

Several types of types in programming languages

Simone Martini

Dipartimento di Informatica – Scienza e Ingegneria
Alma mater studiorum • Università di Bologna

HaPoC 2015, Pisa October 8, 2015

On the types mailing list

From: Vladimir Voevodsky

Date: 12 May 2014

(...)

The concept of types as we use it today has very little [to do] with how types were perceived by Russell or Church.

For them types were a restriction mechanism.

(...) today are a constructive tool. (...)

When and where did types appear in programming languages which were enabling rather than forbidding in nature?

We today conflate:

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

Goal:

separate them and identify when they arrive in the PL literature

We today conflate:

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

Goal:

separate them and identify when they arrive in the PL literature

Antefact: mathematical logic

A type is the range of significance of a variable.

[Russell and Whitehead, 1910]

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

[ibidem]

And then...

Leon Chwistek, in 1921

Frank P. Ramsey in 1926

...

Alonzo Church in 1940

...

Antefact: mathematical logic

A type is the range of significance of a variable.

[Russell and Whitehead, 1910]

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

[*ibidem*]

And then...

Leon Chwistek, in 1921

Frank P. Ramsey in 1926

...

Alonzo Church in 1940

...

The background of the slide features a large, faint watermark of the University of Cologne seal. The seal is circular and contains a central shield with a cross and three stars, the word 'LIBER' on a banner, and a figure seated on the left. Below the shield are two smaller scenes: one of a kneeling figure and another of a kneeling figure with an angel. The text 'R STU' is visible at the top and 'OLOGNA RUM' at the bottom of the seal.

Part I

The term “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)

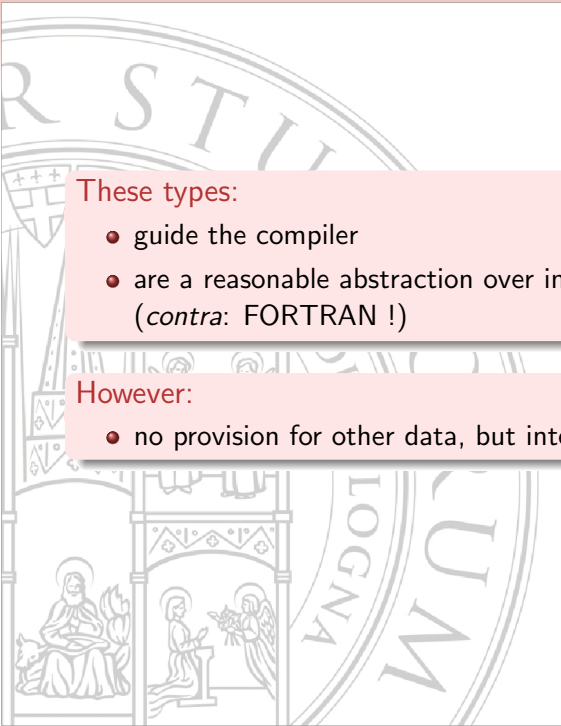
[Backus et al. Proposal for a programming language. ACM Ad Hoc Committee on Languages, 1958.]

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.]

The background of the slide features a large, faint watermark of the University of Padua seal. The seal is circular and contains the text 'UNIVERSITAS PADOVA' at the top and 'MDCCCXXXIII' at the bottom. In the center, there is a shield with a cross and four smaller crosses in the quadrants. Below the shield, there are three figures: a seated figure on the left, a kneeling figure in the middle, and a standing figure on the right. The text 'LOGNA' and 'UM' are also visible on the seal.

These types:

- guide the compiler
- are a reasonable abstraction over implementation details (*contra*: FORTRAN !)

However:

- no provision for other data, but integer, real, Boolean



Moreover

The technical term “type”:

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

The background features a large, faint watermark of the University of Cologne seal. The seal is circular and contains the text 'R STU' at the top and 'OLOGNA RUM' at the bottom. In the center, there is a shield with a cross and three stars, and below it, a scene with a seated figure and a kneeling figure.

Part II

Types as an abstraction mechanism

The modern view

*Type structure is a syntactic discipline
for enforcing levels of abstraction*

[John Reynolds, 1983]



McCarthy: the “weakness” of Algol

1961:

defining new data spaces in terms of given base spaces and (...) defining functions on the new spaces in terms of functions on the base spaces

[John McCarthy. A basis for a mathematical theory of computation, preliminary report. 1961]



Hoare: records and references

1964:

- ordered collection of named *fields*
- **typed** references (like pointers, but no operations)
- non stack-based, dynamically allocated structures

Dahl and Nygaard: *objects ante litteram*

around 1962:

- record class: *activity*;
- record: *process*;
- record field: *local variable of a process*

- a “process” encapsulates both data objects and their operators: an *object* (Alan Key, 1976).



Hoare: records and references, 2

With Hoare's paper, types become a general abstraction mechanism:

[Our proposal] *is no arbitrary extension to an existing language, but represents a genuine abstraction of some feature which is fundamental to the art or science of computation.*

[Tony Hoare, 1964]

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 STUDI' at the top and 'LOGNA' and 'UM' at the bottom. In the center, there is a shield with a cross and three stars, and below it, a building with a tower. At the bottom of the seal, there are three figures: a seated figure with a dog, a kneeling figure, and a standing figure holding a book.

Hoare: records and references, 3

- 1 from simple to structured values
- 2 types are a **general** modelling tool
- 3 robust abstraction over the memory layout

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 even may operate upon them in a realistic fashion.

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

Robust abstraction

It was a firm principle of our implementation that the results of any program, even erroneous, should be comprehensible without knowing anything about the machine or its storage layout

[Tony Hoare, 2014, personal communication]



Algol W,
circa 1970

Every value is said to be of a certain type.

The following types of structured values are distinguished:

array: (...), record: (...).

[Algol W reference manual, 1972]

Towards ADTs

Morris, 1973 and Reynolds, 1974

The meaning of a syntactically-valid program in a “type-correct” language should never depend upon the particular representation used to implement its primitive types.

*The main thesis of [Morris 1973] is that this property of **representation independence** should hold for user-defined types as well as primitive types.*

[Reynolds, 1974]

ADTs and objects

Why at the beginning of the 80s

abstract data types

give way to objects?

kind of algebraic structures
with nice mathematical semantics

with difficult semantics

flexibility, see later (paper)

ADTs and objects

Why at the beginning of the 80s

abstract data types

give way to objects?

kind of algebraic structures
with nice mathematical semantics

with difficult semantics

flexibility, see later (paper)



Part III

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.



Yet, compare:

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

[Russell and Whitehead, 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]



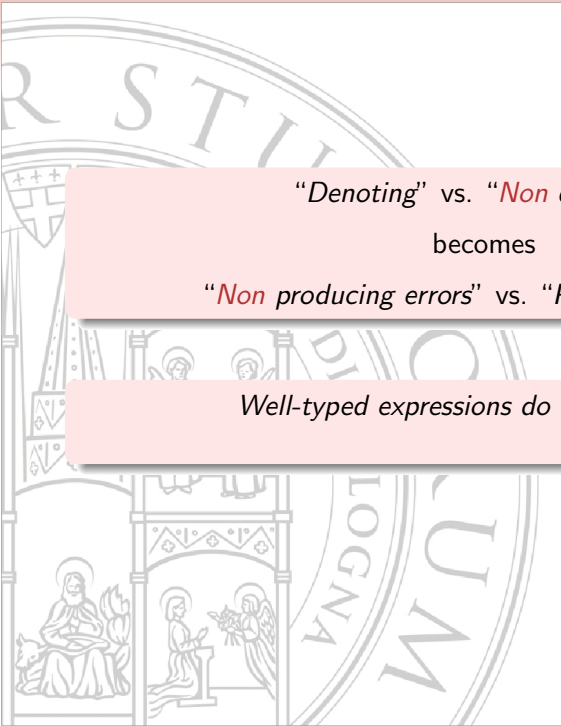
Yet, compare:

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

[Russell and Whitehead, 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 given 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]



“Denoting” vs. *“Non denoting”*

becomes

“Non producing errors” vs. *“Producing errors”*

Well-typed expressions do not go wrong.

[Milner, 1978]

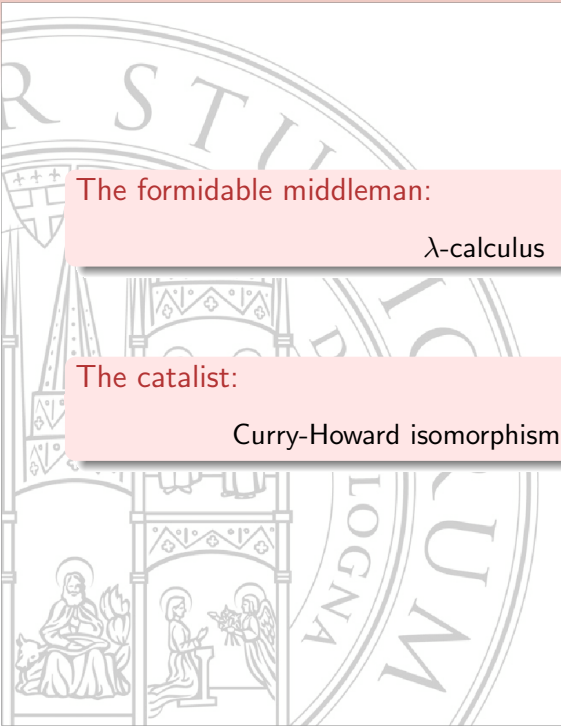


The formidable middleman:

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 .

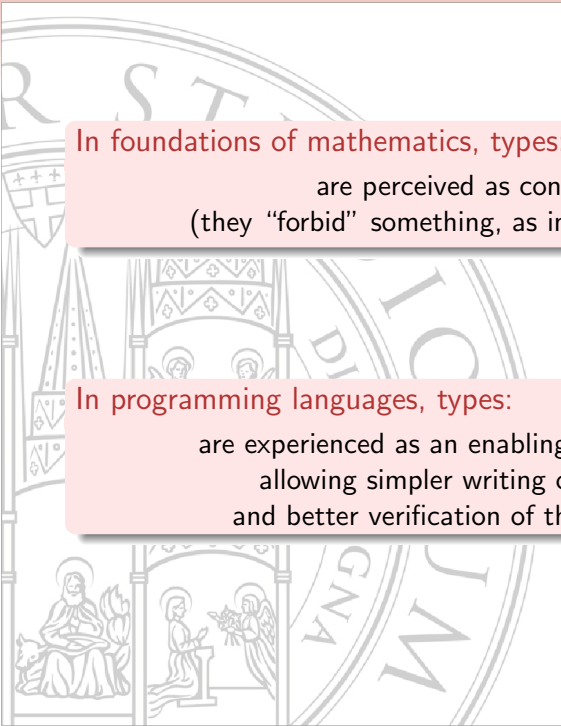


In foundations of mathematics, types:

- 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.

Computer science never used ideological glasses
(types *per se*; constructive mathematics *per se*; logic *per se*; etc.),
but exploited what it found useful for the design
of more elegant, economical, usable artefacts.