ELSEVIER

Theoretical Computer Science 143 (1995) 123-135

Theoretical Computer Science

Note

Optimal simulation of two-dimensional alternating finite automata by three-way nondeterministic Turing machines

Akira Ito*, Katsushi Inoue, Itsuo Takanami, Yue Wang

Department of Computer Science and Systems Engineering, Faculty of Engineering, Yamaguchi University, Tokiwadai 2557, Ube 755, Japan

Received April 1994; revised August 1994
Communicated by M. Nivat

Abstract

We show that $n \log n$ space is sufficient for three-way nondeterministic Turing machines (3NTs) to simulate two-dimensional alternating finite automata (AFs), where n is the number of columns of rectangular input tapes. It is already known that $n \log n$ space is necessary for 3NTs to simulate AFs. Thus, our algorithm is optimal in the sense of space complexity point of view.

1. Introduction

Recently, Jiang et al. [7] has shown interesting properties of two-dimensional alternating finite automata (AFs). For example, the class of sets accepted by AFs are not closed under complementation, and two-dimensional alternating finite automata with only universal states (UFs) are not equal to the complements of two-dimensional nondeterministic finite automata (NAs). These results contradict our earlier expectation with usual sense.

In order to draw them out, the same paper reveals the fact that three-way nondeterministic Turing machines (3NTs) require at least $n \log n$ space to simulate UFs, where n is the number of columns of a rectangular input tape.

This paper will show that $n \log n$ space is also sufficient for 3NTs to simulate AFs. Combined with Jiangs' revelation, we can say that $n \log n$ space is necessary and sufficient, thus optimal for 3NTs to simulate AFs and UFs.

* Corresponding author.

SSDI 0304-3975(94)00235-5

2. Preliminaries

In contrast with ordinary one-dimensional Turing machines [2] which are given one-dimensional strings as inputs, a two-dimensional Turing machine [3] is an off-line Turing machines whose input tapes are rectangular in shape. It can move around on the input tape to the left, right, up, or downward, but never falls off the boundary symbols #'s surrounding over the four border edges of the input tape. Well-known concepts on the ordinary automata theory, such as nondeterminism, alternation, one-way movement of input head, space complexity, etc., are naturally extended to the two-dimensional automata, except slight modifications: One-way movement of the input head becomes three-way movement of left, right, and downward directions. Also, two-dimensional space complexity functions here have two variables m and n, which represent the number of rows and columns of the input tapes, respectively.

A *configuration* of two-dimensional alternating Turing machine M on input tape x is a triple $(x,(i,j),(q,\alpha,k))$, where x is a rectangular input tape for M, (i,j) is the position of the input head, q is a state of the finite control, α is the content of the working tape, and k is the position of the working tape head. The *initial configuration* of M on x is $I_M(x)=(x,(1,1),(q_0,\lambda,1))$, where q_0 is the initial state of M and λ denotes the empty string. The set of all possible configurations of M on x is denoted by $C_M(x)$. For two configurations $c,c'\in C_M(x)$, we write $c\vdash_M c'$, if M can go from c to c' in one step, according to the transition rules δ.

A two-dimensional alternating Turing machine M is called $L(m,n)$ *space-bounded* if for each m,n $(m,n\geqslant 1)$ and for each input tape x with m rows and n columns, when M accepts x, there exists an accepting computation tree such that each node labeled with configuration $(x,(i,j),(q,\alpha,k))$ satisfies $|\alpha|\leqslant L(m,n)$. Also, M is called $L(m)$ *space-bounded* if for each m $(m\geqslant 1)$ and for each square input tape x with m rows (and m columns), when M accepts x, there exists an accepting computation tree such that each node labeled with configuration $(x,(i,j),(q,\alpha,k))$ satisfies $|\alpha|\leqslant L(m)$.

A two-dimensional finite automaton can be regarded as a two-dimensional Turing machine whose space complexity function is constantly bounded, i.e., $L(m,n)\leqslant d$ for some constant d. See Fig. 1 for the schematic view of a two-dimensional finite automaton.

Let AF, UF, and 3NT($L(m,n)$) denote a *two-dimensional alternating finite automaton*, a *two-dimensional alternating finite automaton with only universal states*, and a *three-way nondeterministic* $L(m,n)$ *space-bounded two-dimensional Turing machine*, respectively [3–5].

For any family of two-dimensional automata $\mathscr{C}$, the class of sets of rectangular input tapes accepted by $\mathscr{C}$ is denoted by $\mathscr{L}[\mathscr{C}]$ and the class of sets of "square" input tapes accepted by $\mathscr{C}$ is denoted by $\mathscr{L}[\mathscr{C}^s]$, that is, the superscript s indicates the restriction of their input tapes to square ones.

For convenience, we simply denote a configuration of two-dimensional alternating finite automaton M by a triple $(x,q,(i,j))$, where x is the input tape for M, q is a state of the finite control of M, and (i,j) is an input head position.

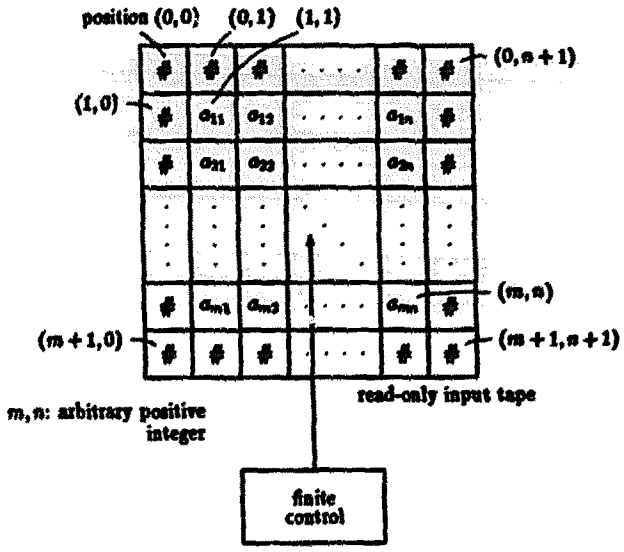

Fig. 1. Two-dimensional finite automaton.

Normally, the acceptance of alternating machines is defined in connection with the existence of an accepting computation tree [3]. It can, however, also be defined by the deterministic procedure given in Fig. 2. This is essentially the same scheme given by Chandra et al. [1].

Let each element of the set $SATURATE(C_M(x), \emptyset)$ after Line (1) of the program *GENERAL* be called *generalized accepting configuration*, or *g.a. configuration* in short. In terms of [4], a generalized accepting configuration is the configuration c such that there exists a c-accepting computation tree (c-accepting computation tree itself is the computation tree whose root is labeled with c and whose leaves are all labeled with accepting configurations).

The set A as an argument of the function *SATURATE* is assumed to be that of configurations which had been already judged as g.a. configurations. In the constraint that the elements of A are fixed, the function *SATURATE* produces additional g.a. configurations only from the target set D, excluding the outer set $C_M(x) - D$, which is beyond D. We also neglect the part $D \cap A$ ($\subseteq A$) of D from the output range since it is already known as g.a. configurations.

The outer loop of the function is repeated until there is no more configuration in D to be judged as g.a. configuration. At each step of the inner for-loop, each configuration in D which has not yet been judged as a g.a. configuration is tested by means of conjunction or disjunction on the previous judgment of its immediate successors. Thus, the algorithm proceeds in the bottom-up manner from the leaves (=accepting configurations) up to the root (=the initial configuration) of the computation tree of M on x.

```
main program GENERAL:
begin
if I_M(x)∈SATURATE(C_M(x),∅) then accept else reject                    (1)
end.

function SATURATE (var D, A: subsets of C_M(x)): a subset of D − A
begin
E := {c | c is an accepting configuration in D ∪ A};
loop do
  Δ := ∅;
  for each c∈D − E do
    if (c is existential and ∃c'∈E[c ⊢_M c'])
       or (c is universal and ∀c'(c ⊢_M c')[c'∈E])
    then Δ := Δ + {c}[1]
  endfor;
  E := E + Δ;
  if Δ = ∅ then return E − A;
endloop
end.
```

Fig. 2.

```
main program MODIFIED:
begin {assume C1 ∪ C2 = C_M(x) & C1 ∩ C2 = ∅}
A := ∅;
repeat
  ΔA1 := SATURATE(C1, A);
  A := A + ΔA1;
  ΔA2 := SATURATE(C2, A);
  A := A + ΔA2
until ΔA2 = ∅;
if I_M(x)∈A then accept else reject
end.
```

Fig. 3.

It should be noted that the final set of generalized accepting configurations will never change even if we evaluate it in any order. In other words, the calculating order is restricted only with the partial order relation $\vdash_M$. Based on this property of g.a. configurations, we can take another strategy. For example, we partition the set $C_M(x)$ into two disjoint parts $C1$ and $C2$ earlier, then alternatively change one part by the other as the scope of the evaluation in the function *SATURATE*, see Fig. 3. It is clear

[1] For two sets A and B, $A + B$ denotes the set $A \cup B - A \cap B$.

that this startegy will finally produce the same set of g.a. configurations as the program *GENERAL*.

3. Results

In that follows, we show that two-dimensional alternating finite automata (AFs) can be simulated by three-way nondeterministic $n \log n$ space-bounded Turing machines (3NT($n \log n$)s), where n is the number of columns of the input tapes. Note that this space bounding function $n \log n$ does not depend on the number m of rows of the given input tapes.

Theorem. $\mathscr{L}[\mathrm{AF}] \subseteq \mathscr{L}[3\mathrm{NT}(n \log n)]$

Proof. Assume that an input tape x with m rows and n columns is given to an AF M. Let Q be the set of states of the finite control of M.

First of all, with such a principle that the bottom area of x is saturated earlier than the top area, we can further modify the program *MODIFIED* described in the preceding section into a "recursive program" as shown in Fig. 4.

---

main program *RECURSIVE*:
global const $C[0..m+1]$: where $C[i] = \{(x,q,(i,j)) \mid q \in Q \ \&\ 0 \leqslant j \leqslant n+1\}$
$(0 \leqslant i \leqslant m+1)$;
global var $A[-1..m+2]$: where $A[i] \subseteq C[i]$ $(0 \leqslant i \leqslant m+1)$ and
$A[-1] = A[m+2] = \emptyset$;
begin
$A[i] := \emptyset$ for each i $(0 \leqslant i \leqslant m+1)$;
ROLLER(0);
if $I_M(x) \in A[1]$ **then** accept **else** reject
end.

procedure *ROLLER*(**var** i: row index of x):
begin
if $i = m+2$ **then return**;
loop do
ROLLER$(i+1)$;
$\Delta Ai :=$ *SATURATE*$(C[i], A[i-1] \cup A[i] \cup A[i+1])$;
if $\Delta Ai = \emptyset$ **then return**;
$A[i] := A[i] + \Delta Ai$
endloop
end.

Fig. 4.

Of course, *SATURATE* is the function described in the preceding section. Note that two configurations $c = (x, q, (i, j))$ and $c' = (x, q', (i', j'))$ satisfy the relation $c \vdash_M c'$ only when $|i - i'| \leqslant 1$. Thus, as actual arguments of *SATURATE*, we have only to employ the subset of A corresponding to the current row $A[i]$ and its two adjacent rows $A[i-1] \cup A[i+1]$, not whole the elements of A.

The program *RECURSIVE* proceeds as if it presses and flattens the ground level by perpendicular moves of a roller with horizontal axis. Fig. 6 illustrates the behaviors of the program for the computation graph shown in Fig. 5, which is derived from the pair of some AF and some x. The reader can see from Fig. 6 that configurations of the bottom area are evaluated as early as possible than those of the top area.

Here, we construct a three-way nondeterministic Turing machine M' that simulates the running process of the deterministic program RECURSIVE, not directly simulating the four-way alternating automaton M itself. Since M' can visit each row only once from the 1st row down to the bottom boundary row of the given input tape x, it must guess the consequences of the entire calls for the subroutine *ROLLER* evoked on the current row, which are not necessarily consecutive events.

In the program presented in Fig. 7, the array AL is used as an argument of *SATURATE*, which corresponds to $A[i-1] \cup A[i] \cup A[i+1]$ of the program RECURSIVE. $\Delta G_k[1]$ is the guessed elements that would be added to $A[i+1]$ by the kth execution of *ROLLER*$(i+1)$. $\Delta A_k[0]$ corresponds to the output ΔAi from the function *SATURATE*. The two variables $\Delta AL0$ and $\Delta G_k[0]$ are used to check the correctness of the guesses. The integer variable *maxc* represents the number of times of calls for the subroutine *ROLLER* on the preceding $(i-1)$st row. $K(Q, n)$ denotes the set $\{(q, j) \mid q \in Q \ \& \ 0 \leqslant j \leqslant n+1\}$ and q_0 is the initial state of M.

When the latest value of the output ΔAi from the function *SATURATE* of the former program *RECURSIVE* is empty (except when the very first call of it on each row), our simulating program stops the honest trace of the further recursion which will be evoked here, since there would be no change in the array variable A. In order to save the wasteful job of this kind, it simply sets $\Delta G_k[1]$ to empty as shown in Line (2) and on the next row it skips the simulations of such calls evoked just above as shown in Line (1). On the ith $(0 \leqslant i \leqslant m)$ row, the correctness of each guess $\Delta G_i[0]$, which had been guessed on the preceding $(i-1)$st row, is checked at Line (4) by the value of $\Delta A_k[0]$ (which has been probably influenced by the recent guess $\Delta G_k[1]$ during the computation of the function *SATURATE*).

On the $(m+1)$st row, we can faithfully fix the value of $\Delta G_k[1]$ to empty set for each k as shown in Line (3), since the recursive call for *ROLLER*$(m+2)$ evoked here immediately returns without anything done. Therefore, the correctness of all the guesses described above is lastly confirmed when M' visits the bottom boundary row.

Table 1 illustrates the process when the key variables of the program are changing their contents in an accepting computation of M' on the computation graph shown in Fig. 6.

It is clear that the program above correctly simulates the program *RECURSIVE*, thus $T(M') = T(M)$.

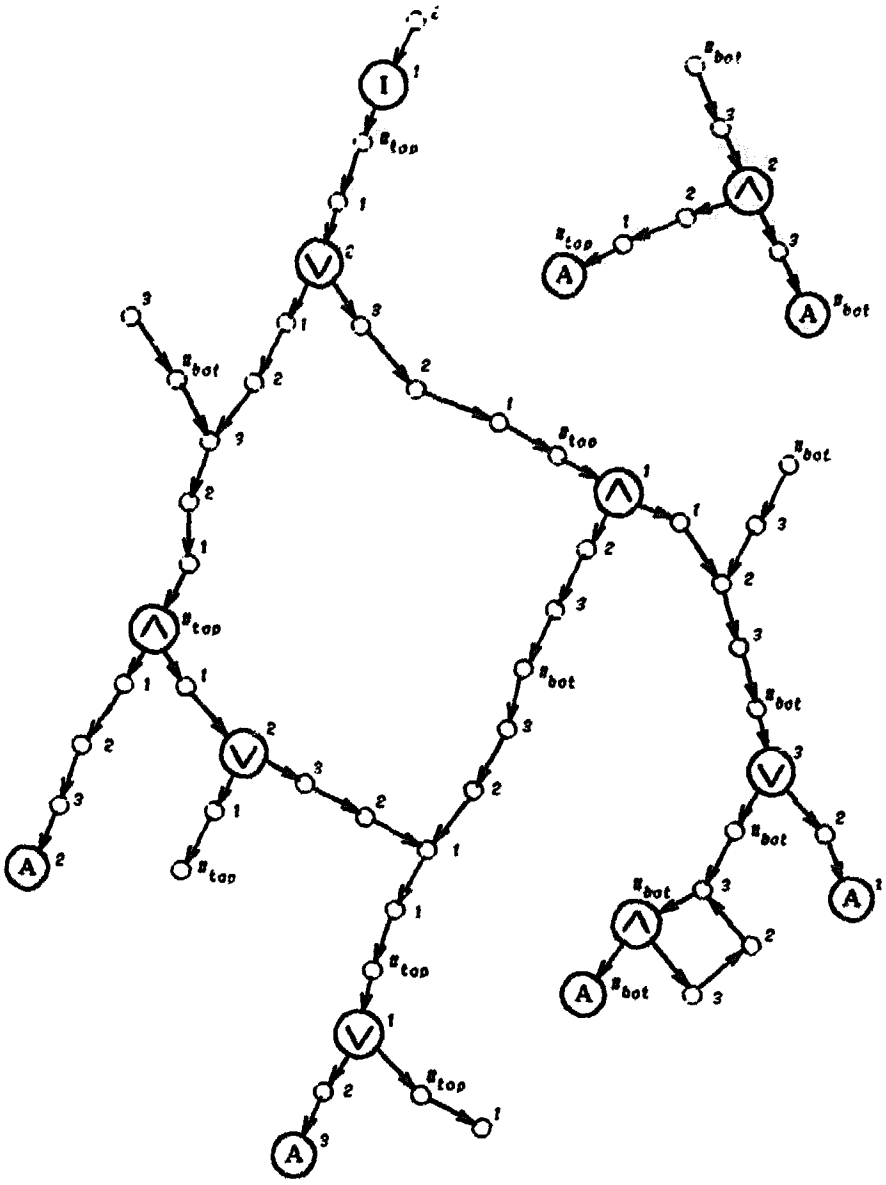

Fig. 5. A computation graph derived from the pair of an alternating machine M and an input tape x with three rows. Here, ①, ⓐ, ⓥ, and Ⓐ denote an initial, universal, existential, and accepting configuration, respectively. Small numbers denote the row number components of configurations of M. $\#_{top}$ ($\#_{bot}$) stands for the top (bottom) boundary row.

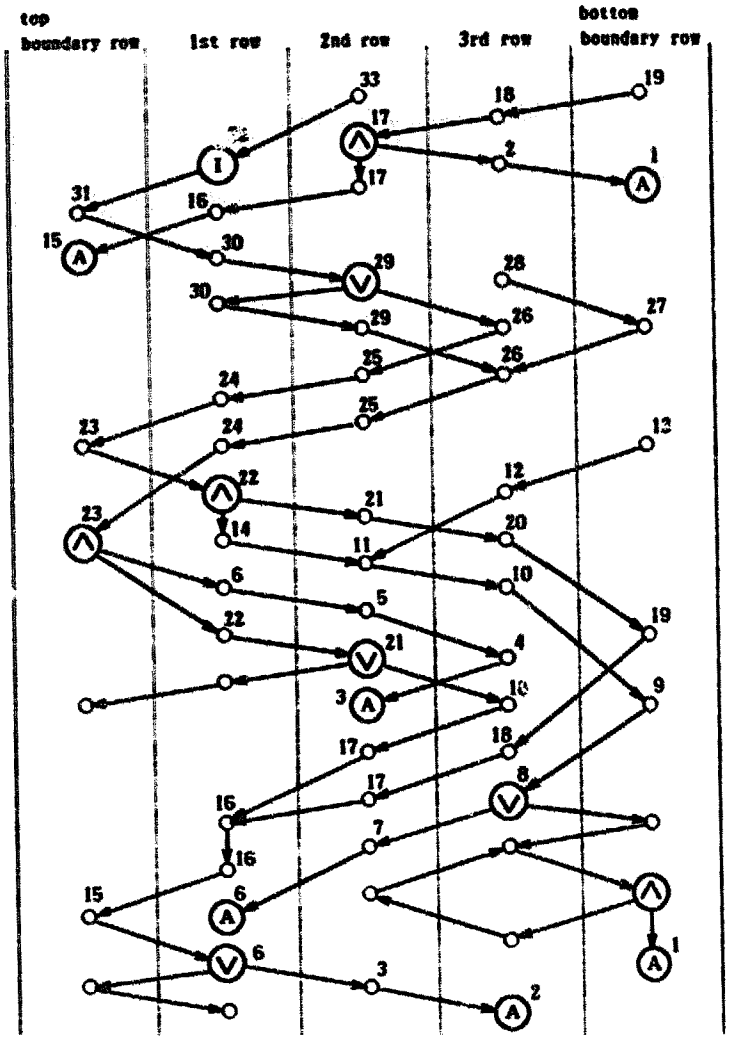

Fig. 6. Behavior of the program *RECURSIVE* on the graph shown in Fig. 5. Script numbers denote the order of configurations when they are judged as generalized accepting configurations.

Now, we consider the amount of space used by M'. For the simplicity of the argument, we distinguish between all empty sets of $\Delta A_k[0]$, $\Delta G_k[1]$, $\Delta A_k[-1]$ and $\Delta G_k[0]$. For instance, we consider them as indexed with integer numbers such as $\emptyset_1, \emptyset_2$, and so on.

First of all, we classify $\bigcup_{k\geq 1}\{\Delta A_k[r]\}$ $(-1 \leqslant r \leqslant 0)$ into two disjoint sets $\Lambda[r]$ and $\Psi[r]$ as follows.

$$\Lambda[r] = \{\Delta A_k[r] \mid \Delta A_k[r] \neq \emptyset\},$$

$$\Psi[r] = \{\Delta A_k[r] \mid \Delta A_k[r] = \emptyset\}.$$

```
main program THREE-WAY_NONDETERMINISTIC:
var AL[−1..1]: where AL[r] ⊆ K(Q,n) (−1 ⩽ r ⩽ 1);
  ΔAL0: where ΔAL0 ⊆ K(Q,n);
  ΔG_k[0..1] (k ⩾ 1): where ΔG_k[r] ⊆ K(Q,n) (0 ⩽ r ⩽ 1);
  ΔA_k[−1..0] (k ⩾ 1): where ΔA_k[r] ⊆ K(Q,n) (−1 ⩽ r ⩽ 0);
begin
maxc := 1;
for i = 0 to m + 1 do
  move to the ith row (when i = 0, assume #'s on the 1st row);
  AL[r] := ∅ for each r(−1 ⩽ r ⩽ 1);
  k := 0;
  for Q = 1 to maxc do
    if not (Q ⩾ 2 and ΔA_{Q−1}[−1] = ∅) then do                          (1)
      ΔAL0 := ∅;
      loop do
        k := k + 1;
        if k ⩾ 2 and ΔA_{k−1}[0] = ∅ then ΔG_k[1] := ∅                   (2)
          else if i = m + 1 then ΔG_k[1] := ∅                           (3)
          else guess ΔG_k[1] ⊆ K(Q,n) − AL[1];
        AL[1] := AL[1] + ΔG_k[1];
        ΔA_k[0] := SATURATE({(x,q,(i,j)) | q∈Q & 0 ⩽ j ⩽ n + 1}, AL);
        if ΔA_k[0] = ∅ then exit loop;
        AL[0] := AL[0] + ΔA_k[0];
        ΔAL0 := ΔAL0 + ΔA_k[0]
      endloop;
      if i ≠ 0 and ΔAL0 ≠ ΔG_i[0] then reject                           (4)
    end if;
    if i ≠ 0 then AL[−1] := AL[−1] + ΔA_i[−1]
  endfor;
  maxc := k;
  if i = 1 and (q0, 1) ∉ AL[0] then reject;
  for k = 1 to maxc do
    ΔA_k[−1] := ΔA_k[0];
    ΔG_k[0] := ΔB_k[1]
  endfor
endfor;
accept
end.
```

Fig. 7.

Table 1
Running process of the program *THREE-WAY_NONDETERMINISTIC* lead to the acceptance.[a]

Top boundary row ($i = 0$)			1st row ($i = 1$)			2nd row ($i = 2$)			3rd row ($i = 3$)			Bottom boundary row ($i = 4$)		
k/l	$\Delta G_k[1]$	$\Delta A_k[0]$	k/l	$\Delta G_k[1]$	$\Delta A_k[0]$	k/l	$\Delta G_k[1]$	$\Delta A_k[0]$	k/l	$\Delta G_k[1]$	$\Delta A_k[0]$	k/l	$\Delta G_k[1]$	$\Delta A_k[0]$
												1/1	0	1
												2/1	0	0
									1/1	1	2	3/2	[0]	0
									2/1	0	0			
						1/1			3/2	[0]	4	4/4	[0]	0
							2	3						
									4/2	0	0			
						2/1	4	5	5/3	[0]	0			
						3/1	0	0						
			1/1			4/2			6/5			5/7	[0]	9
										[0]	8			
												6/7	0	0
							[0]	7						
									7/5	9	10	7/8	[0]	0
									8/5	0	0			
				3, 5	6									
						5/2			9/6			8/10	[0]	13
										[0]	12			
							8, 10	11				9/10	0	0
									10/6	13	0			
						6/2	12	0						
			2/1	7, 11	14	7/3	[0]	0						
			3/1	0	0									

1/1			4/2			8/5			11/9			10/12	[0]	19	
							[0]	17		[0]	18	11/12	0	0	
				[0]	16				12/9	19	20	12/13	[0]	0	
									13/9	0	0				
	6, 14	15				9/5	18, 20	21	14/10	[0]	0				
						10/5	0	0							
			5/2	17, 21	22	11/6	[0]	0							
			6/2	0	0										
2/1			7/3			12/8			15/13			13/16	[0]	27	
										[0]	26	14/16	0	0	
							[0]	25	16/13	27	28	15/17	[0]	0	
				[0]	24				17/13	0	0				
	16, 22	23				13/8	26, 28	29	18/14	[0]	0				
						14/8	0	0							
			8/3	25, 29	30	15/9	[0]	0							
			9/3	0	0										
3/1						16/11	[0]	33	19/17	[0]	0				
			10/4	[0]	32										
	24, 30	31				17/11	0	0							
			11/4	33	0										
4/1	32	0													
maxc = 4			*maxc* = 11			*maxc* = 17			*maxc* = 19			*maxc* = 15			

[a] In the table, each number denotes the configurations which are told off in such a number in Fig. 6. The symbol [0] denotes the empty set generated at Line (2) of the program. Note that the variables $AG_i[1]$ and $AA_i[0]$ become the other variables $AG_i[0]$ and $AA_i[-1]$ in the next row, respectively.

We also classify $\bigcup_{k\geq 1}\{\Delta G_k[r]\}$ $(0 \leq r \leq 1)$ into three disjoint sets $\Gamma[r]$, $\Phi[r]$, and $\Theta[r]$:

$$\Gamma[r] = \{\Delta G_k[r] \mid \Delta G_k[r] \neq \emptyset\},$$

$$\Phi[r] = \{\Delta G_k[r] \mid \Delta G_k[r] = \emptyset \ \& \ \Delta A_{k-1}[r-1] \neq \emptyset\},$$

$$\Theta[r] = \{\Delta G_k[r] \mid \Delta G_k[r] = \emptyset \ \& \ \Delta A_{k-1}[r-1] = \emptyset\}.$$

Note that the elements of $\Theta[1]$ ($\Theta[0]$) correspond to the symbols [ϕ]'s in Table 1.

At the top boundary row (i.e., $i = 0$), it clearly holds that $|\Psi[0]| = 1$, $|\Phi[1]| \leq 1$, and $|\Theta[1]| = 0$. Below, we focus on these quantities $|\Psi[0]|$, $|\Phi[1]|$, and $|\Theta[1]|$ of the general case (i.e., $i > 0$). Define

$$\alpha: \Psi[0] \to \Gamma[0] + \Phi[0] \text{ such that } \alpha(\Delta A_k[0]) = \Delta G_k[0],$$

$$\beta: \Phi[1] \to \Lambda[0] \text{ such that } \beta(\Delta G_k[1]) = \Delta A_{k-1}[0],$$

$$\gamma: \Theta[1] \to \Psi[0] \text{ such that } \gamma(\Delta G_k[1]) = \Delta A_{k-1}[0].$$

Clearly, the mapping α is both one-one and onto and the other mappings β, γ are one-one.

From the injection β, it follows that $|\Phi[1]| \leq |\Lambda[0]| = O(n)$ (so, $|\Phi[0]| = O(n)$). From this and the bijection α, we get $|\Psi[0]| = |\Gamma[0]| + |\Phi[0]| = O(n)$. From this and the injection γ, we also get $|\Theta[1]| \leq |\Psi[0]| = O(n)$.

At this point, we have the upper bound $O(n)$ on the number of empty sets appeared in each variable $\Delta A_k[0]$ and $\Delta G_k[1]$.

Note that, to record a single element of $K(Q, n)$, M needs $O(\log n)$ storage space of the working tape. As the conclusion, we can say that the total space used by M' is bounded by $O(n \log n)$. □

Remark. It should be noted that the square tape version $\mathscr{L}[AF^s] \subseteq \mathscr{L}[3NT^s(m \log m)]$ of our main theorem had been already known: In [6], the following facts are shown.

$$\mathscr{L}[AF^s] \subseteq \mathscr{L}[DCA^s(O(1))] \quad \text{and} \quad \mathscr{L}[CA^s(O(1))] \subseteq \mathscr{L}[3NT^s(m \log m)],$$

where DCA(O(1)) (CA(O(1))) denotes two-dimensional deterministic (nondeterministic) cellular automata with constant state changes per each cell [6]. A simple extension of this to the general rectangular case will induce the relation

$$\mathscr{L}[AF] \subseteq \mathscr{L}[3NT(n \log m + n \log n)],$$

which is, however, weaker than ours.

Fact (Jiang et al. [7] $L(m) = o(m \log m)$, *then* $\mathscr{L}[UF^s] \not\subseteq \mathscr{L}[3NT^s(L(m))]$.

From this (and the obvious fact $\mathscr{L}[UF] \subseteq \mathscr{L}[AF]$), we get the following corollary.

Corollary. *$n \log n$ space is necessary and sufficient for* 3NTs *to stimulate* AFs (UFs).

It is known [7] that n space is necessary and sufficient for 3NTs to stimulate the complements of AFs (the necessity is straightforward).

Acknowledgement

The authors would like to thank an anonymous referee for helpful comments and suggestions.

References

[1] A.K. Chandra, D.C. Kozen and L. Stockmeyer, Alternation, *J. Assoc. Comput. Mach.* **28** (1981) 114–133.

[2] J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation* (Addison Wesley, Reading, MA, 1979).

[3] K. Inoue, I. Takanami and H. Taniguchi, Two-dimensional alternating Turing machines, *Theoret. Comput. Sci.* **27** (1983) 61–83.

[4] A. Ito, K. Inoue and I. Takanami, Deterministic two-dimensional on-line tessellation acceptors are equivalent to two-way two-dimensional alternating finite automata through 180°-rotation, *Theoret. Comput. Sci.* **66** (1989) 273–287.

[5] A. Ito, K. Inoue, I. Takanami and H. Taniguchi, Two-Dimensional alternating Turing machines with only universal states, *Inform. and Contr.* **55** (1982) 193–221.

[6] A. Ito, K. Inoue, I. Takanami and H. Taniguchi, Relationships of the accepting powers between cellular space with bounded number of state-changes and other automata, *Trans. IECE Japan* **J68-D** (1985) 1562–1570 (in Japanese); translation appear in: *Systems Comput. Japan* **17** (1986) 63–72.

[7] T. Jiang, O.H. Ibarra and H. Wang, Some results concerning 2-D on-line tessellation acceptors and 2-D alternating finite automata, *Theoret. Comput. Sci.* **125** (1994) 243–257.