A review of the simply typed λ-calculus

Andrej Bauer

March 24, 2022

The λ-calculus is the abstract theory of functions, just like group theory is the abstract theory of symmetries. There are two basic operations that can be performed with functions. The first one is the *application* of a function to an argument: if f is a function and a is an argument, then fa is the application of f to a. The second operation is *abstraction*: if x is a variable and t is an expression in which x may appear, then there is a function f defined by

$$fx = t\ .$$

Here we gave the name f to the newly formed function. But we could have expressed the same function without giving it a name; this is usually written as

$$x \mapsto t\ ,$$

and it means "x is mapped to t". In λ-calculus we use a different notation, which is more convenient when abstractions are nested:

$$\lambda x.\, t\ .$$

This operation is called λ-*abstraction*. For example, $\lambda x.\, \lambda y.\, (x + y)$ is the function which maps an argument a to the function $\lambda y.\, (a + y)$.

In an expression $\lambda x.\, t$ the variable x is *bound* in t.

There are two kinds of λ-calculus, the *typed* and the *untyped* one. In the untyped version there are no restrictions on how application is formed, so that an expression such as

$$\lambda x.\, (xx)$$

is valid, whatever it means. In typed λ-calculus every expression has a *type*, and there are rules for forming valid expressions and types. For example, we can only form an application f, a when a has a type A and f has a type $A \to B$, which indicates a function taking arguments of type A and giving results of type B. The judgment that expression t has a type A is written as

$$t : A\ .$$

To computer scientists the idea of expressions having types is familiar from programming languages, whereas mathematicians can think of types as sets and read $t : A$ as $t \in A$. We will concentrate on the typed λ-calculus.

We now give a precise definition of what constitutes a *simply-typed λ-calculus*. First, we are given a set of *simple types*, which are generated from *basic types* by formation of products and function types:

$$\begin{aligned} \text{Basic type } \mathtt{B} &::= \mathtt{B}_0 \mid \mathtt{B}_1 \mid \mathtt{B}_2 \cdots \\ \text{Simple type } A &::= \mathtt{B} \mid A_1 \times A_2 \mid A_1 \to A_2. \end{aligned}$$

Function types associate to the right:

$$A \to B \to C \equiv A \to (B \to C)\ .$$

We assume there is a countable set of *variables* x, y, u, ... We are also given a set of *basic constants*. The set of *terms* is generated from variables and basic constants by the following grammar:

$$\begin{aligned} \text{Variable } v &::= x \mid y \mid z \mid \cdots \\ \text{Constant } c &::= \mathtt{c}_1 \mid \mathtt{c}_2 \mid \cdots \\ \text{Term } t &::= v \mid c \mid * \mid \langle t_1, t_2 \rangle \mid \mathtt{fst}\, t \mid \mathtt{snd}\, t \mid t_1\, t_2 \mid \lambda x : A\,.\, t \end{aligned}$$

In words, this means:

1. a variable is a term,
2. each basic constant is a term,
3. the constant $*$ is a term, called the *unit*,
4. if u and t are terms then $\langle u, t \rangle$ is a term, called a *pair*,
5. if t is a term then $\mathtt{fst}\, t$ and $\mathtt{snd}\, t$ are terms,
6. if u and t are terms then $u\, t$ is a term, called an *application*
7. if x is a variable, A is a type, and t is a term, then $\lambda x : A\,.\, t$ is a term, called a *λ-abstraction*.

The variable x is *bound* in $\lambda x : A \,.\, t$. Application associates to the left, thus $s\,t\,u = (s\,t)\,u$. The *free variables* $\mathsf{FV}(t)$ of a term t are computed as follows:

$$\begin{aligned}
\mathsf{FV}(x) &= \{x\} \qquad \text{if } x \text{ is a variable} \\
\mathsf{FV}(a) &= \emptyset \qquad \text{if } a \text{ is a basic constant} \\
\mathsf{FV}(\langle u, t\rangle) &= \mathsf{FV}(u) \cup \mathsf{FV}(t) \\
\mathsf{FV}(\mathtt{fst}\, t) &= \mathsf{FV}(t) \\
\mathsf{FV}(\mathtt{snd}\, t) &= \mathsf{FV}(t) \\
\mathsf{FV}(u\, t) &= \mathsf{FV}(u) \cup \mathsf{FV}(t) \\
\mathsf{FV}(\lambda x .\, t) &= \mathsf{FV}(t) \setminus \{x\} \;.
\end{aligned}$$

If x_1, …, x_n are *distinct* variables and A_1, …, A_n are types then the sequence

$$x_1 : A_1, \ldots, x_n : A_n$$

is a *typing context*, or just *context*. The empty sequence is sometimes denoted by a dot $\cdot$, and it is a valid context. Context are denoted by capital Greek letters Γ, Δ, …

A *typing judgment* is a judgment of the form

$$\Gamma \mid t : A$$

where Γ is a context, t is a term, and A is a type. In addition the free variables of t must occur in Γ, but Γ may contain other variables as well. We read the above judgment as "in context Γ the term t has type A". Next we describe the rules for deriving typing judgments.

Each basic constant c_i has a uniquely determined type C_i,

$$\frac{}{\Gamma \mid \mathsf{c}_i : C_i}$$

The type of a variable is determined by the context:

$$\frac{}{x_1 : A_1, \ldots, x_i : A_i, \ldots, x_n : A_n \mid x_i : A_i}\ (1 \le i \le n)$$

The constant $*$ has type $\mathtt{1}$:

$$\frac{}{\Gamma \mid * : \mathtt{1}}$$

The typing rules for pairs and projections are:

$$\frac{\Gamma \mid u : A \qquad \Gamma \mid t : B}{\Gamma \mid \langle u, t\rangle : A \times B} \qquad \frac{\Gamma \mid t : A \times B}{\Gamma \mid \mathtt{fst}\, t : A} \qquad \frac{\Gamma \mid t : A \times B}{\Gamma \mid \mathtt{snd}\, t : B}$$

The typing rules for application and λ-abstraction are:

$$\frac{\Gamma \mid t : A \to B \qquad \Gamma \mid u : A}{\Gamma \mid t\,u : B} \qquad\qquad \frac{\Gamma, x : A \mid t : B}{\Gamma \mid (\lambda x : A\,.\,t) : A \to B}$$

Lastly, we have *equations* between terms; for terms of type A in context Γ,

$$\Gamma \mid u : A\,, \qquad\qquad \Gamma \mid t : B\,,$$

the judgment that they are equal is written as

$$\Gamma \mid u = t : A\,.$$

Note that u and t necessarily have the same type; it does *not* make sense to compare terms of different types. We have the following rules for equations:

1. Equality is an equivalence relation:

$$\frac{}{\Gamma \mid t = t : A} \qquad \frac{\Gamma \mid t = u : A}{\Gamma \mid u = t : A} \qquad \frac{\Gamma \mid t = u : A \qquad \Gamma \mid u = v : A}{\Gamma \mid t = v : A}$$

2. The weakening rule:

$$\frac{\Gamma \mid u = t : A}{\Gamma, x : B \mid u = t : A}$$

3. Unit type:

$$\frac{}{\Gamma \mid t = * : \mathtt{1}}$$

4. Equations for product types:

$$\frac{\Gamma \mid u = v : A \qquad \Gamma \mid s = t : B}{\Gamma \mid \langle u, s\rangle = \langle v, t\rangle : A \times B}$$

$$\frac{\Gamma \mid s = t : A \times B}{\Gamma \mid \mathtt{fst}\,s = \mathtt{fst}\,t : A} \qquad\qquad \frac{\Gamma \mid s = t : A \times B}{\Gamma \mid \mathtt{snd}\,s = \mathtt{snd}\,t : A}$$

$$\frac{}{\Gamma \mid t = \langle \mathtt{fst}\,t, \mathtt{snd}\,t\rangle : A \times B}$$

$$\frac{}{\Gamma \mid \mathtt{fst}\,\langle u, t\rangle = u : A} \qquad\qquad \frac{}{\Gamma \mid \mathtt{snd}\,\langle u, t\rangle = t : A}$$

5. Equations for function types:

$$\frac{\Gamma \mid s = t : A \to B \qquad \Gamma \mid u = v : A}{\Gamma \mid s\,u = t\,v : B}$$

$$\frac{\Gamma, x : A \mid t = u : B}{\Gamma \mid (\lambda x : A\,.\,t) = (\lambda x : A\,.\,u) : A \to B}$$

$$\frac{}{\Gamma \mid (\lambda x : A\,.\,t)u = t[u/x] : A} \qquad (\beta\text{-rule})$$

$$\frac{}{\Gamma \mid \lambda x : A\,.\,(t\,x) = t : A \to B} \;\text{ if } x \notin \mathsf{FV}(t) \qquad (\eta\text{-rule})$$

This completes the description of a simply-typed λ-calculus.

Apart from the above rules for equality we might want to impose additional equations. In this case we do not speak of a λ-calculus but rather of a *λ-theory*. Thus, a λ-theory $\mathbb{T}$ is given by a set of basic types, a set of basic constants, and a set of *equations* of the form

$$\Gamma \mid u = t : A \,.$$

We summarize the preceding definitions.

Definition 1 A *simply-typed λ-calculus* is given by a set of *basic types* and a set of *basic constants* together with their types. *A simply-typed λ-theory* is a simply-typed λ-calculus together with a set of equations.

We use letters $\mathbb{S}$, $\mathbb{T}$, $\mathbb{U}$, ... to denote theories.

Example 2 The theory of a group is a simply-typed λ-theory. It has one basic type $\mathtt{G}$ and three basic constant, the unit $\mathtt{e}$, the inverse $\mathtt{i}$, and the group operation $\mathtt{m}$,

$$\mathtt{e} : \mathtt{G}\,, \qquad \mathtt{i} : \mathtt{G} \to \mathtt{G}\,, \qquad \mathtt{m} : \mathtt{G} \times \mathtt{G} \to \mathtt{G}\,,$$

with the following equations:

$$\begin{aligned}
x : \mathtt{G} &\mid \mathtt{m}\langle x, \mathtt{e}\rangle = x : \mathtt{G} \\
x : \mathtt{G} &\mid \mathtt{m}\langle \mathtt{e}, x\rangle = x : \mathtt{G} \\
x : \mathtt{G} &\mid \mathtt{m}\langle x, \mathtt{i}\,x\rangle = \mathtt{e} : \mathtt{G} \\
x : \mathtt{G} &\mid \mathtt{m}\langle \mathtt{i}\,x, x\rangle = \mathtt{e} : \mathtt{G} \\
x : \mathtt{G}, y : \mathtt{G}, z : \mathtt{G} &\mid \mathtt{m}\langle x, \mathtt{m}\langle y, z\rangle\rangle = \mathtt{m}\langle \mathtt{m}\langle x, y\rangle, z\rangle : \mathtt{G}
\end{aligned}$$

These are just the familiar axioms for a group.

Example 3 In general, any algebraic theory $\mathbb{A}$ determines a λ-theory. There is one basic type $\mathtt{A}$ and for each operation f of arity k there is a basic constant $\mathtt{f} : \mathtt{A}^k \to \mathtt{A}$, where $\mathtt{A}^k$ is the k-fold product $\mathtt{A} \times \cdots \times \mathtt{A}$. It is understood that $\mathtt{A}^0 = \mathtt{1}$. The terms of $\mathbb{A}$ are translated to the terms of the corresponding λ-theory in a straightforward manner. For every axiom $t = u$ of $\mathbb{A}$ the corresponding axiom in the λ-theory is

$$x_1 : \mathtt{A}, \ldots, x_n : \mathtt{A} \mid t = u : \mathtt{A}$$

where $x_1, \ldots, x_n$ are the variables occurring in t and u.

Example 4 The theory of a directed graph is a simply-typed theory with two basic types, $\mathtt{V}$ for vertices and $\mathtt{E}$ for edges, and two basic constant, source $\mathtt{src}$ and target $\mathtt{trg}$,

$$\mathtt{src} : \mathtt{E} \to \mathtt{V} \,, \qquad\qquad \mathtt{trg} : \mathtt{E} \to \mathtt{V} \,.$$

There are no equations.

Example 5 An example of a λ-theory is readily found in the theory of programming languages. The mini-programming language PCF is a simply-typed λ-calculus with a basic type $\mathtt{nat}$ for natural numbers, and a basic type $\mathtt{bool}$ of Boolean values,

$$\text{Basic type } \mathtt{B} ::= \mathtt{nat} \mid \mathtt{bool}.$$

There are basic constants zero $\mathtt{0}$, successor $\mathtt{succ}$, the Boolean constants $\mathtt{true}$ and $\mathtt{false}$, comparison with zero $\mathtt{iszero}$, and for each type A the *conditional* $\mathtt{cond}_A$ and the *fixpoint* operator $\mathtt{fix}_A$. They have the following types:

$$\begin{aligned}
\mathtt{0} &: \mathtt{nat} \\
\mathtt{succ} &: \mathtt{nat} \to \mathtt{nat} \\
\mathtt{true} &: \mathtt{bool} \\
\mathtt{false} &: \mathtt{bool} \\
\mathtt{iszero} &: \mathtt{nat} \to \mathtt{bool} \\
\mathtt{cond}_A &: \mathtt{bool} \to A \to A \\
\mathtt{fix}_A &: (A \to A) \to A
\end{aligned}$$

The equational axioms of PCF are:

$$\begin{gathered}
\cdot \mid \mathtt{iszero}\ \mathtt{0} = \mathtt{true} : \mathtt{bool} \\
x : \mathtt{nat} \mid \mathtt{iszero}\,(\mathtt{succ}\, x) = \mathtt{false} : \mathtt{bool} \\
u : A, t : A \mid \mathtt{cond}_A\, \mathtt{true}\ u\ t = u : A \\
u : A, t : A \mid \mathtt{cond}_A\, \mathtt{false}\ u\ t = t : A \\
t : A \to A \mid \mathtt{fix}_A\ t = t\,(\mathtt{fix}_A\ t) : A
\end{gathered}$$

Example 6 Another example of a λ-theory is the *theory of a reflexive type.* This theory has one basic type $\mathtt{D}$ and two constants

$$\mathtt{r} : \mathtt{D} \to \mathtt{D} \to \mathtt{D} \qquad\qquad \mathtt{s} : (\mathtt{D} \to \mathtt{D}) \to \mathtt{D}$$

satisfying the equation

$$f : \mathtt{D} \to \mathtt{D} \mid \mathtt{r}\,(\mathtt{s}\,f) = f : \mathtt{D} \to \mathtt{D} \tag{1}$$

which says that $\mathtt{s}$ is a section and $\mathtt{r}$ is a retraction, so that the function type $\mathtt{D} \to \mathtt{D}$ is a subspace (even a retract) of $\mathtt{D}$. A type with this property is said to be *reflexive*. We may additionally stipulate the axiom

$$x : \mathtt{D} \mid \mathtt{s}\,(\mathtt{r}\,x) = x : \mathtt{D} \tag{2}$$

which implies that $\mathtt{D}$ is isomorphic to $\mathtt{D} \to \mathtt{D}$.

Untyped λ-calculus We briefly describe the *untyped λ-calculus*. It is a theory whose terms are generated by the following grammar:

$$t ::= v \mid t_! \, t_2 \mid \lambda x.\, t\ .$$

In words, a variable is a term, an application $t\,t'$ is a term, for any terms t and t', and a λ-abstraction $\lambda x.\, t$ is a term, for any term t. Variable x is bound in $\lambda x.\, t$. A *context* is a list of distinct variables,

$$x_1, \ldots, x_n\ .$$

We say that a term t is valid in context Γ if the free variables of t are listed in Γ. The judgment that two terms u and t are equal is written as

$$\Gamma \mid u = t\ ,$$

where it is assumed that u and t are both valid in Γ. The context Γ is not really necessary but we include it because it is always good practice to list the free variables.

The rules of equality are as follows:

1. Equality is an equivalence relation:

$$\frac{}{\Gamma \mid t = t} \qquad \frac{\Gamma \mid t = u}{\Gamma \mid u = t} \qquad \frac{\Gamma \mid t = u \qquad \Gamma \mid u = v}{\Gamma \mid t = v}$$

2. The weakening rule:

$$\frac{\Gamma \mid u = t}{\Gamma, x \mid u = t}$$

3. Equations for application and λ-abstraction:

$$\frac{\Gamma \mid s = t \qquad \Gamma \mid u = v}{\Gamma \mid s\,u = t\,v} \qquad \frac{\Gamma, x \mid t = u}{\Gamma \mid \lambda x.\, t = \lambda x.\, u}$$

$$\frac{}{\Gamma \mid (\lambda x.\, t)u = t[u/x]} \tag{β-rule}$$

$$\frac{}{\Gamma \mid \lambda x.\, (t\, x) = t} \text{ if } x \notin \mathsf{FV}(t) \tag{η-rule}$$

The untyped λ-calculus can be translated into the theory of a reflexive type from Example 6. An untyped context Γ is translated to a typed context Γ^* by typing each variable in Γ with the reflexive type $\mathtt{D}$, i.e., a context $x_1, \dots, x_k$ is translated to $x_1 : \mathtt{D}, \dots, x_k : \mathtt{D}$. An untyped term t is translated to a typed term t^* as follows:

$$\begin{aligned} x^* &= x \qquad \text{if } x \text{ is a variable} , \\ (u\,t)^* &= (\mathtt{r}\, u^*)t^* , \\ (\lambda x.\, t)^* &= \mathtt{s}\,(\lambda x : \mathtt{D}\,.\, t^*) . \end{aligned}$$

For example, the term $\lambda x.\,(x\,x)$ translates to $\mathtt{s}\,(\lambda x : \mathtt{D}\,.\,((\mathtt{r}\,x)\,x))$. A judgment

$$\Gamma \mid u = t \tag{3}$$

is translated to the judgment

$$\Gamma^* \mid u^* = t^* : \mathtt{D} . \tag{4}$$

Exercise* **7** Prove that if equation (3) is provable then equation (4) is provable as well. Identify precisely at which point in your proof you need to use equations (1) and (2). Does provability of (4) imply provability of (3)?