A HISTORY OF F#

FROM EUCLID TO TYPE PROVIDERS

Rachel Reese
@rachelreese
rachelree.se

By and large, the goal of introducing new programming languages has been to make it simpler to express more complex behavior.

Gul A. Agha.
Actors, a model of concurrent computation in distributed systems.
Ph.D. dissertation. March 1985.

Mother Tongues

Tracing the roots of computer languages through the ages

Just like half of the world's spoken tongues, most of the 2,300-plus computer programming languages are either endangered or extinct. As powerhouses C/C++, Visual Basic, Cobol, Java and other modern source codes dominate our systems, hundreds of older languages are running out of life.

An ad hoc collection of engineers-electronic lexicographers, if you will-aim to save, or at least document the lingo of classic software. They're combing the globe's 9 million developers in search of coders still fluent in these nearly forgotten lingua frangas. Among the most endangered are Ada, APL, B (the predecessor of C), Lsp, Oberon, Smalltalk, and Simula.

Code-raker Grady Booch, Rational Software's chief scientist, is working with the Computer History Musuem in Silicon Valley to record and, in some cases, maintain languages by writing new compilers so our ever-changing hardware can grok the code. Why bother? "They tell us about the state of software practice, the minds of their inventors, and the technical, social, and economic forces that shaped history at the time," Booch explains. "They'll provide the raw material for software archaeologists, historians, and developers to learn what worked, what was brilliant, and what was an utter failure." Here's a peek at the strongest branches of programming's family tree. For a nearly exhaustive rundown, check out the Language List at *HTTP://www.informatik.uni-freiburg.de/Java/misc/lang_list.html.* - Michael Mendeno

Key

1954 **Year introduced**

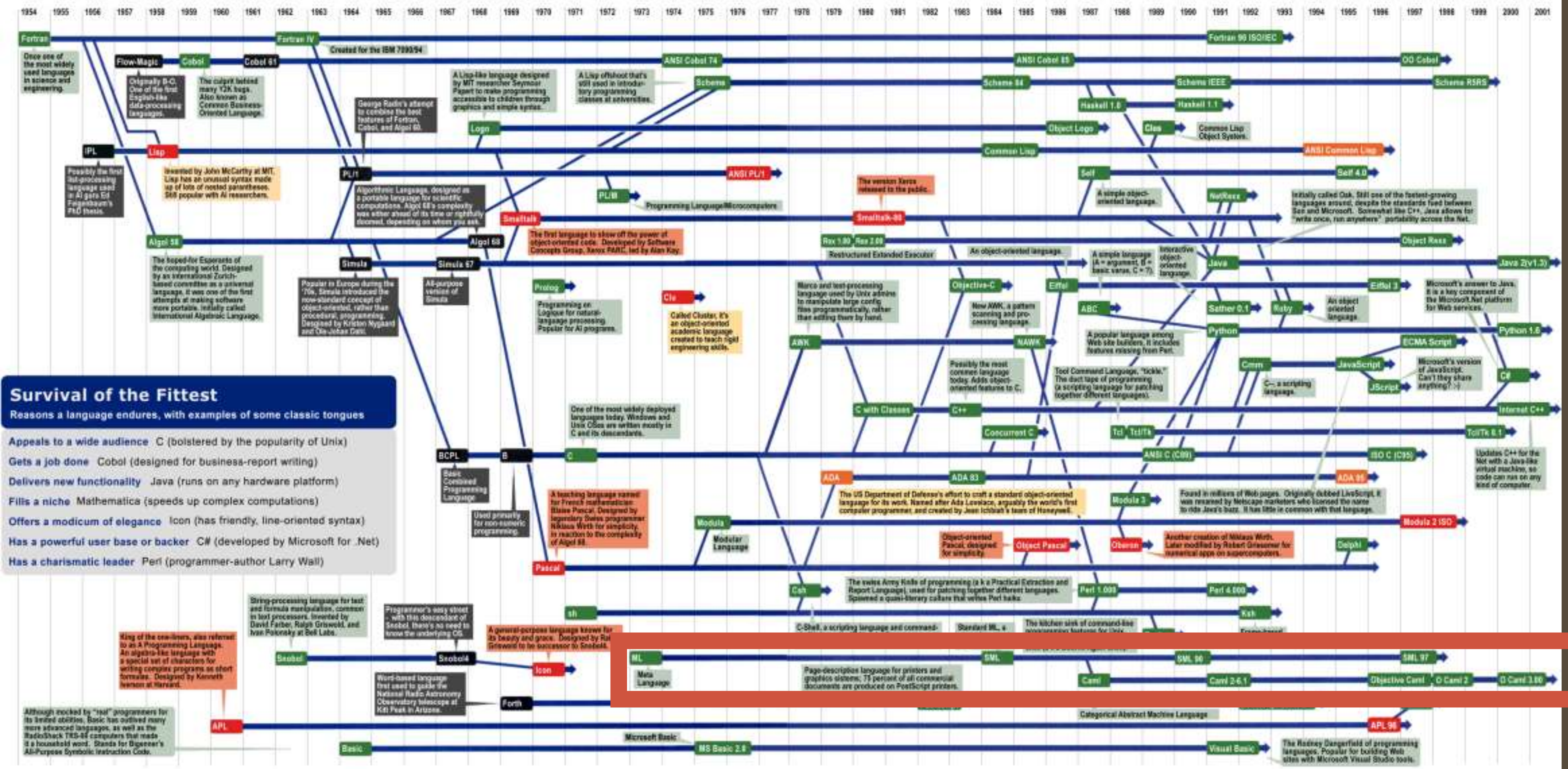
🟩 **Active:** thousands of users

🟧 **Protected:** taught at universities; compilers available

🟥 **Endangered:** usage dropping off

⬛ **Extinct:** no known active users or up-to-date compilers

➡ **Lineage continues**



Once one of the most widely used languages in science and engineering.

Created for the IBM 7090/94

Originally B-O. One of the first English-like data-processing languages.

The culprit behind many Y2K bugs. Also known as Common Business-Oriented Language.

George Radin's attempt to combine the best features of Fortran, Cobol, and Algol 60.

A Lisp-like language designed by MIT researcher Seymour Papert to make programming accessible to children through graphics and simple syntax.

A Lisp offshoot that's still used in introductory programming classes at universities.

Common Lisp Object System.

Possibly the first list-processing language used in AI guru Ed Feigenbaum's PhD thesis.

Invented by John McCarthy at MIT. Lisp has an unusual syntax made up of lots of nested parentheses. Still popular with AI researchers.

Algorithmic Language, designed as a portable language for scientific computations. Algol 68's complexity was either ahead of its time or rightfully doomed, depending on whom you ask.

Programming Language/Microcomputers

The version Xerox released to the public.

A simple object-oriented language.

Initially called Oak. Still one of the fastest-growing languages around, despite the standards fued between Sun and Microsoft. Somewhat like C++, Java allows for "write once, run anywhere" portability across the Net.

The first language to show off the power of object-oriented code. Developed by Software Concepts Group, Xerox PARC, led by Alan Kay.

Restructured Extended Executor

An object-oriented language.

A simple language (A = argument, B = basic verse, C = 7).

Interactive object-oriented language.

The hoped-for Esperanto of the computing world. Designed by an international Zurich-based committee as a universal language, it was one of the first attempts at making software more portable. Initially called International Algebraic Language.

Popular in Europe during the '70s, Simula introduced the now-standard concept of object-oriented, rather than procedural, programming. Designed by Kristen Nygaard and Ole-Johan Dahl.

All-purpose version of Simula

Programming en Logique for natural-language processing. Popular for AI programs.

Called Cluster, it's an object-oriented academic language created to teach rigid engineering skills.

Marco and text-processing language used by Unix admins to manipulate large config files programmatically, rather than editing them by hand.

New AWK, a pattern scanning and processing language.

An object oriented language.

Microsoft's answer to Java. It is a key component of the Microsoft.Net platform for Web services.

A popular language among Web site builders. It includes features missing from Perl.

Possibly the most common language today. Adds object-oriented features to C.

Tool Command Language, "tickle." The duct tape of programming (a scripting language for patching together different languages).

C--, a scripting language.

Microsoft's version of JavaScript. Can't they share anything? :-)

One of the most widely deployed languages today. Windows and Unix OSes are written mostly in C and its descendants.

Updates C++ for the Net with a Java-like virtual machine, so code can run on any kind of computer.

Basic Combined Programming Language

Used primarily for non-numeric programming.

A teaching language named for French mathematician Blaise Pascal. Designed by legendary Swiss programmer Niklaus Wirth for simplicity, in reaction to the complexity of Algol 68.

The US Department of Defense's effort to craft a standard object-oriented language for its work. Named after Ada Lovelace, arguably the world's first computer programmer, and created by Jean Ichbiah's team of Honeywell.

Found in millions of Web pages. Originally dubbed LiveScript, it was renamed by Netscape marketers who licensed the name to ride Java's buzz. It has little in common with that language.

Modular Language

Object-oriented Pascal, designed for simplicity.

Another creation of Niklaus Wirth. Later modified by Robert Griesemer for numerical apps on supercomputers.

The swiss Army Knife of programming (a k a Practical Extraction and Report Language), used for patching together different languages. Spawned a quasi-literary culture that writes Perl haiku.

C-Shell, a scripting language and command-

Standard ML, a

The kitchen sink of command-line programming features for Unix

Frame-based

String-processing language for text and formula manipulation, common in text processors. Invented by David Farber, Ralph Griswold, and Ivan Polonsky at Bell Labs.

Programmer's easy street - with this descendant of Snobol, there's no need to know the underlying OS.

A general-purpose language known for its beauty and grace. Designed by Ralph Griswold to be successor to Snobol4.

King of the one-liners, also referred to as A Programming Language. An algebra-like language with a special set of characters for writing complex programs as short formulas. Designed by Kenneth Iverson at Harvard.

Meta Language

Page-description language for printers and graphics sistems; 75 percent of all commercial documents are produced on PostScript printers.

Word-based language first used to guide the National Radio Astronomy Observatory telescope at Kitt Peak in Arizona.

Categorical Abstract Machine Language

Although mocked by "real" programmers for its limited abilities, Basic has outlived many more advanced languages, as well as the RadioShack TRS-80 computers that made it a household word. Stands for Bigenner's All-Purpose Symbolic Instruction Code.

Microsoft Basic

The Rodney Dangerfield of programming languages. Popular for building Web sites with Microsoft Visual Studio tools.

Survival of the Fittest

Reasons a language endures, with examples of some classic tongues

Appeals to a wide audience C (bolstered by the popularity of Unix)

Gets a job done Cobol (designed for business-report writing)

Delivers new functionality Java (runs on any hardware platform)

Fills a niche Mathematica (speeds up complex computations)

Offers a modicum of elegance Icon (has friendly, line-oriented syntax)

Has a powerful user base or backer C# (developed by Microsoft for .Net)

Has a charismatic leader Perl (programmer-author Larry Wall)

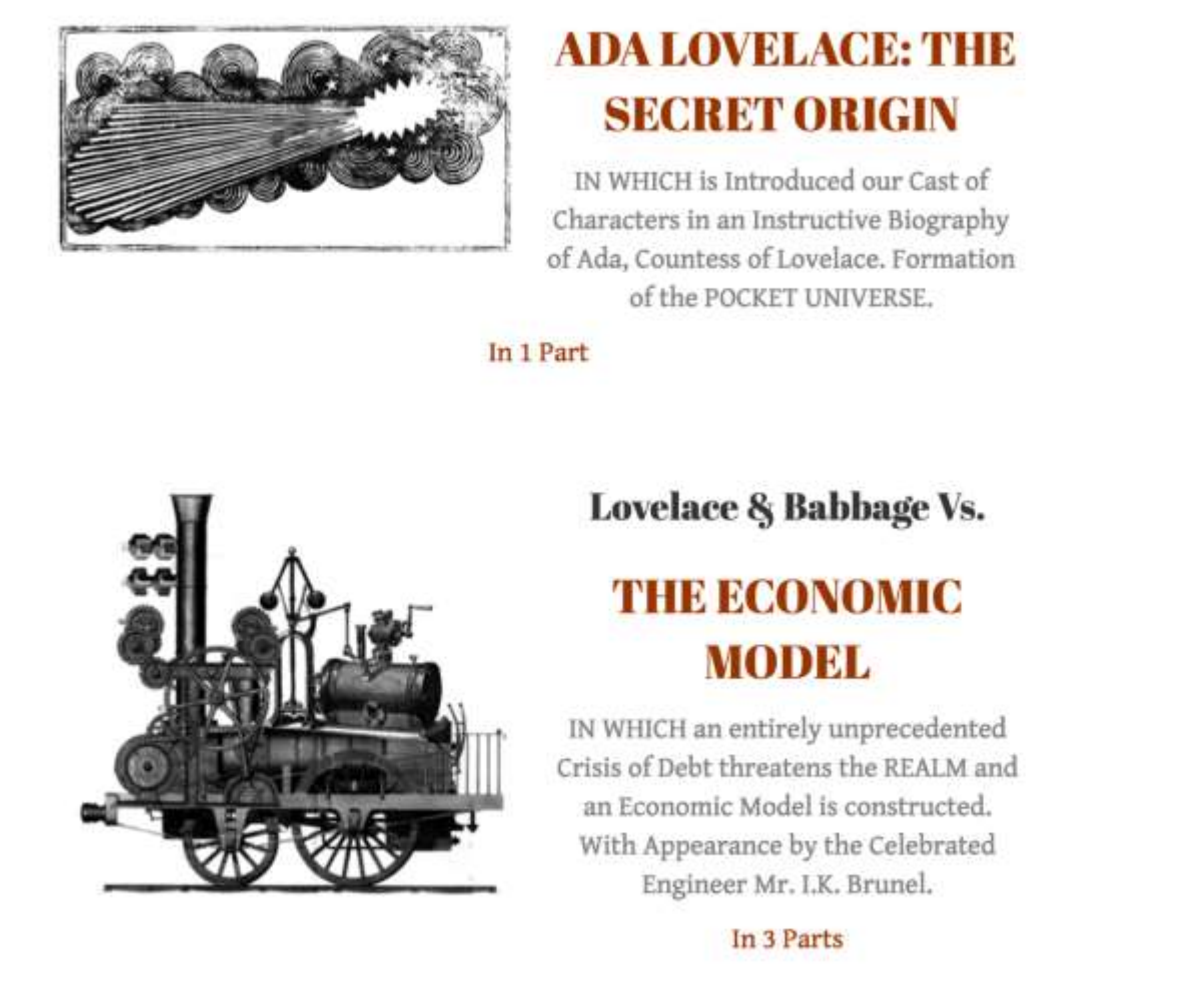
Sources: Paul Boutin; Brent Hailpern, associate director of computer science at IBM Research; The Retrocomputing Museum; Todd Proebsting, senior researcher at Microsoft; Gio Wiederhold, computer scientist, Stanford University

PREHISTORY

EUCLID, ~300 BC

GOTTFRIED LEIBNIZ, 1646-1716

CHARLES BABBAGE (1791-1871) & ADA LO



ADA LOVELACE: THE SECRET ORIGIN

IN WHICH is Introduced our Cast of Characters in an Instructive Biography of Ada, Countess of Lovelace. Formation of the POCKET UNIVERSE.

In 1 Part

Lovelace & Babbage Vs.

THE ECONOMIC MODEL

IN WHICH an entirely unprecedented Crisis of Debt threatens the REALM and an Economic Model is constructed. With Appearance by the Celebrated Engineer Mr. I.K. Brunel.

In 3 Parts

GOTTLOB FREGE (1848-1925) & GIUSEPPE PEANO (1858-1932)

DAVID HILBERT, 1862-1943

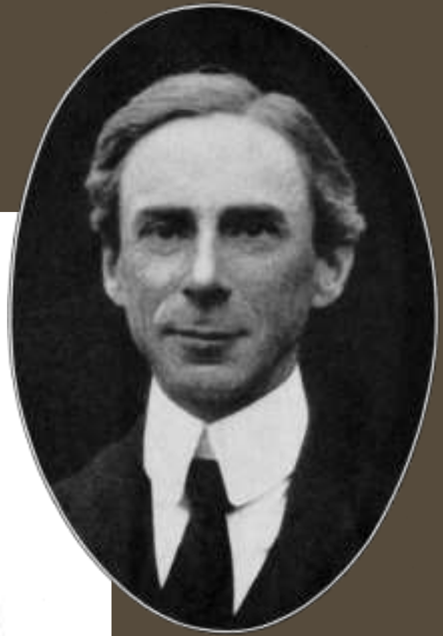
MOSES SCHÖNFINKEL, 1889-1942

HILBERT’S 23 QUESTIONS, 1900

1900: Hilbert poses his "23 questions" (now known as Hilbert's problems) at the Second International Congress of Mathematicians in Paris in 1900.

The second is simply, “Prove that the axioms of mathematics are consistent.”

PRINCIPIA MATHEMATICA



$$\ast 54 \cdot 43. \quad \vdash :. \alpha, \beta \in 1 . \supset : \alpha \cap \beta = \Lambda . \equiv . \alpha \cup \beta \in 2$$

Dem.

$$\vdash . \ast 54 \cdot 26 . \supset \vdash :. \alpha = \iota`x . \beta = \iota`y . \supset : \alpha \cup \beta \in 2 . \equiv . x \neq y .$$

$$[\ast 51 \cdot 231] \qquad \equiv . \iota`x \cap \iota`y = \Lambda .$$

$$[\ast 13 \cdot 12] \qquad \equiv . \alpha \cap \beta = \Lambda \qquad (1)$$

$$\vdash . (1) . \ast 11 \cdot 11 \cdot 35 . \supset$$

$$\vdash :. (\exists x, y) . \alpha = \iota`x . \beta = \iota`y . \supset : \alpha \cup \beta \in 2 . \equiv . \alpha \cap \beta = \Lambda \qquad (2)$$

$$\vdash . (2) . \ast 11 \cdot 54 . \ast 52 \cdot 1 . \supset \vdash . \text{Prop}$$

From this proposition it will follow, when arithmetical addition has been defined, that $1 + 1 = 2$.

HILBERT’S SECOND PROBLEM, REDUX. 1928.

Recasts his second problem into three questions:

1) Is mathematics complete?

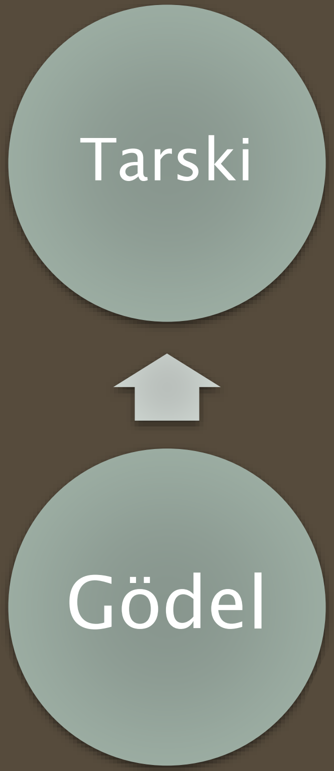
2) Is mathematics consistent?

3) Is mathematics decidable?

1930S

GÖDEL’S INCOMPLETENESS THEOREMS, 1931

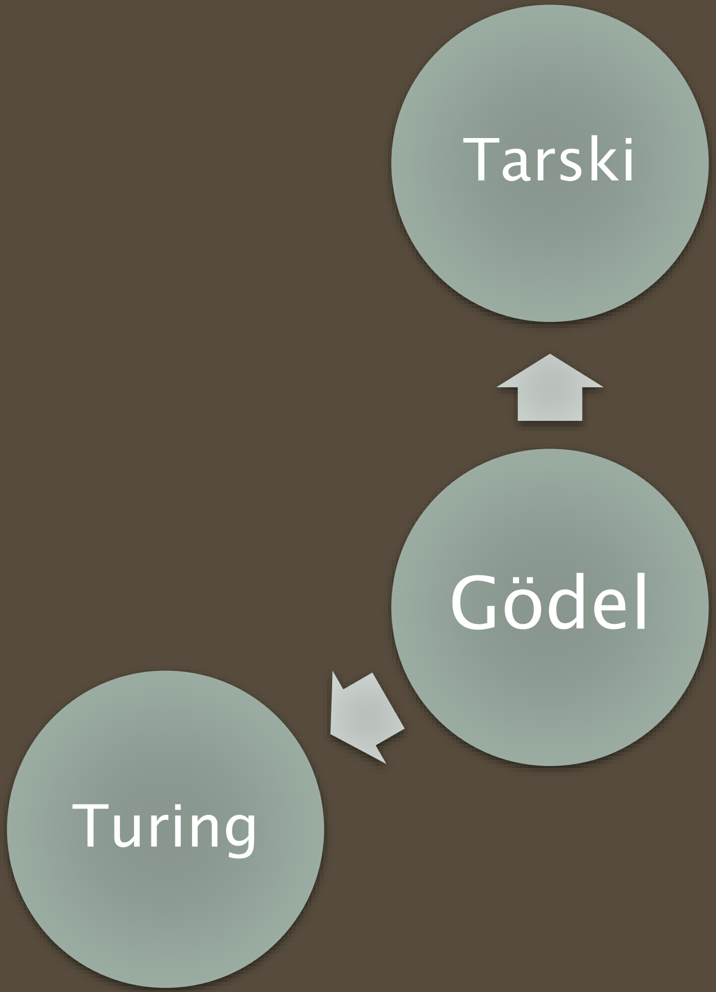
GÖDEL STARTED A FAD



UNDEFINABILITY THEOREM, 1936

Arithmetical truth cannot be defined in arithmetic.

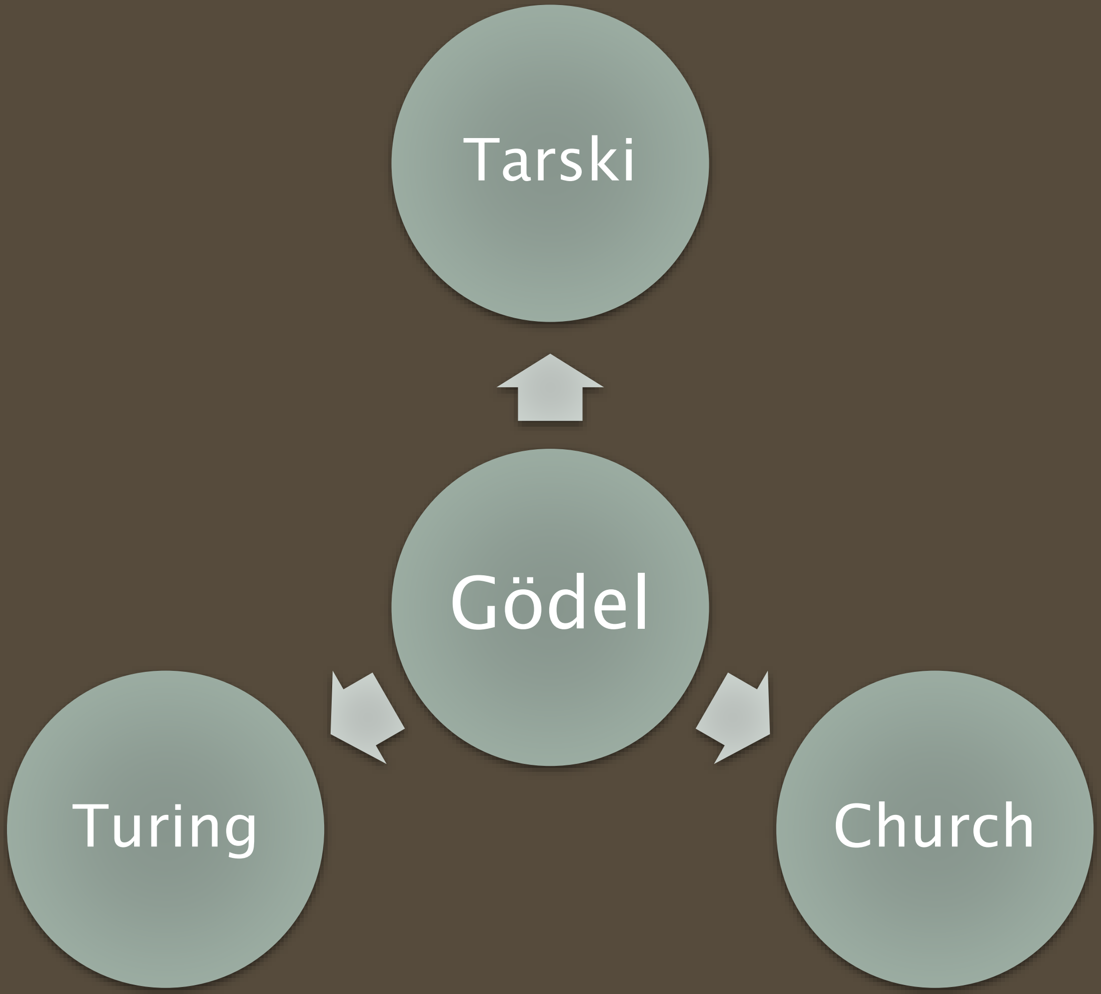
GÖDEL STARTED A FAD



ALAN TURING & HALTING PROBLEM, 1936

Can you determine, from a description of an arbitrary computer program and an input, whether the program will finish running or continue to run forever?

GÖDEL STARTED A FAD



ENTSCHEIDUNGSPROBLEM, 1936

Can an algorithm be derived that takes a statement of first-order logic, and determines whether that statement is valid in every structure satisfying the axioms?

-- or --

Can a given statement be proven from the axioms, using the rules of logic?

ALONZO CHURCH

1940S & 1950S

EARLY COMPUTERS

Z1, Z2, Z3: Z1 proposed 1935, Z3 delivered 1941

ENIAC: 1943 (proposed) 1946 (delivered)

EDVAC: 1944 (proposed) 1949 (delivered)

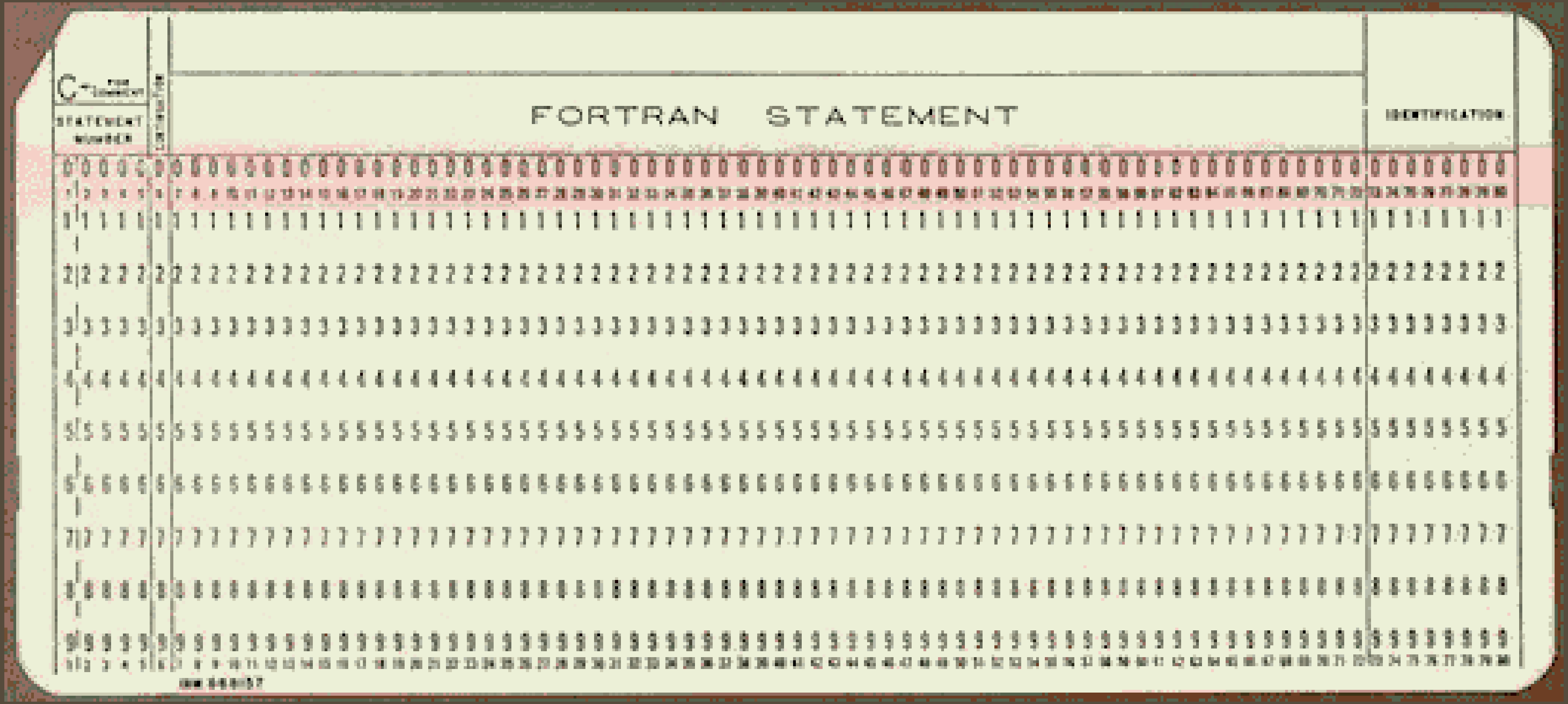
Manchester Baby: 1946 (proposed) 1948 (delivered)

EDSAC: 1946 (proposed) 1949 (delivered)

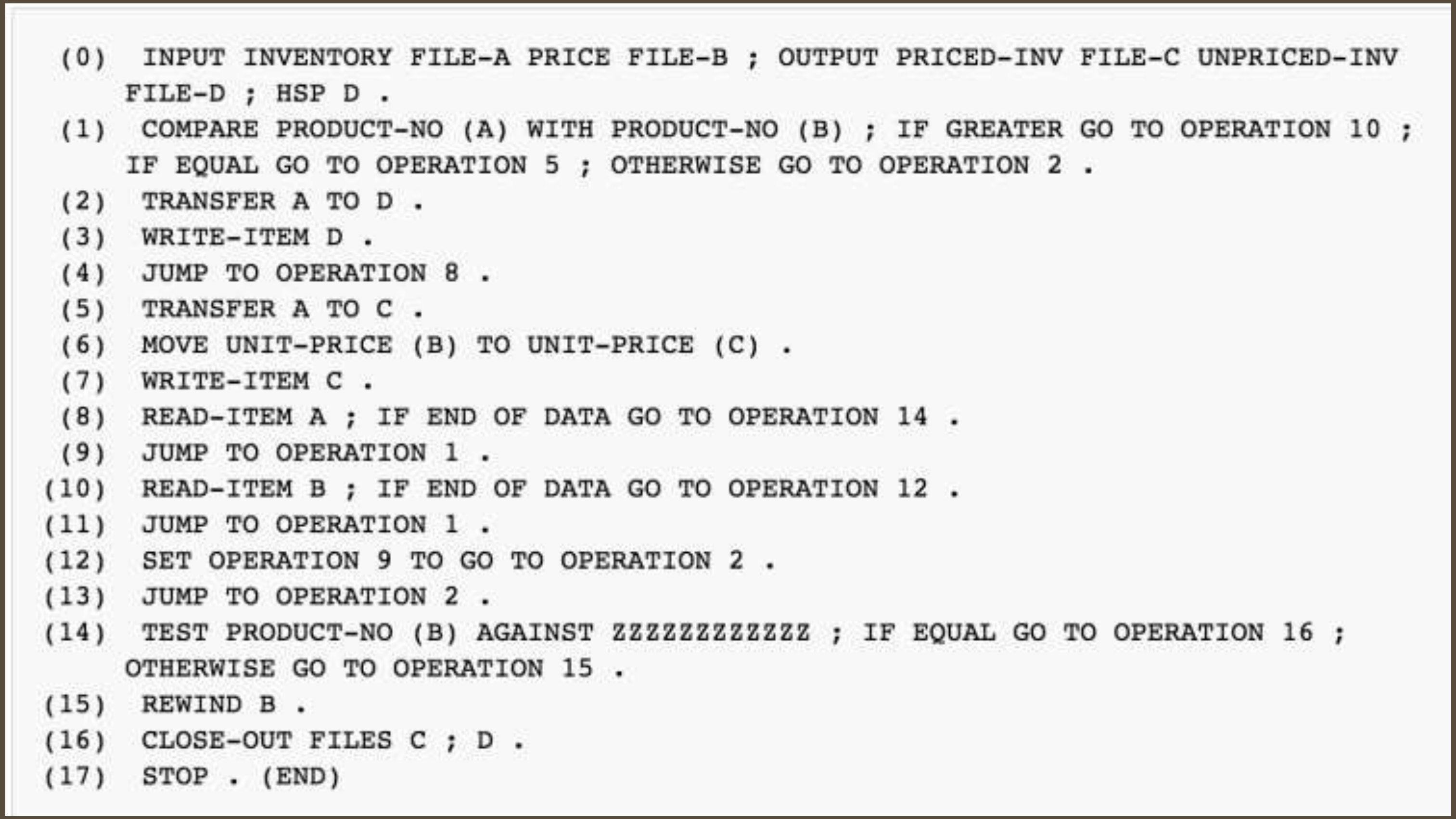
Pilot ACE: 1950 (delivered)

JOHN VON NEUMANN, 1903-1957

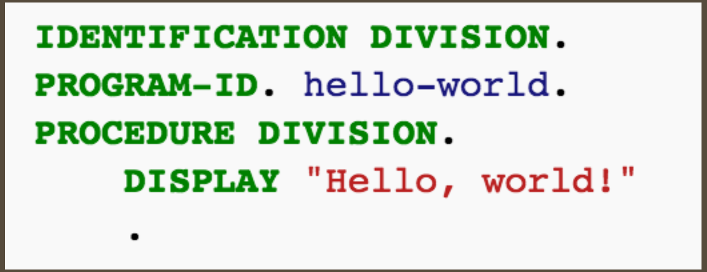
FORTRAN, 1954



FLOW-MATIC, 1955 & COBOL, 1959

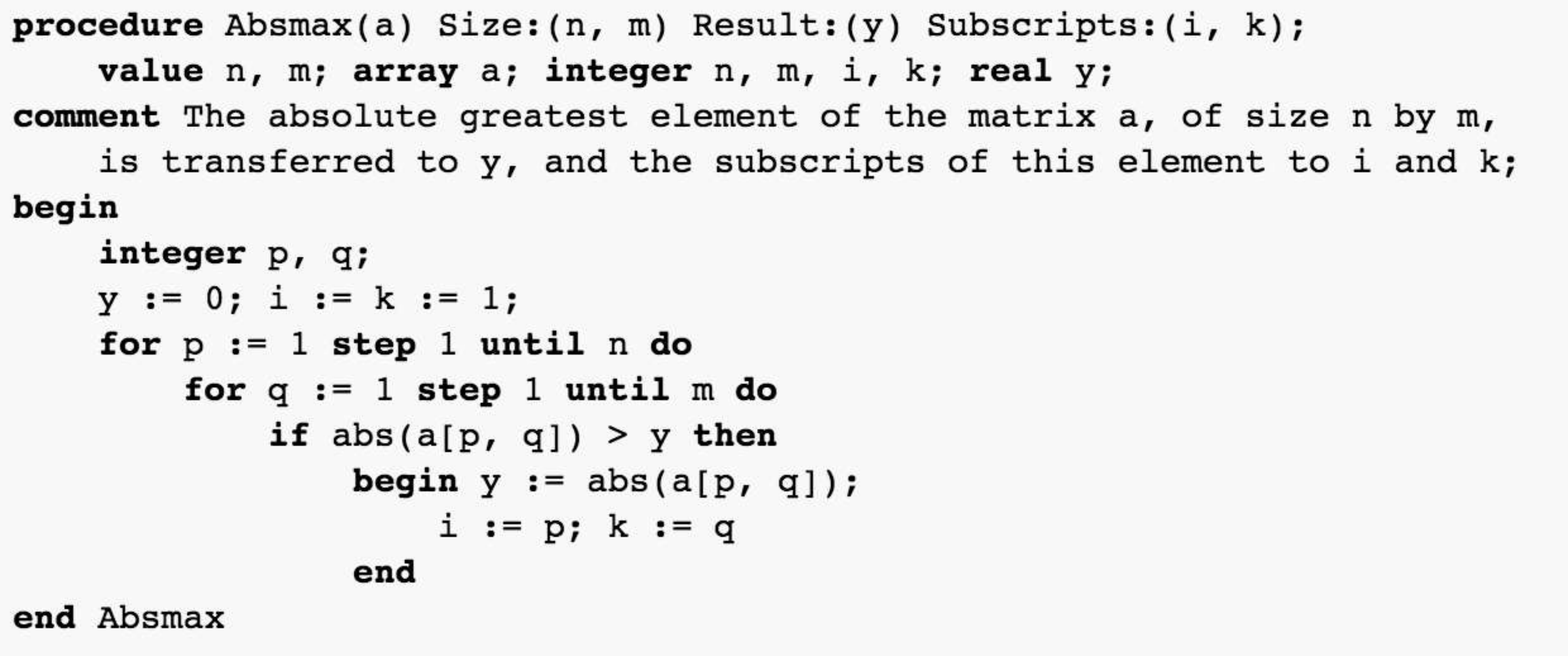


```
 (0)  INPUT INVENTORY FILE-A PRICE FILE-B ; OUTPUT PRICED-INV FILE-C UNPRICED-INV
     FILE-D ; HSP D .
 (1)  COMPARE PRODUCT-NO (A) WITH PRODUCT-NO (B) ; IF GREATER GO TO OPERATION 10 ;
     IF EQUAL GO TO OPERATION 5 ; OTHERWISE GO TO OPERATION 2 .
 (2)  TRANSFER A TO D .
 (3)  WRITE-ITEM D .
 (4)  JUMP TO OPERATION 8 .
 (5)  TRANSFER A TO C .
 (6)  MOVE UNIT-PRICE (B) TO UNIT-PRICE (C) .
 (7)  WRITE-ITEM C .
 (8)  READ-ITEM A ; IF END OF DATA GO TO OPERATION 14 .
 (9)  JUMP TO OPERATION 1 .
(10)  READ-ITEM B ; IF END OF DATA GO TO OPERATION 12 .
(11)  JUMP TO OPERATION 1 .
(12)  SET OPERATION 9 TO GO TO OPERATION 2 .
(13)  JUMP TO OPERATION 2 .
(14)  TEST PRODUCT-NO (B) AGAINST ZZZZZZZZZZZZ ; IF EQUAL GO TO OPERATION 16 ;
     OTHERWISE GO TO OPERATION 15 .
(15)  REWIND B .
(16)  CLOSE-OUT FILES C ; D .
(17)  STOP . (END)
```



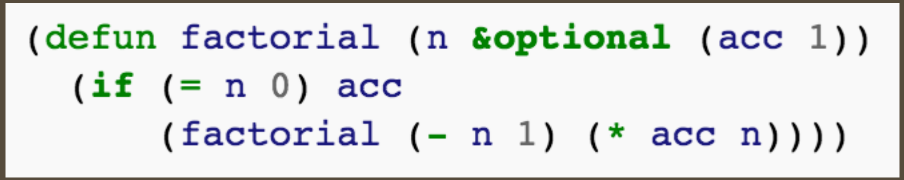
```
IDENTIFICATION DIVISION.
PROGRAM-ID. hello-world.
PROCEDURE DIVISION.
    DISPLAY "Hello, world!"
    .
```

ALGOL, 1958

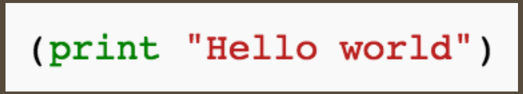


```
procedure Absmax(a) Size:(n, m) Result:(y) Subscripts:(i, k);
    value n, m; array a; integer n, m, i, k; real y;
comment The absolute greatest element of the matrix a, of size n by m,
    is transferred to y, and the subscripts of this element to i and k;
begin
    integer p, q;
    y := 0; i := k := 1;
    for p := 1 step 1 until n do
        for q := 1 step 1 until m do
            if abs(a[p, q]) > y then
                begin y := abs(a[p, q]);
                    i := p; k := q
                end
end Absmax
```

LISP, 1958



```
(defun factorial (n &optional (acc 1))
  (if (= n 0) acc
      (factorial (- n 1) (* acc n))))
```



```
(print "Hello world")
```

"Lisp is the medium of choice for people who enjoy free style and flexibility."

– Gerald Jay Sussman (introduction to [Friedman, 1987], p. ix)

Lisp first had: tree data structures, automatic storage management, dynamic typing, conditionals, higher-order functions, recursion, and the self-hosting compiler. (First? REPL)

MERVYN PRAGNELL’S LOGIC STUDY GROUP, LATE 1950S

The sessions were held illicitly after-hours at Birkbeck College (University of London), without the knowledge or permission of the college authorities. Pragnell knew a lab tech with a key that would let them in.

Members included: Christopher Strachey, Peter Landin, Rod Burstall, Dana Scott, Robin Milner (and more)...

LANGUAGE HISTORY CHART

First letter of each name has been aligned with the approximate date on which work began.

THIS TYPE STYLE indicates languages of major importance, because of their wide usage or technical significance.

THIS TYPE STYLE indicates languages of moderate importance.

THIS TYPE STYLE is used for all other languages.

Parentheses were used to indicate alternate names, or the later addition of the sequence number "1."

━━━━ indicates that the second language is a direct extension of the first.

———— indicates that the second language is an approximate extension of the first, i.e., very similar to the first, but not completely upward compatible.

— — — — indicates strong influence; sometimes the second language is "like, or in the style of" the first.

- - - - - - - indicates an approximate subset

Each of the following marks is associated with the language above or to its left:

● indicates preliminary or informal specifications or manual

■ indicates a public manual, or formal publication via technical paper, or public presentation

▲ release for usage outside development group

General Comments

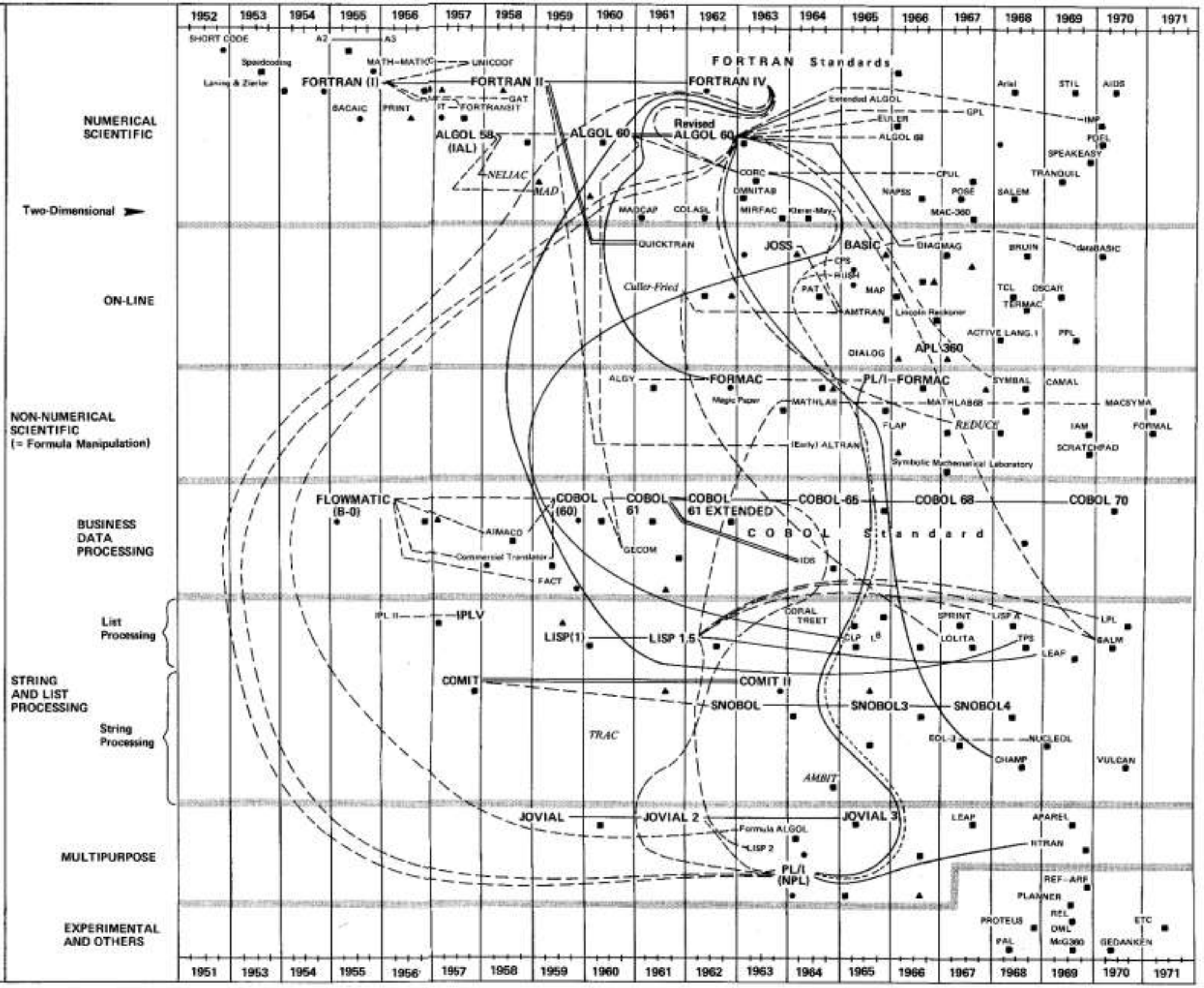
This chart represents only the personal opinions of the author as far as value judgments are involved, and the author's best estimate in many cases as far as dates are involved. The indications of the start of the work are the most questionable.

The information for languages in 1971 is based solely on those listed in "Roster of Programming Languages - 1971," Computers and Automation, Vol. 20, No. 6B (June 1971), pp. 6-13.

In most cases, dialects with differing names have been omitted. This has the unfortunate effect of appearing to minimize the importance of some languages which spawned numerous versions under differing names (e.g., JOSS).

Languages for specialized application areas (e.g., simulation, machine tool control, civil engineering, systems programming) have not been included because of space considerations. This explains the absence of such obvious languages as APT, GPSS, SIMSCRIPT, COGO, BLISS.

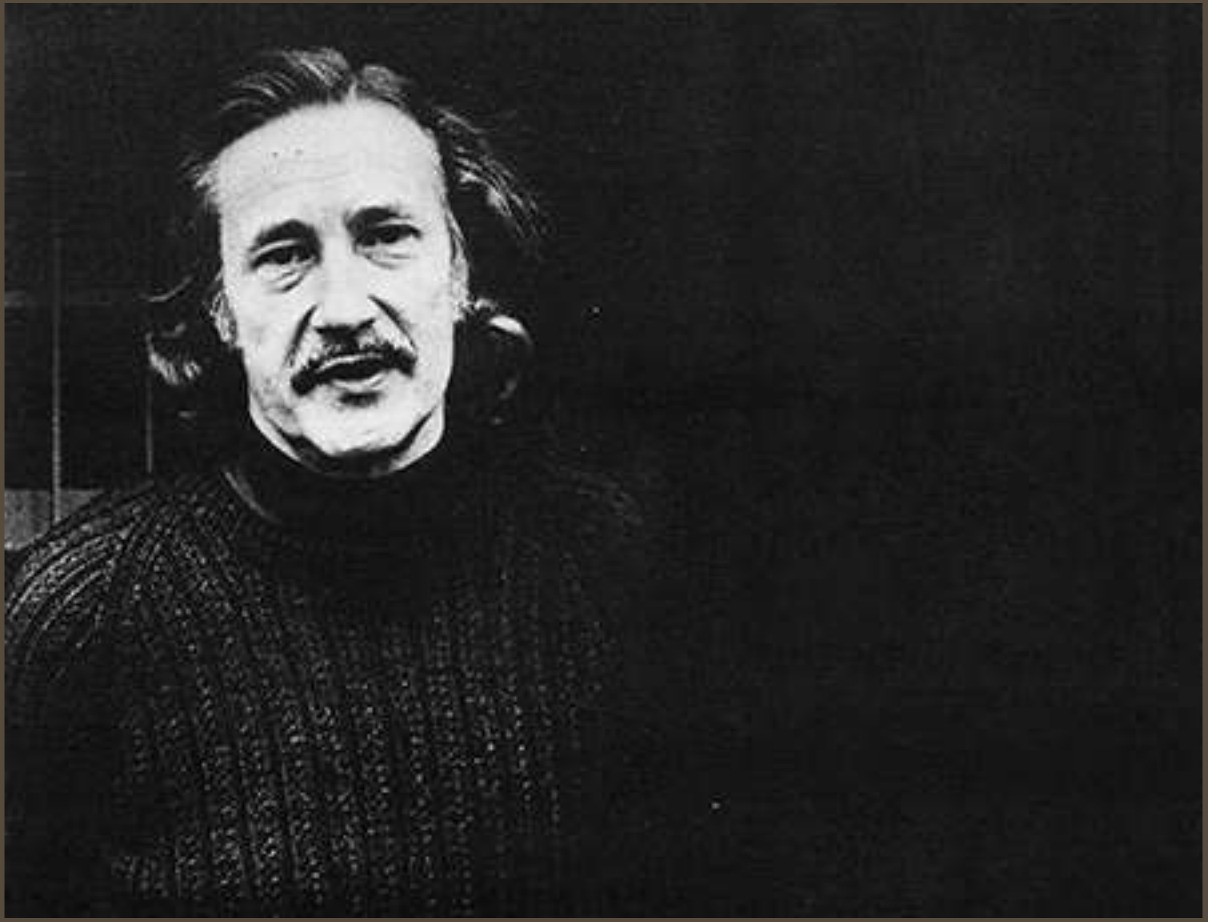
Acknowledgment: The idea for such a chart in such a format came from the one by Christopher J. Shaw entitled "Milestones in Computer Programming" and included with the [ACM Los Angeles Chapter] SIGPLAN notices, February 1965.



"Programming Languages: History and Future"
by Jean E. Sammet
Communications of the ACM, Vol. 15, July 1972

1960S

CHRISTOPHER STRACHEY, 1916-1975



PETER LANDIN, 1930 - 2009

PETER LANDIN’S PAPERS

Mechanical evolution of expressions (1964).

A correspondence between ALGOL-60 and Church's lambda-notation (1965).

The next 700 programming languages (1966).

Programs and their proofs: an algebraic approach (1968). With Rod Burstall.

ISWIM, 1966

“If you See What I Mean”

ISWIM is an imperative language with a functional core, consisting of a syntactic sugaring of lambda calculus, plus mutable variables and assignment and a powerful control mechanism—the J operator. Because it is based on the lambda calculus, ISWIM has higher order functions and lexically scoped variables.

PETER LANDIN’S CONTRIBUTIONS

Term “syntactic sugar”

Functional programming languages

Domain-specific languages

Graph reduction

Sharing

Strictness analysis

Where expressions

Disentangling nested applications into flat where expressions

Closures

First-class continuations

Algebraic data types

Streams

Connection between streams and coroutines

SECD machine

Significant whitespace (the off-side rule)

Delayed evaluation

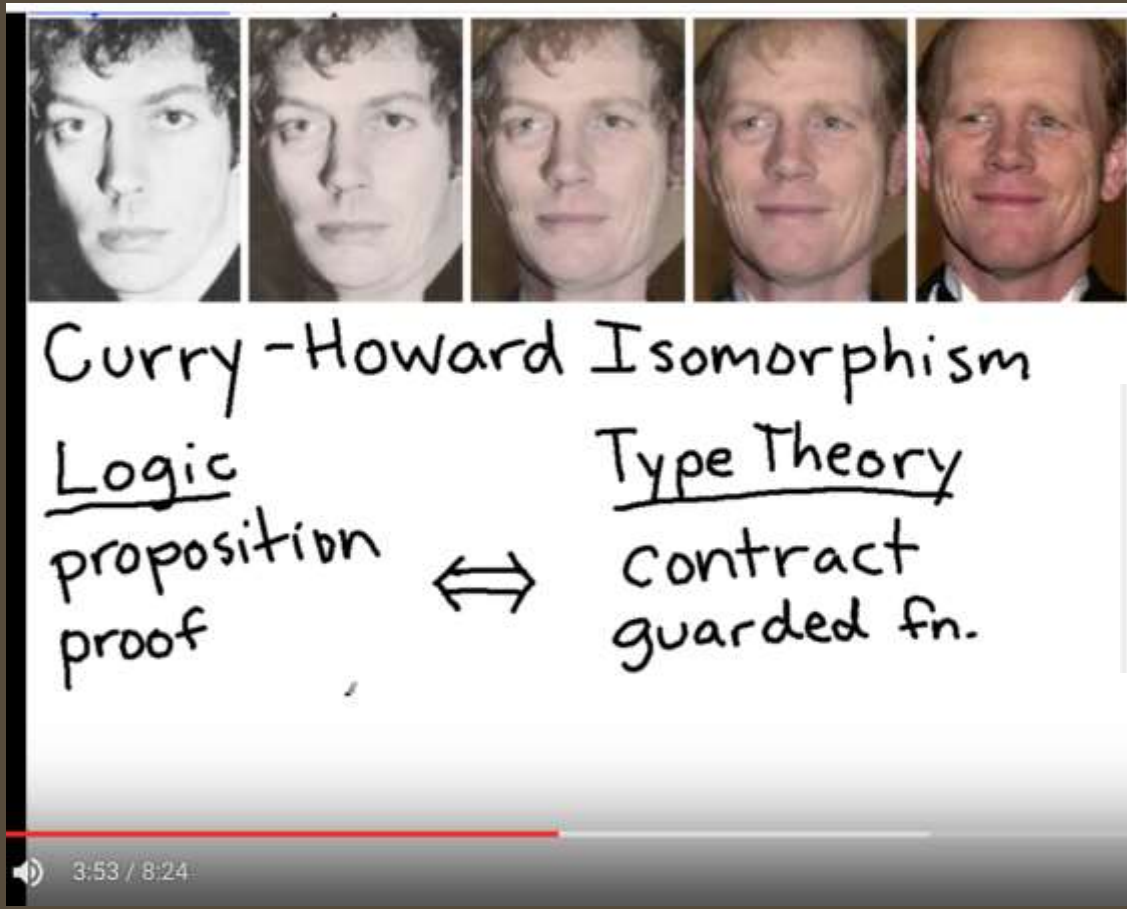
Partial evaluation

Circularity to implement recursion

J Operator

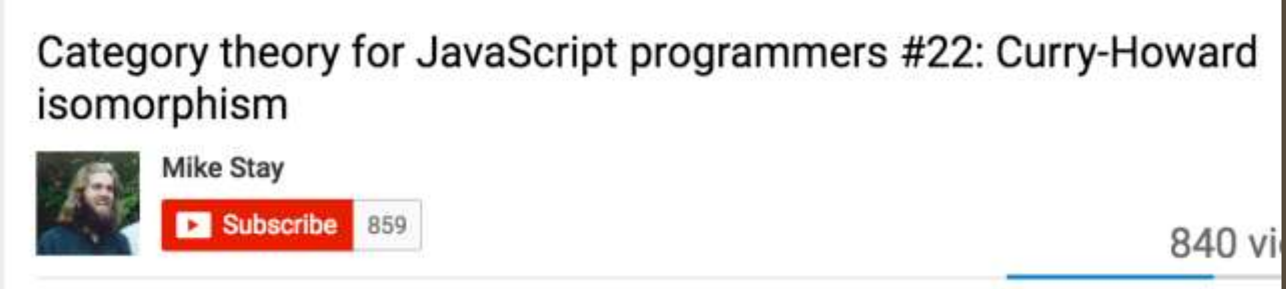
HASKELL CURRY, 1900-1982

CURRY-HOWARD ISOMORPHISM



Curry-Howard Isomorphism

Logic		Type Theory
proposition	⇔	contract
proof		guarded fn.



Category theory for JavaScript programmers #22: Curry-Howard isomorphism

Mike Stay

1970S

THEOREM PROVERS (TO TODAY)

LCF

HOL series (HOL88, HOL90, HOL98, HOL4, HOL Light, ProofPower, HOL Zero)

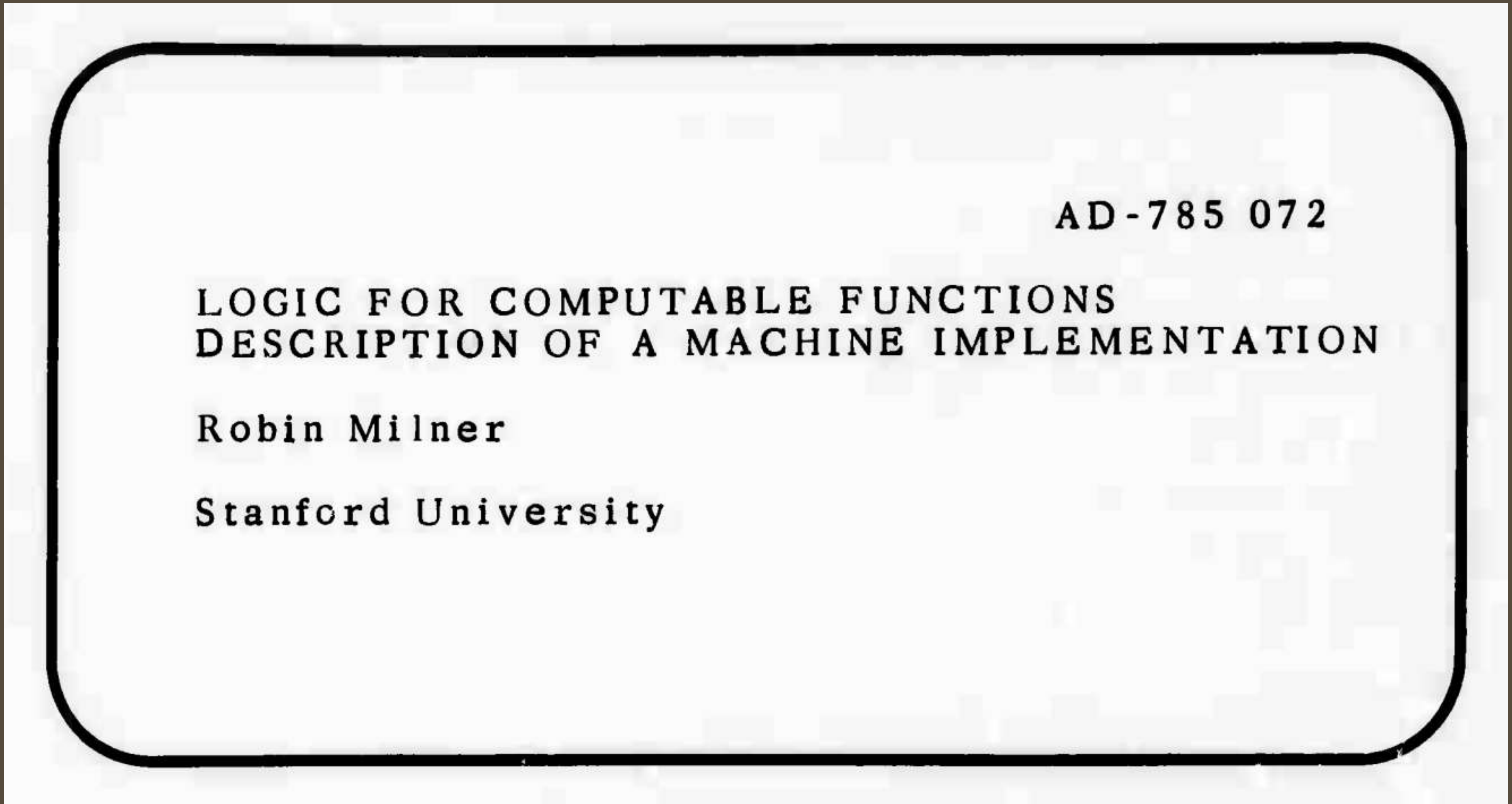
F*

Coq

Isabelle

Agda

LCF, 1972



AD-785 072

LOGIC FOR COMPUTABLE FUNCTIONS
DESCRIPTION OF A MACHINE IMPLEMENTATION

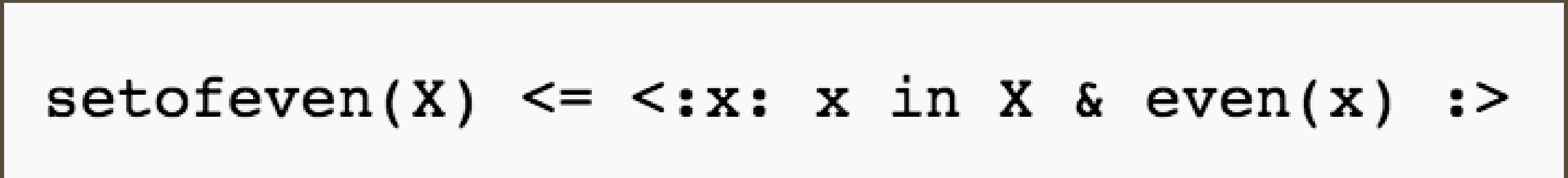
Robin Milner

Stanford University

ROBIN MILNER, 1934-2010

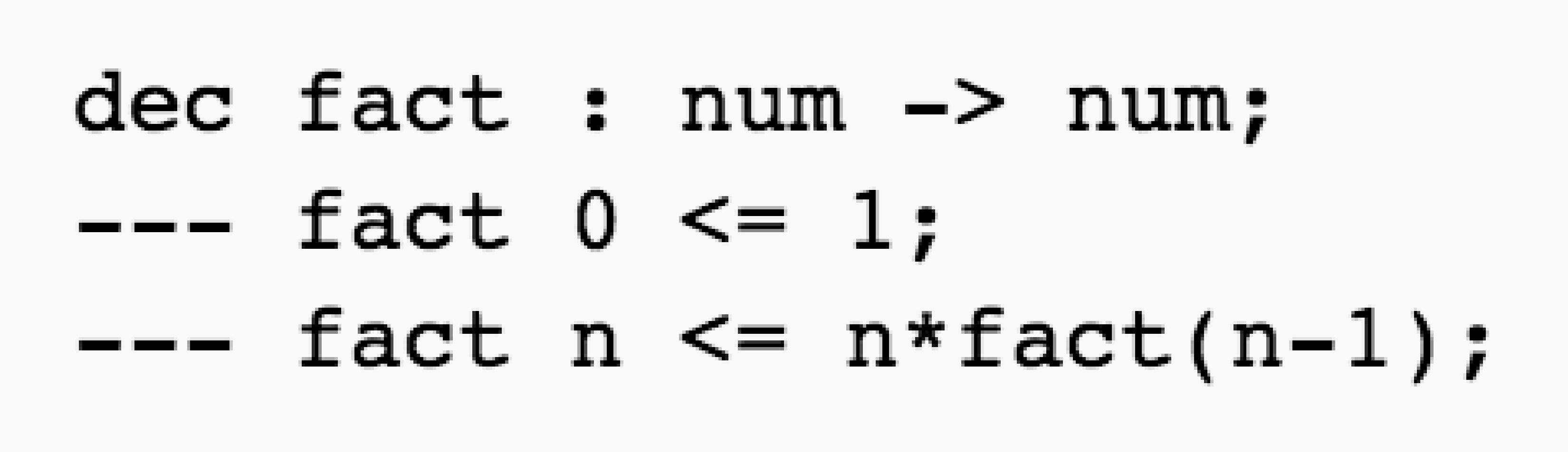
“The idea of a machine proving theorems in logic, and the idea of using logic to understand what a machine was doing... This double relationship began to inspire me because it was clearly not very simple.“

NPL, 1977



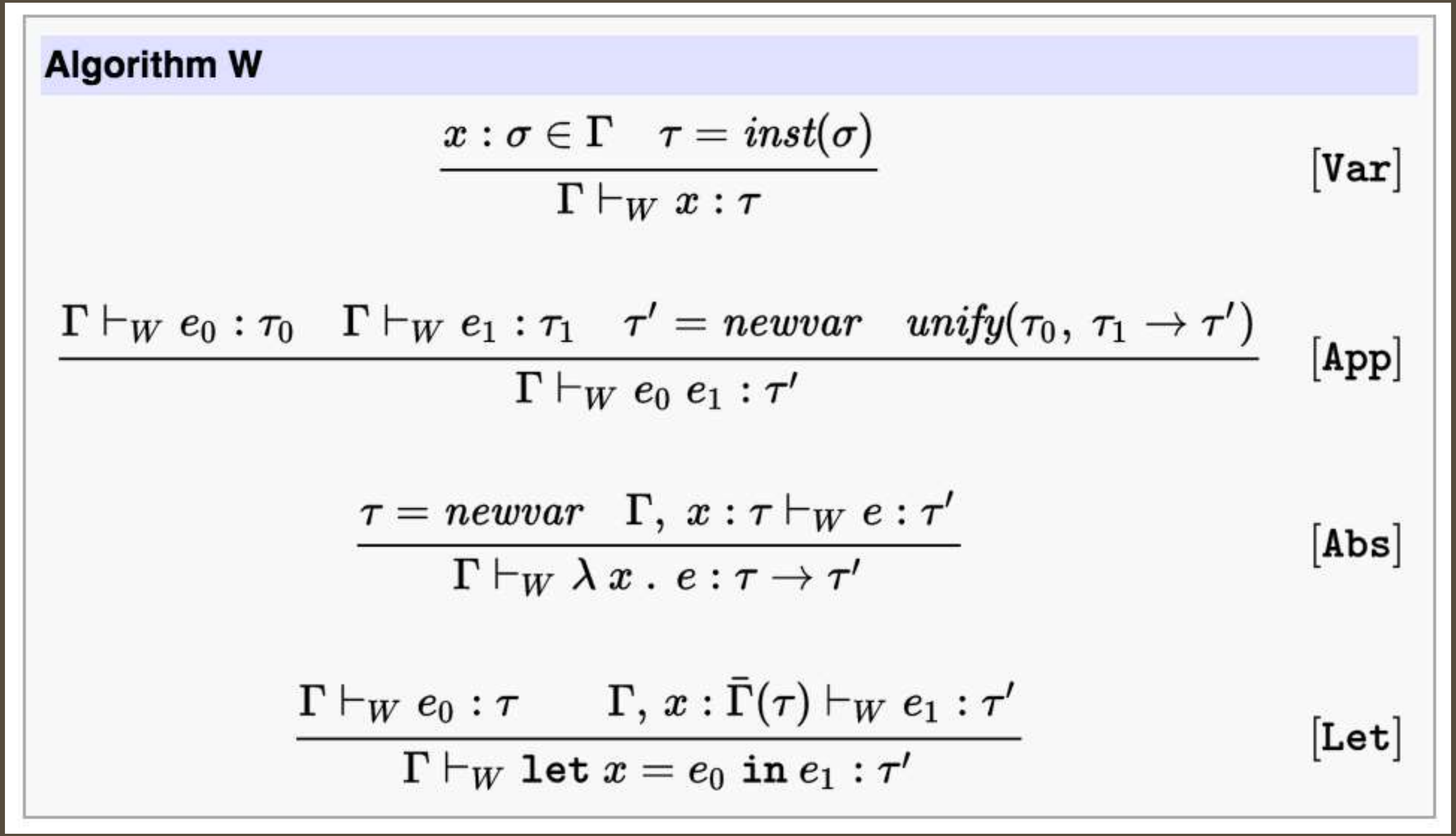
```
setofeven(X) <= <:x: x in X & even(x) :>
```

HOPE, 1977



```
dec fact : num -> num;
--- fact 0 <= 1;
--- fact n <= n*fact(n-1);
```

HINDLEY-MILNER TYPE INFERENCE, 1978



Algorithm W

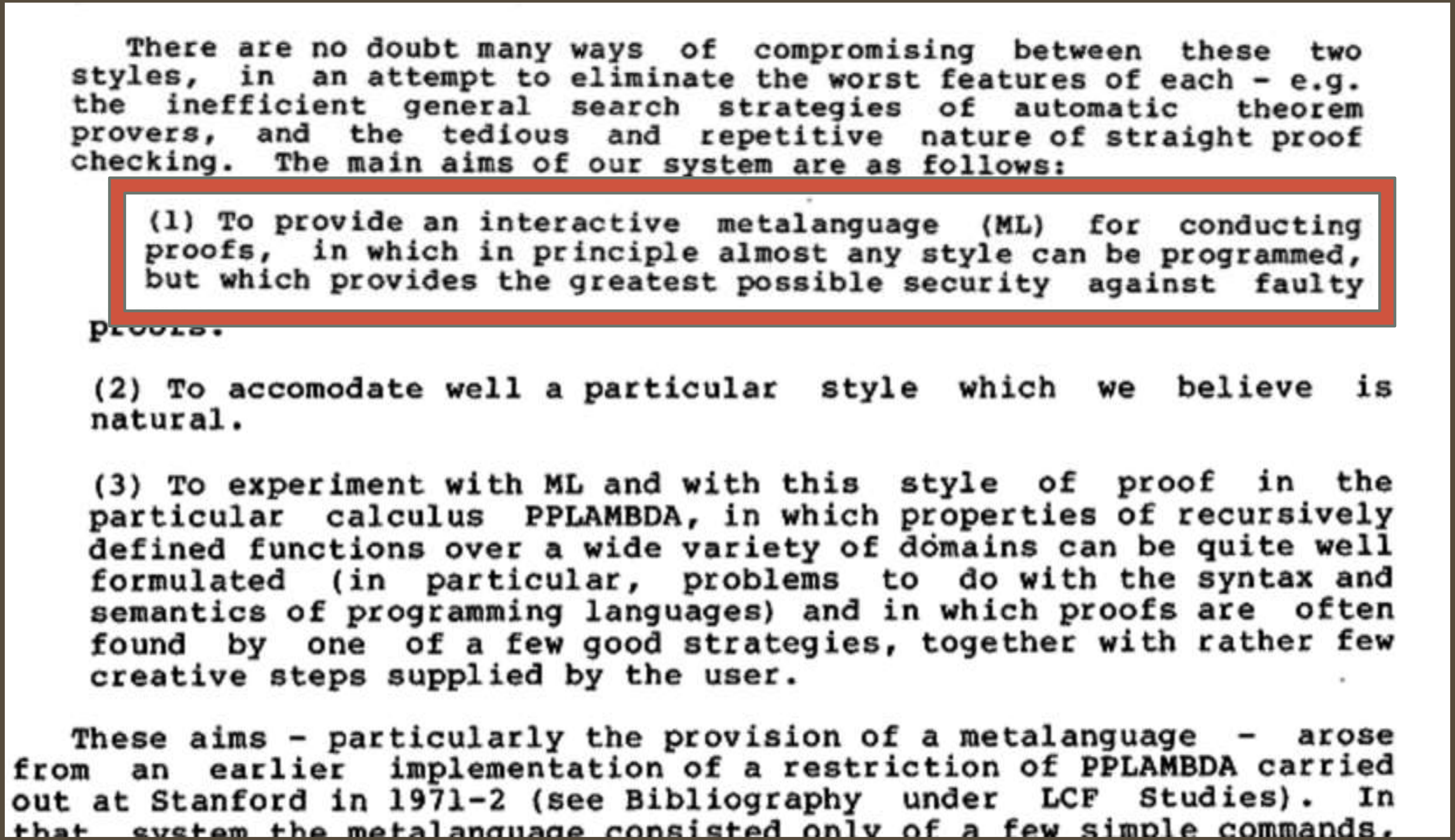
$$\frac{x : \sigma \in \Gamma \quad \tau = \mathit{inst}(\sigma)}{\Gamma \vdash_W x : \tau} \quad [\mathtt{Var}]$$

$$\frac{\Gamma \vdash_W e_0 : \tau_0 \quad \Gamma \vdash_W e_1 : \tau_1 \quad \tau' = \mathit{newvar} \quad \mathit{unify}(\tau_0,\ \tau_1 \to \tau')}{\Gamma \vdash_W e_0\ e_1 : \tau'} \quad [\mathtt{App}]$$

$$\frac{\tau = \mathit{newvar} \quad \Gamma,\ x : \tau \vdash_W e : \tau'}{\Gamma \vdash_W \lambda\, x\ .\ e : \tau \to \tau'} \quad [\mathtt{Abs}]$$

$$\frac{\Gamma \vdash_W e_0 : \tau \qquad \Gamma,\ x : \bar{\Gamma}(\tau) \vdash_W e_1 : \tau'}{\Gamma \vdash_W \mathtt{let}\ x = e_0\ \mathtt{in}\ e_1 : \tau'} \quad [\mathtt{Let}]$$

EDINBURGH LCF, 1979



There are no doubt many ways of compromising between these two styles, in an attempt to eliminate the worst features of each - e.g. the inefficient general search strategies of automatic theorem provers, and the tedious and repetitive nature of straight proof checking. The main aims of our system are as follows:

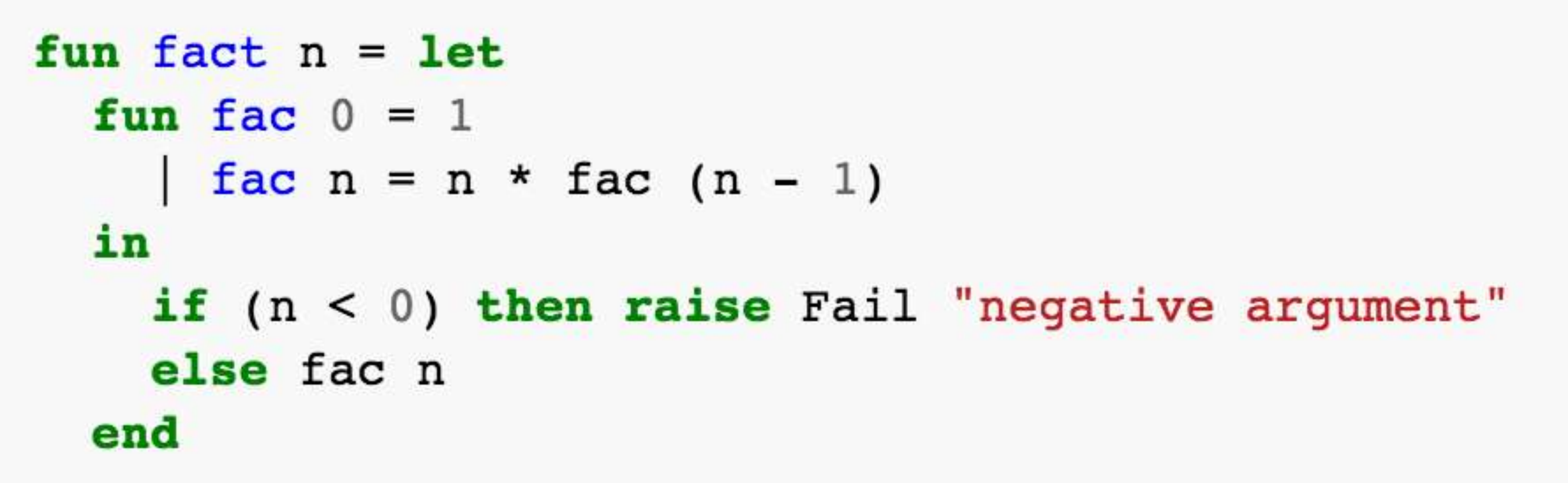
(1) To provide an interactive metalanguage (ML) for conducting proofs, in which in principle almost any style can be programmed, but which provides the greatest possible security against faulty proofs.

(2) To accomodate well a particular style which we believe is natural.

(3) To experiment with ML and with this style of proof in the particular calculus PPLAMBDA, in which properties of recursively defined functions over a wide variety of domains can be quite well formulated (in particular, problems to do with the syntax and semantics of programming languages) and in which proofs are often found by one of a few good strategies, together with rather few creative steps supplied by the user.

These aims - particularly the provision of a metalanguage - arose from an earlier implementation of a restriction of PPLAMBDA carried out at Stanford in 1971-2 (see Bibliography under LCF Studies). In that system the metalanguage consisted only of a few simple commands.

ML, 1979



```
fun fact n = let
  fun fac 0 = 1
    | fac n = n * fac (n - 1)
  in
    if (n < 0) then raise Fail "negative argument"
    else fac n
  end
```

1980S

LUCA CARDELLI & VAX ML, 1981

New features:

- New labelled record and union types

- The ref type for mutable values

- Declaration combinators for building compound declarations

- Modules

- Stream I/O w/ bi-directional streams.

Importance:

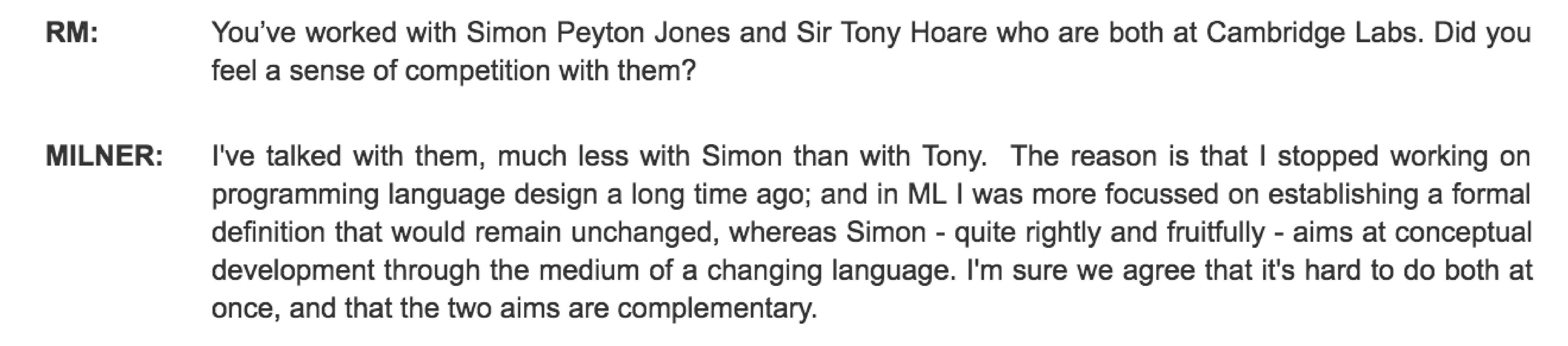
Demonstrates viability of ML a general language with an efficient implementation.

Creates incentive to control proliferation of dialects

An immediate precursor of Standard ML

A testbed for early experiments with Standard ML design

STANDARD ML DESIGN STARTS 1983.



RM: You've worked with Simon Peyton Jones and Sir Tony Hoare who are both at Cambridge Labs. Did you feel a sense of competition with them?

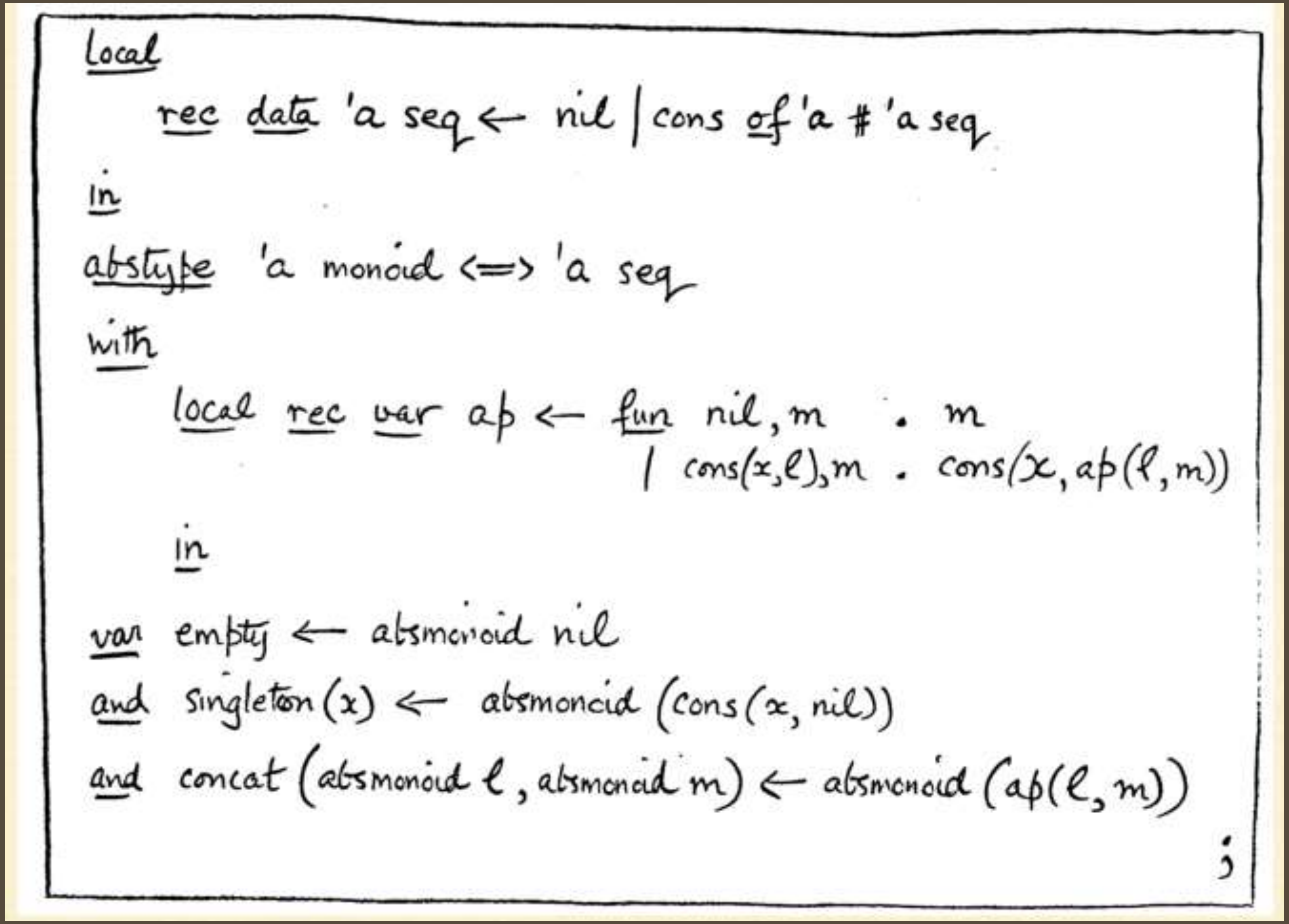
MILNER: I've talked with them, much less with Simon than with Tony. The reason is that I stopped working on programming language design a long time ago; and in ML I was more focussed on establishing a formal definition that would remain unchanged, whereas Simon - quite rightly and fruitfully - aims at conceptual development through the medium of a changing language. I'm sure we agree that it's hard to do both at once, and that the two aims are complementary.

First participants:

Rod Burstall, Luca Cardelli, Guy Cousineau ,Mike Gordon, David MacQueen, Robin Milner, Kevin Mitchell, Alan Mycroft, Larry Paulson, David Rydeheard, Don Sannella, John Scott, Brian Monahan, Stefan Sokolowski, Gerard Huet, Peter Mosses, David Schmidt

1990S

STANDARD ML, 1990



```
local
    rec data 'a seq ← nil | cons of 'a # 'a seq
in
abstype 'a monoid <=> 'a seq
with
     local rec var ap ← fun nil, m        .  m
                           | cons(x,ℓ),m  .  cons(x, ap(ℓ,m))

     in

var empty ← absmonoid nil
and singleton(x) ← absmonoid (cons(x, nil))
and concat (absmonoid ℓ, absmonoid m) ← absmonoid (ap(ℓ, m))
                                                                  ;
```

OBJECT-ORIENTED PROGRAMMING GOES VIRAL

C++, 1983. C++ 2.0 1989.

New features in 2.0 included multiple inheritance, abstract classes, static member functions, const member functions, and protected members.

Java, initiated 1991. First version, 1995.

C# & .NET, first initiated, 1999. First released: 2002.

ML-2000 TALKS START, 1993

Should we add OO capabilities?

OTHER WILD MLS APPEAR

Standard ML of NJ

=> MLj

=> SML.NET

CAML, 1985

=> Caml Light, 1991

=> Caml Special Light, 1995

=> OCaml, 1996

=> F#, 2005

ANU ML, late 80s

Harlequin ML, 1996

Moscow ML, 1994

PolyML

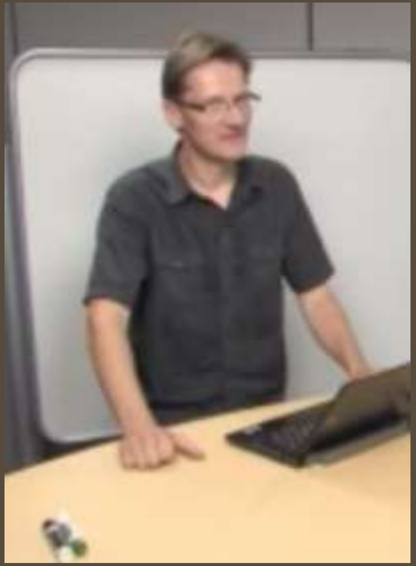
MLKit

Mlton

MLWorks

SML '97

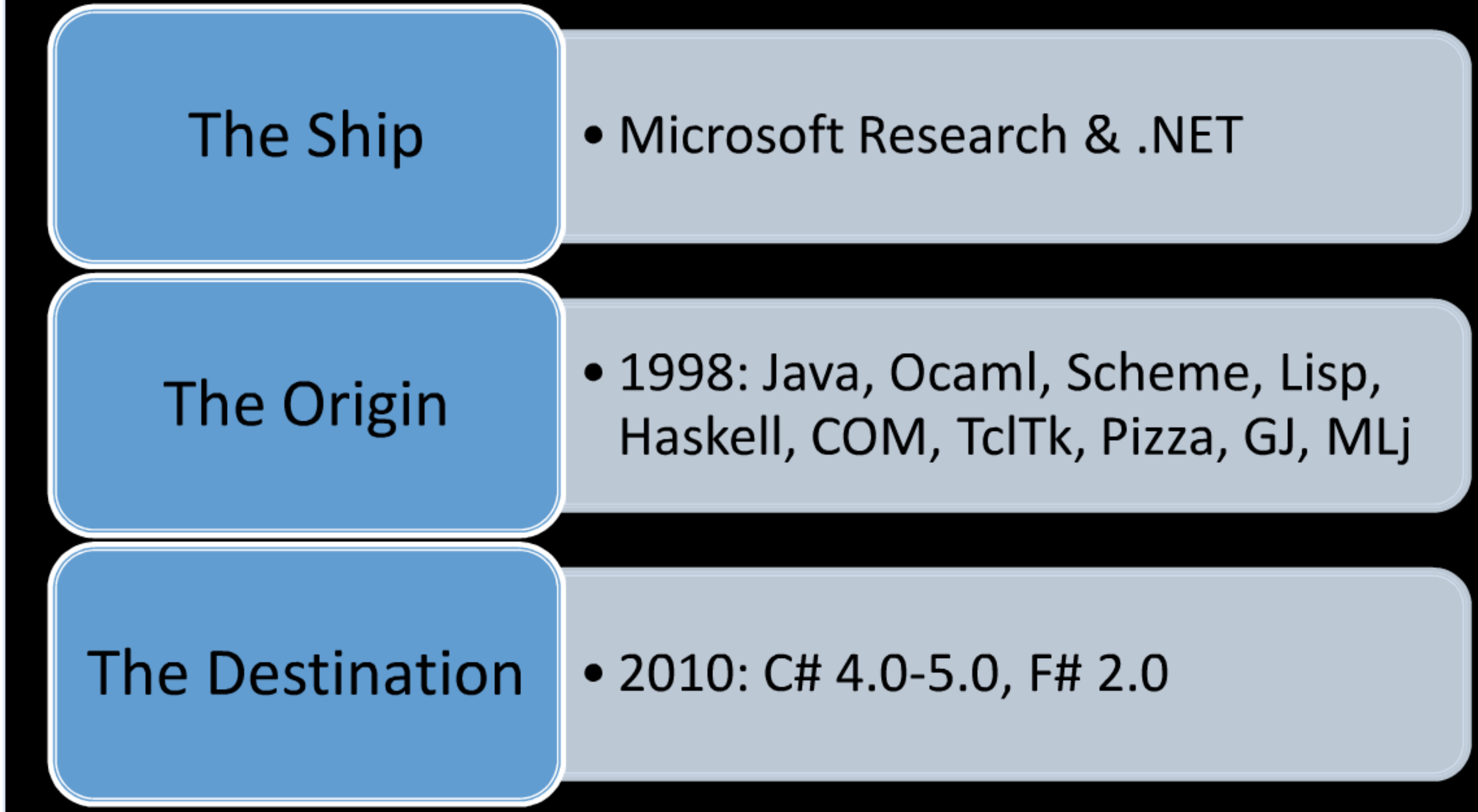
QML



2000S

DON SYME

My F# Journey to 2010



The Ship

- Microsoft Research & .NET

The Origin

- 1998: Java, Ocaml, Scheme, Lisp, Haskell, COM, TclTk, Pizza, GJ, MLj

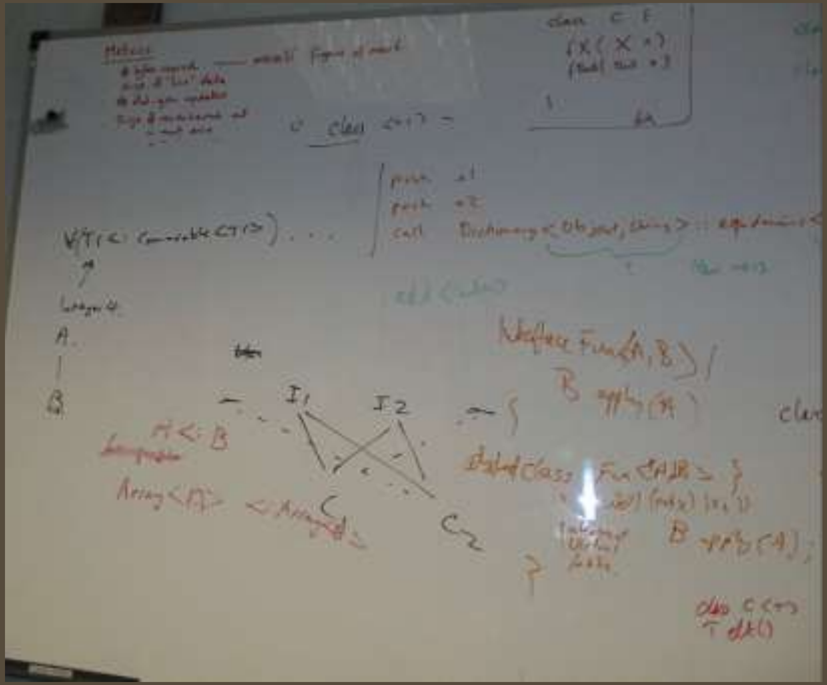
The Destination

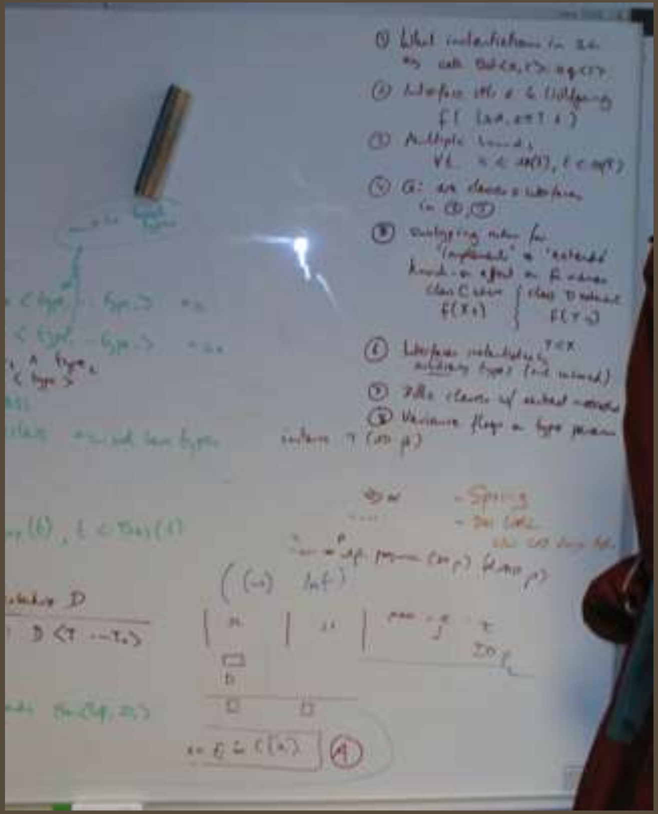
- 2010: C# 4.0-5.0, F# 2.0

> So much of what I loved about programming, was just missing from the experience of programming in Java. And that, in a sense, was a big driving motivation. And as C# 1.0 occurred, I was faced with a desperate situation: that I might actually have to do all my programming in either Java or C# 1.0. And I actually had to do personally do something about this at Microsoft, either by improving C# to the point that it was what I wanted to use, or by working on a new language.

Don Syme
F# History, Today, Tomorrow talk
SPLASH 2010

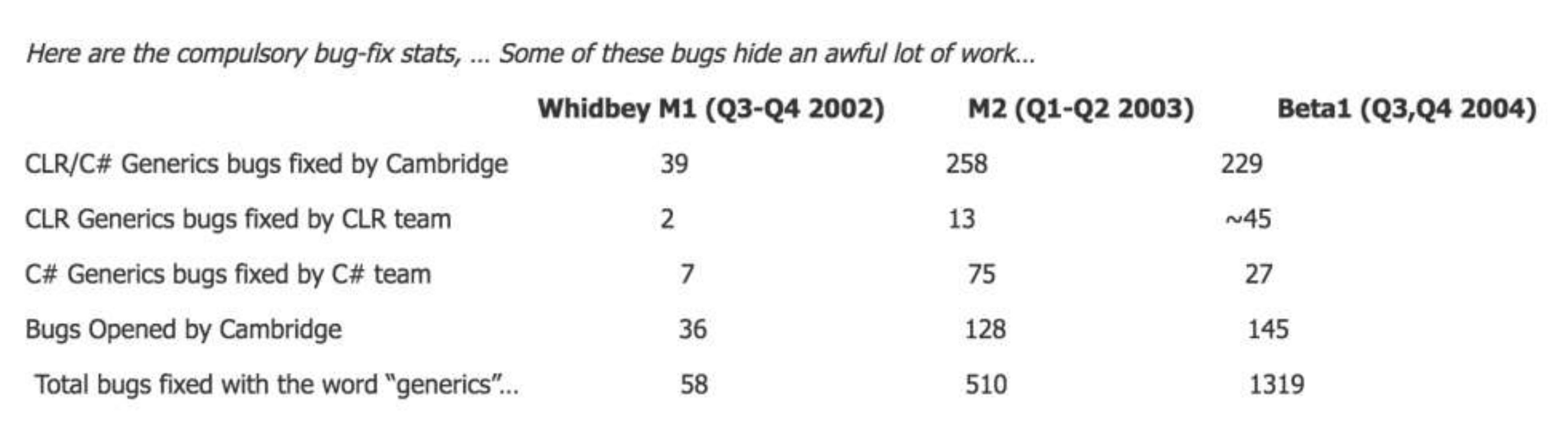
PUSH FOR .NET GENERICS MAKE IT IN TO C# 2.0, 1999-2004





Microsoft .net

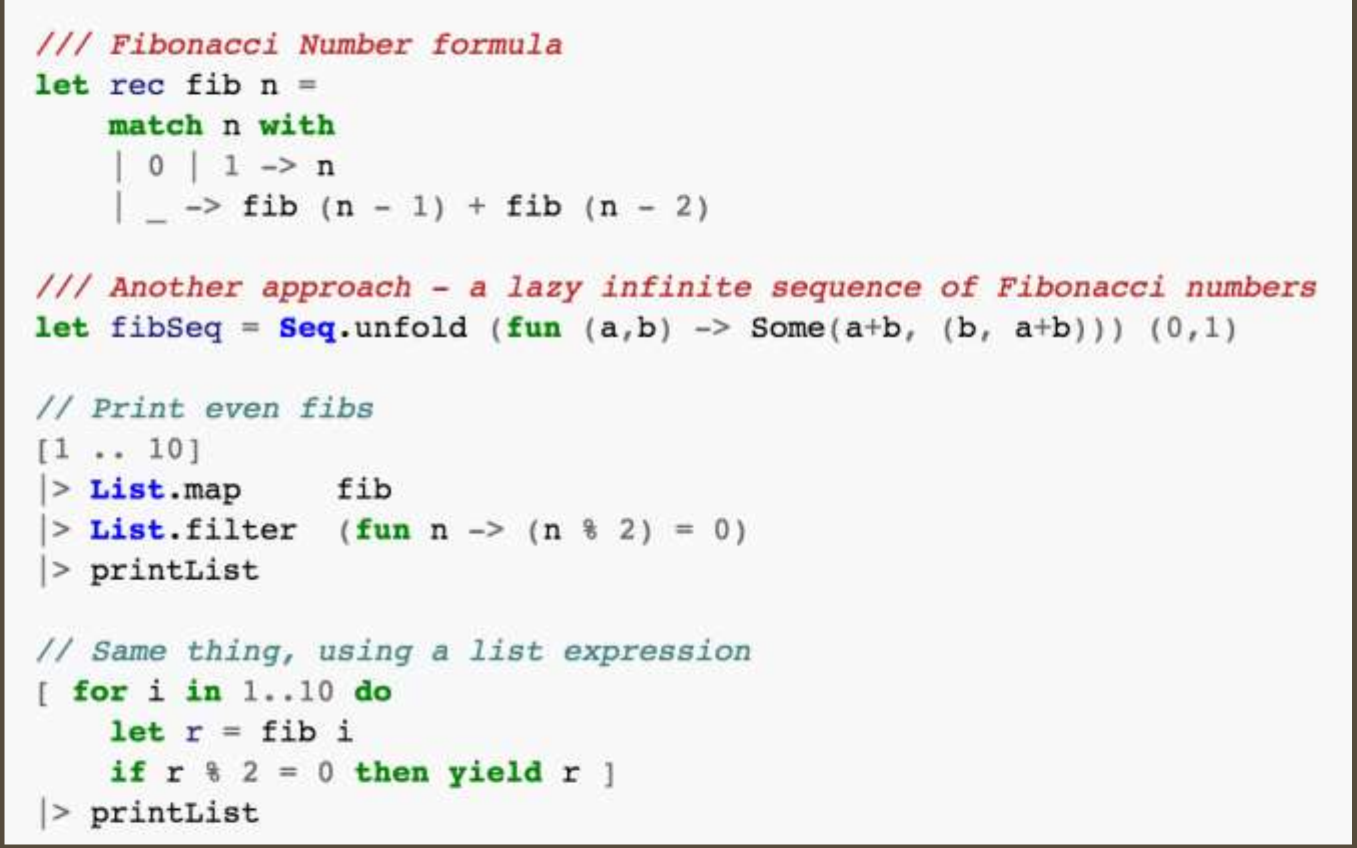
BUG FIXES FOR GENERICS, 2002-2004



Here are the compulsory bug-fix stats, … Some of these bugs hide an awful lot of work…

	Whidbey M1 (Q3-Q4 2002)	**M2 (Q1-Q2 2003)**	**Beta1 (Q3,Q4 2004)**
CLR/C# Generics bugs fixed by Cambridge	39	258	229
CLR Generics bugs fixed by CLR team	2	13	~45
C# Generics bugs fixed by C# team	7	75	27
Bugs Opened by Cambridge	36	128	145
Total bugs fixed with the word “generics”…	58	510	1319

IN WHICH WE FINALLY MEET THE HERO OF OUR STORY: F#



```
/// Fibonacci Number formula
let rec fib n =
    match n with
    | 0 | 1 -> n
    | _ -> fib (n - 1) + fib (n - 2)

/// Another approach - a lazy infinite sequence of Fibonacci numbers
let fibSeq = Seq.unfold (fun (a,b) -> Some(a+b, (b, a+b))) (0,1)

// Print even fibs
[1 .. 10]
|> List.map     fib
|> List.filter  (fun n -> (n % 2) = 0)
|> printList

// Same thing, using a list expression
[ for i in 1..10 do
    let r = fib i
    if r % 2 = 0 then yield r ]
|> printList
```

A FEW PAPERS RELATED TO F#

Tomas Petricek and Don Syme, The F# Computation Expression Zoo, in *Proceedings of Practical Aspects of Declarative Languages*, ACM, 2014.

Don Syme, Kenji Takeda, Keith Battocchi, Donna Malayeri, and Tomas Petricek, Themes in Information-Rich Functional Programming for Internet-Scale Data Sources, ACM, 24 January 2013.

Don Syme, Keith Battocchi, and Gordon Hodgenson, Creating a Type Provider (F#), Microsoft, 12 January 2013.

Don Syme, Adam Granicz, and Antonio Cisternino, Expert F# 3.0, Apress, 7 November 2012.

Don Syme, Keith Battocchi, Kenji Takeda, Donna Malayeri, Jomo Fisher, Jack Hu, Tao Liu, Brian McNamara, Daniel Quirk, Matteo Taveggia, Wonseok Chae, Uladzimir Matsveyeu, and Tomas Petricek, F#3.0 - Strongly-Typed Language Support for Internet-Scale Information Sources, no. MSR-TR-2012-101, 21 September 2012.

Tomas Petricek, Don Syme, and Alan Mycroft, Extending Monads with Pattern Matching, in*Proceedings of Haskell Symposium*, ACM, 2011.

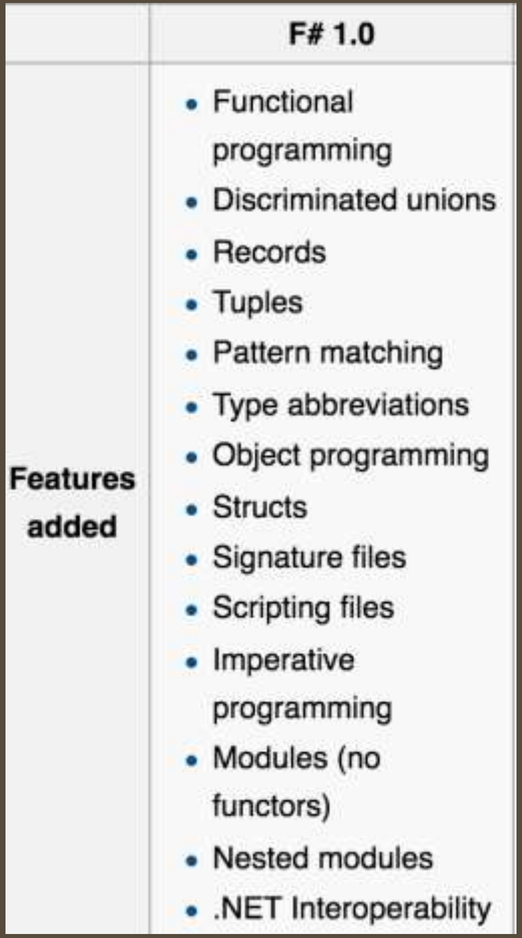
Tomas Petricek and Don Syme, Joinads: a retargetable control-flow construct for reactive, parallel and concurrent programming, in *Proceedings of Practical Aspects of Declarative Languages*, ACM, 2011.

Donald Syme, Tomas Petricek, and Dmitry Lomov, The F# Asynchronous Programming Model, in*In Proceedings of Principles and Applications of Declarative Languages, 2011*, ACM SIGPLAN, 2011.

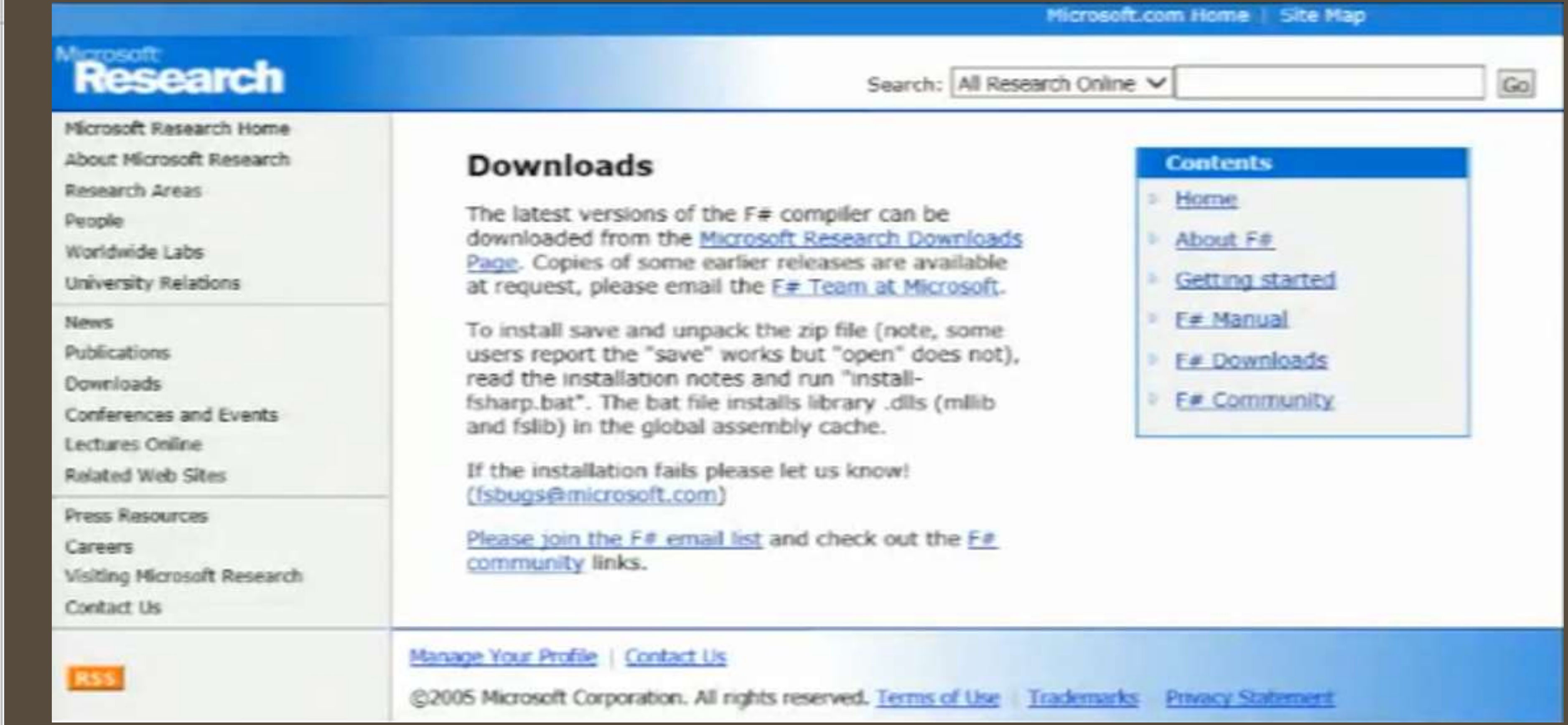
Tomas Petricek and Donald Syme, Collecting hollywood's garbage: avoiding space-leaks in composite events, in *ISMM '10 Proceedings of the 2010 international symposium on Memory management* , ACM SIGPLAN, 2010.

Don Syme, The F# Draft Language Specification, Microsoft, 1 February 2009.

F# 1.0, 2005



	F# 1.0
Features added	• Functional programming • Discriminated unions • Records • Tuples • Pattern matching • Type abbreviations • Object programming • Structs • Signature files • Scripting files • Imperative programming • Modules (no functors) • Nested modules • .NET Interoperability



Downloads

The latest versions of the F# compiler can be downloaded from the Microsoft Research Downloads Page. Copies of some earlier releases are available at request, please email the F# Team at Microsoft.

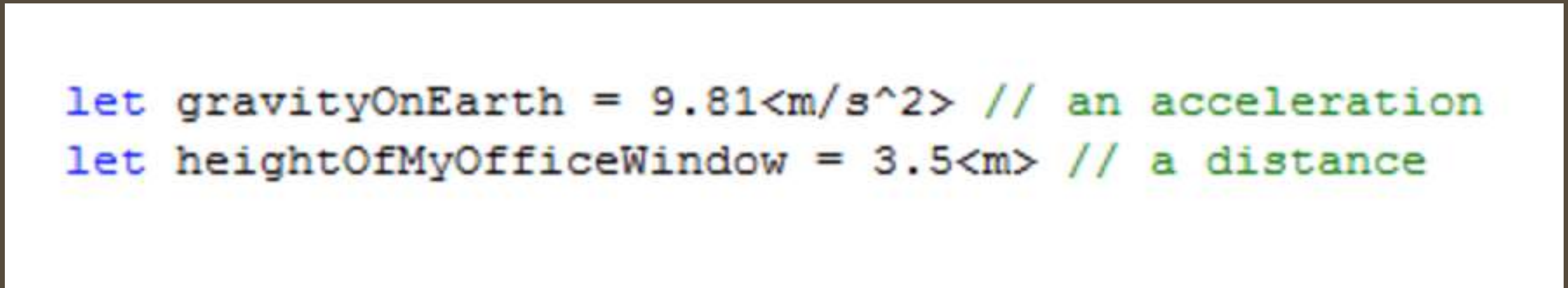
To install save and unpack the zip file (note, some users report the "save" works but "open" does not), read the installation notes and run "install-fsharp.bat". The bat file installs library .dlls (mllib and fslib) in the global assembly cache.

If the installation fails please let us know! (fsbugs@microsoft.com)

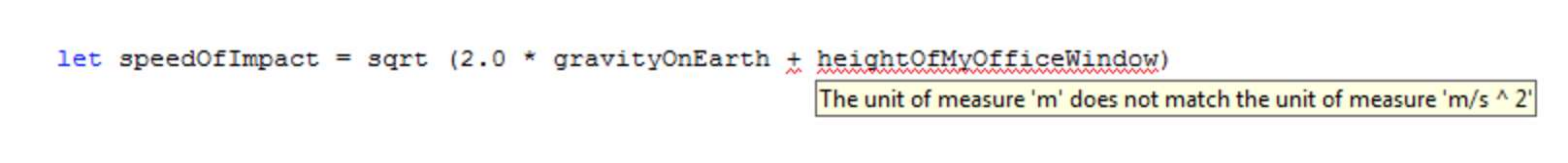
Please join the F# email list and check out the F# community links.

©2005 Microsoft Corporation. All rights reserved. Terms of Use | Trademarks | Privacy Statement

UNITS OF MEASURE, 2008



```
let gravityOnEarth = 9.81<m/s^2> // an acceleration
let heightOfMyOfficeWindow = 3.5<m> // a distance
```



```
let speedOfImpact = sqrt (2.0 * gravityOnEarth + heightOfMyOfficeWindow)
```

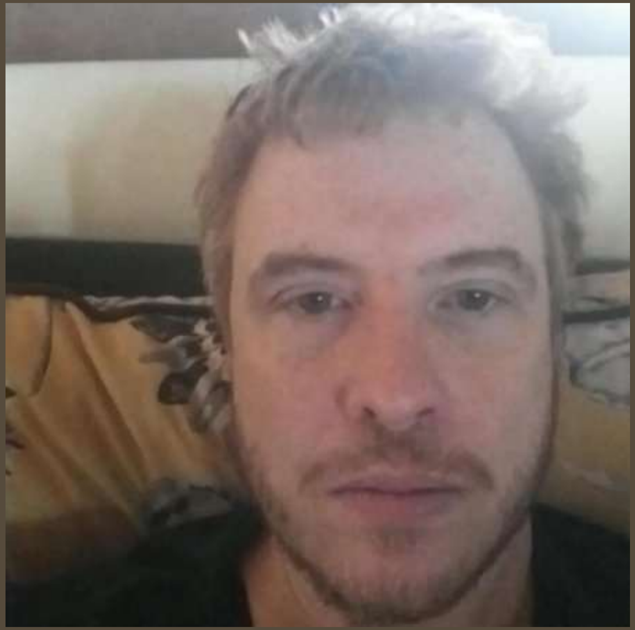
The unit of measure 'm' does not match the unit of measure 'm/s ^ 2'

TYPE PROVIDERS, 2009

”Let’s #r a database!”

2010: Jomo Fisher prototypes the Excel and Freebase type providers

2011: Decided to make it a feature of F# 3.0. Was called “awesome typing.”

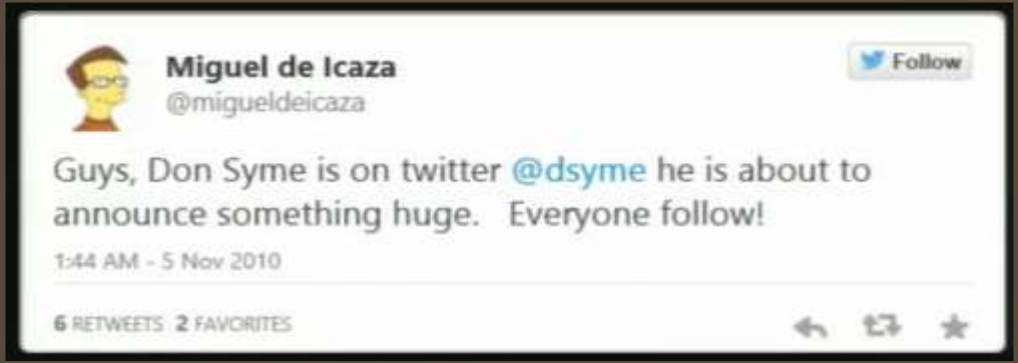


2010S

F# 2.0 INCLUDED IN VISUAL STUDIO 2010!

+

F# IS OPEN SOURCED! 2010



Miguel de Icaza
@migueldeicaza

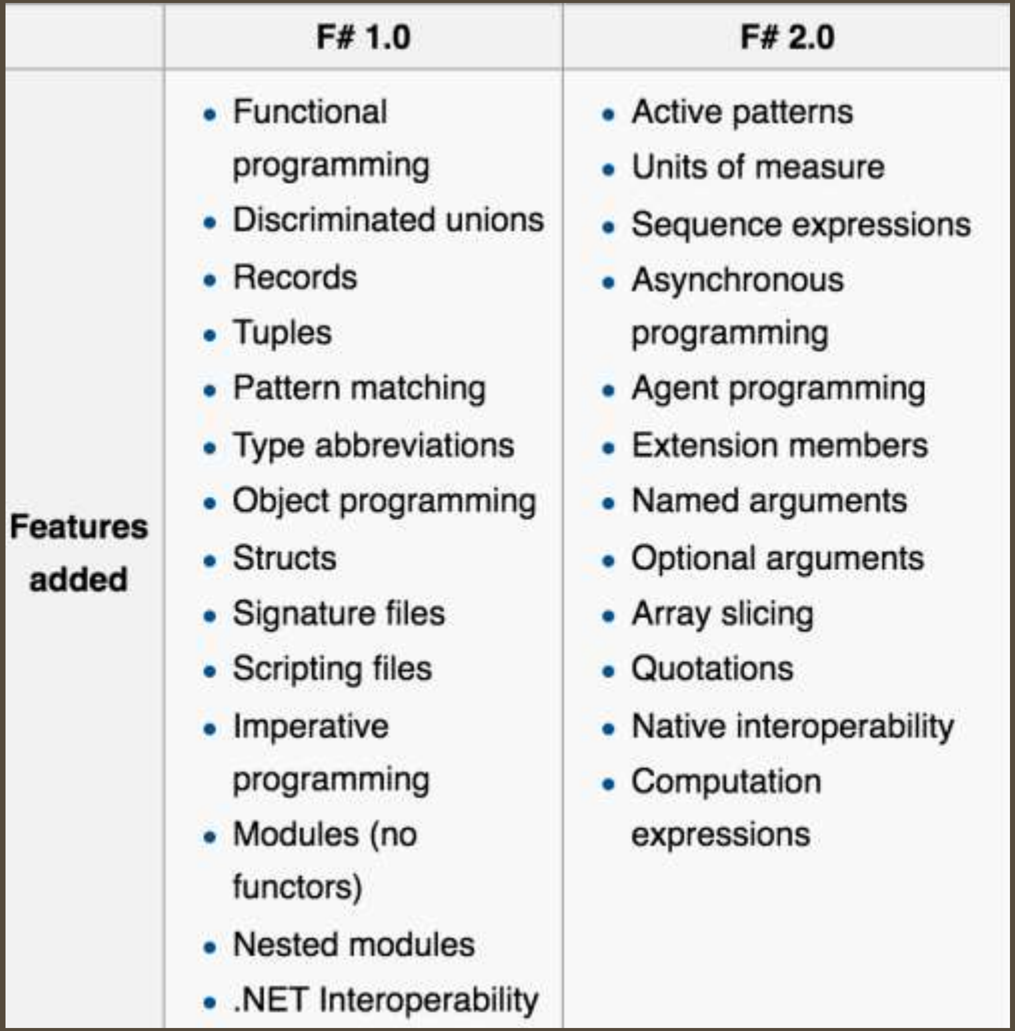
Guys, Don Syme is on twitter @dsyme he is about to announce something huge. Everyone follow!

1:44 AM - 5 Nov 2010

6 RETWEETS 2 FAVORITES

Uses the Apache license

F# 2.0, 2010



	F# 1.0	F# 2.0
Features added	• Functional programming • Discriminated unions • Records • Tuples • Pattern matching • Type abbreviations • Object programming • Structs • Signature files • Scripting files • Imperative programming • Modules (no functors) • Nested modules • .NET Interoperability	• Active patterns • Units of measure • Sequence expressions • Asynchronous programming • Agent programming • Extension members • Named arguments • Optional arguments • Array slicing • Quotations • Native interoperability • Computation expressions

FSHARP.ORG, 201

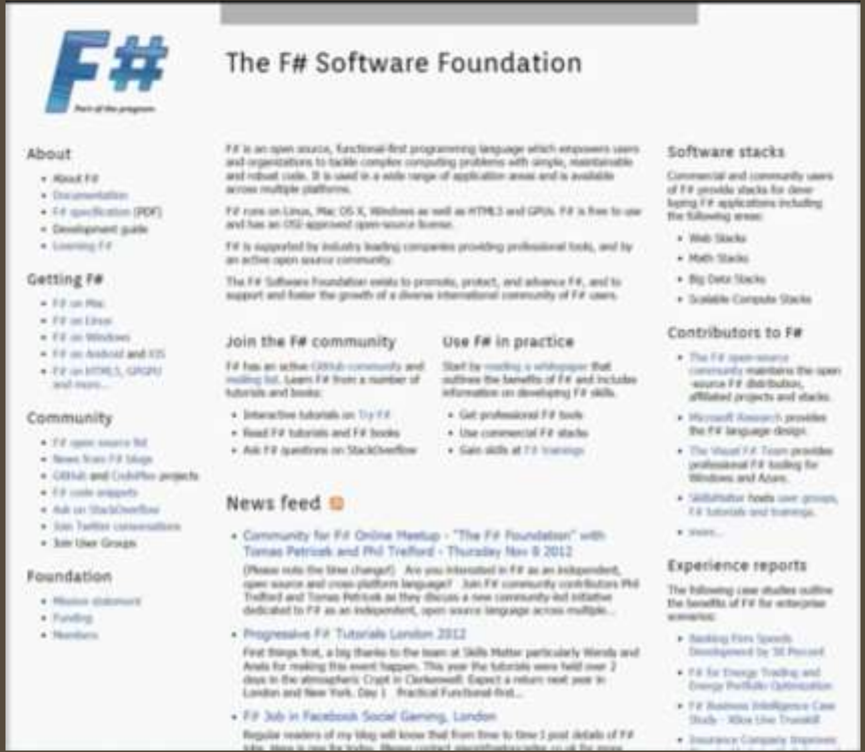
Created by Phil Trelford, Tomas Petricek, Don Syme

2012: Early talks were fueled by much tea and cake. Domain purchased, and folks joined by emailing.

2014: Formally organized.

2015: Board elections held; 501c(3) Charitable organization.

2016: Nearly 1000 members, training programs, working groups (join!).



The F# Software Foundation

About

- About F#
- Documentation
- F# specification (PDF)
- Development guide
- Learning F#

Getting F#

- F# on Mac
- F# on Linux
- F# on Windows
- F# on Android and iOS
- F# on HTML5, GPGPU and more...

Community

- F# open source list
- News from F# blogs
- GitHub and CodePlex projects
- F# code snippets
- Ask on StackOverflow
- Join Twitter conversations
- Join User Groups

Foundation

- Mission statement
- Funding
- Members

F# is an open source, functional-first programming language which empowers users and organizations to tackle complex computing problems with simple, maintainable and robust code. It is used in a wide range of application areas and is available across multiple platforms.

F# runs on Linux, Mac OS X, Windows as well as HTML5 and GPUs. F# is free to use and has an OSI-approved open-source license.

F# is supported by industry leading companies providing professional tools, and by an active open source community.

The F# Software Foundation exists to promote, protect, and advance F#, and to support and foster the growth of a diverse international community of F# users.

Join the F# community

F# has an active GitHub community and mailing list. Learn F# from a number of tutorials and books:

- Interactive tutorials on Try F#
- Read F# tutorials and F# books
- Ask F# questions on StackOverflow

Use F# in practice

Start by reading a whitepaper that outlines the benefits of F# and includes information on developing F# skills.

- Get professional F# tools
- Use commercial F# stacks
- Gain skills at F# trainings

News feed

- Community for F# Online Meetup - "The F# Foundation" with Tomas Petricek and Phil Trelford - Thursday Nov 8 2012
  (Please note the time change!) Are you interested in F# as an independent, open source and cross-platform language? Join F# community contributors Phil Trelford and Tomas Petricek as they discuss a new community-led initiative dedicated to F# as an independent, open source language across multiple...
- Progressive F# Tutorials London 2012
  First things first, a big thanks to the team at Skills Matter particularly Wendy and Anela for making this event happen. This year the tutorials were held over 2 days in the atmospheric Crypt in Clerkenwell. Expect a return next year in London and New York. Day 1 Practical Functional-first...
- F# Job in Facebook Social Gaming, London
  Regular readers of my blog will know that from time to time I post details of F#

Software stacks

Commercial and community users of F# provide stacks for developing F# applications including the following areas:

- Web Stacks
- Math Stacks
- Big Data Stacks
- Scalable Compute Stacks

Contributors to F#

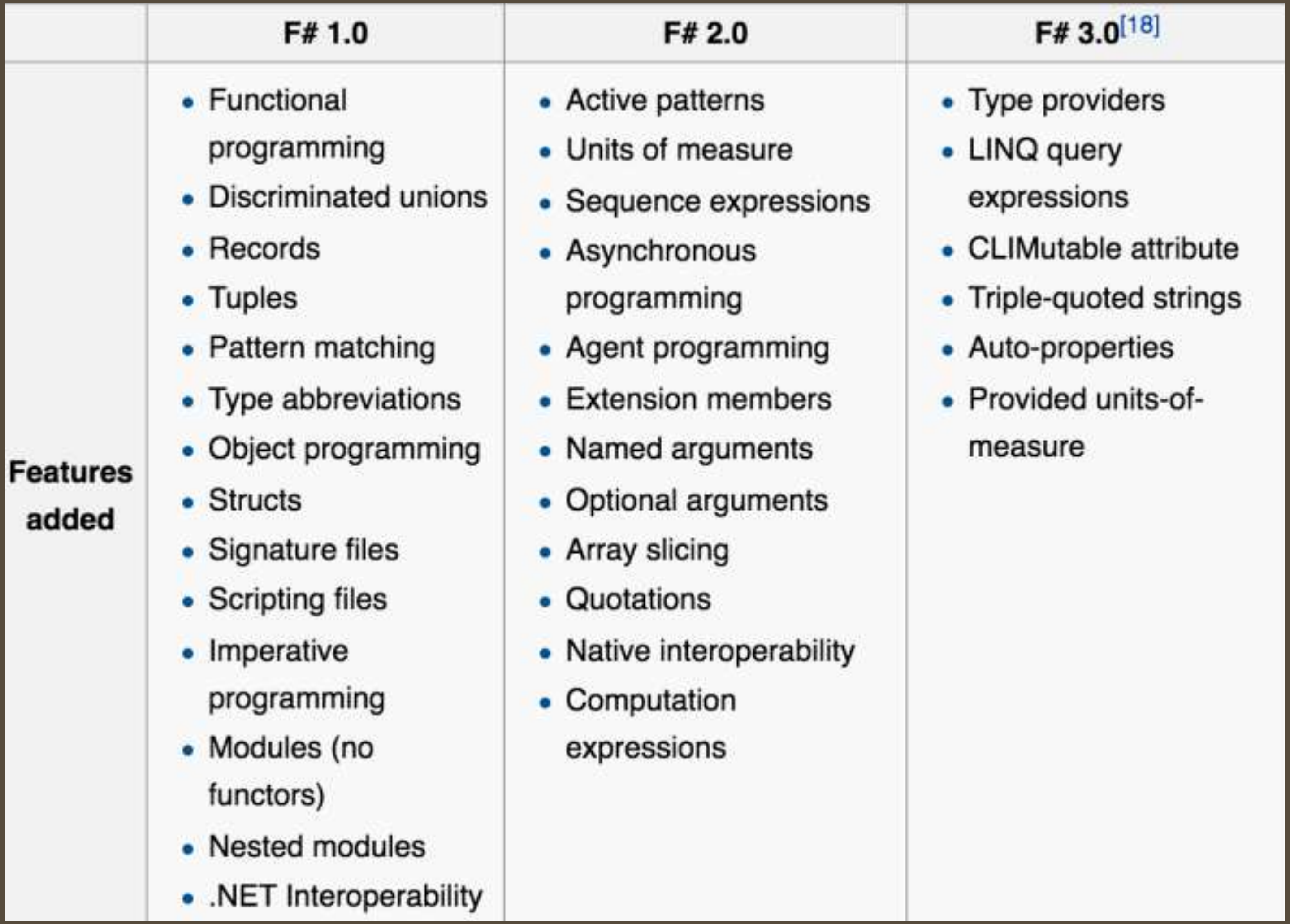
- The F# open-source community maintains the open-source F# distribution, affiliated projects and stacks.
- Microsoft Research provides the F# language design.
- The Visual F# Team provides professional F# tooling for Windows and Azure.
- SkillsMatter hosts user groups, F# tutorials and trainings.
- more...

Experience reports

The following case studies outline the benefits of F# for enterprise scenarios:

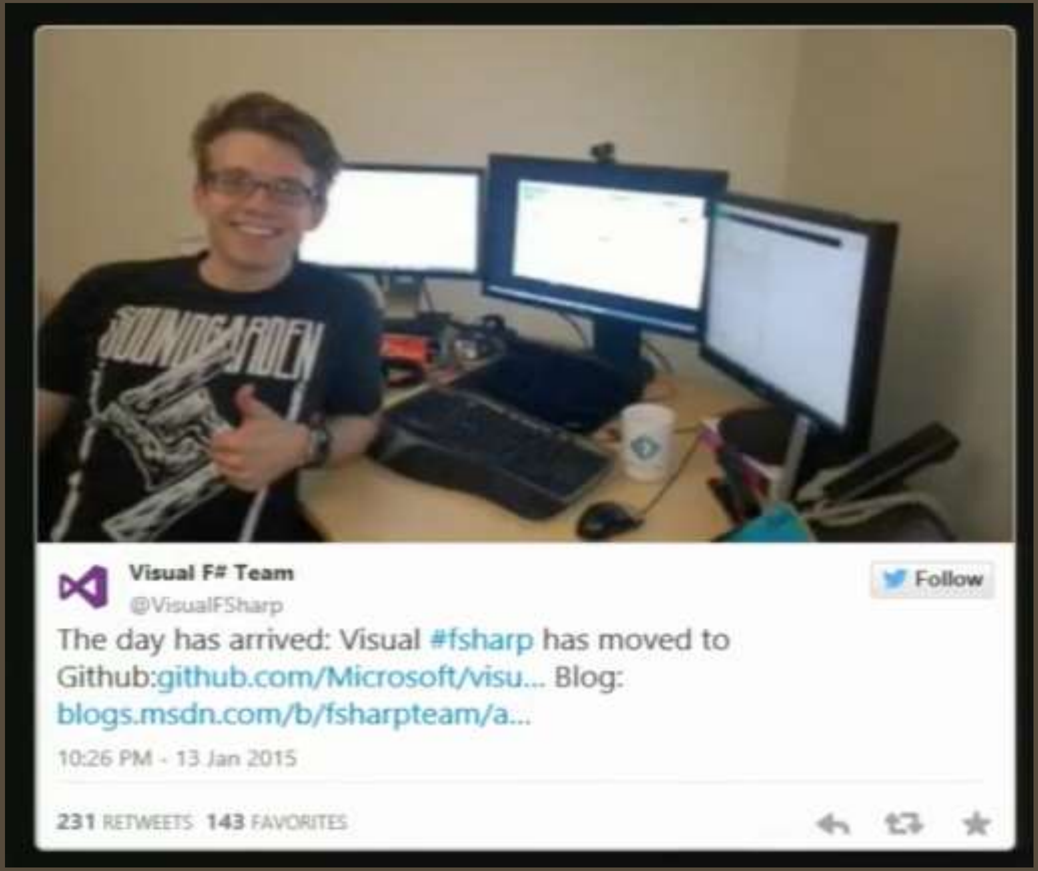
- Banking Firm Speeds Development by 50 Percent
- F# for Energy Trading and Energy Portfolio Optimization
- F# Business Intelligence Case Study - XBox Live Trueskill
- Insurance Company Improves

F# 3.0, 2012



	F# 1.0	F# 2.0	F# 3.0[18]
Features added	• Functional programming • Discriminated unions • Records • Tuples • Pattern matching • Type abbreviations • Object programming • Structs • Signature files • Scripting files • Imperative programming • Modules (no functors) • Nested modules • .NET Interoperability	• Active patterns • Units of measure • Sequence expressions • Asynchronous programming • Agent programming • Extension members • Named arguments • Optional arguments • Array slicing • Quotations • Native interoperability • Computation expressions	• Type providers • LINQ query expressions • CLIMutable attribute • Triple-quoted strings • Auto-properties • Provided units-of-measure

F# MOVED TO GITHUB, 2015



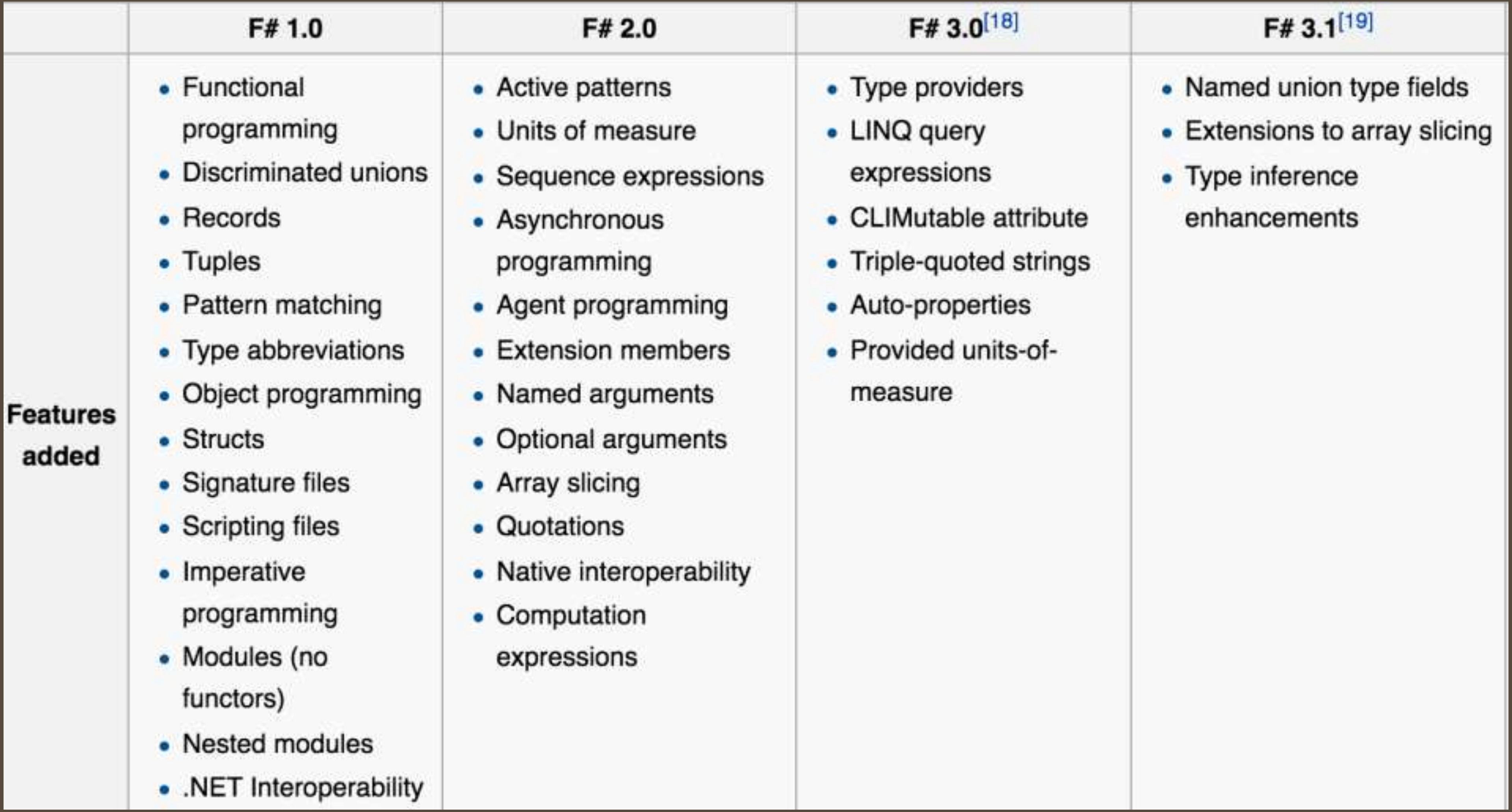
Visual F# Team
@VisualFSharp

The day has arrived: Visual #fsharp has moved to Github:github.com/Microsoft/visu... Blog: blogs.msdn.com/b/fsharpteam/a...

10:26 PM - 13 Jan 2015

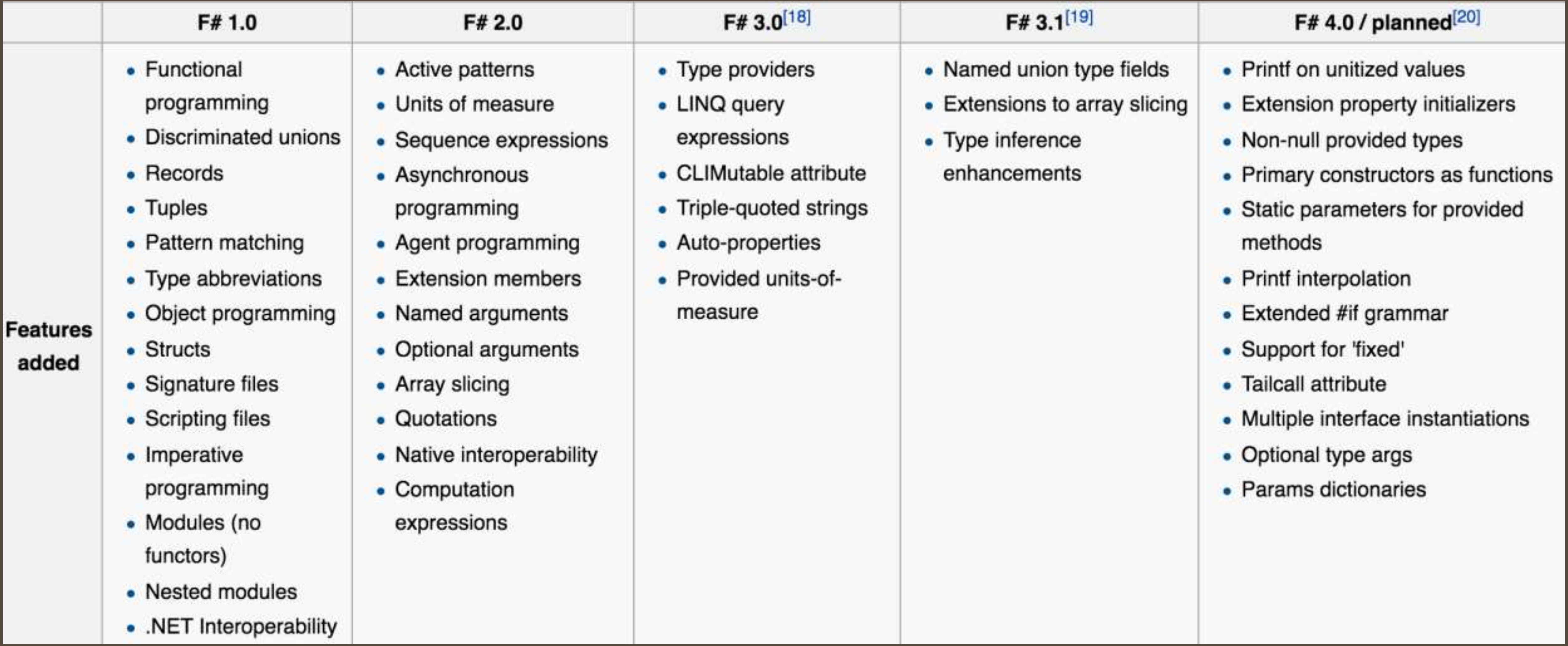
231 RETWEETS 143 FAVORITES

F# 3.1, 2015



	F# 1.0	F# 2.0	F# 3.0[18]	F# 3.1[19]
Features added	• Functional programming • Discriminated unions • Records • Tuples • Pattern matching • Type abbreviations • Object programming • Structs • Signature files • Scripting files • Imperative programming • Modules (no functors) • Nested modules • .NET Interoperability	• Active patterns • Units of measure • Sequence expressions • Asynchronous programming • Agent programming • Extension members • Named arguments • Optional arguments • Array slicing • Quotations • Native interoperability • Computation expressions	• Type providers • LINQ query expressions • CLIMutable attribute • Triple-quoted strings • Auto-properties • Provided units-of-measure	• Named union type fields • Extensions to array slicing • Type inference enhancements

F# 4.0, 2016



	F# 1.0	F# 2.0	F# 3.0[18]	F# 3.1[19]	F# 4.0 / planned[20]
Features added	• Functional programming • Discriminated unions • Records • Tuples • Pattern matching • Type abbreviations • Object programming • Structs • Signature files • Scripting files • Imperative programming • Modules (no functors) • Nested modules • .NET Interoperability	• Active patterns • Units of measure • Sequence expressions • Asynchronous programming • Agent programming • Extension members • Named arguments • Optional arguments • Array slicing • Quotations • Native interoperability • Computation expressions	• Type providers • LINQ query expressions • CLIMutable attribute • Triple-quoted strings • Auto-properties • Provided units-of-measure	• Named union type fields • Extensions to array slicing • Type inference enhancements	• Printf on unitized values • Extension property initializers • Non-null provided types • Primary constructors as functions • Static parameters for provided methods • Printf interpolation • Extended #if grammar • Support for 'fixed' • Tailcall attribute • Multiple interface instantiations • Optional type args • Params dictionaries

Thank you to our contributors

Visual F# 4.0 marks the first major-version release of the F# language and VS tools to include community contributions. As such, we offer a heart-felt "thank you!" to all of the F# community developers who contributed code, opened issues, or dogfooded early builds.

TODAY: TYPE PROVIDERS ARE BLOSSOMING

+ many more!

AND BY TYPE PROVIDERS, I MEAN THE COMMUNITY.

http://fsharp.org, FSSF Slack

F# Weekly

http://fsharpforfunandprofit.com

F# Conf videos

c4fsharp.net

fssnip.net

#fsharp on twitter

GitHub.com/fsprojects

Google groups

A HISTORY OF F#

FROM EUCLID TO TYPE PROVIDERS

Rachel Reese
@rachelreese
rachelree.se