Church–Turing Thesis

CSCI 2210

2023-11-15

Turing Computable Functions

We have seen what it means for language to be **decidable** by Turing machine:

There is a TM that for any string over the input alphabet *halts* and either accepts or rejects the string, according to its state.

However, a TM may leave some contents in its memory when it halts.

By halting in its accepting state and and leaving a string in its memory, a TM M realizes a **partial function** $f : \Sigma^* \rightharpoonup \Gamma^*$.

If M halts on all input strings then it realizes a (total) **function** $f : \Sigma^* \to \Gamma^*$.

That is to say, the machine M **computes** the function f.

We can use strings to encode numbers (for example in binary).

If machine M computes function $f : \mathbb{N} \to \mathbb{N}$ we say that f is **Turing computable**.

Lambda Computable Functions

We've seen how we can encode numbers in λ-calculus as well.

Let $\overline{n}$ be the *Church numeral* for $n \in \mathbb{N}$.

A λ-term M realizes a partial function $f : \mathbb{N} \rightharpoonup \mathbb{N}$ with

$$f(n) = r \qquad \text{if} \qquad \mathrm{M}\,\overline{n} \downarrow_\beta \overline{r}$$

If $\forall\, n \in \mathbb{N} \,.\, \exists\, r \in \mathbb{N} \,.\, \mathrm{M}\,\overline{n} \downarrow_\beta \overline{r}$ then M also computes a function $f : \mathbb{N} \to \mathbb{N}$ and we say that f is **lambda computable**.

Church-Turing Equivalence

Theorem (Turing)

The functions $f : \mathbb{N} \to \mathbb{N}$ that are Turing-computable are the same ones that are λ-computable.

$$\text{Turing-computable functions} \quad = \quad \lambda\text{-computable functions}$$

Effective Computability

Before the time of automated computation a “computer” was a person who followed a formal description of an algorithm to compute a function.

A function that can be computed in this way, ignoring constraints on time, patience, paper, etc., is called **effectively computable**.

Church and Turing independently proposed their respective models of computation as an answer to the question, *which functions are effectively computable?*

Each asserted that the functions computable in their model of computation coincide with the effectively computable ones:

$$\lambda\text{-computable} \underbrace{\quad=\quad}_{\text{Church's thesis}} \text{effectively computable} \underbrace{\quad=\quad}_{\text{Turing's thesis}} \text{Turing-computable}$$

But because the functions computable in their models coincide, they were arguing for the same thing.

Church-Turing Thesis

The **Church-Turing Thesis** is the proposition that the *effectively computable* functions coincide with the λ*-computable* / *Turing-computable* functions.

This is *not* a mathematical proposition, but a partly empirical and partly philosophical one.

It is *empirical* in the sense that it can be *disproved* by exhibiting a function computable by an idealized person following an algorithm that cannot be computed in either model of computation.

It is *philosophical* in the sense that it is not clear what *resources* such an idealized person should have access to.

Beyond the Church-Turing Thesis

In the original CTT, *effectively computable* refers to what a person can do, given unbounded resources, patience, and time but without employing any creativity or judgement.

But what are the resources?

What if the algorithm starts with building an electronic computer?

Or a quantum computer?

Or a wormhole to another galaxy?

Physical Church-Turing Thesis

The **physical Church-Turing thesis** (which was *not* asserted by Church or Turing) asserts that the λ/Turing-computable functions coincide with those that can be computed by *any physically realizable means*.

This thesis could be refuted at any moment by *constructing* a machine that computes a function not computable by TM.

So far this thesis has not been refuted.

Feasible Computability

Computability is not the bright line it may naively appear.

If we don’t have the resources (time, memory, energy) to run a computation and obtain the result, then knowing that the computation *will* eventually yield an answer does us little good.

There are variations of CTT that make empirical claims about the functions that are **feasibly computable**.

These are generally functions that require resources *polynomial* in the size if their input.

Relative Computability

Rather than search the world for physical phenomena that can expand the set of computable functions, we can posit models of computation that are more powerful than those we know how to realize.

We can explore which functions are computable *in such models*, whether or not they are physically realizable.

An intriguing example is *Accelerating Turing Machines*, where each computational step takes half the time of the previous step.

In this model the *halting problem* is decidable:

On input $\langle \mathrm{M}\,, w \rangle$ begin by writing a symbol representing “no”, then simulate running M on w.

If the simulation ever halts then change the answer to “yes”.

In $\sum_{n \in \mathbb{N}} 1/2^n = 2$ units of time you will know whether the simulation ever halted.