An implementable definition of α-equivalence in λ-calculus

Functional Programming Project

Fabio Brau

January 25, 2021

Contents

1 Introduction

This project's aim is to provide a formal definition of α-equivalence on the λ-terms without seeing it as a transitive closure of a relation and without assuming conventions on the name of the variables.

2 Lambda Terms

From an intuitive point of view, a λ-term is a representation of a mathematical function written as a combination of *variables*. It is quite surprising to notice that "a systematic notation for functions is lacking in ordinary mathematics" [Curry]. In fact, the meaning of $f(x)$, that is the usual accepted notation to indicate a function (*Euler Notation*), is not uniquely determined and has to be deduced from the context. Sometimes, with this notation, we refer to a function depending on the variable x; sometimes we refers to the evaluation of the function in a value (or in a point, a vector, a matrix, a set and so on) equal to x.

An unambiguous way to indicate that the function f depends on the variable x could be the notation: $x \mapsto f(x)$. And, we can use the notation $f(x)$ when we want to evaluate f in the object x. The introduction of the symbol λ could be helpful, in a typographic sense, by shortening the notation $x \mapsto f(x)$ in $\lambda x.f(x)$.

Definition 1. Let be $V = \{v_1, v_2, \dots\}$ an infinite set of *variables*. The set of the λ-terms, indicated with Λ, is recursively defined as follow

- $V \subseteq \Lambda$, i.e., each variable is a λ-term;
- If M is a λ-term, then $\lambda x.M \in \Lambda$, for any variable x, is a λ-term called *abstraction*;
- If M, N are λ-terms then (MN) is a new λ-term called *application*.

It's compulsory to observe that the symbols $M, N, \dots, v_0, v_1, \dots$ have to be intended as names of λ-terms. The *assignment* of a name to a λ-term, indicated with $=$, only holds and has sense in the metalanguage. For example, in the formula $M = \lambda x.x$, the symbol M is just the label of the λ-term $\lambda x.x$.

By assuming that the abstraction and the application are associative on the left (as operators of λ-terms in the metalanguage) we can introduce the following notation.

Notation 1. Let x, y be two variables and M a λ-term, then the formula $\lambda xy.M$ is a shorter notation for $\lambda x.\lambda y.M$ that, by the left associative convention, represents uniquely $\lambda x.(\lambda y.M)$. Similarly, if X, Y, Z are λ-terms, then the formula XYZ uniquely represents $((XY)Z)$ by the left associative rule.

Definition 2. We will say that two λ-terms M, N are *syntactically equivalents*, and we will write $M \equiv N$, if each one can be mutually translated into the other trough the notation Notation 1.

2.1 Bound and Free Variables

Let us consider the λ-expression $M = y(\lambda x.x)$. It is evident that the two variable x, y has a different meaning in the λ-term. The variable x, that appears in M under the scope λ, is *bound* and is not a constant of the λ-term. In a certain sense, that it will be more clear later, the variable x can be substituted with another variable without that M loses her meaning. Differently the variable y is *free* and stores a different information that characterizes the term M. Observe, for example, that the term M could represent the formula $\int_0^y x\,dx$, in which the variable x is usually called "silent".

Definition 3. Given a λ-term M, the set $\mathcal{F}(M)$ of the *free variables* of M is recursively defined as follow:

- If $M = x$, where $x \in V$, then $\mathcal{F}(M) = \{x\}$;
- If $M = \lambda x.M_1$, then $\mathcal{F}(M) = \mathcal{F}(M_1) \setminus \{x\}$;
- If $M = M_1M_2$, then $\mathcal{F}(M) = \mathcal{F}(M_1) \cup \mathcal{F}(M_2)$.

Analogously, the set $\mathcal{B}(M)$ of the *bound variables* of M is recursively defined as follow:

- If $M = x$, where $x \in V$, then $\mathcal{B}(M) = \emptyset$;
- If $M = \lambda x.M_1$, then $\mathcal{B}(M) = \mathcal{B}(M_1) \cup \{x\}$;
- If $M = M_1M_2$, then $\mathcal{B}(M) = \mathcal{B}(M_1) \cup \mathcal{B}(M_2)$;

By induction on the construction of Λ, it is easy to prove that the definition above is well-placed.

Observation 1. Observe that, by the definition above, a variable x can be either a free and a bound variable of a λ-term M. For example, the λ-term $M = x(\lambda x.xx)$ is such that $\mathcal{F}(M) = \mathcal{B}(M) = \{x\}$.

2.2 Substitution without Capture

Let us consider a λ-term M and x a free variable of M. In this section we want to define the *substitution operation* in order to substitute the variable x with another λ-term N in M.

The following definition was first proposed by [Curry] in order to avoid the issue known as *binding of a free variable.*

Definition 4 (Substitution)**.** Let $M, N \in \Lambda$, let us define the operation of *substitution* that acts by substituting a variable x in M with N, returning a new λ-term $M[x := N]$. The definition is recursive on the construction of Λ.

Case 1 If M is a variable:

- If $M = x$, then $M[x := N] \equiv N$;
- If $M = y$ and $y \neq x$, then $M[x := N] \equiv y$;

Case 2 If $M = M_1M_2$ is an application:

- $M[x := N] \equiv (M_1[x := N])(M_2[x := N])$;

Case 3 If M is an abstraction:

- If $M = \lambda x.M_1$, then $M[x := N] \equiv M$;

- If $M = \lambda y.M_1$ and $x \neq y$, then

$$M[x := N] \equiv \lambda z.M_1[y := z][x := N]$$

  where: $z = y$ if $x \notin \mathcal{F}(M_1)$ or $y \notin \mathcal{F}(N)$; z is choose to be not in M and not in N otherwise;

The first and the second case are intuitive. Regarding the third case few more words are required. If M is an abstraction of the form $\lambda x.M_1$, substituting x with N has no meaning because x is bound. Differently, if we want to substitute x in a λ-term of the form $\lambda y.M_1$ (and $x \neq y$), then consider the following example. Let $M = \lambda y.x$ and $N = y$. Observe that y is free in N and is bound in M, and if we change x with N without introducing a new variable z we will obtain $\lambda y.y$ that is not equivalent, in a sense that it will be more clear later, to the desired result. This phenomena is called *binding* of a free variable.

Lemma 1. *For any $M, N \in \Lambda$, the following statements hold.*

1. *If $x \notin \mathcal{F}(M)$ and $x \notin \mathcal{B}(M)$, then $M[x := N] \equiv M$;*
2. *If $y \notin \mathcal{F}(M)$ and $y \notin \mathcal{B}(M)$, then $M \equiv M[x := y][y := x]$;*
3. *If $x \in \mathcal{F}(M)$ and $y \notin \mathcal{F}(M)$, then $x \notin \mathcal{F}(M[x := y])$ e $y \in \mathcal{F}(M[x := y])$;*

Proof. Each statement can be proved by (strong) induction on the construction of the λ-terms. The key idea is to observe that $\Lambda = \cup_{n=0}^{\infty}\Lambda_n$ where: $\Lambda_0 = V$ and Λ_n contains all the λ-terms that can be generated by application or abstraction of λ-terms in Λ_k with $k < n$. After that, by supposing that independently $1, 2, 3$ hold for Λ_k with $k < n$, it is easy by applying the Definition 4 to prove that the statement hold also for Λ_n. $\square$

3 α-equivalenza

From an intuitive point of view two λ-terms are α-equivalents if each one can be translated to the other by changing one or more bound variables. This definition hides a number of complications, and it is not easy to state it in a formal manner.

Church solves the problem by including the α-equivalence in a semantic level by adding the statement $\lambda x.M = \lambda z.M[x := z]$ (called α) in the list of the *conversion rules*. Moreover, he required different assumptions on the construction of the λ-terms, for example in his formulation only abstraction trough free variables was accepted ($\lambda y.x$ is not considered a λ-term).

In Curry's formulation [Curry] a different idea is used:

1. The definition of α-equivalence is provided for abstractions by stating $\lambda x.M \equiv_\alpha \lambda z.M[x := z]$.
2. The relation $\equiv_\alpha$ is then extended to the whole λ-terms by transitive closure.

In [Barendregt], the definition of α-equivalence is included in the syntactic equivalence and it is defined informally as: "M is α-congruent to N, if N results from M by a series of changes of the bound variables". Moreover, the author needs a variable convention: When two terms are in the same statement (or definition, theorem, etc.) then different names for the variables have to be used.

Even if the definition above are totally valid, by a practical point of view they contains two uncomfortable things: They don't provide an operative definition (Curry). They use conventions on the terms (Church, Berendregt). The following lines provide an operative definition of α-equivalence that doesn't requires further conventions or assumption and that can be easily implemented trough a functional paradigm.

Definition 5 (α-equivalence)**.** Let us define $\equiv_\alpha$ recursively and in parallel with the construction of Λ. Let Λ_0 be the set of the terms of rank 0 (i.e, of the form x for each variable in V). Then it holds

$$\forall x, y \in \Lambda_0, \quad x \equiv_\alpha y \iff x = y.$$

Let us consider the set Λ_k of the λ-terms of rank k defined as $\Lambda_k = \hat{\Lambda}_k \cup \bar{\Lambda}_k$ where

$$\hat{\Lambda}_k = \{\lambda x.M \ : M \in \Lambda_{k-1},\, x \in V\}$$
$$\bar{\Lambda}_k = \{(MN), (NM) \ : \ M \in \Lambda_{k-1},\, N \in \Lambda_i,\, i < k\}$$

Let us define over this set the relation $\equiv_\alpha$ as follow

- If $M, N \in \hat{\Lambda}_k$, with $M = \lambda x.M_1$ and $N = \lambda y.N_1$, then
  - If $x = y$, then
  $$M \equiv_\alpha N \iff M_1 \equiv_\alpha N_1$$
  - If $x \neq y$, then
  $$M \equiv_\alpha N \iff M_1 \equiv_\alpha N_1[y := x] \quad \wedge \quad (y \notin \mathcal{F}(M_1) \wedge x \notin \mathcal{F}(N_1))$$
- If $M, N \in \bar{\Lambda}_k$, with $M = M_1M_2$ and $N = N_1N_2$, then
$$M \equiv_\alpha N \iff M_1 \equiv_\alpha N_1 \wedge M_2 \equiv_\alpha N_2$$

Observing that $\Lambda = \cup_{k=0}^{\infty} \Lambda_k$, and observing that $\Lambda_i \cap \Lambda_j = \emptyset$, we can extend $\equiv_\alpha$ to the whole set of λ-terms.

Proposition 1. *The relation $\equiv_\alpha \subseteq \Lambda \times \Lambda$ is an equivalence.*

Proof. We want to proceed by strong induction over the rank. By construction $\equiv_\alpha$ is an equivalence over Λ_0. Let us suppose that $\equiv_\alpha$ is an equivalence for Λ_i with $i < k$. We want to prove *reflexivity, symmetry and transitivity* property.

Reflex. If $M \in \hat{\Lambda}_k$ with $M = \lambda x.M_1$, then $M \equiv_\alpha M \iff M_1 \equiv_\alpha M_1$ that is true because $\equiv_\alpha$ is of equivalence on Λ_{k-1}. If $M \in \bar{\Lambda}$ with $M = M_1 M_2$, then $M \equiv_\alpha M$ if and only if $M_1 \equiv_\alpha M_1$ and $M_2 \equiv_\alpha M_2$, that is true because $\equiv_\alpha$ is of equivalence on Λ_i with $i < k$.

Symmet. Let $M \equiv_\alpha N$. If $M, N \in \hat{\Lambda}$ with $M = \lambda x.M_1$ and $N = \lambda y.N_1$, then by construction: if $x = y$ and $M_1 \equiv_\alpha N_1$ implies that $N_1 \equiv_\alpha M_1$; otherwise, if $x \neq y$, then $M_1 \equiv_\alpha N_1[y := x]$ (with $x \notin \mathcal{F}(N_1)$ and $y \notin \mathcal{F}(M_1)$) and so by Lemma 1 $M_1[x := y] \equiv_\alpha N_1[y := x][x := y] \equiv N_1$.

□

Per induzione forte e Lemma 1 si dimostra che $\equiv_\alpha$ è di equivalenza su Λ_k per ogni k. In conclusione, essendo $\Lambda = \cup_k \Lambda_k$, le relazioni di equivalenza si estendono ad una relazione di equivalenza sulle lambda espressioni.

Lemma 2. *La sostituzione è invariante per α-equivalenza. Formalmente, se $M \equiv_\alpha M'$ e se $N \equiv_\alpha N'$ allora*

$$M[x := N] \equiv_\alpha M'[x := N']$$

4 Equivalenza Semantica

Possiamo ora definire una prima versione della equivalenza semantica come segue

Definition 6 (Teoria λ)**.** Teoria del primo ordine sul linguaggio Λ con relazione di equivalenza semantica $\doteq$ per cui valgono i seguenti assiomi

(α) $M \equiv_\alpha N \Rightarrow M \doteq N$;

(β) $(\lambda x.M)N \doteq M[x := N]$;

(ξ) $M \doteq N \Rightarrow \lambda x.M \doteq \lambda x.N$;

(I) $\doteq \subseteq \Lambda \times \Lambda$ è di equivalenza;

(II) $M \doteq N \Rightarrow \quad ZM \doteq ZN \wedge MZ \doteq NZ$;