# Simply-Typed Lambda-Calculus

Jacques Garrigue, 2018/05/29

In the original $\lambda$-calculus, functions have no explicit domain and range, contrary to mathematics. For instance, applying $\mathsf{c}_+$ to something that is not a Church numeral is meaningless, but there is no way to detect it. In typed lambda-calculi, the domain and range of functions are expressed by *types*. In simple versions, any value belongs only to a single type.

## 1 Types and terms

There are two kinds of types: basic types and function or structure types.

$$\begin{array}{rcl} b & ::= & \mathsf{int} \mid \mathsf{bool} \mid \ldots \\ t & ::= & b \mid t \to t \mid t \times t \end{array}$$

We also need to add type information and typed constants to the syntax of terms.

$$M ::= x \mid c_t \mid (\lambda x{:}t.M) \mid (M\ M) \mid (M, M)$$

In combination with the introduction of constants, we add $\delta$-rules to reduce them.

$$\begin{array}{lcl} (\lambda x{:}\tau.M)\ N & \to & [N/x]M \\ \mathsf{fst}_{\tau\times\theta\to\tau}\ (M, N) & \to & M \\ \mathsf{snd}_{\tau\times\theta\to\theta}\ (M, N) & \to & N \\ \mathsf{add}_{\mathsf{nat}\to\mathsf{nat}\to\mathsf{nat}}\ 0_{\mathsf{nat}}\ N & \to & N \\ \mathsf{add}_{\mathsf{nat}\to\mathsf{nat}\to\mathsf{nat}}\ (\mathsf{s}_{\mathsf{nat}\to\mathsf{nat}}\ M)\ N & \to & \mathsf{s}_{\mathsf{nat}\to\mathsf{nat}}\ (\mathsf{add}_{\mathsf{nat}\to\mathsf{nat}\to\mathsf{nat}}\ M\ N) \\ \mathsf{if0}_{\mathsf{nat}\to\tau\to\tau\to\tau}\ 0_{\mathsf{nat}}\ M\ N & \to & M \\ \mathsf{if0}_{\mathsf{nat}\to\tau\to\tau\to\tau}\ (\mathsf{s}_{\mathsf{nat}\to\mathsf{nat}}\ P)\ M\ N & \to & N \\ \ldots & & \end{array}$$

In the above $\delta$-rules, $\mathsf{add}$ and $\mathsf{s}$ have a fixed type, but $\mathsf{fst}$, $\mathsf{snd}$ and $\mathsf{if0}$ have *type schemes*, which represent types sharing the structure. Here $\tau$ and $\theta$ may be replaced by any type.

## 2 Type derivation

The well-formedness of a term is expressed by the following typing judgment.

$$\Gamma \vdash M : \tau$$

$M$ is a term $\tau$ is a type. The typing context $\Gamma$ is a sequence of variables associated to their types $(x_1 : \tau_1, \ldots, x_n : \tau_n)$.

A typing judgment is provable if it can be derived using the type inference rules in figure 1.

| | |
|---|---|
| **Variable** | $\Gamma \vdash x : \tau \quad (x : \tau \in \Gamma)$ |
| **Constant** | $\Gamma \vdash c_\tau : \tau$ |
| **Abstraction** | $\dfrac{\Gamma, x : \theta \vdash M : \tau}{\Gamma \vdash \lambda x{:}\theta.M : \theta \to \tau}$ |
| **Application** | $\dfrac{\Gamma \vdash M : \theta \to \tau \quad \Gamma \vdash N : \theta}{\Gamma \vdash (M\ N) : \tau}$ |
| **Product** | $\dfrac{\Gamma \vdash M : \tau \quad \Gamma \vdash N : \theta}{\Gamma \vdash (M, N) : \tau \times \theta}$ |

Figure 1: Type inference rules

**Example of type derivation**

$$\dfrac{\dfrac{\dfrac{x : \mathsf{nat} \vdash \mathsf{s}_{\mathsf{nat}\to\mathsf{nat}} : \mathsf{nat} \to \mathsf{nat} \quad x : \mathsf{nat} \vdash x : \mathsf{nat}}{x : \mathsf{nat} \vdash (\mathsf{s}_{\mathsf{nat}\to\mathsf{nat}}\ x) : \mathsf{nat}}}{\vdash \lambda x{:}\mathsf{nat}.(\mathsf{s}_{\mathsf{nat}\to\mathsf{nat}}\ x) : \mathsf{nat} \to \mathsf{nat}} \qquad \vdash 0_{\mathsf{nat}} : \mathsf{nat}}{\vdash ((\lambda x{:}\mathsf{nat}.(\mathsf{s}_{\mathsf{nat}\to\mathsf{nat}}\ x))\ 0_{\mathsf{nat}}) : \mathsf{nat}}$$

**Exercise 1** *Write the type derivation for*

$$\lambda f : \mathsf{nat} \to \mathsf{nat}.\lambda x : \mathsf{nat}.f\ (f\ x)$$

# 3 Properties

The following theorems are essential properties of typed lambda-calculus.

**Theorem 1 (Subject reduction)** *If $\Gamma \vdash M : \tau$ can be derived and $M \to N$, then $\Gamma \vdash N : \tau$ is derivable.*

**Theorem 2 (Termination)** *If $\Gamma \vdash M : \tau$ is derivable for some $\Gamma$ and $\tau$, then there is no infinite reduction starting from $M$.*

Termination suggests that not all programs can be written in the simply-typed lambda-calculus. For instance, the non-terminating term $(\lambda x.x\ x)(\lambda x.x\ x)$ cannot be given a type.

To see this, let's try to build a derivation of $\lambda x : \tau.x\ x$.

$$\dfrac{\dfrac{x : \tau \vdash x : \tau \to \theta \quad x : \tau \vdash x : \tau}{x : \tau \vdash x\ x : \theta}}{\vdash \lambda x : \tau.x\ x : \tau \to \theta}$$

By analysis of the rules, this is the only possible derivation for $\lambda x : \tau.x\ x$. However the **Variable** rule at the top-left requires $\tau = (\tau \to \theta)$, which no $\tau$ of finite size can satisfy ($\tau$ and $\tau \to \theta$ have different sizes).

Similarly, being able to define $Y$ would imply non-termination, so $Y$ cannot be defined.

# 4 Curry-Howard correspondance

If in each type inference rule we remove the term part, keeping only the context and type parts, we obtain inference rules for the (explicit) natural deduction version of intuitionistic propositional logic. Constants become axioms.

$$\mathbf{Ax}\ \frac{A \in \Delta}{\Delta \vdash A} \qquad \supset \mathbf{I}\ \frac{\Delta, A \vdash B}{\Delta \vdash A \supset B} \qquad \supset \mathbf{E}\ \frac{\Delta \vdash A \supset B \quad \Delta \vdash A}{\Delta \vdash B}$$

$$\wedge\mathbf{I}\ \frac{\Delta \vdash A \quad \Delta \vdash B}{\Delta \vdash A \wedge \Delta \vdash B} \qquad \wedge\,\mathbf{E}_1\ \Delta \vdash A \wedge B \supset A \qquad \wedge\,\mathbf{E}_2\ \Delta \vdash A \wedge B \supset B$$

Here $\Delta$ is a sequence of propositions, $\mathbf{I}$ stands for *introduction* and $\mathbf{E}$ for *elimination.*

The following rules describe an equivalent implicit version of natural deduction, which omits $\mathbf{Ax}$ and $\Delta$, subsumed by $\supset\mathbf{I}$ .

$$\supset \mathbf{I}\ \frac{\begin{array}{c}[A]^{(n)}\\ \vdots \\ B\end{array}}{A \supset B}{}^{(n)} \qquad \supset \mathbf{E}\ \frac{A \supset B \quad A}{B}$$

$$\wedge\mathbf{I}\ \frac{A \quad B}{A \wedge B} \qquad \wedge\,\mathbf{E}_1\ A \wedge B \supset A \qquad \wedge\,\mathbf{E}_2\ A \wedge B \supset B$$

For instance, the derivation of $\lambda x : \tau \times \theta.(\mathsf{snd}\ x, \mathsf{fst}\ x)$

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash \mathsf{snd} : \tau \times \theta \to \theta \quad \Gamma \vdash x : \tau \times \theta}{\Gamma \vdash (\mathsf{snd}\ x) : \theta} \quad \dfrac{\vdash \mathsf{fst} : \tau \times \theta \to \tau \quad \Gamma \vdash x : \tau \times \theta}{\Gamma \vdash (\mathsf{fst}\ x) : \tau}}{\Gamma = x : \tau \times \theta \vdash (\mathsf{snd}\ x, \mathsf{fst}\ x) : \theta \times \tau}}{\vdash \lambda x : \tau \times \theta.(\mathsf{snd}\ x, \mathsf{fst}\ x) : \tau \times \theta \to \theta \times \tau}$$

becomes the (implicit) logical proof tree

$$\dfrac{\dfrac{\dfrac{A \wedge B \supset B \quad A \wedge B^{(1)}}{B} \quad \dfrac{A \wedge B \supset A \quad A \wedge B^{(1)}}{A}}{B \wedge A}}{A \wedge B \supset B \wedge A}{}^{(1)}$$

Moreover, this proof can be rebuilt from the term part alone, so that one can say that a term is a proof of its type.

| $\lambda$-calculus | logic |
|---|---|
| Type | Proposition |
| Type derivation | Proof |
| Term | Proof |
| $\to$ | $\supset$ |
| $\times$ | $\wedge$ |

**Exercise 2** *Write a type derivation for $\lambda x : A.\lambda y : B.x$, and the corresponding logical proof tree (explicit or implicit).*