Fadil Al Turki

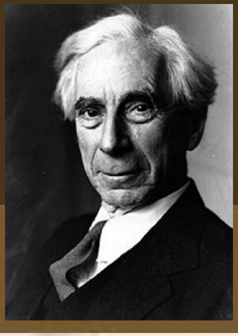
An Escape !

How do we read this:

$R = \{A \mid A \text{ not} \in A\}$?

R is the set of all sets that do not contain themselves as members

Russell's Paradox



$$R \in R \iff R \ \textbf{not} \in R$$

Attempts to a new Foundation of Mathematics

Set Theory is the foundation Mathematics..

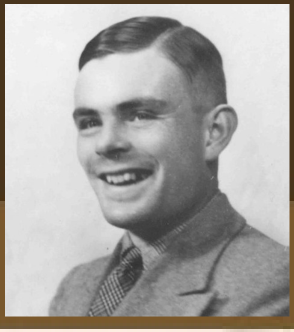
- Ernst Zermelo: (ZFC) : more Axioms to Frege's
- Russell's Type Theory
- Alonzo Church: λ − Calculus
- and others..

Alonzo Church
1903 – 1995

In the 1930's

- Tried to base mathematics on **functions** .. not on **sets**
- a **new tool** for investigating recursion theory
- a new paradigm : **Funcitonal Programming**

Alan Turing
1912 -1954



At the same time !

- **Godel's** arithmetic formal language → Turing Machines
- → **Theory of Computation**
- Entscheidungsproblem **: Decision Problem**
- Halting Problem illustrated **undecidable**
- **Church–Turing Thesis**

- **Turing Machines ⇐⇒ λ − Calculus**

Church vs Turing

Church

- **Invented a Formal System**
- **Defined what is a computational function in the system**

→ Functional Programming

Turing

- **Invented a set of machines**
- **Defined what is a computational function via the machines**

→ Imperative Programming

What is λ - Calculus?

- Formal System
- Investigates:

✓ Funciton Definiton
✓ Funciton Applicaiton
✓ Recursion

Some Feel the Calculus

- an **Expression** = a **function** with 1 **argument**
- an **Argument** = a **function** with 1 **argument**
- **value** of a **function** = a **function** with 1 **argument**

Some Feel of the Calculus

Functions:

- Have **no names**
- Define **expressions** (actions) **applied** to **arguments**

How Does it Look Like?

$$(\lambda\ x.x)\ 3$$

function arg exp applied to

Formal Definition

- **V** : a variable – idenitifer
- **λ V.E** : an **abstraction** (definition)
- **V**: variable
- **E**: lambda expression
- **E E`**: an **application** of **E** with an agrument **E`**

Some Examples

- $(\lambda\ x.x)\ a = a$
- $(\lambda\ x.y)\ a = y$
- $(\lambda\ x.xa)\ a = aa$
- $(\lambda\ x.xx)\ a = aa$
- $(\lambda\ x.xx)\ (\lambda\ y.y) = (\lambda\ y.y)(\lambda\ y.y)$

Free and Boound Variables

- Variables are either **free** or **bound**
- **λ x. xy**: **x** : bound, **y** : is free
- **x** is associated to a **λ**

Free and Bound Formally

Free Variables in **λ** are defined inductively**:**

- in an expression V (a variable): V is free

- in an expression **λ V.E**: **all** occurrences are **free** in E **except** for **V**. Here, **V** is **bound**.

- in an expression **E E`**, free occurrences are all free occurrences in **E** and **E`**

Free and Bound Examples

- (λ xy.yx) (λ x.y)
- (λ x.zx) (λ y.yx)

Changing Bound Variables

α – conversion

- a means to **rename** bound variables
- $\lambda\ x.x \rightarrow \lambda\ y.y$
- $\lambda\ x.\lambda\ x.x \rightarrow \lambda\ y.\lambda\ x.x$
- $\lambda\ x.\lambda\ x.x \rightarrow \lambda\ y.\lambda\ x.y$ (diff meaning!)

α - Conversion

α – conversion rules are not trivial

- renamed vars are those **bound** to the same abstraction

$$\lambda\, x.\lambda\, x.x \rightarrow \lambda\, y.\lambda\, x.y$$

- not possible if a var is captured by another abstraction

$$\lambda\, x.\lambda\, y.x \rightarrow \lambda\, y.\lambda\, y.y$$

Replace vars with Expressions

Substitution

Replace a variable **V** with **E`**, whenever V is free in E.

For **λ V.E**

E[V:= E`]

Substitution

Rules are defined inductively:

1. $V[V := E] == E$
2. $W[V := E] == W$, if W and V are different
3. $(E1\ E2)[V := E] == (E1[V := E]\ E2[V := E])$
4. $(\lambda\ V.\ E')[V := E] == (\lambda\ V.\ E')$
5. $(\lambda\ W.\ E')[V := E] == (\lambda\ W.\ E'[V := E])$, if V and W are different and W is not free in E.
6. $(\lambda\ W.\ E')[V := E] == (\lambda\ W'.\ E'[W := W'])[V := E]$, if V and W are different and if W' is not free in E.

Function Application

β -Reduction

$$(\lambda\ V.E\ )\ E` \rightarrow E\ [V{:=}E`]$$

Some Conventions

- $\lambda xy...z.E \equiv \lambda x(\lambda y(\ldots(\lambda z\ E))$: 1 arg in pure lambda
- $EAB..Z \equiv (\ldots((MA)B)\ldots Z)$: Left Associative
- Parathesis are for Clarity

Extensionality

- 2 functions are the same ⟸⟹ they give same resutls for all arguments

- known as **η – conversion**

- convert between **λx.fx** and **f**

  if x is not free in **f**.
- conversions may not be equivelent
- a program **λx.fx** terminates while **f** does not!

How about Paradoxes?

- **λ- Calculus** could not avoid the set-theoritic paradoxes
- It was a fresh air.. and brought up the new paradigm of functional programing..
- It is a minimilistic programing lanuagage.
- Used to study computibility.
- There is a lot more in **λ-Calculus**

Thank You

More on **λ- Calculus**:

- Wikipeida
- http://mathworld.wolfram.com
- http://planetmath.org
- A Tutorial Introduction to the Lambda Calculus by Ra´l Rojas
- A short introduction to the Lambda Calculus Achim Jung

Image: http://www.gravestmor.com