数理解析研究所講究録
591 巻 1986 年 228-236

Low-Level Tradeoffs between Cross And Alternation

岩間一雄

Kazuo Iwama
Kyoto Sangyo University
Kyoto 603

1. Introduction.

What essentially makes alternating Turing machines (ATMs) more efficient than deterministic or nondeternministic Turing machines (DTMs or NTMs) is, as its name suggests, not the introduction of the universal states itself but the alternation between the existential and universal configurations. The number of alternations is therefore quite natural as a measure of computational complexity and actually almost all papers on aternating devices discussed alternation-bounded models [1-7]. However, there have been very few results stating a concrete image of the power of alternation like "what more we get if we allow more alternations". As far as the author knows, the famous hierarchy theorem, $\Sigma_k^p=A\Sigma_k^p$ and $\Pi_k^p=A\Pi_k^p$ for each finite k, is the only one from this point of view.

In this paper we focus on low-level complexity using the single-read/write-tape TMs as the computation model. It is shown that the precise lower bound of the cross complexity for nonregular languages is

$$\log n,\ \log\log n,\ \log\log\log n,\ \ldots,\ \log^{[k]} n,\ \ldots,\ \log^* n$$

if we allow

$$0,\ 1,\ 2,\ \ldots,\ k-1,\ \ldots,\ \text{unbounded}$$

alternations to the device, respectively (0=deterministic and 1=nondeterministic). Thus we have the following trade-off between the cross complexity c(n) and the alternation complexity a(n) for this lower bound

$$\log^* c(n)+a(n)=\log^* n-1$$

under the condition that a(n) grows more slowly than log*n. If $a(n)\geq \log^* n-1$ then $c(n)=\log^* n$.

This trade-off seems beautiful. An weak point might be the popularity and/or the importance of the model and the measure, the single-tape TMs and the cross complexity. The following remarks will compensate hopefully: (i) As for the lower bound of resource usage for nonregular languages, only the cross complexity of the single-tape TMs reflects the distinction among DTMs, NTMs and ATMs. (For example, the lower bound of space usage is loglogn for both NTMs and ATMs. See [2].) Thus inefficientness of the model works well here when discussing this sort of low-level complexity. (ii) The proof of the present trade-off theorem will clarify intuitively the structure how one more alternation gives us one more logarithm, which might be useful for the future stdudy on alternating devices.

Throughout this paper, TMs (DTMs, NTMs and ATMs) mean always those device having a single right-infinite read/write tape. Input strings are given initially on the leftmost portion of the tape. ATMs M have universal and

existential states but no negating states in this paper. We assume that all ATMs in this paper have the universal initial state, but of course the similar result holds for the other type of ATMs. For a given input string x, a computation (tree) p of M on x is the following tree T: (i) T's root is (associated with) the initial configuration of M on x. (ii) If a node v in T is an existential configuration c then v has at most one son among c's proper successors. (iii) If v in T is a universal configuration c then v has the sons corresponding to all of c's successors. The computation p is said to be accepting if its all sons are accepting configurations.

The ith boundary means the boundary between the ith and the (i+1)st cells on the tape. Take some node v of the computation tree p and let r be the path traversing the tree from the root to v. Then it is said that (i) path r (or node v) crosses the ith boundary k times if M's tape head crosses the boundary k times (from left to right for the first time, right to left for the second time and so on) while operating along with the path r, (ii) r (or v) makes k crosses if for any $i \geq 1$, r crosses the ith boundary at most k times, (iii) computation p crosses the ith boundary k times if all the nodes of p crosses the boundary at most k times and (iv) p makes k crosses if all nodes makes at most k crosses. The ATM M is said to be c(n) cross-bounded if for any string of length n that is accepted by M, there is an accepting computation tree that makes at most c(n) crosses.

Also for alternation, we use the similar notations such as "node v makes k alternations", "computation p makes k alternations" and "ATM M is a(n) alternation-bounded". For precise definition of all those notions see e.g., [1,4].

Let n be a positive integer. Then in this paper logn always denotes $|\log_2 n|$ where $|r|$, for real r, denotes the least integer $\geq$ r. Also for each integer $k \geq 2$,

$$\log^{[k]} n = \log(\log^{[k-1]} n)$$

and $\log^{[1]} n = \log n$. Suppose that $\log^{[k]} n = 1$ and $\log^{[k-1]} n > 1$. Then log*n is defined as log*n=k, that means

$$2^{2^{\cdot^{\cdot^{\cdot^{2}}}}} \quad (k \text{ twos}) \quad \geq n.$$

2. Languages accepted with $\log^{[k]} n$ crosses and with k-1 alternations simultaneously

In this section, it is shown that there is a nonregular language that is accepted by some $c\log^{[k]} n+(k-1)$ cross- and k-1 alternation-bounded ATM for any $k \geq 1$ and any small constant c. Let bin(i) denote the string over {0,1} whose reverse is the binary representation of the integer $i \geq 0$ with no leading 0's. However, as a special case, $bin(2^j-1)=11\ldots10$, i.e., we put one leading 0 if the binary number consists of only 1's. For example, bin(6)=011 and bin(15)=11110. Now let

$$BIN=\{\#\} \cup \{\#bin(0)\#bin(1)\#\ldots\#bin(n)\#\mid n \geq 0\}.$$

The idea is that if x is in BIN and $|x|=n$ then the length of each block surrounded by two #'s is at most logn.

We extend this idea one more step. Let PREBIN={x|there is y in {0,1}* such

that xy in BIN} and let $bin^+(m)$ denote the string x such that $|x|=m$ and x is in PREBIN. Now we define DBLBIN as contains all the strings of the following double track structure:

$$\begin{array}{l} \#u_0\#u_1\#\ldots\#u_i\#\ldots\#u_n\# \\ \#v_0\#v_1\#\ldots\#v_i\#\ldots\#v_n\# \end{array}$$

such that (i) $n\geq 0$ and (ii) $u_i=bin(i)$ and $v_i=bin^+(|u_i|)$ for each i. From now on in this section, to avoid useless confusion, we will use the term "tape" for a string with two or more tracks and therefore "string" will be used only for that on some particular track. Similarly for subtapes and substrings. DBLBIN includes, for example,

$$\begin{array}{l} \#0\#1\#01\#110\#001\#101\#\ldots\ldots\#0110101011\# \\ \#\#\#\#\#\#0\#\#0\#\#\#0\#\#\#0\#\#\ldots\ldots\#\#0\#1\#01\#11\# \end{array}$$

It should be noted that there is a cloglogn cross-bounded and 1 alternation-bounded ATM (i.e., having just universal states) that recognizes BIN or the same cross-bounded NTM that recognizes the complement of BIN. The next lemma will illustrate very well how alternation works to save crosses. (The lemma is simply a special case of Lemma 2 but we nevertheless prove it for this reason.)

Lemma 1. There is a cloglogn cross- and 2 alternation-bounded ATM M that recognize DBLBIN for any small constant c. (Recall that "2 alternation-bounded" means M can change from universal configurations to existential ones just once.)

Proof. Recall that M's initial state is universal. At the very beginning M splits universally into three submachines M_0, M_1 and M_2. M_0 checks the syntax of a given input x: (i) Its upper track is in $\{0,1,\#\}^*$ and no two #'s appear consecutively. Furthermore that has to begin with #0#... . Each subtape of x whose upper track is of the form $\#(0+1)^+\#$ will be called a large block. (ii) Every large block B has its lower track of the form $\#\#((0+1)^+\#)^*(0+1)^*\#$ beginning with ##0#... . Again each subtape of x whose lower track is of the form #(0+1)*# except for the leftmost two successive #'s will be called a small block. It should be noted that the small block holds on its lower track #bin(i)# for some i. But the last lower block of each large block B may not be such a complete string, usually #z# where z is a prefix of bin(j) for some j. We will call this last small block a rightmost small block (of B). Obviously M_0 needs just one scan from left to right and needs only universal states (deterministic finite automaton).

Suppose that the input x passed the above syntax test. One can see that x is in DBLBIN if and only if the following conditions (A) and (B) are met. (A) For all large blocks their lower track can be written as $\#bin^+(m)\#$ for some $m\geq 1$. In more detail, for every two consecutive small blocks b_1 and b_2 of every large block B, if b_1 has the lower track that can be written as #bin(i)# for some i and b_2 as #bin(j)# for some j then j=i+1. (This condition will be slighly modified if b_2 is the rightmost small block of B.) (B) For any two consecutive large blocks B_1 and B_2, if B_1 has the upper track that can be written as #bin(i)# for some i and B_2 as #bin(j)# for some j then j=i+1. Rewrite the upper track of B_1 from #bin(i)# into #bin(i+1)#. (Note that this can be done deterministically with one scan from left to right with no overflow. Recall that if $i=2^h-1$ then bin(i) has an extra leading 0.) Then one can see that the

condition above is equivalent to the following: For every pair (b_1,b_2) of small blocks, b_1 is taken from B_1 and b_2 from B_2, (i) b_1 and b_2 have the same upper tracks or (ii) b_1 and b_2 have different lower tracks. Again a slight modification should be made if b_1 and/or b_2 are the rightmost small blocks.

Submachine M_1 is responsible to testing condition (A). While scanning the input from left to right, at each boundary between large bloks, M_1 splits universally into "continue scanning" process and "checking the curreant large block" process. The latter process again splits into "continue scanning" process and "check the current consecutive small blocks" process at each boundary of small blocks. Now suppose that M_1 (the check process) is about to test the consecutive small blocks b_1 and b_2. What M_1 is to do is very simple, rewriting the lower track of subtape b_1b_2 from #bin(i)#bin(j)# into $bin(i+1)$bin(j)$ deterministically with a single scan from left to right. (The symbol $ acts as a delimiter.) If bin(j)=11...10, then the third $ is put on the leading 0, one symbol left to the normal position.

Then M_1 calls a subroutine denoted by EQ(dwn,$-$-$,L) that checks whether the two strings z_1 and z_2 in {0,1}* appearing in the context $\$z_1\$z_2\$$ are the same. The first argument shows that such z_1 and z_2 are on the lower track ("up" if on the upper track) and the third argument shows that z_1 and z_2 exist to the left of the current head position. All the states of subroutine EQ are universal. It checks (i) whether $|z_1|=|z_2|$ and (ii) for every pair (a_1,a_2) of symbols, a_1 in z_1 and a_2 in z_2, if $a_1 \neq a_2$ then the position of a_1 (=the number of symbols between the left $ and a_1) $\neq$ the position of a_2. Both conditions (i) and (ii) can be verified with $\log|z_1|$ crosses.

If b_2 is the rightmost small block (M_1 can tell that fact after scanning b_2) then M_1 calls not EQ but PRE(dwn,$-$-$,L) that tests whether the two strings z_1 and z_2 has the relation $z_1=z_2z_3$ for some z_3 in {0,1}*. What PRE is to do is very close to EQ and may be omitted.

Subroutine M_2 is responsible to condition (B). While scanning the input from left to right, at each boundary between large blocks, M_2 splits universally into "continue scanning" process and "check the current consecutive large blocks" process. M_2 in the latter process rewrites the upper track of the left large block out of the two consecutive ones from #bin(i)# into #bin(i+1)#. While carrying out this rewriting, at each boundary of small blocks, M_2 again splits universally into "continue rewriting" process and "take the left small block" process. The latter process still continues rewriting the upper track and at the same time changes a portion of the tape near the picked small block b_1 (holding, say, bin(j) on the lower track) from

```
.....#--------------------------------#.....
.....#.........#bin(j)#.............#.....
```

into

```
.....#----------$-------$-------------#.....
.....#.........$bin(j)#$.............#.....
```

where --- shows a string without #'s or $'s. Namely, M_2 changes the leftmost symbol of this small block and the leftmost symbol of the next small block on both tracks into $.

Then M_2 just skips the rest of the large block. After entering the next large block, M_2 again splits universally at each boundary of small tracks into

"continue scanning" process and "take the right small block" process. M_2 in the latter process delimits that small block b_2 exactly as b_1 above using \$'s. Then M_2 calls two subroutines EQ(up,\$-\$\$-\$,L) and DIF(dwn,\$-\$\$-\$,L) existentially (recall that M_2 are now testing condition (B)). Subroutine DIF checks whether z_1 and z_2 are different, that can be done with $\log|z_1|$ crosses using just existential states (\$-\$\$-\$ shows that z_1 and z_2 appear in the context $\$z_1\$(0+1+\#)^*\$z_2\$$).

Unfortunately this construction is not sufficient. For M_2 changes its configuration from universal into existential when it calls EQ and DIF, and then M_2 has to change again from existantial to universal in EQ, that is not 2 alternation-bounded but 3. The following modification is enough. Instead of calling EQ and DIF at the same time, M_2 first calls only EQ so that M_2 can continue staying in universal states. Recall the construction of EQ. In EQ if M_2 finds the two symbols a_1 and a_2 such that $a_1 \neq a_2$ and they are at the same position, then M_2 calls DIF(dwn,\$-\$\$-\$,L) or DIF(dwn,\$-\$\$-\$,R) depending on the head position at that moment and this is the only one moment M_2 changes from universal states into existential states.

We need to treat a couple of exceptional cases. Suppose that b_1 is the rightmost small block of the large block B_1. Then in the next large block B_2, M_2 looks at only the rightmost small block as b_2 (or if the taken small block is not the rightmost then M_2 falls accepting states immediately). The way of delimiting is also a little different: If the upper track of the large block B_2 is not of the form 1*0 then the right delimiter \$ is put on # (one symbol left to the normal position). Otherwise the right delimiter is put on the leading 0 (and the same position on the lower track also) in order to neglect that 0 when compairing the two strings on the upper track. Furthermore M_2 calls EQ(up,\$-\$\$-\$,L) and EQ(dwn,\$-\$\$-\$,L) universally. Suppose that b_1 is located right before the rightmost one. Then as b_2, M_2 picks only the small block at the same position (right before the rightmost) in the next large block B_2. Then again M_2 calls two EQ's universally.

Thus we have constructed M as a 2 alternation-bounded ATM. Observe that M_2 can make the maximum number of crosses, that is one by the first scan + those by EQ + those by DIF or

$$2\log|z_1|+1.$$

If the input x ($|x|=n$) is accepted then it is guaranteed that

$$|z_1| < \log\log n.$$

It should also be noted that the subroutines EQ, DIF and PRE can be constructed so as to save the number of crosses by any constant rate if we increase the number of states. Thus we can claim the lemma.

The next lemma is a generalization of Lemma 1. Again the string $\mathrm{bin}^+(m)$ or the language ({#}PREBIN)*{#} that contains such strings as is on the lower track of DBLBIN, plays a key role. In the next lemma, the tape has another two tracks, totally four tracks t_1, t_2, t_3 and t_4 from up to down. Suppose that some track t_i holds a string in ({#}PREBIN)*{#}. Then each subtape of the tape which has a string of the form #(0+1)*# on track t_i is called t_i-block. (If we let the lower track of the double-track-structure tape in Lemma 1 be t_2, then the small block we called before is called t_2-block by the new definition.) Suppose that every t_i-block of the tape has, on track t_j ($t_i \neq t_j$), a string

#bin$^+$(m)# for some m. Then t_j is said to be one level finer than t_i. (In DBLBIN, the lower track is one level finer than the upper track.)

Let QUDBIN be the set of quadruple-track-structure tapes x such that track t_1 of x holds a string in BIN, t_2 is one level finer than t_1, t_3 is one level finer than t_2 (or we may say that t_3 is two levels finer than t_1 if t_2 is clear) and all cells of track t_4 hold the special character & called a blank.

Lemma 2. There is a $\log^{[k]}n+(k-1)$ cross- and (k-1) alternation-bounded ATM T that recognizes QUDBIN for any small constant c.

Proof. Basic idea is the recursive call of the subroutines used in the previous lemma. See [3].

Lemma 3. There is a log*n-c cross- and log*n-c alternation-bounded ATM that recognizes QUDBIN for any positive constant c.

Proof. See [3].

3. Lower bounds.

In this section we prove that the result obtained in the previous section is the best possible for not only the particular languages (DBLBIN and QUDBIN) but also for any nonregular language in the sense that if ATM M can make essentially less crosses or alternations then M can recognize only a regular set. Basic tool in the proof is crossing trees, a generalization of well-known crossing sequences. Before we introduce the crossing trees, we need a couple of preliminalies.

Lemma 4. Suppose that a string x is accepted by k cross- and d alternation-bounded ATM M. Then there exists a (k cross- and d alternation-bounded) accepting computation tree T such that at any node in T if M's head is on the ith cell then $i \leq |x|+g(k)$ where g is some integer function determined by M.

Proof. See [3].

A configuration with cross count is (s,j,x) where s and j are the same as those of the configurations (the current state and head position, respectively). x is called the tape with cross count being of the form

$$x_{ch}(1)x_{cr}(1)x_{ch}(2)x_{cr}(2)\ldots x_{ch}(i)x_{cr}(i)\ldots x_{ch}(m)x_{cr}(m)$$

where $x_{ch}(i)$ shows the symbol on the ith cell of the tape and $x_{cr}(i)$ is the number of times M already crossed the ith boundary. (The "tape" means of course its nonblank portion or $x_{cr}(m-1)\neq 0$ and $x_{cr}(m)=0$.)

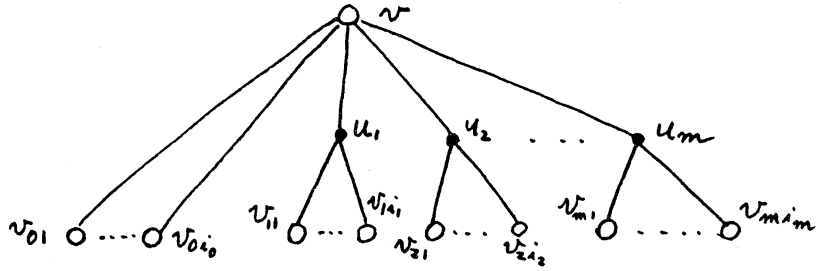
Next we generalize the notion of the computation tree. The computation tree having root (s,j,x) is the same as the normal computation tree except that its root is the explicitly given (not necessarily initial) configuration (s,j,x) and each node including the root is associated with the configuration with cross count.

From now on, unless otherwise stated, a configuration means the configuration with cross count. Let i and k be positive integers and T be the computaion tree having root (s,j,x) such that $x_{cr}(p)\leq k$ for all p and $j\geq i+1$. Then the k cross-bounded cross section of T at the ith boundary is denoted by $SEC_i^k(T)$. $SEC_i^k(T)$ is the set of the configurations c such that (i) c can be reached from the root of T, (ii) all c's ancestors have head position $\geq$ i+1, and (iii) the following (A) or (B) or (C) holds: (A) c's head position is i, namely, just crossed the boundary for the first time since the root of T. (B) c

is an accepting configuration. (C) c is a rejecting configuration including the following cases: (C1) c is not accepting and has no successors. (C2) c crossed some boundary k+1 times. (C3) c's head position goes off the area described in Lemma 4. $SEC_i^k(T)$ is similarly defined if the root's head position j is less than or equal to i.

Recall that the computaion tree is constructed by the principal rule "one son from existential node and all sons from universal node". From now we will call this computation tree the DNF computation tree and introduce the dual one, called the CNF computaion tree, that is constructed by the rule "one son from universal node and all sons from existential node". $SEC_i^k(T)$ is similarly defined for the CNF computation trees T.

Now we are ready to construct the k cross-bounded crossing tree at the ith boundary $CRS_i^k(x_0)$ for an input string x_0. (i) The node set is decomposed into two disjoint subsets called labeled nodes and unlabeled nodes. The labeled nodes have labels of the form (s,cr,y) where s is in the union of M's state set and {A,R} (A and R mean acceptance and rejection, respectively), cr is 0 or 1 and y is right(x) or left(x) (x is the current tape, left(x) is x's prefix until the ith cell and right(x) is the rest of x). (ii) The root is specially labeled with ROOT. (iii) Each labeled node v has its sons like



where $u_1, \ldots, u_m$ are (posibly the empty set of) unlabeled nodes, $v_{01}, \ldots, v_{0i_0}$ (called direct sons), $v_{11}, \ldots, v_{mi_m}$ (called indirect sons) are all labled nodes. When we refer to the level of nodes or the hight of the tree, we count only labled nodes. Therefore if v is at the jth level then all v_{01} through v_{mi_m} are at the (j+1)st level. The root is at the 0th level.

The rule of the construction is as follows. First of all a node becomes a leaf if the first element of the label (state) is A or R. Otherwise suppose that v is at the hth level ($h \leq k$) for some odd h, its label is (s_h, cr_h, y_h) and its father has lebel $(s_{h-1}, cr_{h-1}, y_{h-1})$. Note that y_h is the left half of the tape and y_{h-1} is the right half. Then obtain all DNF computation trees T (if s_h is existential, CNF computation trees otherwise) having root $(s_h, i+1, w_h y_{h-1})$ and all corresponding cross sections $SEC_i^k(T)$. It is very important to observe the following: The "w_h" above should be the left half of the tape at the moment of node v. However, one can notice that if our objective is only to get $SEC_i^k(T)$ then we do not need any information about this w_h since we are interested in only the right side of the boundary at that stage.

Suppose that some $SEC_i^k(T)=\{c\}$ for a single configuraion $c=(s,i,w_h y)$ and c can be reached from the root of T with no alternation. Then we have a direct son v_0 corresponding to this c. Its label is (s,cr,y) where cr=1 iff y has some $y_{cr}(j)=k$. Otherwise all the configurations in $SEC_i^k(T)$ become a set of indirect sons $\{w_{j1}, w_{j2}, \ldots, w_{ji_j}\}$ having a common unlabeled father.

If h=1 (v is at the first level), then we let z=right(x_0). The construction is similar when h is even. If h=0 (v is the root), then we obtain all CNF computation trees having root (s_0,1,left(x_0)) where s_0 is the initial state. That is the construction of $CRS_i^k(x_0)$.

From $CRS_i^k(x_0)$ T we construct the reduced $CRS_i^k(x_0)$, denoted by R-$CRS_i^k(x_0)$, by the following step-by-step modification. R-$CRS_i^k(x_0)$ has two labeling function state(v) and tape(v). To get the final R-$CRS_i^k(x_0)$ S_f, we first let S_0 be the same as T except that if v in T has label (s,cr,y) then in v of S_0, state(v)=(s,cr) and tape(v)={y}. (The reason why tape(v) becomes a set will be given in a moment.) Let the current R-$CRS_i^k(x_0)$ be S. Consider the following two operations: (i) If v_{0i} and v_{oj} are direct sons of some common father v in S and if state(v_{0i})=state(v_{0j}) then merge them into v_0 such that state(v_0)=state(v_{oi}) and tape(v_0)=tape(v_{0i})$\cup$tape(v_{oj}). (The merged node v_0 has the sons of both v_{0i} and v_{oj}.) (ii) For a node v, let subtree(v) denote the subtree of S beginning with node v. Define subtree $=_{st}$ subtree(w) as follows: If v and w are leaves then subtree(v) $=_{st}$ subtree(w) iff state(v)=state(w). Otherwise subtree(v) $=_{st}$ subtree(w) iff state(v)=state(w) and they have the same number of sons v_1, ..., v_j and w_1, ..., w_j for some j such that {subtree(v_1), ..., subtree(v_j)} $=_{st}$ {subtree(w_1), ..., subtree(w_j)}. Now the sedcond operation is as follows: If v and w have a common father and if subtree(v)$=_{st}$subtree(w) then trim subtree(w) and change the labeling function tape into tape(v_j) = tape(v_j) tape(w_j) where v_j and w_j are two same-position nodes of the two subtrees.

It is not difficult to see that R-$CRS_i^k(x)$ is determined uniquely by the input string x, the boundary under the attention and the integer k (upper limit of crosses). We sometimes denote it by R-$CRS_{x;y}^k(xy)$, xy is the input and the boundary is between x and y, if it is more adequate. Now we are ready to show the following two fundamental lemmas on R-$CRS_i^k(x)$. (Proofs are omitted, see [3].)

Lemma 5. R-$CRS_i^k(x)$ with only labeling function state has enough information to decide whether or not x is accepted with k crosses but not k-1.

Lemma 6. If R-$CRS_{x;yz}^k(xyz)$ $=_{st}$ R-$CRS_{xy;z}^k(xyz)$ then R-$CRS_{x;z}^k(xz)$ $=_{st}$ R-$CRS_{x;yz}^k(xyz)$ $=_{st}$ R-$CRS_{xy;z}^k(xyz)$.

Then what we have to do essentially to get the following lemmas is to count the number of different R-$CRS_i^k(x)$'s in the sense of $=_{st}$. (For detail see [3].)

Lemma 7. Let M be f(n) cross-bounded and k alternation-bounded (k is a constant) ATM. Then M can recognize only a regular set if f(n)=o($\log^{[k+1]}$n).

Lemma 8. Let M be f(n) cross-bounded ATM. Then M can recognize only a regular set unless f(n)$\geq$log*n-c for some constant c but for a finite number of n's.

4. Conclusion.

By summing up all eight lemmas we can get our main theorem.

Theorem. The cross/alternation tradeoff described in Section 1 holds regarding their precise lower bound for nonregular languages.

References.

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

[2] Iwama, K., ASPACE(o(loglogn)) is regular, Res. Rept. KSU/ICS 86-01, Institute of Comput. Sci., Kyoto Sangyo Univ., Kyoto, Japan (1986).

[3] Iwama, K., Low-level tradeoffs between cross and alternation, Res. Rept. KSU/ICS 86-02, Institute of Comput. Sci., Kyoto Sangyo Univ., Kyoto, Japan (1986).

[4] Ladner, R.E., Lipton, R.J., and Stockmeyer, L.J., Alternating pushdown and stack automata, SIAM J. Comput., 13,1 (1984) 135-155.

[5] Ladner, R.E., Stockmeyer, L.J., and Lipton, R.J., Alternation bounded auxiliary pushdown automata, Inf. Contr., 62 (1984) 93-108.

[6] Ruzzo, W.L., Tree-size bounded alternation, J. Comput. System Sci., 21 (1980) 218-235.

[7] Volger, H., Turing machines with linear alternation, theories of bounded concatination and the decision problem of first theories, Theoret. Comput. Sci., 23 (1983) 333-337.