comunicaciones técnicas

1980 Serie Naranja: Investigaciones No. 242

Stanislaw Radziszowski*

* Visiting from University of Warsaw

Recibida: 24 de octubre de 1980.

INSTITUTO DE INVESTIGACIONES
EN MATEMATICAS APLICADAS
Y EN SISTEMAS
UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

APARTADO POSTAL 20-726
MEXICO 20, D. F.
548-54-65

CONTENTS

Abstract:

We consider alternating Turing machines with the bound on the number of alternations given by some function. By restricting the class of machines to those operating in polynomial time we obtain the hierarchy of classes between $\bigcup_{n \geq 0} \Sigma_n^p$ (the sum of polynomial-time hierarchy of Stockmeyer) and PSPACE. We exhibit some problems to be complete in a special sense in the class of problems solvable by alternating Turing machines performing at most f(n) alternations. Also conditional inequalities between classes are derived. The second part of the paper relates these results to the measure STA introduced by Berman [2]. Several properties of that measure are presented.

Key words: alternation, nondeterminism, complexity classes, PSPACE, PTIME, bounded alternation, STA measure.

PART I

I. Introduction

Deep conviction and intuition that the class PTIME (problems solvable by deterministic Turing machines running in polynomial time) is properly included in the class PSPACE (problems solvable by Turing machines using polynomial amount of memory) has supported many investigations considering complexity classes probably lying essentially between PTIME and PSPACE. The most known candidate for such a class is NPTIME (the class of problems solvable in polynomial time by nondeterministic Turing machines). We know that:

$$\text{PTIME} \subseteq \text{NPTIME} \subseteq \text{PSPACE}, \qquad (1)$$

but whether these inclusions are proper remains an open question. The first inclusion, called the $P \neq NP$ conjecture, was posed by Cook in [7] and has been studied intensively in [1], [2], [3], [4], [8], [17], [18] among others.

The works of Stockmeyer and Wrathall have provided new insight into the second inclusion. They constructed the hierarchy called "the polynomial-time hierarchy", $\{\Sigma_k^p : k \geq 0\}$, of which the first step Σ_1^p is equal to NPTIME. They proved that this hierarchy has the property:

$$\text{NPTIME} \subseteq \bigcup_{k \geq 1} \Sigma_k^p \subseteq \text{PSPACE}.$$

In the first part of the paper we investigate what probably is a nonempty gap between the sum of Stockmeyer's hierarchy $\bigcup_{k\geq o} \Sigma_k^p$ and PSPACE. Stockmeyer proved that a sufficient condition for the proper inclusion:

$$\bigcup_{k\geq o} \Sigma_k^p \subsetneq \text{PSPACE} \qquad (2)$$

is that the polynomial-time hierarchy is infinite, but we do not know whether this condition is necessary. If the reader is convinced that (2) does not hold (or better still, if he can prove it), we suggest that he stop reading the text at this point.

The heart of Stockmeyer's hierarchy is the finite number of alternating quantifiers leading some relation computable in polynomial time on a deterministic machine. Using the notion of an alternating Turing machine (Chandra and Stockmeyer [6]) we extend this classification by letting the number of alternations to be dynamic and bounded by some function.

On the other hand from [6] we know that alternating Turing machines running in polynomial time (abbreviate ATMpol) recognize exactly the problems lying in PSPACE.

Sections II and III recall the basic facts regarding the polynomial-time hierarchy of Stockmeyer and alternating Turing machines. The next section gives a new proof of the theorem that PSPACE is equal to the class of languages accepted by alternating Turing machines running in polynomial

time (ATMpol). In the proof we use a special kind of reduction that preserves the number of alternations. In the later sections we exploit this technique to show that some problems are complete in the class of languages recognizable by ATMpol with the number of alternations bounded by some function f (n) (abbreviate ASTATE(f(n))). These classes appear to be probable candidates to lie properly between $\bigcup_{k \geqslant o} \Sigma_k^p$ and PSPACE. We present three problems complete in the class ASTATE (f(n)): the first is based on a diagonalization method, the second is a restricted version of quantified Boolean formulas B_ω while the third is a game.

Towards the end of the first part of the paper we sketch a program of further research.

The second part of the work relates these results to the STA measure defined by Berman [2]. The class STA(s(n), t(n), a(n)) denotes the class of languages which can be recognized by ATM with simultaneous bounds on space, time and alternations given by the functions s(n), t(n) and a(n). Several properties of STA measure are derived, among others such as STA(pol,pol,pol) = STA(pol,exp,pol). The classes ASTATE(f(n)) considered in part I appear to be a special case of the STA measure, since we have the equality ASTATE(f(n)) = STA(pol,pol,f(n)).

II. Preliminaries

a) The polynomial-time hierarchy of Stockmeyer.

By the polynomial-time hierarchy we shall mean the following family of classes of languages:

$\{\Sigma_i^p, \Pi_i^p : i \geq 0\}$,

where:

1. $\Sigma_o^p = \Pi_o^p = \text{PTIME}$,

2. $L \in \Sigma_{n+1}^p$ iff there exist a relation $R \in \Pi_n^p$ and a polynomial p, such that:
$x \in L \Leftrightarrow \exists |y| < p(|x|)\ R(x,y)$,

3. $K \in \Pi_{n+1}^p$ iff there exist a relation $S \in \Sigma_n^p$ and a polynomial r, such that:
$x \in K \Leftrightarrow \forall |y| < r(|x|)\ S(x,y)$.

This definition, due to Wrathall [18] differs from the original one, but she proved that they are equivalent. The most important properties of this hierarchy are as follows [17], [18]:

a) $\text{PTIME} \subseteq \Sigma_i^p \subseteq \bigcup_{i \geq 0} \Sigma_i^p \subseteq \text{PSPACE}$,

b) $\Sigma_1^p = \text{NPTIME}$,
$\Pi_1^p = \text{co-NPTIME} = \{L : \bar{L} \in \text{NPTIME}\}$,

c) $\Sigma_n^p \cup \Pi_n^p \subseteq \Sigma_{n+1}^p \cap \Pi_{n+1}^p$ for all $n \geq 0$;

for all $n \geq 0$ the problem whether the inclusion is proper remains open,

d) $\Sigma_i^p = \Sigma_{i+1}^p \Rightarrow (\forall j \geq i)\Sigma_j^p = \Sigma_i^p$;

for any $i \geq 0$ we do not know whether the hypothesis of the implication is true.

We will use the family of problems $\{B_n : n \geq o\}$ and B_ω stricly connected with the polynomial-time hierarchy.

Let us denote:

$X_j = \{x_{j1},\ldots,x_{jk_j}\}$, $1 \leq j \leq n$, the finite sets of Boolean variables,

$QX_j = Qx_{j1}\cdots Qx_{jk_j}$, where Q is the existential quantifier $\exists$ or universal quantifier $\forall$.

Then B_n, B_ω are defined as follows:

$B_n = \{F(X_1,\ldots X_n)$: $F(X_1,\ldots,X_n)$ is the Boolean formula with the set of variables $\cup \{X_i : 1 \leq i \leq n\}$ and $(\exists X_1)(\forall X_2)(\exists X_3)\ldots(Q_n X_n)\ F(X_1,\ldots,X_n) = 1\}$,

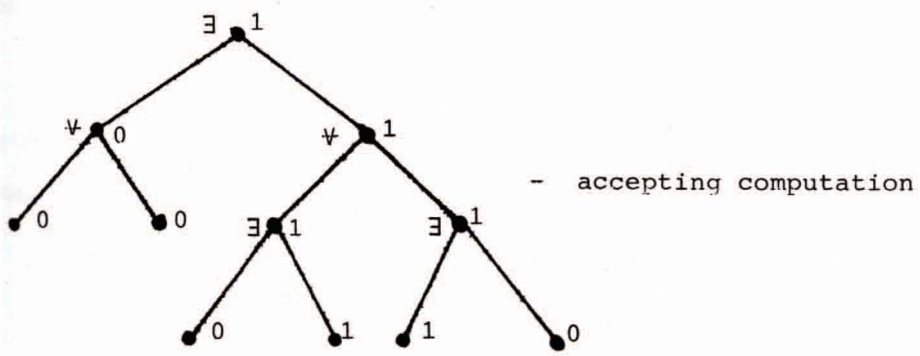
$B_\omega = \bigcup_{n=1}^{\infty} B_n$.

Let us note that B_1 is exactly equal to the NPTIME-complete set of satisfiable Boolean formulas (Cook [7]). In [17] , [18] it is showed that B_ω is log-space complete in PSPACE and B_n is log-space complete in Σ_n^p .

b) Alternation.

By an alternating Turing machine (ATM) we shall mean the machine defined by Chandra and Stockmeyer [6]. The alternating Turing machine is a nondeterministic Turing machine, where the set of states Q is divided into existential states E and universal states U ($Q=E \cup U, E \cap U = \emptyset$) The essential difference between nondeterministic and alternating machines is in the notion of accepting computation. The formal definition is rather complicated, but briefly speaking the procedure for checking if the input x is accepted by an A ϵ ATM is the following:

Since we are dealing only with recursive languages, we can assume without loss of generality, that all computations of A (treated as a nondeterministic machine) for input x are finite. Consider the computation tree of A for x:



- accepting computation

The nodes of the tree represent the configurations of machine A, the root is the initial configuration of A for x.

We call the configuration existential (marked $\exists$) if its state belongs to the set E and universal (marked $\forall$) if its state belongs to U. The sons of a node C are those configurations of A which are direct succesors of C (still A treated as an nondeterministic machine). The leaves of the tree are the terminal configurations of A.

We label all leaves which are accepting configurations by 1, other leaves by 0. Then, proceeding from the bottom to the top, label all nodes of the tree taking Boolean sum (product) of the labels of the node's sons, if the node is an existential (universal) configuration. Finally, x is accepted iff the root of the tree is labeled by 1.

In other words, x is accepted iff in the computation tree of A for x we can choose a subree D, such that all nodes of D are labeled by 1 and the labeling "agrees" with the type of node (existential: one outcoming branch; universal: all outcoming branches are included in the subtree D).

Alternating Turing machine $A \in ATM$ accepts x in fewer than k steps if there exists a subtree D as described above, which has depth no greater than k.

The fundamental theorem [6], which we will use, says that:

$$ATIME(pol) = PSPACE, \qquad (3)$$

where ATIME(pol) denotes the class of languages recognizable by alternating Turing machines running in polynomial time.

Other properties of ATM were studied by Paul and Reischuk [14]. They show that alternation is more powerful than nondeterminism.

In this paper we assume that the initial configuration of an alternating machine is existential.

III. Connections between polynomial-time hierarchy and alternation.

The theorem describing the computational power of alternating Turing machines, ATIME (pol) = PSPACE, is proved in [6] by a direct simulation of a machine operating in polynomial time, and vice versa. We shall prove the same theorem, but using another technique. Our proof uses the PSPACE complete set B_ω - we show directly that it is complete in ATIME(pol) and hence we can conclude the equality (3). The transformation of the proof, apart from providing a better understanding of the theorem, has two advanteges. In the new proof it is easy to count the number of alternations required for the solution of a problem. Furthermore, our proof, after some slight modifications, will permit us to show similar results dealing with classes lying between $\cup\{\Sigma_n^p: n \geq 0\}$ and PSPACE.

Theorem 1:

$$\text{ATIME(pol)} = \text{PSPACE}$$

Proof:

The proof proceeds in two steps. The first, an obvious one, is to show that $B_\omega \in \text{ATIME(pol)}$. In the second it is demonstrated that each language $L \in \text{ATIME(pol)}$ can be reduced in polynomial time to B_ω. Hence we have:

$$L \in \text{PSPACE} \Leftrightarrow L \leq_{\text{pol-time}} B_\omega \Leftrightarrow L \in \text{ATIME(pol)},$$

because both classes PSPACE and ATIME(pol) are closed under

polynomial-time reducibility.

1) $B_\omega \in ATIME(pol)$.

Consider the following alternating program:

1. Given Boolean formula α of length n determine the number k - the number of changes of the type of quantifier;
2. Q: = E ; i:=1;
3. if Q=E then existentially execute for all $x \in X_i$ {x:=0,x:=1};
4. if Q=U then universally execute for all $x \in X_i$ {x:=0,x:=1};
5. if i<k then
   (i:=i+1; Q:=if Q=E then U else E; go to 3);
6. if $\alpha(X_1,\ldots,X_k) = 1$ then ACCEPT else REJECT;

The above program can be easily rewritten into a formal alternating Turing machine running in time $0(n^2)$. The dominant time factor is produced by instruction 6.The correctness of the algorithm follows directly from the definition of an accepting computation of ATM.

2) B_ω is polynomial-time complete in ATIME(pol).

Let us take an arbitrary alternating Turing machine $M \in$ ATMpol running in time p(n) for some polynomial p. On the accepted input x of length n, machine M will execute no more than t=p(n) steps. Denote by L the language accepted by machine M. In polynomial time we will deterministically produce for each x a Boolean formula α_x,

such that:

$$x \in L \Leftrightarrow \alpha_x \in B_\omega.$$

The construction of α_x requires the following lemma:

Lemma:

The relation $x \in L$ can be defined by the formula of the form:

$$\exists y_1 \forall y_2 \exists y_3 \cdots Q_{y_t} \; R(x, y_1, \ldots, y_t), \qquad (4)$$

where:

$y_i \in \bigcup_{j=1}^{t} \{1,\ldots,m\}^j$ for $i=1,\ldots,t$ and m - the range of nondeterminism of machine M,

$R(x,y_1,\ldots,y_t) \in$ PTIME.

Remark 1:

Although formula (4) can seem to be the same as that considered by Chandra and Stockmeyer [6] in the proof of (3), there is an essential difference. They are dealing only with standard alternating Turing machines, which change the type of the state on every move. In such a model the number of alternations is always proportional to the running time of the machine. In our model, the machines alternate only at some points of the computation. Moreover, the changes of quantifiers are not synchronized on the levels of the computation tree. In such a manner the number of alternations can reflect some aspects of the internal complexity of the problem solved on an alternating Turing machine.

Proof of the lemma:

The y_i's represent a finite path in the computation tree of M for x, such that all configurations on y_i are of the same type (E or U) and y_i is maximal, i.e. it cannot be extended without changing the type of configuration. Hence all possible codes of y_i are contained in the set $\bigcup_{j=1}^{t} \{1,\ldots,m\}^j$.

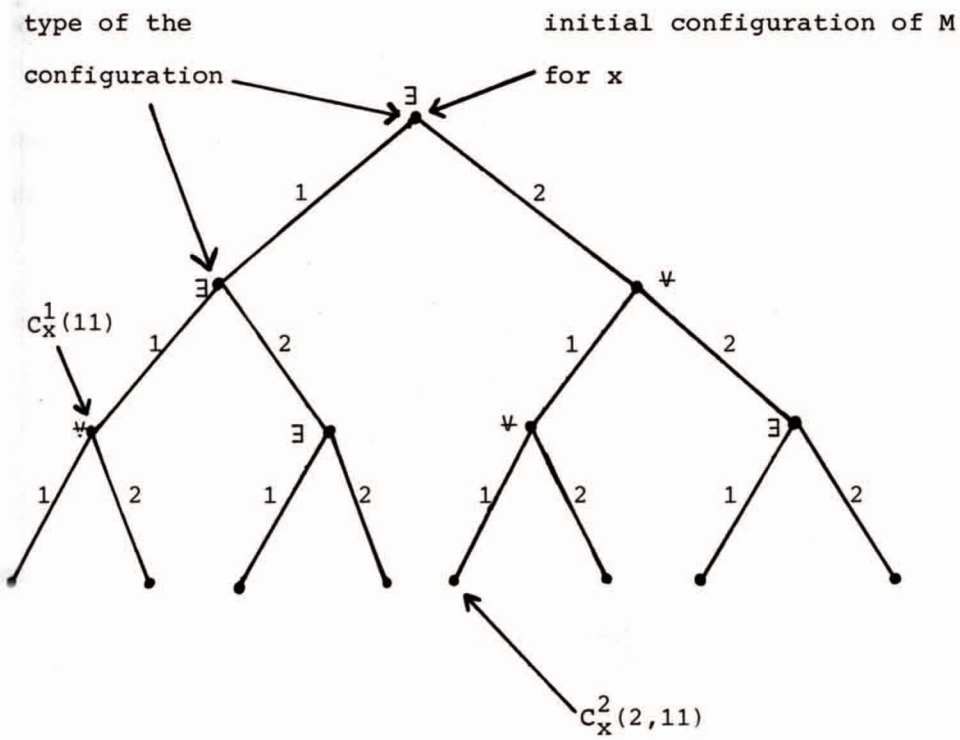
Given a sequence of y_i's we can easily check in polynomial time whether their concatenation $y_1y_2\ldots y_j$ is admissible, that means all y_i's are maximal when passing the computation tree along the path $y_1\ldots y_j$. Hence the following functions are computable in a common polynomial time bound:

$$p_x^j(y_1,\ldots,y_j) = \begin{cases} \text{true} & \text{if } y_1y_2\ldots y_j \text{ is admissible,} \\ \text{false} & \text{otherwise,} \end{cases}$$

$$c_x^j(y_1,\ldots,y_j) = \begin{cases} \text{configuration of M which is computed from the initial configuration for x along the path } y_1\ldots y_j & \text{if } y_1y_2\ldots y_j \text{ is admissible,} \\ \text{undefined} & \text{otherwise,} \end{cases}$$

for j=1,...,t.

For example, consider the computation tree of M for x of the form:



We have:

$$y_i \in \bigcup_{j=1}^{3} \{1,2\}^j,\ t=3,\ m=2,$$

$$p_x^1(11) = p_x^2(11,1) = p_x^1(121) = p_x^2(2,11) = \text{true},$$

$$p_x^2(1,1) = p_x^2(21,1) = p_x^1(22) = \text{false}.$$

Let us denote:

$$ACC_x^j(y_1,\ldots,y_t) = \begin{cases} \text{true} & \text{if } C_x^j(y_1,\ldots y_j) \text{ is an accepting configuration} \\ \text{false} & \text{otherwise,} \end{cases}$$

$$p_x^i(y_1,\ldots,y_t)=\begin{cases} p_x^i(y_1,..,y_i)\wedge(ACC_x^i(y_1,\ldots,y_t)\vee \\ \neg\, p_x^{i+1}(y_1,\ldots,y_{i+1}) & \text{if } i \text{ is odd,} \\ p_x^i(y_1,\ldots,y_i)\wedge ACC_x^i(y_1,\ldots,y_t) & \text{if } i \text{ is even.} \end{cases}$$

Now by the induction on $t\geq 1$ we can prove that R can be written as:

$$\bigvee_{i=1}^{t} p_x^i(y_1,\ldots,y_t) \qquad \text{if } t \text{ is even,} \quad (5)$$

and

$$\bigvee_{i=1}^{t-1} p_x^i(y_1,\ldots,y_t) \vee (p_x^t(y_1,\ldots,y_t)\wedge ACC_x^t(y_1,\ldots,y_t)) \qquad \text{if } t \text{ is odd.} \quad (6)$$

For $t=1,2$ the formulas (6) and (5) have the form $(p_x^1 \wedge c_x^1)$ and $p_x^1 \wedge (c_x^1 \vee \neg p_x^2) \vee (p_x^2 \wedge c_x^2)$ respectively $(c_x^i = ACC_x^i)$. Then the formulas:

$$(\exists y_1)\ p_x^1 \wedge c_x^1, \quad (7)$$

$$(\exists y_1)(\forall_{y_2})\ (p_x^1 \wedge (c_x^1 \vee \neg p_x^2)) \vee (p_x^2 \wedge c_x^2), \quad (8)$$

define the relation $x \in L$ for $t=1,2$. Obviously, (7) deals with an ordinary nondeterministic computation. In (8) the first part says that it is possible to accept x without alternation and formula $(p_x^2 \wedge c_x^2)$ describes a possible accepting computation with one change of quantifier.

Generalizing to an arbitrary $t \geq 1$, formulas (5) and (6) preceded by a suitable prefix of quantifiers describe the possibility that $x \in L$ is accepted in $i=0,1,\ldots,t-1$ alternations.

This completes the proof of the lemma, since formulas (5) and (6) are computable from $x, y_1, \ldots, y_t$ in a common polynomial time bound. □

Remark 2:

Now, it would be possilbe to transform R directly to some Boolean formula α, such that $R \equiv (\exists S)\alpha$, using a well known method for expressing the next-move relation by a propositional formula (where S is a set of additional Boolean variables). However we are interested in eliminating the quantifier $(\exists S)$, therefore we proceed in another way. □

In the final part of the proof of theorem 1 we shall use the circuit value problem CVP which is log-space complete in PTIME (Ladner [13]). In the circuit value problem, we are given a list L of assignments to $C_1, \ldots, C_n$ of the form:

$C_i := 0$,

$C_i := 1$,

$C_i := C_j \wedge C_k$, $j,k<i$

$C_i := C_j \vee C_k$, $j,k<i$

$C_i := \neg C_j$, $j<i$,

such that each C_i occurs on the left exactly once. The problem is to determine the final Boolean value of C_n.

Let $\bar{x}$, $\bar{y}_i$ denote the binary code of x, y_i. Treating $\bar{x}$ and $\bar{y}_i$'s as sets of propositional variables and using the completeness of CVP in PTIME, we can transform, in polynomial time, the relation R to some instance B of CVP, such that:

$$R(x,y_1,\ldots,y_t) \Leftrightarrow B(\bar{x}, \bar{y}_1,\ldots,\bar{y}_t)=1, \qquad (9)$$

where $\bar{x}$, $\bar{y}_2,\ldots,\bar{y}_t$ occur in B only as some assignments of the type $C_i:=0$ or $C_i:=1$. These initial values of C_i encode the arguments of R, while the structure of R is reflected in the graph of the circuit B.

Consider two formulas:

$$\alpha_1 = l_1 \wedge l_2 \ \ldots \ \wedge l_n \Rightarrow C_n,$$

$$\alpha_2 = l_1 \wedge l_2 \ \ldots \ \wedge l_n \wedge C_n,$$

where:

$$l_i = \begin{cases} C_i \leftrightarrow 0 & \text{if } C_i := 0 \in \llcorner \\ C_i \leftrightarrow 1 & \text{if } C_i := 1 \in \llcorner \\ C_i \leftrightarrow C_j \wedge C_k & \text{if } C_i := C_j \wedge C_k \in \llcorner \\ C_i \leftrightarrow C_j \vee C_k & \text{if } C_i := C_j \vee C_k \in \llcorner \\ C_i \leftrightarrow \neg C_j & \text{if } C_i := \neg C_j \in \llcorner \end{cases}$$

It is easy to note, that for given $R(x, y_1, \ldots, y_t)$:

$$R(x,y_1,\ldots,y_t) \Leftrightarrow (\forall S)\, \alpha_1(S) \Leftrightarrow (\exists S)\, \alpha_2(S), \qquad (10)$$

where the set of Boolean variables S is defined as follows:

$$S = \{C_i : l_i \text{ is not of the form } C_i \leftrightarrow 0 \text{ or } C_i \leftrightarrow 1\}.$$

The set S is composed from those variables whose values are not coded directly by $R(x,y_1,\ldots,y_t)$.

Finally we can define formula α_x as:

$$\alpha_x = \begin{cases} \alpha_1(\bar{x},\bar{y}_1,\ldots,(\bar{y}_t \cup S)) & \text{if } y_t \text{ is bounded by } \forall \\ \alpha_2(\bar{x},\bar{y}_1,\ldots,(\bar{y}_t \cup S)) & \text{if } y_t \text{ is bounded by } \exists \end{cases}$$

Obviously α_x can be determined in polynomial time. Taking into consideration (9) and (10) this completes the proof of the theorem.

□

The proof of the completeness of B_ω in ATIME(pol) may appear to be rather unnecessarily complicated, but that technique was used to obtain the following corollary:

Corollary 1:

The transformation from the proof of theorem 1 preserves the number of alternations. That is to say, the alternating Turing machine $M \in ATMpol$ making t alternations on input x can be reduced in polynomial time to an instance of B_ω with t alternations of quantifiers.

The resulting formula α_x is a propositional formula, but not in CNF-form (conjunctive-normal form). On the other hand, using standard techniques for describing polynomial-time computations with Boolean formulas we obtain CNF - formula with leading existential quantifier. An interesting open question arises, whether we can prove theorem 1 using formulas from $B_\omega \cap$ CNF, but in such a manner, that the corollary 1 holds.

IV. Bounded alternation - preliminaries

An alternating Turing machine A ϵ ATMpol accepts an input x in k changes of the state type (alternations), if it is possible to choose an accepting subtree D of the computation tree D' of machine A for input x, such that for all paths from the root to leaves in D the number of changes of the type of configuration (existential $\leftrightarrow$ universal) does not exceed k.

Definition

a) Machine A ϵ AYMpol has the type-state complexity f(n) iff A accepts x in f(n) changes of the state type (alternations), for all accepted inputs x of length n.

b) ASTATE(f(n)) is the class of languages, such that:
L ϵ ASTATE(f(n)) iff there exists an alternating Turing machine A ϵ ATMpol, such that A accepts L and A has state type complexity f(n).

□

From the definition we can immediately obtain the following properties of the classes ASTATE(f(n)):

1. $f(n) \leq g(n) \Rightarrow ASTATE(f(n)) \subseteq ASTATE(g(n))$,
   - directly from the definition.
2. If f(n) is monotonic and not bounded by any polynomial then ASTATE(f(n)) = PSPACE,

- from the assumption we are restricted to the class of ATMpol, hence with the exception of finite interval, every machine from ATMpol has state type complexity f(n). Then ASTATE(f(n)) = ATIME(pol), which is equal to PSPACE (theorem 1).

3. ASTATE(0) = Σ_1^p,

- from the assumption the initial configuration of every machine A ϵ ATM(pol) is existential. Then the class ASTATE(0) is equal to the class of languages recognizable by nondeterministic Turing machines running in polynomial time.

Property 3 can be easy generalized to:

4. Σ_k^p = ASTATE(k-1) for $k \geq 1$,

Proof:

$\supseteq$: Repeat exactly the proof of theorem 1 with t=const=k in all places where t denotes the length of the alternating sequence $y_1,\ldots,y_t$.
Note that corollary 1 is important here.

$\subseteq$: Take arbitrary $L \epsilon \Sigma_k^p$. Then there exists a relation $R \epsilon$ PTIME, such that:

$$x \epsilon L \Leftrightarrow \exists y_1 \forall y_2 \ldots Q_{y_k} R(x,y_1,\ldots,y_k),$$

where the range of quantifiers is restricted to y_i's of length bounded by some polynomial $p(|x|)$. The language L is accepted by the following alternating

uring machine A ϵ ATMpol with state type complexity k-1):

1) t:=p(|x|); i:=1; S:=E;

2) while i$\leq$k do

begin

if S=E then existentially construct y_i of length t;

if S=U then universally construct y_i of length t;

if S=E then S:=U else S:=E; i:=i+1

end;

3) if R(x,y_1,...,y_k) then ACCEPT else REJECT;

The above description of the alternating algorithm can easily be transformed to a machine A ϵ ATMpol with state type complexity (k-1). □

Now, using the properties of Stockmeyer's polynomial-time hierarchy, let us note that:

5. ASTATE(i) = ASTATE(i+1) $\Rightarrow$ ($\forall$j)(j$\geq$i $\Rightarrow$ ASTATE(j)=ASTATE(i)),
   -directly from d) section IIa).

6. $ASTATE(const) \stackrel{df}{=} \bigcup_{i=0}^{\infty} ASTATE(i) = \bigcup_{i=0}^{\infty} \Sigma_k^p$,
   - from 4 this section.

7. $B_\omega \in ASTATE(const) \Leftrightarrow PSPACE = \bigcup_{k=0}^{\infty} \Sigma_k^p$,
   - from 6 this section, using the fact that Σ_k^p is closed under log-space reducibility and from log-space completeness of Bω in PSPACE.

8. $B_\omega \in$ ASTATE(cn/log n) for some c>0,

- from the syntax of Boolean formula we can easy derive that the number of alternations of quantifiers is bounded by n/log n, then we use the alternating machine from section III/1 for k = n/log n.

9. For any unbounded monotonic function f(n):

$$\bigcup_{k\geq 0} \Sigma_k^p \subseteq \text{ASTATE}(f(n)) \subseteq \text{PSPACE},$$

- first inclusion follows from properties 6 and 1, the second one from properties 2 and 1 of this section. For any function f(n) both inclusions are open problems when we ask if they are proper.

□

In the following section we shall investigate the classes ASTATE(f(n)) for certain functions f(n). All results are conditional, in the sense that, if PTIME = PSPACE or $\bigcup_{k>0} \Sigma_n^p$ = PSPACE, then all the hierarchy vanishes, but it seems as implausible as the equality PTIME = PSPACE or PTIME = NPTIME.

V. Bounded alternation - completeness

The function f is regular if it is an unbounded, monotonic, polynomial-time computable function from positive integers to positive integers. In this section we shall investigate classes of problems solvable by alternating Turing machines with the state type complexity f(n), where f is any regular function. In general, we do not know whether the class ASTATE(f(n)) is closed under polynomial-time Turing reducibility.

We will say that a reduction r of the class of machines $C \subseteq$ ATMpol to some problem P preserves the number of alternations if:

1) r is a reduction, i.e. for every input x and machine $M \in C$

   x is accepted by M $\Leftrightarrow$ $p=r(x,M) \in P$

2) There exists a machine $K \in$ ATMpol recognizing P, such that the number of alternations made by K on an input p=r(x,M) is not greater than the number of alternations made by machine M on input x.

Further, the problem P is polynomial-time complete in the class C preserving state type complexity if P can be recognized by some machine from the class C and there exists a reduction from C to P computable in polynomial time and preserving the number of alternations.

In the sequel the class of languages ASTATE(f(n)) will be identified with the class of appropriate machines, in this case with the class of machines ATMpol with state type complexity $f(n)$.

Using the method of Baker, Gill and Solovay [1] we prove the following existential theorem:

Theorem 2:

There exists a polynomial-time complete problem in the class ASTATE(f(n)) preserving state type complexity.

Proof:

Every alternating Turing machine can be converted to a machine from the class ATMpol, and with at most f(n) changes of the type of state, by attaching two clocks terminating computations, if time or number of alternations exceeds some predetermined value. We can thereby produce a list of machines accepting all languages from the class ASTATE (f(n)) and such, that each machine from the class ATMpol with state type complexity f(n) is represented in the list by some machine from the class ATMpol with the same state type complexity.

We will denote by A_i the i-th machine from the list and, without loss of generality, we can assume that $p_i(n)$ is a strict upper bound on the running time of A_i, where $p_i(n)= i+n^i$. As usual, the binary code of a sequence $x_1,\ldots, x_k$ will be denoted by $\langle x_1,\ldots,x_k\rangle$.

Consider the set of binary strings:
$K = \{\langle i,x,0^n\rangle : A_i$ accepts x in time not greater than $n\}$.
We shall prove that the set K is polynomial-time complete in ASTATE(f(n)) and the reduction preserves the number of

alternations.

First note, that $K \in ASTATE(f(n))$. An alternating machine $M \in ATMpol$ recognizes K by constructing and simulating machine A_i for n steps. Obviously M has the same state type complexity as A_i, that means f(n).

Let S be an arbitrary language from the class ASTATE(f(n)) Then the set S is accepted by some machine A_i. The reduction is given by the function g computable in polynomial time:

$$g(x) = \langle i, x, 0^{p_i(|x|)} \rangle .$$

Hence we have:

$$\begin{aligned} x \in S &\Leftrightarrow A_i \text{ accepts } x \Leftrightarrow \\ &\Leftrightarrow A_i \text{ accepts } x \text{ in time } \leq p_i(|x|) \text{ and } f(|x|) \text{ alternations} \\ &\Leftrightarrow \langle i, x, 0^{p_i(|x|)} \rangle \in K. \end{aligned}$$

Finally, note that the number of alternations after the reduction remains exactly the same.

□

Now we can conclude the following corollary:

Corollary 2:

If the polynomial-time hierarchy of Stockmeyer $\{\Sigma_i^p : i \geq 0\}$ is infinite then:

$$\bigcup_{k \geq 0} \Sigma_k^p \subsetneq ASTATE(f(n)).$$

Proof:

From property 9 section IV we know that:

$$\bigcup_{k\geq 0} \Sigma_k^p \subseteq \text{ASTATE}(f(n)).$$

Assume $\{\Sigma_k^p : k\geq 0\}$ is infinite and $\text{ASTATE}(f(n)) = \bigcup_{k\geq 0} \Sigma_k^p$.

Consider the set K from the proof of the previous theorem. There has to exist $i\geq 0$, such that $K \in \Sigma_i^p$. The class Σ_i^p is closed under polynomial-time reducibility ([17],[18]). Thus from the completeness of K in $\text{ASTATE}(f(n))$ we receive:

$$\bigcup_{k\geq 0} \Sigma_k^p = \text{ASTATE}(f(n)) \subseteq \Sigma_i^p,$$

which contradicts with the infinity of the family $\{\Sigma_k^p : k\geq 0\}$

□

It would be quite interesting to find a regular function f bounded by some polynomial, such that there exists a set S complete in $\text{ASTATE}(f(n))$ and the reduction decreases the number of alternations. However, under weaker condition, i.e. function $f(n) = \log n$, we can formulate the following corollary:

Corollary 3:

If $B_\omega \in \text{ASTATE}(\log n)$ then $\text{PSPACE} \subseteq \bigcup_{k\geq 0} \text{ASTATE}(c\cdot\log n)$

Proof:

Take an arbitrary language $L \in$ PSPACE and transform it to $B\omega$ in polynomial time $p(n)$. The resulting formula α has length not greater than $p(n)$, so the number of alternations in α is bounded by $p(n)$. From the assumption that $B_\omega \in$

ASTATE(log n) we can decide whether $\alpha \in B\omega$ by alternating Turing machine making $\log(p(n))$ alternations. But there exists some constant c, such that $\log(p(n)) \leq c \cdot \log n$, so L can be recognized in $c \cdot \log n$ alternations. □

The ASTATE(f(n)) - complete problem K exhibited in theorem 2 is somewhat sophisticated. On the basis of the set Bω we can find a more natural problem complete in the class ASTATE(f(n)). Consider the set Bω restricted to these formulas, in which the number of alternations of quantifiers is bounded by f(n), where n is the length of the formula. Denote this set by $B_\omega/f(n)$.

Theorem 3:

The set $B_\omega/f(n)$ is polynomial-time complete in ASTATE(f(n)) and the reduction preserves the state type complexity.

Proof:

Repeat exactly the reduction to the set Bω from the proof of theorem 1, replacing $t=p(n)$ by $t=f(n)$ in all places where t denotes the length of an alternating sequence $y_1,\ldots,y_t$. □

As an application of theorem 3 we can derive:

Corollary 4:

$B_\omega/\log n \in$ PTIME $\Leftrightarrow$ ASTATE(log n) $\subseteq$ PTIME

Proof:

Immediate from theorem 3. □

Using once more the proof of theorem 1 we will establish another problem complete in the class ASTATE(f(n)). Let us recall the circuit value problem CVP which is log-space complete in PTIME. Consider the following two-person game:

Game description (CVP(f))

Input is an instance of CVP problem restricted to a list of assignments of the form:

$$C_i := C_j \vee C_k, \quad j,k<i,$$
$$C_i := C_j \wedge C_k, \quad j,k<i,$$
$$C_i := \neg C_j, \quad j<i, \quad i \leq n$$

with the sequence of positive integers $(a_1,\ldots,a_f)$, where $f=f(n)$ (notice that there are no assignments $C_i:=0$ and $C_i:=1$). There are two players, the player I starts. Move r consists of putting 0 or 1 to the first a_r not yet assigned variables C_i. If there is an insufficient number of unassigned variables then the player who is to move, puts some values to the remaining variables and the gaem is finished.

The player I wins iff the complete assingment which has been produced at some move of the game makes the value of CVP true.

Theorem 4:

The problem of deciding, for any given input of the CVP(f) game, whether the first player has a winning strategy is polyno-

mial-time complete in ASTATE(f(n)) and the reduction preserves the number of alternations.

Proof:

Similar to the proof of the theorem 3.

Remark 3:

The game described above with the sequence of integers $(a_1,\ldots,a_n)$, $a_i=1$ for $i=1,\ldots,n$, is similar to the game G_ω(CNF) considered in [15] and the CVP-game in this case is PSPACE-complete as many other games described in [15] and [9]. So, the interesting case is, for example, to consider the game on CVP for the function f(n) equal to log n

Finally, we shall present one more conditional corollary:

Corollary 5:

If there exists a problem P polynomial-time complete in PSPACE, such that $P \in ASTATE(f(n))$ then:

$$PSPACE \subseteq \bigcup_{pol} ASTATE(f(pol)),$$

In particular if $f(n) = (\log n)^k$ then:

$$PSPACE \subseteq \bigcup_{c\geq 0} ASTATE(c(\log n)^k) \quad \text{for } k\geqslant 1.$$

Proof:

An arbitrary language $L \in$ PSPACE can be transformed in

the polynomial time p(m) to the problem P and recognized by some machine from the class ASTATE (f(n)), where n=p(m)

□

VI. Final remarks

The obvious generalization of the problems considered in the previous sections is to define the hierarchies:

$$A_k^1 = \mathrm{ASTATE}(\log n + k), \qquad \{A_k^1 : k \geqslant 0\},$$

$$A_k^2 = \mathrm{ASTATE}(k \cdot \log n), \qquad \{A_k^2 : k \geqslant 0\},$$

$$A_k^3 = \mathrm{ASTATE}((\log n)^k) \qquad \{A_k^3 : k \geqslant 0\},$$

and to denote:

$$\mathrm{ASTATE}(\log n + \mathrm{const}) = \bigcup_{k \geqslant 0} A_k^1,$$

$$\mathrm{ASTATE}(\mathrm{clog}\ n) = \bigcup_{k \geqslant 0} A_k^2,$$

$$\mathrm{ASTATE}(\mathrm{pol}\ (\log)) = \bigcup_{k \geqslant 0} A_k^3.$$

From property 1 of section IV, we know that the hierarchies $\{A_k^i : k \geqslant 0\}$, $i=1,2,3$ form chains. The function $f(n)$ used in the previous sections can be equal to any of the functions appearing here as the bound of state type complexity. Then we can consider conditional inequalities between these classes, the infinity of the hierarchies and so on. A few examples of such relations were shown in sections IV and V. Another possible one can be stated as follows:

Corollary 6:

If the hierarchy $\{ A_k^3 : k \geq 0\}$ is infinite then:

$$\text{ASTATE}(\text{poly}(\log)) \subsetneq \text{PSPACE}$$

Proof:

The inclusion follows from property 9 of section IV. Assume that $\{ A_k^3 : k \geq 0 \}$ is infinite and PSPACE $\subseteq$ ASTATE(pol(log)). Consider the set B_ω complete in PSPACE. There has to exist some $k \geq 0$, such that $B_\omega \in A_k^3$. Take an arbitrary language $L \in$ PSPACE. Because of corollary 1, L can be reduced in the polynomial time $p(n)$ to B_ω preserving the number of alternations. Then we can recognize L on an alternating Turing machine with state type complexity bounded by:

$$\log(p(n))^k \leq (\deg(p)+c)^k \cdot (\log n)^k \leq (\log n)^{k+1}$$

and the inequalities hold almost everywhere. Now we can conclude that PSPACE $\subseteq A_{k+1}^3$ which contradicts with the infinity of the hierarchy $\{A_k^3 : k \geq 0\}$.

□

Another question arising here is if there exists any relation between the class ASTATE(f(n)) and subclasses of PSPACE defined by computations with bounded amount of memory to some fixed polynomial n^k.

PART II

VII. STA measure.

The ASTATE(f(n)) classes considered in the previous chapters are the special case of the classes implied by the complexity measure STA introduced by Bermen [2]. We say that a language L is in the class STA(s(n), t(n), a(n)) if there exists a single-tape machine $M \in ATM$ which accepts L and for every $x \in L$, $|x|=n$ there is an accepting subtree D' of the computation tree D of M for x, such that:

1. The length of any configuration in D' is bounded by s(n),
2. The depth of the subtree D' is bounded by t(n),
3. The state-type complexity of D' is bounded by a(n).

Hence the class STA(s(n), t(n), a(n)) denotes the class of languages which can be recognized by alternating TM with simultaneous bounds on space, time and alternations given by functions s(n), t(n) and a(n). By replacing some coordinate of STA by a class of function C we shall mean the sum of respective classes STA over C. We will use the following classes of functions:

$c\cdot log = \{c\cdot \log n : c \geqslant 0\}$,

$pol = \{n^k : k \geqslant 0\}$,

$exp = \{2^{n^k} : k \geqslant 0\}$,

$EE = \{2^{Exp} : Exp \in exp\}$.

The special cases of the function a(n) are:

det - only deterministic machines(without alternations),

0 - ordinary nondeterministic machines (without alternations),

k - the constant number of alternations.

We also use '*' to indicate no limit in the given coordinate. Some classes defined by the STA measure can be related to the well known time and space classes in the straightforward manner:

$$\text{PTIME} = \text{STA}(\text{pol}, \text{pol}, \text{det}), \tag{11}$$

$$\text{PSPACE} = \text{STA}(\text{pol},\text{exp},0) = \text{STA}(\text{pol},\text{pol},\text{pol}).$$

Remark 4: There is a slight difference between STA measure defined here and that introduced by Berman. We always assume that the initial configuration of an alternating machine is existential. □

The classes ASTATE(f(n)) considered in the part I can be denoted by:

$$\text{ASTATE}(f(n)) = \text{STA}(\text{pol},\text{pol},f(n)). \tag{12}$$

The next theorem is a generalization of Savitch's result that PSPACE = NPSPACE. If we denote NPSPACE = STA(pol,exp,0) then by simple induction it is possible to show that PSPACE = STA(pol,exp,k) for any constant k. We will prove even more, namely the last equality will hold if we replace the constant k by any polynomial.

Theorem 5:

$$PSPACE = STA(pol,exp,pol). \qquad (13)$$

Proof:

$\subseteq$: immediately from (11).

$\supseteq$: Let L be an arbitrary language from the class STA(pol,exp,pol). Then there exists a machine $M \in ATM$ recognizing the language L and operating in space n^k with the number of alternations bounded by n^r for some fixed integers k and r. We shall construct a deterministic algorithm A accepting the language L and using a polynomial amount of memory.

The construction of A:

Denote by $\vdash$ the next-move function of machine M and by $\vdash^*$ the transitive closure of $\vdash$. Given two configurations C_1, C_2 of M of length n^k we can deterministically check in space n^{2k} whether $C_1 \vdash^* C_2$ and all intermediate configurations between C_1 and C_2 have length not exceeding n^k. Moreover, the following predicates Comp1 and Comp2 are also computable in deterministic space n^{2k}:

Comp1(C_1,C_2) $\Leftrightarrow$ $C_1 \vdash^* C_2$, C_1 is an existential configuration, C_2 is a universal configuration, all configurations appearing between C_1 and C_2 are existential and not longer than n^k,

Comp2(C_1,C_2) $\Leftrightarrow$ $C_1 \vdash^{*} C_2$, C_1 is a universal configuration, C_2 is an existential configuration and all configurations appearing between C_1 and C_2 are universal and not longer than n^k.

The predicates Comp1, Comp2 compute whether it is possible to pass from a configuration C_1 to C_2 using machine M in the space bound n^k with only one alternation at the last application of the next-move function of M.

We shall use one more predicate:

ACC(C) $\Leftrightarrow$ M starting from configuration C, accepts C, operates in space n^k and does not make any alternation along the accepting computation for C.

Let us note, that ACC(C) is also computable in deterministic space n^{2k} for C of length n^k. There are two cases: C can be an existential or universal configuration. In the first case ACC(C) denotes that M accepts C nondeterministically and we can use Savitch's theorem NSPACE(s(n)) $\subseteq$ DSPACE$(s(n)^2)$. The second case is symmetric.

Now we can construct a recursive procedure TEST(C) checking if the alternating machine M starting from an arbitrary configuration C of length n^k accepts C in the space bound n^k.

$$\begin{aligned}\mathrm{TEST}(C) \Leftrightarrow\ & \mathrm{ACC}(C) \ \vee \\ & (\exists C_1)[\mathrm{Comp1}\ (C, C_1) \wedge \\ & \quad [\mathrm{ACC}(C_1) \vee \\ & \quad\ (\forall C_2)\,(\mathrm{Comp2}(C_1, C_2) \Rightarrow \mathrm{TEST}(C_2))]],\end{aligned}$$

where all configurations have length not exceeding n^k (the quantifiers are bounded to the configurations of length n^k). Finally, the algorithm A has the following form:

A: 1. C:=initial configuration of M for input x of length n, the length of C is n^k.

2. If TEST(C) then ACCEPT else REJECT.

Since the machine M operates in space n^k, algorithm A recognizes the language L, i.e. $x \in L$ iff A accepts x.

The recursive calls of the procedure TEST we implement in a natural way on a stack. The quantifiers are realized by simple looping over all the configurations of M of length n^k. The depth of the recursion in the procedure TEST is bounded by $n^r/2$ which is equal to half the number of alternations made by M (we terminate the computation of TEST without accepting if the depth of the recursion exceeds $n^r/2$).

Let T(i) denote a memory requirement of the procedure TEST for configurations C on which machine M alternates at most 2i times. Then we have:

$$\begin{cases} T(0) \leqslant n^{2k} \\ T(i+1) \leqslant n^{2k} + n^k + T(i) \end{cases}$$

Then the total memory requirement of the algorithm A is bounded by $T(n^r/2) \leq 2n^{2k+r}$. Hence the language L belongs to the class PSPACE, which terminates the proof of the theorem. □

As a corollary we can obtain some trade-offs between time and number of alternations using the STA-notation.

Corollary 7:

1. STA(pol,pol,pol) = STA(pol,exp,0) = STA(pol,exp,pol)
2. STA(exp,exp,exp) = STA(exp,EE,0) = STA(exp,EE,exp)

Proof:

1. from (11) and (13)
2. direct "upward translation" of the point 1(for similar techniques see Book [3],[4]).

□

Remark 5:

Another interesting trade-off property of the STA measure was obtained by Berman [2]:

$$\bigcup_{c>1} STA(2^{cn}, 2^{cn^2}, cn) = \bigcup_{c>1} STA(2^{cn}, 2^{cn}, cn).$$

VIII. Compendium of complexity classes.

The STA measure is monotonic with respect to its three arguments. However it seems to be a very difficult question at which points STA is strictly monotonic. We shall regard the classes STA(s(n), t(n), a(n)) only for reasonable cases, i.e. for function s,t,a satisfying $s(n) \leqslant t(n) \wedge a(n) \leqslant t(n)$. Fixing space and time we will be changing the function a(n), which bounds the number of alternations. The resulting complexity classes are gathered in Table 1.

Time, space and the number of alternations are bounded by the following classes of functions:

TIME : pol, exp, EE,

SPACE: log, pol, exp,

ALT : det- deterministic computations

0 - nondeterministic computations

k - constant

logn, n^k, pol, exp.

The first three columns in the table denote the parameters of STA, the fourth is the resulting complexity class, and in the last one we include some remarks. The classes from the fourth column form a nondecreasing chain. From the hierarchy theorem

$$\mathrm{DSPACE}(f) \subsetneq \mathrm{DSPACE}(2^f)$$

we know that in this chain there are at least two points where the succesive classes are different. However, so far we are unable to specify exactly any such point.

TABLE I.

SPACE	TIME	ALT	CLASS	REMARKS
log	pol	det	DLOGSPACE	
log	pol	0	NLOGSPACE	
log	pol	k	STA(log,pol,k)	
log	pol	n^k	STA(log,pol,n^k)	
log	pol	pol	PTIME	[6], $= \bigcup_k$ STA(log,pol,n^k)
pol	pol	det	PTIME	P $\stackrel{?}{=}$ NP conjecture
pol	pol	0	NPTIME	
pol	pol	k-1	Σ_k^p	[17], [18], =ASTATE(k-1)
pol	pol	logn	ASTATE(logn)	sec.V
pol	pol	$f(n) \geq \log n$	ASTATE(f(n))	sec.V
pol	pol	pol	PSPACE	[6], $= \bigcup_k$ STA(pol,pol,n^k)
pol	exp	det	PSPACE	
pol	exp	pol	PSPACE	th.5
pol	exp	2^{n^k}	STA(pol,exp,2^{n^k})	
pol	exp	exp	EXPTIME	[6], $= \bigcup_k$ STA(pol,exp,2^{n^k})
exp	exp	det	EXPTIME	exponential version
exp	exp	0	NEXPTIME	of the P$\stackrel{?}{=}$NP conjecture
exp	exp	pol	STA(exp,exp,pol)	
exp	exp	exp	EXPSPACE	[6], $= \bigcup_k$ STA(exp,exp,2^{n^k})
exp	EE	det	EXPSPACE	
exp	EE	exp	EXPSPACE	Cor.7.2
exp	EE	EE	EE-TIME	

Table I is not complete, in the sense that there are known complexity classes, which cannot be embedded directly into this scheme (or possibly we do not know how to do it). For instance, the class EXPLINTIME= $\bigcup_{c\geq 1}$ DTIME(2^{cn}) lies between PTIME and EXPTIME and we can't precise it more exactly.

Some interesting results dealing with STA complexity classes were obtained in classifying the complexity of a few decidable logical theories:

1. The decision problem for the theory of real addition is complete in the class STA(*, 2^{cn}, n) — Bruss, Meyer [5], Berman [2].
2. The elementary theory of Boolean algebras is complete in the class STA(*,2^{cn}, n) — Kozen [11].
3. The decision problem for Presburger arithmetic is complete in the class STA(*,$2^{2^{cn}}$, n) — Berman [2].

Let us mention also the work of Kintala and Fischer [10] on bounded nondeterminism, where the technique for restricting the computations and the resulting complexity classes seem to have a similar structure as in the present paper.

Acknowledgment.

I am grateful to Felipe Bracho for all his encouragement, helpful discussions and suggestions.

References:

1. T. Baker, J. Gill and R. Solovay, Relativizations of the P = ? NP Question, *SIAM J. Comp.*, Vol. 4, No.4, 1975, pp. 431-442.

2. L. Berman, The complexity of Logical Theories, *Theoretical Computer Science*, Vol. 11, 1980, pp. 71-77.

3. R.V. Book, Comparing Complexity Classes, *JCSS*, Vol. 9,No. 2, 1974, pp. 213-229.

4. R.V. Book, Translational Lemmas, Polynomial Time and $(\text{Log } n)^j$-Space, *Theoretical Computer Science*, Vol. 1,1976, pp. 215-226.

5. A. Bruss, A. Meyer, On Time-Space Classes and their Relation to the Theory of Real Addition, *Theoretical Computer Science*, Vol. 11, 1980, pp. 59-69.

6. A.K. Chandra and L. J. Stockmeyer, Alternation, Proc. 17th Ann. IEEE Symp. on Foundations of Computer Science, New York, 1976, pp. 98-108.

7. S.A. Cook, The Complexity of Theorem-proving procedures, Third ACM Symposium on Theory of Computing, New York, 1971, pp. 151-158.

8. M. Garay, D. Johnson, *Computers and Intractability, A Guide to the Theory of NP-completeness*, W.H. Freeman and Company, San Francisco, 1979.

9. T. Kasai, A. Adachi and S. Iwata, Classes of Pebble Games and Complete Problems, *SIAM J. Comp.*, Vol.8, No.4, 1979, pp. 574-586.

10. C. Kintala, P. Fischer, Refining Nondeterminism in Relativized Polynomial-Time Bounded Computations, *SIAM J. Comp.*, Vol.9, No.1, 1980, pp. 46-53.

11. D. Kozen, Complexity of Boolean Algebras, *Theoretical Computer Science*, Vol. 10, 1980, pp. 221-247.

12. D. Kozen, Lower Bounds for Natural Proof Systems, Proc. 18th IEEE Symp. on Foundations of Computer Science, 1977, pp. 254-266.

13. R. Ladner, The Circuit Value Problem is Logspace Complete for P. *SIGACT NEWS*, Vol. 7, No. 1, 1975.

14. W. Paul and R. Reischuk, On Alternation I and II, GI-Conference on Theoretical Computer Science, 1979.

15. T. J. Schaefer, On the Complexity of Some Two-Person Perfect-Information Games, *JCSS*, Vol. 16, No.2, 1978, pp. 185-225.

16. L.J. Stockmeyer and A.K. Chandra, Provably Difficult Combinatorial Games, *SIAM J. Comp.*, Vol. 8, No.2, 1979, pp. 151-174.

17. L.J. Stockmeyer, The Polynomial-Time Hierarchy, *Theoretical Computer Science*, Vol. 3, 1977, pp. 1-22.

18. C. Wrathall, Complete Sets and the Polynomial-Time Hierarchy, *Theoretical Computer Science*, Vol. 3, 1977, pp. 23-33.