Ecological Turing machines

Bruno Durand[1], Andrei Muchnik[2], Maxim Ushakov[3], and Nikolai Vereshchagin[4]

[1] Laboratoire d'Informatique Fondamentale de Marseille, CNRS - Université de Provence, CMI, 39 rue Joliot-Curie, 13453 Marseille Cedex 13, France; email: Bruno.Durand@lif.univ-mrs.fr

[2] Institute of New Technologies, Moscow e-mail: muchnik@lpcs.math.msu.su

[3] Moscow State University, email: ushakov@mccme.ru

[4] Moscow State Univesity, email: ver@mccme.ru The work was done while visiting LIF, CNRS - University of Provence; also supported in part by the RFBR grants 02-01-22001, 03-01-00475.

Abstract. The goal of this paper is to investigate the power of Turing machines with homogeneous memory. The standard models of Turing machines often use tricks such as "special" symbols (delimiter) or several different tapes. These tricks are not natural: computers use only 0's and 1's and operating systems consider memory as a whole. When memory is divided into several parts, (e.g. hard disks, devices...) they play the same role and we cannot say that one of them is devoted to input while the rest is a working space.

We address the question of computing power of variants of Turing machines with no delimiter, and also investigate how a Turing machine is forced to transform its environment while computing. For this last question, we consider the rest of its tape as an oracle and see whether it is possible to compute all recursive functions relativized to this oracle. If yes it means that, in the considered model, Turing machines can perform computations without destroying their environment (the model is thus called *ecological*), otherwise computing implies environment transformation.

These problems seem at first sight straightforward but they are not. We could prove 4 main results explained below, but some very simple and intuitive models are yet not clear (we do not know if they are as powerful as standard Turing machines).

1 Introduction

In "real-life" computers the memory is a uniform sequence of bits, so the standard tricks of adding delimiter symbols seem completely artificial in this context. The separation of the memory into several tapes is not completely artificial since in computer, different kinds of memory are treated separately but the idea that one of them is devoted to input and the other to workspace is not natural.

In this paper, work alphabet is equal to the input alphabet. The input is written on the work tape and no additional tape is permitted. We call such machines "ecological" because they should interact with environment without any pollution. The input string to such a machine is written on the tape starting from the leftmost cell and is not delimited by any end-marker. All the cells to the right of the input may contain any garbage, that is, any symbols of the input alphabet. The computation is required to give the same result whatever garbage follows the input. Obviously, if an ecological machine computes a result on some input x then it also computes the same result on all inputs of the form xy. Hence such a machine can compute only functions f satisfying the following requirement: if $f(x)$ is defined and x is a prefix of x' then $f(x')$ is defined too and $f(x') = f(x)$. Such functions are called *prefix* functions.

We present 4 main results. The first 3 (Theorems 1, 2 and 3) address the computational power of such a model. The last one (Theorem 4) explains how computations modify the environment. The environment can be seen as the content of the memory of the computer before the computation starts, and we want to relate this to oracle computations.

We first focus on the question: "Are ecological Turing machines equivalent to usual Turing machines on the domain of prefix functions?" Our main result states that for the above described model of ecological Turing machines with 1 semi-infinite tape the answer is negative: there is a computable prefix function that cannot be computed by any ecological Turing machine with 1 semi-infinite tape. In this result, it is important that there is no delimiter at the end of the input. Note that if there is a special delimiter at the end of the input then every computable function can be computed by an ecological machine (machines can distinguish the delimiter from other tape symbols but cannot write it).

We consider similar questions for multi-tape Turing machines and multi-tape machines with bi-infinite tapes. For the reasons explained above we do not want to specify different rules for each tapes. They are treated as different memory sources, nothing else. Ecological Turing machines with s semi-infinite tapes receive s input strings, each string is written on a separate tape. We show that, in contrast to the case $s = 1$, for every $s \geq 2$ every computable function f of s arguments that is prefix with respect to each argument can be computed by an ecological machine with s semi-infinite tapes.

In the case of machines with s bi-infinite tapes the input to machines is an s-tuple of pairs of binary strings. Each pair is written on a separate tape: one string before the head and the other after it. For every s such a model is weaker than usual Turing machines: there is a computable prefix function of $2s$ variables that cannot be computed by any ecological Turing machine. Here, it is important that every tape has two inputs. If some tape has only one input written to the right of the head (say) then the resulting ecological model is equivalent to usual Turing machines: for every s every computable prefix function of $2s-1$ arguments can be computed by an ecological Turing machine with s bi-infinite tapes that receives one argument on the first tape and the other $2s-2$ on other tapes, one string to the left of the head and the other to the right of it.

Using the same technique we prove the following fact: there is a total (not computable) function from $\{0,1\}^*$ to $\{0,1\}$ and a bi-infinite binary sequence α

with the following two properties: f is computable relative to α (that is, it can be computed by a usual Turing machine with oracle α) but f cannot be computed by any ecological Turing machine with an extra input tape of the length equal to the length of the input and with a bi-infinite work tape initially containing α.

2 Ecological machines with semi-infinite tapes

Let $\{0,1\}^*$ denote the set of all finite binary strings. Consider first Turing machines with one semi-infinite tape and the tape alphabet $\{0,1,\$\}$ that are not allowed to write \$ (the end-marker). That is, no instruction of the machine has \$ in the right hand side. The computation starts with $x\$$ written in $|x|+1$ leftmost cells of the tape. All the other cells may contain initially any symbols (the garbage). For simplicity we consider only machines outputting 0 or 1. More specifically, we assume that machines have two final states s_0, s_1. The machine halts only if it enters either of these states and the output is 0 (1) when it enters s_0 (s_1). We assume that at any step of the computation the machine knows whether the head is in the leftmost cell. In usual Turing machine model we do not assume this, as the machine can label the leftmost cell by a special marker. For ecological model this assumption is important.

We say that such a machine M computes f if it outputs $f(x)$ on the input x for every x in the domain of f whatever garbage is written to the right of the input. If $f(x)$ is not defined the machine can do anything.

Lemma 1. *Every partial computable prefix function $f:\{0,1\}^* \to \{0,1\}$ can be computed by such a machine.*

Proof. Consider a usual Turing machine M (with no restrictions on the tape alphabet) computing f. Let M have m tape symbols. Fix any k with $2^k > m$ and construct a Turing machine N of the above type simulating M. To this end encode each symbol of the M's tape alphabet by a k-bit sequence so that 0^k is not used as a code. The machine N uses the part of the tape to the right of the \$ sign to simulate the work tape of M. Each symbol of M's tape alphabet is represented as its code and 0^k is used to indicate the first cell of the tape of M that has not yet been visited. The only non-trivial thing is the initialization stage, that is, copying the input to the right of the end-marker \$. First we copy the first symbol of the input replacing it by 1. Then we copy the second one, the third one, and so on, so that at any time the copied part of the input is replaced by $00\dots01$. Thus we can find the first non-copied symbol as the symbol that comes after the first 1 from the left.

Now consider the same model but this time assume that the input x is not delimited by the end-marker. The tape alphabet is $\{0,1\}$. Again all cells to the right of the input may contain a garbage. Formally, every command of a machine has the form $s,a,b \mapsto s',a',m$ where a is the scanned symbol, s is the internal state of the machine, b is equal to 1 if the head scans the leftmost cell and to 0 otherwise, s' is the new state of the machine, $a' \in \{0,1\}$ is the new symbol replacing the scanned one, and $m \in \{-1, 0, 1\}$ is the movement of the head. We say that such a machine M computes a partial function $f : \{0,1\}^* \to \{0,1\}$ if for every x in the domain of f the machine halts in the state $s_{f(x)}$ if at the start the tape contains $x * * * \ldots$ (stars represent any symbols of $\{0,1\}$). Here s_0, s_1 are final states of M. If $f(x)$ is not defined the machine can do anything. Such machines are called *ecological Turing machines with one semi-infinite tape.* Obviously every ecological machine computes a partial prefix function f from $\{0,1\}^*$ to $\{0,1\}$.

The power of ecological machines seems to be very restricted: to perform a computation, the machine should write something on the tape, thus corrupting the input data. Therefore, it seems quite plausible that the ecological computational model is weaker than the usual one. Our main result confirms this intuition: there is a computable partial prefix function that cannot be computed by ecological machines.

Theorem 1. *There is a computable partial prefix function $f : \{0,1\}^* \to \{0,1\}$ that is not computable by ecological Turing machines with one semi-infinite tape.*

Proof. Let $g : \mathbb{N} \to \mathbb{N}$ be a computable function that grows sufficiently fast. We will see further how much is this "sufficiently". Consider the following function $f(x)$. It is defined on all the strings x of the form $x = 0^k 1yuv$ where $|y| = 2^{g(k)}$, $|u| = g(k)$ and v is any string. Consider u as the binary expansion of a number n in the range $0, \ldots, 2^{g(k)} - 1$. Then $f(x) = 1$ if nth bit of y is 1 and 0 otherwise.

Obviously f is a computable prefix function. Informally, ecological machines cannot compute f because while the head moves from the beginning of the tape to the place where n is written, the machine should store somewhere the distance between the current position of the head and the beginning of n. Thus it has to corrupt some part of y.

Assume that the function f is computable by some ecological machine with one semi-infinite tape. Fix a large k and a string y of length $2^{g(k)}$. Assume that the string y contains both 0 and 1. Run the machine on the input $0^k 1y000\ldots$. At some time $\tilde{t}$ the head reaches the cell with the first bit of n, that is, the cell number $N = k + 1 + 2^{g(k)} + 1$. We claim that $\tilde{t}$ is much larger than N.

Let $k < i < N$ be a boundary between two consecutive tape cells (k is the boundary between the last 0 and 1 in $0^k 1$). Consider the sequence of states of the machine at all the times when the head crosses the boundary i (we consider the run up to the time $\tilde{t}$ only). This sequence is called the *crossing sequence* [5] *at boundary i*. We will prove that crossing sequences at different boundaries are different (as sequences).

Lemma 2. *If a machine halts on all inputs of the form $0^k 1\alpha$ ($\alpha : \mathbb{N} \to \{0,1\}$) and on an input I of this form it reaches for the first time the cell $N > k + 1$ at the time $\tilde{t}$, then all crossing sequences of the machine up to time $\tilde{t}$ at all boundaries i where $k < i < N$ are different.*

[5] The notion of crossing sequence was used in [2, 3] to prove that one-tape Turing machines cannot recognize the symmetry of input x in $o(|x|^2)$ steps.

Assume the lemma is true.

Let y of length $2^{g(k)}$ have both 0 and 1. Then on input $0^k 1 y 000\dots$ the machine at some time reaches the cell number $N = k+1+2^{g(k)}+1$. By the lemma all the crossing sequences are different. Let M have Q states. The number of different crossing sequences of length less than l is equal to $Q^0+Q^1+\dots Q^{l-1}$ and is less than Q^l. Letting $l = \frac{g(k)-1}{\log_2 Q}$ we see that for at least half of the boundaries the crossing sequence is longer than $\frac{g(k)-1}{\log_2 Q}$. Therefore the machine M reaches the boundary N not faster than in $\frac{(g(k)-1)2^{g(k)}}{\log_2 Q}$ steps. All the configurations of the machine up to this time are different (a configuration includes the state, the contents of the first $2^{g(k)}+k+1$ cells, and the position of the head). The sets of configurations for different y's are disjoint too, because if we start the machine in each configuration and try all possible n, we can find y.

Hence, the total number of configurations for all y is not less than

$$\frac{(2^{2^{g(k)}}-2)2^{g(k)}(g(k)-1)}{\log_2 Q}.$$

On the other hand, the total number of all possible configurations is

$$Q \cdot 2^{k+1+2^{g(k)}}(k+1+2^{g(k)}).$$

Neglecting multiplicative constants (not depending on k) we see that the first number is equal to $2^{2^{g(k)}+g(k)}g(k)$ and the second number to $2^{2^{g(k)}+g(k)+k}$. Their ratio is $g(k)/2^k$. Letting say $g(k) = 2^{2k}$ we get a contradiction if k is sufficiently large.

It remains to prove the lemma.

Proof. Assume that the crossing sequences at boundaries $i > k$ and $i+j$ in the computation on the input I coincide for some $j > 0$. Let w stand for the part of I to the left of i and v between the boundaries i and $i+j$. We will prove that the machine M does not halt on the input $wvvv\dots$.

The proof is by "cut-and-paste" method which is rather folklore. Nevertheless we put it here for completeness.

Let A denote the computation on the input I and B on the input $wvvv\dots$. Let the crossing sequences in the computation A at boundaries i and $i+j$ be $q_1, q_2, \dots, q_n$. We indicate only the states of the machine when it crosses the boundary; the direction is obvious: odd states correspond to crossings from left to right and even states correspond to crossings from right to left.

Let us partition the tape into zones as follows. The zone 0 stretches from the beginning of the tape up to the boundary i, zone 1 from the boundary i up to $i+j$, zone 2 from $i+j$ to $i+2j$ etc.

We will compare the computation B within zone k with the computation A within zone $p(k)$ where

$$p(k) = \begin{cases} 0, & \text{if } k = 0, \\ 1, & \text{if } k \neq 0. \end{cases}$$

We say that computation B is correct at time T with respect to zone k if there is a time t in the computation A such that the following holds.

a. The content of zone k in the computation B at the time T is equal to the content of zone $p(k)$ in the computation A at the time t.
b. Let l denote the number of crossings, in the computation A, of the left boundary of zone $p(k)$ up to time t (if $p(k) = 0$ then l is undefined), and m the number of crossings of the right boundary. Then in the computation B up to time T the left boundary of zone k has been crossed exactly l times, in the states $q_1, \dots, q_l$, and the right boundary has been crossed exactly m times in the states $q_1, \dots, q_m$. This implies that if at the time t in computation A the head is to the left of (to the right of, within) zone $p(k)$ then at time T in computation B the head is to the left of (to the right of, within) the zone k.
c. If in the computation B at time T the head is within the zone k and scans mth symbol of the zone, and the machine is in the state q then in the computation A at time t the head scans mth symbol of the zone $p(k)$, and the machine is in the same state q.

Let us prove by induction that the computation B is correct at all the times with respect to all zones. The base of induction is evident. Let us prove the induction step.

Assume first that at the time $T+1$ the machine is in the same zone k as it is at time T. By induction hypothesis computation B is correct at time T with respect to zone k with some t. The states of the machine at time T in the computation B and at time t in the computation A coincide, and the scanned symbols coincide too, as the contents of the corresponding zones coincide. Therefore the states, head positions and contents of zones at times $T + 1$ and $t + 1$ coincide. Hence B is correct at time $T + 1$ with respect to zone k. The correctness with respect to other zones obviously is not disturbed.

Assume that the machine crosses a boundary between zones, say, comes from zone k into zone $k + 1$. By induction hypothesis for zone k there is a time t_k in the computation A. At time $t_k + 1$ in the computation A the head comes out the zone $p(k)$ into the zone $p(k) + 1$ in the state q_{m+1}, where m is the number from the item b. In the computation B, at time T the content of zone k coincides with the content of zone $p(k)$ at time t_k in the computation A. Therefore, in the computation B, the head crosses the boundary between zones in the same state q_{m+1}. Hence the computation is correct at time $T+1$ with respect to zone k with $t = t_k + 1$.

Let us prove that the computation is correct at time $T + 1$ with respect to zone $k + 1$ too. At time $T + 1$ the machine crosses the right boundary of zone k and enters zone $k + 1$ via its left boundary. By induction hypothesis the computation is correct at time T with respect to zone $k + 1$. Therefore at some time t in the computation A the content of zone $p(k + 1) = 1$ is equal to the content of zone $k + 1$ at time T in the computation B. And the last time before T in the computation B the head has crossed the right boundary of zone $k + 1$ in the state q_m (from right to left), and the left boundary in the state q_l (from right to left), where l, m are numbers from the item b. We have just proved that at time $T+1$ in the computation B the head returns in the zone $k+1$ in the state q_{l+1}. Let t be equal to the time when in computation A the head crosses the boundary i for $l+1$st time (from left to right). At that time the content of zone 1 in the computation A coincides with the content of zone $k+1$ at time $T+1$ in the computation B and the machine's state is the same as it is equal to q_{l+1}. Hence the computation is correct at time $T+1$ with respect to zone $k+1$ with this t.

The case when the head crosses the boundary from right to left is similar and the induction step is done.

Since computation B is correct at all times with respect to all zones, we conclude that the computation B is infinite. Indeed, at every step the state of the machine is equal to a state of the machine in the computation A before time $\tilde{t}$. Obviously that state is not final and hence in computation B the machine is never in a final state.

Now we proceed to machines having several semi-infinite tapes. Every command of a machine with k tapes has the form

$$s, a_1, b_1, \dots, a_k, b_k \mapsto s', a_1', \dots, a_k', m_1, \dots, m_k$$

where $a_1, \dots, a_k$ are the scanned symbols, s is the internal state of the machine, b_i is equal to 1 if the head on the ith tape scans the leftmost cell and to 0 otherwise, s' is the new state of the machine, $a_i' \in \{0,1\}$ is the new symbol replacing the scanned symbol on ith tape, and $m_i \in \{-1, 0, 1\}$ is the movement of the head on ith tape. We say that such a machine M computes a partial function $f : (\{0,1\}^*)^k \to \{0,1\}$ if for every x in the domain of f the machine halts in the state $s_{f(x_1,\dots,x_k)}$ if at the start the ith tape contains $x_i * * * \dots$. If $f(x_1, \dots, x_k)$ is not defined the machine can do anything. Such machines are called *ecological Turing machines with k semi-infinite tapes*. Obviously every ecological machine computes a partial prefix function f from $(\{0,1\}^*)^k$ to $\{0,1\}$. The latter means that if $f(x_1, \dots, x_k)$ is defined and x_i' extends x_i then $f(x_1', \dots, x_k')$ is defined and equal to $f(x_1, \dots, x_k)$.

Theorem 2. *For every $k \geq 2$ every partial computable prefix function $f\colon (\{0,1\}^*)^k \to \{0,1\}$ is computable by an ecological machine with k semi-infinite tapes.*

Proof. First we define another computational model—a version of counter machines. The machine has a finite state control unit and n *counters*, each containing a natural number, initially zero. Besides, the machine has k semi-infinite tapes, all of them one-way (one-way tape is a tape with a head that can move only to the right and at the start scans the leftmost cell). The control unit can increment and decrement the counters by 1 (decrementing a counter whose value is zero does not change its value), and ask whether the value of a counter is zero. Thus the difference with the model in the statement of the theorem is as follows: the machine has extra n counters but is not allowed to move heads to the left. Formally, every command of a machine has the form

$$s, a_1, \dots, a_k, b_1, \dots, b_n \mapsto s', i_1, \dots, i_n, m_1, \dots, m_k$$

where s is the internal state of the machine, $a_1, \ldots, a_k$ are the scanned symbols, b_j is 1 if the value c_j stored in the jth counter is zero and 0 ohterwise, s' is the new state of the machine, $i_j \in \{-1, 0, 1\}$ is the increment/decrement of the jth counter (after executing the command the new content is equal to $c'_j = c_j + i_j$), and $m_i \in \{0, 1\}$ is the movement of the head on ith tape (note that $m_i = -1$ is not allowed). The input to such a machine is a tuple of k binary strings written initially on the tapes and followed by garbage. We say that such a machine M computes a partial function $f : (\{0,1\}^*)^k \to \{0, 1\}$ if for every x in the domain of f the machine halts in the state $s_{f(x_1,\ldots,x_k)}$ if at the start the ith tape contains $x_i * * * \ldots$. If $f(x_1, \ldots, x_k)$ is not defined the machine can do anything. We will call these machines n *counter machines*, or machines with n counters.

Lemma 3. *For some constant n every partial computable function $f\colon (\{0,1\}^*)^k \to \{0, 1\}$ is computable by a machine with n counters.*

Proof. If we have a sufficient number of counters, we can add, subtract, multiply, and divide numbers in two given counters. For example, to add numbers in counters A and B we need to decrease B simultaneously increasing A until B becomes zero. In a similar way we can implement subtraction; multiplication and division are implemented via addition and subtraction. So, we can check primality, and find primes in increasing order.

Let $f\colon (\{0,1\}^*)^k \to \{0, 1\}$ be a partial computable function. Then f is computable by a Turing machine M having k semi-infinite one-way tapes and an extra read/write two-way work tape (as the work tape has no input at the start, we can use it in usual way as the memory in the computation). It suffices to simulate such a machine by a counter machine. To this end we need to keep the content of the work tape of M in one of the counters. We can do this by encoding each tape symbol as a number, and storing in the first counter the number $T = 2^{t_1}3^{t_2}5^{t_3}\ldots p_N^{t_N}$, where 2, 3, 5, …, p_N are consecutive primes, and t_1, t_2, …, t_N are codes of symbols of the tape of the simulated machine. The second counter keeps the position of the head on the work tape of the simulated machine.

Moving the head of M on the work tape corresponds to increasing/decreasing the second counter. Changing the scanned symbol on the work tape corresponds to dividing/multiplying (several times) the value T of the first counter by p_i where i is the value stored in the second counter. This can be done using constant number of other counters as described above.

Moving the heads of M on input tapes corresponds to moving the same heads on input tapes (recall that the simulating machine has the same number if input tapes).

Lemma 4. *Every computable prefix function $f\colon (\{0,1\}^*)^k \to \{0, 1\}$ is computable by a machine with 2 counters.*

Proof. By Lemma 3, for some n there is a machine with n counters that computes f. Encode values of counters into one number $S = 2^{c_1}3^{c_2}\ldots p_n^{c_n}$, where, as before, p_i is ith prime number and c_i is the value of ith counter. We store the number S in the first counter. The second one will be used for intermediate computations.

Now we have to show how to simulate operations with counters. Incrementing and decrementing ith counter correspond to multiplying and dividing S by p_i. Multiplying is done like this: while the first counter is not zero, decrement it and then p_i times increment the other one. Division is done in a similar way.

When the simulated machine moves its head on any of the tapes so does the simulating 2 counter machine.

Note that the simulating 2 counter machine constructed in the proof of Lemma 3 has the following property: at any step of the simulation the second counter is empty when the 2 counter machine moves any of its heads. We call such 2 counter machines *correct*.

Thus, given a correct machine M_1 with 2 counters that computes f, we have to construct an ecological machine M_2 computing f. We will assume first that $k = 2$. Let $s_0 s_1 s_2 \dots$ be the content of the first tape of M_1 and i the position of the head on it, $t_0 t_1 t_2 \dots$ the content of the second tape of M_1 and j the position of the head on it. Let C_1, C_2 be the contents of the counters. The corresponding configuration of the ecological machine looks as follows: the first tape contains $00 \dots 1 s_{i+1} s_{i+2} \dots$, the second tape $00 \dots 1 t_{j+1} t_{j+2} \dots$. The position of the head is C_1 on the first tape and C_2 on the second one.

If the counter machine increments/decrements a counter, the ecological machine just moves the corresponding head. As the ecological machine knows whether the head is in the beginning of the tape, it knows whether the counter is empty. Assume that the counter machine moves either of its heads to the right and scans a new input symbol. Note that in this case the second counter is empty and hence the second head of the ecological machine is in the beginning of the tape. If the counter machine moves the second head then the ecological machine finds the first 1 on the second tape, replaces it by 0, scans the next symbol, replaces it by 1, and returns to the beginning of the tape. If the counter machine moves the first head then the ecological machine first copies the content of the counter to the second tape (moving both heads simultaneously in the opposite directions until the first one is in the beginning) and then does the same thing. After that it copies the content of the counter back on the first tape.

For $k > 2$ the simulation is entirely similar: we use the first two tapes as explained above to store counters. The heads on other tapes of simulating machine move just as the heads of the simulated machine (and they do not change the contents of the tapes).

Remark 1. It is important, in the above theorem, that the simulating machine knows whether its heads are at the leftmost cell. If machines do not know that and are required not to try to come beyond the tape then the resulting model is weaker than usual Turing machine model. This can be proven by arguments similar to those used in the proof of Theorem 3 below.

3 Ecological machines with bi-infinite tapes

Now we proceed to machines having several bi-infinite tapes. Every command of a machine with k bi-infinite tapes has the form

$$s, a_1, \dots, a_k \mapsto s', a'_1, \dots, a'_k, m_1, \dots, m_k$$

where $a_1, \dots, a_k$ are the scanned symbols, s is the internal state of the machine, s' is the new state of the machine, $a'_i \in \{0,1\}$ is the new symbol replacing the scanned symbol on ith tape, and $m_i \in \{-1, 0, 1\}$ is the movement of the head on ith tape.

There are two options to give inputs to machines with bi-infinite tapes. The first one is to write inputs (followed by garbage) to the right of the initial positions of heads. That is, an ecological machine with k semi-infinite tapes receives k inputs and computes a prefix function of k variables. In this case every computable prefix function is computable by an ecological machine.

The second option is to write two inputs on one bi-infinite tape: one input to the right of the initial position of the head and another one to the left in the reverse order. Note that if on at least one tape there is only one input then the ecological model is equivalent to the usual one. Therefore we will assume that every tape has two inputs. We say that such a machine M computes a partial function $f : (\{0,1\}^*)^{2k} \to \{0,1\}$ if for every x in the domain of f the machine halts in the state $s_{f(x_1,y_1,\dots,x_k,y_k)}$ if at the start the ith tape contains $\dots * * * y_i^R x_i * * * \dots$ and the head is between y_i^R and x_i (y^R means y in reversed order). If $f(x_1, y_1, \dots, x_k, y_k)$ is not defined the machine can do anything. Thus an ecological machine with k bi-infinite tapes computes a prefix function of $2k$ arguments. It turns that for every k such machines are weaker than usual ones.

Theorem 3. *For every $s \geq 1$ there exists a computable prefix function $f\colon (\{0,1\}^*)^{2s} \to \{0,1\}$ that is computable in the usual sense, but is not computable by any ecological Turing machine with s bi-infinite tapes.*

Proof. First, we prove the statement for $s = 1$. Consider the following function $f(x, y)$. It is defined on all the strings x, y of the form $y = 0^k 1zuv$, $x = rw$, $|r| = |z| = 2^{2k}$, $|u| = 2k + 1$. Consider u as the binary expansion of a number n in the range $0, \dots, 2^{2k+1} - 1$. Then $f(x, y) = 1$ if nth bit of the string rz is 1 and 0 otherwise.

Informally, ecological machines cannot compute f because while the head moves from the beginning of the tape to the place where n is written, the machine should store somewhere the distance between the current position of the head and the beginning of n. Thus it has to corrupt some part of rz.

Fix k, r, z, P where P is a number in the range $1, \dots, 2^{2k}$ and r, z are binary strings of length 2^{2k}. Run the machine with r to the left of the head and $0^k 1z$ to the right of the head. Consider the first time T the head moves beyond position P or $-P$ (position $-P$ is the Pth cell to the left of the initial position of the head). Let a stand for the content of the tape from the position -2^{2k} up to the position of the head and b the content of the tape from the position of the head up to the position $2^{2k}+k+1$ (at the time T). Let q be the state of the machine at the time T. To derive a contradiction we will prove that both r, z and P can be reconstructed given the pair $\langle q, ba\rangle$ unless the string rz consists of blocks 00 and 11. Note that we are not given the lengths of a, b and do not know the place where b ends in the string ba. Here the order in which we concatenate b and a is important—we do not claim that r, z and P can be reconstructed given the pair $\langle q, ab\rangle$. To reconstruct r, z and P it is enough to find a and b (then we can run the machine from the configuration at time T and try all possible n; recall that the machine has not yet visited the place where n is written before time T). To this end try all possible divisions of ba into b' and a' starting with the longest b'. For each division and for all $n \le |rz|$ run the machine in the state q, with ba on the tape before the head, and $b'n$ after it. Combine the machine's outputs into string $r'z'$. If b' is longer than b, $b' = bb''$, then $r'z'$ will consist of blocks 00 and 11. Indeed, the machine will consider b'' as the most significant bits of n. And therefore the answer will not depend on several least significant bits of actual n. Hence the resulting string $r'z'$ will consist of blocks 00 and 11. This can be easily checked. If this is the case, the division is wrong, unless the string rz consists of blocks 00 and 11, and we proceed to the next division until the correct division is found.

The number of different triples $\langle r, z, P\rangle$ such that the string rz does not consist of blocks 00 and 11 is about $2^{2^{2k+1}}2^{2k}$ (it is equal to $(2^{2^{2k+1}} - 2^{2^{2k}})2^{2k}$) and the number of different pairs $\langle q, ba\rangle$ is $Q2^{2^{2k+1}+k+1}$ where Q stands for the number of the states of the machine. For large enough k the latter number is less than the former one and we obtain a contradiction.

The proof generalizes easily to the case $s > 1$. In that case the function $f(x_1, y_1, \ldots, x_s, y_s)$ is defined as follows. It is defined when $y_1 = 0^k 1 z_1 u_1 v_1$, $x_1 = r_1 w_1$, and $y_i = z_i u_i w_i$, $x_i = r_i w_i$, for $i > 1$, and $|r_i| = |z_i| = 2^{2k}$, $|u_i| = 2k + 1 + \lceil \log_2 s\rceil$. Let n_i be the number whose binary expansion is u_i and let n be equal to the sum of all n_i modulo $s2^{2k+1}$. Then $f(x_1, y_1, \ldots, x_s, y_s)$ is the nth bit of the string $Z = r_1 z_1 \ldots r_s z_s$.

The above proof is modified as follows. We consider the tuple $\langle r_1, z_1, \ldots, r_s, z_s, P\rangle$, where P is a number in the range $1, \ldots, 2^{2k}$, and consider the first time when some head moves beyond position P or $-P$. The strings a_i and b_i are defined in the same way as above: a_i is the content of ith tape from the position -2^{2k} to the current position of the head and b_i is the content of the ith tape from the position of the head up to position 2^{2k} ($2^{2k} + k + 1$ if $i = 1$).

To derive a contradiction we need to prove that Z and P can be reconstructed given $\langle q, b_1 a_1, \ldots, b_s a_s\rangle$ unless Z consists of blocks of 00 and 11. To do this, we find a_i and b_i for all i in succession as follows. Try all possible divisions of $b_i a_i$ into a'_i and b'_i starting from the longest b'_i. For each division run the machine, writing on all tapes except ith one the strings $b_1 a_1, \ldots, b_s a_s$ both before the heads and after them. On ith tape write $b_i a_i$ before the head, and $b'_i n$ after it (for all $n \le s2^{2k+1}$). If b'_i is longer than b_i, the resulting string Z' will consist of blocks 00 and 11. Thus we will know that the division is incorrect, as, for the correct division, the string Z' is a cyclic shift of Z, so it cannot consist of such blocks.

4 Computing functions with an oracle

Consider now ecological Turing machines of the following type. The machine has a two-way read/write finite input tape of the length equal to the length of the input. It has a bi-infinite work tape initially containing a bi-infinite binary sequence α. The machine knows whether the head on the input tape is in the beginning of the tape or in the end. Both the input tape alphabet and the work tape alphabet are binary.

Theorem 4. *There is a total function from $\{0,1\}^*$ to $\{0,1\}$ and a bi-infinite binary sequence α such that f is computable relative to α but f cannot be computed by any ecological Turing machine of the above described type.*

The proof is omitted due to space restrictions.

5 Open questions

Many questions about the relative power of usual and ecological computational models remain open. We list some of them.

1. What about ecological machines with one bi-infinite tape and one semi-infinite tape? Are they weaker than usual machines?

2. Is there an analog of Theorem 4 for machines with one semi-infinite tape?

3. Is every prefix partial function $f(x,y)$ computable in polynomial time $\text{poly}(|x|+|y|)$ computable also in polynomial time by an ecological machine with two semi-infinite tapes? Note that the simulation in the proof of Theorem 2 has exponential overhead in time.

4. What can be said of the power of ecological RAM (Random access machines)? The input data to such a machine is an infinite sequence of natural numbers stored in its registers.

References

1. V.N. Agafonov. Complexity of algorithms and computations: A course for students of University of Novosibirsk, part 1. Publishing house of University of Novosibirsk, 1975. (Russian)
2. Ya.M. Barzdin. Complexity of symmetry recognition on Turing machines, Problemy kibernetiki, v. 15 (1965), 245–248. (Russian)
3. F.C. Hennie. One tape off-line Turing machine computations. Information and Control, 8:6 (1965) 553–578.