λ-Calculus

Professor: Suman Saha

The λ-Calculus

Alonzo Church, 1930s

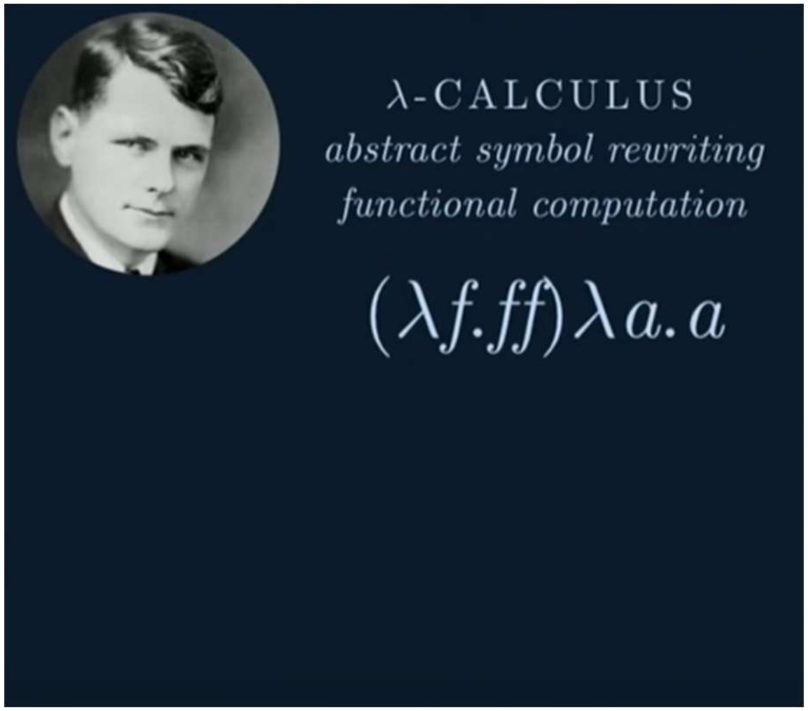
A pure λ-term is defined inductively as follows:

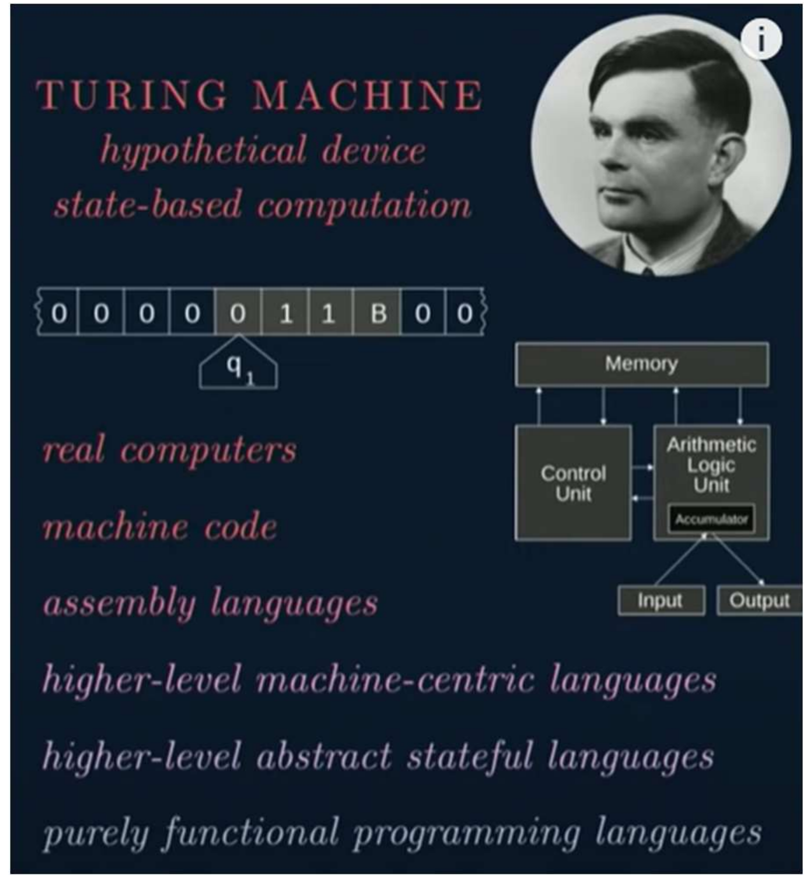
- Any variable x is a λ-term
- If t is a λ-term, so is $\lambda x.\, t$ (abstraction)
- If t_1, t_2 are λ-terms, so is $t_1 t_2$ (application)

Analogy in C:
Abstraction: `int f (int x) {return x+1}`
Application: `f(2)`

The λ-Calculus Syntax

The λ-Calculus to Programming Language





Anonymous Functions

Functions has the form of $\lambda x.\, t$

Functions are ***anonymous***

In C, we write

```
int id (int x) {return x}
```

In λ-calculus, we write

$$\lambda x. \qquad x$$

λ-Term Example

Identity function: $\lambda x. x$

Application: $(\lambda x. x)\ y$ [apply identity function to parameter y]

How to parse a λ-term

1. λ binding extends to the rightmost part

   $\lambda x. x\ \lambda y. y\ z$ is parsed as $\lambda x. (x\ (\lambda y. (y\ z)))$

2. Applications are left-associative

   $t_1\ t_2\ t_3$ is parsed as $(t_1\ t_2)\ t_3$

λ-Term Example (Parsing)

λ-Term Example

How to parse a λ-term

1. λ binding extends to the rightmost part

   $\lambda x. x\ \lambda y. y\ z$ is parsed as $\lambda x. (x\ (\lambda y. (y\ z)))$

2. Applications are left-associative

   $t_1\ t_2\ t_3$ is parsed as $(t_1\ t_2)\ t_3$

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

$x\ \ \lambda x.\ y\ x\ \lambda z. z$?

λ-Term Example (Parsing)

Number of Parameters

In the pure λ-calculus, λ only bind one parameter

For convenience, we write
$\lambda x\ y.\, t$ as a ***shorthand*** for $\lambda x.\, (\lambda y.\, t)$

This process of removing parameters is called ***currying***

The λ-Calculus

A pure λ-term t is defined inductively as follows:

- Any variable x is a λ-term
- If t is a λ-term, so is $\lambda x. t$ (abstraction)
- If t_1, t_2 are λ-terms, so is $t_1 t_2$ (application)

We use $x, y, z, \ldots$ for variables

The definition above defines an infinite set, named t

Syntax vs. Semantics

Syntax: the structure/form of lambda terms

$$t ::= \; x \mid t_0 \; t_1 \mid \lambda x.\, t$$

Semantics: the meaning of lambda terms

- Lambda calculus defines computation
- What computation is defined by t ?

Capture-Avoiding Substitution

In λ-calculus, computation is defined by
capture-avoiding substitution

$t\{y/x\}$ means substitute ***all free*** x in t with y

same x?

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

bound

free

Analogy in C:

```
int x;
…
int f (int x) {return x}
```

Capture-Avoiding Substitution (Example)

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

Bound vs. Free Variables

In $(\lambda x.\, t)$, the variable x in t is ***bound*** to λx

A variable is ***free*** if it is not bound to any λ

A variable is bound to the closest λ

Examples

$(\lambda x.\, x)\ x$ applies the identity function to x (i.e., the x after dot is bound to λ)

$\lambda x.\, \lambda x.\, x$ is a function that takes a parameter, and returns the identity function (i.e., the inner-most x is bound to the second λ)

More Formally …

In ($\lambda x. t$), all free variables x in t is ***bound*** to λx

A variable is ***free*** if it is not bound to any λ

Systematically, we define free variables as follows:

- $\mathrm{FV}(x) = \{x\}$
- $\mathrm{FV}\ (t_1\ t_2) = \mathrm{FV}(t_1) \cup \mathrm{FV}(t_2)$
- $\mathrm{FV}\ (\lambda x. t) = \mathrm{FV}\ (t) - \{x\}$

Let Var(t) be all variables used in t, the bound variables in t (written $\mathrm{BD}(t)$) are
$\mathrm{BD}(t) = \mathrm{Var}(t) - \mathrm{FV}(t)$

Free Variables: Examples

- FV((x y))

- FV((λ (x) x) y)

Free Variables: Example

- FV((λ (x) x) x)

Free Variables: Example

- FV((λ (y) ((λ (x) (z x)) x)))

α-Reduction (Informal)

Identity function: $\lambda x. x$

is the same as $\lambda y. y$ and $\lambda z. z$ etc.

Observation: the name of a parameter is irrelevant in λ-calculus

Analogy in C:

```
            int f (int x) {return x}
Is same as  int f (int y) {return y}
Is same as  int f (int z) {return z}
```

α-Reduction (formal)

$\lambda x.\, t = \lambda y.(t\{y/x\})$ when $y \notin \mathrm{FV}(t)$
where $t\{y/x\}$ is capture-avoiding substitution

$$(\lambda x.\, x\ x) = (\lambda y.\, y\ y)$$
$$(\lambda x.\, x\ x) \neq (\lambda y.\, x\ y)$$
$$x \neq y$$
$$(\lambda x.\ \lambda x.\, x) \neq (\lambda y.\, \lambda x.\, y)$$

Subtle point: what if y is captured in t? Use α-reduction to rename the captured y in t

β-Reduction (Informal)

Identity function: $\lambda x. x$

$(\lambda x. x)\ y = y$

Observation: we can substitute the formal parameter with the true parameter

Analogy in C:
Abstraction: `int f (int x) {return x}`
Application: `f(y) evaluates to y`

β-Reduction

The key reduction rule in λ calculus

$$(\lambda x.\, t_1)\ t_2 = \ t_1\{t_2/x\}$$

Capture-avoiding substitution

β-Reduction Example

Example 1: $(\lambda x.\, x\ x)\ y$

β-Reduction

$(\lambda x.\, t_1)\ t_2 = t_1\{t_2/x\}$

β-Reduction Example

PennState

Example 1: $(\lambda x.\,(x\ x))\ y$

In this case, $(x\ x)$ corresponds to t_1,

y corresponds to t_2

FV(t_2) = y. y is not in t_1, hence, not bound

The first rule of substitution applies, which gives

$(\lambda x.\,(x\ x))\ y = y\ y$

β-Reduction Example

- ((λ (f) (f (f (λ (x) x)))) (λ (x) x))

β-Reduction Example

- ((λ (x) (x x)) (λ (x) (x x)))