CHAPTER 4. COMPUTABILITY AND DECIDABILITY

1. Introduction

By definition, an n-ary function F on $\mathbb{N}$ assigns to every n-tuple $k_1, \ldots, k_n$ of elements of $\mathbb{N}$ a unique $l \in \mathbb{N}$ which is the *value* of F at $k_1, \ldots, k_n$, $F(k_1, \ldots, k_n)$. In most common examples you are also given a procedure (an *algorithm*) to compute the value of the function from the given $k_1, \ldots, k_n$. For example, let $F(k, n)$ be the remainder of n upon division by k, that is, $F(k, n)$ is the unique r with $0 \leq r < k$ such that $n = k \cdot l + r$ for some $l \in \mathbb{N}$ (or $r = n$ if $k = 0$). The value of F at any k, n can be computed using the long division algorithm taught in elementary school.

The following is an informal definition of *algorithm*.

Definition 1.1. An *algorithm* for computing the values of an n-ary function F on $\mathbb{N}$ is a finite list of instructions such that given any input $k_1, \ldots, k_n \in \mathbb{N}$ the instructions can be mechanically carried out, and after a finite number of steps the computation stops and produces the unique answer $F(k_1, \ldots, k_n)$.

An algorithm can be thought of as a computer program, and applying the algorithm as running the program with some specific input. Implicit in the definition is that no random procedure, like flipping a coin, can be used, since this would lead to different answers at different times.

Definition 1.2. A function F on $\mathbb{N}$ is *computable* iff there is some algorithm which computes its values.

"Most" functions on $\mathbb{N}$ are *not* computable, due to the following important result.

Theorem 1.1. *There are only countably many computable functions on* $\mathbb{N}$.

Proof. (outline) Since algorithms are given by a *finite* list of instructions, they are finite sequences of symbols from a finite alphabet. Hence there are only countably many algorithms, and therefore just countably many computable functions on $\mathbb{N}$. □

Since there are uncountably many functions on $\mathbb{N}$ it follows that uncountably many of then are not computable. It is harder to give explicit examples of uncomputable functions, however.

Computability also applies to relations by considering their characteristic functions.

Definition 1.3. Let R be an n-ary relation on $\mathbb{N}$. The *characteristic function* of R is the n-ary function K_R defined by $K_R(k_1, \ldots, k_n) = 1$ if $R(k_1, \ldots, k_n)$ holds and 0 otherwise.

Definition 1.4. A relation R on $\mathbb{N}$ is *decidable* iff its characteristic function is computable.

The connection of computability with **PA** is given by the following remarkable result of K. Gödel used in the proof of his Incompleteness Theorem.

Theorem 1.2. *(a) A function F on $\mathbb{N}$ is computable iff it is definable in* **PA**.

(b) A relation R on $\mathbb{N}$ is decidable iff both it and its negation are definable in **PA**.

The right-to-left implications are not difficult. For example, suppose the unary function F is definable in **PA** by the formula $\varphi(x, y)$. Then $F(k) = l$ iff $\mathbf{PA} \vdash \varphi(\bar{k}, \bar{l})$. Given $k \in \mathbb{N}$ we know there is exactly one l such that $\mathbf{PA} \vdash \varphi(\bar{k}, \bar{l})$ (namely, $l = F(k)$). So, to compute $F(k)$ we simply search the deductions from **PA** until we find a deduction of a sentence of that form, and then that l must be $F(k)$.

The reverse implications are much harder and are what is used in defining a sentence σ of $\mathcal{L}_{\mathcal{N}}$ such that $\mathcal{N} \models \sigma$ but $\mathbf{PA} \not\models \sigma$. The rest of this Chapter is devoted to giving a precise definition of computability and explaining the proof of Theorem 1.2.

2. Defining Computability

We need to have a formal definition of computable functions in order to prove a result like Theorem 1.2. There are many (equivalent) ways in which this can be done, for example using Turing Machines. We choose a recursive definition which is easier to work with, although not as intuitive as Turing's machine approach.

We define the set of computable (or *recursive*) functions on $\mathbb{N}$ as all functions obtained from s, $+$, $\cdot$, $K_<$ and the constant 0 using three rules. The first of these rules is *Composition* (see also section 2 of Chapter 2):

Given an n-ary function H on $\mathbb{N}$ and n k-ary functions $G_1, \ldots, G_n$ on $\mathbb{N}$, we obtain the k-ary function F on $\mathbb{N}$ defined by

$F(x_1, \ldots, x_k) = H(G_1(x_1, \ldots, x_k), \ldots, G_n(x_1, \ldots, x_k))$.

We quietly allow dummy variables and changes in the order of variables so that, for example, the function $H(x, y, z) = x \cdot (y + x) + y \cdot z$ is obtained by Composition from $+$ and $\cdot$.

The second rule is *Primitive Recursion*:

Given an n-ary function G on $\mathbb{N}$ and an $(n+2)$-ary function H on $\mathbb{N}$, we obtain the $(n+1)$-ary function F defined by the pair of equations

$F(0, y_1, \ldots, y_n) = G(y_1, \ldots, y_n)$,

$F(x+1, y_1, \ldots, y_n) = H(x, F(x, y_1, \ldots, y_n), y_1, \ldots, y_n)$.

Ignoring the parameters $y_1, \ldots, y_n$ all this means is that we specify the value of F at $x = 0$ and then specify the value of F at $x + 1$ in terms of the value of F at x. Here too we allow dummy variables and changes in the order of the variables. Examples include $x!$ obtained by Primitive Recursion from

the clauses $0! = 1$ and $(x+1)! = (x+1) \cdot x!$, and $exp(x,y) = x^y$ obtained by Primitive Recursion (on the variable y) from the clauses $x^0 = 1$ and $x^{y+1} = x \cdot x^y$.

The final rule may be less familiar – it is *Unbounded Search*, or *μ-recursion*. We first introduce the μ notation.

Definition 2.1. Let R be an $(n+1)$-ary relation on $\mathbb{N}$ such that for all $x_1, \ldots, x_n \in \mathbb{N}$ there is some $y \in \mathbb{N}$ such that $R(x_1, \ldots, x_n, y)$ holds. Then the n-ary function $(\mu y)[R(x_1, \ldots, x_n, y)]$ is defined to be the *least* $y \in \mathbb{N}$ such that $R(x_1, \ldots, x_n, y)$ holds.

For example, we know that for every $x \in \mathbb{N}$ there is some $y \in \mathbb{N}$ such that $x < y$ and y is prime. Therefore the function $F(x)$ defined as the least $y > x$ such that y is prime is obtained by μ-recursion from the relation "$x < y$ and y is prime".

The rule of *Unbounded Search* or *μ-recursion* is the following:

Given an $(n+1)$-ary function G on $\mathbb{N}$ such that for every $x_1, \ldots, x_n \in \mathbb{N}$ there is some $y \in \mathbb{N}$ with $G(x_1, \ldots, x_n, y) = 0$, we obtain the n-ary function $F(x_1, \ldots, x_n) = (\mu y)[G(x_1, \ldots, x_n, y) = 0]$.

Finally, a function F on $\mathbb{N}$ is *computable*, or *recursive*, iff it can be obtained from s, $+$, $\cdot$, $K_<$ and 0 (the "starting functions") by a finite number of applications of the three rules.

For example, every polynomial with non-negative integer coeficients, $F(x) = a_n \cdot x^n + \cdots + a_1 \cdot x + a_0$ is recursive, being obtained from the starting functions using just composition. $x!$ and x^y are recursive, since they are obtained using Primitive Recursion, as shown above.

We leave it to the reader to show (using Primitive Recursion) that the function K_0 defined by $K_0(0) = 1$ and $K_0(n) = 0$ for all $n > 0$ is recursive. It follows that if G is recursive and for every $x_1, \ldots, x_n \in \mathbb{N}$ there is some $y \in \mathbb{N}$ with $G(x_1, \ldots, x_n, y) > 0$, then the function F defined by $F(x_1, \ldots, x_n) = (\mu y)[0 < G(x_1, \ldots, x_n, y)]$ is recursive.

For example, if K_P is the characteristic function of the set of prime numbers, then $F(x)$ defined as the least y such that $x < y$ and y is prime can be written as $(\mu y)[0 < K_<(x,y) \cdot K_P(y)]$ and therefore is recursive, since K_P is recursive by Lemma 2.2(b) below.

All of the other functions used in elementary number theory can similarly be shown to be recursive.

Definition 2.2. A relation R on $\mathbb{N}$ is *recursive* provided its characteristic function K_R is recursive.

We can frequently show directly that a relation is recursive without considering its characteristic function, using the following facts.

Lemma 2.1. (a) *$<$, $>$, $=$ are recursive.*

(b) *If R is recursive, so is $\neg R$.*

(c) *If R and S are both recursive then so are $(R \wedge S)$ and $(R \vee S)$.*

(d) *If* $R(x_1, \ldots, x_n, y)$ *is recursive and for every* $x_1, \ldots x_n \in \mathbb{N}$ *there is some* $y \in \mathbb{N}$ *such that* $R(x_1, \ldots, x_n, y)$ *holds then the function* $F(x_1, \ldots, x_n) = (\mu y)[R(x_1, \ldots, x_n, y)$ *holds*$]$ *is recursive.*

$\neg R$ is the relation which holds of $k_1, \ldots, k_n$ iff $R(k_1, \ldots, k_n)$ fails. $(R \wedge S)$ is the relation which holds iff both R and S hold, and similarly for $(R \vee S)$.

Using Lemma 2.1 we obtain the following.

Lemma 2.2. (a) *The divisibility relation,* $x|y$*, is recursive.*

(b) *The set of all primes is recursive.*

(c) *The function* p *enumerating all primes in increasing order is recursive.*

Proof. (a) The relation $\big((y = x \cdot z) \vee y < z\big)$ is recursive, by Lemma 2.1(a)(c). $\mathcal{N} \models \forall x \forall y \exists z \big((y = x \cdot z) \vee y < z\big)$. Hence, by Lemma 2.1(d), the function $F(x, y) = (\mu y)[(y = x \cdot z) \vee y < z]$ is recursive. Since $x|y$ iff $F(x, y) \leq y$ we conclude that $x|y$ is recursive.

(b) This is left to the reader.

(c) Use Primitive Recursion — $p(0) = 2$ and $p(n + 1)$ is the least prime strictly bigger than $p(n)$. □

We normally write p_k instead of $p(k)$ in part (c).

In contrast to Lemma 2.1 it is usually *not* true that $\exists y R(x_1, \ldots, x_n, y)$ is recursive provided $R(x_1, \ldots, x_n, y)$ is recursive. We do, however, have the following:

Lemma 2.3. *Assume that* $R(x_1, \ldots, x_n, y, z)$ *is recursive. Then* $\exists y \big(y \leq z \wedge R(x_1, \ldots, x_n, y, z)\big)$ *is recursive.*

Proof. As in the proof that $|$ is recursive,
$F(x_1, \ldots, x_n, z) = (\mu y)\big(R(x_1, \ldots, x_n, y, z) \vee z < y\big)$ is recursive, and
$\exists y \big(y \leq z \wedge R(x_1, \ldots, x_n, y, z)\big)$ holds iff $F(x_1, \ldots, x_n, z) \leq z$. □

Definition 2.3. $rem(x, w)$ is the remainder when x is divided by w, unless $w = 0$ in which case $rem(x, w) = x$.

Lemma 2.4. *The function* rem *is recursive, and can be defined without use of Primitive Recursion.*

Proof. $rem(x, w) = y$ iff $\exists z \big((z \leq x \wedge x = w \cdot z + y \wedge y < w) \vee (w = 0 \wedge y = x)\big)$, and this relation is recursive by Lemma 2.3. Since it is true that for all $x, w \in \mathbb{N}$ there is exactly one y such that $rem(x, w) = y$, it follows by Lemma 2.1(d) that rem is recursive. □

3. Sequences and Sequence Numbers.

Gödel was the first to realize that one can identify finite sequences of natural numbers with natural numbers in such a way that the standard operations on sequences become computable functions of the associated numbers.

Definition 3.1. Let $(k_0, \ldots, k_{n-1})$ be a finite sequence of natural numbers. The *sequence number* of the sequence is

$$\langle k_0, \ldots, k_{n-1} \rangle = 2^{k_0+1} \cdot 3^{k_1+1} \cdot \cdots \cdot p_{n-1}^{k_{n-1}+1}.$$

Lemma 3.1. *If* $\langle k_0, \ldots, k_{n-1} \rangle = \langle l_0, \ldots, l_{m-1} \rangle$ *then* $n = m$ *and* $k_i = l_i$ *for all* $i < n = m$.

Proof. Immediate from the Uniqueness of Prime Power Factorization, since all exponents are positive. □

Theorem 3.1. (a) *The set* Seq *of all sequence numbers is recursive.*

(b) *There is a recursive function* ln *such that for every sequence number* $k = \langle k_0, \ldots, k_{n-1} \rangle$, $ln(k) = n$, *the* length *of the sequence* $(k_0, \ldots, k_{n-1})$.

(c) *There is a binary recursive function* C *such that for every sequence number* $k = \langle k_0, \ldots, k_{n-1} \rangle$ *and every* $i < n$, $C(k, i) = k_i$, *the* i^{th} *term in the sequence.*

(d) *There is a binary recursive function* $*$ *such that for all sequence numbers* $k = \langle k_0, \ldots, k_{n-1} \rangle$ *and* $l = \langle l_0, \ldots, l_{m-1} \rangle$,
$k * l = \langle k_0, \ldots, k_{n-1}, l_0, \ldots, l_{m-1} \rangle$,
the sequence number of the concatenation *of the sequences* $(k_0, \ldots, k_{n-1})$ *and* $(l_0, \ldots, l_{m-1})$.

(e) *There is a binary recursive function* In *such that for every sequence number* $k = \langle k_0, \ldots, k_{n-1} \rangle$ *and every* $i < n$, $In(k, i) = \langle k_0, \ldots, k_{i-1} \rangle$, *the sequence number of the* initial segment of length i *of the sequence* $(k_0, \ldots, k_{n-1})$.

Of course, in parts (b)-(e) the functions must be defined for all natural numbers, but we don't care how this is done.

Notation. We normally write $(k)_i$ for $C(k, i)$.

The rule of Primitive Recursion allows us to define the value of a function at a positive integer n in terms of its value at the immediately preceding integer $n-1$. But there are times we need to define the value of a function at an integer in terms of its values at several preceding integers. For example the Fibonacci numbers are defined by the recursion $f(0) = 0$, $f(1) = 0$, $f(n) = f(n-2) + f(n-1)$ for all $n > 1$. Such definitions are all included under *Course-of-Values Recursion.*

Definition 3.2. Given a function $F(x)$ the *course-of-values of* F is the function $\overline{F}$ defined by $\overline{F}(0) = 1$, $\overline{F}(n) = \langle F(0), \ldots, F(n-1) \rangle$ for $n > 0$.

We can also define the course-of-values of a function with more than one argument in analogous fashion, but for simplicity we restrict to functions of one argument.

Lemma 3.2. *F is recursive iff $\overline{F}$ is recursive.*

Proof. $F(n) = \left(\overline{F}(s(n))\right)_n$, so F is recursive provided $\overline{F}$ is. For the other direction, $\overline{F}$ is defined from F via the primitive recursion $\overline{F}(0) = 1$, $\overline{F}(n+1) = \overline{F}(n) * \langle F(n) \rangle$. □

Theorem 3.2. (Course-of-Values Recursion) *Assume that $H(x)$ is recursive. Then so is F defined by $F(n) = H(\overline{F}(n))$.*

Proof. We define $\overline{F}$ from H via the following primitive recursion. $\overline{F}(0) = 1$, $\overline{F}(n+1) = \overline{F}(n) * \langle H(\overline{F}(n))\rangle$. Then F is also recursive by Lemma 3.2. □

There is also a relational form of Course-of-Values Recursion, which follows easily from the functional form.

Theorem 3.3. *Assume that $S(x)$ is recursive. Then so is R defined by $R(n)$ holds iff $S(\overline{K_R}(n))$ holds.*

Proof. $K_R(n) = K_S(\overline{K_R}(n))$, so K_R is recursive by Theorem 3.2 and hence R is recursive by definition. □

4. Definability of Recursive Functions in **PA**

In this section we show that all recursive functions (and hence also all recursive relations) are definable in **PA**. It suffices to show that the starting functions are definable, and that the three rules of Composition, Primitive Recursion, and μ-recursion preserve definability.

Recall that $F(x_1, \ldots, x_n)$ is definable in **PA** by $\varphi(x_1, \ldots, x_n, y)$ iff F is definable in $\mathcal{N}$ by φ and, in addition, if $F(k_1, \ldots, k_n) = l$ then also $\mathbf{PA} \models \varphi(\bar{k}_1, \ldots, \bar{k}_n, \bar{l})$.

In the Appendix we give results which imply the definability in **PA** of s, $+$, $\cdot$, and $K_<$.

To show that Composition preserves definability in **PA**, suppose (for example) that $F(x) = H(G_1(x), G_2(x))$ where H is definable in **PA** by $\varphi_H(y_1, y_2, z)$ and G_i is definable in **PA** by $\varphi_{G_i}(x, y_i)$. We claim that F is definable in **PA** by $\varphi_F(x, z)$ defined as

$$\exists y_1 \exists y_2 \big(\varphi_H(y_1, y_2, z) \wedge \varphi_{G_1}(x, y_1) \wedge \varphi_{G_2}(x, y_2)\big).$$

First note that $F(k) = m$ iff there are l_1 and l_2 such that $m = H(l_1, l_2)$, $G_1(k) = l_1$, and $G_2(k) = l_2$. Since H, G_1, and G_2 are definable in **PA**, we see that

$$F(k) = m \text{ iff } \mathbf{PA} \models \varphi_H(\bar{l}_1, \bar{l}_2, \bar{m}) \wedge \varphi_{H_1}(\bar{k}, \bar{l}_1) \wedge \varphi_{H_2}(\bar{k}, \bar{l}_2).$$

It follows that φ_F defines F in **PA**.

What we have done is show that the formula which naturally defines F in $\mathcal{N}$ also defines F in **PA**. Similarly with μ-recursion, the formula which naturally defines the resulting function in $\mathcal{N}$ also defines it in **PA**. The problem with Primitive Recursion is that there is no obvious way to even *define* the resulting function in $\mathcal{N}$!

Consider, for example, k^n defined by the recursion $k^0 = 1$ and $k^{n+1} = k^n \cdot k$. As a first attempt at giving an explicit definition of exponentiation, we observe that $k_n = m$ iff there is a sequence $a_0, \ldots, a_n$ such that $a_0 = 1$, $a_n = m$, and for all $i < n$ $a_{i+1} = a_i \cdot k$. What we need is a way of coding a sequence $a_0, \ldots, a_n$ by a number a so that the function which maps (a, i) to a_i is recursive and defined without using Primitive Recursion. Gödel defined such a function for the proof of his Incompleteness Theorem.

Theorem 4.1. *There is a recursive function $\beta(x, y)$, defined without using Primitive Recursion, such that for every n and every sequence $a_0, \ldots, a_n$ there is some a such that $\beta(a, i) = a_i$ for all $i \leq n$.*

We discuss the proof of this result in the next section. What we do here is use this Theorem to show how to define exponentiation without using Primitive Recursion.

$k^n = m$ iff $\exists a\big(\beta(a, 0) = 1 \wedge \beta(a, n) = m \wedge \forall i(i < n \rightarrow \beta(a, i + 1) = \beta(a, i) \cdot k)\big)$. Therefore $G(k, n) = (\mu a)[\beta(a, 0) = 1 \wedge \forall i(i < n \rightarrow \beta(a, i+1) = \beta(a, i) \cdot k)]$ is recursive, and $k^n = \beta(G(k, n), n)$.

The same argument shows that every use of Primitive Recursion can be eliminated, and so every recursive function can be obtained from the starting functions using just Composition and μ-recursion. In particular, then, all recursive functions, and hence relations, are definable in **PA**.

5. The β-function

In this section we show how to prove the following, slightly weaker, form of Theorem 4.1.

Theorem 5.1. *There is a recursive function $\beta(w, x, y)$, defined without using Primitive Recursion, such that for every n and every sequence $a_0, \ldots, a_n$ there are m and a such that for every $i \leq n$, $\beta(m, a, i) = a_i$.*

Our main tool is the Chinese Remainder Theorem. Recall that rem is recursive and defined without using Primitive Recursion (by Lemma 2.4).

Theorem 5.2. *Let $m_0, \ldots, m_n$ be pairwise relatively prime positive integers. Then for any sequence $a_0, \ldots, a_n$ there is some a such that $rem(a, m_i) = a_i$ for all $i \leq n$.*

What we want to do is code $m_0, \ldots, m_n$ by a single number m, uniformly in n. The key Lemma is the following.

Lemma 5.1. *Let $h(m, i) = 1 + (i + 1) \cdot m$. Let m be divisible by every positive integer $\leq n$. Then the numbers $h(m, 0), \ldots, h(m, n)$ are pairwise relatively prime.*

Proof. If the prime p divides both $h(m, i)$ and $h(m, j)$ where $0 \leq i < j \leq n$, then $p|(j - i) \cdot m$, hence $p|m$ (since $1 \leq (j - i) \leq n$ and by the choice of m). Therefore $p|(i + 1) \cdot m$, and so $p|1$ – a contradiction. □

We can now prove Theorem 5.1. Define $\beta(w, x, y)$ to be $rem(x, h(w, y))$. Given n choose m as in Lemma 5.1. Given a sequence $a_0, \ldots, a_n$ of length $n + 1$, let a be the code given by the Chinese Remainder Theorem relative to the relarively prime numbers $h(m, 0), \ldots, h(m, n)$. Then $\beta(m, a, i) = a_i$ for all $i \leq n$ by the Chinese Remainder Theorem.

We can obtain the two argument β function of Theorem 4.1 by coding the pair (m, a) using a definable pairing function, such as $OP(x, y) = (x + y)^2 + x + 1$. The details are left to the reader.