DOI: 10.17516/1997-1397-2021-14-1-69-73
УДК 512.54

A Note on Computation MTs with Time in Instructions or with Tapes of Fixed Length

Vladimir V. Rybakov*
Siberian Federal University
Krasnoyarsk, Russian Federation
A. P. Ershov Institute of Informatics Systems
Novosibirsk, Russian Federation

Received 18.09.2020, received in revised form 23.11.2020, accepted 26.12.2020

Abstract. In this short note we analyze the computation algorithms modelled by Church-Turing-Post machines with algorithms for computation which use amount of time spent for computation (number of steps) in their own definitions. We notice some difference and illustrate that there are distinctions in behaviour of such algorithms; also we consider working of MTs on tapes of fixed length and observe again noticed difference.

Keywords: computations, universal Church-Turing Machines, time of computation.

Citation: V.V. Rybakov, A Note on Computation MTs with Time in Instructions or with Tapes of Fixed Length, J. Sib. Fed. Univ. Math. Phys., 2021, 14(1), 69–73. DOI: 10.17516/1997-1397-2021-14-1-69-73.

1. A small recall

We use here standard well known definitions and results from logic in computer science and computability theory (cf. [1–4]). A Church-Post-Turing machine consists of (i) a finite tape (string) divided into cells, one next to the other. Each cell contains a symbol from some finite alphabet (we use here 1, 0, L, R — left end, right end); (ii) a head that can read and write symbols on the tape and move on the tape left and right one (and only one) cell at a time; (iii) a state register that stores the state of the machine, among these is the special start state with which the state register is initialized and the stop state – receiving the which the machine stops.

As usual a program for Church-Post-Turing machine (MT is sequel) is a finite collection of instructions of sort $q_j a_i \Rightarrow q_m b_i(LRS)$ and the computation by MT is the modification of the tape (string) in accordance with this instructions.

We will consider a modification of this sort MTs by only one more feature — taking to account the amount of steps, that is the amount of the applied instructions

$$q_j a_i \Rightarrow q_m b_i(LRS),$$

during computation on the tape of MT, during transformation by MT its tape (string).

In this note we will consider computation by MT as the procedure of transformation its tape. The algorithms (informal), thus, we will mean as any informal (and formal) instructions for transformation tapes filled in with 0 and 1. So, algorithms works with finite strings (tapes) with cells filled in with 0 or 1 and with symbols L at the beginning and R in the end — denoting

*Vladimir_Rybakov@mail.ru

the beginning of the string and the end of the string respectively (so the strings look as typical input-output tape of the Turing machine). Any algorithm starts on a tape, and, if terminates during run, we call the final state of the tape to be output this algorithm. Standard formal Turing Machines work exactly this way. For example, the standard MT computing a function f, starting on the tape filled in with n 1 terminates on its run in sees output with $f(n)$ 1 if $f(n)$ is defined.

2. What we do

Our considerations will use sequence of programs $MT_n, n \in N$ for MTs which contain all possible (finite) MT-programs. This sequence to be generated by an algorithm GM writing out these programs in the sequence (that algorithm may, for example, be viewed as Monkey–Typing on Typewriter with only care not to repeat the same $MT_n s$; so programs may be with wrong structure, not working correctly, not working at all, etc. So, for just modelling we do not care now about efficiency, etc.)

That algorithm for generating $M_n, n \in N$ may be any one chosen, for example the best (if possible?). But as soon as it is chosen we fix it and work longer with only such one, and any program M_n (sequence of its instructions) to be computed by that GM generating algorithm. So, we may admit that we have a file FGM written in a computer hard drive, where any $n-th$ record is a MT-program, program MT_n, $FGM(n) := MT_n$. We may consider this FGM as a realization of universal Turing machine. In computer science, universal Turing machine (UTM) is a Turing machine that simulates an arbitrary Turing machine on arbitrary input (cf. [7]). We do not restrict ourselves here with consideration just universal Turing functions — algorithms are more general substance.

We consider now an algorithm $A_n, n \in N$, any A_n is based on the MT_n the program MT_n. A_n starts and creates finite strings (tapes) with cells filled in with 0 or 1 and with symbols L at the beginning and R in the end

Definition 1. *Given with any natural number m represented as the string S_m starting with L next filled with m symbols* 1 *and concluded with R the algorithm A_n does the following.*

It launches the program MT_n on S_m,

the STEP in sequel means the application of some instruction

of the program from MT_n and

(i) If MT_n stops on a step before the number of steps exceeds $10^6 \times m \times n$, stops correctly being on stopping internal state and looking on a cell containing a_i, it changes the value a_i to the opposite (1 to 0, 0 to 1) and again stops;

(ii) If MT_n stops on a step before the number of steps exceeds $10^6 \times m \times n$ but stays not in a stopping configuration (does not get stopping internal state, that is it does not have instruction how to continue, got suspended) we simply stop and as earlier above change the value of the cell observed by the reading head to the opposite and stop.

(iii) If MT_n does not stops on a step before the number of steps exceeds $10^6 \times m \times n$ we change the value of the cell observed by the reading head on the step $10^6 \times m \times n$ to the opposite and stop (if it is R we move one cell left and if it is L we move one cell right).

Lemma 1. *The procedure A_n is a computable algorism transforming any given string S_m in a finite string $A_n(S_m)$, this algorithm always terminates after at most $10^6 \times m \times n + 2$ steps of M_n and the output string $A_n(S_m)$ has length at most $m + 10^3 \times m \times n + 4$.*

Proof is evident.

Definition 2. *Consider now the algorithm Alg which transform any given string S_m in $A_m(S_m)$. That procedure again is an algorithm, which terminates at any given input S_m.*

Here there is a small trick — in fact we use potentially infinite number of Church-Turing programs $MT_m, m \in M$. But notice that an algorithm after using it does not cease to be an algorithm

Lemma 2. *There is no Turing machine MT_n such that MT_n transforms any input string S_m as Alg does. That is for any input S_m, starting both on the string S_m they would produce identical output tapes on any step of computation before stop (of applying machine instructions from MT_n and Alg (A_m) respectively) and stop after the same amount of applying such instructions.*

Proof. Assume the opposite, and take such MT_n which works on all strings S_m as Alg does. Launch Alg and MT_n on S_n. Then by definition of Alg we launch A_n on S_n and by definition of A_n we have that some of (i) or (ii) from the definition of A_n above holds, and then the outputs MT_n and Alg will be different or else (iii) holds and then MT_n will take more steps (amount of applications of the machines instructions) comparing with Alg; more precisely - they - steps - instructions — exactly the same for both before Alg stops. But any algorithm does not may stop and not stop simultaneously, and MT_n continues to work. □

Definition 3. *We fix now what we mean by* **modelling** *by a Turing Machine MT an Algorithm B transforming the TM-strings. We say so if starting on any string (only after stating to make transformations on strings), at any step of strings-transformation, produce identical output strings before stop or B stops but MT continue to work.*

Notice that before starting to work on a string the algorithm B may do any preparation job before touching the string (do any necessary magic, as we do above by choosing A_n).

Theorem 1. *There is no Turing machine MT modelling computation of the algorithm Alg on numbers S_n. That is such MT that, for any string S_n (number n), after starting on the string S_n, MT would produce output strings (at any its step) the same as the algorithm Alg starting at S_n does and they would stop simultaneously (at the same number of steps) (Alg always stops.).*

Proof. Assume not and such MT exists. Then for some MT_n programs in MT_n and MT are exactly the same. Then Lemma 2 gives a contradiction. □ □

In general this looks enough curious. Alg cannot be modelled by Turing machines in precise. So, whichever machine MT to be taken — it either gives different with Alg outputs on stops, or else starting to work at a string S_n, Alg stops at a step but MT produces the same sequences of output strings until the final one for Alg and yet compulsory continue to work.

Remark. Now on we would like briefly comment how this approach might be viewed in the light of computable functions and Church-Turing thesis. At first glance all the same all holds (We considered above algorithms of transformation strings with numbers as more general case to get more general results). But else there are serious distinctions in the case of functions computability.

Getting on computable functions — they (computable by MT) are those f for which there is an MT which computes it. That is, given with any input S_n, MT gives output $S_{f(n)}$ if $f(n)$ is defined or does not stop or gets suspended if $f(n)$ is not defined. It is sufficient just to a little adjust the definition of the algorithms A_n above to model it. More precisely if A_n stops in case (i) and outputs the number $r \in N$ it has to change it to $r + 1$ and stop. The rest is the same. Then the modified one accordingly Alg will be an algorithm, which in particular, computes some one argument computable functions.

Notice that we might start with MT_1 and MT_2 as with MTs computing $x + 1$ and $x + 2$, so this Alg indeed computes then a function which domain is not empty — function is non-trivial. And again no Turing machine M, or M_n modelling this function Alg (in the sense pointed above — transforming the machines tape the same way). The proof is as for the our theorem above.

Though, in this case no way to say that there is no usual Turing machine computing Alg as function. The matter is that after long additional computation (after stop Alg in (iii)) the machine MT (admittedly computing as Alg) may return back after some more steps and make all cleaning to make the same configuration as Alg did on the stop. So, this way machine MT may precisely compute the function computable by Alg. So, this case the words may be only that this machine nonetheless, as earlier above, transforms the tape different way and longer as Alg does — uses more applied instructions.

But as to simply (or more generally if you like) looking at the algorithms transforming strings (tapes) of numbers it (the theorem what proved above) looks curious and a bit unusual — there is an algorithm which cannot be precisely (in pointed sense) modelled by Church-Turing machines.

3. Computation on tapes with fixed length

Now we would like to transfer our result to MT working at tapes of fixed length, this way we may illustrate results of previous section on impossibility to precisely model any algorithm with a Turing machine without implementing time (number of steps) while computation.

Consider some finite fixed non-empty alphabet A and the set A^+ of all nonempty worlds over the alphabet, admit yet that 1 and 0 belong to A. For a given mapping $f : A^+ \to A^+$ we say f is normal if for any $w \in A^+$ the length of w is at most length of $f(w)$.

For any MT working at tapes with the alphabet A the normal mapping f_M is defined as follows.

Definition 4. *For any $w \in A^+$ we start MT at w, if MT ever stops in stopping condition the resulting tape is $f_M(w)$. So, $f_M(w)$ may be sometimes not defined, so the mapping in principle may be only partial.*

Theorem 2. *For any A with at most 2 letters there is a computable normal mapping f which is not partial ($f(w)$ is defined for any w) which is different from any f_M for any MT M working at the alphabet A.*

That is again this computable f looks as not modelling precisely by any MT.

Proof. Let again $M_n, n \in N$ be any fixed computable enumeration of all MTs on the alphabet A. Define now the function f as follows. Let $w \in A^+$, if w is not the sequence of 1, put $f(w) = w$. Otherwise, if w is the string of length n of 1-s, we start M_n on w. Then compulsory one of the following events holds.

(1) M_n extends the length of w; (2) M_n does not extend the length of w and correctly or not correctly stops; (3) M_n does not extend the length of w but fall in a loop (not stops).

As soon as one of this holds (that is always computable) we change the first symbol on the tape on anything different from the first symbol on the initial tape w. It is easy to see that f is computable and everywhere defined. Besides, if $f(w)$ ever stops on tape w (that is not computable but nonetheless) of length n with only 1 in cells, $f_{M_n}(w) \neq f(w)$. So, $f \neq f_{M_n}$ for any M_n. □

So, indeed this computable f cannot be precisely modelled by Mts. But if we will extend the length of type w and may then return back to clean up the content, the argument above again does not work.

I thank A. Morozov for his idea to extend my initial results to MTs working on tapes of fixed length as it is represented in Section 3.

Supported by RFBR and Krasnoyarsk Regional Fund of Science, research project 18-41-240005; Supported by RFFI (-RFBR) and Krasnoyarsk Regional Fund of Science, research project 18-41-240005; supported by the Krasnoyarsk Mathematical Centre and financed by the Ministry of Science and Higher Education of the Russian Federation (Grant no. 075-02-2020-1534/1).

References

[1] G.S.Boolos, J.P.Burgess, R.C.Jeffrey, Computability and Logic (4th ed.), Cambridge UK: Cambridge University Press, 2002.

[2] M.Davis, Engines of Logic: Mathematicians and the origin of the Computer (1st ed.), New York, W. W. Norton and Company, 2000.

[3] Neary, Turlough, Woods, Damien, "Small Weakly Universal Turing Machines", 17th International Symposium on Fundamentals of Computation Theory, Lecture Notes in Computer Science, 5699, Springer, (2009b), 262–273.

[4] Y.Rogozhin, Small Universal Turing Machines, *Theoretical Computer Science*, **168**(1996), no. 2, 215–240. DOI: 10.1016/S0304-3975(96)00077-1.

Заметка о вычислениях на машинах Тьюринга со временем вычислений в машинных инструкциях или на лентах фиксированной длины

Владимир В. Рыбаков
Сибирский федеральный университет
Красноярск, Российская Федерация
Институт систем информатики им. А. П. Ершова
Новосибирск, Российская Федерация

Аннотация. В этой короткой статье мы анализируем вычислительные алгоритмы, моделируемые машинами Черча, Тьюринга, Поста в сравнении с алгоритмами, которые используют время вычисления в вычислительных инструкциях. Мы замечаем, что существует некоторое существенное различие в поведении таких вычислений, и иллюстрируем это примерами. Мы рассматриваем работу машин Тьюринга на лентах фиксированной длины и также замечаем примечательное различие.

Ключевые слова: вычисления, алгоритм, универсальные машины Черча-Тьюринга, время вычисления.