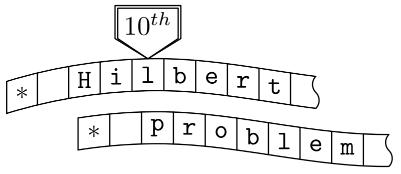


10^{th} Hilbert Problem

**Yu. Matiyasevich, Ya. Abramov, A. Belov-Kanel,
I. Ivanov-Pogodaev, A. Malistov, I. Netay**

C. Turing machines

The number-theoretic techniques developed in the previous chapters will permit us to tackle Hilbert's Tenth Problem from the point of view of computability theory. But first we must clarify what is to be understood by the word "process" when Hilbert speaks of "a process according to which it can be determined by a finite number of operations whether the equation is solvable in rational integers". There are many ways in which this needed clarification can be provided, but the fundamental idea is always the same. We are able to convince ourselves that we have a general method for solving problems of a particular kind if, by using this method, we can solve any individual problem of that kind without employing our creative abilities, that is, so to speak, mechanically.

We deal with computers not only in real life, but also in mathematics. Notion of abstruct machine allows us to formalize idea of *general method* for some class of problems.

Now let us describe *Turing machines.* A Turing machine has memory in the form of a tape divided into cells. The tape has a single end: to make matters definite, we assume it to be on the left. To the right, the tape is potentially infinite. This means that, in contrast with actual physical computers, a computation by a Turing machine will never lead to an abnormal termination with the diagnostic "insufficient memory" On the other hand, any particular computation will require only finitely many cells.

Each cell will either be empty or will contain a single symbol from a finite set of symbols $A = \{a_1, a_2, \dots a_w\}$ called an alphabet. Different machines may have different alphabets. One of the symbols will play a special role in that it will always mark the left-most cell and appear nowhere else. We use the symbol "*" for this marker. In addition, we need a symbol to denote an empty cell, and we follow tradition in using the letter "$\wedge$" for this purpose.

Symbols on the tape are read and written by a *head*, which at each moment of discrete time scans one of the cells. The head can move along the tape to the left and to the right.

At each moment, the machine is in one of finitely many states that, following tradition, we will denote by $q_1, \dots, q_v$. One of the states is declared to be initial, and we shall always suppose that this is q_1. In addition, one or more states are declared to be final.

The next action of a machine is totally determined by its current state and the symbol scanned by the head. In a single step, the machine can change the symbol in the cell, move the head one cell to the left or to the right, and pass into another state. The actions are defined by a set of instructions of the form:

$$\text{«state } q_i + \text{symbol } a_j \rightarrow \texttt{Left}(\texttt{Right}, \texttt{Stop}) + q_k + a_\ell\text{».}$$

This instruction means the the following «if q_i is the current state of the head which are situated in the cell with simbol a_j, then the machine do the following: 1. Symbol a_j must be changed for the symbol a_ℓ; 2. Head moves to the left (L), right (R) or stand (S); 3. Head changes its state for q_k».

These instructions must be written for all possible combinations of symbol q_k and state q_i. The whole pack of these instructions is called *the program* of the machine.

All the machines that we are going to construct will have the same alphabet $\{*, 0, 1, 2, 3, \lambda\}$.

There will be two final states: q_2 and q_3 and we shall interpret reaching q_2 as the answer «yes» and reaching q_3 as the answer «no». The cells containing the symbol «λ» will play the role of proxies for empty cells, in the following sense: only empty cells and cells containing the symbol λ may be situated to the right of a cell containing the symbol λ and for any state q_i the instructions with left-hand sides $q_i\lambda$ and $q_i\wedge$ will have identical right-hand sides.

Example of the simpliest machine. Machine `LEFT` with the following instructions

$$\begin{aligned}
\boldsymbol{q_1*} &\to \boldsymbol{*Sq_2}\\
\boldsymbol{q_1 0} &\to \boldsymbol{0Lq_2}\\
\boldsymbol{q_1 1} &\to \boldsymbol{1Lq_2}\\
\boldsymbol{q_1 2} &\to \boldsymbol{2Lq_2}\\
\boldsymbol{q_1 3} &\to \boldsymbol{3Lq_2}\\
\boldsymbol{q_1\lambda} &\to \boldsymbol{\lambda Lq_2}\\
\boldsymbol{q_1\wedge} &\to \boldsymbol{\lambda Lq_2}
\end{aligned}$$

moves the head one cell to the left unless it was already scanning the leftmost cell marked by the symbol «$*$».

Machine `WRITE`(0) with instructions

$$\begin{aligned}
\boldsymbol{q_1*} &\to \boldsymbol{*Sq_2}\\
\boldsymbol{q_1 0} &\to \boldsymbol{0Sq_2}\\
\boldsymbol{q_1 1} &\to \boldsymbol{0Sq_2}\\
\boldsymbol{q_1 2} &\to \boldsymbol{0Sq_2}\\
\boldsymbol{q_1 3} &\to \boldsymbol{0Sq_2}\\
\boldsymbol{q_1\lambda} &\to \boldsymbol{0Sq_2}\\
\boldsymbol{q_1\wedge} &\to \boldsymbol{0Sq_2}
\end{aligned}$$

writes the symbol «0» to the cell scanned by the head unless it is the leftmost cell containing the marker «$*$». Similar actions are performed by the machines `WRITE`(1), `WRITE`(2), `WRITE`(3) and `WRITE`(λ) the instructions for which can be obtained from written instructions by replacing the symbol «0» in the right-hand sides of the instructions by «1», «2» «3» and «λ» respectively.

Machine `READ`(0) with instructions

$$\begin{aligned}
\boldsymbol{q_1*} &\to \boldsymbol{*Sq_3}\\
\boldsymbol{q_1 0} &\to \boldsymbol{0Sq_2}\\
\boldsymbol{q_1 1} &\to \boldsymbol{1Sq_3}\\
\boldsymbol{q_1 2} &\to \boldsymbol{2Sq_3}\\
\boldsymbol{q_1 3} &\to \boldsymbol{3Sq_3}\\
\boldsymbol{q_1\lambda} &\to \boldsymbol{\lambda Sq_3}\\
\boldsymbol{q_1\wedge} &\to \boldsymbol{\lambda Sq_3}
\end{aligned}$$

determines whether the cell scanned by the head contains the symbol «0» or not and then halts in state q_2 or q_3, accordingly; by our convention these correspond respectively to "yes" or "no" answers. In a similar manner, machines `READ`(1), `READ`(2), `READ`(3), `READ`($*$) determine the presence of symbols «1», «2», «3», «$*$».

⧫ **C0.** a) *Construct the machine* `STOP`, *which goes directly into the final state q_3 from the state q_1.*
b) *Construct the machine* `READNOT`(x) *which recognizes the absence of the symbol x in the cell observed by the head.*

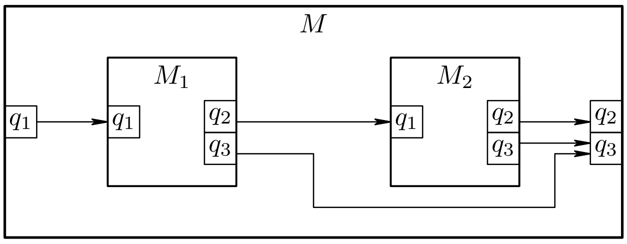
Composition of machines: two ways

The First method for constructing a new Turing machine M from two given machines M_1 and M_2 is as follows:

1. In all instructions of machine M_1, the final state q_2 is replaced by q_{v+1} where v is the number of states of machine M_1 (it should be recalled that final states can occur only in the right-hand sides of instructions).

2. In all instructions of machine M_2 every non-final state q_i is replaced by q_{v+i} (in particular, q_1 is replaced by q_{v+1}.

3. The set of instructions of the new machine M consists of the instructions of both of the given machines, modified as described above.



The action of machine M clearly consists of the consecutive execution of the actions of machines M_1 and M_2 *as originally constituted*, provided that machine M_1 halted in state q_2. To denote machine M, we shall use any one of the three notations:

$$M_1; M_2 \qquad M_1 \textbf{ and } M_2 \qquad \text{или} \qquad \textbf{if } M_1 \textbf{ then } M_2.$$

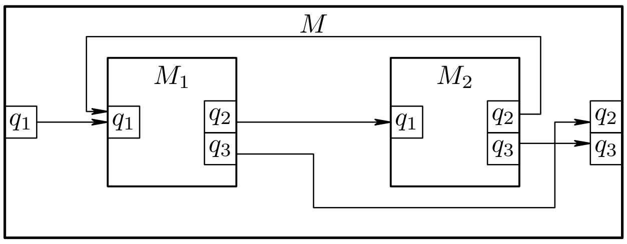
The second method for constructing a new Turing machine M from two given machines M_1 and M_2 allows us to construct cycles of the following types: `FOR i=1 TO N` or `WHILE`. Such type cycles appears not only in programming but in mathematics as well. For example, in order to calculate function $f(n) = 2^{2^n}$ the number 2 needs to be squared n times.

This method is as follows:

1. In all instructions of machine M_1 the final state q_2 is replaced by q_{v+1}, where v is the number of states of machine M_1 and the final state q_3 is replaced by q_2.

2. In all instructions of machine M_2 every non-final state q_i is replaced by q_{v+i} and the final state q_2 is replaced by q_1.

3. The set of instructions of the new machine M consists of the instructions of both of the given machines, modified as described above.



The action of this machine consists in performing in turn the actions of machines M_1 and M_2 as originally constituted until one of them enters the final state q_3

The Turing machine constructed in this way will be denoted by

$$\textbf{while } M_1 \textbf{ do } M_2 \textbf{ od}.$$

The notation introduced above resembles a primitive programming language. (In fact, every such "program" denotes a particular Turing machine.) However, this language is not so stupid as it looks like. It is powerful enouth to emulate any computer. Famous *Church thesis* claims that any algorithm can be realized on the Turing machine. The notion of Turing machine formalizes the notion of mechanical work in completely adecuate way. In the sequel, we identify Turing machines and algorithms.

Machine

$$\texttt{STAR} = \textbf{while } \texttt{READNOT}(*) \textbf{ do } \texttt{LEFT} \textbf{ od}$$

puts the head into the leftmost cell (marked by «∗»)

Machine

$$\texttt{VACANT} = \texttt{STAR};\ \textbf{while } \texttt{READNOT}(\lambda) \textbf{ do } \texttt{RIGHT} \textbf{ od}$$

puts the head into the leftmost cell containing the symbol «λ» if such exists; otherwise it puts the head into the leftmost empty cell.

♦ **C1.** a) *Construct the machine* `JUMP`*, which moves the head to the right until the first cell containing the symbol «0»; is reached; if all cells containing «0» are to the left of the head, then the machine will never halt.*
b) *Construct the sequence of machines* `FIND`(k)*, which move head to the cell containing the symbol «0» with number k from the left.*

We can store tuples of numbers on the tape of a Turing machine: for storing the tuple $(a_1, \ldots, a_n)$ we should use the symbols 1, separated by the symbols 0. For example, the tuple $(3, 1, 2, 0, 2)$ can be written by the following way: $*0111010110011\lambda\ldots$. So, Turing machine recieve some tuple and make transformation with it.

♦ **C2.** *Construct the Turing machines for transforming the tuple $(a_1, \ldots, a_n)$ into the tuple:*
a) $(a_1, \ldots, a_n, 0)$*;*
b) $(a_1, \ldots, a_n + 1)$*;*

♦ **C3.** a) *Construct the Turing machine which transorms the tuple $(a_1, \ldots, a_n)$ into the tuple $(a_1, \ldots, a_{n-1})$, (if $a_n = 0$ then the machine must stop in the state q_3.)*
b) *Construct the Turing machine which trancates the tuple $(a_1, \ldots, a_n)$, yielding the tuple $(a_1, \ldots, a_{n-1})$.*

♦ **C4.** *Construct the Turing machines which transforms the tuple $(a_1, \ldots, a_n)$ into the tuple:*
a) $(a_1, \ldots, a_n, a_k + a_l)$*, for fixed $1 \leqslant k,\ l \leqslant n$;*
b) $(a_1, \ldots, a_n, a_k \times a_l)$*, for fixed $1 \leqslant k,\ l \leqslant n$;*

♦ **C5.** *Construct the machine* `NOTGREATER`(k, l) *which compares the elements a_k и a_l of the tuple $(a_1, \ldots, a_n)$ and stops in state q_2 or q_3 depending on which of the two inequalities $a_k \leqslant a_l$ or $a_k > a_l$ holds.*

♦ **C6.** *Construct the machine which transform the tuple $(a_1, \ldots, a_n)$ into the tuple:*
a) $(a_1, \ldots, a_n, b, c)$*, where (b, c) is the pair that follows immediately after the pair (a_{n-1}, a_n) in Cantor numeration;*
b) $(a_1, \ldots, a_n, b, c)$*, where (b, c) is the pair with Cantor number a_n.*

♦ **C7.** *Consider the equation $D(a_1, \ldots, a_n, x_1, \ldots x_{m+1}) = 0$. Construct the Turing machine which uses the the tuple $(a_1, \ldots, a_n, y_0)$ and determines whether y_0 is the Cantor number of the tuple $(x_1, \ldots, x_{m+1})$ that satisfy the equation.*

♦ **C8.** *Consider the equation $D(a_1, \ldots, a_n, x_1, \ldots x_{m+1}) = 0$. Construct the Turing machine that will eventually halt, beginning with a representation of the tuple $(a_1, \ldots, a_n)$ if and only if the equation is solvable in the unknowns $x_1, \ldots, x_{m+1}$.*

We shall say that a set ω of n-tuples of natural numbers is *Turing semidecidable* if there is a Turing machine M that, beginning in state q_1 with a tape containing the canonical representation of the tuple $(a_1, \ldots, a_n)$ and with its head scanning the leftmost cell on the tape, will eventually halt if and only if $(a_1, \ldots, a_n) \in \omega$. In this case we say that M *semidecides M*. (However, there may be no way to estimate working time or recognize the situation $(a_1, \ldots, a_n) \notin \omega$).

Using the result C7 we prove that every Diophantine set is Turing semidecidable. The aim of the following problems is to establish the converse implication: every Turing semidecidable set is Diophantine.

Let M be a Turing machine that semidecides a set ω consisting of n-tuples of natural numbers. Let $\{\alpha_1, \ldots, \alpha_w\}$ be the alphabet of M. As the machine M carries out its operations, at each moment the symbols of $\{\alpha_1, \ldots, \alpha_w\}$ occupy only a finite initial segment of the tape of length, say, l, and we can therefore represent the tape by the tuple $(s_1, s_2, \ldots, s_m, \ldots, s_{l-1}, s_l)$ consisting of the subscripts of the symbols occurring in the cells.

The current state q_i and the position of the head can be represented by a tuple of the same length $(0, \ldots, 0, i, 0, \ldots, 0)$, in which all elements but one are zero; the only non-zero element is equal to the subscript of the state, and its position corresponds to the position of the head.

The triple consisting of the current contents of the tape, the state, and the position of the head will be called the *configuration.* Clearly, tuples $(s_1, s_2, \ldots, s_m, \ldots, s_{l-1}, s_l)$ and $(0, \ldots, 0, i, 0, \ldots, 0)$ uniquely determine the configuration. To represent these tuples, we shall use positional coding with a fixed base β that must be no less than 3 greater than v the number of states of machine M and greater than w, the number of symbols in the alphabet. The pair (p, t), is called a *configuration code* if p and t are ciphers of the tuples above respectively, to the base β.

So, our first goal will be to construct a Diophantine equation $D(p, t, x_1, \ldots, x_m) = 0$, such that if (p, t) – is the code of a configuration, then the equation is solvable in $x_1, \ldots, x_m$ if and only if machine M beginning in this configuration, eventually halts. We shall not be concerned about whether or not the equation is solvable when (p, t) is not a configuration code.

♦ **C9.** *Let machine M proceed directly from the configuration with code (p, t) to the configuration with code* $[\mathrm{NextP}\,(p, t), \mathrm{NextT}\,(p, t)]$. *Prove that the functions* NextP *and* NextT *is Diophantine.*

♦ **C10.** *Let machine M proceed in k steps from the configuration with code (p, t) to the configuration with code* $[\mathrm{AfterP}\,(p, t, k), \mathrm{AfterT}\,(p, t, k)]$. *Prove that the functions* AfterP *and* AfterT *is Diophantine.*

♦ **C11.** *Consider a Turing machine M. Construct an equation with parameters $a_1, \ldots, a_n$ which is decideable if and only if the M starts with the tuple $a_1, \ldots, a_n$ and halts.*

D. Universal Turing machine

In computer science we have met with different operational systems such as Windows, Dos, Unix which allows to work with any algorithms. But operational system is also a program. In mathematics we have a similar term: *univeral algorithm.* In previous problems we constructed special Turing machine for any new algorithm. But there is an another way. We can construct a new language for writing that Turing machine will do. Turing machine will read algorithm from the tape and follow it. This machine has the same program for different kind of algorithms on the tape. This machine is called the *universal Turing machine.* It is obvious that we can number all algorithms (Turing machines) and say something about Turing machine with number n.

♦ **D1. Coding.** *Invent a method for writing an algorithm (Turing machine) on the tape.*

♦ **D2.** *Prove that there exists a Turing machine which read from its tape numbers n and m and follows the n-numbered algorithm which works with number m.*

♦ **D3. Algorithm uses itself.** *Prove that there exists a Turing machine which read number n and follows n-th algorithm which works with number n.*

♦ **D4. Halting problem.** a) *Assume that there exists a universal Turing machine $U(m, n)$ which check that n-tn Turing machine starts with number m and stops after several steps. Prove that exists a algorithm $V(n)$ which show that n-th algorithm starts with n and stops. Prove that there exists an algorithm $T(n)$ which and stops if and only if $V(n)$ does not stop.*
b) *Let k be the number of the algorithm T. Does $T(k)$ stop?*

E. Universal Diophantine equations

Consider the set of equations $U(a_1, \ldots, a_n, k_1, \ldots, k_\ell, y_1, \ldots, y_v) = 0$. Suppose that there are two groups of parameters: parameters-elements $a_1, \ldots, a_n$ and parameters-codes $k_1, \ldots, k_\ell$.

Consider some Diophantine equation with n parameters $D(a_1, \ldots, a_n, x_1, \ldots, x_m) = 0$. Let us fix the parameters combinations such that this equation is decidable. Suppose that we can choose the values of the codes $k_1, \ldots, k_l$ such that the equation $U(a_1, \ldots, a_n, k_1, \ldots, k_\ell, y_1, \ldots, y_v) = 0$ is decidable with the same parameters combinations. If we can choose the codes such way for any equation $D(a_1, \ldots, a_n, x_1, \ldots, x_m) = 0$, then the equation $U(a_1, \ldots, a_n, k_1, \ldots, k_\ell, y_1, \ldots, y_v) = 0$ is called *universal*. We can say that every universal equation provides *coding* of Diophantine equations with fixed dimension. We can consider the n-tuple $[k_1(D), \ldots, k_\ell(D)]$ as the code for the equation $D = 0$.

⧫ **E1.** *Consider a universal Diophantine equation. Suppose that we can increase the number of unknowns. Prove that we can transorm the equation to the form with one code and the same number of parameters.*

⧫ **E2.** *Suppose that there exists a universal Diophantine equation which is coding one-dimension Diophantine sets.* ($n = 1$.) *Prove that there exists a constant m such that for any n there exists an universal Diophantine equation with $u = 1$ и $v = m$.*

⧫ **E3.** *Invent the coding (several natural numbers) for equation and potential solution such that there exists a Diophantine function to determine that this is the solution for this equations.*

⧫ **E4.** *Construct a universal Diophantine equation.*

⧫ **E5.** *Construct a Diophantine set M such that $\overline{M}$ is not Diophantine* ($\overline{M} \cap M = \varnothing$ and $\overline{M} \cup M = \mathbb{N}$).

⧫ **E6.** *Consider the set M of decidable Diophantine equations without parameters. Prove that $\overline{M}$ is not Diophantine.*

F. Final problem

⧫ **F1.** *Prove that there are no Turing machine that starts with number k and stops in the state q_2 or q_3 in complience with that the Diophantine equation with number k decidable or not.*