KYBERNETIKA — VOLUME 15 (1979), NUMBER 2

A Turing Machine Space Hierarchy

STANISLAV ŽÁK

This paper introduces a new, finer space complexity measure of computations on Turing machines. The complexity of a computation on a Turing machine now takes into account also the capacity of the finite control. It is proved that a slight enlarging (by an additive constant) of the complexity bound increases the computing power. The proofs are based on a new principle of diagonalization. The results are similar for deterministic and nondeterministic off-line Turing machines, auxiliary pushdown automata, auxiliary counter automata and also for their versions with an oracle.

INTRODUCTION

A classical problem of the theory of computational complexity is the question to find the slightest enlarging of the complexity bound which increases the computing power. This paper continues the relatively long tradition of investigation of this question for the case of space complexity of computations on Turing machines (TM).

The earlier results in this area can be found, for example, in [1]. They are of the form: For a function t on natural numbers, let $SPACE(t)$ be the class of languages accepted by Turing machines such that, working on an input word of the length n, the machine never uses more than $t(n)$ tape squares. Then if t_2, t_1 are functions, t_2 is constructable in a certain sense and if $\liminf (t_1(n)/t_2(n)) = 0$, then $SPACE(t_2) - SPACE(t_1) \neq \emptyset$.

A progress has been made in [3]. Here the complexity of a computation of a TM is given not only by the number of tape squares used during the computation but also by the tape alphabet size of the TM and by the number of its heads. This allows the author to prove new results of the following form: For t a function on natural numbers, for natural numbers m, l, $m \geq 2$, $l \geq 1$, let $SPACE(t, m, l)$ be the class of languages accepted by nondeterministic Turing machines with m tape symbols

and l heads such that for each accepted word of the length n there is an accepting computation which doesn't require more than $t(n)$ tape squares. If t_2 is a (fully) constructable function, then

$$SPACE(t_2, m, l+1) - \bigcup\{SPACE(t_1, m, l) \mid t_2(n) - t_1(n+1) \to \infty\} \neq \emptyset .$$

In this paper, a new type of space complexity of computations on TM's is defined and investigated. It is based not only on the number of tape squares visited during the computation, on the tape alphabet size of a TM and on the number of its heads, but also on the capacity of its finite control. (A similar idea can be found in [6] and [7]). Thus from a possible list of basic structural features of a TM also the last item is taken into account in our approach. The complexity of a computation of a TM with m tape symbols and l heads is defined as the sum of the number of tape squares visited during the computation and of the length of the program of the machine.

This approach gives the possibility to prove separation results such as: There is a constant a such that

$$SPACE(t_2 + a, m, l) - \bigcup\{SPACE(t_1, m, l) \mid \liminf (t_2(n) - t_1(n+1)) \geqq 0\} \neq \emptyset ,$$

and to obtain a new complexity hierarchy which refines earlier hierarchies. The results are formulated for off-line TM's, TM's with auxiliary pushdown stores (= AuxPDA) as in [5] and also for TM's and AuxPDA's with oracles. Their proofs are based on a certain principle of diagonalization.

For the first time this principle was used by the author in [8] for proving that linear bounded automata (lba) with $k+1$ symbols, $k \geqq 2$, accept some languages that cannot be accepted by lba's with k symbols. We defined the space complexity taking into account also the capacity of finite control for the first time in [9]. In that paper, the first theorem concerning this complexity was proved by application of the principle of diagonalization mentioned above.

This paper extends ideas from [9]. It consists of three chapters. The first chapter is concerned with diagonalization, the second contains all complexity results and in the third a comparison with some earlier results is made.

CHAPTER 1

The aim of this chapter is to introduce a new principle of diagonalization. The first part of the chapter, consisting in a great deal of the basic definition of a mapping called the result of testing process (rtp) and of a theorem, exhibits the logical structure of that principle without any care of existence and complexity aspects. The second part of the chapter is formed by a lemma which ensures the existence of the rtp-mappings, introduces first complexity aspects and in its proof provides a construction of such mappings. For gaining an intuitive insight into the principle of diagonalization

it is advisable to follow simultaneously the proof of the theorem and the construction of rtp-mappings.

Let us first recall some usual definitions and conventions. An alphabet is a non-empty finite set of symbols, all alphabets are subsets of a fixed infinite set containing, among others, the symbols b, 0, 1, 2, $\circ$, $\#$, $*$, S. A string or a word over an alphabet is a finite sequence of its symbols, Λ denotes the empty word, $|u|$ is the length of the word u. A language over an alphabet is a set of strings over this alphabet. If X is an alphabet then X^* (X^+, X^n) is the language of all words (of positive length, of the length n, respectively) over X. Two words may be concatenated which yields a similar operation for languages. $\mathcal{N}$ denotes the set of natural numbers. If a is a symbol and $i \in \mathcal{N}$ then a^i is a string of a's of the length i. By a function or by a bound we always mean a mapping of $\mathcal{N}$ into itself. The identity function will be denoted id, $\mathrm{id}(n) = n$. For two functions f, g we shall write $f \leqq g$ iff $(\forall n \in \mathcal{N})\,(f(n) \leqq g(n))$. For two sets A, B, by the expression $A \subset B$ we shall mean that A is a proper subset of B. From time to time in the following text we shall use the if ... then ... else construction, well-known from the programming languages.

We shall call two languages L_1, L_2 equivalent ($L_1 \sim L_2$) iff they differ only in a finite number of strings. If $\mathscr{L}$ is a class of languages then $\mathscr{E}\mathscr{L}$ will be the class of all languages for which there are equivalent languages in $\mathscr{L}$.

By a program system we mean a pair (P, F) where P is a language and F is a mapping of P into a set of languages over an alphabet. In this context, P is called the set of programs and its elements are called programs. In what follows, if we use the phrase "Let P be a set of programs", we implicitly understand that P is the first item of a program system. Its second item will have the general denotation L and L_p will mean the language which corresponds to the program $p \in P$. The set of all such L_p for all $p \in P$ will be denoted by $\mathscr{L}(P)$.

For a program p and a word u, we say that p accepts u ($p!\, u$) iff $u \in L_p$.

Definition. Let p be a program and Q a set of programs. We say that p diagonalizes Q iff there is a finite set F such that $(\forall q \in Q - F)\,(p!\, q \leftrightarrow \neg q!\, q)$.

Lemma 1. Let p be a program and Q a set of programs. If there are infinitely many programs from Q with the same language as the program p then p does not diagonalize Q.

Proof. There are infinitely many $q \in Q$ such that $p!\, q \leftrightarrow q!\, q$.

Definition. Let Q, R be sets of programs, ε a function and RTP a mapping of $\mathcal{N}$ without some initial segment $\{0, 1, \ldots, k\}$ into the set Q. If for all $q \in Q$ the sets

$$R_q = \{r \in R \mid RTP(\varepsilon(|r|)) = q \wedge \neg(q!\, r \leftrightarrow \neg r!\, r)\}$$

are infinite then RTP is called the result of a testing process with the function ε on the sets Q, R (in short, rtp with ε on Q, R).

The existence of such a mapping will be proved later in Lemma 2.

Theorem 1. Let Q, R be sets of programs, RTP an rtp with ε on Q, R, N a program and z a mapping from R into the set of natural numbers. If for all $q \in Q$ there are infinitely many $r \in R_q$ such that

(1) $N!\, r0^{z(r)} \leftrightarrow \neg r!\, r$,

(2) $(\forall j, 0 \leqq j < z(r))\,(N!\, r0^j \leftrightarrow RTP(\varepsilon(|r|))!\, r0^{j+1})$,

then $L_N \notin \mathscr{E}\mathscr{L}(Q)$.

Proof. Suppose $L_N \sim L_q$ for some $q \in Q$. Take r, $r \in R_q$, satisfying conditions (1), (2) of the theorem and larger than all the words which belong to one of the languages L_N, L_q only. Since $r \in R_q$, the conditions (a) $RTP(\varepsilon(|r|)) = q$ and (b) $\neg(q!\, r \leftrightarrow \neg r!\, r)$ hold.

Now, we are going to prove $q!\, r \leftrightarrow \neg r!\, r$. This will be a contradiction with (b).

First, we shall prove $q!\, r \leftrightarrow q!\, r0^{z(r)}$. If $z(r) = 0$, this is trivial. If $z(r) > 0$ then, for all i, $0 \leqq i < z(r)$, the following statements are equivalent:

(i) $q!\, r0^i$,

(ii) $N!\, r0^i$,

(iii) $RTP(\varepsilon(|r|))!\, r0^{i+1}$,

(iv) $q!\, r0^{i+1}$.

(i) $\leftrightarrow$ (*ii*) follows from the fact that r has been chosen sufficiently large.

(ii) $\leftrightarrow$ (iii) is ensured by the condition (2) of the theorem.

(iii) $\leftrightarrow$ (iv) follows from the condition (a).

We have $q!\, r \leftrightarrow q!\, r0^{z(r)}$. However, since r is sufficiently large, $q!\, r0^{z(r)} \leftrightarrow$ $\leftrightarrow N!\, r0^{z(r)}$ and $N!\, r0^{z(r)} \leftrightarrow \neg r!\, r$ according to the condition (1) of the theorem. We have $q!\, r \leftrightarrow q!\, r0^{z(r)} \leftrightarrow \neg r!\, r$. This is a contradiction with (b). Q.E.D.

The next lemma and its proof concern Turing machines and Turing machines with oracles. We will deal with various types of deterministic and nondeterministic TM's, considered as accepting devices.

We say that a TM T accepts a word u if there is a computation of T on u which stops in a final accepting state. If T is a deterministic single-tape TM and accepts a word u, then $T(u)$ denotes the word written on the tape after the computation of T on u has been finished.

A Turing machine with oracle $A(A \subseteq \mathcal{N})$ is a Turing machine which, among its tape, has a fixed one, on which a special symbol S may be written. The set of states of the machine includes three special states q, YES, NO. If it enters the state q, then

in the next step if the number of occurrences of S on its fixed tape belongs to A, it must enter the state *YES*, otherwise the state *NO*.

In this chapter we shall use only deterministic single-tape Turing machines with oracles. – A function ε will be called (A-) recursive if there is a deterministic Turing machine (with oracle A) such that for all $n \in \mathcal{N}$ $T(1^n) = 1^{\varepsilon(n)}$.

A language over an alphabet X is called recursively enumerable (A-recursively enumerable) if it is accepted by a Turing machine (Turing machine with oracle A) and it is called (A-) recursive if moreover its complement in X^* is also (A-) recursively enumerable.

If P is a set of programs then $!_P$ is the binary relation $\{(p, u) \mid p \in P, u \in L_p\}$. – The graph of a binary relation H on a set of strings is the set $\{u2v \mid (u, v) \in H\}$.

Lemma 2. (rtp-lemma). (a) Let Q, R be nonempty sets of programs and ε a function. If no program from Q diagonalizes R and if $\varepsilon \leqq \mathrm{id}$ and $\lim_{n\to\infty} \varepsilon = \infty$ then there is an rtp with ε on Q, R.

(b) Let A be an oracle. If, in addition, the sets Q, R, $Q \subseteq \{b, 1\}^*$, are (A-)recursively enumerable languages and the graphs of the relations $!_Q$, $!_R$ are (A-)recursive and also the function ε is (A-)recursive then there is a deterministic Turing machine T (with oracle A) with one tape and with one head such that

(1) During the computation on the input word 1^k, T uses only the input cell and two adjacent cells;

(2) T writes only the symbols 1, b (1, b, S);

(3) There is a constant c such that the mapping $RTP = \{(k, T(1^k)) \mid k \in \mathcal{N}, k \geqq c\}$ is an rtp with ε on Q, R.

In fact, we have two lemmas – the version without an oracle and the relativized version. The same is true for the proof.

Proof of (a). Since no $q \in Q$ diagonalizes R, R is infinite. Let $\{r_j\}$ be a sequence of all programs from R and $\{q_i\}$ a sequence of all programs from Q with infinitely many occurrences of each of them.

We construct a sequence $\{v_n\}$ of words: Let $\{m_n\}$ be a sequence of natural numbers. Let $v_0 = \Lambda$. For $n > 0$, the words v_n are of the form

$$v_n = v_{n-1}[\circ \circ q_{i_n} \circ q_{i_{n+1}} \circ 1^{i_n} \circ r_{j_n} \circ 1^{j_n} \circ x_n \circ b^{m_n} \circ \circ]$$

where $[\ ]$ is a homomorphic binary coding in the alphabet $\{b, 1\}$, $x_n \in \{0, 1\}$, $i_1 = 1$, $j_1 = 1$ and the numbers i_{n+1}, j_{n+1} and also x_n depend on the truth value of the following expression

$$\text{(A)} \qquad \neg(q_{i_n}!\, r_{j_n} \leftrightarrow \neg r_{j_n}!\, r_{j_n}) \wedge \varepsilon(|r_{j_n}|) > |[*]\, v_{k_n}[\#]|\,.$$

Here k_n is defined as follows. Let $K_n = \{k \mid k < n \wedge i_k \neq i_n\}$. If $K_n = \emptyset$, then $k_n = 0$, otherwise k_n is the maximal member of K_n.

We define: If (A) holds then $x_n = 1$ and $i_{n+1} = i_n + 1, j_{n+1} = 1$, else $x_n = 0$ and $i_{n+1} = i_n, j_{n+1} = j_n + 1$.

Then we define a mapping RTP. For $k \in \mathcal{N}$, $RTP(k) = q_{i_{n+1}}$ where $n = \max\{m \mid |[*]v_m[\#]| < k\}$. Later, we shall prove that RTP is an rtp with ε on Q, R.

Remark. The words v_n are rather complicated. But the elements 1^{i_n}, 1^{j_n}, q_{i_n+1}, x_n, b^{m_n} are contained in v_n and symbols $*$, $\#$ occur in the formulation of (A) and in the definition of RTP for purposes of part (b) of proof only. For purposes of part (a) it is sufficient to define $v_0 = \Lambda$, $v_n = v_{n-1}q_{i_n}r_{j_n}$ and to write, in (A) ... $\varepsilon(|r_{j_n}|) > |v_{k_n}|$ and in the definition of RTP, ... $n = \max\{m \mid |v_m| < k\}$.

The sequence $\{v_n\}$ may be called the testing process. Let us observe the suffixes $[\circ \circ q_{i_n} \circ \ldots \circ b^{m_n} \circ \circ]$ $(q_{i_n}r_{j_n})$ in the words v_n. We see that each member of the sequence $\{q_i\}$ is tested whether it diagonalizes R. The testing of each q_i starts with the program r_1. Let $\bar{r}_i$ be the program with which the testing of q_i finishes. According to (A), $\bar{r}_i$ is so large that $\varepsilon(|\bar{r}_i|)$ is greater than the length of the shortest word $[*]\, v_n[\#]$ (v_n) containing the whole testing of q_{i-1}. However, since ε is majorized by the identity, $\varepsilon(|\bar{r}_i|)$ is smaller than the length of any word v_n containing the whole testing of q_i, because $\bar{r}_i$ is a part of each such word. Let v_k be the word of maximal length which is smaller than $\varepsilon(|\bar{r}_i|) - |[* \#]|$ $(\varepsilon(|\bar{r}_i|))$. We know that v_k contains the whole testing of q_{i-1} but it does not contain the whole testing od q_i. So, $i_{k+1} = i$. We have $RTP(\varepsilon(|\bar{r}_i|)) = q_i$.

For proving that RTP is an rtp with ε on Q, R, it suffices to show that the sets R_q from the definition of rtp are infinite. We define, for all $q \in Q$, $R'_q = \{\bar{r}_i \mid q_i = q\}$. We have proved $R'_q \subseteq R_q$. Each R'_q is infinite, since each $q \in Q$ occurs infinitely many times in $\{q_i\}$ and it is not possible to find infinitely many q_i $(q_i = q)$ which are rejected during the testing on the same program $r = \bar{r}_i$.

Proof of (b). We say that a sequence $\{a_i\}$ of words over an alphabet is (A)-effective iff there is a deterministic Turing machine (with oracle A) rewriting the unary code of any natural number i to the word a_i.

Fact. There are a sequence $\{m_n\}$, words v_n and a deterministic single-tape Turing machine T' (with oracle A) such that T' has one head which writes the symbols $1, b$ $(1, b, S)$ only, its tape is infinite to the right only and limited from the left by the symbol $*$, and T' is such that

(i) during the computation on the empty input word, T' writes the words v_n on its tape (it rewrites v_1 to v_2, v_2 to v_3, ...);

(ii) for writing the word v_n, T' uses $|v_n|$ tape squares only;

(iii) if there is the symbol $\#$ in a cell of its tape then the head of T' moves between the symbols $*$, $\#$ only. First it writes the longest v_n that is possible to write

between $*$, $\#$, and then it writes the program $q_{i_{n+1}}$ as the result of the computation (precisely: $T'(b^l \#) = q_{i_{n+1}}$ where $n = \max \{m \mid |v_m| < l\}$.)

Proof sketch. Since the sets Q, R are (A-)recursively enumerable we can choose the sequences $\{q_i\}$, $\{r_j\}$ (A-)effective. Moreover, the function ε and the graphs of the relations $!_Q$, $!_R$ are (A-)recursive. Therefore it is possible to construct the words v_{n+1} from the words v_n (A-)recursively. The condition (iii) is ensured by the possibility to choose the numbers m_n sufficiently large. The condition (ii) is clear since the program $q_{i_{n+1}}$ is a part of the word v_n and its value (q_{i_n} or q_{i_n+1}) is given by x_n.

We choose the machine T so that T rewrites the input word 1^k to the word $[*]\, b^{k-|[**]|}[\#]$, then it computes between the words $[*]$, $[\#]$, simulating the work of T' between the symbols $*$, $\#$, and then it leaves only the resulting program ($q_{i_{n+1}}$) on its tape.

We see that T satisfies the requirements (1), (2) of the lemma and also that $T(1^k) = q_{i_{n+1}}$ where $n = \max \{m \mid |[*]\, v_m[\#]| < k\}$. Therefore the mapping RTP defined in the formulation of the lemma is an rtp with ε on Q, R since it is the same as the RTP introduced in the proof of (a). Q.E.D.

CHAPTER 2

In this chapter, we introduce various models of Turing machines and define a new type of space complexity measure. We shall formulate some lemmas concerning minimal constructable functions and prove two general separation results for languages over two- and one-letter alphabets by application of the Theorem 1 and of the rtp-lemma. We will conclude the chapter with some corollaries and remarks.

We shall use a special Turing machine model that has a read-only input tape and a single read-write worktape. The input strings are words over the alphabet $\{0, 1\}$. They are placed between two special markers $\%$, $\S$ on the input tape and are read by a single read-only input head which is allowed to move freely between the markers. The worktape is limited from the left by the symbol $*$ and is infinite to the right. We allow any fixed finite number of freely moving, but initially left adjusted, read-write heads on the worktape. The worktape heads can detect each other and they are never required to write conflicting symbols on a single tape square in the same step or to shift to the left from the left end (marked by $*$) of the worktape. The worktape alphabet contains a special symbol $\#$ which will be called endmarker. At least one and at most two cells contain the endmarker in each configuration during each computation. Such two cells must be adjacent. All cells to the right of the endmarker (end markers) are blank and the initial part of the worktape including the first blank past the endmarker contains all heads. The endmarker (endmarkers) are shifted by rewriting during the computation. The initial configuration contains the endmarker in the first worktape cell.

Such a Turing machine works as an acceptor and may be deterministic or nondeterministic. It is called Turing machine with endmarker and if it has l worktape heads and m worktape symbols it is called (m, l)-Turing machine with endmarker (only the symbols different from $*$, $\#$ are counted including the blank symbol b).

Remark. The definition is the same as in [3] except for the endmarker. A similar definition without endmarker is possible: (m, l)-Turing machine such that in each configuration in any computation all nonblanks are to the left from all blanks on the worktape, all worktape heads are scanning the initial part of the worktape including the leftmost blank. The leftmost blank here is our endmarker.

We shall also use (m, l)-Turing machines with oracles – see the definition in Chapter 1. The third computational device we will use is the (m, l)-auxiliary pushdown automaton (AuxPDA) with endmarker. It is the (m, l)-Turing machine with endmarker and with an additional pushdown store. The organization of the pushdown store is the same as in [1]. An AuxPDA whose pushdown store is a counter only is called an auxiliary counter automaton (AuxCA) – see [1]. Since the AuxPDA is a special type of TM, we already know what is an AuxPDA with an oracle.

Let us fix the numbers $l\,(l \geqq 1)$, $m\,(m \geqq 2$, or, for the case of TM's with oracles, $m \geqq 3)$, the pushdown alphabet size and an oracle A. We have defined twelve types of accepting devices: (m, l)-Turing machine with endmarker, (m, l)-AuxPDA and (m, l)-AuxCA with endmarker, each of them either deterministic or nondeterministic, and either with oracle or without oracle. However, the definitions, theorems and their proofs in what follows are very similar in all the twelve cases; that is why, as a rule, we will use only the word "machine" in the following text. If we replace all occurrences of the word "machine" in all definitions, theorems and proofs for example by the phrase "deterministic (m, l)-AuxPDA with endmarker and with oracle A" then we will get definitions, theorems and proofs correct for this case.

Definition. Let a, c be words and M a machine. Then we put $tape_{M,c}(a) = \infty$ iff M cannot reach any accepting state from the configuration where the word a is written on the input tape, $c\,\#$ forms the maximal nonblank initial segment of the worktape, all heads are left adjusted, M is in the initial state (and its auxiliary pushdown store contains only the start symbol). Otherwise $tape_{M,c}(a)$ = the minimal number of distinct worktape squares visited during an accepting computation of M.

Lemma 3. (Universal machine) There is a recursive set $\mathscr{S}$, $\mathscr{S} \subseteq \{b, 1\}^*$, in one-one correspondence with the set of all machines, and a machine U such that for each $s \in \mathscr{S}$ and for each input word u the equality $tape_{M_s,\Lambda}(u) + |s| = tape_{U,s}(u)$ holds (where M_s stands for the machine corresponding to s).

Observe that in fact we have twelve lemmas.

Proof. There is a recursive set $\mathscr{S}'$, $\mathscr{S}' \subseteq \{0, 1\}^+$, in one-one correspondence with the set of all machines, and an $(m + 1, l)$-machine U' such that the equality

$tape_{M_s,A}(u) + |s| = tape_{U',s}(u)$ holds for all $s \in \mathscr{S}'$. ($\mathscr{S}'$ is the usual set of codes of machines.)

Consider the work of the machine U'. It simulates the computation of M_s on the input word u. U' has the code s written on its worktape and, to the right of this code, U' writes the words that M_s writes on its worktape during the computation on u. The first worktape head of U' has two tasks: to simulate the first head of M_s, and to work within the code s. U' uses its additional $m + 1$-st symbol in two cases: It marks the cell, where the first head of U' must come back to simulate the first head of M_s, and it is used within the code s. We will eliminate the first case.

There is an $(m + 1, l)$-machine U'' working as the machine U' with the only exception that it shifts the given code s in such a way that the cell where the first head of U'' must come back to simulate the first head of M_s is the first cell to the right (left) of the code s. This is possible as we can choose $\mathscr{S}' \subseteq 1^{100}\{00, 01\}^+ 1^{100}$, for example. Then U'' uses its $m + 1$-st worktape symbol within the code s only.

Let $[\ ]$ be a coding of the alphabet $\{0, 1, \circ\}$ in the alphabet $\{1, b\}$, e.g. $[0] = 1b1bbb1$, $[1] = 1bb1bb1$, $[\circ] = 1bbb1b1$. Let us put $\mathscr{S} = \{[s] \mid s \in \mathscr{S}'\}$. There is an (m, l)-machine U which works like the $(m + 1, l)$-machine U'' with the only exception that it treats a block consisting of seven symbols as one within the given code s. U is our machine from the lemma.

Let $\mathscr{S}$ be as in Lemma 3. Let $s \in \mathscr{S}$ be a code and u a word. We define $tape_s(u) = tape_{M_s,A}(u)$ and $space_s(u) = tape_s(u) + |s|$. Our definition of *space* differs from the definition in [3], *Space* of Seiferas is our *tape*. – Further, $L(M_s)$ will denote the language accepted by the machine M_s.

If t is a bound and $s \in \mathscr{S}$ then by t-tape-cut-off of the language $L(s) = L(M_s)$ we mean the set $L'(s) = \{u \mid tape_s(u) \leqq t(|u|)\}$. We say that machine M_s accepts its language within tape bound t if $L(s) = L'(s)$. – If t is a bound we put

$$TAPE(t, m, l) = \{L \mid (\exists s \in \mathscr{S})(L = L'(s) = L(s))\}$$

and

$$CTAPE(t, m, l) = \{L \mid (\exists s \in \mathscr{S})(L = L'(s))\}.$$

If S is a bound then by S-space-cut-off of the language $L(s) = L(M_s)$ we mean the set $L_S(s) = \{u \mid space_s(u) \leqq S(|u|)\}$. We say that the machine M_s accepts its language within space bound S if $L(s) = L_S(s)$. – If S is a bound we put

$$SPACE(S, m, l) = \{L \mid (\exists s \in \mathscr{S})(L = L_S(s) = L(s))\}$$

and

$$CSPACE(S, m, l) = \{L \mid (\exists s \in \mathscr{S})(L = L_S(s))\}.$$

Let S be a bound and G_S the graph of the relation $\{(s, u) \mid s \in \mathscr{S}, u \in L_S(s)\}$. We know that for the case of machines without oracle, if S is recursive then G_S is also recursive, and, for the case of machines with oracle A, if S is A-recursive then G_S is also A-recursive.

We say that a bound t is (m, l)-fully constructable, in short (m, l)-constructable, if there is a deterministic machine such that it accepts the language 1^+ within tape bound t and, for all $n \in \mathcal{N}$, it terminates its work on the input word 1^n by printing the symbol # into the $t(n)$-th worktape cell and by erasing symbols in all other worktape cells.

Theorem 2. If t is an (m, l)-constructable bound and $\lim_{n\to\infty} t = \infty$ then there is a language L such that

(1) $L \subseteq 0^+1^+0^*$,

(2) $L \in TAPE(t, m, l)$,

(3) $L \notin \mathscr{C}CSPACE(S, m, l)$, where $S(0) = 0$, $S(n+1) = t(n)$.

Proof. First, we shall choose a set Q of programs such that

$$\mathscr{L}(Q) = CSPACE(S, m, l)$$

and a set R of programs, both satisfying the conditions of the rtp-lemma. Secondly, we shall construct a machine N such that N accepts its language within tape bound t and this language has the properties (1), (2) from Theorem 1. By application of this theorem we will get $L(N) \notin \mathscr{C}\mathscr{L}(Q)$.

Let us write $Q = \mathscr{S}$ and for $q \in Q$, $L_q = L_S(q)$. Q is a recursive set and the graph of the relation $!_Q$ is also (A-)resursive.

Let us put $\varepsilon(n) = \min\{n, t(n)\}$ for all $n \in \mathcal{N}$. ε is (A-) recursive, $\lim_{n\to\infty} \varepsilon = \infty$ and $\varepsilon \leqq \mathrm{id}$.

Let $\{s_i\}$ be an effective sequence of programs from $\mathscr{S}$ such that each $s \in \mathscr{S}$ occurs infinitely many times in it.

Let h be a homomorphism with $h(0) = b$, $h(1) = 1$. There is a deterministic Turing machine M (with oracle A) with one tape infinite only to the right, with one head writing the symbols $1, b\,(1, b, S)$ only, and such that for all $s \in \mathscr{S}$, $u \in \{0, 1\}^*$, during the computation on the input words $s\,h(u)$, M decides whether the word u belongs to $L_S(s)$. Existence of such a machine M is ensured by the construction of the sets $\mathscr{S}$, $\mathscr{S}'$ in the proof of Lemma 3. — We define $tape_M(sv) =$ the number of tape squares either originally occupied by symbols of the word sv or visited during the computation of M on this word.

Let $\mathrm{bin}(j), j \in \mathcal{N}$, be a binary representation of the number j in $\{b, 1\}$. If $u = \mathrm{bin}(j)$ then let us put $\mathrm{val}(u) = j$. For $s \in \mathscr{S}$, $j \in \mathcal{N}$, we define $z(0^{\mathrm{val}(s)}1^j) = \min\{z \mid t(\mathrm{val}(s_i) + j + z) \geqq tape_M(sb^{\mathrm{val}(s)}1^j)\}$. For all $i \in \mathcal{N}$, we also define

$$j_i = \min\{j > 0 \mid (\forall k)\,(0 \leqq k < z(0^{\mathrm{val}(s_i)}1^j) \to t(\mathrm{val}(s_i) + j) \leqq t(\mathrm{val}(s_i) + j + k))\}.$$

Existence of such numbers j is ensured by the assumption that $\lim_{n\to\infty} t = \infty$.

Finally, let us define $R = \{r_i \mid i \in \mathcal{N}\}$, where $r_i = 0^{\mathrm{val}(s_i)}1^{j_i}$ and $L_{r_i} = L_S(s_i)$.

Let us observe that (a) a portion of tape of the length $t(|r_i| + z(r_i))$ is sufficient for deciding whether $r_i \in L_{r_i}$ $(r_i \,!\, r_i)$, and

(b) $$(\forall i)(\forall k)(0 \leqq k < z(r_i) \to \varepsilon(|r_i|) \leqq t(|r_i 0^k|)) .$$

No program from Q diagonalizes R because for each program in Q there are infinitely many programs in R with the same language – see the construction of the sequence $\{s_i\}$ of the set R, and Lemma 1. R is an $(A\text{-})$ recursive set and the graph of the relation $!_R$ is also an $(A\text{-})$ recursive language. – The sets Q, R and the function ε have the properties required in the rtp-lemma. This lemma gives us the possibility to define a constructive rtp with ε on Q, R. Let RTP be such a mapping.

Now, we are ready to construct the machine N. N starts its computation with checking whether the input word is of the form $0^i1^j0^k$, $i \geqq 1$, $j \geqq 1$, $k \geqq 0$. Then it writes its endmarker into the $t(n)$-th worktape square, where $n = i + j + k$. This is the last endmarker shift during the whole computation. We have $L(N) \subseteq 0^+1^+0^*$ and $L(N) \in TAPE(t, m, l)$.

Then N tries to write the word bin (i) b^i1^j on its worktape and to process this word as M does. For each i, bin $(i) \in \mathscr{S}$,

(1) if $tape_M(\text{bin}\,(i)\, b^i1^j) \leqq t(n)$, then

(2) N accepts iff $0^i1^j \notin L_S(\text{bin}\,(i))$; else N works as follows:

(3) N tries to put the symbol 1 into its $\varepsilon(i + j)$-th worktape square as follows. N works like the machine constructing t, simulating the endmarker of this machine by the symbol 1 in the rightmost nonblank cell which does not contain the endmarker. If $\varepsilon(i + j) \leqq t(n)$ then N works on the initial part of its worktape of the length $\varepsilon(i + j)$ in a similar manner as the Turing machine T from the rtp-lemma works on the input word $1^{\varepsilon(i+j)}$. Let $RTP(\varepsilon(i + j)) = s \in \mathscr{S}(= Q)$ be the resulting program. Then N works like the machine U (Lemma 3) does on the input word $0^i1^j0^{k+1}$ according to the code s. N accepts iff there is an accepting computation of U that does not require more than $t(n)$ squares, i.e.

(4) N accepts iff $tape_{U,s}(0^i1^j0^{k+1}) \leqq t(n) = t(i + j + k)$. N simulates the endmarker of U by the symbol 1 in the rightmost nonblank cell to the left from its own endmarker.

Now, we must prove that $L(N) \notin \mathscr{CL}(Q) = \mathscr{C}CSPACE(S, m, l)$. We shall apply Theorem 1. We have the sets Q, R and the mappings RTP, ε, z. We prove that the language $L(N)$ satisfies conditions (1), (2) of this theorem.

We shall demonstrate the equivalence $r0^{z(r)} \in L(N) \leftrightarrow \neg r!\, r$ for all but a finite number of programs r from R. – If $r \in R$, then $r = r_i = 0^{\text{val}(s_i)}1^{j_i}$ for some $i \in \mathcal{N}$. N treats the input word as follows: N writes its endmarker in the $t(\text{val}\,(s_i) + j_i + z(0^{\text{val}(s_i)}1^{j_i}))$-th worktape square and accepts iff $0^{\text{val}(s_i)}1^{j_i} \notin L_S(s_i)$-see the definition of the function z and (1), (2) and also observation (a) above.

Now, we shall demonstrate that for all sufficiently large $r \in R\,(\forall k, 0 \leqq k < z(r))$. $.(r0^k \in L(N) \leftrightarrow RTP(\varepsilon(|r|))!\, r0^{k+1})$ (condition 2). Let r be a program from R. Then

$r = r_i = 0^{\mathrm{val}(s_i)}1^{j_i}$ for some $i \in \mathcal{N}$. N processes the input word $0^{\mathrm{val}(s_i)}1^{j_i}0^k$, where $k < z(0^{\mathrm{val}(s_i)}1^{j_i})$, as follows: N writes its endmarker in the $t(\mathrm{val}(s_i) + j_i + k)$-th worktape square. Then it works according to (3) since $k < z(0^{\mathrm{val}(s_i)}1^{j_i})$ – see the definition of the function z and (1). According to the definitions of ε and of the numbers j_i, we know that $\varepsilon(\mathrm{val}(s_i) + j_i) \leqq t(\mathrm{val}(s_i) + j_i) \leqq t(\mathrm{val}(s_i) + j_i + k)$ – see also observation (b). Therefore N works as the machine U does on the input word $0^{\mathrm{val}(s_i)} . 1^{j_i}0^{k+1}$ according to the program $RTP(\varepsilon(|r|)) = s \in \mathcal{S}(= Q)$. The following statements are equivalent:

(i) $r0^k \in L(N)$,

(ii) $tape_{U,s}(r0^{k+1}) \leqq t(n)$, where $s = RTP(\varepsilon(|r|))$ – see (4),

(iii) $tape_{M_s,A}(r0^{k+1}) + |s| \leqq t(n) = S(n+1)$ – see Lemma 3,

(iv) $space_s(r0^{k+1}) \leqq S(n+1)$ – see the definition of $space_s$,

(v) $r0^{k+1} \in L_S(s)$ – since $|r0^{k+1}| = n + 1$,

(vi) $s!\, r0^{k+1}$ – since $L_S(s) = L_s$,

(vii) $RTP(\varepsilon(|r|))!\, r0^{k+1}$.

The language $L(N)$ satisfies the conditions of Theorem 1 and therefore $L(N) \notin \mathcal{CSL}(Q) = \mathcal{CSPACE}(S, m, l)$. Q.E.D.

Lemma 4. Let t be a bound and $\lim_{n\to\infty} t = \infty$. If t is constructable by a deterministic (m, l)-Turing machine with endmarker and with oracle A or without oracle then there is a constant K_t such that

$$\log_m n - l \,.\, \log_m \log_m n - K_t \leqq t(n) \quad \text{for all} \quad n \in \mathcal{N}\,, \quad n > 1\,.$$

In [3] the same fact is proved for the case of bounds t fully constructable by (m, l)-Turing machines without endmarkers and without oracles. Our lemma can be proved in the same manner.

Lemma 5. There is a nondecreasing (m, l)-constructable surjection $\varepsilon' : \mathcal{N} \to \mathcal{N}$ onto $\mathcal{N}$ such that for a constant K and for all sufficiently large $n \in \mathcal{N}$

$$\varepsilon'(n) < \log_m n - l \,.\, \log_m \log_m n + K\,.$$

Proof. Let M be a deterministic machine which accepts the language 1^* such that its input head moves one cell to the right in each step of each its computation; on its worktape, M writes all words from its worktape alphabet, provided with suffixes $\#$, $\#\#$. First M writes the words of the length 2, then the words of the length 3, 4 and so on. M proceeds so that the new word is always obtained by rewriting the previous one. Before writing a new word, its l worktape heads scan all possible l-tuples of accessible squares. This process has only n steps, where n is the length of

the input word. After n steps, M writes its endmarker in the rightmost worktape square so far scanned and it erases symbols in the other squares.

Let ε' be the function constructed by M. ε' is a nondecreasing surjection.

Now we must prove our inequality. We know that

$$(\varepsilon'(n) - 4)^l \, m^{\varepsilon'(n)-4} < n \,, \tag{1}$$

since during the computation on the input word 1^n, M writes a word of the length $\varepsilon'(n) - 1$, therefore it has already written all the words of the length $\varepsilon'(n) - 2$ and for writing these words it has needed more than $(\varepsilon'(n) - 4)^l \, m^{\varepsilon'(n)-4}$ steps. Obviously, there is a constant K such that for all $n \in \mathcal{N}$, $n > 1$,

$$m^{K-4}(1 - (l \,.\, \log_m \log_m n)/\log_m n + (K - 4)/\log_m n)^l \geqq 1 \,.$$

We have

$$(\log_m n - l \,.\, \log_m \log_m n + K - 4)^l \, m^{K-4} \,.\, n/(\log_m n)^l \geqq n$$

and

$$(\log_m n - l \,.\, \log_m \log_m n + K - 4)^l \, m^{\log_m n - l \,.\, \log_m \log_m n + K - 4} \geqq n \,.$$

Therefore $\log_m n - l \,.\, \log_m \log_m n + K > \varepsilon'(n)$ – see (1). Q.E.D.

Lemma 6. There is a nondecreasing surjection $\varepsilon' : \mathcal{N} \to \mathcal{N}$, $\varepsilon' \leqq \mathrm{id}$ and ε' constructable by a machine without auxiliary pushdown store such that, for each bound S constructable by a machine without auxiliary store with $\lim_{n \to \infty} S = \infty$, there is a constant K_S satisfying the inequality $\varepsilon'(n) - K_S < S(n)$ for all $n \to \mathcal{N}$.

Remark. The function $\varepsilon' - K_S$ is also (m, l)-constructable.

Proof. See Lemmas 4 and 5.

Lemma 7. For each nondecreasing and unbounded recursive function h there is a nondecreasing surjection g such that g is constructable by a (1, 1)-AuxCA with endmarker and $g \leqq h$, $g \leqq \mathrm{id}$.

Proof. Let h be a nondecreasing and unbounded recursive function. For all $n \in \mathcal{N}$, let us define $f(n) = \min\{n, h(n)\}$ and $F(n) = \min\{m \mid f(m) > n\}$. – There is a deterministic Turing machine T with one tape infinite in both directions and with one head such that during the computation on the empty input word T writes the words $0^1 1^{F(1)} 0^{n(1)}$, $0^2 1^{F(2)} 0^{n(2)}$, …, $0^i 1^{F(i)} 0^{n(i)}$, …, where for all $i \in \mathcal{N}$, $n(i)$ is so large that each word written on the tape before $0^i 1^{F(i)} 0^{n(i)}$ is shorter than this word. T proceeds so that the new word of the type $0^i 1^{F(i)} 0^{n(i)}$ is always obtained by rewriting the previous one.

There is a deterministic automaton A with finite control and with two counters which simulates the Turing machine T. Each configuration of T with the head scanning

the first symbol of the word u written on the tape corresponds to a situation of A when the first counter is empty and the second counter contains $[u]$ where $[\ ]$ is a mapping into 1^* which is strictly increasing in the length of argument. A detailed description of this technique can be found in [1].

Now, we shall construct a deterministic (1, 1)-AuxCA M with endmarker. M has an input tape, a worktape with one head using only the symbols b, # and an auxiliary counter. M uses its counter and its input tape in the same way as the automaton A uses its two counters. The left input marker % of M represents the bottom of the second counter of A and the position of the input tape of M gives us the content of this counter of A. The endmarker # of M (on its worktape) is shifted one cell to the right each moment when the word of type $0^m1^{F(m)}0^{n(m)}$ has been completed – in such a moment the input tape of M considered as the second counter of A contains the word $[0^m1^{F(m)}0^{n(m)}]$ (i.e. the input head is in the $[0^m1^{F(m)}0^{n(m)}]$-th cell and the auxiliary counter contains the start symbol only). M stops its computation when its input head reaches the right input marker § for the first time.

Let g be the function constructed by M. Then g is unbounded and nondecreasing and is also a surjection – since the sequence $\{|[0^m1^{F(m)}0^{n(m)}]|\}$ is strictly increasing.

We will prove $g(n) < f(n)$ for all sufficiently large $n \in \mathcal{N}$. We have $g(n) \leqq \leqq \max\{m \mid [0^m1^{F(m)}0^{n(m)}]$ is the content of the input tape of M considered as the second counter of A during the computation of M on the input word $1^n\} \leqq \leqq \max\{m \mid |[0^m1^{F(m)}0^{n(m)}]| \leqq n\} \leqq \max\{m \mid F(m) < n\} = \max\{m \mid \min\{q \mid f(q) > > m\} < n\} < f(n)$.

If the last inequality does not hold then $q_0 = \min\{q \mid f(q) > f(n)\} < n$ and $f(q_0) > > f(n)$ – a contradiction, since f is nondecreasing. Q.E.D.

Theorem 3. Let t be an (m, l)-constructable bound which majorizes a nondecreasing and unbounded recursive function. Then there is a language L such that

(1) $L \subseteq 1^+$,

(2) $L \in TAPE(t, m, l)$,

(3) $L \notin \mathcal{C}SPACE(S, m, l)$, where $S(0) = 1$, $S(n + 1) = t(n)$, $n \in \mathcal{N}$.

Theorem 3a. (Only for the case of machines without auxiliary pushdown store) Let t be an (m, l)-constructable bound with $\lim_{n\to\infty} t = \infty$. Then there is a language L such that (1), (2), (3) from Theorem 3 hold.

Proof of Theorems 3 and 3a. First let us notice that if t is our function from Theorem 3 or 3a then t majorizes an (m, l)-constructable nondecreasing surjection. This follows from Lemmas 6 and 7. The plan of the construction of a machine N whose language has the properties stated in the theorem is the same as in the proof of Theorem 2.

(1) Let us put $Q = \mathscr{S}$ and for $q \in Q$, $L_q = L_S(q)$. Such a set Q is recursive, the graph of the relation $!_Q$ is (A-) recursive.

Let $\{s_i\}$ be an effective sequence of programs from $\mathscr{S}$ such that each $s \in \mathscr{S}$ occurs infinitely many times in it. Before constructing the set R, we must realize that the language accepted by N would be a subset of 1^+ and therefore N would understand unambiguously the input word from 1^+ as a program from the set R enlarged by an additional string of 1's.

We begin by the construction of words v_i. Let $\{m_i\}$ be any sequence of natural numbers and ε' a nondecreasing (m, l)-constructable surjection majorized by t and by identity. We define $v_1 = [\circ s_1 \circ 1^{n_1} \circ x_1 \circ 1 \circ 1^{m_1} \circ]$ where $[\ ]$ is a binary coding of the alphabet $\{1, 0, b, \circ\}$ in $\{b, 1\}$, n_1 is a natural number and

(2) if $1^{n_1} \in L_{s_1} = L_S(s_1)$ then $x_1 = 1$, else $x_1 = 0$. If we have v_i then $v_{i+1} = [\circ s_{i+1} \circ 1^{n_{i+1}} \circ q_{i+1} \circ \overline{i+1} \circ 1^{m_{i+1}} \circ]$, where $n_{i+1} = \min\{n \mid \varepsilon'(n) > |v_i|\}$; $\overline{i+1}$ is the binary code of $i + 1$ and

(2) if $1^{n_{i+1}} \in L_{s_{i+1}} = L_S(s_{i+1})$ then $x_{i+1} = 1$, else $x_{i+1} = 0$.
For $a \in \mathscr{N}$, $a > |v_1|$, let us define $i_a = \max\{i \mid |v_i| < a\}$.

Lemma 8. There are words v_i and a deterministic Turing machine M (with oracle A) such that M has one tape and one head which writes the symbols 1, b (1, b, S) only, its tape is infinite to the right only, and M is such that for all a, $a > |v_1|$, M rewrites, using the input cells only, each word $b^{a-1}1$ into the word: if $|v_{i_a}| < a$ then $v_{i_a} b^{a-1-|v_{i_a}|} 1$ else v_{i_a}.

Proof. Let us describe the main features of the action of the machine M. M starts its computation on the input word $b^{a-1}1$ by constructing the elements $[s_1]$, $[x_1]$, $[1^{n_1}]$ of the word v_1, then it constructs the word v_1 choosing m_1 so large that all squares used in the construction of the elements $[s_1]$, $[1^{n_1}]$, $[x_1]$ now contain symbols of the word v_1. If M has written the word v_i it starts to construct the elements $[s_{i+1}]$, $[1^{n_{i+1}}]$, $[x_{i+1}]$, $[\overline{i+1}]$ of the word v_{i+1} without erasing the word v_i. Then it writes the word v_{i+1}, choosing m_{i+1} so large that all squares having contained the word v_i or used during the construction of the elements $[s_{i+1}], \ldots, [\overline{i+1}]$, now contain symbols of the word v_{i+1}. If the initial segment of the length a is not sufficiently large to construct the word v_{i+1} then $v_{i_a} = v_i$. □

(3) We fix the sequence $\{v_i\}$ from this lemma and define $R = \{1^{n_i} \mid i \in \mathscr{N}\}$ and $L_{1^{n_i}} = L_{s_i} = L_S(s_i)$.

The set R and the graph of the relation $!_R$ are (A-) recursive languages and no program from Q diagonalizes R (see the construction of R and $\{s_i\}$ and Lemma 1).

Let us define, for all $m \in \mathscr{N}$, $k_m = \max\{i \mid |v_i| \leqq \varepsilon'(m)\}$ and $\varepsilon(m) = |v_{k_m}|$. Obviously, we have

Lemma 9. ε is nondecreasing, unbounded, majorized by the identity and (A-) recursive.

Now, we see that the sets of program Q, R and the function ε satisfy the conditions of the rtp-lemma and that therefore we are allowed to choose an rtp RTP with ε on Q, R which is constructive in the sense of the rtp-lemma.

We define a function z' by putting $z'(n_i) = \min\{n \mid \varepsilon'(n_i + n) = |v_i|\}$ for all $i \in \mathcal{N}$ and $z'(n) = 0$ for all $n \in \mathcal{N}$ different from the n_i's. The definition is correct since $\varepsilon'(n_i) < |v_i|$ and ε' is a nondecreasing surjection.

Now we are ready to construct the machine N and to prove that its language has the properties (1), (2) from Theorem 1.

N has l worktape heads, uses m worktape symbols and accepts strings of 1's. It starts computation on the input word 1^n by printing its endmarker into the $t(n)$-th worktape square (t is constructable); this is the last endmarker shift during the computation. We have $L(N) \subseteq 1^+$ and $L(N) \in TAPE(t, m, l)$. – Then N puts the symbol 1 into the $\varepsilon'(n)$-th square (ε' is also constructable) and writes the word v_{k_n} on its worktape – see Lemma 8 and the definition of k_n. If $|v_{k_n}| = \varepsilon'(n)$ then N accepts iff $x_{k_n} = 0$. If $|v_{k_n}| < \varepsilon'(n)$ then N writes the program $RTP(|v_{k_n}|) = RTP(\varepsilon(n))$ on its worktape (rtp-lemma) and erases all the other symbol except the endmarker. Now N, has the program $RTP(\varepsilon(n))$ on its worktape. According to this program, N continues to work as the universal machine U on the input 1^{n+1}. It simulates the endmarker of U by the symbol 1 – the rightmost nonblank to the left from the endmarker which is fixed in the $t(n)$-th cell. N accepts iff there is an accepting computation of the universal machine U on the input 1^{n+1} which does not require more than $t(n)$ cells.

Now we want to apply Theorem 1. We define the mapping z, $z(r) = z'(|r|)$ for all $r \in R$. We shall prove that $r1^{z(r)} \in L(N) \leftrightarrow \neg r!\, r$. Let us choose $r = 1^{n_i}$ arbitrarily and put $n = n_i + z'(n_i)$. During the computation on the input word 1^n, N decides whether $|v_{k_n}| < \varepsilon'(n)$. We know that $\varepsilon'(n) = \varepsilon'(n_i + z'(n_i)) = |v_i| = |v_{k_n}|$. Therefore N accepts iff $x_{k_n} = 0$ iff $x_i = 0$ iff $1^{n_i} \notin L_{1^{n_i}}$ iff $\neg r!\, r$. Thus we have $r1^{z(r)} \in L(N) \leftrightarrow$ $\leftrightarrow \neg r!\, r$.

Further, for all $r \in R$ sufficiently large and for all j, $0 \leqq j < z(r)$, we shall prove $r1^j \in L(N) \leftrightarrow RTP(\varepsilon(|r|))!\, r1^{j+1}$. Let us choose a program $r \in R$, $r = 1^{n_i}$, and a natural number j, $0 \leqq j < z'(n_i) = z(r)$, arbitrarily and put $n = n_i + j$. During the computation on the word 1^n, N decides whether $|v_{k_n}| < \varepsilon'(n)$. We know that

$$|v_i| = \varepsilon'(n_i + z'(n_i)) > \varepsilon'(n) = \varepsilon'(n_i + j) \geqq \varepsilon'(|r|) =$$
$$= \varepsilon'(n_i) > \varepsilon'(n_{i-1} + z'(n_{i-1})) = |v_{i-1}|\,.$$

So $\varepsilon'(n) > |v_{k_n}| = |v_{i-1}|$. Therefore N continues to work as the universal machine U on the input word 1^{n+1} according to the program $RTP(\varepsilon(n)) = RTP(\varepsilon(|r|))$. Let us write s instead of $RTP(\varepsilon(|r|))$. It is clear that the following statements are equivalent.

(i) $r1^j \in L(N)$,

(ii) $tape_{U,s}(r1^{j+1}) \leqq t(n)$ – this follows from the construction of N,

(iii) $tape_{M_r,A}(r1^{j+1}) + |s| \leqq t(n)$ – see Lemma 3,

(iv) $space_s(r1^{j+1}) \leqq t(n) = S(n+1)$ – see the definition of $space_s$,

(v) $r1^{j+1} \in L_S(s)$ – since $|r1^{j+1}| = n+1$,

(vi) $RTP(\varepsilon(|r|))!\, r1^{j+1}$ – since $L_S(s) = L_s$.

We have shown that the language $L(N)$ satisfies the conditions (1), (2) of Theorem 1. Therefore $L(N) \notin \mathscr{CL}(Q) = \mathscr{C}CSPACE(S, m, l)$. Q.E.D.

Corollaries.

Let t be an (m, l)-constructable bound as in Theorem 2 (Theorem 3). There is a language L such that

(1) $L \subseteq 0^+1^+0^*$ $(L \subseteq 1^+)$, and either

Co 1: (2) $L \in TAPE(t, m, l)$,

(3) $L \notin A = \mathscr{C} \cup \{SPACE(S', m, l) \mid \liminf (t(n) - S'(n+1)) \geqq 0\}$,

or

Co 2: (2) $L \in SPACE(t + a, m, l)$ where a is a constant,

(3) $L \notin \mathscr{C}CSPACE(S, m, l)$ where $S(0) = 0$ and $S(n+1) = t(n)$, $n \in \mathcal{N}$, and $L \notin \mathscr{C} \cup \{SPACE(S', m, l) \mid \liminf (t(n) - S'(n+1)) \geqq 0\}$.

Proof of Co 1. Let $B = CSPACE(S, m, l)$ where $S(n+1) = t(n)$. We shall prove $A \subseteq \mathscr{C}B$ and apply Theorem 2 (3). Let us take $L \in A$. There is an $x \in \mathscr{S}$ such that $L \sim L(x) = L_{S'}(x)$, where $\liminf (t(n) - S'(n+1)) \geqq 0$. There is some n_0 such that the inequality $S(n+1) = t(n) \geqq S'(n+1)$ holds for all $n > n_0$. Therefore $L \sim L(x) = L_{S'}(x) \sim L_S(x) \in B$. Hence $L \in \mathscr{C}B$.

Proof of Co 2. Let a be the length of the program of the machine N from the proof of Theorem 2 (3). $L(N)$ belongs to $TAPE(t, m, l)$ and also to $SPACE(t + a, m, l)$. The sets from the condition (3) are the same as the sets from the third conditions of Theorem 2 (3) and Corollary 1.

Let t be an (m, l)- constructable bound as in Theorem 2 (3). If there is a constant b such that the inequality $t(n+1) - t(n) \leqq b$ holds for all $n \in \mathcal{N}$ then there is a language L satisfying the following conditions:

(1) $L \subseteq 0^+1^+0^*$ $(L \subseteq 1^*)$,

(2) $L \in SPACE(t + a, m, l)$, where a is a constant, and either

Co 3: $L \notin \mathcal{S} \cup \{SPACE(S', m, l) \mid \liminf (t(n) - S'(n)) \geqq b\}$,

or

Co 4: $L \notin \mathcal{S}SPACE(t - b, m, l)$.

Proof of Co 3. If $\liminf (t(n+1) - S'(n+1)) \geqq b$ then $\liminf (t(n) - S'(n+1)) \geqq 0$ and Co 2.

Proof of Co 4. Trivial.

Remarks.

It may seem that all applications of Theorem 1 are rather complicated, but this is not the truth. The following provides an example.

Theorem 4. Let t be an (m, l)-constructable bound which majorizes a nondecreasing and unbounded recursive function. Then there is a language L such that

(1) $L \subseteq 0^+1^*$,

(2) $L \in TAPE(t, m, l)$,

(3) $L \notin \mathcal{S}CSPACE(S, m, l)$ where $S(n+1) = t(n)$.

Proof (sketch). Let ε be an (m, l)-constructable nondecreasing and unbounded function which is majorized by the identity and by t. Let us define $\mathcal{Q} = \mathcal{S}, \mathcal{L}(\mathcal{Q}) = CSPACE(S, m, l)$, $\{s_i\}$, $R = \{0^i \mid i > 0, i \in \mathcal{N}\}$, $L_{0^i} = L_S(s_i)$. – A machine N is now constructed in a similar way as the machine N in the previous proofs. N accepts only the words from 0^+1^*. During the computation on the input word 0^i1^j, $i + j = n$, N puts its endmarker into the $t(n)$-th worktape square and tries to decide whether $0^i \in L_{0^i}$. If the segment of the worktape of the length $t(n)$ is sufficient for the decision then N decides and accepts iff $0^i \notin L_{0^i}$, else N computes the number $\varepsilon(i)$ and completes the test on the tape of length $\varepsilon(i)$. Afterwards N works according to the program $RTP(\varepsilon(i))$. This completes our sketch of proof.

What happens if we change the type of machines considered? Let us have machines with read-write input head but let us leave the definition of constructable functions without any changes, i.e. a function is (m, l)-constructable iff it is constructable by an (m, l)-machine with read-only input head. It can easily be shown that Theorems 2, 3 are still valid for the new type of machines.

Let us define, for a word u, a language L, and a family of languages $\mathcal{L}$, $Shadow\ u = 1^{|u|}$, $Shadow\ L = \{Shadow\ u \mid u \in L\}$, $Shadow\ \mathcal{L} = \{Shadow\ L \mid L \in \mathcal{L}\}$.

For the case of nondeterministic machines we can prove the following

Theorem 5. (For nondeterministic machines with read-write input head) Let t be an (m, l)-constructable bound which majorizes a nondecreasing and unbounded recursive function. Then there is a language L such that

(1) $L \subseteq 1^+$,

(2) $L \in TAPE(t, m, l)$,

(3) $L \notin Shadow\ \mathscr{E}\ CSPACE(S, m, l)$, where $S(n) = t(n-1)$ for all $n > 0$.

The proof is the same as in the case of Theorem 3, with slight changes only. First, in the definition of $\mathscr{L}(Q)$ – (1) in the first part of the proof of Theorem 3 – we put "$L_q = Shadow\ L_S(q)$", then in the definition of the words v_i – (2) in the same proof, before Lemma 8 – we put "if $1^{n_i} \in Shadow\ L_S(s_i)$ then $x_i = 1$ else $x_i = 0$", and in the definition of $\mathscr{L}(R)$ – (3) after Lemma 8 – we put "$L_{1^{n_i}} = Shadow\ L_S(s_i)$".

Further, before working as the universal machine U, machine N changes nondeterministically the input word in any word from $\{0, 1\}^*$ of the same length. Finally, there will be a change of some last lines in the proof of Theorem 3, as follows: "The following statements are equivalent.

(i) $r1^j \in L(N)$,

(ii) $(\exists u \in \{0, 1\}^{n+1})\,(tape_{U,s}(u) \leqq t(n))$ – this follows from the construction of N,

(iii) $(\exists u \in \{0, 1\}^{n+1})\,(tape_{M_*,A}(u) + |s| \leqq t(n))$ – by Lemma 3,

(iv) $(\exists u \in \{0, 1\}^{n+1})\,(space_s(u) \leqq t(n) = S(n+1))$ – by definition of $space_s$,

(v) $(\exists u \in \{0, 1\}^{n+1})\,(u \in L_S(s))$,

(vi) $1^{n+1} = r1^{j+1} \in Shadow\ L_S(s)$,

(vii) $RTP(\varepsilon(|r|))!\ r1^{j+1}$ – since $s = RTP(\varepsilon(|r|))$ and $L_s = Shadow\ L_S(s)$".

CHAPTER 3

The main aim of this chapter is to show the relation between the results from Chapter 2 and some results of earlier work of J. I. Seiferas [3] and I. H. Sudborough [5].

We repeat the definitions of complexity classes and constructable functions from these papers, using our notation.

Let M be an (m, l)-machine without endmarker. We define $tape_M(u) =$ minimum of the distinct worktape squares visited during an accepting computation of M on the input word u if M accepts, and $tape_M(u) = \infty$ otherwise. Now, it is possible to define the classes $TAPE(t, m, l)$.

A function f is fully (m, l)-constructable if there is a deterministic (m, l)-machine M without endmarker such that $L(M) = 1^*$ and $tape_M(1^n) = f(n)$.

We know that each function fully constructable by an (m, l)-machine without endmarker is also constructable by an (m, l)-machine with endmarker and that each function constructable by an (m, l)-machine with endmarker is also fully constructable by an $(m, l+1)$-machine without endmarker (the $(l+1)$-st head simulates the endmarker).

Let t be a bound which is fully constructable by an (m, l)-machine without endmarker. Let us suppose $\lim_{n\to\infty} t = \infty$ or, as in Theorem 3, t majorizes a nondecreasing and unbounded recursive function. We define

$$A = \bigcup\{TAPE(S_1, m, l) \mid t(n) - S_1(n+1) \to \infty\}$$

for the case of machines without endmarker,

$$B = \bigcup\{SPACE(S_2, m, l) \mid t(n) - S_2(n+1) \to \infty\}$$

for the case of machines with endmarker,

$$C = TAPE(t, m, l)$$

for the case of machines with endmarker,

$$D = TAPE(t, m, l+1)$$

for the case of machines without endmarker.

Theorems in [3], [4], [5] most frequently state – for machines without oracles – that $D - A \neq \emptyset$, or that $D - A$ contains a language over the alphabet $\{1\}$. We shall prove here that this is implied by our Theorems 2, 3.

Fact 1. $A \subseteq B$.

Proof. Let $L \in A$. Then L is accepted by an (m,l)-machine without endmarker within tape bound S_1, where $t(n) - S_1(n+1) \to \infty$. L is also accepted by an (m, l)-machine M_x with endmarker within the same tape bound S_1 – this implies that L is accepted by M_x within space bound S_2, where $S_2(n) = S_1(n) + |x|$. We have

$$t(n) - S_2(n+1) = t(n) - S_1(n+1) - |x| \to \infty .$$

So $L \in B$.

Fact 2. $C \subseteq D$.

Proof. Let $L \in C$. Then L is accepted by an (m, l)-machine with endmarker within tape bound t. Then L is also accepted by an $(m, l+1)$-machine without endmarker within the same bound t. The $(l+1)$-st head simulates the endmarker and its shifts on the worktape. The other heads detect the new head as in the previous case they detected the endmarker.

Fact 3. The set $C - B$ contains a language L, $L \subseteq 0^+1^+0^*(L \subseteq 1^+)$.

Proof. See Corollary 1.

The fact $D - A \neq \emptyset$ is implied by our Facts 1–3.

We shall try to find an infinite chain of complexity classes between the sets B and C. For the case of machines with endmarker, for all natural numbers k we define

$$E(k) = \{L \mid (\exists s \in \mathscr{S})(L = L(x) \sim L_{t+k}(x))\}.$$

The following Propositions P1-P5 hold for each $k \in \mathcal{N}$.

P1. $E(k) \subseteq C$.

Proof. Let $L \in E(k)$. Then $L \sim L_{t+k}(x) \subseteq L^{t+k}(x)$. We have $L(x) \sim L^{t+k}(x)$. Clearly, there is a machine $M_{x'}$ such that $L(x') = L^{t+k}(x') = L(x) = L$ and a machine $M_{x''}$ such that $L(x'') = L(x'') = L^{t+k}(x') = L$. So $L \in C$.

P2. $E(k) \subseteq E(k+1)$.

Proof. Let $L \in E(k)$. Then $L = L(x) \sim L_{t+k}(x) \subseteq L_{t+k+1}(x) \subseteq L(x)$. So $L \in \in E(k+1)$.

P3. $(\exists a)(E(a) - B \neq \emptyset)$.

Proof. For all $a \in \mathcal{N}$, we know that $E(a) \supseteq SPACE(t + a, m, l)$. Then apply Corollary 2.

(A) In the following let us suppose that there is a constant b such that for all $n \in \mathcal{N}$, $|t(n+1) - t(n)| \leqq b$.

P4. $B \subseteq E(k)$.

Proof. Let $L \in B$. Then $L = L(x) = L_S(x)$, where $t(n) - S(n+1) \to \infty$, and also – according to the supposition (A) – $t(n+1) - S(n+1) \to \infty$. Therefore there is some n_0 such that $t(n) + k \geqq S(n)$ holds for each $n \geqq n_0$. Hence $L_{t+k}(x) \sim L_S(x) = = L(x) = L$. So $L \in E(k)$.

P5. There is an infinite sequence of natural numbers $\{k_i\}$ such that $E(k_i) \subset \subset E(k_{i+1})$, and $E(k_{i+1}) - \mathscr{E} E(k_i) \neq \emptyset$.

Proof. Let us fix k, $k \geqq b$. There is a constant a_k such that for $G = SPACE(t + + k + a_k, m, l)$ and

$$H = \mathscr{E} \cup \{SPACE(S, m, l) \mid \liminf (t(n) + k - S(n)) \geqq b\}$$

the set $G - H$ contains a language over the alphabet $\{0, 1\}$ ($\{1\}$) – see Corollary 3. We know that $E(k + a_k) \supseteq G$ and we shall prove $\mathscr{E} E(k - b) \subseteq H$. Let $L \in E(k - b)$. Then $L = L(x) \sim L_{t+k-b}(x)$. Obviously there is S such that (1) $L = L(x) = L_S(x)$, and (2) $S(n) = t(n) + k - b$ for all sufficiently large n. We have $\liminf (t(n) +$

$+ k - S(n)) \geqq b$. So $L \in H$ and therefore $E(k + a_k) - \mathscr{E} E(k - b) \neq \emptyset$, and $E(k - b) \subset E(k + a_k)$.

For the constant a from P3 let us write $k_0 = a$ and $k_{i+1} = k_i + b + a_{k_i+b}$ for $i \in \mathscr{N}$. Let us verify $E(k_i) \subset E(k_{i+1})$. By writting $k = k_i + b$ we have $E(k_i) = E(k - b)$, and $E(k_{i+1}) = E(k_i + b + a_{k_i+b}) = E(k + a_k)$. Q.E.D.

Under the assumption (A), the propositions P1—P5 can be summed up in the following chain of inclusions

$$A \subseteq B \subset E(k_0) \subset \mathscr{E} E(k_1) \subset \mathscr{E} E(k_2) \subset \ldots \subset \mathscr{E} E(k_i) \subset \ldots \subset C \subseteq D .$$

Remark. Evidently, the following inclusions hold.

$$E(k) = \{L \mid (\exists x \in \mathscr{S})(L = L(x) \sim L_{t+k}(x)\} \supseteq$$
$$\supseteq \{L \mid (\exists x \in \mathscr{S})(L = L(x) = L_{t+k}(x)\} = SPACE(t + k, m, l) \supseteq$$
$$\supseteq SPACE(t, m, l) .$$

Hence we have also an infinite chain of sets $\mathscr{E} E(k_i)$ between $SPACE(t, m, l)$ and $C = TAPE(t, m, l)$.

(Received September 26, 1978.)

REFERENCES

[1] J. H. Hopcroft, J. D. Ullman: Formal Languages and their Relation to Automata. Adison-Wesley, Reading, Mass. 1969.

[2] H. Rogers Jr.: Theory of Recursive Functions and Effective Computability. McGraw-Hill, New York 1967.

[3] J. I. Seiferas: Nondeterministic time and space complexity classes. Project MAC, TR-137, 1974.

[4] J. I. Seiferas: Techniques for separating space complexity classes. JCSS *14* (1977) 1, 73—99.

[5] I. H. Sudborough: Separating tape bounded auxiliary pushdown automata classes. Proceedings of the Ninth Annual ACM Symposium on Theory of Computing, Boulder, Colorado, May 2—4, 1977, pp. 208—217.

[6] B. A. Trachtenbrot: Optimal computations and the frequency pheuomenon of Jablonskij. (In Russian.) Algebra i Logika (Seminar) *4/5* (1965), 79—83.

[7] B. A. Trachtenbrot: About normed signaling functions of computations on Turing machines. (In Russian.) Algebra i Logika *5/6* (1966), 61—70.

[8] S. Žák: Functions realizable on Turing machines with bounded memory. (In Czech.) 1975, unpublished, Master thesis, Faculty of Mathematics and Physics, Charles University.

[9] S. Žák: A space hierarchy of languages. (In Czech.) 1977, unpublished, RNDr thesis, Faculty of Mathematics and Physics, Charles University.

RNDr. Stanislav Žák, Matematický ústav ČSAV (Mathematical Institute — Czechoslovak Academy of Sciences), Žitná 25, 115 67 Praha 1. Czechoslovakia.