ON THE NOTION OF A COMPUTER

Z. PAWLAK

Institute of Mathematics, Polish Academy of Sciences, Warsaw, Poland

In the theory of mathematical machines various machines are considered, such as Turing machines, push-down machines, finite automata, etc. but little attention is given to formal definition of digital computers, programs and the study of their properties.

The author's belief as well as that of many other people working in computer field is that a further development of this field requires more close relations with the existing computers.

The paper contains formal definitions of a computer, a universal computer and a program. In the proposed language one can define and study machines which seem to be fairly good models of real computers. Some elementary theorems concerning computers are stated. One can find more details concerning the outlined topics in PAWLAK [1967].

1. Computers

1.1. *Memory*

Let A, Σ, V, be sets. Elements of A are called *addresses*, Σ is refered to as an *alphabet* and elements of Σ are called *symbols*. Elements of V are called *markers*. We allow the sets A, Σ to be finite or infinite and the set V is assumed to be always finite. By Λ we denote the distinguished symbol of Σ called the *empty* symbol.

Let C and L be the sets of functions with domain and codomain as given below

$$C \subseteq \Sigma^A \qquad L \subseteq A^V .$$

Every $c \in C$ is called *content* of the memory and every $l \in L$ is called *location* of the memory. We assume that for every $c \in C$, $c(x) \neq \Lambda$ for almost all $x \in A$.

DEFINITION 1. The memory is a system $P = \langle M, I, O \rangle$, where $M = C \times L$

is referred to as a set of *memory states*, and I, O are *input* and *output functions* of a memory with domain and codomain as follows:

$$I: \Sigma \times A^n \times M \to M, \qquad O: A^n \times M \to M \times \Sigma,$$

where $n=0, 1, 2, \ldots$ is fixed for given memory.

The function O may be considered as a pair of functions

$$O_1: A^n \times M \to M \quad \text{and} \quad O_2: A^n \times M \to \Sigma.$$

We can extend functions I and O for finite sequences of symbols and obtain new input and output functions I^*, O^*:

$$I^*: \Sigma^k \times A^n \times M \to M \quad \text{and} \quad O^*: A^n \times M \to M \times \Sigma^k,$$

where $k=0, 1, 2, \ldots$ is some fixed number for given memory.

Thus with every memory P we can associate the *memory function*

$$\Pi_P: \Sigma^k \times A^n \times M \to \Sigma^k$$

defined as follows

$$\bar{y}_k = \Pi_P(\bar{x}_k, \bar{a}_n, m) = O_2^*(\bar{a}_n, I^*(\bar{x}_k, \bar{a}_n, m)),$$

where $\bar{y}_k = y_1, \ldots, y_k$, $\bar{x}_k = x_1, \ldots, x_k$, $\bar{a}_n = a_1, \ldots, a_n$ and $y_i, x_i \in \Sigma$, $a_i \in A$, $m \in M$.

Two memories P and P' are said to be equivalent if and only if

$$\Pi_P = \Pi_{P'}.$$

If the memory function does not depend on some arguments we shall omit those arguments and write simple for example $\Pi_P(\bar{x}_k)$.

Example 1. One address memory. Let N denote the set of natural numbers 0, 1, 2, ... We assume for this memory $A = N$, $\Sigma = N \cup A$ and the set of markers V consists of only one element v. Input function for one address memory is as follows

$$\begin{aligned} I(x, a, m) &= m_1 = \langle c_1, l_1 \rangle, \quad \text{where} \\ c_1(z) &= \begin{cases} x & \text{if} \quad z = l_1(v), \\ c(z) & \text{if} \quad z \neq l_1(v), \end{cases} \\ l_1(v) &= a, \end{aligned}$$

and $a, z \in A$, $x \in \Sigma$, $m \in M$, $m = \langle c, l \rangle$.

The output function for this memory is

$$O(a, m) = \langle m_2, c(a) \rangle, \quad \text{where} \quad m_2 = \langle c_2, l_2 \rangle \quad \text{and} \quad c_2 = c,$$
$$l_2(v) = a.$$

The extended input and output functions for one address memory are

$$I^*(\bar{x}_k, a, m) = m_1 = \langle c_1, l_1 \rangle, \quad \text{where}$$
$$c_1(z) = x_i \quad \text{if} \quad z = a + i - 1 \quad \text{and} \quad c_1(z) = c(z) \quad \text{otherwise},$$
$$l_1(v) = a, \qquad 1 \leqslant i \leqslant k.$$
$$O^*(a, m) = \langle m_2, \bar{x}_k \rangle, \quad \text{where} \quad m_2 = \langle c_2, l_2 \rangle \quad \text{and} \quad c_2 = c,$$
$$l_2(v) = a,$$
$$x_i = c(a + i - 1).$$

One can easily show that for one address memory $\Pi_p(X) = X$ for all $X \in \Sigma^k$.

Example 2. Stack memory. The sets A, Σ, V are the same as in the Example 1. Input function for stack memory is the following

$$I(x, m) = m_1 = \langle c_1, l_1 \rangle, \quad \text{where}$$
$$c_1(z) = \begin{cases} x & \text{if} \quad z = l_1(v), \\ c(z) & \text{if} \quad z \neq l_1(v), \end{cases}$$
$$l_1(v) = l(v) + 1.$$

Output function for this memory is

$$O(m) = \langle m_2, c(l(v)) \rangle, \quad \text{where} \quad m_2 = \langle c_2, l_2 \rangle \quad \text{and} \quad c_2 = c,$$
$$l_2(v) = l(v) - 1 \quad \text{for} \quad l(v) > 0 \quad \text{and undefined for} \quad l(v) = 0.$$

The extended input and output functions for stack memory are as follows

$$I^*(\bar{x}_k, m) = m_1 = \langle c_1, l_1 \rangle, \quad \text{where}$$
$$c_1(z) = \begin{cases} x_i & \text{if} \quad z = l(v) + i \qquad (1 \leqslant i \leqslant k) \\ c(z) & \text{otherwise}, \end{cases}$$
$$l_1(v) = l(v) + i.$$
$$O^*(m) = \langle m_2, \bar{x}_k \rangle, \quad \text{where} \quad m_2 = \langle c_2, l_2 \rangle \quad \text{and} \quad c_2 = c,$$
$$l_2(v) = l(v) - (k - 1),$$
$$x_i = c(l(v) - i + 1).$$

One can easily verify that $\Pi_p(X) = X^{-1}$ for all $X \in \Sigma^k$, where X^{-1} is to mean $x_k, \ldots, x_1$.

1.2. *Instructions*

With every computer there is associated the finite set of *instructions*, $R = \{r_0, r_1, \ldots, r_s\}$. Instruction $r \in R$ is a function $r: A^n \times M \to M$, where $n \geqslant 0$ is some fixed number for a given computer.

Two instructions r and r' are said to be *equivalent* if and only if for all

$\bar{a}_n \in A^n$ and for all $m \in M$

$$r(\bar{a}_n, m) = r'(\bar{a}_n, m).$$

If for all $\bar{a}_n \in A^n$ and for all $m \in M$

$$r(\bar{a}_n, m) = m$$

then r is called *identity instruction* and will be denoted by r_0.

Composition of instructions r_1 and r_2 is the instruction r such that

$$r = r_1(\bar{b}_n, r_2(\bar{a}_n, m)), \qquad \bar{a}_n, \bar{b}_n \in A^n$$

written short as $r = r_1 r_2$.

The instruction

$$r' = \underbrace{r \ldots r}_{p}$$

is called an *iteration* of the instruction r and is written $r' = r^p$.

With every computer there is associated a finite set of *operations*, $F = \{f_0, f_1, \ldots, f_p\}$. Operation $f \in F$ is a function

$$f: \Sigma^{n_1} \to \Sigma^{n_2}, \qquad n_1, n_2 > 0.$$

Instruction r is called *admissible* for the memory P and the set of operations F if r can be represented in the form

$$r(\bar{a}_n, m) = I^*\{f[O_2(\bar{a}_n, m)], \bar{a}_n, O_1^*(\bar{a}_n, m)\},$$

where f is some operation from F and I^*, O^* are the extended input and output functions of the memory P.

Example 1. Transfer instruction. Let P be the memory for which $A = N$, $\Sigma = N \cup A$, $V = \{v_1, v_2\}$. Let us assume the following input and output functions for the memory P:

$I^*(x, a, b, m) = m_1 = \langle c_1, l_1 \rangle$, where $x \in \Sigma$, $a, b \in A$, $m, m_1 \in M$ and

$$c_1(z) = \begin{cases} x & \text{if } z = l(v_2) = b, \\ c(z) & \text{if } z \neq l(v_2), \end{cases}$$

$$l_1(y) = \begin{cases} b & \text{if } y = v_2, \\ l(y) & \text{if } y \neq v_2. \end{cases}$$

$$O^*(a, b, m) = \langle m_2, x \rangle, \quad \text{where } m_2 = \langle c_2, l_2 \rangle \text{ and } c_2 = c,$$

$$l_2(y) = \begin{cases} a & \text{if } y = v_1, \\ l(y) & \text{if } y \neq v_1, \end{cases}$$

$$x = c(l_2(v_1)) = c(a).$$

Let us denote the transfer instruction by $T(a, b, m)$ and assume that the set of computer operations F contains the identity operation $i(x)=x$. We define transfer instruction as

$$T(a,b,m)=m'=\langle c', l'\rangle\,,\quad \text{where}$$
$$c'(z)=\begin{cases} c(a) & \text{if} \quad z=l(v_2),\\ c(z) & \text{if} \quad z\neq l(v_2),\end{cases}$$
$$l'(y)=\begin{cases} a & \text{if} \quad y=v_1,\\ b & \text{if} \quad y=v_2.\end{cases}$$

One can easily see that so defined transfer instruction is admissible for the assumed set of operations and assumed memory because

$$T(a,b,m)=I^*\{i[O_2^*(a,b,m)],a,b,O_1^*(a,b,m)\}\,,$$

for any a, b, m.

Example 2. Two address addition instruction. Let us consider memory with A, Σ, V and I^* the same as in the Example 1 but the output function defined as follows

$$O^*(a,b,m)=\langle m_2, \bar{x}_2\rangle\,,\quad \text{where}\quad m_2=\langle c_2, l_2\rangle \quad \text{and} \quad c_2=c\,,$$
$$l_2(y)=\begin{cases} a & \text{if} \quad y=v_1,\\ b & \text{if} \quad y=v_2,\end{cases}$$
$$x_1=c(l_2(v_1))=c(a)\,,$$
$$x_2=c(l_2(v_2))=c(b)\,.$$

Let $A(a,b,m)$ denote a two address addition instruction, defined as $A(a,b,m)=m'=\langle c', l'\rangle$ where

$$c(z)=\begin{cases} c(a)+c(b) & \text{if} \quad z=l(v_2)=b\,,\\ c(z) & \text{if} \quad z\neq l(v_2),\end{cases}$$
$$l(y)=\begin{cases} a & \text{if} \quad y=v_1,\\ b & \text{if} \quad y=v_2.\end{cases}$$

One can easily verify that if the computer operations set contains addition, then the instruction $A(a,b,m)$ is admissible

$$A(a,b,m)=I\{+[O_2(a,b,m)]\,a,b,O_1(a,b,m)\}\,.$$

In the sequel it will be assumed that all the instructions are admissible, thus "instruction" will always mean "admissible instruction".

DEFINITION 2. Instruction which changes the content of at most one address in the memory or changes the location of at most one marker in the memory is called *simple*.

THEOREM 1. Every instruction can be represented as a composition of simple instructions.

1.3. *Conditions*

With every computer we associate a finite set $W=\{W_1, W_2, \dots, W_t\}$, $W_i \subset M$. The elements of W are called *conditions*. We say that the memory state $m \in M$ satisfies the condition W_i if and only if $m \in W_i$.

We say that a condition W_i is admissible for the memory P if and only if

$$W_i=\{m : m \in M \quad \text{and} \quad O_2^*(\bar{a}_n, m)=x\}$$

for some $\bar{a}_n$, where O_2^* is the output function of the memory P and x is some fixed symbol of the alphabet Σ. We shall consider only admissible conditions in this paper. For example for one address memory the condition may be the set

$$W_i=\{m : m \in M \quad \text{and} \quad c(a)=0\}, \qquad 0 \in \Sigma,$$

for some $a \in A$.

1.4. *Control*

Let Q be a finite set of numbers $\{1, 2, \dots, s\}$. A *graph* will be defined as the system $G=\langle Q, Q', h_0, h_1\rangle$, where $Q' \subset Q$ and $h_0 : Q-Q' \to Q$, $h_1 : Q-Q' \to Q$. Q is referred to as the set of *points* of G and Q' is referred to as the set of *end* points of G. $q \in Q-Q'$ is called *initial* point of G, if for every $q' \in Q$, $q \neq h_i(q')$, $i=0,1$.

The sequence $q_1, \dots, q_r$, $q_i \in Q$ is called the *path* from q_1 to q_r in G, if for all i, $1 \leqslant i < r$, $q_{i+1}=h_j(q_i)$, $j=0,1$.

By the *flow graph* we shall mean the graph G which satisfies the following conditions:

1°. G contains exactly one initial point, written q_0.

2°. The set of end points is not empty.

3°. For every point $q \in Q-q_0$ there is a path from q_0 to q in G.

4°. For every point $q \in Q-Q'$ there is a path from q to q', where $q' \in Q'$.

DEFINITION 3. The control S of the computer is a system $S=\langle G, \varphi, \psi, \nu\rangle$ where G is the flow graph and φ, ψ, ν are functions with domains and codomains as given below

$$\varphi : Q \to R\,, \qquad \psi : Q \to W\,, \qquad \nu : M \times Q \to M \times Q\,,$$

and R, W are some fixed sets of instructions and conditions respectively, M is the fixed set of memory states and Q is the set of points of the graph G. We assume that for every end point of Q we associate the identity instruction r_0.

Elements of Q are also called *control states*. Elements of the set $T = M \times Q$ are called *computer states*. The function ν is called *transition* function. If q_0 is the initial state of the control then $\langle m, q_0 \rangle$ is called the *initial state of the computer*; if q is the end state of the control, then $\langle m, q \rangle$ is called the *end state of* the computer, where $m \in M$ is some state of the memory. Let $t = \langle m, q \rangle$ and $t' = \langle m', q' \rangle$.

Transition function will be defined as follows

$$\begin{aligned} \nu(t) &= t'\,, \quad \text{where} \\ m' &= [\varphi(q)]\,(\bar{a}_n, m) \\ q' &= \begin{cases} h_0(q) & \text{if} \quad m' \in \psi(q)\,, \\ h_1(q) & \text{if} \quad m' \notin \psi(q)\,. \end{cases} \end{aligned}$$

1.5. *Computers*

DEFINITION 4. Computer is a system $\mathscr{M} = \langle P, R, W, S \rangle$, where P, R, W, S are the memory, the set of instructions, the set of the conditions and the control of the computer respectively.

The sequence $t_0, t_1, \ldots, t_k$ is called the *computation* of the computer $\mathscr{M}$ if and only if for each $t_i \in T$ (where T is the set of states of $\mathscr{M}$), and for every i, $1 \leqslant i < k$, $t_{i+1} = \nu(t_i)$ and t_0, t_k are the initial state and the end state of the computer $\mathscr{M}$ respectively. t_i are called *steps* of the computation.

t_k will be denoted by $\mathrm{Com}(t_0)$. The function Com may be considered as a pair of functions Com_1 and Com_2 such that $\mathrm{Com}_1(t_0) = m$ and $\mathrm{Com}_2(t_0) = q$, and $\mathrm{Com}(t_0) = t_k = \langle m, q \rangle$.

Thus with every computer $\mathscr{M}$ there is associated the function

$$\phi_{\mathscr{M}}(\bar{x}_k, \bar{a}_n, m) = O_2^* \{\bar{a}_n, \mathrm{Com}_1 [I^*(\bar{x}_k, \bar{a}_n, m), q_0]\}\,,$$

where I^* and O^* are input and output functions of the memory of the computer $\mathscr{M}$.

DEFINITION 5. We say that the function $f(x_1, \ldots, x_k)$ is *computable* by the computer $\mathscr{M}$ if and only if $f = \phi_{\mathscr{M}}$ for some $\bar{a}_n$ and m.

DEFINITION 6. The set $\Sigma' \subset \Sigma^*$ is *decidable* on the computer $\mathcal{M}$ if and only if for all $x \in \Sigma^*$ there exist such $\bar{a}_n$ and m that

$$\phi_{\mathcal{M}}(x, \bar{a}_n, m) = \begin{cases} 0 & \text{if} \quad x \in \Sigma' \\ 1 & \text{if} \quad x \notin \Sigma', \end{cases}$$

where 0, 1 are some distinguished elements of Σ.

DEFINITION 7. The set $\Sigma' \subset \Sigma^*$ is *generable* on $\mathcal{M}$ if and only if for all $x \in \Sigma'$ there is a sequence $\bar{x}_k = x_1, \dots, x_k$, $x_i \in \Sigma$ such that

$$\phi_{\mathcal{M}}(\bar{x}_k, \bar{a}_n, m) = x \quad \text{for some} \quad \bar{a}_n \quad \text{and} \quad m.^*$$

Computers $\mathcal{M}$ and $\mathcal{M}'$ are equivalent if and only if $f_{\mathcal{M}} = f_{\mathcal{M}'}$, where $f_{\mathcal{M}}$ denotes the function computable by the computer $\mathcal{M}$, and similar $f_{\mathcal{M}'}$.

Note. If the function f is given and we search for the computer $\mathcal{M}$ such that $f = \phi_{\mathcal{M}}$ one may speak of *synthesis* of computer $\mathcal{M}$. If computer $\mathcal{M}$ is given and we search for the function $\phi_{\mathcal{M}}$ one can speak about the *analysis* of the computer $\mathcal{M}$.

2. Universal computers

2.1. *Classes of computers*

DEFINITION 8. Two computers $\mathcal{M}$ and $\mathcal{M}'$ are of the same class if and only if the memories, the instructions and the conditions of both computers are identical.

In other words the computers belonging to the same class may differ at most in the control.

Let $\mathcal{K} = \{\mathcal{M}_1, \mathcal{M}_2, \dots\}$ be the class of computers. Then by $f_{\mathcal{K}} = \{f_{\mathcal{M}_1}, f_{\mathcal{M}_2}, \dots\}$, where $f_{\mathcal{M}_i}$ is the function computable by the computer $\mathcal{M}_i$ – we denote the class of functions computable by the computers of the class $\mathcal{K}$. Two classes of computers $\mathcal{K}$, $\mathcal{K}'$ are equivalent if and only if $f_{\mathcal{K}} = f_{\mathcal{K}'}$.

One can easily define the class of one address computers, the class of two address computers etc. and show that these classes are equivalent.

2.2. *Universal computer*

Let $\mathcal{K}$ be the class of computers with alphabet Σ. Computers from $\mathcal{K}$ are denoted by $\mathcal{M}$. Let $\mathfrak{M}$ be the following computer not belonging to the class $\mathcal{K}$. For the sake of simplicity we assume that the computer $\mathfrak{M}$ has the same

* Definitions 5, 6 and 7 are modified versions of definitions given in SCOTT [1967].

set of addresses and the same alphabet as the computers from the class $\mathscr{K}$, i.e. $A=0, 1, 2, \ldots$ and $\Sigma=0, 1, 2\ldots$. The input and output functions for the computer $\mathfrak{M}$ we assume as follows

$$\mathbf{I}^*: \mathscr{K} \times \Sigma^k \times \mathbf{M} \to \mathbf{M} \quad \text{and} \quad \mathbf{O}^*: \mathbf{M} \to \mathbf{M} \times \Sigma^k,$$

where $\mathbf{M}$ denotes the set of memory states of the computer $\mathfrak{M}$.

DEFINITION 9. The computer $\mathfrak{M}$ is a universal computer for the class of computers $\mathscr{K}$ if and only if for all $\bar{x}_k \in \Sigma^k$, $\mathscr{M} \in \mathscr{K}$ there exist $\mathbf{m} \in \mathbf{M}$ such that

$$\mathbf{O}_2^* \{\text{Com}_1 [I^*(\mathscr{M}, \bar{x}_n, \mathbf{m}), \mathbf{q}_0]\} = f_{\mathscr{M}}(\bar{x}_n),$$

where $\mathbf{q}_0$ is the initial control state of $\mathfrak{M}$.

2.3. *Synthesis of universal computer*

Let $\mathscr{M}=\langle P, R, W, S\rangle$ be any computer which belongs to $\mathscr{K}$. We shall now define computer $\mathfrak{M}=\langle \mathbf{P}, \mathbf{R}, \mathbf{W}, \mathbf{S}\rangle$ in terms of computers of the class $\mathscr{K}$ and then we show that $\mathfrak{M}$ is universal computer for the class of computers $\mathscr{K}$. In order to define $\mathfrak{M}$ we have to give $\mathbf{P}, \mathbf{R}, \mathbf{W}, \mathbf{S}$. Let us start with the construction of $\mathbf{P}$.

2.3.1. Memory of the computer $\mathfrak{M}$

We recall that as the set of addresses for $\mathbf{P}$ we assumed the set of natural numbers $N=0, 1, 2, \ldots$. Let $\mathbf{C}$ be the set of content functions of $\mathbf{P}$, and let Q be the set of control states of $\mathscr{M}$. By $\mathbf{C}_r = \mathbf{C}|(N-Q)$ we denote the set of partial content functions with domain restricted to the set $N-Q$. We shall call functions from $\mathbf{C}_r$ the *reduced content functions*. The set of markers $\mathbf{U}$ in $\mathbf{P}$ consists of one element $\mathbf{u}$. In order to define the input and output functions for $\mathbf{P}$ we have to introduce some additional notions.

Let M be the set of memory states of $\mathscr{M}$, and let $S=\langle G, \varphi, \psi, v\rangle$ be the control of $\mathscr{M}$, where $G=\langle Q, Q', h_0, h_1\rangle$. We introduce 1–1 mappings $\kappa: M \to \mathbf{C}_r$, $\rho: R \to \mathbf{R}$, where $R=\langle r_0, r_1, \ldots, r_s\rangle$ is the set of instructions of the computer $\mathscr{M}$ and $\mathbf{R}=\langle \mathbf{r}_0, \mathbf{r}_1, \ldots, \mathbf{r}_s\rangle$, $\mathbf{r}_i: A^n \times \mathbf{C}_r \to \mathbf{C}_r$ – such that for all $m \in M$, $\bar{a}_n \in A^n$ and $r \in R$

$$\kappa(r(\bar{a}_n, m)) = [\rho(r)](\bar{a}_n, \kappa(m)).$$

By φ_ρ is to mean the function $\varphi_\rho: Q \to \mathbf{R}$ such that $\varphi_\rho(q) = \rho(\varphi(q))$, for all $q \in Q$.

Let us denote by $W=\{W_1, \ldots, W_p\}$ the set of conditions of $\mathscr{M}$. By ω is to mean 1–1 mapping $\omega: W \to \mathbf{W}$, where

$$\mathbf{W} = \{\mathbf{W}_1, \mathbf{W}_2, \ldots, \mathbf{W}_s\}, \qquad \mathbf{W}_i \subset \mathbf{C}_r$$

such that for all $m \in M$

$$\kappa(m) \in \omega(W_i) \text{ if and only if } m \in W_i.$$

ψ_ω denotes the function $\psi_\omega: Q \to \mathbf{W}$ such that $\psi_\omega(q) = \omega(\psi(q))$, for all $q \in Q$. The set of functions $\langle \varphi_\rho, \psi_\omega, h_0, h_1 \rangle$ we shall denote by Θ. Let δ be 1–1 mapping

$$\delta: \Theta \to \Sigma',$$

where $\Sigma' \subset \Sigma$ and Σ' is finite. δ will be called the *encoding function*. Thus we are able now to represent $\Theta = \langle \varphi_\rho, \psi_\omega, h_0, h_1 \rangle$ in the alphabet of $\mathfrak{M}$.

The input function $\mathbf{I}^*$ of $\mathbf{P}$ we define as follows

$$\mathbf{I}^*(\mathscr{M}, \bar{x}_k, \mathbf{m}) = \mathbf{m}' = \langle \mathbf{c}', \mathbf{l}' \rangle, \quad \text{where}$$

$$\mathbf{c}'(z) = \begin{cases} \delta(\Theta(z)) & \text{if } 0 < z \leqslant |Q|, \\ \kappa(I^*(\bar{x}_k, \bar{a}_n, m)) & \text{if } z > |Q|, \end{cases}$$

$$\mathbf{l}'(\mathbf{u}) = q_0,$$

and I^* is the input function of $\mathscr{M}$, $|Q|$ denotes the number of elements of Q.

The output function $\mathbf{O}^*: \mathbf{M} \to \mathbf{M} \times \Sigma^k$ of $\mathfrak{M}$ satisfy the condition:

for all $m \in M$ there exists such $\mathbf{l} \in \mathbf{L}$ that $O_2^*(m) = \mathbf{O}_2^*(\kappa(m), \mathbf{l})$.

Thus we defined the memory of $\mathfrak{M}$.

2.3.2. Instructions of the computer $\mathfrak{M}$

Now we have to define the instructions of $\mathfrak{M}$. The set of instructions of $\mathfrak{M}$ will consist of two instructions $\mathscr{R}_0$, $\mathscr{R}$. $\mathscr{R}_0$ is the identity instruction and $\mathscr{R}$ is defined in the following way:

$$\mathscr{R}(\mathbf{m}) = \mathbf{m}' = \langle \mathbf{c}'_r, \mathbf{l}' \rangle, \quad \text{where} \quad \mathbf{m} = \langle \mathbf{c}_r, \mathbf{l} \rangle \quad \text{and}$$

$$\mathbf{c}'_r = [\varphi_\rho(\mathbf{l}(\mathbf{u}))](\mathbf{c}_r),$$

$$\mathbf{l}'(\mathbf{u}) = \begin{cases} h_0(\mathbf{l}(\mathbf{u})) & \text{if } \mathbf{c}'_r \in \psi_\omega(\mathbf{l}(\mathbf{u})) \\ h_1(\mathbf{l}(\mathbf{u})) & \text{if } \mathbf{c}'_r \notin \psi_\omega(\mathbf{l}(\mathbf{u})). \end{cases}$$

2.3.3. Conditions of the computer $\mathfrak{M}$

In the computer $\mathfrak{M}$ we shall consider only one condition $\mathscr{W}$

$$\mathscr{W} = \{\mathbf{m} : \mathbf{m} \in \mathbf{M} \text{ and } \mathbf{l}(\mathbf{u}) \in Q' \text{ and } \psi_\omega \mathbf{l}(\mathbf{u}) = r_0\},$$

where Q' is the set of end states of M and r_0 is the identity instruction of $\mathfrak{M}$. The condition $\mathscr{W}$ will be written STOP.

2.3.4. Control of the computer $\mathfrak{M}$

The control of $\mathfrak{M}$ we shall simply give in the form of a table

	$\bar{\varphi}$	$\bar{\psi}$	$\bar{h}_0$	$\bar{h}_1$
$\mathbf{q}_0$	$\mathscr{R}_0$	STOP	$\mathbf{q}_1$	$\mathbf{q}_1$
$\mathbf{q}_1$	$\mathscr{R}_0$	STOP	$\mathbf{q}_2$	$\mathbf{q}_3$
$\mathbf{q}_2$	$\mathscr{R}_0$	STOP	–	– END
$\mathbf{q}_3$	$\mathscr{R}$	STOP	$\mathbf{q}_1$	$\mathbf{q}_2$

In the table the control states are $\mathbf{q}_0, \mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3$ and $\mathbf{q}_2$ being the end state. $\bar{\varphi}, \bar{\psi}, \bar{h}_0, \bar{h}_1$ are the corresponding functions of the computer $\mathfrak{M}$.

Thus we completed the definition of the computer $\mathfrak{M}$. Now we are able to prove the following theorem

THEOREM 2. The computer $\mathfrak{M}$ is universal computer for the class of computer $\mathscr{K}$.

PROOF. In order to prove this theorem we have to show that for every function $f_{\mathscr{M}}$ and for all $\bar{x}_k$ there is a computation in $\mathfrak{M}$ such that

$$1)\qquad \mathbf{O}_2^* \{\mathrm{Com}_1 [\mathbf{I}^*(\mathscr{M}, \bar{x}_n, \mathbf{m}), \mathbf{q}_0]\} = f_{\mathscr{M}}(\bar{x}_n)$$

for some $\mathbf{m} \in \mathbf{M}$. Because $f_{\mathscr{M}}$ is computable by $\mathscr{M}$ therefore there is in $\mathscr{M}$ the computation such that

$$2)\qquad O_2^* \{\mathrm{Com}_1 [I^*(\bar{x}_k, \bar{a}_n, m), q_0]\} = f_{\mathscr{M}}(\bar{x}_k)$$

for some $\bar{a}_n$ and m. From the definition of $\mathbf{I}^*$ it follows that to the initial state of the computer $\mathscr{M}$ corresponds exactly one initial computer state of M, which is

$$\mathbf{t}_0 = \langle \mathbf{I}^*(\mathscr{M}, \bar{x}_k, \mathbf{m}), \mathbf{q}_0 \rangle .$$

From the definition of the control of $\mathfrak{M}$ results that to each step of the computation in $\mathscr{M}$ corresponds exactly one step in the computation in $\mathfrak{M}$ – such that $\mathbf{t}_{i+1} = \bar{v}(\mathbf{t}_i)$ if and only if $t_{i+1} = v(t_i)$, where $\bar{v}$ is the transition function of $\mathfrak{M}$. Further from the definition of the control of $\mathfrak{M}$ it follows that $\mathfrak{M}$ is in final state if and only if $\mathscr{M}$ is in its final state. By the definition of ρ we have

$$\langle \kappa \{\mathrm{Com}_1 [I^*(\bar{x}_k, \bar{a}_n, m), q_0]\}, 1 \rangle = \mathbf{Com}_1 [\mathbf{I}^*(\mathscr{M}, \bar{x}_k, \mathbf{m}), \mathbf{q}_0] .$$

By the definition of the output function $\mathbf{O}^*$ and by 2) we obtain 1) which completes the proof.

In this manner one can define various universal computers, for example one address universal computer, two addresses universal computer etc. and prove some properties of this computers.

3. Programs

Let $\mathscr{P}=\mathbf{C}|Q$ be the set of the partial content functions of the universal computer $\mathfrak{M}$ with domain restricted to the set Q, such that for all $q\in Q$, $\mathscr{M}\in\mathscr{K}$,

$$\not{p}(q)=\delta(\Theta(q)), \qquad \not{p}\in\mathscr{P}.$$

Definition 10. Each function $\not{p}\in\mathscr{P}$ is called a *program* in $\mathfrak{M}$; $\Theta(q)$ is called a *program instruction* (which is to be distinguished from *machine instructions* considered in the first section of this paper); q is called *label* or an *address* of the program instruction $\Theta(q)$; $\delta(\Theta(q))$ is called a *code* of $\Theta(q)$.

With every program $\not{p}\in\mathscr{P}$ in $\mathfrak{M}$ we can associate a function $f_{\not{p}}$ computable by the program $\not{p}$ on the machine $\mathfrak{M}$. Two programs $\not{p}$ and $\not{p}'$ are said to be equivalent if and only if $f_{\not{p}}=f_{\not{p}'}$.

It seems very important to study in detail the question of equivalence of programs but this is not the aim of this paper. We are going now to state without proof theorem concerning the form of program instructions in universal computers.

Theorem 3. For every universal computer $\mathfrak{M}$ there exists an equivalent universal computer $\mathfrak{M}'$ such that for each program instruction

$$\Theta(q)=\langle\varphi_{\rho}(q),\psi_{\omega}(q),h_0(q),h_1(q)\rangle$$

of $\mathfrak{M}'$, any of below given properties may hold

a) $h_0(q)=q+1$ for all $q\in Q$,

a') $h_1(q)=q+1$ for all $q\in Q$,

b) if $\varphi_{\rho}(q)\neq\mathbf{r}_0$ then $h_0(q)=h_1(q)=q+1$ for all $q\in Q$,

c) if $\varphi_{\rho}(q)=\mathbf{r}_0$ then $h_0(q)=q+1$ for all $q\in Q$,

c') if $\varphi_{\rho}(q)=\mathbf{r}_0$ then $h_1(q)=q$ for all $q\in Q$.

Much attention has been recently paid to the semantics of programming languages. It seems that the presented formalization of computers and programs contribute to this problem. The meaning of the program $\not{p}$ can be defined as the computation carried out on the universal computer $\mathfrak{M}$ according to the program p. Thus we can define the valuation of programs in the computer states in such a manner that the computation associated with the program yields the value of the computed functions. Thus we have the method for solving the following problem: let p be program of a universal computer $\mathfrak{M}$, and let h be a computable function. We ask whether $h=f_p$. This seems to be of some interest not only for the theory of programming but also for practical computation.

Acknowledgement

The author gratefully acknowledges many helpful suggestions of Dr. A. Mazurkiewicz and Dr. A. Wakulicz.

References

DAVIS, M., Computability and unsolvability (New York, McGraw Hill, 1958).

ELGOT, C.C. and A. ROBINSON, Random-access stored-program machines, an approach to programming languages, J. ACM 11 (1964) 365–399.

HERMES, H., Aufzählbarkeit, Entscheidbarkeit, Berechenbarkeit (Berlin, Springer, 1961)

KALMÁR, L., On an algebraic theory of automatic digital computers, Colloq. Foundations of Mathematics, Mathematical Machines and their Applications, Tihany, Sept. 1962, p. 129.

PAWLAK, Z., Organization of digital computers (in polish). Manuscript of lectures held at Warsaw University in 1966/67.

SCOTT, D., Some definitional suggestions for automata theory (Stanford University, 1967).