Lambda Calculus and Turing Machines

Semester Project

Anurag Sharma

Department of Mathematics and Statistics
IIT Kanpur

April 15, 2017

Outline

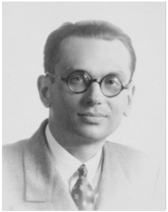
Outline

Motivation for the problem

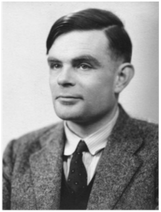
Ideas of "Effective" Computability

A thing in 1930's

What does it mean for a function $f : \mathcal{N} \rightarrow \mathcal{N}$ to be computable



Kurt Gödel



Alan Turing

Alonzo Church

Outline

Competing ideas

To each his own

Alonzo Church

Defined an idealized programming language called the **lambda calculus**, and postulated that a function is computable if and only if it can be written as a lambda term.

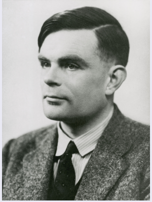
Competing ideas

To each his own

Alonzo Church

Defined an idealized programming language called the **lambda calculus**, and postulated that a function is computable if and only if it can be written as a lambda term.

Alan Turing



Proposed an idealized computer we now call a **Turing machine**, and postulated that a function is computable if and only if it can be computed by such a machine.

Outline

- A countable set of variables $\mathcal{V} = x, y, z \ldots$
- Special symbols "λ", "(" , ")" , "."

Lambda terms : $x \mid (MN)$[Application] $\mid (\lambda x.M)$[Abstraction]

- A countable set of variables $\mathcal{V} = x, y, z \ldots$
- Special symbols "λ", "(" , ")" , "."

$$\text{Lambda terms} : x \,|\, (MN)[\text{Application}] \,|\, (\lambda x.M)[\text{Abstraction}]$$

Let Λ^* be the set of strings (finite sequences) made using the the special symbols and variables of the language. The set of lambda terms is the smallest subset $\Lambda \subseteq \Lambda^*$ such that:

- Whenever $x \in V$ then $x \in \Lambda$
- Whenever $M, N \in \Lambda$ then $(MN) \in \Lambda$
- Whenever $x \in V$ and $M \in \Lambda$ then $(\lambda x.M) \in \Lambda$

Free and Bound variables

Free Variables: Variables not bound by a lambda abstraction.
Examples:

$$y \text{ is free variable in } \lambda x.(xy)$$

Rules:

$$FV(x) = \{x\}$$
$$FV(MN) = FV(M) \cup FV(N)$$
$$FV(\lambda x.M) = FV(M) \backslash \{x\}$$

Free and Bound variables

Free Variables: Variables not bound by a lambda abstraction.
Examples:

$$y \text{ is free variable in } \lambda x.(xy)$$

Rules:

$$FV(x) = \{x\}$$
$$FV(MN) = FV(M) \cup FV(N)$$
$$FV(\lambda x.M) = FV(M) \backslash \{x\}$$

Example:
Free variables in $\lambda x.(xyz)$ are y and z
Free variables in $(\lambda x.(xyz))(\lambda y.(xy))$ are x, y and z

Outline

α **equivalence**: Terms are equivalent up to renaming of bounded terms

Example: $\lambda x.xy \equiv_\alpha \lambda z.zy$

α **equivalence**: Terms are equivalent up to renaming of bounded terms

Example: $\lambda x.xy \equiv_\alpha \lambda z.zy$

β **reduction**: $(\lambda x.e_1)e_2 \rightarrow_\beta e_1[e_2/x]$
where $e_1[e_2/x]$ denotes the result of substituting e_2 for all free occurrences of x in e_1

Example:

- $(\lambda x.xy)z \rightarrow_\beta zy$
- $(\lambda x.xx)(\lambda y.y) \rightarrow_\beta (\lambda y.y)(\lambda y.y)$
- $(\lambda x.x^2)(3) \rightarrow_\beta 3^2 = 9$

Outline

For each natural number n, we define a lambda term $\overline{n}$, called the n^{th} *Church numeral*, as $\overline{n} = \lambda fx.f^n x$.

$$\overline{1} = \lambda fx.fx$$
$$\overline{2} = \lambda fx.f(fx)$$
$$\dots$$

Successor Function

succ $= \lambda nfx.f(nfx)$

$$\begin{aligned}
\textbf{succ}\ \overline{n} &= (\lambda nfx.f(nfx))(\lambda fx.f^n x) \\
&\to_\beta \lambda fx.f((\lambda fx.f^n x)fx) \\
&\twoheadrightarrow_\beta \lambda fx.f(f^n x) \\
&= \lambda fx.f(f^{n+1} x) \\
&= \overline{n+1}
\end{aligned}$$

$\textbf{add} = \lambda nmfx.nf(mfx)$

Applying it to $\overline{1}$ and $\overline{2}$

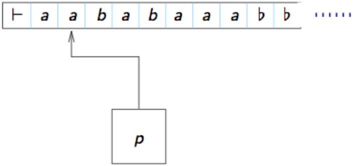
$$\begin{aligned}
\overline{1} &= \lambda fx.fx \\
\overline{2} &= \lambda fx.f(fx) \\
\textbf{add}\ \overline{1}\ \overline{2} &= \lambda nmfx.nf(mfx)\overline{1}\ \overline{2} \\
&\rightarrow_\beta \lambda fx.\overline{1}f(\overline{2}fx) \\
&\rightarrow_\beta \lambda fx.(\lambda fx.fx)f((\lambda fx.ffx)fx) \\
&\twoheadrightarrow_\beta \lambda fx.(\lambda fx.fx)fffx) \\
&\rightarrow_\beta \lambda fx.fffx \\
&= \overline{3}
\end{aligned}$$

Similarly we can define other functions:

$\textbf{mult} = \lambda nmf.n(mf)$

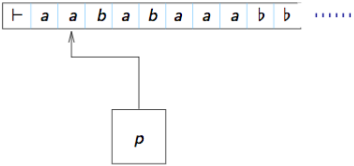
$\textbf{or} = \lambda ab.aab$

Figure: Turing Machine



- An infinite tape
- Read and Write header
- A set of states $(p, q, r \dots)$ and symbols $(a, b \dots)$

Figure: Turing Machine



Depending on the current state and the currently scanned input symbol, the machine erases the symbol and writes a new symbol into that cell.
It moves the current position one cell to the left or right, and changes to a new state.

Outline

Proof of Equivalence

Setup

Proof Technique: For every λ-definable sequence we construct a Turing Machine that computes the sequence.

Proof Technique: For every λ-definable sequence we construct a Turing Machine that computes the sequence.
We say that a sequence γ whose n^{th} figure is $\phi_\gamma(n)$ is λ-definable or effectively calculable if $1 + \phi_\gamma(n)$ is a λ-definable function of n.
i.e there is a well-formed-formula M_γ such that following holds true for all integers n:

$$\{M_\gamma\}(N_n) \twoheadrightarrow_\beta N_{1+\phi_\gamma(n)}$$

i.e $\{M_\gamma\}(N_n)$ is convertible into $\lambda x x^{|}.x(x(x^{|}))$ or into $\lambda x x^{|}.x(x^{|})$ according as the n^{th} figure of γ is 1 or 0.

We need three Turing machines:

$\mathcal{L}_1$, which if supplied with a well-formed-formula, M obtains any formula into which M is convertible.
$\mathcal{L}_2$ which successively obtains all the formulae into which M is convertible
$\mathcal{L}_3$ compares each of the reduction with N_1 or N_0 and writes 1 or 0 on the tape.

Now the machine $\mathcal{L}$ is such that it has n-sections each for computing the n^{th} figure of the sequence γ. The first stage in each of the section is the formation of $\{M_\gamma\}(N_n)$. This formula is sent to $\mathcal{L}_2$ which subsequently obtains each formula to which it can be converted. And then $\mathcal{L}_3$ does the comparison and writes 0 or 1 on the tape.