On quasi-unilateral universal Turing machines

Maurice Margenstern
Université de Metz, I.U.T. de Metz, LRIM and LIAFA.

Abstract. Four universal Turing machines are given with strong limitations on the number of *left instructions* in their program. One of the machines is morever non-erasing, *i.e.* it does not erase any 1 written on the tape.

Keyword : theory of computation

In this paper, we consider only *deterministic* Turing machines on alphabet $\{0,1\}$, with one head and one tape, infinite in both directions. Symbol 0 plays here the rôle of the blanck symbol.

Recall that *instructions* of such a Turing machine look like the following :

(I)

i	x	M	y	**j**

where, scanning the current cell under state **i**, the head of the machine reads symbol x, then overwrites it with symbol y, performs displacement M in order to scan the right neighbouring cell if $M = R$, the left neihgouring one if $M = L$, the same place if $M = S$, and then enters under state **j** for the next step of the computation. Symbol M is called the the *move* of instruction (I).

In the sequel, M does not take value S, except, possibly, for a halting instruction.

Call instruction (I) a *right* instruction if $M = R$, a *left* one if $M = L$. A *unilateral* Turing machine is a machine the instructions of which have all the same move but, possibly, halting instructions. Notice that we may choose that there is a single halting instruction.

It is well known, that for *unilateral* machines, the halting problem is decidable.

Up to what extent is it necessary to remove unilaterality in order to obtain universal Turing machines ?

In an almost unnoticed paper, [9], Pavlotskaya states the following property :

Theorem 1 : Pavlotskaya [1973] *For any Turing machine on alphabet* $\{0,1\}$ *with a single left instruction in its program, the halting problem is decidable.*

In [1] and [5], we extend this result a bit : to two left states in a non-erasing context. Namely :

Theorem 2 : Margenstern [1993] *For any non-erasing Turing machine on alphabet* $\{0,1\}$ *with precisely two left instructions in its program, the halting problem is decidable,*

where *non-erasing* means that the machine is never permitted to change a 1 into 0.

On the other hand, we have frontier results in both the erasing and the non-erasing cases :

Theorem 3 : Margenstern and Pavlotskaya [1995] *There is a universal Turing machine on* $\{0,1\}$ *with precisely two left instructions in its program.*

And for non-erasing machines :

Theorem 4 : Margenstern [1994] *There is a non-erasing universal Turing machine on* $\{0,1\}$ *with precisely three left instructions in its program. Moreover, such a machine may be constructed with* 218 *states.*

The goal of this paper is to indicate how to construct these universal machines. A sketchy proof of theorem 3 is given in section 6. We begin with the non-erasing case which involves other universal Turing machines with only *a few* left instructions :

Theorem 5 : (see §3) *There is a universal Turing machine on* $\{0,1\}$ *with precisely six left instructions in its program. Moreover, such a machine may be constructed with* 59 *states.*

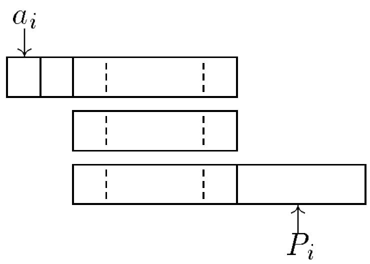
Notice that the latter machine is a small universal one. The smallest presently known universal Turing machine on $\{0,1\}$ belongs to Rogozhin with a machine with 21 states ([12]). But this machine has 20 left instructions.

Theorem 6 : (see §4) Margenstern [1993] *There is a universal Turing machine on* $\{0,1\}$ *with precisely three left instructions in its program. Moreover, such a machine may be constructed with* 190 *states.*

All the considered machines use the same technique, which we call *Minsky-Rogozhin* technique since, first completely formulated in [8], it was impressively used in [10], see also [11]. This technique consists in simulating any 2-*tag-system* by a Turing machine under a suitable encoding.

1. Recall on tag-systems

Particular case of Post systems, a p-tag-system is a computation associated to a mapping from alphabet A into A^*. Call the image of a its production. One step of the computation is constituted by performing the following three operations, illustrated, below, with $p = 2$:



where P_i is the production of a_i

First, remember the first letter of the tagged word, *i.e.*, submitted to the tag-system. Second, erase the first p letters of the word. Third, append the production corresponding to a_i to the right end of what remains of the tagged word. Repeat this process on the word obtained until either the tagged word has less than p letters, or the first erased letter is a *halting* one. Halting letters are distinguished letters of A which, by definition, makes the computation of the tag-system to halt when they are scanned as a first letter of the tagged word. A single halting letter suffices.

The reason why these systems are considered is the following property, proved, for instance, in [8] :

Theorem Coke-Minsky [1962] *Any deterministic Turing machine can be simulated by a 2-tag system.*

Wang proved that, on the contrary, 1-tag-systems do have a decidable halting problem ([13]).

Let us illustrate a *tag-computation* on the following example (from now on, we say tag-sytem for 2-tag-system) :

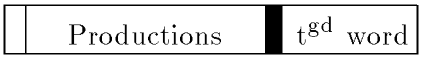
$$(1) \qquad \boxed{\begin{array}{l} a \longrightarrow b \\ b \longrightarrow bc \\ c \longrightarrow\, ! \end{array}}$$

Starting from word bbb we get : $bbb \longmapsto bbc \longmapsto cbc \longmapsto c\,!$, where symbol ! indicates that the computation is completed.

2. Simulating technique

The tape of the simulating machine replicates the *simulated* configuration under a suitable encoding. In order to encode any given alphabet, letters are encoded as 1^n0, where 0 plays the rôle of a delimiter. Productions are encoded as concatenations of encoded letters and are delimited by an extra 0. If a, b and c are encoded, respectively, 10, 110 and 1110, productions of the above tag-system are encoded as 0110011011100, where the first 0 is a left delimiter for all the productions. In the machines for which we indicate, below, a construction, *the number of 1's in the code of a letter is the number of 0's lying between the first 0 of the productions and the first 1 of the production corresponding to the letter, first 0 included.*

Simulating one step of the tag-system computation is indicated by the following figure, splitting the cycle of Turing steps into five stages :

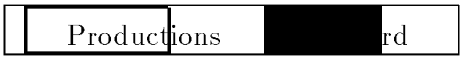


a) initial configuration

b) locating the production,

which takes place while the first letter of the tagged word is being erased

c) erasing the second letter



d) copying the production

e) preparing the next step of the tag-system

The precise fashion for performing these stages is different in the various machines to which we turn now.

3. The machine with six left instructions

In this machine, the left instructions are split into two groups of three ones, each group following, say, this pattern :

	0	1
i	$R\,\mathbf{k}$	$L\,\mathbf{j}$
j	$L\,\mathbf{i}$	$L\,\mathbf{i}$

where, in each (reading) instruction of this table, unchanged elements are not mentionned.

Each group defines a move from the right end of the configuration to its left end. These groups define two *signals*. One says : *'repeat the previous stage'*, the other one : *'turn to the other stage'*. The main two stages are indicated by the odd bit which precedes the first 1 of the simulating configuration, and which we call the *stage bit*.

The encoding is adapted to this situation : every cell of the simulated configuration is encoded by a four-celled unit, say four-bits unit. Bits 0 and 2 are set to 1 for signalizing the simulated configuration and making it possible to cross over the configuration from right to left via the above instructions. Bit 1 is the content of the simulated cell and bit 3 is a marker. Let **0**, **0'**, **1** and **1'** denote, respectively `1010`, `1011`, `1110` and `1111`. Unit `0000` delimitates both ends of the simulated configuration.

During the locating stage (*stage bit* to 0), each **1** in the first letter a_i of the tagged word is marked as **1'** while, during the next crossing over from left to right, a corresponding **0** is marked as **0'** in the productions. Starting with $\mathbf{1}^n\mathbf{0}$, the encoding of a_i becomes $\mathbf{1'}^n\mathbf{0}$ when the locating stage is completed. The machine detects this situation by finding a **0** instead of a **1** after the block of **1'** units.

The replicating stage (*stage bit* to 1) is then entailed by calling the second group of left instructions. The machine searches the first unmarked symbol. If it is a **1**, it marks it and goes to the right of the configuration for writing **1** on the tape. If it is a **0**, the machine marks it as **0'** and scans the next unit : if the latter happens to be **1**, a **0** is written on the tape and the machine goes back to the left via the *repeat* signal. If the next unit happens to be **0**, then a **0** is written on the tape by an other group of states and the *change* signal is taken for coming back to the stage bit.

As the *change* signal scans stage bit to 1, a group of states is called in order to unmark the productions : the next locating stage will take place with the next return to the left. The process is completed when three consecutive **0'** units are noticed. The same thing is used to recognize the halting production : when the machine counts three consecutive **0'** units while it looks for a symbol to replicate during the replicating stage.

4. The erasing machine with three left instructions

It appears that the simulating process can be performed with only one left *signal*. The price for this is increasing the complexity of the encoding in order to put on the tape the *signals* that cannot be provided by the states.

Intuitively, the machine mimicks a three-headed, one tape machine on alphabet $\{0, 1\}$ which would simulate a tag-system. For this purpose, coming back to the figure of section 2, let us suppose that the first head is at the beginning of the productions, the second at the beginning of the tagged word and the third at the end of that word. During the locating stage, the first two heads move more or less in parallel. During the replication, the first and the third move strictly parallel. It can be noticed now that a *single* move to the left is enough for turning to the next step of the tag-computation : the coming back of the first head to its starting position.

We simulate these heads with a single one by setting a marker on the tape at the starting points of each mimicked head. When the simulating head scans one marker (from left to right), it *becomes* the corresponding head. Of course, more or less parallel and strictly parallel actions are delayed in time.

The activity of the machine is a constant replication cycle : each encountered unit is copied right after the right end of the configuration, after crossing it over to the right. Then the machine goes back to the left end, always via the same group of left instructions. As a consequence, the machine has 'forgotten' what it had previously performed when its head is again scanning the left end of the configuration. From time to time, when the machine scans a particular unit, markers are placed or extra units are copied according to which stage is currently being executed.

The encoding is based on ten bit units, with as previously, even bits set to 1 and the useful information spread over the odd bits. We may represent these units by symbols of a greater alphabet. In this machine the above initial configuration looks like the following :

$$ru01110011101111111100m111011101110v$$

Here, u, m, and v are the starting positions of the heads of the tricephalous machine; r stands for a *status word* indicating which stage of the tag-computation is currently being performed. Both r and v indicates the locating stage. The replicating one is indicated by, respectively c and k. Both symbols, one at each end, allow to manage a transition between the main stages of the simulation without breaking the basic cycle of copying the current left end word to the right end of the configuration.

We have no room here for the table of the auxiliary machine working with this greater alphabet (see [2]).

We can only indicate the succession of key configurations as follows :

locating stage :

split itself into two sub-stages : marking first the current 1 in the tagged-word, then, after the configuration has been almost completely replicated, marking a new 0 in the productions :

$$\begin{array}{l} ru\,\boxed{011100\ldots00}\,m1\,\boxed{110\ldots1110}\,v \\ \triangle \end{array}$$

which becomes :

$$\begin{array}{r} r\,\boxed{}\,my\,\boxed{}\,vu \\ \triangle \end{array}$$

where we do not repeat the unchanged information inside the boxes ;

when r comes close to v :

$$\begin{array}{l} rvu0\,\boxed{11100\ldots00}\,m\,\boxed{} \\ \triangle \end{array}$$

the configuration is transformed into this one :

$$\begin{array}{r} rua\,\boxed{}\,m \\ \triangle \end{array}$$

replicating stage :

the locating stage is completed when 0 comes right after m :

$$\begin{array}{l} ruabbbaa\,\boxed{1110\ldots00}\,m01110\,\boxed{1110}\,v \\ \triangle \end{array}$$

the configuration is transformed into a hybrid one, containing the left delimiter of 'locating' ones, and the right delimiter of 'replicating' ones :

$$\begin{array}{r} rabbbaa\,\boxed{}\,myyyyy\,\boxed{}\,ku \\ \triangle \end{array}$$

Notice that y is 'swallowed' by r when the latter comes right to it, which allows to actually erase the fisrt two letters of the tagged word.

then, from a starting replicating configuration :

$$cu\ \boxed{0,1 := a,b}\ 1\ \boxed{110\ldots00}\ \boxed{}\ m\ \boxed{}\ k$$
△

where the empty box corresponds, in our example, to a void zone of cells, but could be, of course, a large one with an other set of productions.

we come to this one, marking the symbol to be next copied :

$$c\ \boxed{0,1 := a,b}\ y\ \boxed{}\ \boxed{}\ m\ \boxed{}\ ku$$
△

which is actually performed when c comes to k :

$$cku\ \boxed{0,1 := a,b}\ y\ \boxed{}\ \boxed{}\ m\ \boxed{}$$
△

the configuration becomes :

$$cu\ \boxed{0,1 := a,b}\ b\ \boxed{}\ \boxed{}\ m\ \boxed{}\ 1k$$
△

The completion of the replicating stage is detected by scanning a 0 right after scanning a 0 transformed into x. Then k is transformed into v, and when c comes right to v, v is changed into r, a's and b's are changed into 0's and 1's, the needed 0 is copied and v is written right after it. Thus, a new starting configuration of the locating stage is reached.

The halting of the process is obtained as follows : we may suppose that the tag-system contains a single halting letter. It is encoded by the number of 0's on the tape which lie between u and m. So when, in the replicating stage, looking for a 0 or 1 to replicate the head falls onto m, it means that the located production is the halting one, whence the halting condition for our Turing machine.

By translating this machine into a machine on $\{0,1\}$, we can obtain by compressing the resulting table a machine with 190 states, the full table of which with suitable explanations can be found in [2].

5. The non-erasing machine with three left instructions

Let us indicate here an alternative proof of the first sentence in theorem 5. Consider a *fixed* tag-system. Then, in order to simulate it, the above locating stage may be internalized in the states of the machine. The only needed move to the left is the coming back of the machine head to the beginning of the new tagged word. This may be simulated within the three left instructions condition as shows the right table below.

If we take a tag-system simulating a universal Turing machine, we then obtain by the same way a universal Turing machine with precisely three left instructions. The simulating configuration is now constituted of the single encoding of the tagged word which looks like as :

$$\ldots 0110110110 \ldots$$

where letters a, b and c may be simply encoded 10, 110, 1110.

It is now not very difficult to transpose that construction into the non-erasing context.

Take the same encoding as previously, but add a new letter to the left of the initial configuration, a for instance, which will play the rôle of a *trash* :

$$\ldots 010110110110 \ldots$$

	0	1
1		$0\,R\,2$
2	$R\,6$	$0\,R\,3$
3	$R\,13$	$0\,R\,4$
4	$R\,5$	
5	**!**	$0\,R$
6	$R\,7$	$0\,R$
7	$1\,R\,9$	$R\,8$
8	$R\,7$	R
9	$1\,R\,11$	
10	$R\,1$	
11	$L\,12$	L
12	$R\,10$	$L\,11$
13	$R\,14$	$0\,R$
14	$1\,R\,16$	$R\,15$
15	$R\,14$	R
16	$1\,R\,17$	
17	$R\,18$	
18	$1\,R\,19$	
19	$1\,R\,20$	
20	$1\,R\,11$	

The table, right, is the non-erasing transposition of the above one. It differs from this one by an extra state which allows to cross over the *trash* of 1's from left to right. When the machine scans a 0, it changes it into 1 and goes to the first 1 in the first letter of the tagged word under state 1. By this way, it adds the encoding of the first letter to the trash. The same thing happens with the second letter when delimiter 0 in the first letter is changed into 1.

Applying this technique to a universal tag-system, this gives a simple proof of the first sentence in theorem 4, see [4].

The same internalization of the locating stage shows that :

	0	1
1	$1\,R$	$R\,2$
2	$1\,R\,6$	$R\,3$
3	$1\,R\,13$	$R\,4$
4	$1\,R\,5$	
5	R **!**	R
6	$R\,7$	R
7	$1\,R\,9$	$R\,8$
8	$R\,7$	R
9	$1\,R\,11$	
10	$R\,21$	
11	$L\,12$	L
12	$R\,10$	$L\,11$
13	$R\,14$	R
14	$1\,R\,16$	$R\,15$
15	$R\,14$	R
16	$1\,R\,17$	
17	$R\,18$	
18	$1\,R\,19$	
19	$1\,R\,20$	
20	$1\,R\,11$	
21	$1\,R\,1$	R

Theorem 7 : *A non-erasing universal Turing machine on $\{0,1\}$ with two heads and without left instructions can be constructed, provided that moves of type $M = S$ are allowed.*

However, if moves of type $M = S$ too are not allowed, the machine has a decidable halting problem, in the erasing case too.

A machine with 218 states

In order to obtain a machine with a rather small number of states, we may proceed as indicated above, in the erasing context.

The global activity is again a cycle of copying a (ten bit) unit right to the right end of the configuration. As 1's cannot be erased, erasing is obtained by *drowning* information under contiguous blocks of erasing units `1111111111`. The same principles govern here each stage of the previous simulating process that we may first modelize on a machine with a richer alphabet as done in [3].

Some care must be taken of the encoding. Replacement of the symbols must obey an order induced by the non-erasing condition in order to be possible : let $\sigma \prec \tau$ iff any bit set to 1 in $c(\sigma)$ is also set to 1 in $c(\tau)$, where $c(a)$ is the *encoding* of a.

The encoding is given by the following tables :

delimiters :

r	:	`1010101010`	(00000)
c	:	`1010101110`	(00010)
m	:	`1011101010`	(01000)
v	:	`1011111010`	(01100)
k	:	`1011111110`	(01110)

symbols and markers :

0	:	`1110101010`	(10000)
1	:	`1111101010`	(11000)
a	:	`1110111010`	(10100)
b	:	`1111111010`	(11100)
t	:	`1111101110`	(11010)
z	:	`1111111111`	(11111)
w	:	`1011111111`	(01111)

unmarkers :

x	:	`1110111110`	(10110)
y	:	`1111111110`	(11110)
u	:	`1011101110`	(01010)

We have the following *precedence* relations :

$$r \prec k \quad m, 0, 1 \prec t \quad 0, 1, a, b, u \prec y$$
$$0 \prec a \quad 1 \prec b \qquad m \prec u$$

We now indicate the succession of key configurations for this machine, as for the previous one :

locating stage :

starting configuration :

$$r\ \boxed{011100\ldots00}\ m1\ \boxed{110\ldots1110}\ v$$
$$\triangle$$

which becomes :

$$z\ \boxed{}\ mt\ \boxed{}\ vr$$
$$\triangle$$

then,

$$\ldots zmt\ \boxed{}\ vr\ \boxed{}$$
$$\triangle$$

and then :

$$\ldots zzt\ \boxed{}\ vr\ \boxed{}\ m$$
$$\triangle$$

when z comes to v, the delayed marking in the productions is performed ;

starting from :

$$\ldots zvr0\ \boxed{11100\ldots00}\ m\ \boxed{}$$
$$\triangle$$

we get :

$$\ldots zzra\ \boxed{}\ m\ \boxed{}\ v$$
$$\triangle$$

when 0 appears right after m, the locating is completed :

$$\ldots zrabbbaa\ \boxed{1110\ldots00}\ m01110\ \boxed{1110}\ v$$
$$\triangle$$

a hybrid configuration appears, with r from 'locating' and k from 'copying' :

$$\ldots zzabbbaa\ \boxed{}\ mttttt\ \boxed{}\ kr$$
$$\triangle$$

t's are *swallowed*, which erases the second letter of the tagged word and a starting replicating configuration is reached :

$$\ldots zzcabbbaa\ \boxed{}\ m\ \boxed{}\ k$$
$$\triangle$$

replicating stage :

starting point :

$$\ldots zc\ \boxed{0{,}1 := a, b}\ 1\ \boxed{110\ldots00}\ \boxed{}\ m\ \boxed{}\ k$$
$$\triangle$$

marking and copying :

$$\ldots zz\ \boxed{0{,}1 := a, b}\ b\ \boxed{}\ \boxed{}\ m\ \boxed{}\ z1kc$$
$$\triangle$$

from that we get, successively :

$$\ldots zkc\ \boxed{0{,}1 := a, b}\ b\ \boxed{}\ \boxed{}\ m\ \boxed{}\ 1$$
$$\triangle$$

$$\ldots zc\ \boxed{0{,}1 := a, b}\ b\ \boxed{}\ \boxed{}\ m\ \boxed{}\ 1k$$
$$\triangle$$

when a 0 changed into a is followed by a 0, the replicating stage is completed :

$$\ldots zc\,\boxed{0{,}1 := a, b}\,00\boxed{}\,m\,\boxed{}\,\boxed{1110\ldots0}\,k$$
$$\triangle$$

it needs to copy a 0 and to change a's and b's back to 0's and 1's, which is annouced by xy :

$$\ldots zz\,\boxed{0{,}1 := a, b}\,a0\boxed{}\,m\,\boxed{}\boxed{}\,z0xy$$
$$\triangle$$

x indicates to put r right after the configuration :

$$\ldots zxy\,\boxed{0{,}1 := a, b}\,a0\boxed{}\,m\,\boxed{}\boxed{}\,0$$
$$\triangle$$

and y is the signal of the unmarking stage ; below, y *swallows* a and copies 0 at the end of the configuration :

$$\ldots zya\,\boxed{0{,}1 := a, b}\,a0\boxed{}\,m\,\boxed{}\boxed{}\,r$$
$$\triangle$$

$$\ldots zzy\,\boxed{0{,}1 := a, b}\,a0\boxed{}\,m\,\boxed{}\boxed{}\,r0$$
$$\triangle$$

b's are unmarked the same way ; 0's and 1's are simply copied ;

for m, in order to lower the number of states, y changes it into u and m is copied at the end of the configuration : the reason is that y recognizes m, to be erased, from r, to remain unchanged, after reading the first two bits ;

later, z changes u into y.

The halting is recognized as in the case of the previous machine.

The translation of the obtained auxiliary machine leads, after compression, to a machine with 218 states. Its full table is given in [3] together with detailed explanation for implementing on $\{0, 1\}$ the simulating process.

6. The erasing machine with two left instructions

This machine, explained in [7] and [6], is based on another principle : it directly implements a two register machine which simulates a universal Turing machine.

Here, we only recall the basic steps of the simulation.

Let x and y be the content of the registers of the two register machine, say R, to be simulated. This content, together with the number of the R-instruction to be performed, say i are encoded in $Y = 2^K.2^x.3^y + 6.i$. Incrementing x or y by one means multiplying $2^x.3^y$ by 2 or by 3. Decrementing these numbers by one means divison of $2^x.3^y$ by 2 or 3. Indeed, as easily seen, only division by 6 is needed.

In order to perform these operations with only two left instructions, one needs a *uniform* algorithm for computing multiplication by 2, by 3 and division by 6. In [7] and [6], this is obtained for the indicated multiplications together with pseudo-division by 6 which is the mapping

$$x \mapsto x + \frac{x}{6} = \frac{7.x}{6}.$$

Let Y be represented in unary on the tape of the Turing machine to be constructed. The basic idea of this uniform algorithm consists in deleting the first L 1's of Y and then, after the head has reached the rightmost 1 of Y, to append $\alpha.L$ 1's with $\alpha = \frac{1}{2}, \frac{2}{3}$ or $\alpha = \frac{1}{7}$ for, respectively, mulitplication by 2, by 3 and pseudo-division by 6.

Let $d_0 = 0$ be the position of the leftmost 1 on the Turing tape for a starting configuration and $f_0 = Y + \alpha.L$. The back and forth motion just described allows to successively define $d_{i+1} = d_i + L$ and $f_{i+1} = f_i + \alpha.L$. The machine performs this motion until $d_{i+1} + L > f_i$. It is easy to see that the head must then perform $\dfrac{\alpha}{(1-\alpha)}.(f_i - d_{i+1})$ in order to reach a cell v on the tape such that $v - d_0 = \dfrac{y}{1-\alpha}$.

For unifying the three needed operation, it is enough to take a multiple of 42 as number L. If we take $L = 42.2^K$, and if $2^K > 6.i$, whatever i, the number of the simulated R-instruction, then the above described algorithm performs the needed operation, and the remainder of division by L allows to save the value of i : this is necessary to make the Turing machine turn to the simulation of the next R-instruction.

Details of the implementation are given in [7] and [6].

Acknowledgement

Grant *HighTech. EV N*∘950525 given by NATO Division of Scientific Affairs at the occasion of Liudmila Pavlotskaya's visit in June 1995, gave the best conditions for completing the final versions of the proof of theorem 3.

References

1. Margenstern M. Décidabilité du problème de l'arrêt pour les machines de Turing non-effaçante sur $\{0,1\}$ et à deux instructions gauches. LITP research report N°94.03 (Institut Blaise Pascal)
2. Margenstern M. Machine de Turing universelle sur $\{0,1\}$ à trois instruc-
. -tions gauches. LITP research report N°93.36 (Institut Blaise Pascal) (1993)
3. Margenstern M. Une machine de Turing universelle sur $\{0,1\}$, non-effaçante et à trois instructions gauches. LITP research report N°94.08 (Institut Blaise Pascal)
4. Margenstern M. Une machine de Turing universelle non-effaçante à exactement trois instructions gauches, C.R. Acad. Sci. Paris, **320-I**, 101-106, (1995)
5. Margenstern M. Non-erasing Turing machines : a new frontier between a decidable halting problem and universality, L.N.C.S, **911**, 386-397, *Proceedings of LATIN*'95 (1995)
6. Margenstern M, Pavlotskaya L. Deux machines de Turing universelles : l'une sur $\{0,1\}$ avec deux instructions gauches, l'autre sur $\{0,1,2\}$ avec une seule instruction gauche, LITP research report N°95/25 (Institut Blaise Pascal)
7. Margenstern M, Pavlotskaya L. Deux machines de Turing universelles à au plus deux instructions gauches, C.R. Acad. Sci. Paris, **320-I**, 1395-1400, (1995)
8. Minsky M.L. Computation : Finite and Infinite Machines. Prentice Hall, Englewood Cliffs, N.J. (1967)
9. Pavlotskaya L.M. : Razreshi- most' problemy ostanovki dlja nekotorykh klassov mashin Tjuringa. Matematicheskie Zametki, **13**, (6), 899-909, Ijun' 1973 899-909. (transl. Solvability of the halting problem for certain classes of Turing machines, Notes of the Acad. Sci. USSR, 13 (6) Nov.1973, 537-541)
10. Rogozhin Ju.V. : Sem' universal'nykh mashin Tjuringa. Matematicheskie Issledovanija, **69**, 76-90, 1982, (Seven universal Turing machines) (in Russian)
11. Rogozhin Ju.V. : On Small Universal Turing Machines, TCS, **168-2**, 276-290, 1996
12. Rogozhin Ju.V. : A universal Turing machine with two symbols and 21 states, private communication.
13. Wang H. Tag Systems and Lag Systems, Math. Annalen, 152, 65-74, (1963)