

Primitive Recursive Functions

We define and study the functions that express the “easy” computations of arithmetic.

In this chapter, our viewpoint is *semantic*: we study objects from “reality”—natural numbers, prime numbers, addition, and so on.

In the next chapter, we will connect this with the language $\mathcal{L}_0$ of Peano arithmetic, which provides the *syntactic* viewpoint in which everything is expressed as sequences of symbols. The bridge between the two will be the representability theorem: primitive recursive functions and predicates can be represented in the language of Peano arithmetic.

1. Primitive recursive functions

Primitive recursive functions are the functions that are “easy to compute” in the sense of the previous chapter: their values can be computed by an algorithm using only “for” loops.

They are functions $f : \mathbb{N}^p \rightarrow \mathbb{N}$. They are defined from three basic functions and two elementary operations. Nevertheless, this class contains all the usual functions and operations on natural numbers: addition, multiplication, exponentiation, and so on.

1.1. Definition

We write $\mathcal{F}_p = \{f : \mathbb{N}^p \rightarrow \mathbb{N}\}$ and $\mathcal{F} = \bigcup_{p \geq 1} \mathcal{F}_p$. Recall that $\mathbb{N}^p = \mathbb{N} \times \cdots \times \mathbb{N}$. We will often denote the p -tuple $(x_1, \dots, x_p) \in \mathbb{N}^p$ by $\mathbf{x}$. Thus, the elements of $\mathcal{F}$ are functions of p variables (where p may vary from one function to another).

The definition of recursive functions is based on three basic functions and two operations.

Basic functions

- Z : the zero function

$$\begin{aligned} Z : \mathbb{N} &\longrightarrow \mathbb{N} \\ x &\longmapsto 0 \end{aligned}$$

- S : the successor function

$$\begin{aligned} S : \mathbb{N} &\longrightarrow \mathbb{N} \\ x &\longmapsto x + 1 \end{aligned}$$

- P_i^p : the projections (for $1 \leq i \leq p$).

$$\begin{aligned} P_i^p : \mathbb{N}^p &\longrightarrow \mathbb{N} \\ (x_1, \dots, x_p) &\longmapsto x_i \end{aligned}$$

Composition operation

- Given: $f_1, \dots, f_q : \mathbb{N}^p \rightarrow \mathbb{N}$ and $g : \mathbb{N}^q \rightarrow \mathbb{N}$.
- Composition: the function $g(f_1, \dots, f_q) : \mathbb{N}^p \rightarrow \mathbb{N}$ defined by

$$\mathbf{x} \mapsto g(f_1(\mathbf{x}), \dots, f_q(\mathbf{x}))$$

Recursion operation

- Given:
 - $g : \mathbb{N}^p \rightarrow \mathbb{N}$ (base case)
 - $h : \mathbb{N}^{p+2} \rightarrow \mathbb{N}$ (recursion step)
- New function $f : \mathbb{N}^{p+1} \rightarrow \mathbb{N}$ defined by:

$$f(\mathbf{x}, 0) = g(\mathbf{x}) \quad \text{and} \quad f(\mathbf{x}, y + 1) = h(\mathbf{x}, y, f(\mathbf{x}, y))$$

This should be viewed as recursion, just as for any sequence defined by an initial value u_0 and a relation $u_{n+1} = h(n, u_n)$. The recursion is on the last variable (here y).

Definition.

The set PR of **primitive recursive functions** is the smallest subset of $\mathcal{F}$ that contains Z, S, P_i^p (for every $p \geq 1, 1 \leq i \leq p$) and is closed under composition and recursion.

What does “being closed” mean? For example, for composition, if $f_1, \dots, f_q \in PR$ and $g \in PR$, then the composite $g(f_1, \dots, f_q)$ is still in PR . The same applies to recursion.

Note. The term “primitive recursive” is poorly chosen, but it is the standard terminology. A more evocative name would be “easily computable” function, but that term is used in other situations.

1.2. First examples

(a) Constant functions $\mathbb{N} \rightarrow \mathbb{N}$

Constant functions $f : \mathbb{N} \rightarrow \mathbb{N}$ are primitive recursive. We know this for the zero function Z . For the constant 1, we have $S(Z(x)) = S(0) = 1$, so $f = S \circ Z$ is primitive recursive as the composite of two basic functions. More generally, $x \mapsto k$ belongs to PR (by composing S with itself k times and applying the result to Z).

(b) Constant functions $\mathbb{N}^2 \rightarrow \mathbb{N}$

Let $k \in \mathbb{N}$ be fixed. Define:

- base case: $g : \mathbb{N} \rightarrow \mathbb{N}$, the constant function equal to k , which belongs to PR ;
- recursion step: $h : \mathbb{N}^3 \rightarrow \mathbb{N}$ defined by $h(x, y, z) = z$. This is the projection P_3^3 , so it is primitive recursive.

The recursion operation constructs a function $f : \mathbb{N}^2 \rightarrow \mathbb{N}$ in PR such that $f(x, 0) = k$ and $f(x, y + 1) = f(x, y)$. By induction on y , $f(x, y) = k$ for every y . Thus $f : \mathbb{N}^2 \rightarrow \mathbb{N}, (x, y) \mapsto k$ is primitive recursive.

By induction on p , one could prove that every constant function $f : \mathbb{N}^p \rightarrow \mathbb{N}$ is primitive recursive.

(c) Elementary arithmetic operations

Proposition 1.

The following functions are primitive recursive:

- *addition*: $\mathbb{N}^2 \rightarrow \mathbb{N}, (x, y) \mapsto x + y$;
- *multiplication*: $\mathbb{N}^2 \rightarrow \mathbb{N}, (x, y) \mapsto x \times y$;
- *exponentiation*: $\mathbb{N}^2 \rightarrow \mathbb{N}, (x, y) \mapsto x^y$.

Proof.

- Addition satisfies the recursion relation $x + (y + 1) = (x + y) + 1$. For the base case, let $g : \mathbb{N} \rightarrow \mathbb{N}$ be defined by $g(x) = x$ (in fact, $g(x) = P_1^1(x) = x$, so it is primitive recursive). Next, let $h : \mathbb{N}^3 \rightarrow \mathbb{N}$ be defined by $h(x, y, z) = S(z)$ (in fact, $h = S \circ P_3^3$, so h is primitive recursive). The recursion operation yields a primitive recursive function $f : \mathbb{N}^2 \rightarrow \mathbb{N}$.

Let us verify that $f(x, y) = x + y$. Fix x and proceed by induction on y .

- For $y = 0$, $f(x, 0) = g(x) = x = x + 0$.
- Suppose $f(x, y) = x + y$. Then:

$$f(x, y + 1) = h(x, y, f(x, y)) = S(f(x, y)) = f(x, y) + 1 = (x + y) + 1 = x + (y + 1)$$

We have proved by induction on y that $f(x, y) = x + y$ for all x, y . We knew that f was primitive recursive, so addition is as well.

- For multiplication, use the relation $x \times (y + 1) = x \times y + x$. This is left as an exercise.
- Exponentiation is also left as an exercise.

□

(d) Truncated subtraction**Proposition 2.**

- The predecessor function $\text{prec} : \mathbb{N} \rightarrow \mathbb{N}$,

$$\text{prec}(x) = \begin{cases} x - 1 & \text{if } x \geq 1 \\ 0 & \text{if } x = 0 \end{cases}$$

is primitive recursive.

- The **truncated subtraction** $\mathbb{N}^2 \rightarrow \mathbb{N}, (x, y) \mapsto x \dot{-} y$, is also primitive recursive.

$$x \dot{-} y = \begin{cases} x - y & \text{if } x \geq y \\ 0 & \text{otherwise} \end{cases}$$

Proof.

- The function $\text{prec} : \mathbb{N} \rightarrow \mathbb{N}$ satisfies the following recursion relation: $\text{prec}(0) = 0$, and for $y \geq 0$, $\text{prec}(y + 1) = y$. This intuitively makes it primitive recursive. Technically, however, it is a little more complicated because the recursion operation produces a function $f : \mathbb{N}^{p+1} \rightarrow \mathbb{N}$. Proceed as follows:

- $g(x) = Z(x) = 0$ belongs to PR.
- $h(x, y, z) = P_2^3(x, y, z) = y$ also belongs to PR.

Thus, the function $f : \mathbb{N}^2 \rightarrow \mathbb{N}$ defined by $f(x, 0) = 0$ and satisfying $f(x, y + 1) = h(x, y, f(x, y)) = y$ belongs to PR . Hence $f(x, y)$ is primitive recursive.

Note: $f(x, y)$ is a function of two variables in which x serves no purpose. We eliminate it through the following composition: $\text{prec} : \mathbb{N} \rightarrow \mathbb{N}$, defined by $\text{prec}(y) = f(P_1^1(y), P_1^1(y)) = f(y, y)$, which is therefore primitive recursive.

- Truncated subtraction satisfies the recursion relation $x \dot{-} 0 = x$ and $x \dot{-} (y + 1) = \text{prec}(x \dot{-} y)$. Let $g(x) = P_1^1(x) = x$ and $h(x, y, z) = \text{prec}(P_3^3(x, y, z)) = \text{prec}(z)$. Since g and h are primitive recursive, the recursion operation yields a primitive recursive function $f : \mathbb{N}^2 \rightarrow \mathbb{N}$ satisfying $f(x, y) = x \dot{-} y$.

□

1.3. Remarks

- **Another definition of PR .** We can also define the set of primitive recursive functions “from below”.

$$PR_0 = \{Z, S\} \cup \{P_i^p\}_{p \geq 1, 1 \leq i \leq p}$$

$$PR_{n+1} = PR_n \cup \{\text{functions obtained from } PR_n \text{ by composition}\}$$

$$\cup \{\text{functions obtained from } PR_n \text{ by recursion}\}$$

Then $PR = \bigcup_{n \geq 0} PR_n$.

- **Variables.** We can permute the variables however we wish. For example, if $f : \mathbb{N}^2 \rightarrow \mathbb{N}$, $(x, y) \mapsto f(x, y)$ is primitive recursive, then $(x, y) \mapsto f(y, x)$ is also primitive recursive. Indeed, $f(y, x) = f(P_2^2(x, y), P_1^2(x, y))$. Thus, we often use the variables x, y, z directly and in any order we wish.
- **Functions $f : \mathbb{N}^0 \rightarrow \mathbb{N}$.** The elements of $\mathbb{N}^p$ are p -tuples $(x_1, \dots, x_p)$. For $p = 0$, $\mathbb{N}^0$ consists of a single 0-tuple, denoted by $()$. Thus $\mathbb{N}^0 = \{()\}$, and a function $f : \mathbb{N}^0 \rightarrow \mathbb{N}$ has only one value (because its domain has only one element), so it is a constant function.

Until now, we excluded the case $p = 0$. It can be useful to allow it and to adopt the convention that functions $f : \mathbb{N}^0 \rightarrow \mathbb{N}$ are primitive recursive (they are regarded as constants).

This convention allows us to use the recursion operation with $p = 0$. It simplifies the proof that prec is primitive recursive (indeed, the variable x was unnecessary in the proof). The proof that $n!$ is primitive recursive by this method is left as an exercise.

1.4. Algorithms

Let us connect this with the previous chapter. If $f : \mathbb{N}^p \rightarrow \mathbb{N}$ is primitive recursive, then for every $\mathbf{x} \in \mathbb{N}^p$, we can compute $f(\mathbf{x})$ with an algorithm using only “for” loops (with a well-controlled number of iterations).

To prove that the functions in PR satisfy a certain property, it suffices to verify it for the three basic functions and show that the property is preserved by composition and recursion.

Clearly, $Z(n) = 0$, $S(x) = x + 1$, and $P_i^p(\mathbf{x}) = x_i$ are easy to compute. If f is the composite $g(f_1, \dots, f_q)$ and there are easy algorithms for $f_1, \dots, f_q$ and g , then these algorithms compute $y_1 = f_1(\mathbf{x}), \dots, y_q = f_q(\mathbf{x})$, followed by $g(y_1, \dots, y_q)$, thereby computing $f(\mathbf{x})$ through a finite sequence of easy algorithms.

If f is obtained by recursion, $f(\mathbf{x}, 0) = g(\mathbf{x})$ is assumed to be computable by an easy algorithm. This lets us compute $f(\mathbf{x}, 1) = h(\mathbf{x}, 0, f(\mathbf{x}, 0))$, where h is also computed by an easy algorithm. Step by step, we compute $f(\mathbf{x}, 2), f(\mathbf{x}, 3), \dots$ up to the desired value.

2. Sets and predicates

2.1. Sets

Definition.

Let $A \subset \mathbb{N}^p$. The **characteristic function** $\chi_A : \mathbb{N}^p \rightarrow \mathbb{N}$ is defined by:

$$\chi_A(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \in A \\ 0 & \text{otherwise} \end{cases}$$

It is also sometimes denoted by $\mathbb{1}_A$.

Definition.

A set $A \subset \mathbb{N}^p$ is a **primitive recursive set** if its characteristic function χ_A is primitive recursive.

For example, the set $A \subset \mathbb{N}$ of even natural numbers is primitive recursive. Indeed, its characteristic function can be defined by the following recursion relation:

$$\chi_A(0) = 1 \quad \text{and} \quad \chi_A(n+1) = 1 \dot{-} \chi_A(n)$$

This makes χ_A primitive recursive.

Proposition 3.

If A and B are primitive recursive subsets of $\mathbb{N}^p$, then $A \cup B$, $A \cap B$, and $\mathbb{N}^p \setminus A$ are also primitive recursive sets.

For the proof, we need the **sign function** $\text{sgn} : \mathbb{N} \rightarrow \mathbb{N}$ defined by:

$$\text{sgn}(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n > 0 \end{cases}$$

It is primitive recursive because it can also be written as $\text{sgn}(n) = 1 \dot{-} (1 \dot{-} n)$.

Proof. The characteristic functions of these sets can be expressed using the primitive recursive functions χ_A and χ_B :

- $\chi_{A \cap B} = \chi_A \times \chi_B$
- $\chi_{A \cup B} = \text{sgn}(\chi_A + \chi_B)$
- $\chi_{\mathbb{N}^p \setminus A} = 1 \dot{-} \chi_A$

□

Proposition 4.

If f is primitive recursive and $A = \{\mathbf{x} \in \mathbb{N}^p \mid f(\mathbf{x}) = 0\}$, then A is a primitive recursive set.

Proof. We have $\chi_A(\mathbf{x}) = 1 \dot{-} f(\mathbf{x})$.

□

Proposition 5 (Definition by cases).

Let $g, h : \mathbb{N}^p \rightarrow \mathbb{N}$ be primitive recursive functions and let $A \subset \mathbb{N}^p$ be a primitive recursive set. The function $f : \mathbb{N}^p \rightarrow \mathbb{N}$ defined by:

$$f(\mathbf{x}) = \begin{cases} g(\mathbf{x}) & \text{if } \mathbf{x} \in A \\ h(\mathbf{x}) & \text{otherwise} \end{cases}$$

is primitive recursive.

Proof. We can write $f(\mathbf{x}) = g(\mathbf{x}) \times \chi_A(\mathbf{x}) + h(\mathbf{x}) \times \chi_{\mathbb{N}^p \setminus A}(\mathbf{x})$. It is expressed using sums and products of primitive recursive functions and is therefore primitive recursive as well.

□

2.2. Predicates

Definition.

A p -place predicate $P : \mathbb{N}^p \rightarrow \{T, F\}$ is a **primitive recursive predicate** if the set

$$A = \{\mathbf{x} \in \mathbb{N}^p \mid P(\mathbf{x}) \text{ is true}\}$$

is a primitive recursive set.

In other words, this means that the characteristic function:

$$\chi_P(\mathbf{x}) = \begin{cases} 1 & \text{if } P(\mathbf{x}) \text{ is true} \\ 0 & \text{otherwise} \end{cases}$$

is primitive recursive.

Proposition 6.

If P and Q are primitive recursive p -place predicates, then $\neg P$, $P \wedge Q$, $P \vee Q$, $P \rightarrow Q$, and $P \leftrightarrow Q$ are also primitive recursive.

Proof. Let A and B be the primitive recursive sets associated with P and Q , respectively.

- The set $\mathbb{N}^p \setminus A$ is primitive recursive, so $\neg P$ is a primitive recursive predicate. (Another proof: $\chi_{\neg P}(\mathbf{x}) = 1 \dot{-} \chi_P(\mathbf{x})$.)
- The set associated with $P \wedge Q$ is $A \cap B$, which is primitive recursive (alternatively, $\chi_{P \wedge Q}(\mathbf{x}) = \chi_P(\mathbf{x}) \times \chi_Q(\mathbf{x})$).
- The set associated with $P \vee Q$ is $A \cup B$ (alternatively, $\chi_{P \vee Q}(\mathbf{x}) = \text{sgn}(\chi_P(\mathbf{x}) + \chi_Q(\mathbf{x}))$).
- The connectives $\rightarrow$ and $\leftrightarrow$ can be expressed using the preceding connectives.

□

Proposition 7.

The predicates “ $x > y$ ” and “ $x \geq y$ ” are primitive recursive.

Proof. Their characteristic functions are:

- $\chi_{>}(x, y) = \text{sgn}(x \dot{-} y)$
- $\chi_{\geq}(x, y) = \text{sgn}((x + 1) \dot{-} y)$

□

2.3. Bounded schemas

Searching for elements defined by a minimum, or by the quantifiers $\forall$ or $\exists$, preserves primitive recursiveness provided the search is bounded. This will be fundamental for the applications to arithmetic discussed immediately below.

Bounded minimization schema

Let $P(\mathbf{x}, t)$ be a predicate. Fix $\mathbf{x}$ and y . We can consider:

$$\min\{t \leq y \mid P(\mathbf{x}, t) \text{ is true}\}.$$

If the set $\{t \leq y \mid P(\mathbf{x}, t) \text{ is true}\}$ is empty, by convention we set $\min = y + 1$.

If A is a set, we similarly define:

$$\min\{t \leq y \mid (\mathbf{x}, t) \in A\}$$

Proposition 8.

If P is a primitive recursive predicate, then the function $f(\mathbf{x}, y)$ defined by:

$$f(\mathbf{x}, y) = \min\{t \leq y \mid P(\mathbf{x}, t) \text{ is true}\}$$

is primitive recursive.

The same holds for a primitive recursive set A .

Example.

Consider the integer square-root function $\mathbb{N} \rightarrow \mathbb{N}$, which maps y to $\lfloor \sqrt{y} \rfloor$ (for example, $\lfloor \sqrt{10} \rfloor = 3$). In fact, $\lfloor \sqrt{y} \rfloor$ is the least natural number whose successor has a square greater than y ; that is:

$$\lfloor \sqrt{y} \rfloor = \min\{t \leq y \mid (t+1)^2 > y\}$$

It is therefore primitive recursive.

Proof. Intuitively, for fixed $\mathbf{x}$ and y , only finitely many values $t \leq y$ must be tested, so this is easy to compute algorithmically. Fix $\mathbf{x}$ and find the recursion formula.

- Base case: ($y = 0$). If $P(\mathbf{x}, 0)$ is true, set $g(\mathbf{x}) = 0$; otherwise, set $g(\mathbf{x}) = 1$ (by the convention $\min = y + 1$).
- Recursion formula. We seek $f(\mathbf{x}, y + 1) = \min\{t \leq y + 1 \mid P(\mathbf{x}, t)\}$. A definition by cases gives the formula (see Proposition 5):

$$f(\mathbf{x}, y + 1) = \begin{cases} f(\mathbf{x}, y) & \text{if } f(\mathbf{x}, y) \leq y \\ y + 1 & \text{if } f(\mathbf{x}, y) = y + 1 \text{ and } P(\mathbf{x}, y + 1) \text{ is true} \\ y + 2 & \text{if } f(\mathbf{x}, y) = y + 1 \text{ and } P(\mathbf{x}, y + 1) \text{ is false} \end{cases}$$

□

Bounded quantification**Proposition 9.**

Let $A \subset \mathbb{N}^{p+1}$ be a primitive recursive set. Then the sets:

- $B = \{(\mathbf{x}, y) \mid \forall t \leq y, (\mathbf{x}, t) \in A\}$
- $C = \{(\mathbf{x}, y) \mid \exists t \leq y, (\mathbf{x}, t) \in A\}$

are also primitive recursive.

Proof. Let $A_t = \{(\mathbf{x}, t) \in A\}$. Then $\chi_B = \chi_{A_0} \times \chi_{A_1} \times \cdots \times \chi_{A_y} = \prod_{t=0}^y \chi_{A_t}$ and $\chi_C = \text{sgn}(\sum_{t=0}^y \chi_{A_t})$. The bounded sum and product are easily defined by recursion. □

This also works in predicate form. For example, “being a perfect square” is a primitive recursive predicate. Recall that y is a perfect square if there exists t such that $y = t^2$ ($0, 1, 4, 9, 16, 25, \dots$ are perfect squares). This predicate is defined by bounded quantification:

$$\exists t \leq y, y = t^2.$$

2.4. Arithmetic

Proposition 10.

The following functions are primitive recursive:

- $q(x, y)$, the quotient in the Euclidean division of x by y (with the convention $q(x, 0) = x + 1$);
- $\pi(n)$, which computes the n th prime number;
- $\text{exponent}(n, i)$, which computes the exponent of $\pi(i)$ in the prime factorization of n .

Proposition 11.

The set $\mathcal{P}$ of prime numbers is primitive recursive.

We prove these two propositions together.

Proof.

- $q(x, y) = \min\{q \leq x \mid (q + 1) \times y > x\}$ is therefore primitive recursive.
- The remainder $r(x, y) = x \dot{-} (q(x, y) \times y)$ is primitive recursive as well.
- The predicate $\text{is_divisible}(x, y)$ (which is true if y divides x) is obtained from $r(x, y) = 0$. It is therefore primitive recursive.
- Being prime means being strictly greater than 1 and having no divisors other than 1 and itself. Thus $\mathcal{P}$ is a primitive recursive set:

$$\mathcal{P} = \{x \mid x > 1\} \cap \{x \mid \forall y \leq x \quad (y = 1 \vee y = x \vee \neg \text{is_divisible}(x, y))\}$$

Equivalently, the predicate $\text{is_prime}(n)$ is primitive recursive.

- The function $\pi(n)$ that returns the n th prime number (denoted by π_n , with $\pi_1 = 2$) satisfies the following recursion relation, which makes it primitive recursive:

$$\pi(n + 1) = \min \{t \leq \pi(n)! + 1 \mid t > \pi(n) \wedge \text{is_prime}(t)\}$$

Recall that there is always a prime number between π_n and $\pi_n! + 1$ (or even between m and $2m$).

- The exponent of π_i in the prime factorization of n is defined by:

$$\text{exponent}(n, i) = \min \{k \leq n \mid \neg \text{is_divisible}(n, \pi_i^{k+1})\}$$

It is therefore primitive recursive.

□

Let us finish with a famous counterexample.

Example.

The Ackermann function $A : \mathbb{N}^2 \rightarrow \mathbb{N}$ is not primitive recursive. It is defined as follows:

- $A(0, n) = n + 1$
- $A(m + 1, 0) = A(m, 1)$
- $A(m + 1, n + 1) = A(m, A(m + 1, n))$

For fixed m , the function $A_m : \mathbb{N} \rightarrow \mathbb{N}, n \mapsto A(m, n)$ is primitive recursive. But the function $A : \mathbb{N}^2 \rightarrow \mathbb{N}$ is not, because its nested recursion makes it grow faster than any primitive recursive function.

3. Application to Gödel coding

We will justify, in the world of natural numbers, that all the operations we wish to perform on Gödel numbers are carried out by primitive recursive functions; in other words, as presented earlier, the computations are performed by algorithms with “for” loops.

3.1. Length, exponent, concatenation, substitution

Consider a Gödel number $n = \pi_1^{c_1} \times \cdots \times \pi_l^{c_l}$ (with $c_i \geq 1$).

- First, we want to recover l from n by a primitive recursive procedure:

$$\text{length}(n) = \min\{i \leq n \mid \pi_{i+1} \text{ does not divide } n\}$$

- We have already studied the function $\text{exponent}(n, i)$, which returns the exponent c_i . Thus, we can decode a Gödel number using a primitive recursive function.
- We saw the concatenation formula in the previous chapter. If $n = \prod_{i=1}^l \pi_i^{\alpha_i}$ and $m = \prod_{j=1}^{l'} \pi_j^{\beta_j}$, then:

$$n * m = n \times \prod_{j=1}^{l'} \pi_{j+l}^{\beta_j}$$

These are finite products, so this is a primitive recursive operation.

- Consider the function $\text{substitution}(p, v, n)$: for a formula $\mathcal{P}$ (given by its Gödel number p), with free variable x (given by its code v), it returns the Gödel number of $\mathcal{P}(n)$ (that is, $\mathcal{P}(n/x)$).

First extract each code c_i from $\mathcal{P}$ and compare it with the code of the variable x . If they are equal, replace that symbol with the term n using concatenation. This is a sequence of primitive recursive functions, so substitution is primitive recursive.

3.2. Variables, terms, formulas

Deciding whether a natural number n represents a variable, a term, or a valid formula gives primitive recursive predicates. We refer to the previous chapter without further details.

- $\text{is_variable}(n)$
- $\text{is_term}(n)$
- $\text{is_formula}(n)$

3.3. Peano axioms

We are given a natural number n (a Gödel number) and must decide whether the formula it encodes is one of the Peano axioms. Consider the first Peano axiom “ $\forall x S(x) \neq 0$ ”, which we would write in $\mathcal{L}_0$ as “ $\forall x \neg Sx = 0$ ”; denote its Gödel number by n_1 .

To determine whether n encodes this axiom, it suffices to test whether $n = n_1$. We do this for all seven axioms.

The difficulty comes from the induction axioms:

$$\text{Rec}_{\mathcal{P}(x), y} : \mathcal{P}(0) \wedge [\forall y \mathcal{P}(y) \rightarrow \mathcal{P}(S(y))] \rightarrow \forall x \mathcal{P}(x)$$

This is an axiom schema: there is one axiom for each formula $\mathcal{P}(x)$. Thus, a priori, there are infinitely many formulas and variables to test. Fortunately, for a given Gödel number n , if x is a variable occurring

in the formula encoded by n , then its code v is necessarily less than n . Likewise, if $\mathcal{P}$ is a subformula of the formula encoded by n , then its code p also satisfies $p \leq n$.

We can therefore construct a primitive recursive predicate of the form:

$$\begin{aligned} \text{is_induction_axiom}(n) = & \exists \mathcal{P} \text{ with code } p \leq n \quad \exists x \text{ with code } v \leq n \quad \exists y \text{ with code } w \leq n \\ & \text{such that is_formula}(\mathcal{P}) \text{ AND is_variable}(x) \text{ AND is_variable}(y) \\ & \text{AND } n \text{ encodes the formula } \mathcal{R}ec_{\mathcal{P}(x),y} \end{aligned}$$

Thus, we can recognize all the Peano axioms by a primitive recursive procedure.

3.4. Hilbert-style proofs

In the first two parts of this book, we presented proofs and their rules of inference in natural-deduction notation. As its name suggests, natural-deduction proofs are written in a form close to that of a mathematical proof. However, they have assumptions and temporary assumptions, and this presentation is not suited to algorithmic processing.

We now consider Hilbert-system notation, which is better suited to Gödel coding.

- First, we dispense with assumptions. If we had to prove $\mathcal{P} \vdash \mathcal{Q}$, we would prove $\vdash (\mathcal{P} \rightarrow \mathcal{Q})$.
- There is only one rule of inference for propositional logic:

$$\mathcal{P}, \mathcal{P} \rightarrow \mathcal{Q} \quad \vdash \quad \mathcal{Q} \quad (\text{modus ponens})$$

- On the other hand, axioms appear: formulas assumed to be always true (regardless of what we wish to prove).
- For propositional logic, there are three axioms (variants exist; this is the Łukasiewicz variant):
 1. $\mathcal{P} \rightarrow (\mathcal{Q} \rightarrow \mathcal{P})$ (adding an assumption)
 2. $(\mathcal{P} \rightarrow (\mathcal{Q} \rightarrow \mathcal{R})) \rightarrow ((\mathcal{P} \rightarrow \mathcal{Q}) \rightarrow (\mathcal{P} \rightarrow \mathcal{R}))$ (distribution of implication)
 3. $(\neg \mathcal{Q} \rightarrow \neg \mathcal{P}) \rightarrow (\mathcal{P} \rightarrow \mathcal{Q})$ (contrapositive)
- Thus, to check whether a sequence $\mathcal{P}_1, \dots, \mathcal{P}_l$ of formulas is indeed a proof of $\mathcal{Q}$, it suffices that for every i :
 - either $\mathcal{P}_i$ is one of the axioms;
 - or $\mathcal{P}_i$ is obtained by *modus ponens* from preceding lines, that is, there exist $j, k < i$ such that $\mathcal{P}_k$ is $\mathcal{P}_j \rightarrow \mathcal{P}_i$;

and finally, we check that $\mathcal{P}_l = \mathcal{Q}$.

- For first-order logic, we add the following axioms:
 - $(\forall x \mathcal{P}(x)) \rightarrow \mathcal{P}(t)$
 - $[\forall x (\mathcal{P} \rightarrow \mathcal{Q}(x))] \rightarrow [(\mathcal{P} \rightarrow \forall x \mathcal{Q}(x))]$ if $\mathcal{P}$ has no free variable x .
 - $x = x$
 - $x = y \rightarrow (\mathcal{P}(x) \rightarrow \mathcal{P}(y))$
- We also add a second rule of inference, a variant of the $\forall$ -introduction rule:

$$\text{if } \vdash \mathcal{P} \quad \text{then } \vdash \forall x \mathcal{P}$$

3.5. Being a proof

We can now see that checking whether a super-Gödel number N is a proof of n is easy using the Hilbert system. Recall that there are no assumptions (to prove $\Gamma \vdash Q$, we prove $\vdash (\Gamma \rightarrow Q)$), that the rules of inference are very simple (*modus ponens* and $\forall$ introduction), and that a proof is merely a linear sequence of formulas.

We thus construct a primitive recursive predicate $\text{is_proof}(N, n)$ that checks whether the sequence of formulas $\mathcal{P}_i$ encoded by the super-Gödel number N is a valid proof of the formula encoded by the Gödel number n .

This predicate is constructed as follows (for propositional logic):

- extract the expressions $\mathcal{P}_i$, $i = 1, \dots, l$, from N and the expression Q from n ;
- check that these expressions are valid formulas;
- for each $i = 1, \dots, l$:
 - check whether $\mathcal{P}_i$ is an axiom;
 - or whether $\mathcal{P}_i$ follows by *modus ponens* from preceding lines (that is, whether there exist $j, k < i$ such that $\mathcal{P}_k$ is $\mathcal{P}_j \rightarrow \mathcal{P}_i$).
- check whether the last formula $\mathcal{P}_l$ is indeed Q .