

2. A Précis of Classical Computability Theory

Turing machines form the core of computability theory, or recursion theory as it is also known. This chapter introduces basic notions and results and readers already familiar with them can safely skip it. The exposition is based on standard references [18, 39, 67, 83, 109]. In the discussion that follows, the symbol $\mathbb{N}$ will stand for the set of positive integer numbers including zero and $\mathbb{Q}$ will stand for the set of rational numbers.

2.1 Turing Machines

As explained in the introduction, a Turing machine is a conceptual computing device consisting of an *infinite tape*, a *controlling device*, and a *scanning head* (see Fig. 2.1). The tape is divided into an infinite number of cells. The scanning head can read and write symbols in each cell. The symbols are elements of a set $\Sigma = \{S_1, \dots, S_n\}$, $n \geq 1$, which is called the *alphabet*. Usually, there is an additional symbol, $\sqcup$, called the *blank* symbol, and when this symbol is written on a cell by the scanning head, the effect of this operation is the erasure of the symbol that was printed on this particular cell. At any moment, the machine is in a *state* q_i , which is a member of a finite set $Q = \{q_0, q_1, \dots, q_r\}$, $r \geq 0$. The controlling device is actually a lookup table that is used to determine what the machine has to do next at any given moment. In particular, the action a machine has to take depends on its current state and the symbol that is printed on the cell the scanning head has just finished scanning. If no action has been specified for a particular combination of state and symbol, the machine halts. Usually, the control device is specified by a finite set of *quadruples*, which are special cases of expressions.

Definition 2.1.1 An *expression* is a string of symbols chosen from the list $q_0, q_1, \dots; \sqcup, S_1, \dots; R, L$.

A quadruple can have one of the following forms:

$$q_i S_j S_k q_l, \quad (2.1)$$

$$q_i S_j L q_l, \quad (2.2)$$

$$q_i S_j R q_l. \quad (2.3)$$

Note that $L, R \notin \Sigma$ and $S_j, S_k \in \Sigma \cup \{\sqcup\}$. The quadruple (2.1) specifies that if the machine is in state q_i and the cell that the scanning head scans contains the symbol S_j , then the scanning head replaces S_j by S_k and the machine enters state q_l . The quadruples (2.2) and (2.3) specify

that if the machine is in state q_i and the cell that the scanning head scans contains the symbol S_j , then the scanning head moves to the cell to the left of the current cell or to the cell to the right of the current cell, respectively, and the machine enters the state q_l . Sometimes the following quadruple is also considered:

$$q_i S_j q_k q_l. \quad (2.4)$$

This quadruple is particularly useful if we want to construct a Turing machine that will compute *relatively computable functions*. These quadruples provide a Turing machine with a means of communicating with an external agency that can give correct answers to questions about a set $A \subset \mathbb{N}$. In particular, when a machine is in state q_i and the cell that the scanning head scans contains the symbol S_j , then the machine can be thought of asking the question, “Is $n \in A$?” Here n is the number of S_1 ’s that are printed on the tape. If the answer is “yes,” then the machine enters state q_k ; otherwise it enters state q_l . Turing machines equipped with such an external agency are called *oracle machines*, and the external agency is called an *oracle*.

Turing machines are used to compute the value of functions $f(n_1, \dots, n_m)$ that take values in $\mathbb{N}^m$. Each argument $n_i \in \mathbb{N}$ is represented on the tape by preprinting the symbol S_1 on $n_i + 1$ consecutive cells. Typically, such a block of cells is denoted by $\overline{n_i}$. Argument representations are separated by a blank cell (i.e., a cell on which the symbol $\sqcup$ is printed), while all other cells are empty (i.e., the symbol S_0 has been preprinted on each cell). It is customary to represent such a block of cell with the expression

$$\overline{(n_1, n_2, \dots, n_m)} = \overline{n_1} \sqcup \overline{n_2} \sqcup \dots \sqcup \overline{n_m}.$$

If α is an expression, then $\langle \alpha \rangle$ will denote the number of S_1 contained in α . In addition,

$$\langle \overline{m-1} \rangle = m \quad \text{and} \quad \langle \alpha\beta \rangle = \langle \alpha \rangle + \langle \beta \rangle.$$

It is also customary to use the symbol 1 for S_1 . Thus, the sequence 3, 4, 2 will be represented by the following three blocks of 1’s:

$$1111 \sqcup 11111 \sqcup 111.$$

The machine starts at state q_0 and the scanning head is placed atop the leftmost 1 of a sequence of n blocks of 1’s. If the machine has reached a situation in which none or more than one quadruple is applicable, the machine halts. Once the machine has terminated, the result of the computation is equal to the number of cells on which the symbol S_1 is printed.

Although the description presented so far is quite formal for my own taste, still fuzzy versions of the Turing machine are extensions of the “standard” formal definition that is given below.

Definition 2.1.2 A Turing machine $\mathcal{M}$ is an octuple $(Q, \Sigma, \Gamma, \delta, \sqcup, \triangleright, q_0, H)$, where

- Q is a finite set of states.
- Σ is the input alphabet.
- Γ is an alphabet, called *working alphabet*, where $\Sigma \subseteq \Gamma$.

The Turing machine's scanning head moves back and forth along the tape. The number that the scanning head displays is its current state, which changes as it proceeds.

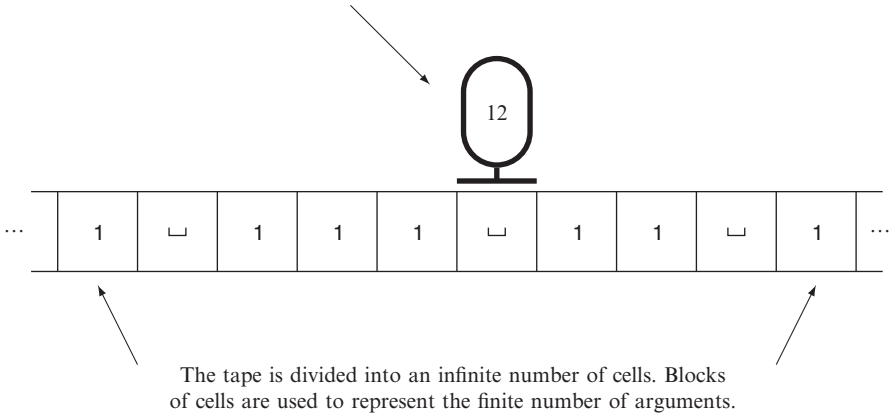


Figure 2.1: A typical Turing machine

- $\sqcup \in \Gamma$ is the blank symbol.
- $\triangleright \in \Gamma$ is the *left end symbol*.
- $q_0 \in Q$ is the initial state.
- $H \subseteq Q$ is the set of halting or accepting and rejecting states.
- δ , the *transition function*, is a function from $(Q \setminus H) \times \Gamma$ to $Q \times \Gamma \times \{L, R, N\}$ such that $\mathcal{M}$ may perform an instruction $(q', Y, D) \in Q \times \Gamma \times \{L, R, N\}$, if $\mathcal{M}$ is in state q , the scanning head has just read the symbol X , and $\delta(q, X) = (q', Y, D)$. Depending on the value of D , the machine will move to the left (L), to the right (R), or it will stay still when $D = N$ and the scanning head will overwrite X with Y . Also, $\delta(q, \triangleright) = (q, \triangleright, R)$, which means that whenever the scanning head has read the symbol $\triangleright$, it immediately moves to the right.

Note that here I defined a machine that has a unidirected tape and not a bidirectional tape. One can get the bidirectional version by eliminating all references to the $\triangleright$ symbol.

A *configuration* of $\mathcal{M}$ is an element from

$$C(\mathcal{M}) = \{\triangleright\}\Gamma^*Q\Gamma^+ \cup Q\{\triangleright\}\Gamma^*,$$

where AB denotes strings that are formed by concatenating a string that belongs to A with a string that belongs to B ; also, A^* is the set of all finite words over A ,¹ that is,

$$A^* = \{\varepsilon\} \cup A \cup (A \times A) \cup (A \times A \times A) \cup \dots,$$

1. Alternatively known as the *free monoid with base A*.

ε is the empty word, and $A^+ = A^* \setminus \{\varepsilon\}$. If $w_1 q a w_2$ is a configuration, then $w_1 \in \{\triangleright\} \Gamma^*$, $w_2 \in \Gamma^*$, $a \in \Gamma$, and $q \in Q$. Moreover, this configuration means that a machine $\mathcal{M}$ is in state q , the content of the tape is $\triangleright w_1 a w_2 \sqcup \sqcup \sqcup \dots$, and the scanning head sits atop the $(|w_1| + 1)$ -th cell, where $|s|$ is the length of string s . The *initial* configuration is $q_0 \triangleright x$, where x is the input fed to the machine. A configuration whose state component is in H is called a *halted* configuration.

A *step* is a relation $\vdash_{\mathcal{M}}$ on the set of configurations defined as follows:

- (i) $\dots x_{i-1} q x_i x_{i+1} \dots \vdash_{\mathcal{M}} \dots x_{i-1} q' y x_{i+1} \dots$, if $\delta(q, x_i) = (q', y, N)$;
- (ii) $\dots x_{i-1} q x_i x_{i+1} \dots \vdash_{\mathcal{M}} \dots x_{i-2} q' x_{i-1} y x_{i+1} \dots$, if $\delta(q, x_i) = (q', y, L)$;
- (iii) $\dots x_{i-1} q x_i x_{i+1} \dots \vdash_{\mathcal{M}} \dots x_{i-1} y q' x_{i+1} \dots$, if $\delta(q, x_i) = (q', y, R)$; and
- (iv) $\dots x_{n-1} q x_n \vdash_{\mathcal{M}} \dots x_{n-1} y q' \sqcup$, if $\delta(q, x_n) = (q', y, R)$.

A *computation* by $\mathcal{M}$ is a sequence of configurations $C_0, C_1, \dots, C_n$, for some $n \geq 0$ such that

$$C_0 \vdash_{\mathcal{M}} C_1 \vdash_{\mathcal{M}} C_2 \vdash_{\mathcal{M}} \dots \vdash_{\mathcal{M}} C_n.$$

A computation from C_0 to C_n can be written compactly as $C_0 \vdash_{\mathcal{M}}^* C_n$.

A computation by $\mathcal{M}$ on input x is a series of actions that start at configuration $C_0 = q_0 \triangleright x$ and is either *infinite* (i.e., nonterminating) or stops at configuration $w_1 q w_2$, where $q \in H$. Assume that $H = \{q_a, q_r\}$, where q_a is the *accepting* state and q_r is the *rejecting* state. Then a computation is called *accepting* if it finishes in the configuration $w_1 q_a w_2$ and *rejecting* if it finishes in the configuration $w_1 q_r w_2$. Furthermore, if a computation on input x is accepting or rejecting, we say that the corresponding machine *accepts* or *rejects* x , respectively. More generally, the words accepted by $\mathcal{M}$ form a formal language $L(\mathcal{M})$.

Typically, a *formal language*, or simply a language, over an *alphabet* Σ is subset of Σ^* , which is often defined by means of a formal *grammar*.

Definition 2.1.3 A grammar is defined to be a quadruple $G = (V_N, V_T, S, \Phi)$ where V_T and V_N are disjoint sets of terminal and nonterminal (syntactic class) symbols, respectively; S , a distinguished element of V_N , is called the starting symbol. Φ is a finite nonempty relation from $(V_T \cup V_N)^* V_N (V_T \cup V_N)^*$ to $(V_T \cup V_N)^*$. In general, an element (α, β) is written as $\alpha \rightarrow \beta$ and is called a production or rewriting rule [134].

The formal language *decided* by some Turing machine $\mathcal{M}$ is defined by

$$L(\mathcal{M}) = \left\{ w \mid (w \in \Sigma^*) \wedge (\mathcal{M} \text{ accepts } w) \right\}.$$

$\mathcal{M}$ *decides* a language L if for any word $w \in \Sigma^*$ either $w \in L$ and $\mathcal{M}$ *accepts* w or $w \notin L$ and $\mathcal{M}$ *rejects* w . A language is called *recursive* or *decidable* if there is a Turing machine that decides it. Also, $\mathcal{M}$ *semidecides* a language L when the machine halts for input w if and only if $w \in L$. A language L is *recursively enumerable* or *semidecidable* if there is a Turing machine that semidecides it.

$\mathcal{M}$ computes a function $F : \Sigma^* \rightarrow \Gamma^*$ if

$$\text{for all } x \in \Sigma^* : q_0 \triangleright x \vdash_{\mathcal{M}}^* q_a \triangleright F(x).$$

If $F : \Sigma^* \rightarrow \Gamma^*$ is computable by a Turing machine, it is called *recursive*.

Let $\mathcal{M}$ be a Turing machine and let

$$\Psi_{\mathcal{M}}^{(n)}(x_1, x_2, \dots, x_n)$$

be a partial function of n arguments. Then, $\mathcal{M}$ computes $\Psi_{\mathcal{M}}^{(n)}$ if for each n -tuple of arguments, $\mathcal{M}$ halts after a finite number of steps. If $\mathcal{M}$ does not terminate on a tuple $(k_1, \dots, k_n)$, then $\Psi_{\mathcal{M}}^{(n)}$ is undefined on this tuple. $\mathcal{M}$ computes f if for all $(x_1, \dots, x_n)$, $\Psi_{\mathcal{M}}^{(n)}(x_1, \dots, x_n)$ is defined and equal to $f(x_1, \dots, x_n)$. Now, it is possible to construct a Turing machine $\mathcal{M}'$ that will have as input the description of the controlling device of another Turing machine $\mathcal{M}$ and its arguments. Clearly, both the description of the controlling device and the arguments of the machine have to be encoded. In order to encode the various symbols, one may start by using an injective mapping (i.e., one that preserves distinctness) from the class of symbols into the integers. Thus, each symbol is identified with an integer “label.” More generally, sentences are mapped to a unique integer by employing similar methods. Mappings like this are called *codings* and the labels are called *code numbers*. The most common coding is the Gödel numbering, which was invented by Gödel, and the details of this coding are given below.

Suppose that we associate with each basic symbol of a Turing machine an odd number greater than or equal to 3 as follows:

$$\begin{array}{cccccccccccc} 3 & 5 & 7 & 9 & 11 & 13 & 15 & 17 & 19 & 21 & \dots \\ \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \\ R & L & S_0 & q_0 & S_1 & q_1 & S_2 & q_2 & S_3 & q_3 & \dots \end{array}$$

For each i , S_i is associated with $4i + 7$ and q_i is associated with $4i + 9$. In order to define the encoding of a Turing machine, first we need to define the encoding of an expression and then the encoding of a sequence of expressions.

Definition 2.1.4 Assume that M is a string of symbols $\gamma_1, \gamma_2, \dots, \gamma_n$ and that $a_1, a_2, \dots, a_n$ are the corresponding integers associated with these symbols. The Gödel number of M is the integer

$$\text{Gn}(M) = \prod_{k=1}^n \left(\text{Pr}(k) \right)^{a_k},$$

where $\text{Pr}(k)$ is the k th prime number in order of magnitude.

Example 2.1.1 If $M = q_1 S_0 S_2 q_1$, then $\text{Gn}(M) = 2^{13} \cdot 3^7 \cdot 5^{15} \cdot 7^{13}$, that is,

$$\text{Gn}(M) = 52,974,066,440,027,250,000,000,000,000.$$

Definition 2.1.5 Suppose that $M_1, M_2, \dots, M_n$ is a finite sequence of expressions. Then the Gödel number of this sequence is the integer:

$$\prod_{k=1}^n \left(\text{Pr}(k) \right)^{\text{Gn}(M_k)}.$$

Definition 2.1.6 Assume that $M_1, M_2, \dots, M_n$ is any arrangement of the quadruples of a Turing machine $\mathcal{M}$ without repetitions. Then the Gödel number of the sequence $M_1, M_2, \dots, M_n$ is a Gödel number of $\mathcal{M}$.

Clearly, a Turing machine consisting of n quadruples has $n!$ different Gödel numbers.

Definition 2.1.7 A universal Turing machine $\mathcal{U}$ is a Turing machine that can be employed to compute any function of one argument that an ordinary Turing machine $\mathcal{M}$ can compute.

Practically, this means that given a Turing machine $\mathcal{M}$ with a Gödel number m that computes the function $f(x)$, then

$$\Psi_{\mathcal{U}}^{(2)}(m, x) = f(x) = \Psi_{\mathcal{M}}^{(1)}(x).$$

Thus, if the number m is written on the tape of $\mathcal{U}$, followed by the number x , $\mathcal{U}$ will compute the number $\Psi_{\mathcal{M}}^{(1)}(x)$. Also, the universal Turing machine can be used to compute functions with n arguments, but I am not going to describe how this can be done (see [39] for the relevant details).

Function $\Psi_{\mathcal{U}}^{(2)}$ is just an example of a function that has as arguments a “program” and its “input.” Another interesting example of such a function is the so-called halting function:

$$h(m, x) = \begin{cases} 1, & \text{when } \mathcal{M} \text{ starts with input } x \text{ and eventually stops,} \\ 0, & \text{otherwise,} \end{cases}$$

where m is the Gödel number of $\mathcal{M}$. Whether this function is computable is equivalent to the halting problem. This, in turn, can be summarized as follows: is there an *effective* procedure such that given any m and any x we can determine whether $\Psi_{\mathcal{U}}^{(2)}(m, x)$ is defined or not?

It can be shown using the *diagonalization argument* that there is no such method. In particular, by using this principle one can easily prove that no Turing machine can determine whether $\Psi_{\mathcal{U}}^{(2)}(m, x)$ is defined. But what exactly is this argument?

The diagonalization argument was devised by Georg Ferdinand Ludwig Philip Cantor to show that the set of real numbers is not countably infinite. It is based on the diagonalization principle, a fundamental proof technique, which can be stated as follows:

Principle 2.1.1 (Diagonalization Principle) Assume that R is a binary relation on a set A . Also, assume that D , the diagonal set for R , is the set

$$\{a \mid (a \in A) \wedge ((a, a) \notin R)\}.$$

For each $a \in A$, suppose that $R_a = \{b \mid (a, b) \in R\}$. Then D is distinct from each R_a .

Paulo Cotogno [38] has presented a simplified proof of the insolvability of the halting problem: Assume that $\Psi_1^{(1)}, \Psi_2^{(1)}, \Psi_3^{(1)}, \dots$ is an enumeration of the computable (partial) functions. In addition, let us define the halting function

$$f(x, y) = \begin{cases} 1, & \text{if } \Psi_x^{(1)}(y) \text{ converges,} \\ 0, & \text{if } \Psi_x^{(1)}(y) \text{ does not converge,} \end{cases}$$

and the diagonal monadic function

$$g(x) = \begin{cases} 1, & \text{when } f(x, x) = 0, \\ \perp, & \text{when } f(x, x) = 1. \end{cases}$$

Here $\perp$ denotes an *undefined* or *divergent* value, depending on the context. Suppose that $g(x)$ is computable. Then there is an i such that $g(x) = \Psi_i^{(1)}(x)$. This implies that $\Psi_i^{(1)}(i) = 0$, that is, $g(i) = 0$. The last equation is equivalent to $f(i, i) = 0$ and this implies that $\Psi_i^{(1)}(i)$ is undefined, which is obviously a contradiction.

The interesting thing about this proof is that it holds for all definitions of computability. In other words, if one considers a class of machines, then it is impossible for them to compute their own halting functions. Indeed, most models of computation with *hypercomputational*² capabilities have been examined by Toby Ord and Tien Kieu [97], who have concluded that none of them is able to solve its own halting problem. However, this does not mean that some machine cannot solve the halting problem for Turing machines or some hypermachines—there is no logical inconsistency here.

2.2 Extensions of Turing Machines

In this section, I will briefly present three standard extensions of the Turing machine: nondeterministic, probabilistic, and multitape Turing machines. It has been argued that any probabilistic Turing machine can be transformed into a nondeterministic one by ignoring the probabilities. However, in the lights of the discussion in Sect. 1.2, where it was argued that nondeterminism is primitive notion, while randomness is not, this “equivalence” is meaningless.

2.2.1 Nondeterministic Turing Machines

Quite naturally, Turing [135, p. 232] was the first to describe an extension to his standard model of computation. He called these extended Turing machines *c-machines* and described them as follows:

For some purposes we might use machines (choice machines or c-machines) whose motion is only partially determined by the configuration (hence the use of word “possible” in §1). When such a machine reaches one of these ambiguous configurations, it cannot go on until some arbitrary choice has been made by an external operator. This would be the case if we were using machines to deal with axiomatic systems.

In a sense, one could argue that c-machines correspond to machines that function in a non-deterministic way; nevertheless, there is a precise formulation of the latter:

Definition 2.2.1 An octuple $(Q, \Sigma, \Gamma, \Delta, \sqsubset, \triangleright, q_0, H)$ is a *nondeterministic Turing machine* $\mathcal{N}$ if $\Delta : (Q \setminus H) \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R, N\})$ is the transition function and all others are as for the standard Turing machine.

2. In a nutshell, hypercomputation is the idea that there are conceptual and real computing machines that transcend the capabilities of the Turing machine. See [126] for an overview of the field of hypercomputation.

A configuration of $\mathcal{N}$ is an element from

$$C(\mathcal{N}) = \{\triangleright\}\Gamma^*Q\Gamma^* \cup Q\{\triangleright\}\Gamma^*.$$

The configuration $q_0 \triangleright w$ is the *initial configuration of $\mathcal{N}$ on w* , where w is the input fed to the machine. A configuration whose state component is in H is called a halted configuration.

A step of $\mathcal{N}$ is a binary relation $\vdash_{\mathcal{N}} \subseteq C(\mathcal{N}) \times C(\mathcal{N})$ that is defined as follows:

- (i) $x_1x_2\dots x_{i-1}qx_ix_{i+1}\dots x_n \vdash_{\mathcal{N}} x_1x_2\dots x_{i-1}q'yx_{i+1}\dots x_n$, if $(q', y, N) \in \Delta(q, x_i)$;
- (ii) $x_1x_2\dots x_{i-1}qx_ix_{i+1}\dots x_n \vdash_{\mathcal{N}} x_1x_2\dots x_{i-2}q'x_{i-1}yx_{i+1}\dots x_n$, if $(q', y, L) \in \Delta(q, x_i)$;
- (iii) $x_1x_2\dots x_{i-1}qx_ix_{i+1}\dots x_n \vdash_{\mathcal{N}} x_1x_2\dots x_{i-1}yq'x_{i+1}\dots x_n$, if $(q', y, R) \in \Delta(q, x_i)$; and
- (iv) $x_1x_2\dots x_{n-1}qx_n \vdash_{\mathcal{N}} x_1x_2\dots x_{n-1}yq'\sqcup$, if $(q', y, R) \in \Delta(q, x_i)$.

A computation of $\mathcal{N}$ is a sequence $C_0, C_1, \dots, C_n$ of configurations such that

$$C_i \vdash_{\mathcal{N}} C_{i+1},$$

for $i = 0, 1, 2, \dots, n$. A *computation of $\mathcal{N}$ on an input x* is any computation starting from the initial configuration and either halts, in which case it halts in a configuration w_1qw_2 , or is infinite. When a machine halts in an accepting/rejecting state, we say that $\mathcal{N}$ accepts/rejects the input word x . The language $L(\mathcal{N})$ accepted by the nondeterministic Turing machine $\mathcal{N}$ is

$$\begin{aligned} L(\mathcal{N}) &= \left\{ x \mid (x \in \Sigma^*) \wedge (q_0 \triangleright x \vdash_{\mathcal{N}} yq_az \text{ for some } y, z \in \Gamma^*) \right\} \\ &= \left\{ x \mid (x \in \Sigma^*) \wedge (\mathcal{N} \text{ accepts } x) \right\}. \end{aligned}$$

2.2.2 Probabilistic Turing Machines

Santos [112] introduced the *probabilistic Turing machine* as a probabilistic extension of ordinary Turing machines.

Definition 2.2.2 A probabilistic Turing machine P is a triple (S, Q, p) , where S is a set of symbols that the machine can print, Q is the set of internal states, $S \cap Q = \emptyset$, and $p : Q \times S \times V \times Q \rightarrow [0, 1]$, where $V = S \cup \{R, L, N\}$ and $R \notin S$, $L \notin S$, and $N \notin S$. Function p satisfies the following conditions:

- (i) $\sum_{v \in V} \sum_{q' \in Q} p(q, s, v, q') = 1$ for every $q \in Q, s \in S$; and
- (ii) for every $s \in S, p(q, s, N, q') = 0$ if $q \neq q'$.

Given that a machine is at state q and the symbol just scanned is s , the value of function p is the conditional probability of the “next act.”

Definition 2.2.3 Assume that $Z = (S, Q, p)$ is a probabilistic Turing machine. Then $\alpha \in (S \cup Q)^*$ is an *expression* of Z . α is an instantaneous expression of Z if and only if it contains exactly one $q \in Q$ and q is not the rightmost symbol. α is a tape expression if and only if it consists entirely of symbols from S . When α is an instantaneous expression of Z that contains $q \in Q$ and u is the symbol immediately to the right of q , then q is called the state of Z at α and u the symbol scanned by Z at α . The tape expression obtained by removing q from α is called the expression on the tape of Z at α .

Definition 2.2.4 Assume that $Z = (S, Q, p)$ is a probabilistic Turing machine. Then, for every instantaneous expression α and β of Z , define

$$q_Z(\alpha, \beta) = \begin{cases} p(q, s, s', q') & \text{if } \alpha = \gamma q s \delta, \quad \beta = \gamma q' s' \delta, \quad s' \in S, \\ p(q, s, R, q') & \text{if } \alpha = \gamma q s s' \delta, \quad \beta = \gamma s q' s' \delta, \quad s' \in S, \\ & \text{or } \alpha = \gamma q s, \quad \beta = \gamma s q' \sqcup, \\ p(q, s, L, q') & \text{if } \alpha = \gamma s' q s \delta, \quad \beta = \gamma q' s' s \delta, \quad s' \in S, \\ & \text{or } \alpha = q s \delta, \quad \beta = q' \sqcup s \delta, \\ 0 & \text{otherwise,} \end{cases}$$

where $q_Z(\alpha, \beta)$ is the probability that when the current instantaneous expression of Z is α , the next one will be β ; γ and δ are (possibly empty) tape expressions of Z .

One can extend $q_Z(\alpha, \beta)$ as follows:

$$\begin{aligned} q_Z^0(\alpha, \beta) &= 1 \quad \text{if } \alpha = \beta, \\ &= 0 \quad \text{if } \alpha \neq \beta, \\ q_Z^n(\alpha, \beta) &= \sum_{\gamma} q_Z^{n-1}(\alpha, \gamma) q_Z^0(\gamma, \beta), \end{aligned}$$

where the summation ranges over all instantaneous expressions γ . Also, $q_Z^n(\alpha, \beta)$ can be viewed as the probability that instantaneous expression of Z will be β *after n steps* provided that Z *starts* with instantaneous expression α . It can be proved that for every instantaneous expression α and nonnegative integer n it holds that

$$\sum_{\beta} q_Z^n(\alpha, \beta) \leq 1.$$

Definition 2.2.5 Suppose that $Z = (S, Q, p)$ is a probabilistic Turing machine. Then, for all instantaneous expressions α and β of Z and for all $n = 1, 2, \dots$ define

$$t_Z^{(n)}(\alpha, \beta) = p(q, s, N, q) q_Z^{(n-1)}(\alpha, \beta),$$

where q is the state of Z at β and s is the symbol scanned by Z when at β . Furthermore, define

$$t_Z(\alpha, \beta) = \sum_{n=1}^{\infty} t_Z^{(n)}(\alpha, \beta),$$

where $t_Z^{(n)}(\alpha, \beta)$ can be viewed as the probability that the machine starts with instantaneous expression α and terminates with instantaneous expression β in n steps. The interpretation of $t_Z^{(n)}(\alpha, \beta)$ is obvious.

Definition 2.2.6 A k -ary *random* function ϕ is a function from the collection of all $(k + 1)$ -tuples of nonnegative integers to $[0, 1]$ that satisfies the following inequality:

$$\sum_{m=0}^{\infty} \phi(m_1, m_2, \dots, m_k, m) \leq 1$$

for all k -tuples $(m_1, m_2, \dots, m_k)$.

Definition 2.2.7 Suppose that $Z = (S, Q, p)$ is a probabilistic Turing machine. Then, all positive integers k are associated with a k -ary random function $\Phi_Z^{(k)}$ as follows:

$$\Phi_Z^{(k)}(m_1, m_2, \dots, m_k, m) = \sum_{\langle \beta \rangle = m} t_Z(\alpha, \beta),$$

where

$$\alpha = q_1 \overline{(m_1, m_2, \dots, m_k)}, \quad q_1 \in Q$$

and the summation ranges over all instantaneous expressions β of Z such that $\langle \beta \rangle = m$.

Here q_1 plays the role of the initial state of a stochastic sequential-like machine. However, if this initial state is not given, but, instead, the initial distribution is given, then

$$\Phi_Z^{(k)}(m_1, m_2, \dots, m_k, m) = \sum_{\langle \beta \rangle = m} \sum_{i=1}^{|Q|} h(q_i) t_Z(\alpha_i, \beta),$$

where

$$\alpha_i = q_i \overline{(m_1, m_2, \dots, m_k)}, \quad \text{and} \quad q_i \in Q$$

and $|Q|$ is the cardinality of Q .

Definition 2.2.8 A k -ary random function ϕ is computable if and only if $\phi = \Phi_Z^{(k)}$ for some probabilistic Turing machine Z .

Santos [114] generalized his definition of a probabilistic Turing machine as follows:

Definition 2.2.9 A probabilistic Turing machine is specified by a quintuple (A, B, Q, p, h) where

- A is the printing alphabet;
- B is the auxiliary alphabet;
- Q is the set of internal states;
- $p : Q \times U \times V \times Q \rightarrow [0, 1]$, where $U = A \cup B$, $V = U \cup Q \cup \{R, L, N\}$, $R \notin U \cup Q$, $L \notin U \cup Q$, and $N \notin U \cup Q$, gives the probability of the next action; and
- $h : Q \rightarrow [0, 1]$ is the probability that the initial state is q .

Table 2.1: The definition of $q_{Z,T}(\alpha, \beta)$ (see Definition 2.2.11), where $\gamma, \delta \in (A \cup B)^*$, $q, q' \in Q$, and $u, u' \in A \cup B$

$$\begin{aligned}
 q_{Z,T}(\alpha, \beta) &= p(q, u, u', q') && \text{if } \alpha = \gamma qu \delta, \beta = \gamma q' u' \delta, u' \neq u; \\
 &= p(q, u, R, q') && \text{if } \alpha = \gamma qu u' \delta, \beta = \gamma u q' u' \delta, \gamma u \neq \sqcup, \\
 &&& \text{or } \alpha = qu u' \delta, \beta = q' u' \delta, u = \sqcup, \\
 &&& \text{or } \alpha = \gamma qu, \beta = \gamma u q' \sqcup, \gamma u \neq \sqcup, \\
 &&& \text{or } \alpha = qu, \beta = q' \sqcup, u = \sqcup; \\
 &= p(q, u, L, q') && \text{if } \alpha = \gamma u' qu \delta, \beta = \gamma q' u' u \delta, u \delta \neq \sqcup, \\
 &&& \text{or } \alpha = \gamma u' qu, \beta = \gamma q' u', u = \sqcup, \\
 &&& \text{or } \alpha = qu \delta, \beta = q' \sqcup u \delta, u \delta \neq \sqcup, \\
 &&& \text{or } \alpha = qu, \beta = q' \sqcup, u = \sqcup; \\
 &= p(q, u, u, q') + && \text{if } \alpha = \gamma qu \delta, \beta = \gamma q' u \delta; \\
 &\quad \sum_{q'' \in Q} p(q, u, q', q'') T(\langle \alpha \rangle) + && \\
 &\quad \sum_{q'' \in Q} p(q, u, q'', q') [1 - T(\langle \alpha \rangle)] && \\
 &= 0 && \text{otherwise}
 \end{aligned}$$

Functions p and h must satisfy the following conditions:

- (i) for all $q \in Q, u \in U, \sum_{v \in V} \sum_{q' \in Q} p(q, u, v, q') = 1$; and
- (ii) $\sum_{q \in Q} h(q) = 1$.

Function p is called the *transition* function and h is called the *initial distribution*. When for a particular $q_0 \in Q$ it holds that $h(q_0) = 1$ while $h(q) = 0$ for $q \neq q_0$, then q_0 is the initial state.

A probabilistic machine is *deterministic* if and only if the range of p and h is the set $\{0, 1\}$. Also, a machine is called *simple* if and only if $p(q, u, v, q') = 0$ for all $q, q' \in Q, u \in A \cup B$, and $v \in Q$.

Remark 2.2.1 An ordinary Turing machine is a deterministic probabilistic Turing machine while machines of the type of Definition 2.2.2 are simple machines.

Given a machine Z an expression of this machine is an element of $(A \cup B \cup Q)^*$. Also, a word is an element of A^* .

Definition 2.2.10 Assume that Z is a probabilistic Turing machine. Then an expression α of Z is an instantaneous description of Z if and only if

- (i) α contains exactly one $q \in Q$ and q is not the rightmost symbol of α ;
- (ii) the leftmost symbol of α is not $\sqcup$; and
- (iii) the rightmost symbol of α is not $\sqcup$ unless it is the symbol immediately to the right of q .

The collection of all instantaneous descriptions of some machine Z will be denoted by $\mathcal{J}(Z)$. Also, if α is an expression of some machine Z , then $\langle \alpha \rangle$ will denote the word of Z obtained by removing from α all symbols that do not belong to A provided that α contains symbols from A ; otherwise $\langle \alpha \rangle = \perp$.

Definition 2.2.11 Assume that $Z = (A, B, Q, p, h)$ is a probabilistic Turing machine and T a random set³ in A^* . For every $\alpha, \beta \in \mathcal{J}(Z)$, $q_{Z,T}(\alpha, \beta)$ is defined as in Table 2.1 in p. 23.

The expression $q_{Z,T}(\alpha, \beta)$ denotes the probability that the *next* instantaneous description of Z relative to T will be β provided that Z *starts* with the instantaneous description α .

Definition 2.2.12 For all $\alpha, \beta \in \mathcal{J}(Z)$ and $n = 0, 1, 2, \dots$ define $q_{Z,T}^{(i)}$ as follows:

$$\begin{aligned} q_{Z,T}^{(0)}(\alpha, \beta) &= 1 \text{ if } \alpha = \beta, \\ q_{Z,T}^{(0)}(\alpha, \beta) &= 0 \text{ if } \alpha \neq \beta, \\ q_{Z,T}^{(n)}(\alpha, \beta) &= \sum_{\gamma \in \mathcal{J}(Z)} q_{Z,T}^{(n-1)}(\alpha, \gamma) q_{Z,T}(\gamma, \beta). \end{aligned}$$

The expression $q_{Z,T}^{(n)}$ is the probability that the instantaneous description of Z relative to T will be β *after* n *steps* provided that Z *starts* with α .

Definition 2.2.13 For all $\alpha, \beta \in \mathcal{J}(Z)$ and $n = 1, 2, \dots$, define

$$t_{Z,T}^{(n)}(\alpha, \beta) = p(q, u, N, q) q_{Z,T}^{(n-1)}(\alpha, \beta),$$

where q is the state of Z at β and u the symbol scanned by Z at β . In addition, define

$$t_{Z,T}(\alpha, \beta) = \sum_{n=1}^{\infty} t_{Z,T}^{(n)}(\alpha, \beta).$$

The expression $t_{Z,T}^{(n)}(\alpha, \beta)$ denotes the probability that Z will *terminate* with β relative to T *after* n *steps* provided that Z *starts* with α . Also, $t_{Z,T}(\alpha, \beta)$ is the probability that Z will *terminate* with β relative to T *after a finite number of steps* given that Z *starts* with α .

The following definitions are needed for the exposition in Sect. 4.7. Assume that $\mathcal{A}$ is the set $\{a_1, a_2, \dots, a_m\}$, where $m > 0$ and $a_1 = 1$, and that T is a random set in $\mathcal{A}^*$.

Definition 2.2.14 Function $\mathcal{N} : \mathcal{A}^* \rightarrow \mathbb{N}$ is defined as follows:

$$\begin{aligned} \mathcal{N}(w) &= \sum_{i=1}^n k_i m^{i-1} & \text{if } w = a_{k_n} a_{k_{n-1}} \dots a_{k_1}, n > 0, \\ &= 0 & \text{if } w = \varepsilon. \end{aligned}$$

Conversely, function $\mathcal{W} : \mathbb{N} \rightarrow \mathcal{A}^*$ is defined as follows:

$$\begin{aligned} \mathcal{W}(n) &= w & \text{if and only if } \mathcal{N}(w) = n, \\ &= \varepsilon & \text{when } n < 0. \end{aligned}$$

3. A *random* set C in an ordinary set X is characterized by a function $C : X \rightarrow [0, 1]$, where $C(x)$ denotes the probability that $x \in X$.

Function $\mathcal{N}$ is called *codifier* and function $\mathcal{W}$ is called *decodifier*. These names are justified by the following lemmata:

Lemma 2.2.1 *There is a simple deterministic probabilistic Turing machine $Z = (A, B, Q, p, q_0)$ such that for every T and $w \in \mathcal{A}^*$*

$$\begin{aligned} t_{Z,T}(q_0 w, \beta) &= 1 && \text{if } \beta = q\mathcal{N}(w), \\ &= 0 && \text{otherwise.} \end{aligned}$$

Machine Z will be called the coding machine with final state q .

Lemma 2.2.2 *There is a simple deterministic probabilistic Turing machine $Z = (A, B, Q, p, q_0)$ such that for every T and nonnegative n*

$$\begin{aligned} t_{Z,T}(q_0 \bar{n}, \beta) &= 1 && \text{if } \beta = q\mathcal{W}(n), \\ &= 0 && \text{otherwise,} \end{aligned}$$

where q is a terminating state of Z .

Machine Z will be called the decoding machine with final state q .

2.2.3 Multitape Turing Machines

A multitape Turing machine is one that has several tapes, where each tape has its own scanning head and all of them are connected to the unique controlling device. At any given moment, the machine is in a unique state. The scanning heads conclude reading the symbols scanned in a single step. Depending on the current state and these symbols, the machine rewrites some cells or moves some scanning heads to the left or the right while it changes its state. Obviously, a multitape machine with just one tape is an ordinary Turing machine.

Definition 2.2.15 Assume that k is a positive integer. Then, a *k-tape Turing machine* is an octuple $(Q, \Sigma, \Gamma, \delta, \sqsubset, \triangleright, q_0, H)$, where with exception of δ , the transition function, all other components are as in the definition of the ordinary Turing machine (see Definition 2.1.2). The transition function is a function from $(Q \setminus H) \times \Gamma^k$ to $Q \times (\Gamma \times \{L, R, N\})^k$. That is, for each state q and each k -tuple of tape symbols $(a_1, \dots, a_k)$, $\delta(q, (a_1, \dots, a_k)) = (q', (b_1, \dots, b_k))$, where q' is the new state and each b_j is the “action” taken by the machine on tape j .

Computation takes place in all tapes of a multitape Turing machine. Therefore, a configuration of such a machine must include information about all tapes:

Definition 2.2.16 A *configuration* of a multitape Turing machine $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, \sqsubset, \triangleright, q_0, H)$ with k tapes is an element from

$$C(\mathcal{M}) = (\{\triangleright\}\Gamma^*)^k Q (\Gamma^+)^k \cup Q (\{\triangleright\}\Gamma^*)^k.$$

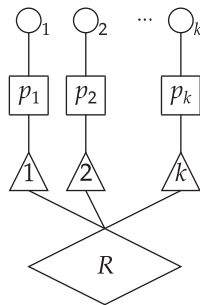
2.3 The “Most General” Approach to the Concept of an Algorithm

In 1953, Andrei Nikolaevich Kolmogorov [76] presented the sketch of a new definition of the notion of algorithm. Later on, a complete formulation was presented in a paper co-authored by Kolmogorov and Vladimir Andreevich Uspensky [78]. The basic idea behind this formulation is that an algorithm Γ is applied to a “condition” $A \in \mathfrak{A}(\Gamma)$, where $\mathfrak{A}(\Gamma) \subset \mathfrak{S}(\Gamma)$, $\mathfrak{A}(\Gamma)$ is the class of initial states, and $\mathfrak{S}(\Gamma)$ is the class of possible states, to get a “solution” B that is extracted from the terminal state $T \in \mathfrak{C}(\Gamma)$, where $\mathfrak{C}(\Gamma)$ is the class of terminal states. Function Ω_Γ takes a state and generates the next state. Thus, a computation is a series of state transformations: $A_1 = \Omega_\Gamma(A)$, $A_2 = \Omega_\Gamma(A_1)$, ..., $A_n = \Omega_\Gamma(A_{n-1})$. The process stops when either a terminal state has been reached or when a state cannot be handled by function Ω_Γ . However, it is quite possible that the process may never terminate. Now what is really interesting is that Grigoriev [61] found a way to show that Kolmogorov’s notion of algorithm is stronger than Turing machines. In different words, Grigoriev found an algorithm à la Kolmogorov–Uspensky that has hypercomputational properties. In what follows, I will briefly present algorithms à la Kolmogorov–Uspensky, in general, and Grigoriev’s algorithm, in particular.

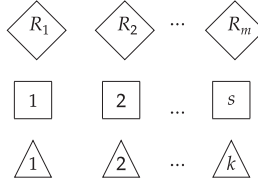
2.3.1 States

A state S is defined in terms of a finite number of elements $\bigcirc_1, \dots, \bigcirc_k$, where each $\bigcirc_i$ has one of the types $T_0, T_1, \dots, T_n$, and in terms of groups of relations having one of the types $R_1, \dots, R_m$. For each type of relation R_i only a fixed number k_i of related elements will be considered. Note that the number of elements, types, and relations are fixed for any algorithm and cannot be changed. In general, each relation R_j is not symmetric. Thus, the order in which elements are related plays an important role for states. Also, the index of the type of each element $\bigcirc_j$ is placed inside the circle as in $\textcircled{j}$. As for the indices adjoining the circles, they have nothing to do with the structure of the state.

The notion of a state depends on the presentation of a definite order of all relations in which each element occurs. Thus, the links between elements $\bigcirc_i$ and a relation of type R can be visualized by the following schema:



Here p_j indicates the “area” that a particular relation of type R occupies in the ordered sequence of relations in which the element $\bigcirc_j$ is involved. By introducing additional types of elements

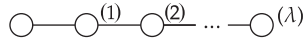


one can replace old algorithms with new ones that differ from the old ones in the way we express states. In particular, there will be only one type of relation between two elements. For example, such a relation between elements $\bigcirc_1$ and $\bigcirc_2$ will be represented as follows:



A direct consequence of this transformation is that all elements related to a particular element will be of *different* types. In fact, this consequence is a condition on all elements that are related to some other element.

One has to specify a method by which the “active part” of state is chosen. Here the term “active part” refers to the structure that uniquely determines the transformation that maps S to $S' = \Omega(S)$. More specifically, it is assumed that in the “active part” of a state there is a distinguished *initial* element. Since Kolmogorov wanted to reduce complexity, it was felt that all elements of a state should be linked to the initial element $\bigcirc$ by a chain



of length $\lambda \leq N$, where N is a constant number, which is associated with each algorithm. In addition, the operation $S' = \Omega(S)$ must rely solely on information about the structure of the “active part” of S . For example, assume that initial elements are of types T_0 and T_1 . Then, when a transformation of the initial element from type T_0 to one of type T_1 and this implies that the terminal state has been reached, then this is the simplest case of all possible transformations.

2.3.2 Basic Ideas from Combinatorial Topology

Kolmogorov’s concept of an algorithm uses notions from combinatorial topology. So, since these notions are not widely known, I will briefly introduce them. This exposition is based on [106]. In what follows, I assume a basic understanding of Euclidean spaces.⁴

A system of points $x_0, x_1, \dots, x_n$ of an n -dimensional linear space $\mathbb{R}^n$ is called *independent* if the system of vectors

$$(x_1 - x_0), \dots, (x_k - x_0)$$

is linearly independent. Assume that a and b are two distinct points of the Euclidean space $\mathbb{R}^n$. The set of all points $x \in \mathbb{R}^n$ of the form $x = \lambda a + \mu b$, where λ and μ are real numbers such that

$$\lambda + \mu = 1, \quad \lambda \geq 0, \quad \text{and} \quad \mu \geq 0,$$

will be called the *segment* $(a, b) = (b, a)$ with endpoints a and b . A set M of points of the Euclidean space $\mathbb{R}^n$ is called *convex* if $a \in M$ and $b \in M$ imply $(a, b) \in M$.

4. Readers not familiar with these notions can consult Springer’s “Encyclopedia of Mathematics” web page.

Definition 2.3.1 Suppose that $a_0, a_1, \dots, a_r$ is a system of independent points of the n -dimensional Euclidean space $\mathbb{R}^n$, $r \leq n$. The set A^r of the points x of the space $\mathbb{R}^n$ of the form

$$x = \lambda^0 a_0 + \lambda^1 a_1 + \dots + \lambda^r a_r,$$

where $\lambda^0, \lambda^1, \dots, \lambda^r$ are real numbers that satisfy the conditions

$$\lambda^0 + \lambda^1 + \dots + \lambda^r = 1 \quad \text{and} \quad \lambda^i \geq 0, \quad i = 0, 1, \dots, r,$$

is called an r -dimensional simplex, or just an r -simplex, and is written as $A^r = (a_0, a_1, \dots, a_r)$. The points $a_0, a_1, \dots, a_r$ are called *vertexes* and are contained in the simplex A^r .

Assume that $A^r = (a_0, a_1, \dots, a_r)$ is an r -simplex in $\mathbb{R}^n$ and that $a_k = a_{i_k}$, $k = 0, 1, \dots, s$, $0 \leq s \leq r$, a subset of the vertexes of A^r . The vertexes $a_0, a_1, \dots, a_r$ are independent which means that the vertexes $a_0, a_1, \dots, a_s$ are also independent and so $C^s = (a_0, a_1, \dots, a_s)$ is a simplex in $\mathbb{R}^n$. The simplex C^s is called an s -dimensional *face*, or just a *face*, of the simplex A^r . Two simplexes A and B of the Euclidean space $\mathbb{R}^n$ are *properly situated* either if they are nonintersecting or if their intersection $A \cap B$ is a common face of A and B . Two faces of a simplex are always properly situated.

Definition 2.3.2 A finite set K of simplexes of the Euclidean space $\mathbb{R}^n$ is called a *geometric complex*, or just a *complex*, if K satisfies the following conditions:

- (i) if A is a simplex of K , then every face of A is also in K ; and
- (ii) every two simplexes of K are properly situated.

A *subcomplex* of a complex K is any complex L all of those simplexes that are contained in K .

2.3.3 The Concept of an Algorithm

For an algorithm Γ there is an ordered set $\mathfrak{T}$ of classes, unlimited in size, of *elements* $T_0, T_1, \dots, T_n$. These elements are used to build the *states*. Also, they are mutually disjoint and their union is denoted by T . As was noted above, the elements of T will be denoted by circles $\bigcirc$. When an element belongs to some class T_j , we may also say that this is an element of type T_j .

In what follows the term complex on a set $\mathfrak{T}$ will refer to an ordinary one-dimensional complex of vertexes from T , that is, the union $K = K_0 \cup K_1$, where $K_0 = \{\bigcirc_i \mid \bigcirc_i \in T\}$ is a finite set of *vertexes* and $K_1 = \{\bigcirc_k - \bigcirc_l \mid \bigcirc_k, \bigcirc_l \in K_0\}$ contains certain pairs of elements of K_0 that are called *edges* of the complex K .

The set $\mathfrak{S}_K$ contains those complexes of K on $\mathfrak{T}$ that have the following properties:

- (i) given a fixed vertex, then all vertexes that are joined by edges to this vertex have different types of T_i ; and
- (ii) there exists only one vertex whose type is either T_0 or T_1 and which is called the *initial* vertex; all other vertexes are of types T_i , where $i \geq 2$.

The set $\mathfrak{A}_3 \subset \mathfrak{S}_3$ contains the initial vertexes of type T_0 and the set $\mathfrak{C}_3 \subset \mathfrak{S}_3$ contains the initial vertexes of type T_1 . The complexes of $\mathfrak{S}_3$ are considered the states of the algorithm Γ while the complexes of $\mathfrak{A}_3$ are considered the initial states of Γ and the complexes $\mathfrak{C}_3$ the terminal states of Γ :

$$\mathfrak{S}(\Gamma) = \mathfrak{S}_3, \quad \mathfrak{A}(\Gamma) = \mathfrak{A}_3, \quad \mathfrak{C}(\Gamma) = \mathfrak{C}_3.$$

The active part $U(S)$ of a state S is the subcomplex of the complex S that consists of the vertexes and edges that belong to chains of length $\lambda \leq N$ that contain the initial vertex. Here, N is an arbitrary number, fixed for a given algorithm and a chain is a complex of the form $\bigcirc_0 - \bigcirc_1 - \bigcirc_2 - \dots - \bigcirc_\lambda$. The length of the chain is equal to the number of edges occurring in it. A complex is called connected if any two of its vertexes can be joined by a chain.

The *outer part* $V(S)$ of a state S is the subcomplex that consists of the vertexes not connected with the initial vertex by chains of length $\lambda < N$ and of edges occurring in those chains that contain the initial vertex that have length $\lambda \leq N$. The intersection

$$L(S) = U(S) \cap V(S)$$

is the *boundary* of the active part of the state S . It follows that $L(S)$ is a zero-dimensional complex, that is, it has no edges, and consists of those vertexes that can be joined to the initial vertex by chains of length $\lambda = N$ but, at the same time, they cannot be joined to it by shorter chains.

Remark 2.3.1 The operation $\Omega_\Gamma(S) = S'$ must be such that transforming S into S' should require only the transformation into a new complex of the active part $U(S)$ and no change in the structure of the outer part $V(S)$. In addition, the type of the transformation of $U(S)$ must be determined only by the structure of the complex $U(S)$.

Remark 2.3.2 Two complexes K' and K'' are *isomorphic* if one can put them into one-to-one correspondence in the following way:

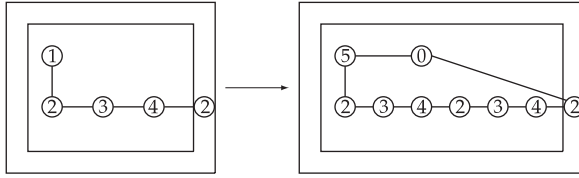
- (i) vertexes correspond to vertexes and edges to edges;
- (ii) an edge that joins the vertexes $\bigcirc'_i$ and $\bigcirc'_j$ corresponds to an edge that joins the vertexes $\bigcirc''_i$ and $\bigcirc''_j$; and
- (iii) corresponding vertexes have the same type.

It is a fact that the number of non-isomorphic active parts $U(S)$ possible in a given algorithm is bounded. This implies that the transformation rules are easy to formulate in finite form.

An algorithm is transformed according to a set of isomorphic pairs of states:

$$\begin{aligned} U_1 &\rightarrow W_1, \\ U_2 &\rightarrow W_2, \\ &\vdots \\ U_k &\rightarrow W_k. \end{aligned}$$

For each pair i there is a mapping φ_i that “sends” the subcomplex $L(U_i)$ to the subcomplex $L'(W_i)$. If for the vertexes $\bigcirc'_\mu \in L(U_i)$ and $\bigcirc_\nu \in L'(W_i)$ holds that $\varphi_i(\bigcirc'_\mu) \mapsto \bigcirc_\nu$, then an arbitrary non-initial vertex of the complex U_i connected with $\bigcirc_\nu$ has the type of one of the vertexes of the subcomplex $L(W_i)$ connected with $\bigcirc'_\mu$. Also, every complex U_i is a state of the class $\mathfrak{A}(\Gamma)$ of “conditions” satisfying the requirement $U(U_i) = U_i$. All $U_1, \dots, U_k$ are assumed to be non-isomorphic and W_i can be any state (i.e., either initial or terminal). Every pair of states is represented graphically in the following way. First, the complexes U_i are drawn on the left and the complexes W_i on the right. The boundary $L(U_i)$ is selected from the complex by framing it into two concentric rectangles. The same applies for the boundary $L'(W_i)$. The figure that follows depicts an isomorphic pair of states:



The isomorphism φ_i between $L(U_i)$ and $L'(W_i)$ is obtained by superimposing the right-hand part of the graphical representation to the left-hand part and by placing the coinciding vertexes between the rectangles.

The domain $\mathfrak{D}(\Gamma)$ of the operation $\Omega_\Gamma(S) = S'$ contains only those states S whose active part $U(S)$ is isomorphic to one of the complexes $U_1, \dots, U_k$.

Assume that the subcomplex $U(S)$ is isomorphic to U_i . Then, since $U(S)$ and U_i are connected complexes that belong to $\mathfrak{S}_3$, if they are isomorphic, there is only one isomorphic correspondence between them, which uniquely determines an isomorphism between $L(S)$ and $L(U_i)$. Furthermore, since there is an isomorphic mapping φ_i from the complexes $L(U_i)$ to the complexes $L'(W_i)$, this induces a uniquely determined isomorphism θ_i^S between $L(S)$ and $L'(W_i)$.

Assume that $S \in \mathfrak{D}(\Gamma)$, where the subcomplex $U(S)$ is isomorphic to the complex U_i . Then, one can construct a complex W' that is isomorphic to the complex W_i and which satisfies the following conditions:

- (i) $W' \cup V(S) = L(S)$; and
- (ii) the isomorphism θ_i^S between the subcomplexes $L(S) \subseteq W'$ and $L'(W_i) \subseteq W_i$ can be extended to an isomorphism between the complexes W' and W_i .

The application of Ω_Γ to the complex S is uniquely, up to isomorphism, defined to be a complex of the form

$$S' = W' \cup V(S).$$

Suppose that S is a terminal state. Then, the connected component⁵ of the initial vertex is considered to be the “solution.” The set of connected components of the initial vertexes of terminal states is denoted by $\mathfrak{B}(\Gamma)$.

5. A connected component of any vertex is the maximal connected subcomplex that contains this vertex.

2.3.4 What Can Be Computed with These Algorithms?

Grigoriev [61] had argued that Kolmogorov–Uspensky algorithms can recognize a predicate that no machine with polynomial-limited accessible memory and in particular no *multidimensional* Turing machine can recognize.⁶ Here memory is a storage structure that is defined in [36] as follows:

Definition 2.3.3 A storage structure $A = \langle L, \varphi_1, \dots, \varphi_p \rangle$ of rank p consists of a countable set L (the locations or cells of Λ) together with the maps $\varphi_1, \dots, \varphi_p$ of L into L , called *shift transformations*.

For example, a multitape Turing machine has a storage structure of rank 2 and the two shift transformations are the left and the right shifts. Also, a polynomial-limited accessible memory is a storage structure defined as follows:

Definition 2.3.4 A storage structure or memory $A = \langle L, \varphi_1, \dots, \varphi_p \rangle$ has polynomial-limited accessibility provided there are constants K and n such that the number of locations accessible in t steps from any given location does not exceed Kt^n . Here a location y is accessible from a location x in t steps provided there is a sequence $\psi_1, \psi_2, \dots, \psi_t$ of shift transformations such that $y = \psi_1\psi_2\cdots\psi_t(x)$.

In order to be able to fully grasp the ideas below, it is necessary to introduce some material from graph theory.

Preliminary Material from Graph Theory

A binary tree is called *directed* if there are either two edges or no edges coming from each node, and except for the root node, there is exactly one edge entering each node. Also, suppose that each edge of a binary tree is labeled by 0 or 1 so that two edges leaving the same node are labeled differently. A tree whose nodes, except the root, are labeled with 0 and 1 is called a labeled tree. For reasons of brevity, I will call *binary words* those words whose “letters” belong to the set $\{0, 1\}$.

Given a directed branch, there is a binary word that is formed as follows: starting from the edge that leaves the root node, the word consists of the labels of each edge. Clearly, the length of a binary word is equal to the number of edges that make up the branch. To each branch corresponds a unique binary word as far it regards the other binary words produced by branches of the same tree. Thus, it is possible to construct an inverse mapping on some set of binary words taking a binary word A to a branch α_A of the labeled tree starting at the root.

Assume that Γ is a labeled tree and that A is a binary word. Then, $D_\Gamma(A)$ is the binary word whose k -th letter is the label of the node which is the end of the k -th edge of the branch α_A . Note that $|D_\Gamma(A)| = |A|$.

A binary tree is called *regular* if all of its directed branches starting from the root and finishing at a leaf node (i.e., a node with no children) have the same length, which is called

6. A multidimensional Turing machine is one in which the tape can be either two or three dimensional. In the case of a two-dimensional machine, the transition function is of the form $\delta : (Q \setminus H) \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, D, U, N\}$, where D and U specify movement of the scanning head up and down, respectively.

the depth of the tree. A regular tree Γ of depth $2n$ will be called semiuniform when for any binary words A_1, A_2, B of length n such that $A_1 \neq A_2$, the right half of the words $D_\Gamma(A_1B)$ and $D_\Gamma(A_2B)$ are distinct. Also, a regular tree Γ of depth 2^L will be called uniform if for all $n, k \in \mathbb{N}$ such that $(n+1)2^{k+1} \leq 2^L$ any regular subtree of Γ of depth 2^{k+1} whose root is located at depth $n2^{k+1}$ is semiuniform.

Lemma 2.3.1 *There is an infinite sequence of uniform trees $\Gamma_0, \Gamma_1, \dots$ of depth $2^0, 2^1, \dots$, respectively, such that for $k \leq n$, Γ_k is an initial subtree of depth 2^k of Γ_n .*

Locally Complex Functions

A function g is *almost complex* if there are numbers ℓ_0 and θ ($0 \leq \theta < 1$) such that for any $n, k \in \mathbb{N}$ and for any words A and C in the alphabet Q with $|A| = n2^{k+1}$ and $\ell_0 \leq |C| = 2^k$, there is no equivalence class (see Sect. B.1) of the relation $\equiv_{A,C}$ that contains more than $\beta^{\theta 2^k}$ members, where β is the number of elements of Q . (The relation $\equiv_{A,C}$ is defined as follows: $X_1 \equiv_{A,C} X_2$ if $|X_1| = |X_2| = |C|$ such that $g(AX_1c_1 \dots c_i) = g(AX_2c_1 \dots c_i)$, $i = 1, 2, \dots, \ell$.)

A function f is *locally complex* if there is an almost complex function g and two sequences $(\alpha_n)_{n \in \mathbb{N}}$ and $(\beta_n)_{n \in \mathbb{N}}$ such that $\limsup_{k \rightarrow \infty} \beta_k = \infty$, where

$$\limsup_{n \rightarrow \infty} = \lim_{n \rightarrow \infty} \left(\sup_{m \geq n} \beta_m \right) = \inf \left\{ \sup \{ \beta_m : m \geq n \} : n \geq 0 \right\},$$

and for all words A and B for which

$$|A| = \sum_{i=1}^k \alpha_i + \sum_{j=1}^{k-1} \beta_j \quad \text{and} \quad |B| \leq \beta_k$$

the equality $f(AB) = g(B)$ holds.

Lemma 2.3.2 *No locally complex function f is recognizable in real time by a system with polynomial-limited accessible memory.*

The Non-recognizable Predicate

The predicate that is not recognizable by a machine with polynomial-limited accessible memory, whereas it is computable by some Kolmogorov–Uspensky algorithm in real time, involves a locally complex function F that takes as arguments binary words and returns either the digit one or the digit zero.

Suppose that a fixed Kolmogorov–Uspensky algorithm K' constructs the tree Γ_L in T_L units of time, where the sequence $(T_L)_{L \in \mathbb{N}}$ is monotone increasing in L . Let us now define the values of function F . Given a binary word A , $F(A) = 1$ if

$$\sum_{i=0}^L T_i + 2^{L+1} \leq |A| < \sum_{i=0}^{L+1} T_i + 2^{L+1}$$

for some L ; otherwise, if

$$\sum_{i=0}^{L-1} T_i + 2^{L-1} \leq |A| < \sum_{i=0}^{L-1} T_i + 2^L$$

for some L , then $F(A)$ is the last letter in the word $D_{\Gamma(L)}(B)$ where

$$A = CB \quad \text{and} \quad |C| = \sum_{i=0}^{L-1} T_i + 2^{L-1} - 1$$

for some binary word C . The predicate that is not recognizable by a machine with polynomial-limited accessible memory, and, thus, by a Turing machine, is $P(A) \Leftrightarrow F(A) = 0$, where A is a binary word.

It is not difficult to devise a Kolmogorov–Uspensky algorithm K that recognizes P in real time. For each L , starting at time $\sum_{i=0}^{L-1} T_i + 2^i$ and finishing at time $\sum_{i=0}^L T_i + 2^i - 1$, algorithm K with the help of K' builds the tree Γ_L . Next, if the input is a word B of length 2^L , the algorithm K does not build the tree Γ_L along the branch α_B and yields the word $D_{\Gamma_L}(B)$. In the time interval from $\sum_{i=0}^{L-1} T_i + 2^L$ to $\sum_{i=0}^L T_i + 2^L - 1$ the output of K is 1.

Lemma 2.3.3 *The function F is locally complex.*

Theorem 2.3.1 *There is a predicate not recognizable in real time by any system with polynomial-limited accessible memory but recognizable in real time by some Kolmogorov–Uspensky algorithm.*

This result can be proved using Lemmas 2.3.2 and 2.3.3 and algorithm K .

2.4 General Recursive Functions

In Sect. 2.1 it was explained how Turing machines compute functions and what makes a function computable by a Turing machine. Despite of this, a question that may naturally pop into one's mind is: Is it possible to tell whether a function is computable by a Turing machine without actually constructing the machine? There is class of numerical functions, that is, functions from nonnegative integers to nonnegative integers, that have exactly this property. These functions are called *recursive* functions. In fact, there are two “subclasses”—primitive and general recursive functions. In what follows, I will present these classes of functions. The exposition of the theory presented in this section is based on a seminal paper by Stephen Cole Kleene [73]. I will start with primitive recursive functions.

Primitive recursive functions are defined in terms of basic functions and function builders. There are three basic or initial functions:

- (i) the *successor* function $S(x) = x + 1$;
- (ii) the *zero* function $z(x) = 0$; and
- (iii) the *projection* functions $U_i^n(x_1, \dots, x_n) = x_i, 1 \leq i \leq n$.

Primitive recursive functions can be defined by applying function builders, or schemas, to the basic functions. There are three function builders:

Composition Suppose that f is a function of m arguments and each of $g_1, \dots, g_m$ is a function of n arguments. Then the function obtained by composition of f and $g_1, \dots, g_m$ is the function h defined as follows:

$$h(x_1, \dots, x_n) = f(g_1(x_1, \dots, x_n), \dots, g_m(x_1, \dots, x_n)).$$

Primitive Recursion A function h of $k + 1$ arguments is said to be definable by (primitive) recursion from the functions f and g , having k and $k + 2$ arguments, respectively, if it is defined as follows:

$$\begin{aligned} h(x_1, \dots, x_k, 0) &= f(x_1, \dots, x_k), \\ h(x_1, \dots, x_k, S(m)) &= g(x_1, \dots, x_k, m, h(x_1, \dots, x_k, m)). \end{aligned}$$

Minimalization The operation of minimalization associates with each total function f of $k + 1$ arguments a function h of k arguments. Given a tuple $(x_1, \dots, x_k)$, the value of $h(x_1, \dots, x_k)$ is the least value of x_{k+1} , if one exists, for which $f(x_1, \dots, x_k, x_{k+1}) = 0$. If no such x_{k+1} exists, then its value is undefined.

Now we are ready to define primitive recursive and general recursive functions.

Definition 2.4.1 The functions that can be obtained from the basic functions by the function builders composition and primitive recursion are called primitive recursive functions.

Definition 2.4.2 The functions that can be obtained from the basic functions by all function builders are called general recursive functions.

Note that general recursive functions are also known as just recursive functions or μ -recursive functions.

We can easily extend the two previous definitions to define A -primitive recursive and A -recursive functions. However, in order to do this, we need to know what the characteristic function of a set is.

Definition 2.4.3 Assume that X is a universe set and $A \subseteq X$. Then the *characteristic function* $\chi_A : X \rightarrow \{0, 1\}$ of A is defined as follows:

$$\chi_A(a) = \begin{cases} 1, & \text{if } a \in A, \\ 0, & \text{if } a \notin A. \end{cases}$$

Note that this particular way of defining a set is actually employed to define fuzzy subsets, multisets, etc., via different types of characteristic functions, but I will say more on this in the next chapter. We are now prepared to define A -primitive recursive and A -recursive functions. Assume that $A \subseteq \mathbb{N}$ is a fixed set.

Definition 2.4.4 A function f is a partial A -recursive function if $f = \Psi_{\mathcal{M}}^A$, where $\Psi_{\mathcal{M}}^A$ is a partial function that denotes the computation performed by an oracle Turing machine $\mathcal{M}$ with oracle A .

Definition 2.4.5 A function f is an A -recursive function if there is an oracle machine $\mathcal{M}$ with oracle A such that $f = \Psi_{\mathcal{M}}^A$ and $\Psi_{\mathcal{M}}^A$ is a total function.

Definition 2.4.6 A set B is recursive in A if χ_B is A -recursive.

Kleen's s-m-n theorem, also called the iteration or the parametrization theorem, is a basic result in recursion theory. In what follows, I will present this important result without a proof.

Any Turing machine is characterized by a set of quadruples, where any two distinct quadruples must differ in their first or second part. This is known as the *consistency restriction*. Obviously, each quadruple is just a string of specific symbols. Furthermore, one can construct strings of these symbols only that, nevertheless, are not quadruples. Also, an algorithmic test would exist for determining, given such a string, whether it represents a quadruple or not. Hence, it is possible to list all sets of quadruples that satisfy the consistency restriction. In particular, such a procedure can arrange the list of sets of quadruples in such a way that each set of quadruples is associated with an integer x . More specifically, each integer x will be associated with the $(x + 1)$ st element of the list. Each element of this list will be denoted by P_x . In addition, $\varphi_x^{(k)}$ is the partial function of k variables determined by P_x , where x is called an index of this function. Usually, the subscript is omitted when its value is clear from the context or when $k = 1$. Now, it is possible to state the s-m-n theorem:

Theorem 2.4.1 *For every $n, m \geq 1$, there exists a recursive function s_n^m of $m + 1$ variables such that for all $x, y_1, \dots, y_m$*

$$\lambda z_1 \dots \lambda z_n \cdot \varphi_x^{(m+n)}(y_1, \dots, y_m, z_1, \dots, z_n) = \varphi_{s_n^m(x, y_1, \dots, y_m)}^{(n)}.$$

Given a partial recursive function φ_x , the symbol W_x will denote its domain.

2.5 Recursive Sets, Relations, and Predicates

It is quite natural to extend the notion of recursiveness to characterize not only functions but also sets, relations, and predicates. Informally, a set is called recursive if we have an effective method to determine whether a given element belongs to the set. However, if this effective method cannot be used to determine whether a given element does *not* belong to the set, then the set is called semirecursive. Formally, a recursive set is defined as follows.

Definition 2.5.1 Let $A \subseteq \mathbb{N}$ be a set. Then we say that A is primitive recursive or recursive if its characteristic function χ_A is primitive recursive or recursive, respectively.

Example 2.5.1 Suppose that Π is the set of all odd natural numbers. Then Π is primitive recursive, since its characteristic function

$$\chi_{\Pi}(a) = R(a, 2)$$

is primitive recursive. Here, $R(x, y)$ returns the remainder of the integer division $x \div y$.

Definition 2.5.2 A set A is called recursively enumerable or semirecursive either if $A = \emptyset$ or if A is the range of a recursive function.

The following result gives another characterization of recursively enumerable sets:

Theorem 2.5.1 *A set A is recursively enumerable if and only if A is the domain of a partial recursive function.*

A -recursively enumerable sets are defined as follows:

Definition 2.5.3 A set B is called A -recursively enumerable either if $B = \emptyset$ or if B is the range of an A -recursive function.

Definitions 2.5.1 and 2.5.2 can be easily extended to characterize *relations*. Note that an n -ary relation on a set A is any subset R of the n -fold Cartesian product $A \times \cdots \times A$ of n factors.

Definition 2.5.4 A relation $R \subseteq \mathbb{N}^m$ is called primitive recursive or recursive if its characteristic function χ_R given by

$$\chi_R(x_1, \dots, x_m) = \begin{cases} 1, & \text{if } (x_1, \dots, x_m) \in R, \\ 0, & \text{if } (x_1, \dots, x_m) \notin R \end{cases}$$

is primitive recursive or recursive, respectively.

Definition 2.5.5 A relation $R \subseteq \mathbb{N}^m$ is called recursively enumerable (or semirecursive) if R is the range of a partial recursive function $f : \mathbb{N} \rightarrow \mathbb{N}^m$.

Let us now see how the notion of recursiveness has been extended to characterize predicates. But first let us recall what a predicate is. Roughly, it is a statement that asserts a proposition that must be either true (denoted by $\#$) or false (denoted by ff). An $\mathbb{N}^n$ function whose range of values consists exclusively of elements of the set $\{\#, ff\}$ is a predicate.

Definition 2.5.6 A predicate $P(x_1, \dots, x_n)$ is called recursive if the set

$$\{(x_1, \dots, x_n) \mid P(x_1, \dots, x_n)\},$$

which is called its *extension*, is recursive.

Definition 2.5.7 The predicate $P(x_1, \dots, x_n)$ is called recursively enumerable (or semirecursive) if there exists a partially recursive function whose domain is the set

$$\{(x_1, \dots, x_n) \mid P(x_1, \dots, x_n)\}.$$

The Arithmetic Hierarchy Let us denote by Σ_0^0 the class of all recursive subsets of $\mathbb{N}$. For every $n \in \mathbb{N}$, Σ_{n+1}^0 is the class of sets that are A -recursively enumerable for some set $A \in \Sigma_n^0$. It follows that Σ_1^0 is the class of recursively enumerable sets. Let us denote by Π_0^0 the class of all subsets of $\mathbb{N}$ whose complements are in Σ_0^0 . In other words, $D \in \Pi_0^0$ if and only if $\mathbb{N} \setminus D \in \Sigma_0^0$. The class Π_1^0 is known in the literature as the class of *co-recursively enumerable* sets. Any recursively enumerable set that is also co-recursively enumerable is a *decidable* set. In fact, the term decidable set is a synonym of the term recursive set.

Let us denote by Δ_n^0 the intersection of the classes Σ_n^0 and Π_n^0 (i.e., $\Delta_n^0 = \Sigma_n^0 \cap \Pi_n^0$). The classes Σ_n^0 , Π_n^0 , and Δ_n^0 form a hierarchy that is called the *arithmetic hierarchy*. The classes that make up this hierarchy have the following properties:

$$\begin{aligned} \Delta_n^0 &\subset \Sigma_n^0, & \Delta_n^0 &\subset \Pi_n^0, \\ \Sigma_n^0 &\subset \Sigma_{n+1}^0, & \Pi_n^0 &\subset \Pi_{n+1}^0, \\ \Sigma_n^0 \cup \Pi_n^0 &\subset \Delta_{n+1}^0, \forall n \geq 1. \end{aligned}$$

Figure 2.2 depicts the relationships between the various classes of the arithmetic hierarchy.

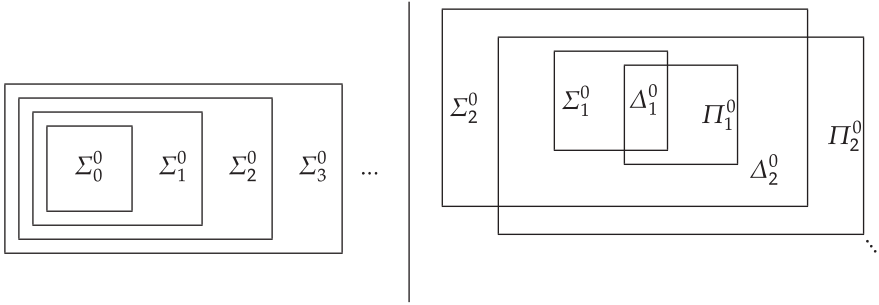


Figure 2.2: The relationships between the various classes of the arithmetic hierarchy

The set-theoretic presentation of the arithmetic hierarchy is not the only possible presentation. Indeed, other presentations based on predicates or relations are also possible. Assume that φ is a formula⁷ in the language of first-order arithmetic (i.e., there are no other nonlogical symbols apart from constants denoting natural numbers, primitive recursive functions, and predicates that can be decided primitive recursively). Then, ϕ is a Δ_0^0 -formula if ϕ contains at most bounded quantifiers. φ is Σ_1^0 if there is a Δ_0^0 -formula $\psi(x)$ such that $\varphi \equiv (\exists x)\psi(x)$. Dually, φ is Π_1^0 if $\neg\varphi$ is Σ_1^0 . More generally, a formula φ is in Σ_{n+1}^0 if there is a formula $\psi(x)$ in Π_n^0 such that $\varphi \equiv (\exists x)\psi(x)$. Dually, φ is Π_{n+1}^0 if $\neg\varphi$ is Σ_{n+1}^0 . Note that Σ_0^0 is the class of all recursive predicates. Suppose that $Q_1 = P_1(x_1)$, $Q_2 = P_2(x_1, x_2)$, $Q_3 = P_3(x_1, x_2, x_3)$, ... are Δ_0^0 -formulas. Then Table 2.2 gives a schematic representation of the classes Σ_n^0 and Π_n^0 .

7. Very roughly, a term yields a value; variables and constants are terms; functions are terms; atoms yield truth values; and each predicate is an atom. An atom is a formula. Given formulas p and q the following are also formulas: $\neg p$, $p \vee q$, $p \wedge q$, $p \Rightarrow q$, $p \equiv q$, $\forall x p$, and $\exists x p$. In the last two cases x is said to be a bound variable, while in all other possible cases it is said to be a free variable.

Table 2.2: A schematic representation of the classes Σ_n^0 and Π_n^0

	$n = 1$	$n = 2$	$n = 3$	...
Σ_n^0	$(\exists x_1)Q_1$	$(\exists x_1)(\forall x_2)Q_2$	$(\exists x_1)(\forall x_2)(\exists x_3)Q_3$	...
Π_n^0	$(\forall x_1)Q_1$	$(\forall x_1)(\exists x_2)Q_2$	$(\forall x_1)(\exists x_2)(\forall x_3)Q_3$	...

Assume that $R \subseteq \mathbb{N}^m$ is a relation. Then $R \in \Sigma_1^0$ (i.e., R is Σ_1^0 -relation) if R is recursively enumerable. Similarly, $R \in \Pi_1^0$ if $\bar{R} \in \Sigma_1^0$ (i.e., if the complement of R with respect to $\mathbb{N}^m$ is a Σ_1^0 -relation). In general, $R \in \Sigma_n^0$ ($n \geq 2$) if there are a $k \in \mathbb{N}$ and a Π_{n-1}^0 -relation $S \subset \mathbb{N}^{m+k}$ such that

$$R = \{(x_1, \dots, x_m) \mid \exists (x_{m+1}, \dots, x_{m+k}) \in \mathbb{N}^k, (x_1, \dots, x_{m+k}) \in S\}.$$

Also, $R \in \Pi_n^0$ if $\bar{R} \in \Sigma_n^0$.

Definition 2.5.8 A is Σ_n^0 -complete if $A \in \Sigma_n^0$ and for all $B \in \Sigma_n^0$, $B \leq_1 A$, where $B \leq_1 A$ is pronounced B is *one-one reducible* to A and means that there is a one-one recursive function f such that for all $x \in B \Leftrightarrow f(x) \in A$.

Similarly, one can define Π_n^0 -completeness.

Definition 2.5.9 A is *many-one reducible*, written as $A \leq_m B$, if there is a recursive function f such that for all x , $x \in A \Leftrightarrow f(x) \in B$.

Typically, the term “m-reducibility” is used instead of the term “many-one reducibility.”

The Analytical Hierarchy The second-order equivalent of the arithmetic hierarchy is called the *analytical hierarchy*. In this hierarchy, quantifiers range over function and relation symbols and over subsets of the universe. In other words, we are talking about second-order logic. A formula φ is a Π_1^1 -formula if $\varphi \equiv (\forall X)\psi(X)$ and $\psi(X)$ is Σ_1^0 . Dually, a formula φ is Σ_1^1 if and only if $\neg\varphi$ is Π_1^1 . More generally, a formula φ is Π_{k+1}^1 if and only if $\varphi \equiv (\forall X)\psi(X)$ and $\psi(X)$ is Σ_k^1 . Dually, φ is Σ_{k+1}^1 if and only if $\neg\varphi$ is Π_{k+1}^1 . Clearly, it is easy to construct a table like Table 2.2 to provide a schematic representation of the analytical hierarchy. Note that the Δ_1^1 -sets are the so-called *hyperarithmetic* sets. In addition, a function $f : \mathbb{N} \rightarrow \mathbb{N}$ is hyperarithmetic if its graph⁸ G_f is a hyperarithmetic relation.

The arithmetic and analytic hierarchies are used to classify functions, sets, predicates, and relations. In particular, the higher the class an object belongs to, the more classically noncomputable it is.

2.6 The Church–Turing Thesis

In Sect. 2.4 I asked whether it is possible to tell if a function is computable by a Turing machine without actually constructing the machine. More generally, one can ask what functions

8. The graph of a function $f : X \rightarrow Y$ is the subset of $X \times Y$ given by $\{(x, f(x)) \mid x \in X\}$. A total function whose graph is recursively enumerable is a recursive function.

and/or numbers are computable by Turing machines? The cornerstone of classical computability theory and a direct answer to this question is the Church–Turing thesis, which can be phrased as follows.

Thesis 2.6.1 *Every effectively computable function is Turing computable, that is, there is a Turing machine that realizes it. Alternatively, the effectively computable functions can be identified with the recursive functions.*

Formally, a function $f : \mathbb{N}^n \rightarrow \mathbb{N}^m$ is Turing computable if there is a Turing machine $\mathcal{M}$ that computes it. But it is not clear at all what is meant by an *effective* procedure or method. Jack Copeland [37] gives four criteria that any sequence of instructions that make up a procedure or method should satisfy in order for it to be characterized as effective:

- (i) each instruction is expressed by means of finite number of symbols;
- (ii) the instructions produce the desired result in a finite number of steps;
- (iii) they can be carried out by a human being unaided by any machinery save paper and pencil; and
- (iv) they demand no insight or ingenuity on the part of the human carrying it out.

In his classical textbook [93], Marvin Minsky defines an effective procedure as “a set of rules which tell us, from moment to moment, precisely how to behave,” provided we have at our disposal a universally accepted way to interpret these rules. Minsky concludes that this definition is meaningful if the steps are actually steps performed by some Turing machine. Another formulation of effectiveness is given in [66]:

[E]very instruction must be sufficiently basic that it can in principle be carried out by a person using only pencil and paper. It is not enough that each operation be clear and unambiguous, but it must also be feasible.

Gábor Etesi and István Németi [48] describe as effectively computable any function $f : \mathbb{N}^k \rightarrow \mathbb{N}^m$ for which there is a *physical computer* realizing it. Here, by “realization by a physical computer” they mean the following:

Let P be a physical computer, and $f : \mathbb{N}^k \rightarrow \mathbb{N}^m$ a (mathematical) function. Then we say that P *realizes* f if an imaginary observer O can do the following with P . Assume that O can “start” the computer P with $(x_1, \dots, x_k)$ as an input, and then sometime later (according to O ’s internal clock) O “receives” data $(y_1, \dots, y_m) \in \mathbb{N}^m$ from P as an output such that $(y_1, \dots, y_m)$ coincides with the value $f(x_1, \dots, x_k)$ of the function f at input $(x_1, \dots, x_k)$.

The same authors, after introducing the notion of *artificial computing systems*, that is, thought experiments relative to a fixed physical theory that involve computing devices, managed to rephrase the Church–Turing thesis as follows.⁹

9. Actually, they call this “updated” version of the Church–Turing thesis the *Church–Kalmár–Turing* thesis, named after Church, László Kalmár, and Turing.

Thesis 2.6.2 *Every function realizable by an artificial computing system is Turing computable.*

Since artificial computing systems are thought experiments relative to a fixed physical theory, the thesis can be rephrased as follows.

Thesis 2.6.3 *Every function realizable by a thought experiment is Turing computable.*

Note that according to Etesi and Némethi, a thought experiment relative to a fixed physical theory is a theoretically possible experiment, that is, an experiment that can be carefully designed, specified, etc., according to the rules of the physical theory, but for which we might not currently have the necessary resources.

Others, like David Deutsch [42], have reformulated the Church–Turing thesis as follows.

Thesis 2.6.4 *Every finitely realizable physical system can be perfectly simulated by a universal model computing machine operating by finite means.*

In the special case of the human mind, this thesis can be rephrased as follows.

Thesis 2.6.5 *The human brain realizes only Turing-computable functions.*

This thesis is the core of computationalism. This philosophy claims that a person’s mind is actually a Turing machine. Consequently, one may go a step ahead and argue that since a person’s mind is a Turing machine, then it will be possible one day to construct an artificial person with feelings and emotions. The mind is indeed a machine, but one that transcends the capabilities of the Turing machine and operates in a profoundly different way (see [126] for more on this matter).

2.7 Computational Complexity

Roughly, there are two kinds of solvable problems that are not too difficult—those that can be solved easily and those that can be solved with difficulty. In particular, when a problem is characterized as difficult, then

- if there is an algorithmic solution to this problem, it can be checked quickly; and
- an algorithmic solution will require an impossibly long time to yield an output.

In different words, it is necessary to examine the operation of the Turing machine that solves a particular problem in order to decide whether a problem is difficult or not. Formally,

Definition 2.7.1 A Turing machine is *polynomial bounded* when there is polynomial $p(n)$ such that for any input x there is no configuration C such that

$$q \triangleright \sqcup x \vdash_{\mathcal{M}}^{p(|x|)+1} C,$$

where $p(|x|) + 1$ is the length of the computation. In different words, the machine always halts after at most $p(|x|)$ steps.

Definition 2.7.2 A language is called *polynomial decidable* if there is polynomially bounded Turing machine that decides it. The class of all polynomial decidable languages is denoted P .

Definition 2.7.3 A nondeterministic Turing machine is said to be *polynomial bounded* when there is a polynomial $p(n)$ such that for any input x , there is no configuration C such that

$$q \triangleright \sqcup x \vdash_{\mathcal{N}}^{p(|x|)+1} C.$$

In different words, no computation of this machine requires more than polynomially many steps.

Definition 2.7.4 The class of all languages that are decided by a polynomially bounded non-deterministic Turing machine is denoted NP .

Theory of Fuzzy Computation

Syropoulos, A.

2014, XII, 162 p. 12 illus. in color., Hardcover

ISBN: 978-1-4614-8378-6