

A conceptual history of programming languages

Simone Martini

séminaire Collegium – November 19, 2018

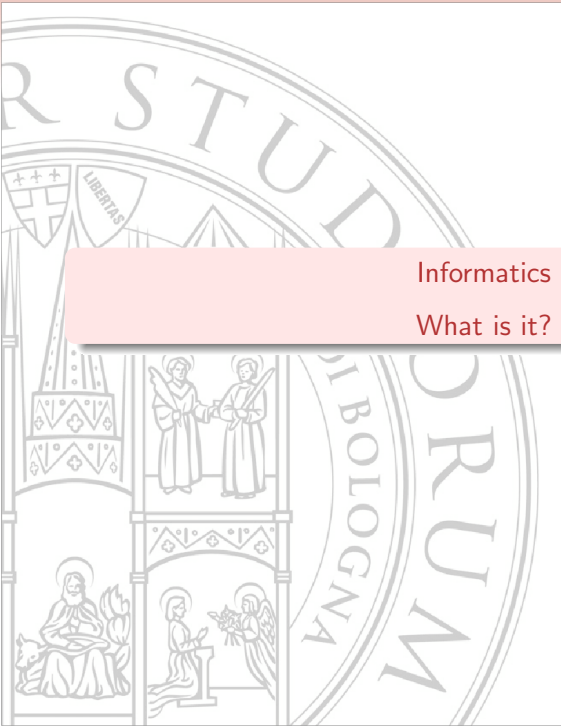


**COLLEGIUM
DE LYON**
Université de Lyon



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA
DIPARTIMENTO DI INFORMATICA - SCIENZA E INGEGNERIA

inria
informatiques mathématiques

The background of the slide features a large, faint watermark of the University of Bologna seal. The seal is circular and contains the text 'R STUD' at the top and 'BOLOGNA' and 'RUM' at the bottom. In the center, there are four panels depicting figures: two standing figures holding staffs, a seated figure with a dog, and a kneeling figure receiving a blessing from a standing figure. A shield with three crosses and the word 'LIBERTAS' is also visible.

Informatics

What is it?

Informatics

- A collection of artefacts (applications)
- A technology making possible those artefacts
- A science founding that technology

In limine

Unique

It concerns an immaterial good: **information**

Used to affect — and to which degree! — the material world

Young

- First electronic computer: 1947
- First computer science department in USA: 1962, Purdue
- First computer science degree in France: 1968
Maitrises d'informatique, Orsay
- First computer science degree in Italy: 1969, Pisa

Resistances, 1962

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

[Conte, Computerworld magazines, 1999]

An epistemology?

- 1 What are the entities with which the science of computation deals?
- 2 What kinds of facts about these entities would we like to derive?
- 3 What are the basic assumptions from which we should start?

[John McCarthy, 1962]

Still in 1967:

*A. Newell, A. Perlis, H. Simon: letter to Science on
"Computer Science"*

An epistemology?

- 1 What are the entities with which the science of computation deals?
- 2 What kinds of facts about these entities would we like to derive?
- 3 What are the basic assumptions from which we should start?

[John McCarthy, 1962]

Still in 1967:

*A. Newell, A. Perlis, H. Simon: letter to Science on
"Computer Science"*

The background of the slide features a large, faint watermark of the seal of the University of Bologna. The seal is circular and contains the text "R STUD" at the top, "DI BOLOGNA" on the left, and "ORUM" at the bottom. The central part of the seal depicts a building with a tower and a shield with a cross and three stars, and the word "LIBERTAS" written on a banner. Below the building are two scenes: one showing a seated figure with a dog and another showing a kneeling figure and a standing figure.

What kind of computer scientist am I?

CS subject areas, excerpt

- 1 Hardware
- 2 Systems and networks
- 3 Software and languages
- 4 Theory of computation
- 5 Information systems
- 6 Methodologies (including AI, simulations, graphics, etc.)
- 7 Security, privacy
- 8 Human-centered and applied computing
(including interfaces, ubiquitous. mobile, etc.)

CS subject areas, excerpt

- 1 Hardware
- 2 Systems and networks
- 3 Software and languages**
- 4 Theory of computation**
- 5 Information systems
- 6 Methodologies (including AI, simulations, graphics, etc.)
- 7 Security, privacy
- 8 Human-centered and applied computing
(including interfaces, ubiquitous. mobile, etc.)

The linguistic methaphor

From the end of the 50s

Programming: in artificial, formal languages

- Tool of the trade
- Object of study
- Meta-language

No scientific discipline exists without first inventing a visual and written language which allows it to break with its confusing past.

[B. Latour, Visualisation and cognition, 1986]

The linguistic methaphor

From the end of the 50s

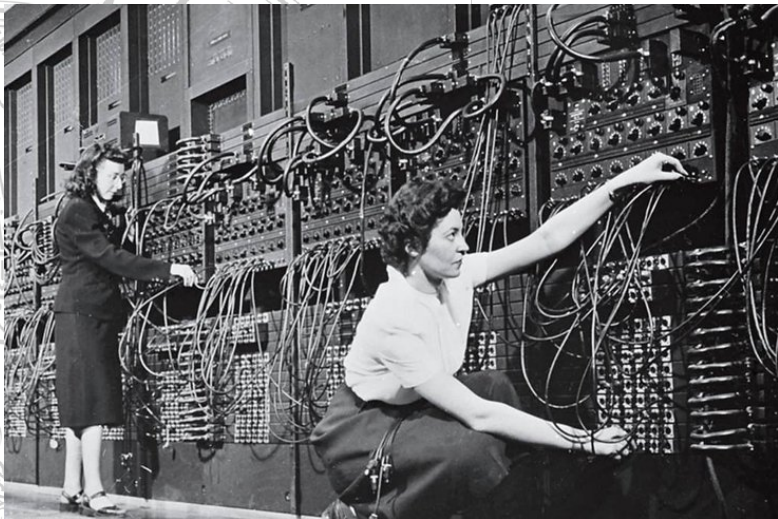
Programming: in artificial, formal languages

- Tool of the trade
- Object of study
- Meta-language

No scientific discipline exists without first inventing a visual and written language which allows it to break with its confusing past.

[B. Latour, Visualisation and cognition, 1986]

Programming the first computer, ENIAC



Program correctness

From the middle of 60s

How to prove a program **correct**?

- How to give semantics to a program
- How to specify its intended behaviour
- How to prove that the two match

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

[J. McCarthy, 1963]

Program correctness

From the middle of 60s

How to prove a program **correct**?

- How to give semantics to a program
- How to specify its intended behaviour
- How to prove that the two match

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

[J. McCarthy, 1963]



The model seems to be...

Structural engineering

- mathematical physics laws
- empirical knowledge

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

McCarthy:

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

The larger context

The quest for a “Mathematical Theory of Computation”

How does mathematical logic fit into this theory?

And for what purposes?

A mathematical theory is the entrance ticket to science

The larger context

The quest for a “Mathematical Theory of Computation”

How does mathematical logic fit into this theory?

And for what purposes?

A mathematical theory is the entrance ticket to science

The larger context

The quest for a “Mathematical Theory of Computation”

How does mathematical logic fit into this theory?

And for what purposes?

A mathematical theory is the entrance ticket to science

Computer Scientist

Education

- PhD, 1988: Pisa
- Post-doc, 1989-1990: Stanford
- Junior faculty, 1988-1994: Pisa
- Professor, 1994-2002: Udine
- Professor and Head of CSE, 2002- : Bologna

Research

- Foundations of programming languages
- Theory and complexity of computation

Foundations

- Use (and develop) theories, in:
 - mathematical logic
 - theory of categories
 - very weak topological spaces
 - formal systems
- To help design **better** languages
- To help prove a program **correct**
- To help obtaining **general** properties, e.g. complexity

Some journals:

- Theoretical Computer Science
- Journal of Logic Programming
- Mathematical Structures in CS
- Archive for Mathematical Logic
- Journal of Symbolic Logic
- Information and Computation
- ACM Transactions on Computational Logic
- ...

Publications

Some journals:

- Theoretical Computer Science
- Journal of Logic Programming
- Mathematical Structures in CS
- Archive for Mathematical Logic
- Journal of Symbolic Logic
- Information and Computation
- ACM Transactions on Computational Logic
- ...



The big project

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

First episode: Types

Why types?

Modern programming languages:

- control flow specification: small fraction
- abstraction mechanisms to model application domains.
- Types are a crucial building block of these abstractions
- And they are a mathematical logic concept, aren't they?

Why types?

Modern programming languages:

- control flow specification: small fraction
- abstraction mechanisms to model application domains.
- Types are a crucial building block of these abstractions
- And they are a mathematical logic concept, aren't they?

Modelling tool

In the simulation of complex situations in the real world, it is necessary to construct in the computer analogues of the objects of the real world, so that procedures representing types of [data] may operate upon them in a realistic fashion.

[Tony Hoare, 1964] (page 46, and, more generally, all Section 4)

Types in programming languages

A (data) type is a collection of (effectively presented) values, together with (effective) operations on those values.

Types in programming languages

A (data) type is a collection of (effectively presented) values, together with (effective) operations on those values.

integer numbers

with $+$, $-$, \times , div , mod

strings of characters

with concatenation, selection of an element, ...

rational numbers

with $+$, $-$, \times , $/$, $\sqrt{\quad}$, ...

Reals: no effective presentation is possible.

Types in programming languages

A (data) type is a collection of (effectively presented) values, together with (effective) operations on those values.

*“Collections of other values” (arrays, records, ...)
With their operations.*

*Functions between two types
With their operations.*

Types for...

At the level of:

- Project: conceptual organisation of data
- Programming: support for correction
- Translation: support for implementation

Types for implementation

Not much interested here

An integer is coded with 32 bits, in two's complement

A rational is coded in IEEE 754 floating point, ...

Crucial information for the translator/interpreter

The type of the FORTRAN language (1956 and ff)

Types for correctness

Correctness

- “minimal”: we don't add integers and strings
- “sufficient”: for a certain class of errors
- “conservative”: “correct” phrases may be type-incorrect

Minimal correctness

In physics: dimensional control

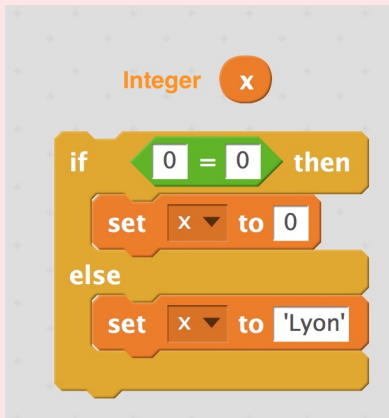
- an acceleration must be m/sec^2
- of course “wrong” formulas may have the right dimension

Sufficient correctness

For a certain class of errors:

- “well typed expressions do not go wrong”
- guarantee that some errors will not happen
- e.g., an integer will never be used as a memory address

Conservative correctness



Evaluation “does not go wrong”
But the program is ill-typed

In the great tradition:

[Types] forbid certain inferences which would otherwise be valid, but [do] not permit any which would otherwise be invalid.

[Whitehead and Russell, Principia Mathematica, 1910]

We shall now introduce a type system which, in effect, singles out a decidable subset of those wfes that are safe; i.e., cannot give rise to ERRORS. This will disqualify certain wfes which do not, in fact, cause ERRORS and thus reduce the expressive power of the language.

[Morris, PhD thesis, 1968]

In the great tradition:

[Types] forbid certain inferences which would otherwise be valid, but [do] not permit any which would otherwise be invalid.

[Whitehead and Russell, Principia Mathematica, 1910]

We shall now introduce a type system which, in effect, singles out a decidable subset of those wfes that are safe; i.e., cannot give rise to ERRORS. This will disqualify certain wfes which do not, in fact, cause ERRORS and thus reduce the expressive power of the language.

[Morris, PhD thesis, 1968]

Principia Mathematica

Types

Stratify the universe to avoid paradoxes

No set can be a member of itself

“The barber who shaves those and only those who do not shave by themselves”



However

Not all the circular definitions are dangerous, and it is a task for the logician to isolate the good ones.

[Dmitry Mirimanoff, 1917]

There are perfectly fine sets which belong to themselves

Non well-founded set-theories



In foundations of mathematics, types:

- never supposed to be used by the working mathematician
- *in principle* could be used, to avoid paradoxes

In programming languages, types:

- are used everyday, by everyone
- should be made more “expressive”, “flexible”



In foundations of mathematics, types:

are perceived as constraints
(they “forbid” something, as in Russell’s quote).

In programming languages, types:

are experienced as an enabling feature (Voevodsky),
allowing simpler writing of programs,
and better verification of their correctness.

Types for...

At the level of:

- Project : conceptual organisation of data
- Programming : support for correction
- Translation : support for implementation

Conceptual organisation

In the simulation of complex situations in the real world, it is necessary to construct in the computer analogues of the objects of the real world, so that procedures representing types of [data] may operate upon them in a realistic fashion.

[Tony Hoare, 1964] (page 46, and, more generally, all Section 4)

Type structure is a syntactic discipline for enforcing levels of abstraction

[John Reynolds, 1983]

Conceptual organisation

In the simulation of complex situations in the real world, it is necessary to construct in the computer analogues of the objects of the real world, so that procedures representing types of [data] may operate upon them in a realistic fashion.

[Tony Hoare, 1964] (page 46, and, more generally, all Section 4)

Type structure is a syntactic discipline for enforcing levels of abstraction

[John Reynolds, 1983]

“Programming” languages

- What we insist in calling **programming** languages
- Are powerful tools to organize, make coherent, and model reality
 - data models
 - procedural models
 - interaction models
 - synchronization models
 - organization models
 - ...

We today conflate:

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

One of the goals:

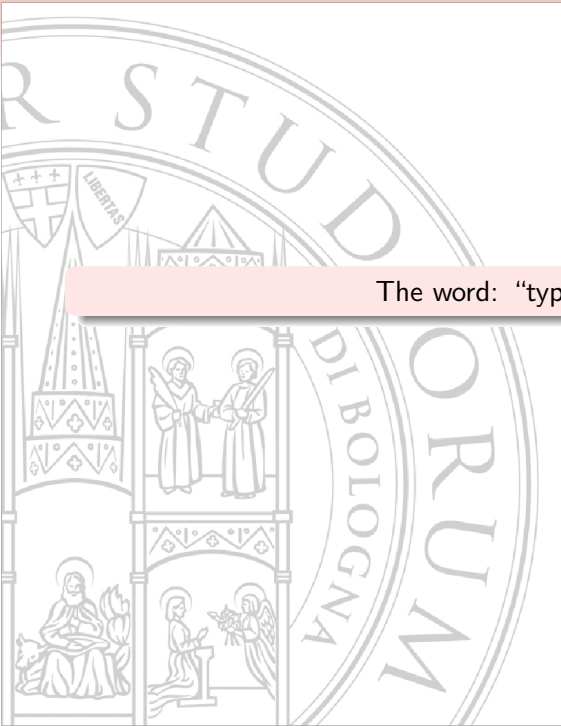
separate them and identify when they arrive in the PL literature

We today conflate:

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

One of the goals:

separate them and identify when they arrive in the PL literature



The word: "type"

Types in early Fortran?

Two types of constants are permissible: fixed points (restricted to integers) and floating points

32 types of statement

[The FORTRAN automatic coding system, 1956]

*Any fixed point (floating point) constant, variable, or subscripted variable is an expression of the same **mode**.*

[ibidem]

Algol 58: types

Type declarations serve to declare certain variables, or functions, to represent quantities of a given class, such as the class of integers or class of Boolean values.

[Perlis and Samelson. Preliminary report: International algebraic language. Commun. ACM 1(12), December 1958.]



No types
in the preparatory papers!

A data symbol falls in one of the following *classes*:

a) Integer b) Boolean c) General

The symbol classification statements are:

INTEGER (s_1, \dots, s_n)

BOOLEAN (s_1, \dots, s_n)

[Backus et al. Proposal for a programming language. ACM Ad Hoc Committee on Languages, 1958.]

Algol 60: maturity

*Integers are of type **integer**. All other numbers are of type **real**.*

*The various “**types**” (**integer**, **real**, **Boolean**) basically denote properties of values.*

[Backus et al. Report on the algorithmic language ALGOL 60. Commun. ACM 3(5), May 1960.]

1950s and 1960s

- Type based distinctions for compilation: always present
- “Type” as a technical term: Algol 58
- (Almost) stable since Algol 60
- Mode
 - in Algol 68, *d’après* early Fortran usage
 - “types (or modes)”, still in Reynolds 1975

1950s and 1960s

- Type based distinctions for compilation: always present
- “Type” as a technical term: Algol 58
- (Almost) stable since Algol 60

- Mode
 - in Algol 68, *d’après* early Fortran usage
 - “types (or modes)”, still in Reynolds 1975

The word: “type”

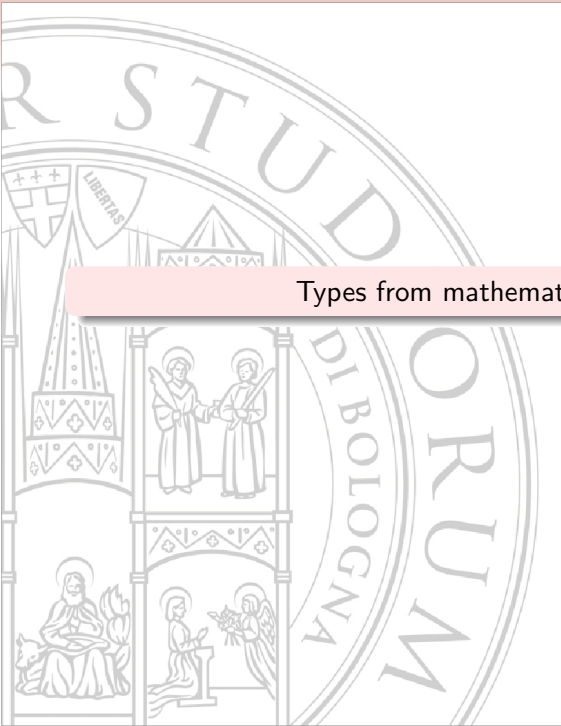
The technical term “type”:

- appears to be a semantical shift from the generic term
- no role of the “type” from mathematical logic

*The use of ‘type,’ as in ‘x is of type **real**,’ was analogous to that employed in logic.*

*Both programming language design and logic dipped into the English language and came up with the same word for **roughly the same purpose**.*

[A. Perlis, The American side of the development of Algol, 1981]



Types from mathematical logic

- certainly people knew “some logic”:
McCarthy, Hoare, Landin, Scott (!), Morris, etc.

but

- Morris (1968) cites Curry (1958), but not Church (1940)
- Reynolds (1974) rediscovers Girard’s System F (1971)
- Milner (1977-78) rediscovers
simple type inference (Hindley, 1969)

Programming languages and proof-theory are talking the same language, but the conflation is anonymous.

- certainly people knew “some logic”:
McCarthy, Hoare, Landin, Scott (!), Morris, etc.

but

- Morris (1968) cites Curry (1958), but not Church (1940)
- Reynolds (1974) rediscovers Girard's System F (1971)
- Milner (1977-78) rediscovers
simple type inference (Hindley, 1969)

Programming languages and proof-theory are talking the same language, but the conflation is anonymous.

- certainly people knew “some logic”:
McCarthy, Hoare, Landin, Scott (!), Morris, etc.

but

- Morris (1968) cites Curry (1958), but not Church (1940)
- Reynolds (1974) rediscovers Girard’s System F (1971)
- Milner (1977-78) rediscovers
simple type inference (Hindley, 1969)

Programming languages and proof-theory are talking the same language, but the conflation is anonymous.



The formidable middleman:

λ -calculus

The catalyst:

Curry-Howard isomorphism, (1969); 1980



The formidable middleman:

λ -calculus

The catalyst:

Curry-Howard isomorphism, (1969); 1980



The explicit recognition:

Per Martin-Löf.

Constructive mathematics and computer programming.
(1979); 1982.

Correlatively, the third form of judgment may be read not only

a is an object of type (element of the set) A ,

a is a proof of the proposition A ,

but also

a is a program for the problem (task) A .

Why this is interesting

Our programming languages are also (a huge part of) the metalanguage in which we express the discipline.

“Programming” languages

No scientific discipline exists without first inventing a visual and written language which allows it to break with its confusing past.

[B. Latour, Visualisation and Cognition: Thinking with Eyes and Hands; 1986]

*Referring to Dagognet, F.: Tableaux et Langages de la Chimie. Paris : Le Seuil 1969;
and to: Ecriture et Iconographie. Paris : Vrin 1973.*

*What we call programming languages are both such a founding language **and** one of the very objects of the discipline.*

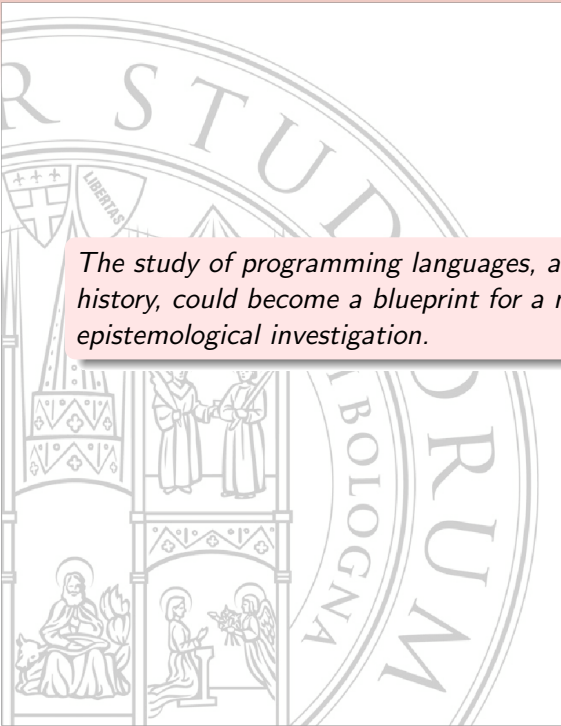
“Programming” languages

No scientific discipline exists without first inventing a visual and written language which allows it to break with its confusing past.

[B. Latour, Visualisation and Cognition: Thinking with Eyes and Hands; 1986]

*Referring to Dagognet, F.: Tableaux et Langages de la Chimie. Paris : Le Seuil 1969;
and to: Ecriture et Iconographie. Paris : Vrin 1973.*

*What we call programming languages are both such a founding language **and** one of the very objects of the discipline.*

The background of the slide features a large, faint watermark of the University of Bologna seal. The seal is circular and contains the text 'R STUD' at the top and 'BOLOGNA' and 'RUM' at the bottom. In the center, there are several figures: two standing figures in the upper part, a seated figure on the left, and a kneeling figure on the right. A shield with three crosses and the word 'LIBERTAS' is also visible.

The study of programming languages, and of their “conceptual” history, could become a blueprint for a more general epistemological investigation.

An example

The unstoppable march of object-oriented programming

- ADT: simple concept with clear mathematical semantics, 1975ff
- Object: complex concept with opaque (mathematical?) semantics

Objects rule the world, not ADTs !
Contrary to what one would expect.

An example

The unstoppable march of object-oriented programming

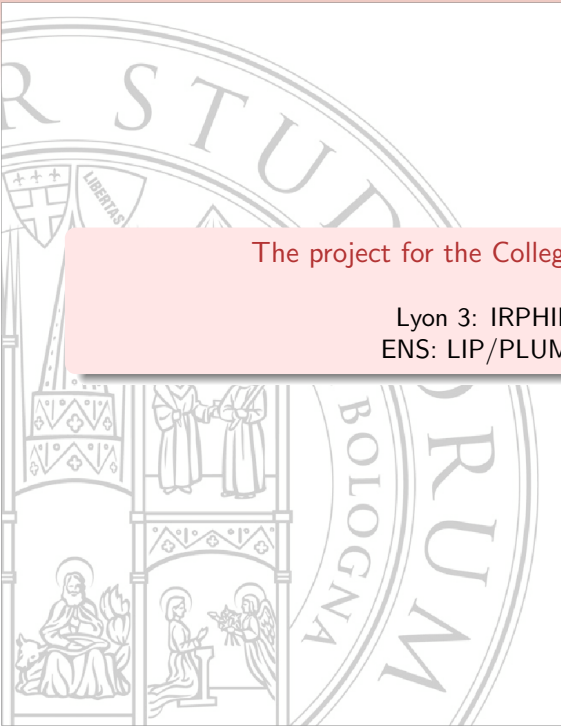
- ADT: simple concept with clear mathematical semantics, 1975ff
- Object: complex concept with opaque (mathematical?) semantics

Objects rule the world, not ADTs !
Contrary to what one would expect.

Conclusion

The history of computer science is innervated by the continuous tension between formal beauty and technological effectiveness. Types in programming languages are an evident example of this dialectics.

We always exploited what we found useful for the design of more elegant, economical, usable artefacts.



The project for the Collegium de Lyon

Lyon 3: IRPHIL
ENS: LIP/PLUME

At the Collegium...

Next steps?

- Continuation passing transformation (van Wijngaarden, 1964)
- Exceptions handlers (PL/I: resume-based; etc.)
- Pinpoint the impact of the Curry-Howard isomorphism
- ...

Methodology

History and Philosophy of Computing: HAPOC

Across disciplines:

for adding to the maturity of computing in general.

Bring together:

computer scientists, historians, and philosophy scholars.

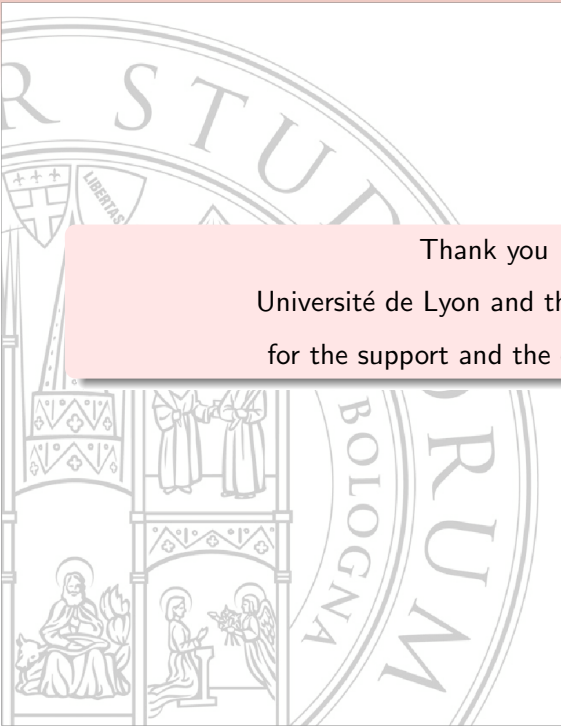
Narrow the gap between a technology and its professional history.

Perspective:

useful for the technician of today, for a better science.

Side paths

- Computational thinking: not just coding
- Programming as interaction (Smalltalk, Logo, . . . , Scratch, etc.)
- Program as inscriptions (Latour)
- Is the “traditional” Mathematical theory of computation still useful?



Thank you

Université de Lyon and the Collegium
for the support and the opportunity

Let us follow Latour. . .

“Programs” are:

- mobile
- immutable when they move
- flat
- “their scale may be changed at will”:
phenomena can be dominated with the eyes and held by hands
- reproduced and communicated at little cost
- may be reshuffled and recombined
- may be made part of a written text
- they merge with geometry (they are a faithful model of reality)

They are *inscriptions*, like geographical maps, or diagrams.

More: programming languages are a formal, general language of (and for) inscriptions.

Let us follow Latour. . .

“Programs” are:

- mobile
- immutable when they move
- flat
- “their scale may be changed at will”:
phenomena can be dominated with the eyes and held by hands
- reproduced and communicated at little cost
- may be reshuffled and recombined
- may be made part of a written text
- they merge with geometry (they are a faithful model of reality)

They are **inscriptions**, like geographical maps, or diagrams.

More: programming languages are a formal, general language of (and for) inscriptions.

Let us follow Latour. . .

“Programs” are:

- mobile
- immutable when they move
- flat
- “their scale may be changed at will”:
phenomena can be dominated with the eyes and held by hands
- reproduced and communicated at little cost
- may be reshuffled and recombined
- may be made part of a written text
- they merge with geometry (they are a faithful model of reality)

They are **inscriptions**, like geographical maps, or diagrams.

More: programming languages are a formal, general language of (and for) inscriptions.