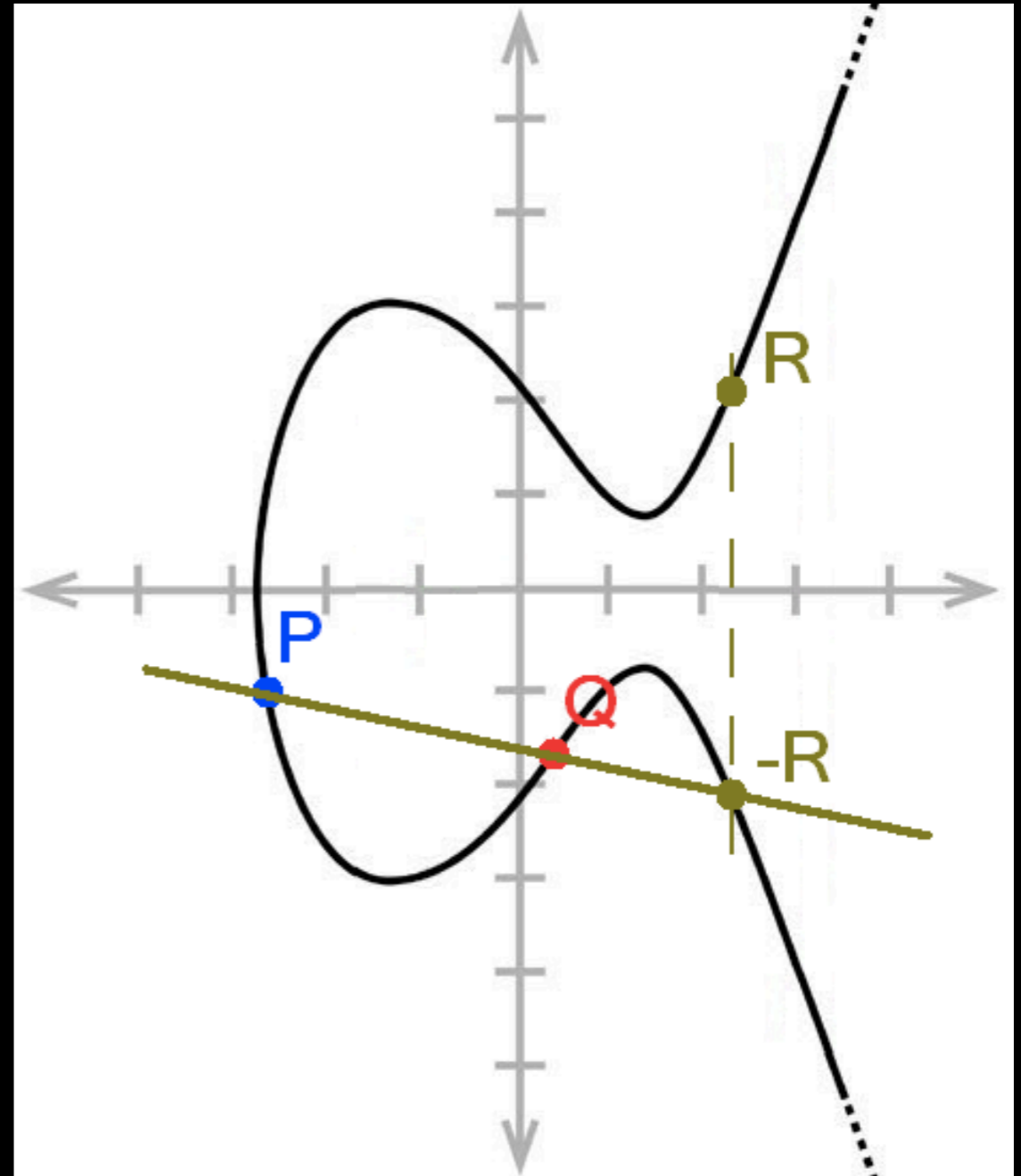
OS Drivers

- Internet nightmare
- Microsoft bad press
 - "Trustworthy Computing" initiative (early 2000s)
 - SLAM project
Static Driver Verifier
- Linux Driver Verification, DDVerify, Avinux
 - BLAST, CPAChecker, SATABS, CBMC




Cryptography

- Verified algorithms:
 - AES, SHA256, Curve25519
 - Signal protocol
- Verified implementations
 - Amazon s2n TLS stack
 - MSR & INRIA Everest project
- EasyCrypt, ProVerif, SAW, Cryptol, FStar and KaRaMeL (OCaml based)



Interactive proof heroism

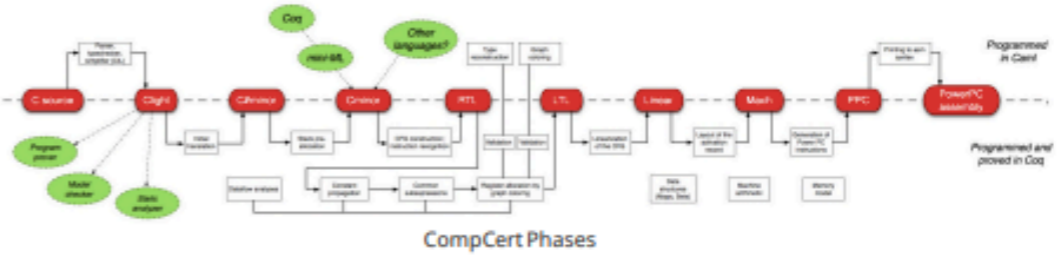
- Interactive proofs
 - Not automatic
 - Labor intensive
 - Arbitrarily deep properties
- Some heroic proofs
 - seL4: real-time microkernel
 - CompCert: C compiler
 - CakeML: ML compiler



CompCert

Formally Verified Optimizing C Compiler

CompCert is an **optimizing C compiler** which is **formally verified**, using machine-assisted mathematical proofs, to **guarantee the absence of compiler bugs**. The code it produces is proved to behave exactly as specified by the semantics of the source C program. This level of confidence in the correctness of the compilation process is **unprecedented** and contributes to meeting the **highest software assurance levels**.



CompCert Phases

"The striking thing about our CompCert results is that the middle-end bugs we found in all other compilers are absent. As of early 2011, the under-development version of CompCert is the only compiler we have tested for which Csmith cannot find wrong-code errors. This is not for lack of trying: we have devoted about six CPU-years to the task. The apparent unbreakability of CompCert supports a strong argument that developing compiler optimizations within a proof framework, where safety checks are explicit and machine-checked, has tangible benefits for compiler users."

Study by Regehr, Yang et al. on a development version of CompCert in 2011

In 2021, the **CompCert** development team received the prestigious **ACM Software System Award**.

Your Benefits:

- Using the **CompCert** C compiler is a natural complement to applying formal verification techniques (static analysis, program proof, model checking) at the source-code level. The correctness proof of **CompCert** guarantees that all safety properties verified on the source code automatically hold for the generated code as well.
- On typical embedded processors, the code generated by **CompCert** usually runs twice as fast as the code generated by GCC without optimizations, and only 20% slower than GCC at optimization level 3.

Availability:

- **CompCert** has been developed at INRIA by architect and lead developer Xavier Leroy with numerous renowned researchers contributing ideas, code, or feedback.
- Source code and documentation of **CompCert**, including the compiler proofs, can be downloaded from the website <http://compcert.inria.fr>. For research purposes, the usage of CompCert is free of charge.
- In 2014, INRIA and AbsInt entered a license agreement to provide commercial licenses to end users. AbsInt offers commercial licenses, provides industrial-strength support and maintenance, and contributes to the advancement of the tool.

Also interactive proofs are fairly big in academic PL?

- Language designs
 - Subtle properties
 - Pen-and-paper proofs
- 2005 "POPLMark challenge"
 - Now "mechanized" proofs
 - Sadly not automatic

Mechanized Metatheory for the Masses: The POPLMARK Challenge

Brian E. Aydemir¹, Aaron Bohannon¹, Matthew Fairbairn², J. Nathan Foster¹,
Benjamin C. Pierce¹, Peter Sewell², Dimitrios Vytiniotis¹, Geoffrey
Washburn¹, Stephanie Weirich¹, and Steve Zdancewic¹

¹ Department of Computer and Information Science, University of Pennsylvania
² Computer Laboratory, University of Cambridge

Subversion Revision: 171

Document generated on: May 11, 2005 at 15:53

Abstract. How close are we to a world where every paper on programming languages is accompanied by an electronic appendix with machine-checked proofs?

We propose an initial set of benchmarks for measuring progress in this area. Based on the metatheory of System F_{\leq} , a typed lambda-calculus with second-order polymorphism, subtyping, and records, these benchmarks embody many aspects of programming languages that are challenging to formalize: variable binding at both the term and type levels, syntactic forms with variable numbers of components (including binders), and proofs demanding complex induction principles. We hope that these benchmarks will help clarify the current state of the art, provide a basis for comparing competing technologies, and motivate further research.

**How did these proofs
happen?**

Part 3.2

50 years of tools

Tools!

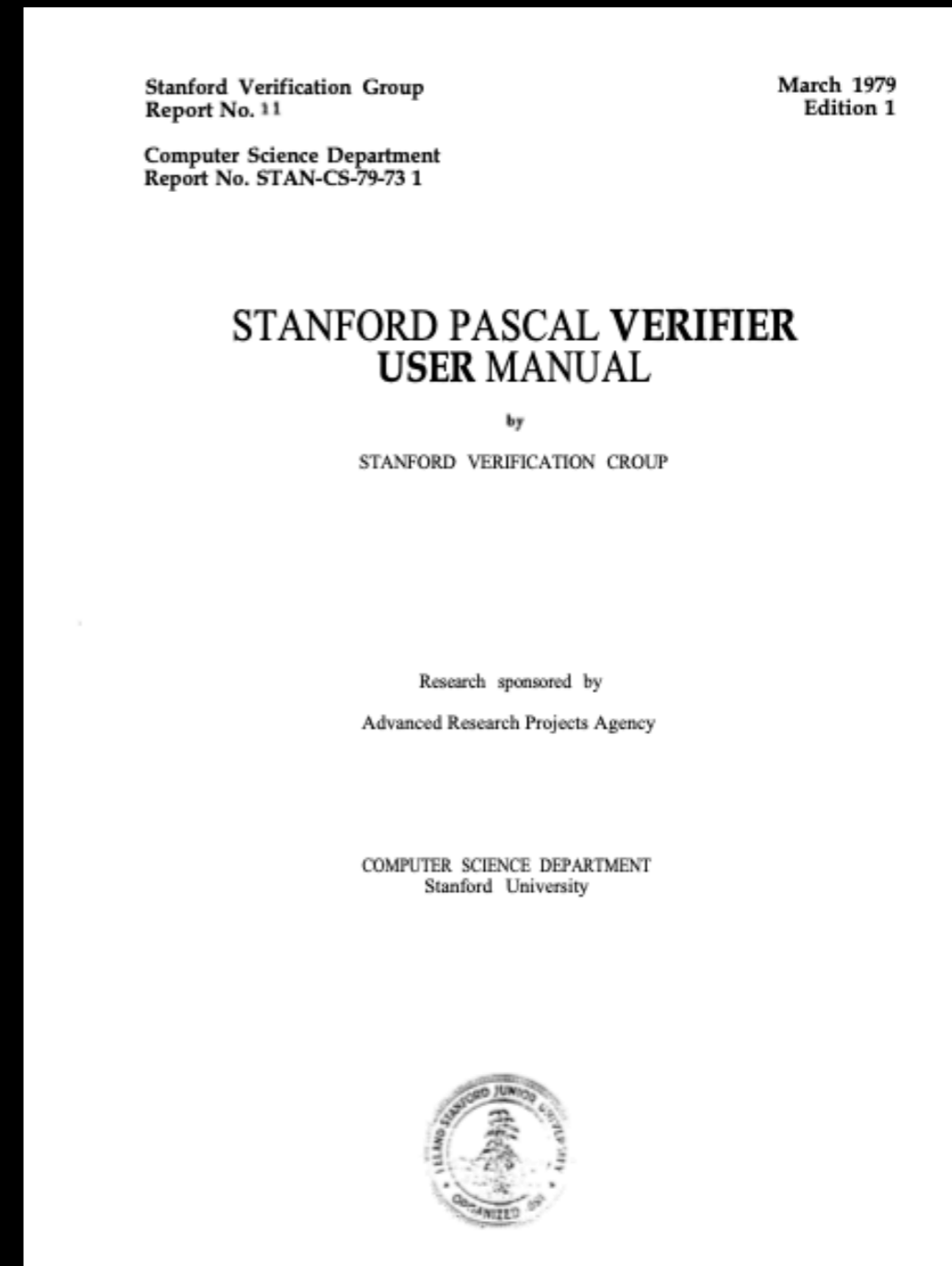
- Lots of tools
 - Deductive Verifiers
 - Abstract interpreters
 - Model checkers
 - Interactive provers
 - Automated provers
 - Many others besides...



https://commons.wikimedia.org/wiki/File:20060513_toolbox.jpg CC-BY-SA 2.5

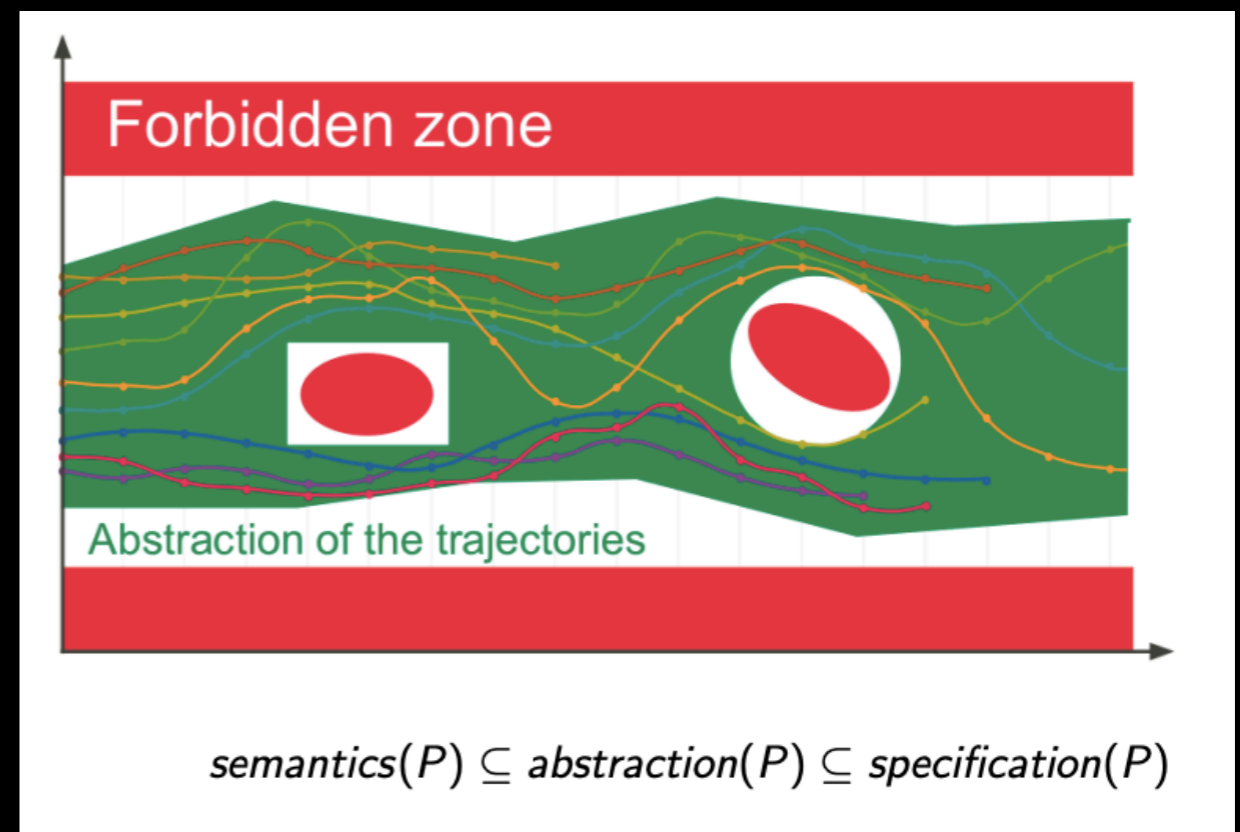
Deductive verifiers ("design by contract" checkers)

- Build verification conditions (VCs) from predicates and program logics
 - Preconditions, postconditions, invariants (implicit or explicit)
- Prove VCs however possible
 - Ideally: mostly-automatic
 - Reality: semi-manually
- Same as 1970s
- Extensions to many PLs (JML, SPARK, Frama-C) or basis of PL designs (Dafny, Wwhiley, Spec#)



Abstract interpreters

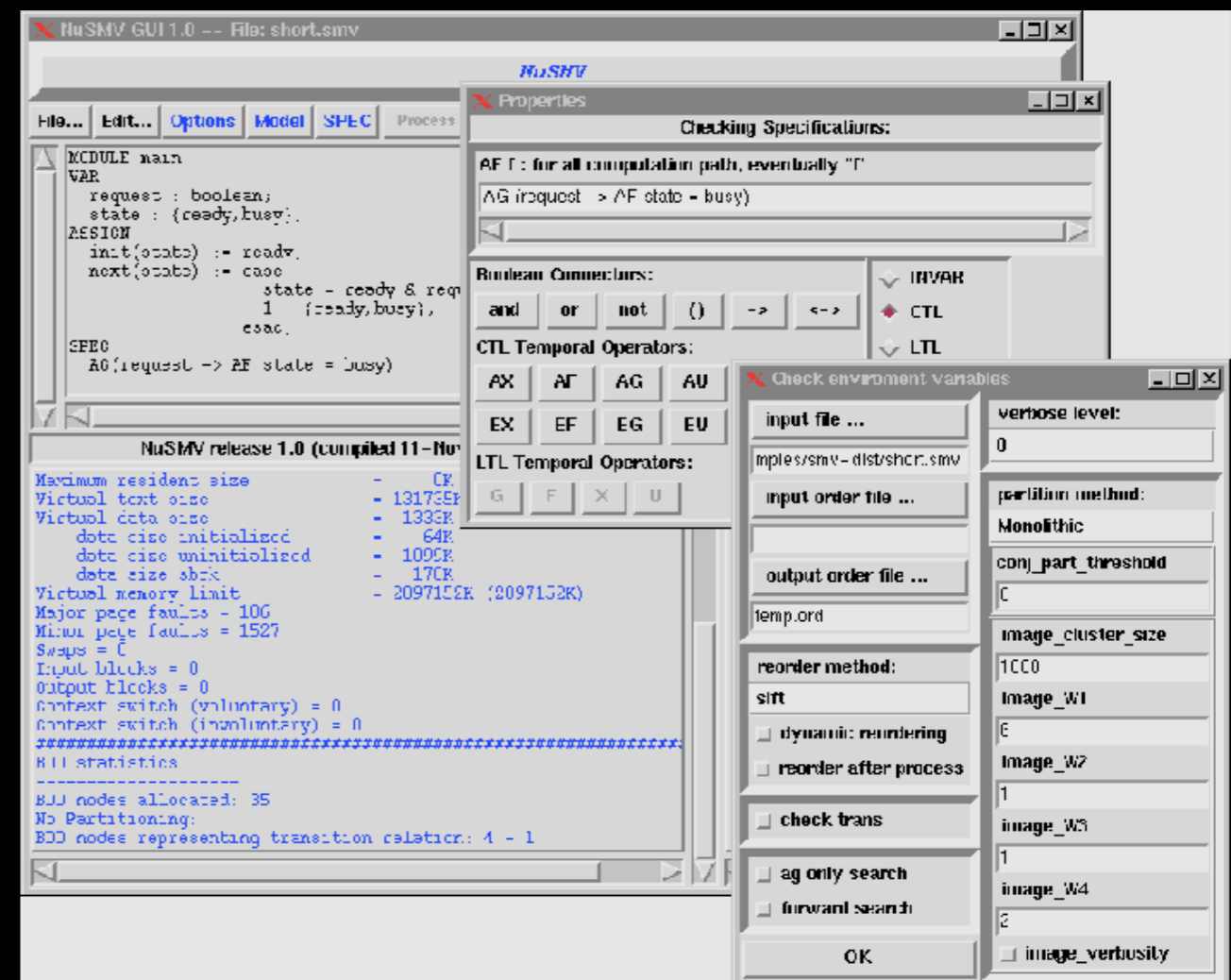
- Check local properties in sound over-approximation of code
 - arithmetic, nullptr, assertions
- Automatic, fast, but imprecise
 - False alarms
- Big in safety-critical
 - Astrée, Polyspace, IKOS
- Some more-general software
 - Facebook Infer, MIRAI



<https://ntrs.nasa.gov/citations/20190032528>

Model checkers

- Check state transition function is logical model of spec formula
 - Temporal: "always, eventually"
 - Often concurrency friendly
 - Automatic but expensive
- Explicit: SPIN, TLA+, CADP, Murphi, DiVinE
- Symbolic: NuSMV, BLAST, SLAM, CBMC
- Big in hardware, telecoms



Interactive provers

- Arbitrarily deep properties
 - Labor intensive
 - Like a very confusing IDE
- Dependent type based
 - CTT (NuPRL -- Cornell!)
 - CoC/CIC (Coq, Lean)
 - UTT (Agda)
 - QTT (Idris)
- Higher-order logic based
 - Isabelle/HOL, HOL4

```

lemma compare_coeff (n : N) (a : N → R) (h : ∑ k in range (n + 1), monomial
→ a k = 0 := by
  intro m hm
  have h' := by congrm (coeff $h (n-m))
  clear h
  rw [finset_sum_coeff, coeff_zero] at h'
  let f : N → R := fun k => if k = m then a m else 0
  have h'' : ∑ b in range (n+1), f b = 0 := by
    rw [<- h']
    apply sum_congr rfl
    intro b hb
    rw [coeff_monomial]
    simp
    have iff : b = m ↔ n-b = n-m := by
      constructor
      . intro bm
        simp [bm]
      intro nbm
      have hb' : b ≤ n := by simp at hb; linarith
      have hm' := Nat.sub_add_cancel hm
      have hb'' := Nat.sub_add_cancel hb'
      linarith [nbm, hm', hb'']

    rcases em (b=m) with bm | nbm
    . have nbm : n-b = n-m := by rw [<-iff]; assumption
      simp [bm, nbm]
    have nbm : n-b ≠ n-m := by contrapose! bm; rw [iff]; assumption
      simp [bm, nbm]
  have h''' : m ∈ range (n+1) := by
    simp
    linarith
  rw [<- add_sum_erase _ _ h'''] at h''
  simp at h''
  assumption

```

Automated provers

- Decide (semi-)decidable logic
 - Fast
 - Subroutines for other tools
- SAT & SMT solvers
 - Z3, CVC, Yices, Alt-Ergo
- FOL provers
 - E, Vampire, SPASS, Prover9

```
(set-info :smt-lib-version 2.6)
(set-logic AUFDLIA)
(set-info :source |
Generated by: Yatin Manerkar
Generated on: 2021-03-02
Generator: UCLID5
Application: Hardware ordering verification
Target Solver: cvc4
|)
(set-info :license "https://creativecommons.org/licenses/by/4.0/")
(set-info :category "industrial")
(set-info :status unsat)

(declare-datatypes ((_tuple_1 0)) (((_tuple_1 (nExists Bool) (nTime Int)
(declare-datatypes ((_tuple_2 0)) (((_tuple_2 (Fetch _tuple_1) (Execute
(declare-datatypes ((_enum_0 0)) (((Read) (Write) (Fence))))))
(declare-datatypes ((_tuple_0 0)) (((_tuple_0 (valid Bool) (globalID Int)
(declare-fun initial_1_mp () (Array Int _tuple_0))
(declare-fun initial_3__uclid_2_nodes_var () (Array Int _tuple_2))
(declare-fun initial_2__uclid_1_testInstrs_bound_input () (Array Int _
(assert (= (uopType (select initial_1_mp 0)) Write))
(assert (= (coreID (select initial_1_mp 0)) 0))
(assert (= (physAddr (select initial_1_mp 2)) 1))
(assert (let ((a!1 (<= (nTime (Execute (select initial_3__uclid_2_node
(nTime (Writeback (select initial_3__uclid_2_nodes_var
(a!3 (not (<= (globalID (select initial_1_mp 2))
(globalID (select initial_1_mp 3)))))
(a!5 (not (and (valid (select initial_1_mp 3))
(valid (select initial_1_mp 2)))))
(let ((a!2 (and (nExists (Writeback (select initial_3__uclid_2_nodes_v
(nExists (Execute (select initial_3__uclid_2_nodes_var
(not a!1)))
(a!4 (and (= (coreID (select initial_1_mp 3))
(coreID (select initial_1_mp 2))
```

Part 3.3

50 years of theory

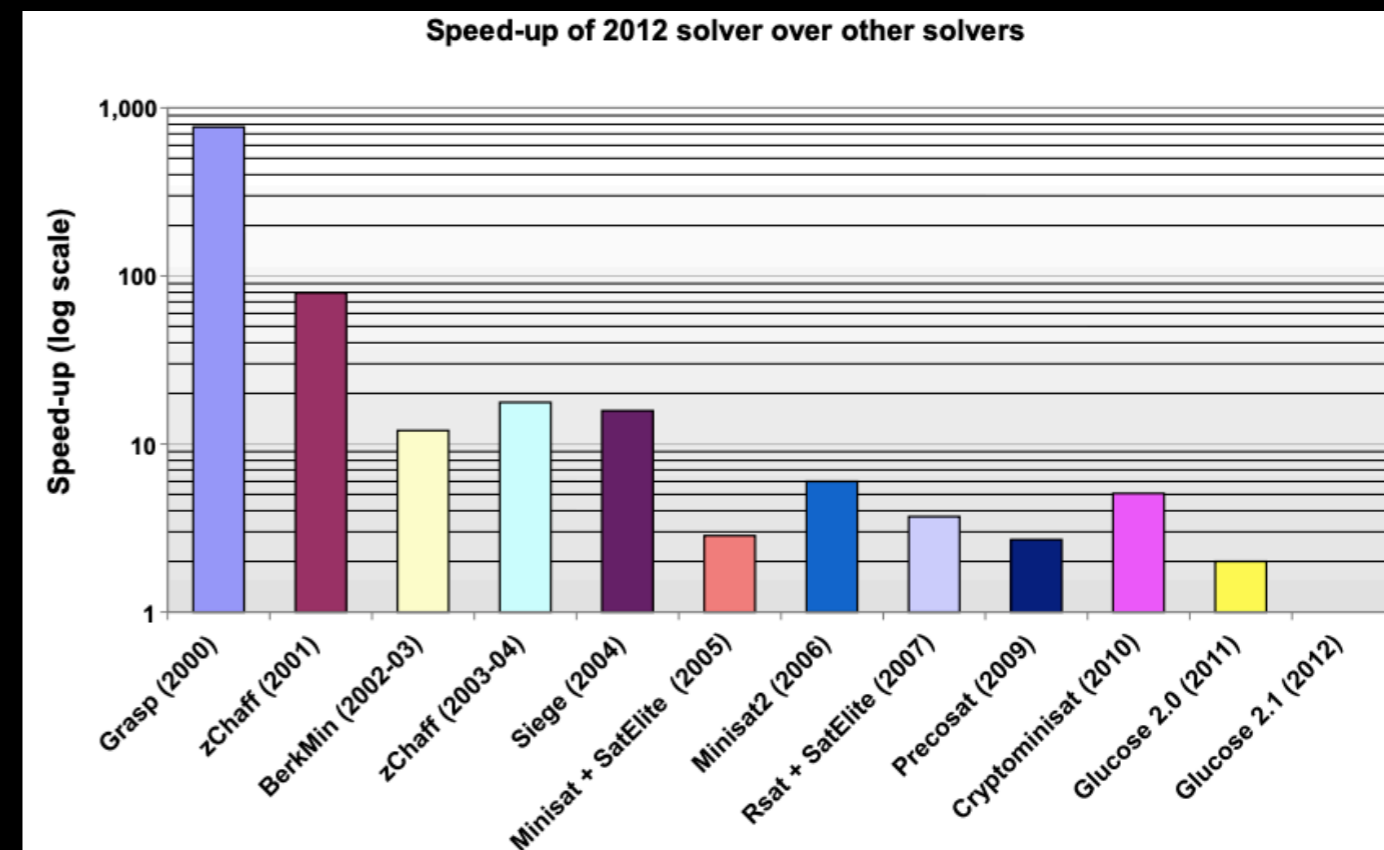
Theory has improved

- Big improvements
- So much theory
- Big delays
- 30+ years sometimes



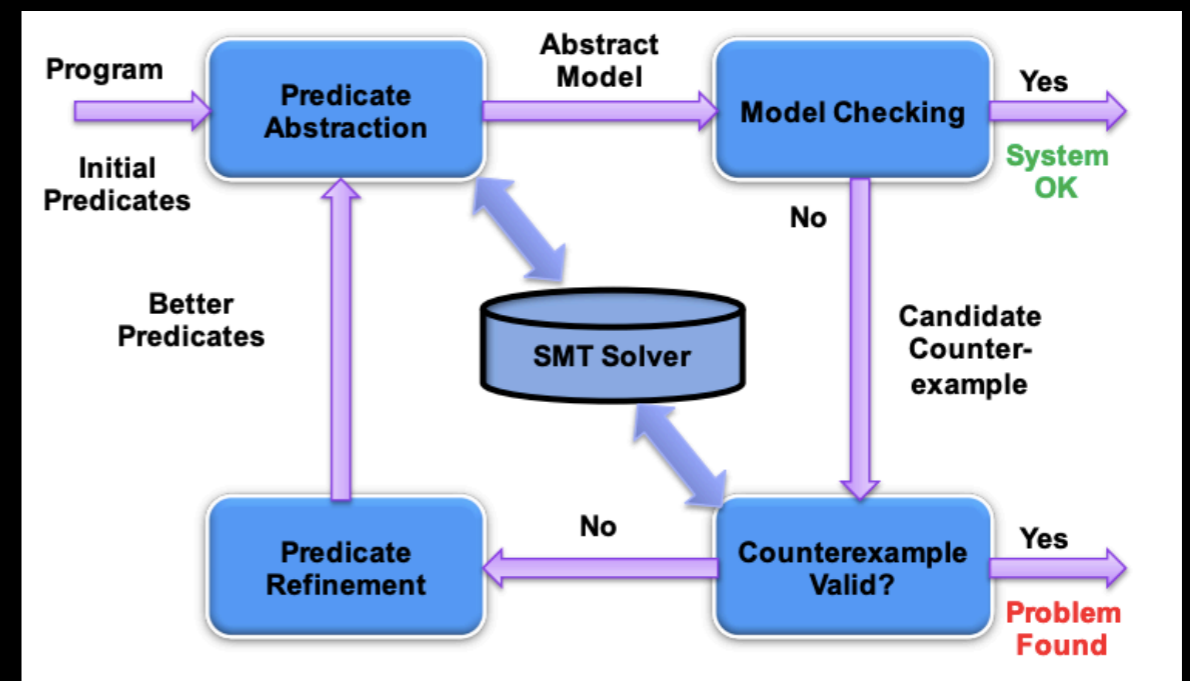
Automatic provers improved an incredible amount

- FOL superposition early 90s
 - *20 years* after KB
- Heuristic SAT revolution (GRASP, Chaff) late 90s
 - *35 years* after DPLL
- SMT solvers: SAT solvers + theory decision procedures



Model checkers improved an incredible amount

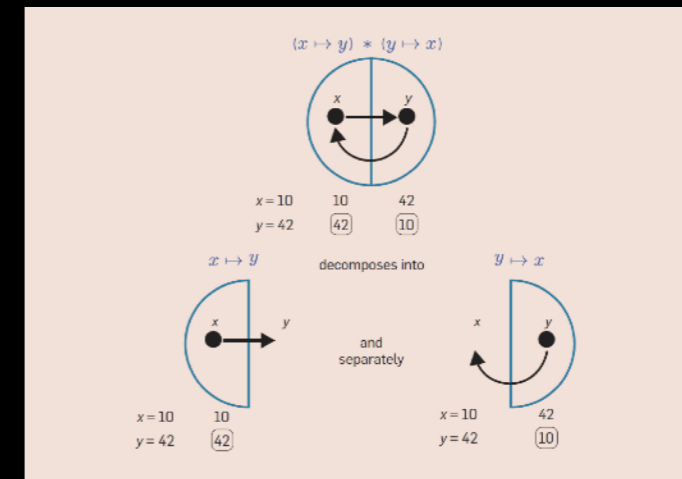
- Partial-order reduction, BDDs, SAT-based BMC, CEGAR, IMC, IC3/PDR
- Concrete-to-symbolic
- Eager-to-lazy
- Finite-to-infinite
- Leveraged improvements in automatic provers



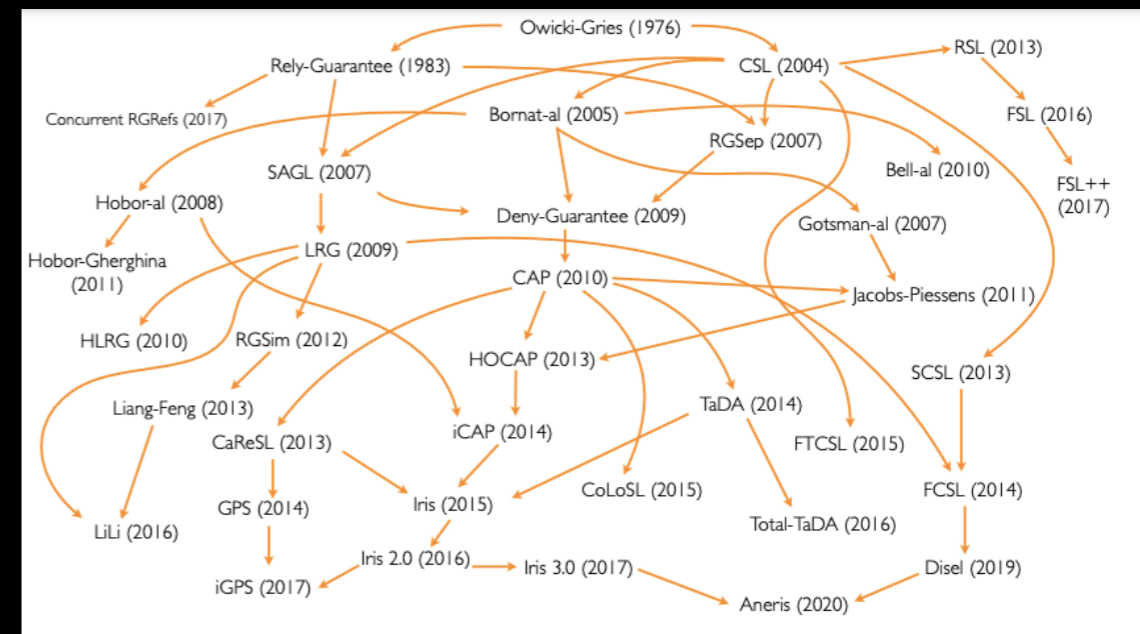
<https://arieg.bitbucket.io/pdf/ModelChecking.pdf>

Program logics got better at mutable aliased state and concurrency

- Separation logic
 - 33 years after Hoare logic
- Concurrent separation logic
 - 31 years after Owicki-Gries
- Linear logic
- Tasty hybrids (eg. Rust)



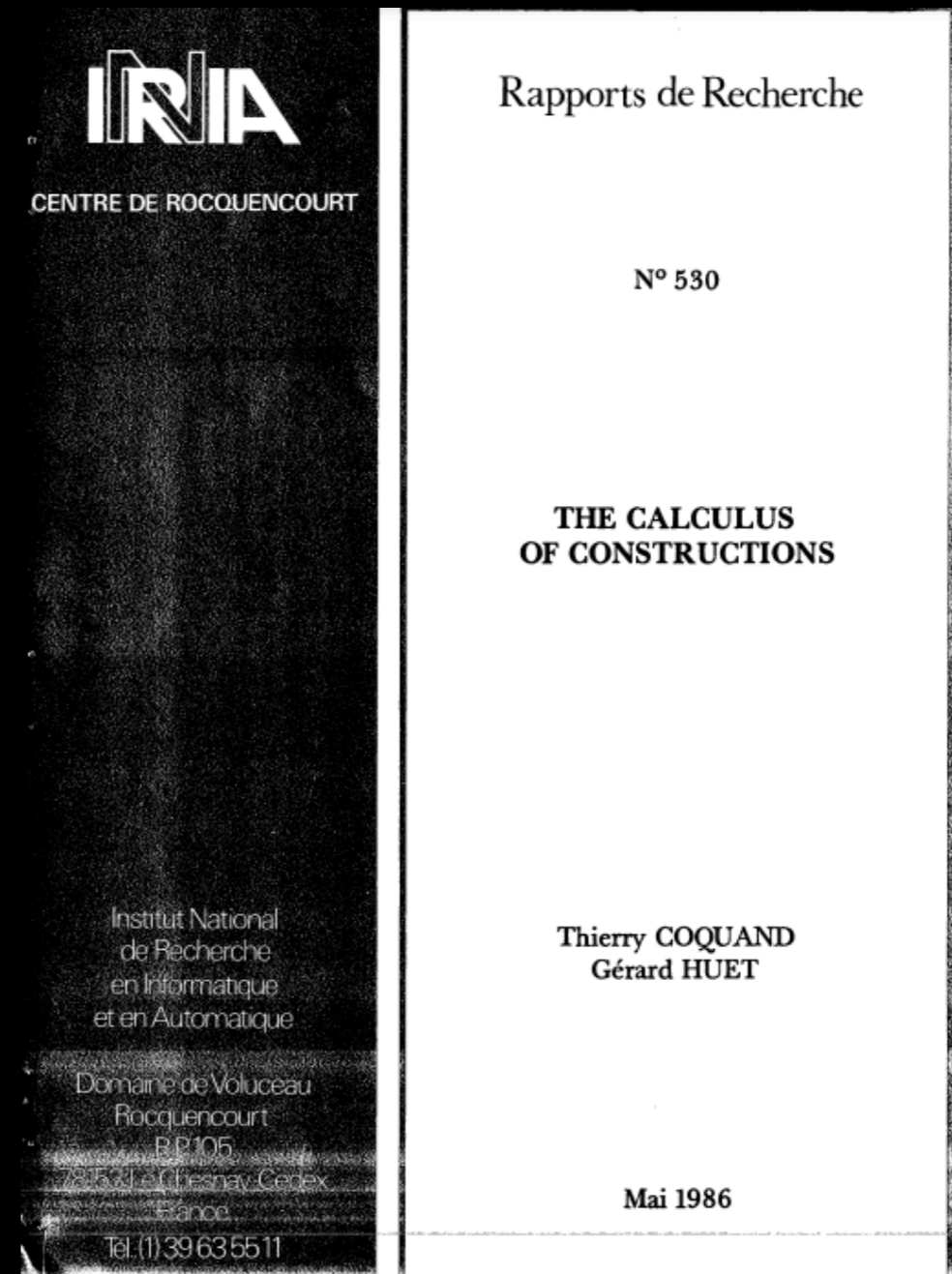
<https://cacm.acm.org/magazines/2019/2/234356-separation-logic/fulltext>



<https://ilyasergey.net/assets/other/CSL-Family-Tree.pdf>

Interactive provers improved

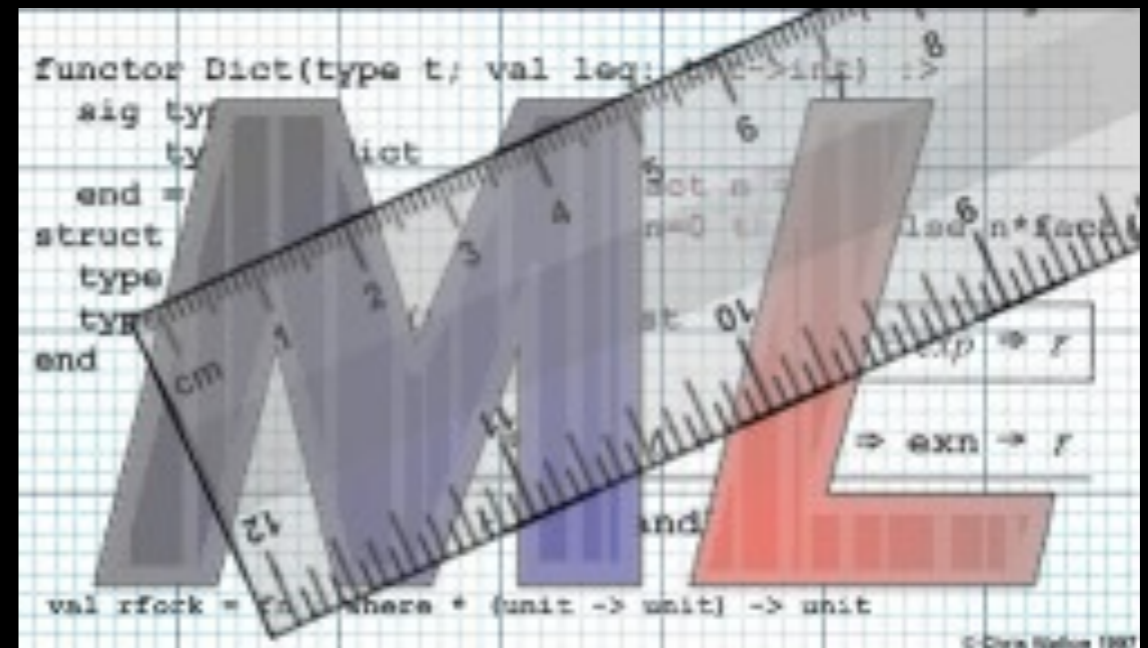
- Dependent-type extensions:
 - Universe hierarchies
 - (Co-)Inductive types
 - Dependent patterns
- Higher order unification
- Implicit arguments, typeclasses
- Reflection, automation
 - Awkward segue to .. ML



Part 3.4
50 years of ML
(bonus)

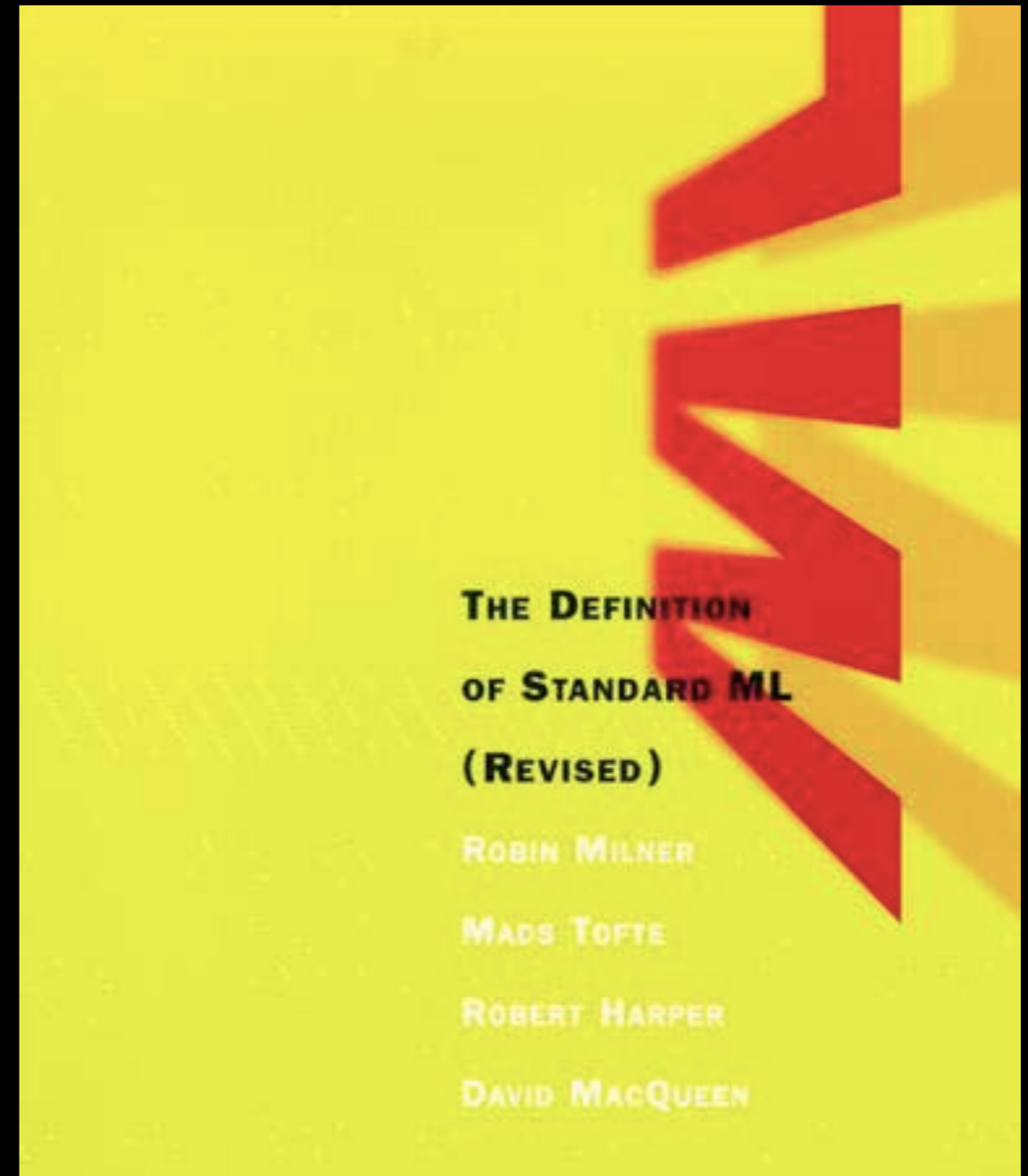
ML is the C of formal world

- ML dialects (OCaml) common
 - Implementation language
 - Spec-and-proof language
 - Object-of-study language
 - All of the above, at once!



Why ML?

- Small, clean, expressive
 - Helps write complex tools
 - Easy to subset or extend
- Correspondence to logic
 - Type *and* proof terms
- Has a formal semantics
 - Easy to study *in* these tools



Cyclone (and Rust)

- Idea: "ML systems language"
 - *Help* mainstream software
 - Bring safety / correctness
- Or: *adapt* ML to systems niche
 - Affine types, borrowing
 - Formal-world ideas

Cyclone: A safe dialect of C

Trevor Jim* Greg Morrisett† Dan Grossman† Michael Hicks†
James Cheney† Yanling Wang†

Abstract

Cyclone is a safe dialect of C. It has been designed from the ground up to prevent the buffer overflows, format string attacks, and memory management errors that are common in C programs, while retaining the ability to write programs that reach deeper than just poor training and effort: they have their roots in the design of C itself.

Take buffer overflows, for example. Every introductory C programming course warns against them and teaches techniques to avoid them, yet they continue to be announced in security bulletins every week.

<https://www.cs.cornell.edu/Projects/cyclone/papers/cyclone-safety.pdf>

The Rust Language

Nicholas Matsakis
Mozilla Research
nmatsakis@mozilla.com

Felix S. Klock II
Mozilla Research
pnkfelix@mozilla.com

1. ABSTRACT

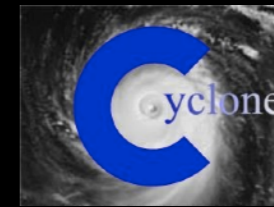
Rust is a new programming language for developing reliable and efficient systems. It is designed to support concurrency and parallelism in building applications and libraries that take full advantage of modern hardware. Rust's static type system is safe¹ and expressive and provides strong guarantees to be free of memory errors (dangling pointers, double frees) as well as data races.

To control aliasing and ensure type soundness, Rust incorporates a notion of *ownership* into its type system. The unique owner of an object can hand that ownership off to new owner; but the owner may also hand off *borrowed references* to (or into) the object. These so-called *borrowed* objects

<http://dx.doi.org/10.1145/2663171.2663188>

50 year tech-transfer from formal world?

- Today: 2023
- T-10: 2013-ish, Rust (Mozilla)
- T-20: 2003-ish, Cyclone (AT&T, Cornell)
- T-30: 1993-ish, OCaml (INRIA)
- T-40: 1983-ish, SML/NJ (AT&T)
- T-50: 1973-ish, ML (Edinburgh)
- (dates fudged, for drama)



Conclusion

Prospects

- Formal world *fairly* niche
- Maybe forever
 - *Mass programming*
 - Proof is hard
 - Often nothing to prove
- Niche will survive
 - Tools never better
 - Relevance never higher



https://commons.wikimedia.org/wiki/File:Leitstand_2.jpg - CC-BY-SA 3.0

Never more active

- Many conferences
 - CAV, IJCAR, CADE, FM, FMCAD, VMCAI, TACAS, ILP, ITP, SAT, ICLP, LICS, FLoC ...
- Many with competitions!
 - SMT-COMP, SV-COMP, HCVC, COCO, QBF, TPTP, MCC, ...
- Some with Cornell faculty as PC members (or chairs!)



Alexandra Silva at FLoC 2022
<https://www.floc2022.org/pictures>

Your course, and Cornell

- Your course is an appetizer
 - Just learning OCaml
- Unit on (manual) proof
 - Curry-Howard is discussed!
- But Cornell is a "formal world site"
 - 40+ year NuPRL project
 - Search "formal", "logic" or "verification" on faculty page
 - Maybe explore?



Either way, let's call it a day

- Hopefully amusing or educational
- Fun for me to put talk together
- Thanks for your time
- Good luck on exams!



Fini

This talk is CC-BY-SA 4.0 because of the wide variety
of amusing images I used with CC-SA licenses