Typed Lambda Calculus

Jacques Garrigue, 2012.07.02

Contrary to mathematical functions, λ-calculus does not define a function's domain and range. For instance, c_+ has no meaning if its arguments are not Church numerals, but there is no way to make this explicit.

1 Types and terms

Typed λ-calculi use *types* in place of sets. In the simply typed λ-calculus, each value belongs to a single type. There are two kinds of types: base types, and functional/structural types.

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

Type information also appears inside λ-terms.

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

In order to manipulate values other than functions, δ-rules are introduced in addition to β-reduction.

$$\begin{array}{lcl} (\lambda x{:}\tau.M)\ N & \to & [N/x]M \\ (\mathsf{fst}_{\tau\times\theta\to\tau}\ (M,N)) & \to & M \\ (\mathsf{snd}_{\tau\times\theta\to\theta}\ (M,N)) & \to & N \\ (\mathsf{s}_{\mathsf{nat}\to\mathsf{nat}}\ n_{\mathsf{nat}}) & \to & (n+1)_{\mathsf{nat}} \\ (\mathsf{if0}_{\mathsf{nat}\to\tau\to\tau\to\tau}\ 0_{\mathsf{nat}}\ M\ N) & \to & M \\ (\mathsf{if0}_{\mathsf{nat}\to\tau\to\tau\to\tau}\ n_{\mathsf{nat}}\ M\ N) & \to & N \\ \ldots & & \end{array}$$

In the above rules, s has a unique type, but fst, snd and $\mathsf{if0}$ can be used with several types. τ and θ represent types that the user can choose as needed.

2 Typing derivation

The following judgment states that M is well-typed.

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

M and τ are respectively a λ-term and its type. Γ is a *typing evironment*, associating variables to their types; it has the form: $\{x_1 : \tau_1, \ldots, x_n : \tau_n\}$.

A typing judgment is correct when it can be derived from the following typing rules.

Variable $\quad \Gamma \vdash x : \tau \quad (x : \tau \in \Gamma)$

Constant $\quad \Gamma \vdash c_\tau : \tau$

$$\textbf{Abstraction}\quad \frac{\Gamma, x : \theta \vdash M : \tau}{\Gamma \vdash \lambda x{:}\theta.M : \tau}$$

$$\textbf{Application}\quad \frac{\Gamma \vdash M : \theta \to \tau \quad \Gamma \vdash N : \theta}{\Gamma \vdash (M\ N) : \tau}$$

$$\textbf{Product}\quad \frac{\Gamma \vdash M : \tau \quad \Gamma \vdash N : \theta}{\Gamma \vdash (M, N) : \tau \times \theta}$$

Example 1 (derivation)

$$\dfrac{\dfrac{\dfrac{x : \mathsf{nat} \vdash \mathsf{s}_{\mathsf{nat}\to\mathsf{nat}} : \mathsf{nat} \to \mathsf{nat} \quad x : \mathsf{nat} \vdash x : \mathsf{nat}}{x : \mathsf{nat} \vdash (\mathsf{s}_{\mathsf{nat}\to\mathsf{nat}}\ x) : \mathsf{nat}}}{\vdash \lambda x{:}\mathsf{nat}.(\mathsf{s}_{\mathsf{nat}\to\mathsf{nat}}\ x) : \mathsf{nat} \to \mathsf{nat}} \qquad \vdash 1_{\mathsf{nat}} : \mathsf{nat}}{\vdash ((\lambda x{:}\mathsf{nat}.(\mathsf{s}_{\mathsf{nat}\to\mathsf{nat}}\ x))\ 1_{\mathsf{nat}}) : \mathsf{nat}}$$

Properties

The following properties are stated for the simply typed λ calculus with only δ-rules for fst and snd.

Theorem 1 (subject reduction) *Whenever $\Gamma \vdash M : \tau$ and $M \to N$ are valid, $\Gamma \vdash N : \tau$ is valid.*

Theorem 2 (termination) *If $\Gamma \vdash M : \tau$, then there is no infinite reduction sequence ($M \to M_1 \to M_2 \to \ldots$).*

Having termination means that some otherwise computable functions cannot be defined. For instance, the term $(\lambda x.x\ x)(\lambda x.x\ x)$ was definable in untyped λ-calculus, but it cannot be typed. Let's try to build a derivation:

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

From the structure of the term, this is the only possible shape for a derivation, but it requires that $\tau = (\tau \to \theta)$. According to our definition of types, this equation has no solution (it would require infinite types).

Similarly, if we introduce a term Y, we can define non-terminating computations, so by contradiction there is no typable version of Y in the simply typed λ-calculus.

3 Relation to Logic

If we only look at types in derivations, we obtain valid derivations for *intuitionistic logic*, which motivated the λ-calculus. In such a system, constants behave as axioms.

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

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

we obtain the following proof in intuitionistic logic:

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

Since we can mechanically find a unique derivation for any well-typed λ-term, we can view it as a proof.

The following relation is called the *Curry-Howard isomorphism*.

λ-calculus	Logic
Type	Proposition
λ-Term	Proof
$\to$	$\Rightarrow$
$\times$	$\wedge$
$+$	$\vee$

4 Universality

As we have seen above, if we limit ourselves to the above definition the typed λ-calculus is not universal. This is due to two different reasons.

The addition of numbers through δ-rules is necessary because we cannot encode Church numerals. More precisely, we can type them, but not in a sufficiently general way.

$$\vdash \lambda f : \tau \to \tau.\lambda x : \tau.f^n\ x : (\tau \to \tau) \to \tau \to \tau$$

The trouble is that in order to define the above term, we need to choose a specific τ. Since Church numerals need to be used with different kinds of f and x, this encoding proves insufficient. This can be solved by the addition of the above δ-rules.

For fixpoints, the problem is different: we cannot define them for any type. But the solution is similar: we can add a δ-rule.

$$\mathsf{Y}_{(\tau\to\tau)\to\tau}\ M \to M\ (\mathsf{Y}_{(\tau\to\tau)\to\tau}\ M)$$

If we add natural numbers and Y, λ-calculus becomes universal. Evaluation can be defined either using directly δ-rules, or through a translation to untyped λ-calculus.

With a stronger type system it becomes possible to encode Church numerals directly. Second-order λ-calculus introduces type variables.

$$\begin{array}{rcl} t & ::= & \ldots \mid \tau \mid \forall\tau.t \\ M & ::= & \ldots \mid \Lambda\tau.M \mid M[t] \end{array}$$

Using them we can encode Church numerals as follows.

$$\begin{array}{l}
\vdash \mathsf{c}_n = \Lambda\tau.\lambda f : \tau \to \tau.\lambda x : \tau.f^n x : \forall\tau.(\tau \to \tau) \to \tau \to \tau \\
\vdash \mathsf{c}_+ = \lambda m : \mathit{Nat}.\lambda n : \mathit{Nat}.\Lambda\tau.\lambda f : \tau \to \tau.\lambda x : \tau.(m[\tau]\ x\ (n[\tau]\ f\ x)) : \mathit{Nat} \to \mathit{Nat} \to \mathit{Nat} \\
\vdash \mathsf{c}_\times = \lambda m : \mathit{Nat}.\lambda n : \mathit{Nat}.\Lambda\tau.\lambda f : \tau \to \tau.(m[\tau]\ (n[\tau]\ f)) : \mathit{Nat} \to \mathit{Nat} \to \mathit{Nat} \\
\vdash \mathsf{c}_{\mathsf{exp}} = \lambda m : \mathit{Nat}.\lambda n : \mathit{Nat}.\Lambda\tau.n[\tau \to \tau]\ (m[\tau]) : \mathit{Nat} \to \mathit{Nat} \to \mathit{Nat}
\end{array}$$

Here $Nat = \forall\tau.(\tau \to \tau) \to \tau \to \tau$.

However, second-order λ-calculus still guarantees termination, and as a result we cannot encode Y in it. This is not necessarily a weakness: if all computations terminate, then we can give a concrete meaning to all terms.

We could also choose to extend λ-calculus with recursive types. They allow to solve equations such as $\tau = (\tau \to \theta)$, and are sufficient to give a type to Y. However, we then loose termination.