Logic, Computability and Incompleteness

Recursive Functions

Wolfgang Schwarz

31 October 2025

Recursive Functions

Recursive Functions

We'd like a precise, formal account of **algorithms**, so that we can study whether there is an algorithm for a given task.

Let's focus on algorithms for computing functions on $\mathbb{N}$.

Gödel/Kleene's idea: an algorithm for computing a function breaks the function down into simpler functions combined by a few basic operations.

Recursive Functions

A function is (partial) **recursive** if it can be built from

- the zero function,
- the successor function,
- the projection functions

by applications of

- composition,
- primitive recursion, and
- minimization.

The partial recursive functions are exactly the Turing-computable functions.

Primitive recursive functions

Primitive recursive functions

Base functions:

- Successor $s(x) = x + 1$.
- Zero $z(x) = 0$.
- Projections $\pi_i^n(x_1, \ldots, x_n) = x_i$.

These are “computable in 1 step”.

Primitive recursive functions

From computable functions f and g, we can build a new function h so that

$$h(x) = f(g(x)).$$

This is still computable.

$$\mathrm{Cn}[f, g](x) = f(g(x)).$$

Primitive recursive functions

From computable functions f and g, we can build a new function h so that

$$\begin{aligned} h(x, 0) &= f(x), \\ h(x, s(y)) &= g(x, y, h(x, y)). \end{aligned}$$

Example:

$$\begin{aligned} \mathsf{add}(x, 0) &= x, \\ \mathsf{add}(x, s(y)) &= s(\mathsf{add}(x, y)). \end{aligned}$$

Primitive recursive functions

From computable functions f and g, we can build a new function h so that

$$\begin{aligned} h(x, 0) &= f(x), \\ h(x, s(y)) &= g(x, y, h(x, y)). \end{aligned}$$

This is still computable: to compute $h(x, 0)$, start at $h(x, 0)$ and loop up to y.

Notation: $\mathrm{Pr}[f, g]$.

Primitive recursive functions

A function is **primitive recursive** if it can be built from

- the zero function,
- the successor function,
- the projection functions

by applications of

- composition and
- primitive recursion.

Primitive recursive functions

Primitive recursive functions are always total.

They can be computed using bounded loops.

Examples:

- Addition
- Multiplication
- Exponentiation
- Factorial
- Any function you can think of

Diagonalizing out

We can mechanically enumerate all primitive recursive functions: $f_1, f_2, f_3, \ldots$.

Define $d(x) = f_x(x) + 1$.

This function is computable.

But it can't be primitive recursive, because it differs from every f_n at input n.

Unbounded search

Unbounded search

A function is **partial recursive** if it can be built from

- the zero function,
- the successor function,
- the projection functions

by applications of

- composition,
- primitive recursion, and
- **minimization**.

Unbounded search

The **minimization** of a function $f(x, y)$ is a function $h(x)$ that returns the least y such that

(i) $f(x, y) = 0$, and
(ii) for all $z < y$, $f(x, z)$ is defined.

If f is computable, so is $\mathrm{Mn}[f]$: to compute $g(x)$, compute $h(x, y)$ for $y = 0, 1, 2, \ldots$ until you hit 0.

$\mathrm{Mn}[f]$ can be turn a total function into a non-total function.

Unbounded search

The **minimization** of a function $f(x, y)$ is a function $h(x)$ that returns the least y such that

(i) $f(x, y) = 0$, and
(ii) for all $z < y$, $f(x, z)$ is defined.

A function f is **regular** if it is total and for all x there is some y with $f(x, y) = 0$.

Regular minimization is minimization applied to regular functions.

It always yields a total function.

Unbounded search

A function is **partial recursive** if it can be built from

- the zero function,
- the successor function,
- the projection functions

by applications of

- composition,
- primitive recursion, and
- **minimization**.

Unbounded search

A function is **(total) recursive** if it can be built from

- the zero function,
- the successor function,
- the projection functions

by applications of

- composition,
- primitive recursion, and
- **regular minimization**.

The Church-Turing Thesis

The Church-Turing Thesis

A function is partial recursive iff it is Turing-computable.

A total function is recursive iff it is Turing-computable.

The Church-Turing Thesis:

A total function is computable iff it is recursive/Turing-computable.

Right to left:

1. Assume that f is recursive.
2. Then f can be constructed from the base functions by composition, primitive recursion, and regular minimization.
3. This construction allows us to specify an algorithm for computing f.

The Church-Turing Thesis:

A total function is computable iff it is recursive/Turing-computable.

Left to right:

1. Assume that f is computable.
2. Then one can specify a mechanical, step-by-step algorithm for converting the input to f into f's output.
3. This algorithm will manipulate finite chunks of symbols at each step, according to predefined rules.
4. Any such algorithm can be implemented by a Turing machine.
5. So f is recursive/Turing-computable.