Types in Programming Languages

Guest Lecture at Neapolis University Pafos

Vitaly Bragilevsky (JetBrains)

April 10, 2025

```
fun main() {
  val someFunction = { f -> 2 + f(1) }
  val result = someFunction { x -> x * 2 }
  println(result)
}
```

Cannot infer a type for this parameter. Please specify it explicitly.

Programming Languages

From Different Points of Views

Who work with programming languages?

- Software developers
- Programming educators
- Compiler developers
- PL designers
- PL researchers

Design Goals for Programming Languages

- · more convenient to write code
- better code quality
- faster (to write, to execute)
- fewer errors in code
- express ideas

PLs often attempt to solve issues with other PLs

- \cdot C \Rightarrow C++ \Rightarrow Rust
- Java \Rightarrow Kotlin

What can be researched about PLs?

- Taxonomy
- Check goals: are they reached?
- Prototyping new features
- Feature compatibility
- Looking for an essence (the intrinsic nature or indispensable quality of something, especially something abstract, which determines its character)

Essence is often best presented with math



Alonzo Church (1903–1995)



Peter Landin (1930–2009)

PLs before 1960s

- FORTRAN (1954)
- · Lisp (1958)
- · Algol (1958)
- · COBOL (1959)

Landin, 1964:

The mechanical evaluation of expressions

"This paper is a contribution to the "theory" of the activity of using computers"

a/(2b+3)
$$/(a,+(\times(2,b),3))$$
 applicative expressions

Essence

λ -calculus (1930s)

Variables: x, y, z... Application: MN + Abstraction: $\lambda x M$ Argument **Function**

Computing a λ -term: β -reduction

$$(\lambda \times \times (xy))\underline{u} \longrightarrow_{\beta} u(uy)$$

λ -calculus is capable of encoding any computation

- Logical values and operations:
 λx.λy.x (True) и λx.λy.y (False)
- Natural numbers: $\lambda x. \lambda y. y$, $\lambda x. \lambda y. xy$, $\lambda x. \lambda y. xxy...$
- · Addition, subtraction, multiplication
- · Data structures: pairs, lists
- Recursion
- \cdot λ -calculus is equivalent to Turing Machines under Church-Turing thesis.

Let's remember this:

Functions and function calls can be modelled by λ -calculus.

Introducing types to λ -calculus

Extending λ -calculus with types

Simply-typed λ -calculus

M:A

Term M has type A

If x is of type A and N of type B, then

$$\lambda x.N:A \rightarrow B$$

M: A $\lambda x.N: A \rightarrow B$ $(\lambda x.N)M: B$

Properties of STLC

- All correctly typed terms can be normalized (their computation terminates)
- · This computational model <u>is weaker</u> that untyped λ -calculus

Let's extend STLC with pairs

$$M:A$$
, $N:B$
 $(M,N):A\times B$
 $\pi_1, \text{fst}:A\times B\to A$
 $\pi_2, \text{snd}:A\times B\to B$
 $\lambda z.(\pi_2 z, \pi_1 z):(B\times A)\to (A\times B)$

Sum Types

```
M:A, N:B

M_L:A+B

N_R:A+B
```

Pattern Matching for Sum Types

$$\begin{array}{ccc} \underline{\mathtt{match}} & V:A+B & \underline{\mathtt{with}} \\ & M_L & \to & use \ M \ of \ type \ A \ and \\ & & return \ value \ of \ type \ C \\ & N_R & \to & use \ N \ of \ type \ B \ and \\ & & return \ value \ of \ type \ C \end{array}$$

Both branches return the same type!

Introducing types to λ -calculus

Detour: Mathematical Logic



Gerhard Gentzen (1909–1945)

- · Natural deduction
- Logical connectors
- Introduction rules
- Elimination rules

Conjunction

$$\frac{A}{A} \wedge \frac{B}{B} \wedge I$$

$$\frac{A \wedge B}{A} \wedge E_l$$

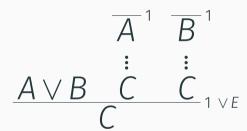
$$\frac{A \wedge B}{B} \wedge E_r$$

Introduction rules for disjunction

$$\frac{A}{A \vee B}^{\vee I_l}$$

$$\frac{B}{A \vee B} \vee B$$

Elimination rule for disjunction



Implication

$$\frac{\overline{A}^{1}}{\overset{\vdots}{B}} \xrightarrow{A \Rightarrow B} \xrightarrow{1 \Rightarrow I}$$

$$A \xrightarrow{A \Rightarrow B} B \Rightarrow E$$

Sample proof:

$$B \wedge A \Rightarrow A \wedge B$$

$$\frac{B \wedge A}{A}^{1}_{\wedge E_{r}} \frac{B \wedge A}{B}^{1}_{\wedge E_{l}}$$

$$\frac{A \wedge B}{B \wedge A \Rightarrow A \wedge B}^{1 \Rightarrow l}$$

Introducing types to λ -calculus

Curry-Howard Correspondence

Logic	Types
Truth	inhabitance
True	any inhabited type, () : ()
False	⊥ (uninhabited type)
proposition A	type, A
proof	term, M: A

Logic	Types
$A \wedge B$ (conjunction)	$A \times B$ (product)
$ \frac{A B}{A \wedge B} \\ \underline{A \wedge B} \\ \underline{A} \qquad \underline{A \wedge B} \\ B $	$ \frac{M:A \qquad N:B}{(M,N):A\times B} \underline{M:A\times B} \qquad \underline{M:A\times B} \underline{\pi_1M:A} \qquad \underline{\pi_2M:B} $

Logic	Types		
$A \lor B$ (disjunction)	A+B (sum, union type)		
$\frac{A}{A \vee B}$ $\frac{B}{A \vee B}$	$\frac{M:A}{M_L:A+B} \qquad \frac{N:B}{N_R:A+B}$		
\overline{A}^{1} \overline{B}^{1}	pattern matching:		
i i	$\underline{\mathtt{match}}$ $V:A+B$ $\underline{\mathtt{with}}$		
$A \lor B$ C C	$M_L \longrightarrow$		
C	$N_R \longrightarrow$		

Logic	Types
$A \Rightarrow B$ (implication)	$A \rightarrow B$ (function type)
$ \begin{array}{c} \overline{A}^{1} \\ \vdots \\ B \\ \overline{A \Rightarrow B} \end{array} $	$ \frac{\overline{x} : A^{-1}}{\vdots} $ $ \frac{N : B}{\lambda x . N : A \to B} $
$\frac{A \qquad A \Rightarrow B}{B}$	$\frac{L:A \qquad M:A \to B}{ML:B}$

function $A \rightarrow \bot$

¬A (negation)

33

Sample program: $(B \times A) \rightarrow (A \times B)$

$$\frac{\overline{Z:B\times A}^{1}}{\pi_{2}Z:A} \frac{\overline{Z:B\times A}^{1}}{\pi_{1}Z:B}$$

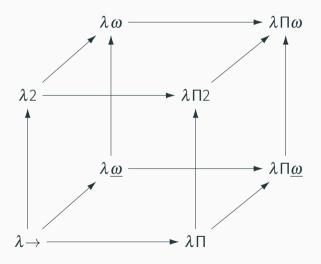
$$(\pi_{2}Z,\pi_{1}Z):A\times B$$

$$\lambda Z.(\pi_{2}Z,\pi_{1}Z):(B\times A)\to (A\times B)$$

$$\frac{\overline{B\wedge A}^{1}}{A} \frac{\overline{B\wedge A}^{1}}{B}$$

$$\frac{\overline{A\wedge B}}{B\wedge A\Rightarrow A\wedge B}$$

λ -cube (Henk Barendregt, 1991)



Logic	Types
Predicate $P(x)$, $x \in X$	dependent type, $T: X \to Type$
$\forall x \in X, P(x)$	$\Pi_{x:X}T(x)$, dependent function
$\exists x \in X, \ P(x)$	$\Sigma_{x:X}T(x)$, dependent pair

PLs with dependent types

- · Coq
- · Agda
- Lean
- Idris
- Arend
- •

$$\begin{array}{c|c}
\hline
B \land A & \hline
A & B \\
\hline
A \land B \\
\hline
B \land A \Rightarrow A \land B
\end{array}$$

$$\frac{\overline{Z:B\times A}}{\underline{\pi_2Z:A}}^1 \quad \overline{Z:B\times A}^1 \\
\underline{\pi_1Z:B} \\
(\pi_2Z,\pi_1Z):A\times B$$

$$\lambda_{Z.}(\pi_2Z,\pi_1Z):(B\times A)\to (A\times B)$$

Example: Lean

```
Logic: proof
example (A B : Prop) : B /\ A -> A /\ B :=
  assume h, and.intro (h.right) (h.left)

Types: term
example (A B : Type) : B × A -> A × B :=
  λ z, (z.2, z.1)
```

Let's remember this:

Programming languages
with rich type systems
are tools for proving theorems.

Type Inference

Type Inference

PCF (Programming Computable Functions)

PCF Syntax

$$t = x$$

$$| n$$

$$| t+t | t-t | t*t | t/t$$

$$| \mathbf{fun} \ \mathsf{X} \rightarrow \mathsf{t} |$$

$$|$$
 let $x = t$ **in** t

(variable)

(numeric literal) (operators)

(function)

(application)

(condition) (recursion)

(let-declaration)

Computing factorial

fix f fun $n \rightarrow \text{ifz } n$ then 1 else n * f(n-1)

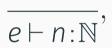
PCF has types

$$T = X$$
 (type variable)
 $| \mathbb{N}$ (type literal)
 $| T \rightarrow T$ (function type)

We can check types

Type relation: $e \vdash t:A$ (term t has type A in environment e):

$$e \vdash x : A$$
 if e contains $x : A$,



$$\frac{e \vdash t: \mathbb{N} \qquad e \vdash u: \mathbb{N}}{e \vdash t \otimes u: \mathbb{N}}$$

$$\frac{e,x:A \vdash t:B}{e \vdash \mathbf{fun} \ x \to t:A \to B},$$

$$\frac{e \vdash u:A \qquad e \vdash t:A \rightarrow B}{e \vdash t u:B}$$

$$\begin{array}{c|cccc}
e \vdash t: \mathbb{N} & e \vdash u: A & e \vdash v: A \\
\hline
e \vdash \mathbf{ifz} \ t \ \mathbf{then} \ u \ \mathbf{else} \ v: A \\
& \underline{e, x: A \vdash t: A} \\
e \vdash \mathbf{fix} \ x \ t: A'
\end{array}$$

$$\begin{array}{c|ccccc}
e \vdash t: A & e, x: A \vdash u: B \\
\hline
e \vdash \mathbf{let} \ x = t \ \mathbf{in} \ u: B
\end{array}$$

Example

$$\vdash$$
 (fun $x \rightarrow 2 + x$) 3: \mathbb{N}

This type system allows several types for a term

fun
$$X \to X$$

 \vdash fun $X \to X : \mathbb{N} \to \mathbb{N}$
 \vdash fun $X \to X : (\mathbb{N} \to \mathbb{N}) \to (\mathbb{N} \to \mathbb{N})$
 \vdash fun $X \to X : X \to X$

- Term is closed (no free variables)
- Type has a free type variable

Type Inference

Hindley-Milner algorithm

Let's infer the type of fun $f \rightarrow 2+f1$

```
\vdash fun f \rightarrow 2 + f 1: ?
                                                   Type equations
                                                   X = \mathbb{N} \to Y
 f: X \vdash 2 + f : ?
 f:X\vdash 2:?
                                                   \mathbb{N} = \mathbb{N}
 f:X\vdash f1:?
                                                   Y = \mathbb{N}
F VI 1 7
```

$$X = \mathbb{N} \to \mathbb{N}, Y = \mathbb{N}$$

$$\vdash$$
 fun $f \rightarrow 2 + f$ 1: $(\mathbb{N} \rightarrow \mathbb{N}) \rightarrow \mathbb{N}$

Hindley algorithm presented above is 2-phase

- 1. Traverse over a term recursively, generate a system of equations: $e \vdash t \rightsquigarrow A$, E.
- 2. Solve system E, substitute a solution to A and get: $e \vdash t:T$.

First phase can be described with a set of rules:

$$\frac{e \vdash x \rightsquigarrow A, \varnothing}{e \vdash n \rightsquigarrow \mathbb{N}, \varnothing}, \frac{e \vdash n \rightsquigarrow \mathbb{N}, \varnothing}{e \vdash u \rightsquigarrow A, E}, \frac{e \vdash t \rightsquigarrow B, F}{e \vdash t \otimes u \rightsquigarrow \mathbb{N}, E \cup F \cup \{A = \mathbb{N}, B = \mathbb{N}\}},$$

$$\frac{e, x: X \vdash t \rightsquigarrow A, E}{e \vdash \mathbf{fun} \ x \rightarrow t \rightsquigarrow X \rightarrow A, E},$$

$$\frac{e \vdash u \rightsquigarrow A, E \qquad e \vdash t \rightsquigarrow B, F}{e \vdash t u \rightsquigarrow X, E \cup F \cup \{B = A \rightarrow X\}},$$

$$\frac{e \vdash t \rightsquigarrow A, E \qquad e \vdash u \rightsquigarrow B, F \qquad e \vdash v \rightsquigarrow C, G}{e \vdash \text{ifz } t \text{ then } u \text{ else } v \rightsquigarrow B, E \cup F \cup G \cup \{A = \mathbb{N}, B = C\}},$$

$$\frac{e, x: X \vdash t \rightsquigarrow A, E}{e \vdash \text{fix } x \ t \rightsquigarrow A, E \cup \{X = A\}},$$

$$\frac{e \vdash t \rightsquigarrow A, E \qquad e, x : A \vdash u \rightsquigarrow B, F}{e \vdash \mathbf{let} \ x = t \ \mathbf{in} \ u \rightsquigarrow B, E \cup F}$$

Robinson algorithm for solving a system of equations:

Shape of equation	Action
$A \rightarrow B = C \rightarrow D$	replace with $A = C$ and $B = D$
$\mathbb{N}=\mathbb{N}$	remove
$\mathbb{N} = A \to B$	ERROR
$A \rightarrow B = \mathbb{N}$	ERROR
X = X	remove
X = A or A = X	if A contains X as a proper part, then
	ERROR
	if A doesn't contain X, then X is re-
	placed with A in all equations

It works!

If this algorithm terminates without an error, then the system becomes:

$$X_1 = A_1, \ldots, X_n = A_n,$$

where X_i — various type expressions, which are not contained in A_i .

More examples

- $\cdot \vdash \mathsf{fun} \ X \to X \leadsto X \to X,$
- $\cdot \vdash (\operatorname{fun} X \to X)(\operatorname{fun} X \to X) \rightsquigarrow Y \to Y,$

Unfortunately, some terms are problematic

let $id = \mathbf{fun} \ X \rightarrow X \ \mathbf{in} \ id \ id$ What's the type of id?

Type Inference Polymorphic types in PCF

Extending types with type schemas with quantors

$$A = X$$

$$| \mathbb{N}$$

$$| A \to A$$

$$S = Y$$

$$| [A]$$

$$| \forall X \leq S$$

Example

fun $X \rightarrow X : \forall X [X \rightarrow X]$

Key ideas

- Programming languages
- Type systems
- Type checking
- Type inference

Recommended reading

- 1. Dowek, Lévy. Introduction to the Theory of Programming Languages (in Russian: Довек, Леви. Введение в теорию языков программирования)
- 2. Pierce. Types and Programming Languages. (in Russian: Пирс. Типы в языках программирования)
- 3. Robert Harper. Practical Foundations for Programming Languages.
- 4. Software Foundations in Coq (interactive textbook): https://softwarefoundations.cis.upenn.edu/