Types in Programming Languages

Guest Lecture at Neapolis University Pafos

Vitaly Bragilevsky (JetBrains)

April 10, 2025

```kotlin
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.
```

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

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

- `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

Arithmetic expressions, functions, and λ-calculus

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)`

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

applicative expressions

Essence

λ-calculus (1930s)

Variables: x, y, z...

Application: MN

Call

Abstraction: $\lambda x.M$

Argument

Body

Function

Computing a λ-term: β-reduction

$$\Big(\lambda x.x(xy)\Big)\underline{u} \longrightarrow_\beta u(uy)$$

λ-calculus is capable of encoding any computation

- Logical values and operations:
  $\lambda x.\lambda y.x$ (True) и $\lambda x.\lambda y.y$ (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
- 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

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

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

$$
\begin{aligned}
M &: A \\
\lambda x.N &: A \to B \\
(\lambda x.N)M &: B
\end{aligned}
$$

Properties of STLC

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

Let's extend STLC with pairs

$$M : A \quad , \quad N : B$$

$$(M, N) \ : \ A \times B$$

$$\pi_1, \texttt{fst} \ : \ A \times B \to A$$

$$\pi_2, \texttt{snd} \ : \ A \times B \to B$$

$$\lambda z.(\pi_2 z, \pi_1 z) \ : \ (B \times A) \to (A \times B)$$

Sum Types

$$M : A \quad , \quad N : B$$

$$M_L \; : \; A + B$$

$$N_R \; : \; A + B$$

Pattern Matching for Sum Types

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

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

$$\frac{A \qquad B}{A \wedge B}\wedge I \qquad \frac{A \wedge B}{A}\wedge E_l \qquad \frac{A \wedge B}{B}\wedge E_r$$

Introduction rules for disjunction

$$\frac{A}{A \vee B}\,\vee I_l \qquad\qquad \frac{B}{A \vee B}\,\vee I_r$$

Elimination rule for disjunction

$$\dfrac{A \vee B \qquad \begin{matrix} \overline{A}^{\,1} \\ \vdots \\ C \end{matrix} \qquad \begin{matrix} \overline{B}^{\,1} \\ \vdots \\ C \end{matrix}}{C}\;1\; \vee E$$

Implication

$$\dfrac{\dfrac{\dfrac{}{A}\,1}{\vdots}\quad B}{A \Rightarrow B}\,1 \Rightarrow I$$

$$\dfrac{A \qquad A \Rightarrow B}{B} \Rightarrow E$$

Sample proof:

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

$$
\dfrac{\dfrac{\dfrac{\dfrac{}{B \wedge A}\,1}{A}\,\wedge E_r \qquad \dfrac{\dfrac{}{B \wedge A}\,1}{B}\,\wedge E_l}{A \wedge B}\,\wedge I}{B \wedge A \Rightarrow A \wedge B}\,1 \Rightarrow I
$$

Introducing types to λ-calculus

Curry–Howard Correspondence

Logic	**Types**
Truth	inhabitance
True	any inhabited type, $() : ()$
False	$\perp$ (uninhabited type)
proposition A	type, A
proof	term, $M : A$

Logic	Types
$A \wedge B$ (conjunction)	$A \times B$ (product)
$\dfrac{A \quad B}{A \wedge B}$	$\dfrac{M : A \quad N : B}{(M, N) : A \times B}$
$\dfrac{A \wedge B}{A} \qquad \dfrac{A \wedge B}{B}$	$\dfrac{M : A \times B}{\pi_1 M : A} \qquad \dfrac{M : A \times B}{\pi_2 M : B}$

Logic	Types
$A \vee B$ (disjunction)	$A + B$ (sum, union type)
$\dfrac{A}{A \vee B} \qquad \dfrac{B}{A \vee B}$	$\dfrac{M : A}{M_L : A + B} \qquad \dfrac{N : B}{N_R : A + B}$
$\dfrac{A \vee B \qquad \begin{array}{c} \overline{A}^{\,1} \\ \vdots \\ C \end{array} \qquad \begin{array}{c} \overline{B}^{\,1} \\ \vdots \\ C \end{array}}{C}$	pattern matching: $\begin{array}{ccc} \underline{\texttt{match}} & V : A + B & \underline{\texttt{with}} \\ M_L & \to & \ldots \\ N_R & \to & \ldots \end{array}$

Logic	Types
$A \Rightarrow B$ (implication)	$A \to B$ (function type)
$$\dfrac{\dfrac{\overline{A}^{\,1}}{\vdots}\\ B}{A \Rightarrow B}$$	$$\dfrac{\dfrac{\overline{x : A}^{\,1}}{\vdots}\\ N : B}{\lambda x.N : A \to B}$$
$$\dfrac{A \qquad A \Rightarrow B}{B}$$	$$\dfrac{L : A \qquad M : A \to B}{M\,L : B}$$
$\neg A$ (negation)	function $A \to \bot$

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

$$\dfrac{\dfrac{\dfrac{\dfrac{}{z : B \times A}\,1}{\pi_2 z : A} \qquad \dfrac{\dfrac{}{z : B \times A}\,1}{\pi_1 z : B}}{(\pi_2 z, \pi_1 z) : A \times B}}{\lambda z.(\pi_2 z, \pi_1 z) : (B \times A) \to (A \times B)}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{B \wedge A}\,1}{A} \qquad \dfrac{\dfrac{}{B \wedge A}\,1}{B}}{A \wedge B}}{B \wedge A \Rightarrow A \wedge B}$$

λ-cube (Henk Barendregt, 1991)

$$
\begin{tikzcd}
& \lambda\omega \arrow[rr] & & \lambda\Pi\omega \\
\lambda 2 \arrow[ur] \arrow[rr] & & \lambda\Pi 2 \arrow[ur] & \\
& \lambda\underline{\omega} \arrow[uu] \arrow[rr] & & \lambda\Pi\underline{\omega} \arrow[uu] \\
\lambda{\to} \arrow[uu] \arrow[ur] \arrow[rr] & & \lambda\Pi \arrow[uu] \arrow[ur] &
\end{tikzcd}
$$

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

PLs with dependent types

- Coq
- Agda
- Lean
- Idris
- Arend
- …

$$\dfrac{\dfrac{\dfrac{}{B \wedge A}\,1}{A} \qquad \dfrac{\dfrac{}{B \wedge A}\,1}{B}}{\dfrac{A \wedge B}{B \wedge A \Rightarrow A \wedge B}}$$

$$\dfrac{\dfrac{\dfrac{}{z : B \times A}\,1}{\pi_2 z : A} \qquad \dfrac{\dfrac{}{z : B \times A}\,1}{\pi_1 z : B}}{\dfrac{(\pi_2 z, \pi_1 z) : A \times B}{\lambda z.(\pi_2 z, \pi_1 z) : (B \times A) \to (A \times B)}}$$

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

$$
\begin{aligned}
t = {} & x && \text{(variable)} \\
\mid {} & n && \text{(numeric literal)} \\
\mid {} & t + t \mid t - t \mid t * t \mid t / t && \text{(operators)} \\
\mid {} & \textbf{fun } x \to t && \text{(function)} \\
\mid {} & t\ t && \text{(application)} \\
\mid {} & \textbf{ifz } t \textbf{ then } t \textbf{ else } t && \text{(condition)} \\
\mid {} & \textbf{fix } x\ t && \text{(recursion)} \\
\mid {} & \textbf{let } x = t \textbf{ in } t && \text{(let-declaration)}
\end{aligned}
$$

Computing factorial

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

PCF has types

$$
\begin{aligned}
T &= X && \text{(type variable)} \\
&\mid \mathbb{N} && \text{(type literal)} \\
&\mid T \to T && \text{(function type)}
\end{aligned}
$$

We can check types

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

$$\frac{}{e \vdash x{:}A}\ \text{if } e \text{ contains } x{:}A,$$

$$\frac{}{e \vdash n : \mathbb{N}},$$

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

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

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

$$\frac{e \vdash t : \mathbb{N} \qquad e \vdash u : A \qquad e \vdash v : A}{e \vdash \textbf{ifz}\ t\ \textbf{then}\ u\ \textbf{else}\ v : A},$$

$$\frac{e, x : A \vdash t : A}{e \vdash \textbf{fix}\ x\ t : A},$$

$$\frac{e \vdash t : A \qquad e, x : A \vdash u : B}{e \vdash \textbf{let}\ x = t\ \textbf{in}\ u : B}.$$

Example

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

This type system allows several types for a term

$$\mathsf{fun}\ x \to x$$

$$\vdash \mathsf{fun}\ x \to x : \mathbb{N} \to \mathbb{N}$$

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

$$\vdash \mathsf{fun}\ x \to x : X \to X$$

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

Type Inference

Hindley–Milner algorithm

Let's infer the type of fun $f \to 2 + f\,1$

$$\vdash \textbf{fun}\ f \to 2 + f\,1 : \color{red}{?}$$

$$f{:}X \vdash 2 + f\,1 : \color{red}{?}$$

$$f{:}X \vdash 2 : \color{red}{?}$$

$$f{:}X \vdash f\,1 : \color{red}{?}$$

$$f{:}X \vdash 1 : \color{red}{?}$$

Type equations

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

$$\mathbb{N} = \mathbb{N}$$

$$Y = \mathbb{N}$$

Solution

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

$$\color{blue}{\vdash \textbf{fun}\ f \to 2 + f\,1 : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}}$$

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:

$$\frac{}{e \vdash x \rightsquigarrow A,\ \varnothing}\ \text{if } e \text{ contains } x{:}A,$$

$$\frac{}{e \vdash n \rightsquigarrow \mathbb{N},\ \varnothing},$$

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

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

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

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

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

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

Robinson algorithm for solving a system of equations:

Shape of equation	Action
$A \to B = C \to D$	replace with $A = C$ and $B = D$
$\mathbb{N} = \mathbb{N}$	remove
$\mathbb{N} = A \to B$	ERROR
$A \to B = \mathbb{N}$	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 replaced 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

- $\vdash \text{fun } x \to x \rightsquigarrow X \to X$,
- $\vdash (\text{fun } x \to x)(\text{fun } x \to x) \rightsquigarrow Y \to Y$,

Unfortunately, some terms are problematic

$$\mathbf{let}\ id = \mathbf{fun}\ x \to x\ \mathbf{in}\ id\ id$$

What's the type of *id*?

Type Inference

Polymorphic types in PCF

Extending types with type schemas with quantors

$$\begin{aligned} A &= X \\ &\mid \mathbb{N} \\ &\mid A \to A \\ S &= Y \\ &\mid [A] \\ &\mid \forall X\ S \end{aligned}$$

Example

$$\text{fun } x \to x : \forall X\ [X \to X]$$

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/`