Foundations of Programming Languages 2023

The λ-calculus

Sandra Alves

September 2023

The concept of a function

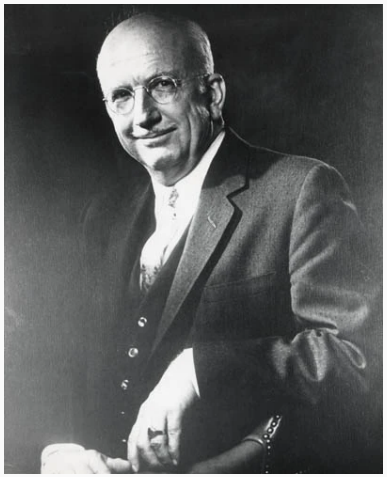
"Underlying the formal calculi which we shall develop is the concept of a function, as it appears in various branches of mathematics, either under that name or under one of the synonymous names, "operation" or "transformation". The study of the general properties of functions, independently of their appeareance in any particular mathematical (or other) domain, belongs to formal logic or lies on the boundary line between logic and mathematics."

- Alonzo Church

A brief history of the λ-calculus

In the 1930's Alonzo Church and Haskell B. Curry introduced the λ-calculus and combinatory logic, respectively.



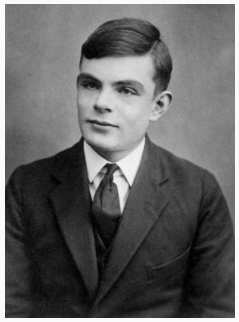



Both systems were intended as a foundational system for logic... But in 1935, Kleene and Rosser found the system to be inconsistent.

The subsystem dealing with λ-terms, reduction and conversion became what is now known as the λ-calculus.

The notion of computability

Around the same time both Church and Turing where trying to formulaly define the notion of algorithm, to answer the Entscheidungsproblem.


Origin of the problem dates back to an idea by Leibniz on the existence of a general method to solve all problems in a universal language.

Turing and Church both provided a negative answer for the problem, whilst effectively defining the notion of computability.

λ-definability and functional programming

The Church-Turing thesis estabilished that both Turing Machines and the λ-calculus define the notion of computable function.

While Turing Machines are in the basis of what became computers and the basis for imperative programming languages...

... functional programming languages found their basis on the λ-calculus.

Defining functions

A function in Haskell:

```
f : Int -> Int
f x = x + 2
```

This corresponds to the mathematical definition:

$$\begin{aligned} \texttt{f}: \quad & \texttt{Int} \to \texttt{Int} \\ & x \mapsto x + 2 \end{aligned}$$

In Haskell one can use the *nameless* notation and just write:

```
\x -> x + 2
```

We call this the *lambda* notation.

The terms of the lambda-calculus

We assume an infinite set of variables $\mathcal{V}$, and define the set of terms inductively:

- If $x \in \mathcal{V}$ then x is a term;
- If M and N are terms then (MN) is a term called an application;
- If M is a term and $x \in \mathcal{V}$ then $(\lambda x M)$ is a term called an abstraction.

Examples:

$$(\lambda x x)$$
$$(\lambda x (\lambda y x))$$
$$(\lambda x (\lambda y (\lambda z (x (y z)))))$$
$$(\lambda x (\lambda y (\lambda z ((x z) (y z)))))$$
$$((\lambda x (x x)) (\lambda x (x x)))$$

Some useful notation

Application associates to the left and abstraction to the right:

$$\begin{aligned}(M_1 M_2 \dots M_n) &= (\dots(M_1 M_2)\dots M_n)\\ (\lambda x_1 x_2 \dots x_n.M) &= (\lambda x_1(\lambda x_2(\dots(\lambda x_n M)\dots)))\end{aligned}$$

We can omit the outermost parenthesis and write the previous terms as:

$$\begin{aligned}I &= \lambda x.x\\ K &= \lambda xy.x\\ B &= \lambda xyz.x(yz)\\ S &= \lambda xyz.xz(yz)\\ \Omega &= (\lambda x.xx)(\lambda x.xx)\end{aligned}$$

Free and bound variables

An occurrence of a variable x in a term M is called *bound*, if x occurs in a subterm $\lambda x.P$ in M. Otherwise the occurrence is called *free*.

Note that a variable x can occur both free and bound in a term:

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

If we represent a λ as a tree, then x is bounded by the closest λx above in the tree of the term:

$$\lambda x.(\lambda x.x)x$$

Free and bound variables

The sets of free and bound variables of M (denoted $\mathrm{fv}(M)$ and $\mathrm{bv}(M)$) are defined inductively:

$$\begin{aligned} \mathrm{fv}(x) &= \{x\} \\ \mathrm{fv}(MN) &= \mathrm{fv}(M) \cup \mathrm{fv}(N) \\ \mathrm{fv}(\lambda x.M) &= \mathrm{fv}(M) \setminus \{x\} \end{aligned}$$

$$\begin{aligned} \mathrm{bv}(x) &= \varnothing \\ \mathrm{bv}(MN) &= \mathrm{bv}(M) \cup \mathrm{bv}(N) \\ \mathrm{bv}(\lambda x.M) &= \mathrm{bv}(M) \cup \{x\} \end{aligned}$$

M is closed iff $\mathrm{fv}(M) = \varnothing$.

Computing with terms

We have now a syntax for function definition and function application:

$f\ x = x + x$ can be represented by $\lambda x.x + x$

and the application $f\ 3$ becomes $(\lambda x.x + x)3$

One would expect the result of $f\ 3$ to be $3 + 3$, which is obtained by:

$$(x + x)[3/x]$$

and for that we rely on the substitution $M[L/x]$, that replaces in M all free occurrences of x by L.

Function with multiple arguments and currying

We have seen that $f\ x = x + x$ becomes $\lambda x.x + x$, but what about $f\ x\ y = x + y$?

This becomes $\lambda xy.x + y$, which in fact is an abreviattion for

$$(\lambda x(\lambda y(x + y)))$$

Note that, in Haskell, one can have:

```
f x y = x+y    or    f(x,y) = x+y
```

which are two different functions, as the first one can be partially applied whereas the second one cannot.

Substitution and capture of variables

Now consider the function $f\ x\ y\ =\ x$.

One would expect that $f\ M\ N$ (that is $x[M/x][N/y]$) would always map to M, right?

But if we take $M = y$ then $x[y/x][N/y] = N$.

This is related to the notion of variable capture, and to avoid it, substitution $M[N/x]$ should only be allowed, in which case we say that x is substitutable by N in M, if x does not occur free in any subterm of M of the form $\lambda y.P$, and $y \in \mathrm{fv}(N)$.

α-conversion

The previous condition can be always be ensure by renaming, when necessary, the bound variables in M. This operation is called α-conversion.

The relation $=_\alpha$ is the smallest reflexive and transitive relation that satisfies the following properties:

- If $y \notin \mathrm{fv}(M)$, then $\lambda x.M =_\alpha \lambda y.M[y/x]$.
- If $M =_\alpha N$ then $\lambda x.M =_\alpha \lambda x.N$, for every variable $x \in \mathcal{V}$.
- If $M =_\alpha N$, then $MP =_\alpha NP$.
- If $M =_\alpha N$, then $PM =_\alpha PN$.

From this point on, we will consider terms modulo $=_\alpha$.

Examples

Which of the following λ-terms are α-equivalent to the term $(\lambda xy.xz)(\lambda xa.x)$?

(i) $(\lambda xy.xz)(\lambda wa.w)$ ✓

(ii) $(\lambda xz.xz)(\lambda wa.w)$ ×

(iii) $(\lambda xy.xw)(\lambda xa.a)$ ×

(iv) $(\lambda ab.az)(\lambda cd.c)$ ✓

Comming back to substitution

The result of substituting the free occurrences of x by L in M is inductively defined as:

$$y[L/x] = \begin{cases} L & \text{if } x = y \\ y & \text{otherwise} \end{cases}$$

$$(MN)[L/x] = (M[L/x])(N[L/x])$$

$$(\lambda y.M)[L/x] = \lambda y.(M[L/x])$$

Note that in $(\lambda y.M)[L/x]$ we do not need to guareentee that $y \neq x$ or that $y \notin fv(L)$, because we can always rename bound variables using α-conversion.

α-equivalent terms and properties

The following properties are valid for all term M, N, L, M', N':

1. $\lambda x.M = \lambda y.M[y/x]$, if $x \notin \mathrm{fv}(M)$.
2. $M[N/x][L/y] = M[L/y][N[L/y]/x]$, if $y \neq x \notin \mathrm{fv}(L)$.
3. $M[y/x][N/y] = M[N/x]$, se $y \notin \mathrm{fv}(M)$.
4. If $MN = M'N'$ then $M = M'$ and $N = N'$
5. If $\lambda x.M = \lambda y.N$ then $M[y/x] = N$ and $N[x/y] = M$.

Exercise

Prove the properties above using structural induction on M.

The β-reduction relation

β-reduction captures the dynamics of the λ-calculus, is given by the single rule:

$$\underbrace{(\lambda x.M)N}_{\beta\text{-redex}} \quad \to_\beta \quad \underbrace{M[N/x]}_{\beta\text{-contractum}}$$

An inductively defined:

(i) If $M \to_\beta N$, then $\lambda x.M \to_\beta \lambda x.N$, for every $x \in \mathcal{V}$.

(ii) If $M \to_\beta N$, then $MP \to_\beta NP$.

(iii) If $M \to_\beta N$, then $PM \to_\beta PN$.

M is in *normal-form* (notation $M \in \mathrm{NF}_\beta$) if M does not contain any β-redex.

Reductions induced by $\to_\beta$

(i) The relation $\twoheadrightarrow_\beta$ is the reflexive and transitive closure of $\to_\beta$.

(ii) The relation $=_\beta$ (called β-equality) is the smallest equivalence relation that contains $\to_\beta$.

(iii) A *β-reduction sequence* is a finite or infinite sequence of the form:

$$M_0 \to_\beta M_1 \to_\beta M_2 \to_\beta \cdots$$

Lemma

(i) *If* $M \to_\beta M'$, *then* $M[N/x] \to_\beta M'[N/x]$.

(ii) *If* $N \to_\beta N'$, *then* $M[N/x] \twoheadrightarrow_\beta M[N'/x]$.

Note: We will omit β from the relations above, when the context is clear.

Examples

What is wrong with the following reductions?

- $(\lambda xy.yx)y \rightarrow_\beta \lambda y.yy$
- $(\lambda xx.xx)y \rightarrow_\beta \lambda x.yy$
- $(\lambda x.xx)(\lambda y.ay)k \rightarrow_\beta (\lambda x.xx)ak \rightarrow_\beta akak$

Contexts

A *context* $C[\ ]$ is a *'term'* containing one or more occurrences of $[\ ]$ (called holes), and such that if M is a term, then replacing the holes in $C[\ ]$ by M gives the term $C[M]$.

A binary relation **R** is *compatible* if

$$M \ \mathbf{R} \ N \Rightarrow C[M] \ \mathbf{R} \ C[N]$$

for all term M, N and all contexts $C[\ ]$ with one hole.

One can define the β-reduction relation as the compatible, reflexive and transitive relation given by the β-rule.

Normal forms

A λ-term $M \in \mathsf{NF}_\beta$, if it does not contain any β-redex.

A λ-term M is normalisable if there exists a reduction sequence:

$$M \to M_1 \to M_2 \to \cdots \to N$$

where $N \in \mathsf{NF}_\beta$.

A λ-term M is strongly normalisable if all the reductions sequences $M \to M_1 \to M_2 \to \cdots$ are finite.

If a term M is not strongly normalisable then we write $\infty(M)$.

Confluence and Termination

A term which is not in NF_β might contain several redexes...

... β-reduction is not deterministic:

$$(\lambda x.y)\Omega \to y \qquad \text{and} \qquad (\lambda x.y)\Omega \to (\lambda x.y)\Omega$$

However, it is confluent therefore normal forms are unique.

... β-reduction is not terminating:

$$\Omega \to \Omega \to \cdots$$

However, there exist terminating strategies.

Reduction graph

The *β-reduction graph* of a term M (denoted by $\mathcal{G}_\beta(M)$) is the pair

$$(\{N \mid M \twoheadrightarrow_\beta N\}, \{(M, N) \mid M \to_\beta N\})$$

It is multigraph since several redexes giving rise to $M_0 \to_\beta M_1$ will correspond to different arcs connecting M_0 to M_1 in $\mathcal{G}_\beta(M)$.

i) Having a normal form does not imply nor implies that the reduction-graph of the term is finite;
ii) If a term is strongly normalisable, then its reduction-graph is both acyclic and finite.

Reduction-graphs represent all the possible ways to reduce a term, and a reduction strategy denotes a way to travel through the reduction-graph.

Strategies

i) A *reduction strategy*, is a map F on terms: $M \twoheadrightarrow F(M)$

ii) F is a *one-step* strategy, then $M \to F(M)$ if M is not in normal form.

Some useful strategies:

- A strategy F is *normalising* if

  $$M \text{ has a normal form} \Rightarrow \exists\ n\ F^n(M) \text{ is a normal form.}$$

- F is *maximal* if the minimum number of steps F needs to reach normal form is the length of the longest finite reduction.
- F is *optimal* if it avoids duplicating redexes.
- minimal, perpetual, etc...

The notion of η-reduction

Another useful notion of reduction is:

$$\eta : \lambda x.Mx \to_\eta M, \text{ if } x \notin \text{fv}(M)$$

Intuitively η deals with extensionality in the λ-calculus.

Extensionality captures the idea of two different terms denoting the same process.

It is clear that $\lambda x.Mx$ and M, yield the same result when applied to the same term N.

Note that $\lambda xy.yx \to_\eta \lambda y.y$ but $\lambda y.yy \not\to\!\!\!\!\to_\eta y$

The notion of $\to_{\beta\eta}$ is just defined as $\to_\beta \cup \to_\eta$.

Coming back to variables...

Barendregt's variable convention:

> *"If $M_1, \ldots, M_n$ occur in a certain mathematical context (e.g. definition, proof), then in these terms all bound variables are chosen to be different from the free variables."*

Does this mean we do not need α-conversion, if we assume from the start that $\mathrm{fv}(M) \cap \mathrm{bv}(M) = \varnothing$? **No!**

$$\begin{array}{lll} (\lambda z.zz)(\lambda xy.xy) & \to & (\lambda xy.xy)(\lambda xy.xy) \\ & \to & \lambda y.(\lambda xy.xy)y \\ & =_\alpha & \lambda y'.(\lambda xy.xy)y' \\ & \to & \lambda y'.(\lambda y.y'y) \end{array}$$

The de Brujin notation

The set of nameless terms has the following alphabet:

$$\lambda, (,), 1, 2, 3, \ldots$$

The set of nameless terms is defined inductively:

- $n \in \mathbb{Z}^+$ is a nameless term;
- If M and N are nameless terms then (MN) is a nameless term;
- If M is a nameless term then (λM) is a nameless term.

Each index represents an occurrence of a variable and denotes the number of binders in scope between that occurrence and its binder.

The de Brujin notation

The term $\lambda z.(\lambda y.y(\lambda x.x))(\lambda x.zx)$ is written as $\lambda(\lambda 1(\lambda 1))(\lambda 21)$.

The notion of β reduction for nameless terms is defined by the following reduction rule:

$$(\beta) : (\lambda M)N \to M[N/1]$$

The de Brujin notation: substitution

In the β-reduction $(\lambda M)N \to M[N/1]$, intuitively three aspects need to be considered:

1. find the variables $n_1, n_2, \ldots, n_k$ in M under the range of the λ in λM;
2. decrease the free variables of M taking into account the removal of the outer binder;
3. replace $n_1, n_2, \ldots, n_k$ with N, suitably increasing the free variables occurring in N each time, to match the number of λ-binders the corresponding variable occurs under when substituted.

The de Brujin notation: substitution

Substitution $M[N/n]$ is inductevly defined as:

$$
\begin{aligned}
m[N/n] &\equiv \begin{cases} m & \text{if } m < n \\ m-1 & \text{if } m > n \\ \text{remane}_{(m,1)}(N) & \text{if } m = n \end{cases} \\
(M_1 M_2)[N/n] &\equiv (M_1[N/n])(M_2[N/n]) \\
(\lambda M)[N/n] &\equiv \lambda(M[N/n+1])
\end{aligned}
$$

The de Brujin notation: renaming indexes

$\mathsf{remane}_{(n,i)}(M)$ is inductively defined as:

$$
\begin{array}{lcl}
\mathsf{remane}_{(n,i)}(j) & \equiv & \begin{cases} j & \text{if } j < i \\ j + n - 1 & \text{if } j \geq i \end{cases} \\
\mathsf{remane}_{(n,i)}(M_1 M_2) & \equiv & (\mathsf{remane}_{(n,i)}(M_1))(\mathsf{remane}_{(n,i)}(M_2)) \\
\mathsf{remane}_{(n,i)}(\lambda M) & \equiv & \lambda(\mathsf{remane}_{(n,i+1)}(M))
\end{array}
$$

Exercises

1. Recall S, K, B and I. Show that:
   1.1 $S(KI) \twoheadrightarrow_{\beta\eta} I$
   1.2 $BI \twoheadrightarrow_{\beta\eta} I$
   1.3 $SKK \twoheadrightarrow_{\beta\eta} I$
   1.4 $S(KS)K \twoheadrightarrow_{\beta\eta} B$
2. Draw the reduction graph of:
   2.1 $K((\lambda x.xx)(II))b$
   2.2 $(\lambda x.Ixx)(\lambda x.Ixx)$
3. Consider the λ-term $M = (\lambda xyz.x(\lambda y.yy)z)(\lambda z.z(zx))(\lambda y.y)$.

   3.1 Indicate a term that is α-equivalent to M, and one that is not.
   3.2 Convert the term M to De Brujin notation.
4. Can you define functions (algorithms) that translate λ-terms to the De Brujin (and back)?