Časopis pro pěstování matematiky

Karel Čulík

A note on comparison of Turing machines with computers

Časopis pro pěstování matematiky, Vol. 100 (1975), No. 2, 118--128

Persistent URL: http://dml.cz/dmlcz/108770

Časopis pro pěstování matematiky, roč. 100 (1975), Praha

A NOTE ON COMPARISON OF TURING MACHINES WITH COMPUTERS

KAREL ČULÍK, Praha

(Received October 25, 1973)

Computers are devices using only functions the domain and range of which is perfectly determined (the movement of a scanning head in Turing machine is no function in this sense) and they are classified by the types and properties of functions used. Two sorts of tape-computers with suitable modifications of addresses are presented which simulate the activity of Turing machines.

1. ADDRESSED TURING MACHINES

A definition of Turing machine has two parts: the first concerns the syntax and the second the semantics, i.e. its activity.

With respect to the syntax (according to e.g. M. DAVIS [3]) a Turing machine is determined by a finite set Z of quadruples which have one of the following three forms

(1) (q, a, q^*, a^*),

(2) (q, a, q^*, R),

(3) (q, a, q^*, L)

and which satisfy the following requirement

(4) no two different quadruples from Z have the same first and second member,

where $q, q^* \in Q$; $a, a^* \in A$; $A \cap \{R, L\} = \emptyset$ and $|Q| = n$. The elements of Q, A are called *inner states*, *basic symbols* respectively. There is one inner state $q_1 \in Q$ distinguished and called *initial* and one basic symbol $a_0 \in A$ called *blank space.*

The activity of Turing machine concerns a two-way-infinite tape divided into squares on which certain basic symbols are printed. Assuming well known determination of the activity in [3] a little modified way is used here. Let $N = \{\ldots, -2,$

$-1, 0, +1, +2, \ldots\}$ be the set of all positive and negative integers, inclusively zero, which are assigned to the individual squares of the tape as their coordinates. The numbers from N may be considered as symbolic addresses of the corresponding squares-memory cells, because the tape plays a role of storage in any case. This is the reason why we are speaking about an *addressed Turing machine.*

Let TD be the set of all *tape descriptions* which are functions t such that

(5) Domain $t = N$, Range $t \subset A$ and there is only a finite number of integers $x \in N$ satisfying the inequality $t(x) \neq a_0$.

With respect to (5) let N_t be the shortest interval of N such that if $x \in N - N_t$ then $t(x) = a_0$. Then $t|_{N_t}$ is called *finite tape description.*

In virtue of (4), the following two binary functions may be determined for the given set Z (by the enumeration of the corresponding triples):

(6) $\varphi = \{((q, a); q^*)$; there exists a quadruple in Z such that q, a, q^* is its first, second, third member respectively$\}$,

and

(7) $\psi = \{((q, a); a^*)$; there exists a quadruple in Z of the form (1) such that q, a, a^* is its first, second, fourth member respectively$\}$.

Now each instantaneous description (see [3]) of the Turing machine Z is determined by a triple $[t, x, q]$ where $t \in TD$, $x \in N$ and $q \in Q$, because by x is determined which square is scanned and by $t(x)$ which symbol is printed in the square scanned. The next instantaneous description $[t^*, x^*, q^*]$ is defined recurrently as follows:

a) if $q, t(x)$ is the first, second member respectively, in a quadruple of T which has the form (p) where $1 \leqq p \leqq 3$, then the condition (p^*) holds, where

(1*) $$t^*(i) = t(i) \text{ for each } i \neq x \text{ where } i = 0, \pm 1, \pm 2, \ldots, \text{ and}$$
$$t^*(x) = \psi(q, t(x)); \quad x^* = x \text{ and } q^* = \varphi(q, t(x));$$

(2*) $$t^* = t; \quad x^* = x + 1 \quad \text{and} \quad q^* = \varphi(q, t(x));$$

(3*) $$t^* = t; \quad x^* = x - 1 \quad \text{and} \quad q^* = \varphi(q, t(x));$$

b) if there does not exist a quadruple of Z such that $q, t(x)$ is its first, second member respectively, then the activity is stopped and t is called *final tape description*;

c) the initial instantaneous description $[t, x, q]$ satisfies $q = q_1$.

The machine Z computes the string function $F_Z = \{(t_0|_{N_{t_0}}, t|_{N_t})$; t_0 is an initial instantaneous description and t is the corresponding final state description$\}$ because, obviously, by $t_0|_{N_{t_0}}$ and $t|_{N_t}$ a pair of finite strings over A is uniquely determined.

If a number function should be computed by a Turing machine Z then all the required numbers must be expressed in well known way by repeating one distinguished symbol, i.e. certain coding and decoding is assumed such that one number may

occupy many neighbouring squares, or, one number is stored at many addresses simultaneously. If also negative integers, rational numbers and r-tuples of such numbers are required further coding and decoding conventions must be added.

2. COMPUTERS AND THEIR PROGRAMS

Using sligthly modified and simplified definitions and notations of [1] and [2], a computer may be characterized as follows: $\mathbf{Cptr} = \langle \mathbf{Obj}, \mathbf{Adr}, \mathbf{Fct} \rangle$, where $\mathbf{Obj}$ is a set of *basic objects* the computer is dealing with, $\mathbf{Adr}$ is a set of *basic addresses* (or names, or identifiers etc.) and $\mathbf{Fct}$ is a set of *basic functions* such that if $f \in \mathbf{Fct}$ then $\text{Domain}\, f \subset \mathbf{Obj}^n \times \mathbf{Adr}^m$ for certain integers $0 \leqq n, m$, and $\text{Range}\, f \subset \mathbf{Obj} \cup \mathbf{Adr}$. If there exists a function $f \in \mathbf{Fct}$ with $n \geqq 1$ and $m \geqq 1$ then the computer is called of a *mixed type*, otherwise, i.e. if always either $n = 0$ or $m = 0$, of a *pure type*. If $f \in \mathbf{Fct}$ then there are the following four possibilities in computers of a pure type:

(8) $\text{Domain}\, f \subset \mathbf{Obj}^n$, $n \geqq 1$, and $\text{Range}\, f \subset \mathbf{Obj}$ (then f is called *operation*);

(9) $\text{Domain}\, f \subset \mathbf{Obj}^n$, $n \geqq 1$, and $\text{Range}\, f \subset \mathbf{Adr}$ (then f is called *condition*);

(10) $\text{Domain}\, f \subset \mathbf{Adr}^m$, $m \geqq 1$, and $\text{Range}\, f \subset \mathbf{Adr}$ (then f is called *address modification*);

and finally

(11) $\text{Domain}\, f \subset \mathbf{Adr}^m$, $m \geqq 1$, and $\text{Range}\, f \subset \mathbf{Obj}$.

In [1] the computer is said to be simple if $\mathbf{Fct} = \mathbf{Opr}$ where $\mathbf{Opr}$ is the set of all operations, and it is called conditional if $\mathbf{Fct} = \mathbf{Opr} \cup \mathbf{Cond}$ where $\mathbf{Cond} \neq \emptyset$ and $\mathbf{Cond}$ is the set of all conditions. Here the conditional computers with address modifications (i.e. if $\mathbf{Fct} = \mathbf{Opr} \cup \mathbf{Cond} \cup \mathbf{Mod}$, where $\mathbf{Mod} \neq \emptyset$ and $\mathbf{Mod}$ is the set of all address modifications) will be considered.

Further the following derived concepts must be added to the characterization of a computer:

$\mathbf{Com}$ is the set of all commands which are strings of symbols of one of the following forms:

(12) $f(x_1, x_2, \ldots, x_n) =: x_0$, where "$f$" $\in \mathbf{SymbOpr}$ and $x_i \in \mathbf{Adr}$ for $i = 0, 1, \ldots, n$;

(13) $x =: y$, where $x, y \in \mathbf{Adr}$;

(14) **Stop**;

(15) $g(x_1, x_2, \ldots, x_n)$, where "g" $\in \mathbf{SymbCond}$ and $x_i \in \mathbf{Adr}$ for $i = 1, 2, \ldots, n$;

(16) x, where $x \in \mathbf{Adr}$;

(17) $h(x_1, x_2, \ldots, x_n) =: x_0$, where "$h$" $\in$ ***SymbMod*** and $x_i \in$ ***Adr*** for $i = 0, 1, \ldots, n$;

(18_1) $\sigma(x) =: y$, where $x, y \in$ ***Adr*** and σ is a new symbol,

(18_2) $y =: \sigma(x)$, where $x, y \in$ ***Adr***, σ is a new symbol,

where ***SymbOpr***, ***SymbCond***, ***SymbMod*** is the set of names of all elements in ***Opr***, ***Cond***, ***Mod*** respectively such that always there is a one-to-one correspondence between names and functions, and "=:" is usual assignement symbol.

In simple computers only commands of forms (12)–(14) are required, in conditional computers the commands of the form (15) and (16) are added (which are conditional and unconditional jumps respectively), and if some modifications of addresses are admitted also the commands of forms (17) and (18) are necessary.

Further

Sta $= \{\sigma; \sigma$ is a function such that Domain $\sigma =$ ***Adr*** and Range $\sigma \subset$ ***Obj*** $\cup$ $\cup$ ***Adr*** $\cup$ ***Com***$\}$ is the set of all states of storage (in simple and conditional computers it is possible to restrict the states σ to a special case when Range $\sigma \subset$ ***Obj***);

AdrCom is the set of strings called addressed commands, which are couples $\langle a; C\rangle$, where $a \in$ ***Adr*** and $C \in$ ***Com*** (such that the address "a" does not occur in the command "C"; for this reason in [1] and [2] a special set of labels or markers is introduced by which the command are labelled and which are the only values of conditions);

Prog is the set of all programs which are finite sequences P of the form $P = (K^{(1)}, K^{(2)}, \ldots, K^{(p)})$, where $K^{(i)} \in$ ***Com*** $\cup$ ***AdrCom*** for each $i = 1, 2, \ldots, p$.

The activity of the computer ***Cptr*** under consideration for the program P and for an initial state of storage $\sigma_0 \in$ ***Sta*** consists in an iterative application of commands (or addressed commands) occurring in P to the current state of storage in the order from the left to the right unless by conditional commands (15) and (16) a new address, and therefore a new addressed command, is determined. It is sufficient to define the next state of storage $\sigma_i = C_i\sigma_{i-1}$ for the current state σ_i to which the command C_i is applied (or executed), and the next command C_{i+1} where $i > 0$. The cases (12)–(18) must be distinguished:

(12*) $C_i = (f(x_1, x_2, \ldots, x_n) =: x_0)$ and C_i is contained in $K^{(j)}$, where $1 \leqq j < p$ (if $j = p$ then after application of C_i the activity is finished, because no C_{i+1} is determined); if $(\sigma_{i-1}(x_1), \sigma_{i-1}(x_2), \ldots, \sigma_{i-1}(x_n)) \in$ Domain f (otherwise the activity is finished) then $\sigma_i(z) = {}_{\mathrm{df}}\sigma_{i-1}(z)$ for each $z \in$ ***Adr*** $- \{x_0\}$ and $\sigma_i(x_0) = {}_{\mathrm{df}}f(\sigma_{i-1}(x_1), \sigma_{i-1}(x_2), \ldots, \sigma_{i-1}(x_n))$, and C_{i+1} is that command contained in $K^{(j+1)}$;

(13*) $C_i = (x =: y)$ and C_i is contained in $K^{(j)}$, where $1 \leqq j < p$ (if $j = p$ then after application of C_i the activity is finished); then $\sigma_i(z) = {}_{df}\sigma_{i-1}(z)$ for each $z \in \textbf{\textit{Adr}} - \{y\}$ and $\sigma_i(y) = {}_{df}\sigma_{i-1}(x)$, and C_{i+1} is that command contained in $K^{(j+1)}$;

(14*) $C_i = \textbf{\textit{Stop}}$; then the activity is finished and stopped and the state σ_{i-1} is called final or resulting state of storage;

(15*) $C_i = (g(x_1, x_2, \ldots, x_n))$; if $(\sigma_{i-1}(x_1), \sigma_{i-1}(x_2), \ldots, \sigma_{i-1}(x_n)) \in$ Domain g and if there exists $K^{(j)}$, $1 \leqq j \leqq p$, which contains the address $g(\sigma_{i-1}(x_1), \ldots$ $\ldots, \sigma_{i-1}(x_n))$ (otherwise the activity is finished) then C_{i+1} is that command contained in $K^{(j)}$, and $\sigma_i = \sigma_{i-1}$;

(16*) $C_i = (x)$; if there exists $K^{(j)}$, $1 \leqq j \leqq p$, which contains the address x (otherwise the activity is finished), then C_{i+1} is that command contained in $K^{(j)}$, and $\sigma_i = \sigma_{i-1}$;

(17*) arises from (12*) by replacement of "f" by "h";

(18_1^*) $C_i = (\sigma(x) =: y)$ and C_i is contained in $K^{(j)}$, where $1 \leqq j < p$ (if $j = p$ then after application of C_i the activity is finished); if $\sigma_{i-1}(x) \in \textbf{\textit{Adr}}$ (otherwise the activity is finished) then $\sigma_i(z) = {}_{df}\sigma_{i-1}(z)$ for each $z \in \textbf{\textit{Adr}} - \{y\}$ and $\sigma_i(y) = {}_{df}\sigma_{i-1}(\sigma_{i-1}(x)) = \sigma_{i-1}^2(x)$, and C_{i+1} is that command which is contained in $K^{(j+1)}$;

(18_2^*) $C_i = (y =: \sigma(x))$ and C_i is contained in $K^{(j)}$, where $1 \leqq j < p$ (if $j = p$ then after the application of C_i the activity is finished); if $\sigma_{i-1}(x) \in \textbf{\textit{Adr}}$ (otherwise the activity is finished) then $\sigma_i(z) = {}_{df}\sigma_{i-1}(z)$ for each $z \in \textbf{\textit{Adr}} -$ $- \{\sigma_{i-1}(x)\}$ and $\sigma_i(\sigma_{i-1}(x)) = {}_{df}\sigma_{i-1}(y)$, and C_{i+1} is that command contained in the $K^{(j+1)}$.

Although the commands of the form (18) are sufficient for our simulation of all Turing machines it should be noted that they may be generalized in a natural way to the form

(19) $\sigma^k(x) =: y$, where $x, y \in \textbf{\textit{Adr}}$, σ is a new symbol as in (18_1) and $k \geqq 2$ is an arbitrary integer.

It is conjectured that this general case is close to the "**ref**"-mechanism in *ALGOL68* [5].

The computer **Cptr** computes the state function $F_{\textbf{Cptr},P} = \{(\sigma_0; \sigma_i)$; σ_i is the final state of storage of **Cptr** which corresponds to the initial state σ_0 in accordance with the program $P\}$, because it assigns states of storage again to states of storage.

Usually there are prescribed input and output addresses to each program P, e.g. $I_P = \{x_1, x_2, \ldots, x_r\} \subset \textbf{\textit{Adr}}$ and $O_P = \{y_1, y_2, \ldots, y_s\} \subset \textbf{\textit{Adr}}$ respectively, and therefore another function $f_{\textbf{Cptr},P} = \{(\sigma_0|_{I_P}; \sigma_i|_{O_P})$; $(\sigma_0; \sigma_i) \in F_{\textbf{Cptr},P}\}$ is called

computable in **Cptr** *by* P, under the condition that Range $\sigma_0|_{I_P} \subset$ **Obj** and Range $\sigma_i|_{O_P} \subset$ **Obj**, which means that s r-ary functions (=operations) in **Obj** are computed (if, e.g. the order of input and output addresses is fixed).

Thus the crucial theoretical question is to decide whether or not an arbitrary function f^* such that Domain $f^* \subset$ **Obj**^{r} and Range $f^* \subset$ **Obj** (i.e. if $s = 1$) is computable in the prescribed **Cptr**, i.e. whether or not there exists a program P such that $f^* = f_{\mathbf{Cptr},P}$, and if the answer is positive, to construct the required program P. In fact, always the function f^* under consideration must be determined by a "program" or by an "algorithm" using some other functions assumed as known and computable, and therefore the crucial practical question is to "translate" the given "program" for one "computer" into program for the second computer.

3. DIFFERENCES BETWEEN TURING MACHINES AND COMPUTERS

(i) A computer has finite storage represented by the set **Adr** but an addressed Turing machine has an infinite storage represented by the set N.

(ii) The storage of a computer has no structure, i.e. all addresses are equivalent each to other because they are all available in any instant (the differences between the registers and other memory cells of main storage of real computers and differences between different sorts of storage as magnetic tape, drum, digs, etc. can but need not be taken in account here), but in each Turing machine its memory has tape structure where in next instant only two neighbouring addresses, and the current address itself, are available.

(iii) The basic objects which are stored at the addresses in computers may be essentially more complex than the basic symbols stored at the addresses of a tape in Turing machines, where also basic objects are stored at many neighbouring addresses. Here is a deep difference in the concept of address.

(iv) In computers it is possible to admit an infinite number of basic objects without any change of the programs, but it has different meaning to admit infinite number of basic symbols in Turing machines.

(v) There is distinguished the computer from its programs, but in Turing machine both these concepts are mixed up in a set of quadruples Z.

In other words, in computers the determination of basic functions (which belong to hard-ware) is separated from that of programs (which belong to soft-ware), but in Turing machines both these parts are mixed up.

(vi) The string functions F_Z and the state functions $F_{\mathbf{Cptr},P}$ or $f_{\mathbf{Cptr},P}$ cannot be compared in any reasonable way because of (iii), i.e. F_Z concerns the sequences of addresses of arbitrary lengths but $f_{\mathbf{Cptr},P}$ concerns fixed sets of addresses.

(vii) In computers the basic operations and conditions may be arbitrary functions of many variables (in real computers they are binary functions usually), i.e. the

objects stored at many different addresses must be used simultaneously, but in Turing machines only unary operations are used, because only one square is scanned.

It follows by (vii) that it is impossible to replace a computer by one single Turing machine; it would be necessary to have several Turing machines together with their composition, or to have a universal Turing machine.

By (iii) it is also impossible to replace a Turing machine by one computer, unless a specialized storage is required, i.e. which have the same tape structure as that of Turing machines has.

Therefore in the following sections certain specialized tape-computers are considered. Then $F_{\mathbf{Cptr},P}$ will be a string function and it may be compared with F_Z for the Turing machine Z.

4. THE FIRST TAPE-COMPUTER AND ITS SIMULATING PROGRAM

Now to an arbitrary addressed Turing machine Z with the tape N the tape-computer $\mathbf{Cptr}_1 = \langle \mathbf{Obj}_1, \mathbf{Adr}_1, \mathbf{Fct}_1 \rangle$ of mixed type is constructed in the following way.

First of all in accordance with Sect. 1 let us introduce the following unary relation in 4-valued logic (i.e. a decomposition of a set into 4 subsets) with values m_1, m_2, m_3, $m_4 \in \mathbf{Adr}$:

(20) $\varrho =_{\mathrm{df}} \{((q, a); m_p)$; there exists a quadruple in Z of the form (p), where $1 \leqq p \leqq 3$, such that q, a is its first, second member respectively$\} \cup \{((q, a); m_4)$; there does not exist a quadruple in Z such that q, a is its first, second member respectively$\}$.

In other words ϱ is a function such that Domain $\varrho = Q \times A$, Range $\varrho \subset \{m_1, m_2, m_3, m_4\}$ and such that a decomposition of $Q \times A$ in at most four classes is determined. Three of these classes correspond to the cases (1), (2) and (3) mentioned in Sect. 1, and the fourth class contains pairs (q, a) for which neither φ nor ψ is defined.

Thus we define: $\mathbf{Obj}_1 = A \cup Q$; $\mathbf{Adr}_1 = N \cup \{r_N, r'_N, r_Q, r'_Q\} \cup \{m_0, m_1, m_2, m_3, m_4\}$, i.e. to the tape-store N some auxiliary addresses – e.g. registers – are added; $\mathbf{Fct}_1 = \{\varphi, \psi, \varrho, +1, -1\}$, where "+1" and "−1" means the addition of plus one and minus one defined in N respectively; $\mathbf{Sta}_1 = \{\sigma$; there exists $t \in TD$ such that $\sigma(x) = t(x)$ for each $x \in N$, $\sigma(r_N) \in N$, $\sigma(r'_N) \in A$, $\sigma(r_Q) \in Q$, $\sigma(r'_Q) \in Q$ and $\sigma(m_p) \in \in \mathbf{Com}$ for $p = 0, 1, 2, 3, 4\}$; the initial state σ_0 satisfies $\sigma_0(r_Q) = q_1$. Finally let us take the following program:

$$
\begin{aligned}
P_1 = (&\langle m_0;\ \sigma(r_N) =: r'_N\rangle,\ \ \varrho(r_Q, r'_N),\\
&\langle m_1;\ \varphi(r_Q, r'_N) =: r'_Q\rangle,\ \ \psi(r_Q, r'_N) =: r'_N,\ \ r'_Q =: r_Q,\ \ m_0,\\
&\langle m_2;\ \varphi(r_Q, r'_N) =: r_Q\rangle,\ \ r_N + 1 =: r_N,\ \ m_0,\\
&\langle m_3;\ \varphi(r_Q, r'_N) =: r_Q\rangle,\ \ r_N - 1 =: r_N,\ \ m_0,\\
&\langle m_4;\ \mathbf{Stop}\rangle).
\end{aligned}
$$

Lemma 1. *The tape-computer* $\mathbf{Cptr}_1$ *by the program* P_1 *simulates the activity of the Turing machine* Z *and therefore* $F_{\mathbf{Cptr}_1, P_1} = F_Z$.

Proof. If $[t, x, q]$ is an initial instantaneous description of the addressed Turing machine Z, i.e. $q = q_1$, then as the corresponding initial state σ_0 for $\mathbf{Cptr}_1$ the following one must be chosen: $\sigma_0(r_N) = x$ and $\sigma_0(y) = t(y)$ for $y \in N$. Further the simulation is clear step by step.

The unsufficiency of this tape-computer consists in the fact, that we are interested in a string function F_Z such that the strings on N consist only of the basic symbols from A and the inner states from Q are not addmitted. Therefore an other tape-computer will be introduced.

5. THE SECOND TAPE-COMPUTER AND ITS SIMULATING PROGRAM

The tape-computer $\mathbf{Cptr}_2 = \langle \mathbf{Obj}_2, \mathbf{Adr}_2, \mathbf{Fct}_2 \rangle$ differ from $\mathbf{Cptr}_1$ by considering of Q as a subset of the set of addresses, by which follows the necessity to modify the set of functions as follows:

(6*) $g_q = \{(a; q^*)$; there exists a quadruple in Z such that q, a, q^* is its first, second, third member respectively$\}$ for each $q \in Q$;

(7*) $f_q = \{(a; a^*)$; there exists a quadruple in Z of the form (1) such that q, a, a^* is its first, second, fourth member respectively$\}$ for each $q \in Q$;

(20*) $h_q = \{(a; m_q^{(p)})$; there exists a quadruple in Z of the form (p), where $1 \leqq p \leqq 3$, such that q, a is its first, second member respectively$\} \cup \{(a; m_q^{(4)})$; there does not exist a quadruple in Z such that q, a is its first, second member respectively$\}$ for each $q \in Q$.

It is important to mention explicitely, that the functions g_q, f_q, h_q for $q \in Q$ arised by suitable partialization of the function φ, ψ, ϱ respectively.

Let Q_g, Q_f, Q_h be the set of all states $q \in Q$ such that $g_q \neq \emptyset$, $f_q \neq \emptyset$, $h_q \neq \emptyset$ is valid respectively. Therefore $Q_g \cap Q_h = \emptyset$ and $Q_f \subset Q_g$.

Thus we define: $\mathbf{Obj}_2 = A$; $\mathbf{Adr}_2 = N \cup Q \cup \{r_N, r'_N\} \cup \{m_q^{(p)};\ q \in Q$ a $p = 1, 2, 3, 4\}$; $\mathbf{Fct}_2 = \mathbf{Opr}_2 \cup \mathbf{Cond}_2 \cup \mathbf{Mod}_2$, where $\mathbf{Opr}_2 = \{f_q; q \in Q_f\}$, $\mathbf{Cond}_2 = \{g_q;\ q \in Q_g\} \cup \{h_q;\ q \in Q_h\}$ and $\mathbf{Mod}_2 = \{+1, -1\}$; $\mathbf{Sta}_2 = \{\sigma$; there exists $t \in TD$ such that $\sigma(x) = t(x)$ for each $x \in N$, $\sigma(r_N) \in N$, $\sigma(r'_N) \in N$, $\sigma(q) \in \mathbf{Com}$ for each $q \in Q$ and $\sigma(m_q^{(p)}) \in \mathbf{Com}$ for each $q \in Q$ and $p = 1, 2, 3, 4\}$; the initial state σ_0 satisfies $\sigma_0(x) = t$.

Finally let us take the following program, if $Q = \{q_1, q_2, \ldots, q_n\}$:

$$
\begin{aligned}
P_2 = (&\langle q_1;\ \sigma(r_N) =: r'_N\rangle,\quad h_{q_1}(r'_N),\\
&\langle m^{(1)}_{q_1};\ f_{q_1}(r'_N) =: \sigma(r_N)\rangle,\quad g_{q_1}(r'_N),\\
&\langle m^{(2)}_{q_1};\ r_N + 1 =: r_N\rangle,\quad g_{q_1}(r'_N),\\
&\langle m^{(3)}_{q_1};\ r_N - 1 =: r_N\rangle,\quad g_{q_1}(r'_N),\\
&\langle m^{(4)}_{q_1};\ \textbf{STOP}\rangle,\\
&\langle q_2;\ \sigma(r_N) =: r'_N\rangle,\ h_{q_2}(r'_N),\\
&\langle m^{(1)}_{q_2};\ f_{q_2}(r'_N) =: \sigma(r_N)\rangle,\quad g_{q_2}(r'_N),\\
&\quad\vdots\\
&\langle m^{(4)}_{q_{n-1}};\ \textbf{STOP}\rangle,\\
&\langle q_n;\ \sigma(r_N) =: r'_N\rangle,\quad h_{q_n}(r'_N),\\
&\langle m^{(1)}_{q_n};\ f_{q_n}(r'_N) =: \sigma(r_N)\rangle,\quad g_{q_n}(r'_N),\\
&\langle m^{(2)}_{q_n};\ r_N + 1 =: r_N\rangle,\quad g_{q_n}(r'_N),\\
&\langle m^{(3)}_{q_n};\ r_N - 1 =: r_N\rangle,\quad g_{q_n}(r'_N),\\
&\langle m^{(4)}_{q_n};\ \textbf{STOP}\rangle),
\end{aligned}
$$

where each group of commands starting with $\langle q_i;\ \sigma(r_N) =: r'_N\rangle$ is superfluous if $q_i \notin Q_h$.

Lemma 2. *The tape-computer* $\mathbf{Cptr}_2$ *by the program* P_2 *simulates the activity of the Turing machine* Z *and therefore* $F_{\mathbf{Cptr}_2, P_2} = F_Z$.

Proof. If $[t, x, q_1]$ is an initial instantaneous description of the addressed Turing machine Z, then the corresponding initial state σ_0 for the $\mathbf{Cptr}_2$ must be chosen as follows: $\sigma_0(r_N) = x$ and $\sigma_0(y) = t(y)$ for each $y \in N$. Further if, e.g. the quadruple (q_1, a_1, q_2, a_2) of the type (1) is applied to $[t, x, q_1]$, where $t(x) = a_1$, then by (1*) the next instantaneous description $[t^*, x^*, q_1^*]$ satisfies: $t^*(i) = t(i)$ for $i \in N - \{x\}$ and $t^*(x) = \psi(q_1, a_1) = a_2$ (which follows by (7)); $x^* = x$ and $q_1^* = \varphi(q_1, a_1) = q_2$ (which follows by (6)). On the other hand according to the P_2 the first command $\sigma(r_N) =: r'_N$ must be applied, where $\sigma = \sigma_0$ (as the current state at the beginning is the initial state σ_0) and therefore by (18_1^*) one gets $\sigma_1(z) = {}_{\mathrm{df}}\sigma_0(z)$ for each $z \in \mathbf{Adr} - \{r'_N\}$ and $\sigma_1(r'_N) = {}_{\mathrm{df}}\sigma_0^2(r_N) = a_1$. Further the next conditional command $h_{q_1}(r'_N)$ is applied and therefore by (15*) $\sigma_2 = \sigma_1$ and by (20*) $h_{q_1}(a_1) = m^{(1)}_{q_1}$. Thus the next command to be executed is $f_{q_1}(r'_N) =: \sigma(r_N)$, where by (7*) $f_{q_1}(a_1) = a_2$ and $\sigma_2(r_N) = \sigma_1(r_N) = \sigma_0(r_N) = x$. Therefore by (12*) $\sigma_3(z) = \sigma_2(z)$ for each $z \in \mathbf{Adr} - \{x\}$ and $\sigma_3(x) = {}_{\mathrm{df}}f_{q_1}(\sigma_2(r'_N)) = a_1$. Now the next command to be executed is $g_{q_1}(r'_N)$, where according to (6*), $g_{q_1}(a_1) = q_2$, and by (15*) $\sigma_4 = \sigma_3$, which means that as the next command will be executed that one addressed by "q_2". Moreover it is

clear that $\sigma_4(i) = t^*(i)$ for each $i \in N$ and $\sigma_4(r_N) = x = x^*$ again. Taking all other possibilities one establishes the required simulation correspondence step by step.

It follows by the lemmas:

Theorem. *Each function computable by a Turing machine is computable by a tape-computer of the type* $\mathbf{Cptr}_1$ *and also of the type* $\mathbf{Cptr}_2$.

The reason for giving this theorem (and both preceeding lemmas also) is to clarify deep differences between computers and Turing machines. It is shown by them that for the simulation purpose of Turing machines the computers must be provided not only by an infinite memory but moreover by a tape-structured infinite memory which requires two infinite functions = address modifications "+1" and "−1". It seems to be highly unconstructivistic and, of course not realizable, to allow any infinite function in the base of a computer itself. Moreover both tape-computers show explicitely that during the computation several functions $\sigma \in \mathbf{Sta}$ must be used, although not explicitly, which probably may have unpredictible properties. This is also no support for the strict constructivistic point of view.

6. CONCLUSIONS

In order to underline the differences between Turing machines and computers it shoulded be remind a note concerning certain classifications of computers in Sect. 2. In general it is unclear how important role is played by the commands of the types (18_1) and (18_2). In any case there is a conjecture that these types of commands are necessary in each complete simulation of Turing machines and therefore that they represent a special tool in constructing of functions. These commands are not expressible using the usual flow-diagrams and therefore it may be conjectured that the functions computed by Turing machines cannot be simulated by flow-diagrams only.

On the other hand it remains open to extend the above mentioned simulation to the universal Turing machine too.

With respect to a classification concerning the functions required by the computer it is clear, that there are many Turing machines simulated by just one-tape-computer using many different programs. The value of these classifications remains unclear because it is easy to provide each tape-computer by all possible unary operations which may be defined in the set A (if A is finite).

The fact that in $\mathbf{Cptr}_2$ only unary operations (and in fact also only unary conditions and modifications) are required, shows that within all the frame of Turing machines or of mentioned tape-computers some important tools are included, or are added by further conventions, if the functions of unary variables should be evaluated.

Moreover in [4] even a more extremal case occurs if all the operations are constant functions, i.e. functions without any variable.

For all the mentioned differences between Turing machines and computers a strong feeling must arise that the recent computer problems cannot be solved using the concepts concerning Turing machines but that the new direct concepts of computers are necessary to introduce and investigate.

References

[1] *Čulík, K.* and *M. A. Arbib*, Sequential and Jumping Machines and their relation to computers, Acta Informatica *2* (1973), 162—171.

[2] *Čulík, K.*, Structural similarity of programs and some concepts of algorithmic method, Lecture Notes in Economics and Mathematical Systems *75*, Springer 1972.

[3] *Davis, M.*, Computability and Unsolvability, McGraw-Hill, N.Y. 1958.

[4] *Wagner, E. G.*, On the structure of programming languages, or, six languages for Turing Machines, 45—53, IEEE conference record of 1967 eight annual symposium on switching and automata theory.

[5] *Wijngaarden, A. van, Mailloux, B. J., Peck, J. E. L., Koster, C. H. A.*, ALGOL 68, Math. Centrum, Amsterdam 1968.

Author's address: 602 00 Brno, Čápkova 31.