Church-Turing Thesis

Peter Mayr

Computability Theory, September 15, 2023

What does it mean to compute something?

Algorithm intuitively: description of procedure for some calculation or task

Entscheidungsproblem (Hilbert, 1928)

Find algorithm that yields for a given statement in first order logic whether it's true or not.

(cf. Gödel's completeness theorem for first order logic)

In 1936 Church and Turing independently show that the Entscheidungsproblem is not solvable in the **formal computational models** they devise:

- λ-**calculus** by Church,
- **Turing machines** by Turing.

Afterwards they prove that the class of functions described by λ-calculus is the same as the those computable by DTMs.

Church-Turing Thesis

Every problem that is solvable by an algorithm (in any intuitive sense) is solvable by a Turing machine.

Not provable but every known computational model can be simulated by DTMs, e.g.

- recursive functions
- register machines
- cellular automata
- actual programming languages

⋮

Examples above are actually **Turing complete** i.e., can simulate TMs as well.

Lambda calculus

- Invented by Church.
- Idea: Objects are anonymous unary functions.
- Typed λ-calculus is the basis of functional programming (e.g. Haskell).

Syntax

λ-**terms** are defined inductively as

- variables $x_1, x_2, \ldots$
- **abstraction**: $(\lambda x.t)$ is a λ-term for t a λ-term and x a variable bound by λ idea: $x \mapsto t$
- **application**: (st) is a λ-term for λ-terms s, t
  idea: $s(t)$

Example

λ-term		interpretation	
$\lambda x.x$		$x \mapsto x$	
$\lambda x.y$		constant $x \mapsto y$	
$\lambda x.x + 1$	informal	$x \mapsto x + 1$	
$\lambda x.(\lambda y.x + y)$		$x \mapsto (y \mapsto x + y)$	currying of $(x,y) \to x+y$
$s := (\lambda x.t)$			
$su = (\lambda x.t)u$		$t(x \leftarrow u)$	

Rules to manipulate λ-terms

- α-**conversion:** Variables bound to an abstraction λ can be renamed.
  $\lambda x. \quad \ldots x \ldots x \ldots \quad \to \quad \lambda y. \quad \ldots y \ldots y \ldots$
- β-**reduction**: Apply functions to arguments if all free (not bound) variables in t remain free. *
  $(\lambda x.s)t \quad \to \quad s[x \leftarrow t]$
- η-**conversion:** Identify functions that give the same outputs.
  $\lambda x.(tx) \quad \to \quad t \quad$ if x is not free in t

$$* \quad \begin{array}{ccc} \lambda x. (\lambda y. x+y) \underbrace{y}_{y \text{ free}} & \not\to_\beta & \lambda y. y+y \\ \alpha \downarrow & & \neq \\ \lambda x (\lambda y. x+y)\ z & \to_\beta & \lambda y. z+y \end{array}$$

A working example

Church numerals

We can represent natural numbers in λ-calculus as follows:

idea

$\mathbf{0} := \lambda f.(\lambda x.x)$ f bound

$f^0(x)$

$\mathbf{1} := \lambda f.(\lambda x.fx)$

$f^1(x)$

$\mathbf{2} := \lambda f.(\lambda x.f(fx))$

$\vdots$

$\mathbf{n} := \lambda f.(\lambda x.f^n x))$ short: $\lambda fx.f^n x$

$f^n(x) \leftrightarrow g^n(x)$

Example

$\text{plus} := \lambda mnfx.(mf)((nf)x)$

Claim: $(\text{plus}\ \mathbf{m})\ \mathbf{n} \to^* \mathbf{m+n}$

$$((\lambda mnfx.(mf)((nf)x))\ \underline{m})\ \underline{n}$$

$$\to_\beta (\lambda nfx\ \underbrace{(\underline{m}f)}\ (nf)x)\ \underline{n}$$

Note $\underline{m}f = (\lambda f.(\lambda x.f^m x))\ f \to_\beta \lambda x.f^m x$

$$\to_\beta \lambda fx.(\lambda x. f^m x)\underbrace{(n f)\, x}_{\to_\beta f^n x}$$

$$\to_\beta \lambda fx.(f^m(f^n x))$$

$$= \underline{m+n}$$

λ-calculus vs TM

λ-calculus and DTMs have the same computational power in the following precise sense.

Theorem

A function $f : \mathbb{N} \to \mathbb{N}$ is computable (by a DTM) iff there exists a λ-term t such that $\forall x, y \in \mathbb{N}$:

$$f(x) = y \text{ iff } t\mathbf{x} \leftrightarrow_\beta \mathbf{y}.$$

Without proof.