Relationships between Pushdown Automata with Counters and Complexity Classes*

by

Burkhard Monien

Institut für Informatik
Universität Hamburg
Hamburg, West Germany

ABSTRACT

In this paper a machine model is defined whose access to the storage cells is controlled by means of address registers. It is shown that every set acceptable by such a machine within time bound $c \cdot n^p$, $p \in \mathbb{N}$, is accepted by a deterministic $2p$-head two-way pushdown automaton which has additional counters of length $\log_2 n$. On the other hand every set acceptable by a deterministic p-head two-way pushdown automaton can be accepted by this machine model within time bound $c \cdot n^p \cdot \log_2 n$.

1. Introduction. The applicability of a programming language depends on the time needed to translate a program into machine language. Therefore the problem has arisen of finding classes of formal languages whose syntax may be analysed within a short time. This research was put into a scheme in 1965 by J. Hartmanis, P. M. Lewis and R. E. Stearns ([6], [7]) who worked out the concept of time complexity and of tape complexity. In the following years many papers were published which show certain relationships between classes of languages (respectively classes of automata) and complexity classes.

In 1971 S. A. Cook found a new way to handle those problems. He used a k-tape Turing machine with an additional pushdown tape as acceptor and defined a new concept of tape complexity by counting only the cells which are used on the k tapes. By this method he obtained a definite relationship between time complexity and tape complexity. His most relevant result to the scope of this paper is the following:

A set L is accepted by a q-head two-way deterministic pushdown automaton (dpda) if and only if there exists a deterministic Turing machine which accepts L in time complexity $P(n)$, where $P(n)$ is a polynomial.

S. A. Cook did not point out in which way q and the degree of the polynomial $P(n)$ depend on each other. A. V. Aho, J. E. Hopcroft, J. D. Ullman and T. Kameda studied these questions.

---

* A result similar to one of the main theorems (theorem 4) of this paper has been proved also by S. A. Cook. Both proofs are based on the same idea but have been found independently.

MATHEMATICAL SYSTEMS THEORY, Vol. 9, No. 3.

Aho, Hopcroft and Ullman [1] showed that if L is a language accepted by a q-head two-way *dpda* then there exists a Turing machine which accepts L in time complexity $n^{2q} \log_2 n$.

T. Kameda [9] showed that if L is accepted by a Turing machine in time complexity n^p then there exists a $(3p+1)$-head two-way *dpda* which accepts L.

Since 1965 it has been primarily the concept of time complexity and tape complexity defined by Hartmanis, Lewis and Stearns that has been used to describe the demand for time and space of an algorithm. However some authors (for example Aho, Hopcroft and Ullman [1] and J. Early [3]) implement their algorithms not on a Turing machine, but measure the time an algorithm demands by counting only the operations which are carried out, presuming that the data used are available without losing time. This method is close to the way real computers work, for real computers calculate an address within a short time before jumping to the corresponding storage cell.

In the present paper we define an automaton which has a register for each working tape and which can take the heads in one step to the addresses given by the registers. In section 2 the "register machine" is defined and some simple results are presented. The concept of time complexity and of address complexity (instead of tape complexity) is defined. In section 3 relationships between complexity classes (relative to the register machine) and m-Counter *PDA* (defined in [9] by T. Kameda) are proved by means of S. A. Cook's idea. No restriction is made on the time function and the length of the counters. In section 4 we consider the important case in which time function and counter length are polynomials. The results gained in this special case improve significantly those known before. In particular, we show that:

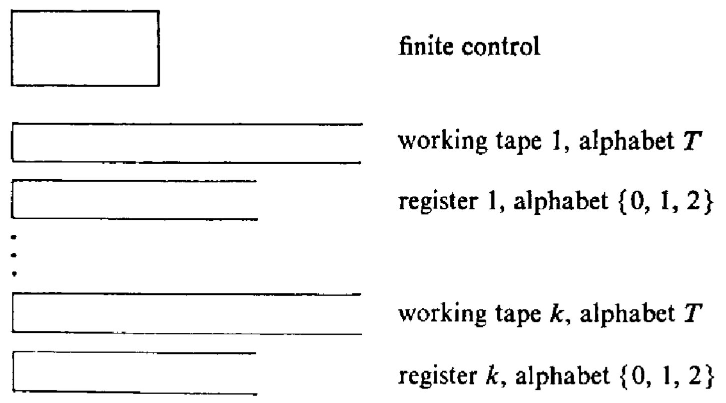
1. If L is accepted by a q-head two-way *dpda*, then there exists a register machine which accepts L in time complexity $c \cdot n^q \cdot \log_2 n$ (c is a constant) and in address complexity $q \cdot \log_2 n + 2 \cdot \log_2 \log_2 n$.

2. If L is accepted by a register machine in time complexity n^p and in address complexity $p \cdot \log_2 n$, then L is accepted by a $2p$-head two-way *dpda* which has additional counters of length $\log_2 n$.

It would be very interesting to find an automaton which accepts exactly the languages which are accepted by a register machine in linear time. The results of the present paper show that such an automaton has a power which is (except additional counters of length $\log_2 n$) between the power of a 1-head two-way *dpda* and of a 2-head two-way *dpda*. Hence, by examining the relationships between context-free languages and p-head two-way dpda ($p = 1$, $p = 2$) we could answer the question whether there is any context-free language not acceptable in linear time by a Turing machine or by a register machine.

2. Machine Model and Preliminary Results. A k-tape register machine M is composed of a finite control and $2k$ tapes which are infinite to the right but have a leftmost cell. The cells of the working tapes are numbered with 1, 2, 3, $\cdots$, beginning with 1 at the leftmost cell. The symbol b is the blank symbol.

A k-tape register machine changes in each move the state and the cells which are scanned by the heads of the $2k$ tapes. Each head may move like the head of a Turing machine, that is, it goes one cell to the left or one cell to the right or


finite control
working tape 1, alphabet T
register 1, alphabet $\{0, 1, 2\}$
working tape k, alphabet T
register k, alphabet $\{0, 1, 2\}$

remains on the same cell. On the other hand, each head of a working tape may jump in one step to the cell whose address is stored in binary notation in the corresponding register on the left hand side of the first occurrence of the symbol 2.

Formal Definition. A *k-tape register machine* is a 5-tuple (T, S, s_0, δ_M, F), where (1) T is a non-empty, finite set (working tape alphabet) with $b \notin T$; (2) S is a non-empty, finite set (set of states); $s_0 \in S$ (start state); $F \subset S$ (set of final states); (3) δ_M is a partial mapping from $S \times ((T \cup \{b\}) \times \{0, 1, 2, b\})^k$ into $S \times (T \times \{-1, 0, +1, R\} \times \{0, 1, 2\} \times \{-1, 0, +1\})^k$. The mapping δ_M is called the next move function.

In each tuple in $S \times (T \times \{-1, 0, +1, R\} \times \{0, 1, 2\} \times \{-1, 0, +1\})^k$ the 3rd, 5th,$\cdots$, $(4k+1)$st components describe the movements of the heads. The symbols $-1, 0, +1, R$ stand for: head moves one cell to the left, does not move, moves one cell to the right, jumps to that cell which is given by the register, respectively.

A *configuration* of M is a $(4k+1)$-tuple $K = (s;\ z_1, \cdots, z_k;\ x_1, \cdots, x_k;\ w_1, \cdots, w_k;\ r_1, \cdots, r_k)$, where $s \in S$; $z_i, x_i \in \mathbb{N}$; $w_i \in (T \cup \{b\})^*$; $r_i \in \{0, 1, 2\}^* - \{\epsilon\}$ for all $i = 1, \cdots, k$.

The machine M is in the configuration K if the following hold for all $i = 1, \cdots, k$: (1) s is the current state of M; (2) the head of working tape i scans the z_ith cell; (3) the head of register i scans the x_ith cell; (4) working tape i stores $w_i = a^i_1 a^i_2 \cdots a^i_{l_i}$, followed by an enumerable number of b's where $z_i = l_i$ or $a^i_{l_i} \in T$ and $z_i < l_i$; (5) register i stores $r_i = c^i_1 \cdots c^i_{\lambda_i}$ followed by an enumerable number of b's. We write $(s;\ z_1, \cdots, z_k;\ x_1, \cdots, x_k;\ w_1, \cdots, w_k;\ r_1, \cdots, r_k) \rightarrow (\bar{s};\ \bar{z}_1, \cdots, \bar{z}_k;\ \bar{x}_1 \cdots \bar{x}_k;\ \bar{w}_1 \cdots, \bar{w}_k;\ \bar{r}_1, \cdots, \bar{r}_k)$ if and only if $\delta_M(s, a^1_{z_1}, c^1_{x_1}, \cdots, a^k_{z_k}, c^k_{x_k}) = (\bar{s}, \bar{a}_1, \gamma_1, \bar{c}_1, \delta_1, \cdots, \bar{a}_k, \gamma_k, \bar{c}_k, \delta_k)$ and the following hold for all $i = 1, \cdots, k$:

(1) If $\gamma_i \in \{-1, 0, +1\}$ then $z_i + \gamma_i \geq 1$ and $\bar{z}_i = z_i + \gamma_i$ else $r_i \in (\{0, 1\}^* - \{\epsilon\}) \cdot \{2\} \cdot \{0, 1, 2\}^*$ and $\bar{z}_i = |r_i|$.

We let $|r_i|$ be that natural number whose binary notation is stored on the left hand side of the first 2 in r_i.

(2) $x_i + \delta_i \geq 1$ and $\bar{x}_i = x_i + \delta_i$.

(3) $\bar{w}_i = a^i_1 \cdots a^i_{\bar{z}_i - 1} \bar{a}_i a^i_{\bar{z}_i + 1} \cdots a^i_{l_i} \underbrace{b \cdots b}_{\mu_i}$, where $\mu_i = \begin{cases} 0, & \text{if } \bar{z}_i \leq l_i \\ \bar{z}_i - l_i, & \text{if } \bar{z}_i > l_i. \end{cases}$

$$(4)\ \bar r_i = \begin{cases} c_1^i \cdots c_{x_i-1}^i \bar c_i c_{x_i+1}^i \cdots c_{\lambda_i}^i, & \text{if } x_i \le \lambda_i \\ r_i \bar c_i, & \text{if } x_i = \lambda_i + 1. \end{cases}$$

We call $K \to \bar K$ a *move* of the register machine M and $\bar K$ the configuration next to K. M *stops* in the configuration K if there is no configuration next to K. $\overset{*}{\to}$ is the transitive closure of $\to$.

Definition. Suppose $M = (T, S, s_0, \delta, F)$ is a k-tape register machine and $w \in T^*$. Let M start in the configuration $(s_0; 1, \cdots, 1; 1, \cdots, 1; w, b, \cdots, b; 12, \cdots, 12)$. We say M *accepts* w if and only if M stops in a final state after a finite number of moves. Let $R(M)$ be the set of all elements of T^* that are accepted by M.

Definition. M accepts $L \subset T^*$ in *time complexity* $T(n)$, if $L = R(M)$ and M stops after at most $T(l(w))$ moves for almost all $w \in T^*$. ($l(w) = n \Leftrightarrow w = a_1 \cdots a_n$, $a_i \in T$ for all $i = 1, \cdots, n$).

Definition. M accepts $L \subset T^*$ in *address complexity* $A(n)$, if $L = R(M)$ and if for almost all $w \in T^*$ and in all configurations, generated by M with input w, only the first $A(l(w))$ cells on the k registers and the first $2^{A(l(w))}$ cells on the k working tapes contain non-blank symbols.

Definition. Let $C_{k-R}(T(n))$ $[C_{k-R}(T(n), A(n))]$ be the family of all sets that are accepted by a k-tape register machine in time complexity $T(n)$ [in time complexity $T(n)$ and in address complexity $A(n)$].

In the same way we define the family $C_{k-T}(T(n))$ taking a deterministic off-line Turing machine with k semi-infinite tapes (see for example Hopcroft-Ullman, [8], page 135) as a machine model.

Every k-tape Turing machine may be identified with a k-tape register machine which never jumps and the registers of which are never altered. Thus we get the following theorem.

THEOREM 1. *For any increasing mapping T: $\mathbb{N} \to \mathbb{N}$ we get*

$$C_{k-T}(T(n)) \subset C_{k-R}(T(n), \mathit{Log}\ T(n)).$$

Definition. For any $n \in \mathbb{N}$ *Log* n is the smallest natural number which is greater than $\log_2 n$.

Obviously $n \le 2^{\mathrm{Log}\, n}$, $\mathrm{Log}\,(n \cdot m) \le \mathrm{Log}\,(n) + \mathrm{Log}\,(m)$.

In general, algorithms run faster on a register machine than on a Turing machine. The next theorem shows what time we may save.

THEOREM 2. *If T, A: $\mathbb{N} \to \mathbb{N}$ are two increasing mappings and if $T(n) \ge n$ for almost all $n \in \mathbb{N}$, then*

$$C_{k-R}(T(n), A(n)) \subset C_{2-T}(T(n) \cdot 2^{A(n)}).$$

Proof. For any $L \in C_{k-R}(T(n), A(n))$ there is a k-tape register machine M_1 which accepts L in time complexity $T(n)$ and in address complexity $A(n)$. We define a two-tape Turing machine M_2 which simulates each move of M_1 in at most $c \cdot 2^{A(n)}$ moves (c is a constant).

Tape 1 of M_2 is divided into $2k$ tracks which contain the same inscriptions as the working tapes and registers of M_1. The method of our proof is nearly the same as in the simulation of $2k$-tape Turing machines on 1-tape Turing machines (see for example Theorem 10.4, [8]). We have only to consider in what way a jump of a head of M_1 is simulated by M_2. In this case M_2 has to decode the corresponding binary address. Machine M_2 starts this computation with the head of tape 1 on the highest digit of the address and with empty tape 2. Then M_2 performs the following computation.

If the head of tape 1 scans the i-th digit of the address and tape 2 stores m in unary notation, where m represents the number given by the higher digits of the address, then M_2 labels the cell scanned on tape 1, goes with its head on tape 1 to the $2m$-th cell, writes $2m$ in unary notation on tape 2, goes with its head on tape 1 to the labelled cell and adds its contents to tape 2, goes (if $i > 1$) with its head on tape 1 to the $(i-1)$-th digit of the address.

At the time this algorithm stops, the address is stored in unary notation on tape 2 and M_2 can move its head on tape 1 to this position. It needs no more than $c\cdot 2^{A(n)}$ moves to do all these operations. □

We will now show what time we lose if we restrict ourself to 1-tape register machines.

THEOREM 3. *If T, A: $\mathbb{N} \to \mathbb{N}$ are two increasing mappings, then*

$$C_{k-R}(T(n), A(n)) \subset C_{1-R}(5\cdot k\cdot T(n)\cdot A(n), A(n)),$$

Proof. For any $L \in C_{k-R}(T(n), A(n))$ there is a k-tape register machine M_1 which accepts L in time complexity $T(n)$ and in address complexity $A(n)$. We define a 1-tape register machine M_2 with $4\cdot k$ tracks on its working tape which simulates each move of M_1 in no more than $5\cdot k\cdot A(n)$ moves. (The idea is similar to the proof of Theorem 10.4, [8].) The $2k$ upper tracks store the inscriptions of the $2k$ tapes of M_1, the $2k$ lower tracks store the positions of the heads in binary notation. Furthermore the contents of the cells which are scanned by M_1 are stored in the finite memory of M_2.

To simulate a move of M_1 the machine M_2 has to perform the following operations:

*1. **Change the stored inscriptions of the working tapes.***

For all $i = 1,\cdots,k$ M_2 carries over the inscription on the $(2k+i)$-th track to the register, jumps on the working tape to the given address and changes the contents of the i-th track as it is determined by the next-move function of M_1. The head on the register goes back to the first cell, M_2 writes 12 on the register and jumps on the working tape to the first cell.

*2. **Change the head position on the working tapes.***

For all $i = 1,\cdots,k$ M_2 changes the inscription on the $(2k+i)$-th track as it is determined by the next move function of M_1 (that is: addition of -1 or 0 or $+1$ or taking over the inscription of the i-th track). Simultaneously the same inscription is written on the register. Then M_2 jumps to the given address and stores the contents of the i-th track in its finite memory. Its head on the register goes back to the first cell, writes 12 on the register and jumps to the first cell.

3. Change of the stored inscriptions of the registers.

The procedure is similar to that described in (1).

4. Change of the head positions on the registers.

The procedure is similar to that described in (2).

For operations (1) and (2) the machine M_2 needs at most $2 \cdot k \cdot A(n)$ moves and for (3) and (4) at most $2 \cdot k \cdot \mathrm{Log}\, A(n)$ moves. Therefore each step of M_1 is simulated by M_2 in no more than $5 \cdot k \cdot A(n)$ moves provided n is big enough. □

If a k-tape register machine accepts a set L in time complexity $T(n)$ then at most $T(n)$ cells on each of the working tapes contain non-blank symbols. Therefore the case $A(n) \leq \mathrm{Log}\, T(n)$ is very important. Theorem 2 and Theorem 3 show:

$$C_{k-R}(T(n), \mathrm{Log}\, T(n)) \subset C_{2-T}(T(n)^2)$$
$$C_{k-R}(T(n), \mathrm{Log}\, T(n)) \subset C_{1-R}(c \cdot T(n) \cdot \mathrm{Log}\, T(n), \mathrm{Log}\, T(n)).$$

The concept of time complexity may be carried over in an obvious way to the computation of functions. A mapping $f\colon \mathbb{N} \to \mathbb{N}$ is called computable in time complexity $T(n)$, if there is a k-tape register machine that starts in the configuration

$$(s_0; 1, \cdots, 1; 1, \cdots, 1; \underbrace{11 \cdots 1}_{n}, b, \cdots, b; 12, \cdots, 12)$$

and for every $n \in \mathbb{N}$ stops after at most $T(n)$ moves in a configuration in which the first working tape contains $\underbrace{11 \cdots 1}_{f(n)}2 \cdots$. Furthermore during all these moves no more than $\mathrm{Log}\, T(n)$ cells on the register and at most the first $T(n)$ cells on the working tapes contain non-blank symbols.

3. Time Complexity of Algorithms. Following T. Kameda [9], we define: A *k-counter-pushdown-acceptor* (*kC-PDA*) M is a 6-tuple $(T, S, s_0, \delta_M, F, c_0)$. It consists of a finite memory (S—set of states, s_0—initial state, F—set of final states) and $k+2$ tapes (T—set of tape symbols, c_0—distinguished tape symbol) with the following properties: (1) tape 1 is a read-only tape. Its head can move in both directions and can move off the input string to the right. (2) tape 2 is a pushdown tape. Its head goes one cell to the left or one cell to the right in each step. (3) tape $3, \cdots$, tape $(k+2)$ are pushdown tapes with only one working symbol (counters). The inscription on the counters always has the form $0\underbrace{11 \cdots 1}_{m}$ with $m \in \mathbb{N} \cup \{0\}$.

The next move function δ_M is defined by:

$$\delta_M(s, a, c, d_1, \cdots, d_k) = (s', \delta, c', d_1', \cdots, d_k'),$$

where

$$s, s' \in S;\ a, c \in T;\ c' \in \{\epsilon\} \cup \{c\} \cdot T;$$

$$\delta \in \{-1, 0, +1\};\ d_i \in \{0, 1\},\ d_i' \in \begin{cases} \{\epsilon, 1, 11\}, & \text{if } d_i = 1 \\ \{0, 01\}, & \text{if } d_i = 0 \end{cases}$$

for all $i = 1, \cdots, k$.

Configurations are defined as in the case of Turing machines. Their notation may be simplified in this case because the positions of the heads on the pushdown tapes are determined by the inscriptions of the tapes. They may be written in the form $(s; i; w, v, l_1, \cdots, l_k)$, where $s \in S$, $i \in \mathbb{N}$ (head position on the input tape), $w, v \in T^*$ (inscriptions on the input tape and on the pushdown tape) and $l_1, \cdots, l_k \in \mathbb{N} \cup \{0\}$.

As usual δ_M induces a partial mapping $\rightarrow$ on the set of configurations.

Definition. A *kC-PDA M accepts* $w \in T^*$, if and only if $(s_0; 1; w, c_0, 0, \cdots, 0) \overset{*}{\rightarrow} (t; l(w); w, \epsilon, l_1, \cdots, l_k)$, where $t \in F$ and $l_1, \cdots, l_k \in \mathbb{N} \cup \{0\}$.

Definition. A *kC-PDA M* accepts $L \subset T^*$ with *counter complexity* $(Z_0(n), Z_1(n), \cdots, Z_k(n))$, if and only if

(1) M accepts $w \Leftrightarrow w \in L$

(2) $(s_0; 1; w, c_0, 0, \cdots, 0) \overset{*}{\rightarrow} (s; i; w, v, l_1, \cdots, l_k)$ implies $i \leq Z_0(l(w))$ and $l_j \leq Z_j(l(w))$ for all $j = 1, \cdots, k$.

Now we prove our first main theorem.

THEOREM 4. *Let L be a language accepted by a 1C-PDA M with counter complexity $(Z_0(n), Z(n))$, where $Z_0(n)$ and $Z(n)$ are increasing and unbounded functions which are computable with time complexity $Z_0(n) \cdot Z(n)$. Then there is a constant $d > 0$ such that*

$$L \in C_{5-R}(T(n), A(n))$$

with

$$T(n) = d \cdot Z_0(n) \cdot Z(n) \cdot Log\,(Z_0(n) \cdot Z(n))$$

and

$$A(n) = Log\, Z_0(n) + Log\, Z(n) + (1+\epsilon)\, Log\, Log\,(Z_0(n) \cdot Z(n))$$

for an arbitrary $\epsilon > 0$.

Proof. Let $L \subset T^*$ and let $w = a_1 a_2 \cdots a_n \in T^*$, each $a_i \in T$, and $l(w) = n$.

Following the idea of Aho-Hopcroft-Ullman, [1], we define a partial mapping E: $S \times \{1, \cdots, Z_0(n)\} \times T \times \{1, \cdots, Z(n)\} \rightarrow S \times \{1, \cdots, Z_0(n)\} \times \{1, \cdots, Z(n)\}$ by $E(s, i, c, l) = (s', i', l')$ if and only if $(s; i; w, c, l) \overset{*}{\rightarrow} (s'; i'; w, \epsilon, l')$ and $1 \leq i, i' \leq Z_0(n)$, $0 \leq l, l' \leq Z(n)$.

In our proof we change this definition slightly and replace $Z_0(n)$, $Z(n)$ by $2^{\mathrm{Log}\, Z_0(n)}$, $2^{\mathrm{Log}\, Z(n)}$.

M accepts w if and only if there exist $t \in F$ and $l \in \mathbb{N} \cup \{0\}$ with $E(s_0, 1, c_0, 0) = (t, n, l)$.

If $E(s, i, c, l) = (s', i', l')$ holds and not $(s; i; w, c, l) \rightarrow (s'; i'; w, \epsilon, l')$, then there exist $s_1, s_2 \in S$; $i_1, i_2, l_1, l_2 \in \mathbb{N}$; $c_1 \in T$ with the property

$$(*) \qquad \begin{cases} (s; i; w, c, l) \rightarrow (s_1; i_1; w, cc_1, l_1) \overset{*}{\rightarrow} \\ \overset{*}{\rightarrow} (s_2; i_2; w, c, l_2) \overset{*}{\rightarrow} (s'; i'; w, \epsilon, l') \text{ and} \\ E(s_1, i_1, c_1, l_1) = (s_2, i_2, l_2) \\ E(s_2, i_2, c, l_2) = (s', i', l') \end{cases}$$

We say that (s_1, i_1, c_1, l_1) and (s_2, i_2, c, l_2) generate (s, i, c, l) if and only if $(*)$ holds.

The register machine $\bar{M}$, which we will define below, does not simulate the moves of M step by step but uses its storage possibilities to avoid the repeating of computations. This is possible because of the following reasons: If $(s_0; 1; w, c_0, 0) \xrightarrow{*} (s; i; w, vc, l)$ holds for some $s \in S$, $i \in \mathbb{N}$, $c \in T$, $v \in T^*$ and $l \in \mathbb{N} \cup \{0\}$, then w can be accepted by M only if M erases c from its pushdown tape during its further computations and these operations only depend on s, i, w, c and l but not on v. More formally, if $(s; i; w, c, l) \xrightarrow{*} (s'; i'; w, \epsilon, l')$ for some $s' \in S$, $i' \in \mathbb{N}$, $l' \in \mathbb{N} \cup \{0\}$, then $(s; i; w, vc, l) \xrightarrow{*} (s'; i'; w, v, l')$ for all $v \in T^*$. For all $v \in T^*$ occurring in a configuration of the upper form M performs these intermediate operations again, whereas $\bar{M}$ performs them only once and stores the result in the form $E(s, i, c, l) = (s', i', l')$.

Our algorithm first computes all 7-tuples (s, i, c, l, s', i', l') such that $(s; i; w, c, l) \rightarrow (s'; i'; w, \epsilon, l')$ and afterwards it generates new elements of $\bar{E} = \{(s, i, c, l, s', i', l') | E(s, i, c, l) = (s', i', l')\}$ by means of $(*)$. It is controlled as in [1] which elements of $\bar{E}$ are chosen in order to generate new elements, and in [1] it is verified that all elements of $\bar{E}$ are computed during this algorithm.

Now we show how the 5 tapes of $\bar{M}$ are organized.

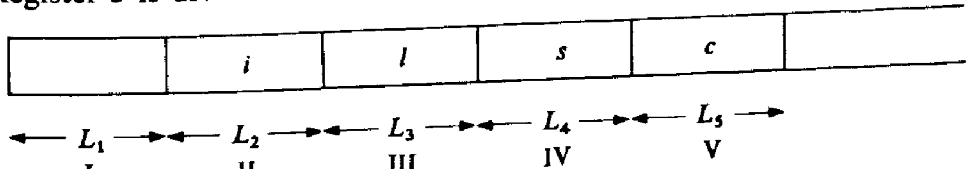
Tape 1 is the input tape (read-only tape).

Tape 2 works like a pushdown tape and stores all elements of $\bar{E}$, that have been computed before, and to which $(*)$ was not yet applied.

Tape 3 stores the elements of $\bar{E}$, that have been computed before, in a form which allows to reach them without losing time. We use the numbers $L_1 = \mathrm{Log}\,(\mathrm{Log}\,|S| + \mathrm{Log}\,Z_0(n) + \mathrm{Log}\,Z(n))$, $L_2 = \mathrm{Log}\,Z_0(n)$, $L_3 = \mathrm{Log}\,Z(n)$, $L_4 = \mathrm{Log}\,|S|$ and $L_5 = \mathrm{Log}\,|T|$. Working tape 3 has two tracks. The lower one stores L_1, L_2, L_3, L_4, L_5 in unary notation. If $E(s, i, c, l)$ has been computed before, then the upper track stores the 3-tuple $(s', i', l') = E(s, i, c, l)$ in the cells $j + i \cdot 2^{L_1} + l \cdot 2^{L_1+L_2} + |s| \cdot 2^{L_1+L_2+L_3} + |c| \cdot 2^{L_1+L_2+L_3+L_4}$, $j = 1, 2, \cdots, 2^{L_1}$. Otherwise these cells are empty (that is they contain the blank symbol b). In this definition we have used two mappings:

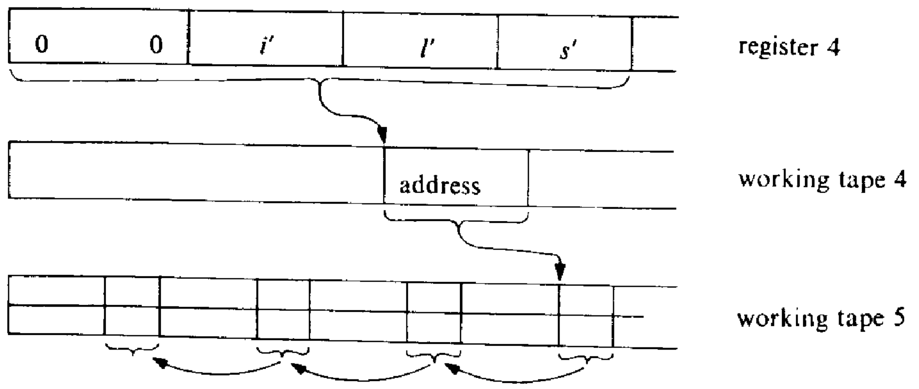
$$\begin{cases} S \rightarrow \mathbb{N} \\ s \rightarrow |s| \end{cases}, \quad \begin{cases} T \rightarrow \mathbb{N} \\ c \rightarrow |c| \end{cases}.$$

Register 3 is divided into 5 segments.

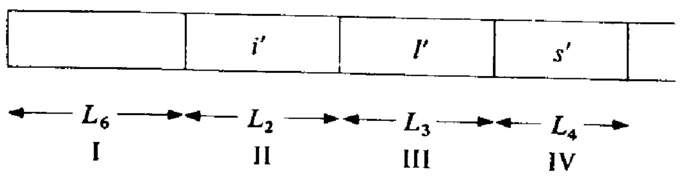


If we write 1, i, l, s, c in the corresponding segments of the register (this needs $L_1 + L_2 + L_3 + L_4 + L_5$ moves) then we can jump in one step to the leftmost of those cells which store $E(s, i, c, l)$.

Tape 4 and *tape 5* store the inverse function of E, which is defined by $EI(s', i', l') = \{(s, i, c, l) | E(s, i, c, l) = (s', i', l')\}$. All 4-tuples out of $EI(s', i', l')$ which have been computed before, are stored by means of address references on working tape 5. Working tape 5 has two tracks. On its upper track it stores a sequence of 4-tuples. 4-tuples which belong to the same $EI(s', i', l')$ are connected by means of address references. These references are stored beneath of the 4-tuple on the lower track. Tape 4 contains for each (s', i', l') the address (on tape 5) of the leftmost cell of that 4-tuple out of $EI(s', i', l')$ which has been computed as the last one. Actually on tape 5 this 4-tuple is stored on the right hand side of any other element out of $EI(s', i', l')$.



Register 4 is divided into 4 segments



Where $L_6 = \text{Log Log}\,(|S|\cdot|T|\cdot 2^{L_2+L_3}\cdot(L_2+L_3+L_4+L_5))$.

Because $EI(s', i', l') \cap EI(s, i, l) = \varnothing$ if $(s', i', l') \neq (s, i, l)$, there are at most $|S|\cdot|T|\cdot 2^{L_2+L_3}$ 4-tuples stored on working tape 5. This needs $|S|\cdot|T|\cdot 2^{L_2+L_3}\cdot(L_2+L_3+L_4+L_5)$ cells and therefore each address which is used on tape 5 can be stored in 2^{L_6} cells. The cells, which are determined by (s', i', l') on working tape 4, are empty if no tuple out of $EI(s', i', l')$ has been computed up to this time. The first 2^{L_6} cells of working tape 4 stores the address of the first empty cell on working tape 5.

$\bar{M}$ starts with w on the input tape. The other tapes are empty. Then $\bar{M}$ performs the following computations.

1. Computation of $L_1, L_2, L_3, L_4, L_5, L_6$.

There exist 1-tape register machines which compute $Z_0(n)$ and $Z(n)$ in time complexity $d\cdot Z_0(n)\cdot Z(n)\cdot \text{Log}\,(Z_0(n)\cdot Z(n))$ and in address complexity Log $(Z_0(n)\cdot Z(n))$. This is proved as in theorem 3. M simulates these register machines on its tape 2. (The organization of the tapes, that we have described above, is not used before the computations of (1) have been finished.) Afterwards $\bar{M}$ computes $L_1,\cdots, L_6$ and stores them in the lower track of tape 3.

If z is a number stored in unary notation on tape 2, then Log z is computed by successive dividing by two. $\bar{M}$ uses tape 3 to store the intermediate results and tape 4 to store the number of divisions. Then Log z is computed in $2z$ moves. Therefore $\bar{M}$ does all these computations in time complexity $\bar{c}\cdot Z_0(n)\cdot Z(n)\cdot \text{Log}\,(Z_0(n)\cdot Z(n))$ and in address complexity Log $(Z_0(n)\cdot Z(n))$.

2. *"Initial elements" of $\tilde{E}$.*

$\bar{M}$ computes all 4-tuples (s, i, c, l) with the property $(s; i; w, c, l) \to (s'; i'; w, \epsilon, l')$ for some s', i', l' with $1 \leq i, i' \leq 2^{L_2}$ and $0 \leq l, l' \leq 2^{L_3}$.

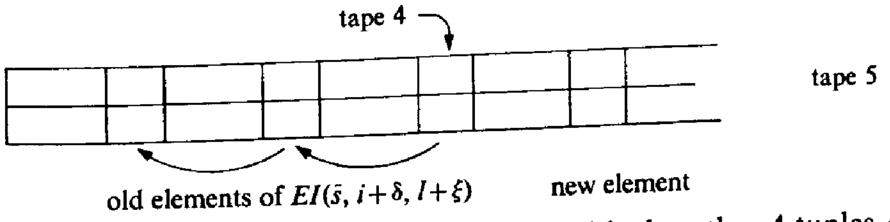
For all i we have to consider only the cases $l = 0$ and $l = 1$, because $(s; i; w, c, 1) \to (s'; i'; w, \epsilon, l')$ is true if and only if $(s; i; w, c, l) \to (s'; i'; w, \epsilon, l-1+l')$ for all $l \geq 1$.

$\bar{M}$ starts with $i = 1$ and increases i during its computations.

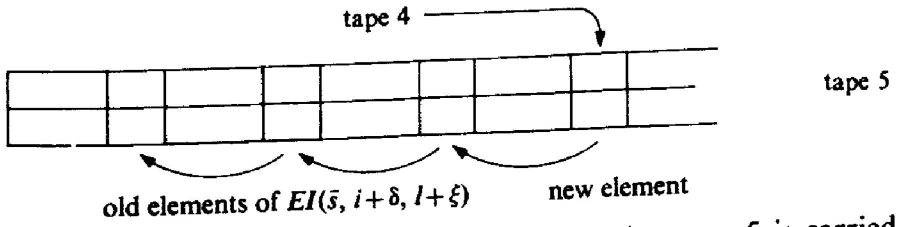
Now let $i \in \{1, \cdots, 2^{L_2}\}$ be fixed and let segment II of register 3 store i in dyadic notation. Let α_i be a_i if $i \leq n$ and b otherwise. For all $(s, c, d) \in S \times T \times \{0, 1\}$ such that $\delta_M(s, \alpha_i, c, d) = (\bar{s}, \delta, \epsilon, \bar{d})$ and $1 \leq i+\delta \leq 2^{L_2}$ for some $\bar{s}, \delta, \bar{d}$; $\bar{M}$ starts the following algorithm with $1 = d$ and $\xi = -1$ if $\bar{d} = \epsilon$, $\xi = 0$ if $\bar{d} = d$, $\xi = +1$ if $\bar{d} = d1$.

(a) Entry on tape 3. $\bar{M}$ writes l, $|s|$, $|c|$ in the segments III, IV, V of register 3 and jumps on working tape 3 to the corresponding cell (that is the cell $i \cdot 2^{L_1} + l \cdot 2^{L_1+L_2} + |s| \cdot 2^{L_1+L_2+L_3} + |c| \cdot 2^{L_1+L_2+L_3+L_4}$). In the cells following to the right $\bar{M}$ writes $(\bar{s}, i+\delta, l+\xi)$.

(b) Entry on tape 4 and tape 5. $\bar{M}$ carries over the address of the first empty cell of working tape 5 to register 5, jumps to the corresponding cell and writes $(i, l, |s|, |c|)$ in the upper track of the cells following to the right. The other 4-tuples out of $EI(s, i+\delta, l+\xi)$ which have been computed so far are stored on tape 5 and connected by means of address references. Tape 4 points to the rightmost one of them.



$\bar{M}$ has to connect the new 4-tuple (s, i, c, l) with the other 4-tuples out of $EI(\bar{s}, i+\delta, l+\xi)$. It writes $i+\delta$, $l+\xi$, $|\bar{s}|$ in the segments II, III, IV of register 4, jumps on tape 4 and tape 5 to the addresses given by the registers and carries over the inscription of the following 2^{L_6} cells of working tape 4 to the lower track of working tape 5. Afterwards $\bar{M}$ jumps on tape 4 to the given address, puts the head of register 5 on the first cell and carries over the contents of register 5 to the following cells of working tape 4.



Finally the address of the first empty cell of working tape 5 is carried over to the first 2^{L_6} cells of working tape 4.

(c) Entry on the pushdown tape (tape 2). $\bar{M}$ writes $(s, i, c, l, \bar{s}, i+\delta, l+\xi)$ on working tape 2.

(d) Increasing of l (if initially $l = 1$). If $0 < l < 2^{L_3}$ then $\bar{M}$ replaces l by $l+1$ and continues at (2) (a).

If all (s, c, d) have been examined and if $i < 2^{L_2}$ then $\bar{M}$ replaces i by $i+1$ and restarts the algorithm.

$\bar{M}$ needs at most $d' \cdot (L_1+L_2+L_3+L_4+L_5+L_6)$ moves to perform the computations of (a), (b), (c), (d) for a fixed i, (s, c, d) and l. Therefore the whole amount of time is bounded above by:

$$\tilde{d} \cdot 2^{L_2} \cdot 2^{L_3} \cdot (L_1+L_2+L_3+L_4+L_5+L_6) \leq \tilde{\tilde{d}} \cdot Z_0(n) \cdot Z(n) \cdot \operatorname{Log}(Z_0(n) \cdot Z(n)).$$

3. *Generation of the rest of $\tilde{E}$.*

$\bar{M}$ reads the first 7-tuple $(s, i, c, l, \bar{s}, \bar{\imath}, \bar{l})$ of the pushdown tape, erases it and writes i, l, $|s|$ on register 4 and $\bar{\imath}$, $\bar{l}$, $|\bar{s}|$ on register 3 into the corresponding segments. c is stored by the finite control. Now $\bar{M}$ computes all tuples which are generated by (s, i, c, l) and by some other tuple (s^*, i^*, c^*, l^*), which has been computed before. Two cases have to be distinguished :

(a) Consider $c_1 \in T$, $s_1 \in S$, $\delta_1 \in \{-1, 0, +1\}$, $\eta_1 \in \{-1, 0, +1\}$ such that $(s_1; i+\delta_1; w, c_1, l+\eta_1) \to (s; i; w, c_1c, l)$ and $E(\bar{s}, \bar{\imath}, c_1, \bar{l})$ has been computed before. If this is true, then $(s_1; i+\delta_1; w, c_1, l+\eta_1) \to (s; i; w, c_1c, l) \xrightarrow{*} (\bar{s}; \bar{\imath}; w, c_1, \bar{l}) \xrightarrow{*} (s'; i'; w, \epsilon, l')$, where s', i', l' are given by: $E(\bar{s}, \bar{\imath}, c_1, \bar{l}) = (s', i', l')$. Therefore $E(s_1, i+\delta_1, c_1, l+\eta_1) = (s', i', l')$.

(b) Consider $c^* \in T$; $s_1, s^* \in S$; $\delta_1, \eta_1 \in \{-1, 0, +1\}$; $i \in \mathbb{N}$, $l \in \mathbb{N} \cup \{0\}$ such that $E(s^*, i^*, c^*, l^*) = (s, i, l)$ has been computed before and $(s_1; i^*+\delta_1; w, c, l^*+\eta_1) \to (s^*; i^*; w, cc^*, l^*)$. If this is true, then $(s_1; i^*+\delta_1; w, c, l^*+\eta_1) \to (s^*; i^*; w, cc^*, l^*) \xrightarrow{*} (s; i; w, c, l) \xrightarrow{*} (\bar{s}; \bar{\imath}; w, \epsilon, \bar{l})$ and we get $E(s_1, i^*+\delta_1, c, l^*+\eta_1) = (\bar{s}, \bar{\imath}, \bar{l})$.

$\bar{M}$ examines all $c_1, s_1, \delta_1, \eta_1$ whether they have the property that was defined in (a). If this is true then we have found an element $(s_1, i+\delta_1, c_1, l+\eta_1, s', i', l')$ of $\tilde{E}$. If this element has not been computed before (to be examined by jumping on tape 3 to the corresponding address), then $\bar{M}$ writes $(s_1, i+\delta_1, c_1, l+\eta_1, s', i', l')$ on the pushdown tape and makes the corresponding entries on tape 3, 4, 5 as it was shown in (2).

If $E(s^*, i^*, c^*, l^*) = (s, i, l)$, then $(s^*, i^*, c^*, l^*) \in EI(s, i, l)$ and all 4-tuples out of $EI(s, i, l)$, which have been computed before, are stored on tape 5 and connected by means of address references. In order to compute all tuples which can be generated by means of (b), $\bar{M}$ jumps on tape 4 to the address which is given by register 4. If the following 2^{L_6} cells are not empty, then $\bar{M}$ carries their contents over to register 5 and starts the following algorithm.

Start: The address stored in register 5 points to a 4-tuple (s^*, i^*, c^*, l^*) with $(s^*; i^*; w, cc^*, l^*) \xrightarrow{*} (s; i; w, c, l) \xrightarrow{*} (\bar{s}; \bar{\imath}; w, \epsilon, \bar{l})$. Now $\bar{M}$ examines for all $s_1 \in S$; $\delta_1, \eta_1 \in \{-1, 0, +1\}$ whether $E(s_1, i^*+\delta_1, c, l^*+\eta_1) = (\bar{s}, \bar{\imath}, \bar{l})$, that means whether $(s_1; i^*+\delta_1; w, c, l^*+\eta_1) \to (s^*; i^*; w, cc^*, l^*)$ holds. If this element of $\tilde{E}$ has not been computed before, then $\bar{M}$ writes the corresponding 7-tuple on the pushdown tape and makes the corresponding entries on tape 3, 4, 5. If the lower track of working tape 5 beneath of (s^*, i^*, c^*, l^*) is not empty, then $\bar{M}$ carries over the contents of these cells to register 5 and continues at Start.

When all the tuples which can be generated by means of (a) and (b) have been computed, $\bar{M}$ performs the operations of 3 once more. This algorithm stops if the pushdown tape is empty. Aho-Hopcroft-Ullman showed in [1] that $\tilde{E}$ has been computed totally at the time the algorithm stops.

We have to estimate the number of moves.

(i) Let $(s, i, c, l, \bar{s}, \bar{i}, \bar{l})$ be a fixed 7-tuple. Then $\bar{M}$ needs no more than $\bar{d}\cdot(L_1+L_2+L_3+L_4+L_5+L_6)$ moves in order to compute all tuples which can be generated by means of (a). These operations are performed only once for each 7-tuple. Therefore the total number of such moves is bounded above by $\bar{d}\cdot|S|\cdot|T|\cdot 2^{L_2}\cdot 2^{L_3}\cdot(L_1+L_2+L_3+L_4+L_5+L_6)$.

(ii) Let $(s, i, c, l, \bar{s}, \bar{i}, \bar{l})$ be a fixed 7-tuple and let $(s^*, i^*, c^*, l^*) \in EI(s, i, l)$ be a fixed 4-tuple. To perform the operations described in (b), $\bar{M}$ needs at most $\bar{d}\cdot(L_1+L_2+L_3+L_4+L_5+L_6)$ moves. The number of different 11-tuples $(s, i, c, l, \bar{s}, \bar{i}, \bar{l}, s^*, i^*, c^*, l^*)$ with $E(s, i, c, l) = (\bar{s}, \bar{i}, \bar{l})$ and $(s^*, i^*, c^*, l^*) \in EI(s, i, l)$ is bounded above by $|T|\cdot|S|\cdot|T|\cdot 2^{L_2}\cdot 2^{L_3}$. This is true because there exist at most $|T|$ 11-tuples with this property if (s^*, i^*, c^*, l^*) is fixed. The operations of (b) are performed only once for each 11-tuple. Therefore the total number of these moves is bounded above by $\bar{d}\cdot|S|\cdot|T|^2\cdot 2^{L_2}\cdot 2^{L_3}\cdot(L_1+L_2+L_3+L_4+L_5+L_6)$.

All time bounds that we got in this proof are bounded above by $c'\cdot Z_0(n)\cdot Z(n)\cdot \text{Log}\,(Z_0(n)\cdot Z(n))$, provided that n is big enough.

The tapes of $\bar{M}$ work with the address complexity: tape 1—Log $Z_0(n)$, tape 2—Log $T(n)$, tape 3—$L_1+L_2+L_3+L_4+L_5$, tape 4—$L_2+L_3+L_4+L_6$, tape 5—Log $(T(n)$. For each $\epsilon > 0$ all these expressions are bounded above by $d_1+\text{Log}\ Z_0(n)+\text{Log}\ Z(n)+\text{Log}\ (c_2+\text{Log}\ Z_0(n)+\text{Log}\ Z(n)) \leq \text{Log}\ Z_0(n)+\text{Log}\ Z(n)+(1+\epsilon)\cdot\text{Log Log}\ (Z_0(n)\cdot Z(n))$, provided that n is big enough. □

It is obvious how theorem 4 may be formulated and proved for kC-PDA, $k > 1$. Now we will show how an algorithm of time complexity $T(n)$ and of address complexity $A(n)$ may be simulated on a kC-PDA.

LEMMA. *A kC-PDA $(k \geq 2)$ is able to do the following computation:*

Start situation: The pushdown tape stores a number m in binary notation. Counter 1 stores a number n.

End situation: The pushdown tape and counter 1 store the same numbers as in the beginning. The finite control stores, whether n is equal to m.

During the computation only the pushdown tape, counter 1 (with maximum length n) and a counter 2 of length Log (Min (n, m)) are used. A detailed proof of this lemma is given in [12].

THEOREM 5. *For each $L \in C_{1-R}(T(n), A(n))$ there exists a $5C$-PDA M that accepts L with counter complexity $(T(n), 2^{A(n)}, A(n), A(n), A(n), \max\,(A(n), \text{Log}\ T(n)))$.*

Proof. I. ***Basic considerations.*** There exists a 1-tape register machine $M_1 = (T, S, s_0, \delta, F)$ that accepts L with time complexity $T(n)$ and in address complexity $A(n)$. We will simulate M_1 on a $5C$-PDA M. The basic idea of the algorithm is the following:

For each $t \in \mathbb{N}$ we define expressions $\tau(t)$ which are determined by the configuration of M_1 after t moves. (Actually these expressions depend not only on t but on two other variables. We give here a simplified approach in order to explain the basic idea.) The pushdown tape of M always stores a sequence of such expressions. If the two expressions which are stored on the top of the pushdown tape belong to the same number t, then M computes $\tau(t+1)$. Otherwise M writes $\tau(0)$ on the top of the pushdown tape. The algorithm starts with $\tau(0)\tau(0)$ on the pushdown tape. We describe how the inscription of the pushdown tape is changed during the first moves of M: $\tau(0)\tau(0) \to \tau(1) \to \tau(1)\tau(0) \to \tau(1)\tau(0)\tau(0) \to \tau(1)\tau(1) \to \tau(2) \to \tau(2)\tau(0) \to \tau(2)\tau(0)\tau(0) \to \tau(2)\tau(1) \to \tau(2)\tau(1)\tau(0) \to \tau(2)\tau(1)\tau(0)\tau(0) \to \tau(2)\tau(1)\tau(1) \to \tau(2)\tau(2) \to \tau(3) \to \cdots \to \tau(t) \to \tau(t)\tau(0) \to \cdots \to \tau(t+1)$.

After the computation of $\tau(t)$ M needs d^{t+1} additional moves to compute $\tau(t+1)$, where d is a constant. (In the simplified case described above exactly 2^{t+1} operations are performed.)

More formally let $\tau(t, j, t')$ be the 9-tuple $(r, t, m, i, j, s, a, \alpha, \beta)$, where $s \in S$, $a \in T$; $r, t, m, i, j \in \{0, 1\}^*$; $\alpha, \beta \in \{0, 1, 2\}$ (t, m, i, j are read as binary notations of natural numbers and are identified with these numbers) and the following holds: (1) After t moves M_1 is in the configuration $(s; m, i; v, u)$, where $v \in T^*$ and $u \in \{0, 1, 2\}^*$. (2) $u = r \cdot \bar{r}$ and $l(r) = j$. (3) a is the content of the m-th working tape cell and α, β are the contents of the i-th and $(j+1)$-th register cell after t' moves of M_1.

(Actually this 9-tuple depends not only on t, j, t' but also on the initial inscription $w \in T^*$ of the working tape of M_1. We describe the algorithm for a fixed $w \in T^*$.)

M starts with $w \in T^*$ on its input tape and with $\tau(0, 0, 0)\tau(0, 0, 0)$ on its pushdown tape, where $\tau(0, 0, 0) = (\epsilon, 0, 1, 1, 0, s_0, a_1, 1, 1)$ and $w = a_1 \cdots a_n$. Its pushdown tape always stores a sequence $(t_1, j_1, t_1')\,(t_2, j_2, t_2') \cdots (t_\mu, j_\mu, t_\mu')$ with $t_\mu' = 0$ and

$$t_\nu' = \begin{cases} t_{\nu+1}, & \text{if } j_{\nu+1} = 0 \\ t_{\nu+1}+1, & \text{if } j_{\nu+1} \neq 0 \end{cases} \quad \left.\text{for all } \nu = 1, \cdots, \mu-1\right\} \qquad (*)$$

Now let $\tau(t, j, t')$, $\tau(t', 0, 0)$ be the 9-tuples on the top of the pushdown tape. Let us assume first, that $t = t'$ and set $\tau(t, j, t') =: (r, t, m, i, j, s, a, \alpha, \beta)$. Then a, α, β are the contents of the m-th working tape cell and of the i-th and $(j+1)$-th register cell after t moves of M_1. Let us assume furthermore that $j = 0$. Then M has all the information which is necessary to simulate the next-move function of M_1.

If $\delta_{M_1}(s, a, \alpha)$ is not defined, then M stops and rejects w. If $\delta_{M_1}(s, a, \alpha) = (\bar{s}, \bar{a}, \delta, \bar{\alpha}, \eta)$ and if $\bar{s} \in F$ then M stops and accepts w. If $\bar{s} \notin F$ and if $\delta \in \{-1, 0, +1\}$ then the 9-tuple $\tau(t+1, 0, 0) = (\epsilon, t+1, m+\delta, i+\eta, 0, \bar{s}, a^*, \alpha^*, 1)$ can be computed, where a^*, α^* are the contents of the $(m+\delta)$-th working tape cell and of the $(i+\eta)$-th register cell in the start configuration of M_1. Before M writes $\tau(t+1, 0, 0)$ on the pushdown tape, it has to give the information about the changing of the m-th working tape cell and the i-th register cell down to the preceding 9-tuple on the pushdown tape in order that $(*)$ holds. That means, the 9-tuple $\tau(t'', j'', t) = (r'', t'', m'', i'', j'', s'', a'', \alpha'', \beta'')$ standing beneath of $\tau(t, 0, t)$ on the pushdown tape has to be replaced by $\tau(t'', j'', t+1)$. This is done by replacing a'' by $\bar{a}$ if $m'' = m$ and α'' by $\bar{\alpha}$ (or β'' by $\bar{\alpha}$) if $i'' = i$ (or $j'' = i$).

On the other hand, if $\delta = R$, then M needs the address which the register of M_1 stores after t moves in order to simulate the $(t+1)$-th move of M_1. At first M replaces the tuple $\tau(t'', j'', t)$ by $\tau(t'', j'', t+1)$ as it was described above and afterwards it generates successively (controlled by the variable j) the first, second, third, 4-th,$\cdots$digit of the address. That means $\tau(t, 0, t)\tau(t, 0, 0) \to \tau(t, 1, 0) = (\beta, t, m, i, 1, s, a^*, \alpha^*, 2)$ [where a^*, α^* are the contents of the m-th working tape cell and of the i-th register cell in the start configuration of M_1] $\to \tau(t, 1, 0)\tau(0, 0, 0) \to \tau(t, 1, 0)\tau(0, 0, 0)\tau(0, 0, 0) \to \cdots \to \tau(t, 1, t')\tau(t', 0, 0)$ for each $0 < t' < t$, $\to \tau(t, 1, t)\tau(t, 0, 0) \to \tau(t, 2, 0) \to \cdots$. If $j \neq 0$ then M replaces $\tau(t, j, t)\tau(t, 0, 0)$ by $\tau(t, j+1, 0) = (r\cdot\beta, t, m, i, j+1, s, a^*, \alpha^*, \beta^*)$ provided that $\beta \neq 2$. If $\beta = 2$ then r represents the address, which M needs to compute the 9-tuple $\tau(t+1, 0, 0) = (\epsilon, t+1, r, i+\eta, 0, s, a^*, \alpha^*, 1)$.

Up to now we considered the case $t = t'$. If $t \neq t'$, then $\tau(t, j, t')$ and $\tau(t', 0, 0)$ are not erased and M writes $\tau(0, 0, 0)$ on the top of the pushdown tape.

During these computations the tuples on the top of the pushdown tape have to be erased in order to decide whether $t = t'$ respectively whether $m = m''$ or $i = i''$ or $i = j''$ (t, t', m, m'', i, i'', j'' as described above) but some of the information which is stored in these tuples has to be used after the comparisons again. This information is stored intermediately by means of the counters of M and by means of the position of the input head.

II. Detailed formulation of the algorithm. In the following $\tau(t', j', 0) = (r', t', m', i', j', s', a', \alpha', 1)$ is the tuple on the top of the pushdown tape and $\tau(t, j, t') = (r, t, m, i, j, s, a, \alpha, \beta)$ is the tuple which stands directly under $\tau(t', j', 0)$ on the pushdown tape. M starts with $w = a_1 \cdots a_n$, $l(w) = n$, on the input tape. The pushdown tape keeps the two tuples $\tau(0, 0, 0)\tau(0, 0, 0)$. Each of its 5 counters store the number 0. M starts its computation with the operations described in (A).

(A) s', a', α' are stored by the finite control. If $j' \neq 0$ (that means: M is in the process of computing an address) then M continues its computation at (C).

If $j' = 0$ then M has to decide whether t is equal to t'. i', m', t' are stored in unary notation by counter 2, counter 1 and the position of the input head. s, a, α, β are stored by the finite control and j is stored by counter 3. First M compares i and i'. Because of the lemma M needs only one additional counter (counter 5) to do this. If $i = i'$ then M compares m and m'. If $i = i'$ and $m = m'$ then M compares t and t'. (If $i \neq i'$ or if $m \neq m'$ then $t \neq t'$.) If $t = t'$ then M continues at (B). Otherwise M writes $\tau(t, j, t')$ and $\tau(t', 0, 0)$ on the top of the pushdown tape and continues at (C).

(B) s, a, α are the state and the contents of the m-th working tape cell and of the i-th register cell after t moves of M_1. If $\delta_{M_1}(s, a, \alpha)$ is not defined then M stops and rejects w. Suppose now $\delta_{M_1}(s, a, \alpha) = (\bar{s}, \bar{a}, \delta, \bar{\alpha}, \eta)$ with $\delta \in \{-1, 0, +1, R\}$ and $\eta \in \{-1, 0, +1\}$. If $\bar{s} \in F$ then M stops and accepts w. Otherwise the three following cases have to be considered.

(1) $j = 0$. If there is a tuple $\tau(t'', j'', t) = (r'', t'', m'', i'', j'', s'', a'', \alpha'', \beta'')$ standing directly under $\tau(t, 0, t)$ and $\tau(t, 0, 0)$ on the pushdown tape, then $\tau(t'', j'', t)$ has to be replaced by $\tau(t'', j'', t+1)$. If $j'' = i$ then β'' is replaced by $\bar{\alpha}$. If $i'' = i$ then α'' is replaced by $\bar{\alpha}$. If $m'' = m$ then a'' is replaced by $\bar{a}$. To perform these operations β'', α'', a'', s'' are stored by the finite control, j'' is stored by

counter 3 and i'' is stored by counter 4 (counter 5 is used to do the comparisons). Now we have to distinguish two cases.

(a) $\delta \in \{-1, 0, +1\}$. Then M erases $\tau(t, 0, t)$ and $\tau(t, 0, 0)$ and writes $\tau(t+1, 0, 0) = (\epsilon, t+1, m+\delta, i+\eta, 0, \bar{s}, a^*, \alpha^*, 1)$ on the top of the pushdown tape. a^*, α^* are the contents of the $(m+\delta)$-th working tape cell and the $(i+\eta)$-th register cell in the start configuration of M_1. M continues at (A).

(b) $\delta = R$. Then M erases $\tau(t, 0, t)$ and $\tau(t, 0, 0)$ and writes $\tau(t, j+1, 0) = (\beta, t, m, i, 1, s, a^*, \alpha^*, 2)$ on the top of the pushdown tape. a^*, α^* are the contents of the m-th working tape cell and the i-th register cell in the start configuration of M_1. M continues at (C).

(2) $j > 0$ and $\beta \neq 2$. Then M erases $\tau(t, j, t)$ and $\tau(t, 0, 0)$ and writes $\tau(t, j+1, 0) = (r \cdot \beta, t, m, i, j+1, s, a^*, \alpha^*, b)$ on the top of the pushdown tape. a^*, α^* are as in (1)(b). M continues at (C).

(3) $j > 0$ and $\beta = 2$. Then M knows the whole address after t moves of M_1. It erases $\tau(t, j, t)$ and $\tau(t, 0, 0)$ and writes $\tau(t+1, 0, 0) = (\epsilon, t+1, r, i+\eta, 0, \bar{s}, a^*, \alpha^*, 1)$ on the top of the pushdown tape. a^*, α^* are the contents of the r-th working tape cell and of the $(i+\eta)$-th register cell in the start configuration of M_1. M continues at (A).

(C) M writes $\tau(0, 0, 0)$ on the top of the pushdown tape and continues at (A).

It is obvious that M accepts the Language L. Its counters are used to store the following numbers: position of the input head —t' ($\leq T(n)$), counter 1—m' ($\leq 2^{A(n)}$), counter 2—i' ($\leq A(n)$), counter 3—j, j'' ($\leq A(n)$), counter 4—i'' ($\leq A(n)$), counter 5 is used to compare m and m', t and t',$\cdots$. Therefore its length is bounded above by Min (Log $T(n)$, Log $2^{A(n)}$). □

4. Multi-Head Pushdown Automata and Languages which are acceptable in Polynomial Time.

Among all the languages that are acceptable by time-bounded register machines those languages which are acceptable in polynomial time are of special interest.

Gray, Harrison, Ibarra defined in [4]: A *q-head two-way deterministic pushdown automaton* (*qh2wdpda*) consists of a pushdown tape and a read-only input tape with q heads which may move in both directions. The next move function is deterministic.

T. Kameda showed in [9] that a language L is accepted by a *qh2wdpda* if and only if there exists a *1C-PDA* which accepts L with counter complexity (n, n^{q-1}).

In the special case which is interesting in this section theorem 4 and theorem 5 may be formulated as follows.

THEOREM 6. *Let L be accepted by a 1C-PDA in counter complexity (n, n^{q-1}), $q \geq 1$. Then there exists a $d > 0$ such that $L \in C_{5-R}(d \cdot n^q \cdot Log\ n, q \cdot Log\ n + (1+\epsilon) \cdot Log\ Log\ n)$ for each $\epsilon > 0$.*

The proof follows immediately from theorem 4.

THEOREM 7. *If $L \in C_{1-R}(d \cdot n^q, q \cdot Log\ n)$, then there exists a 1C-PDA which accepts L with counter complexity $(n, n^{2q-1} \cdot (Log\ n)^4)$.*

Proof. Because of theorem 5 there exists a *5C-PDA* M which accepts L with counter complexity $(d\cdot n^q,\ n^q,\ q\cdot \text{Log } n,\ q\cdot \text{Log } n,\ q\cdot \text{Log } n,\ q\cdot \text{Log } n+d')$, where $d' > 0$ is a constant. First we define a $1C$-PDA M_1 which simulates each move of M and accepts L with counter complexity $(n,\ 2\cdot c\cdot n^{2q-1}\cdot(q\cdot \text{Log } n+1)^3\cdot(q\cdot \text{Log } n+c'))$. Set $r = q\cdot \text{Log } n+1$.

The configuration $(s;\ l_0;\ w,\ v,\ l_1,\ l_2,\ l_3,\ l_4,\ l_5)$ of M is called corresponding to the configuration $(s;\ i;\ w,\ v,\ l)$ of M_1 if $i-1 \equiv l_0 \bmod l(w)$ and $l = l_5+(r+c')\cdot(l_4+r\cdot(l_3+r\cdot(l_2+r\cdot(l_1+(n^q+1)\bar{l}_0))))$, where $\bar{l}_0 = (l_0-i+1)/l(w)$.

If M is in the configuration K and if M_1 is in the configuration corresponding to K and if $K \underset{M}{\to} \bar{K}$ then M_1 is able to compute the configuration corresponding to $\bar{K}$. This is true because M_1 is able to compute r and therefore M_1 gets the numbers $l_5, \cdots, l_0$ by successive dividing of l by r (respectively by $l(w)$). A more detailed proof is given in [12].

The rest of the proof follows because the power of a $1C$-PDA is not changed by multiplying the allowed counter length by a constant factor. □

Theorem 6 and theorem 7 give a good characterization of the class of languages acceptable by register machines in time complexity n^q, $q \in \mathbb{N}$, by means of multi-head two-way deterministic pushdown automata. Any language L, which is accepted by a *qh2wdpda*, is accepted also by a register machine in time complexity $c\cdot n^q\cdot \text{Log } n$ and in address complexity $q\cdot \text{Log } n+2\cdot \text{Log Log } n$. On the other hand any language, which belongs to the class $C_{k-R}(T(n),\ A(n))$ with $T(n) = c\cdot n^q\cdot \text{Log} n$ and $A(n) = q\cdot \text{Log} n+2\cdot \text{Log Log} n$, is accepted by a $(2q)h2wdpda$ which has an additional counter of length $f(n)$, where $f(n) = (\text{Log } n)^\alpha$ for some $\alpha \in \mathbb{N}$. Since Log n is a function which increases very slowly, $\lim_{n\to\infty} f(n)/n^\epsilon = 0$ holds for each $\epsilon > 0$. Therefore the computing power is not very much influenced by multiplying the time complexity by Log n or by adding Log Log n to the address complexity or by adding a counter of length $f(n)$.

Our characterization of time-bounded register machines is better than that one which we know for Turing machines. This is because we only know that a *qh2wdpda*-language is accepted by a Turing machine in time $n^{2q}\cdot \text{Log } n$. Theorem 7 remains true if we replace "register machine" by "Turing machine" and by means of theorem 3 we can get a result about the simulation of k-tape time-bounded Turing machines by pushdown automata. However if we modify the proof of theorem 5 in some points we get a better result.

THEOREM 8. *If* $L \in C_{k-T}(c\cdot n^q)$, *then there exists a* $1C$-PDA *which accepts* L *with counter complexity* $(n,\ n^{2q-1}\cdot \text{Log } n)$.

Because of our characterization of time-bounded register machines, it is possible to get insight into the problem whether a given language belongs to a certain complexity class by studying the class of all the languages which are acceptable by a *qh2wdpda*, where $q \in \mathbb{N}$. It is known that every context-free language is accepted with time complexity n^3 by a Turing machine. It is not known whether there is any context-free language not acceptable in linear time by an off-line Turing machine. The same problem can be stated if "Turing machine" is replaced by "register machine". This leads because of theorem 6, 7 to the question which relationships hold between *qh2wdpda* ($q = 1$, $q = 2$) and context-free languages. J. N. Gray, M. A. Harrison and O. H. Ibarra showed in [4] that *1h2wdpda*-languages are not context-free in general. It is not yet known whether there is any context-free language which is not accepted by a *1h2wdpda*.

Acknowledgement. The author thanks the referee for his helpful suggestions and Mr. Weicker (University of Hamburg) for his careful reading of the manuscript.

REFERENCES

[1] A. V. AHO, J. E. HOPCROFT and J. D. ULLMAN, Time and tape complexity of pushdown automaton languages, *Information and Control* **13** (1968), 186–206.
[2] S. A. COOK, Characterizations of pushdown machines in terms of time-bounded computers, *J. Assoc. Comput. Mach.* **18** (1971), 4–18.
[3] J. EARLEY, An efficient context-free parsing algorithm, *C. Assoc. Comput. Mach.* **13** (1970), 94–102.
[4] J. N. GRAY, M. A. HARRISON and O. H. IBARRA, Two-way pushdown automata, *Information and Control* **10** (1967), 30–70.
[5] M. A. HARRISON and O. H. IBARRA, Multi-tape and multi-head pushdown automata, *Information and Control* **13** (1968), 433–470.
[6] J. HARTMANIS, P. M. LEWIS and R. E. STEARNS, *Hierarchies of Memory Limited Computations*, IEEE Conference Record 1965, 179–190.
[7] J. HARTMANIS and R. E. STEARNS, On the computational complexity of algorithms, *Trans. Amer. Math. Soc.* **117** (1965), 285–306.
[8] J. E. HOPCROFT and J. D. ULLMAN, *Formal Languages and Their Relation to Automata*, Addison-Wesley, Reading, Mass., 1969.
[9] T. KAMEDA, Pushdown automata with counters, *J. Computer and System Sciences* **6** (1972), 138–150.
[10] T. KASAMI, A note on computing time for recognition of languages generated by linear grammars, *Information and Control* **10** (1967), 209–214.
[11] T. KASAMI and T. TORII, A syntax-analysis procedure for unambiguous context-free grammars, *J. Assoc. Comput. Mach.* **16** (1969), 423–431.
[12] B. MONIEN, *Beziehungen zwischen Zeitkomplexitätsklassen und Kellerautomaten mit Zählern*, Berichte Inst. f. Informatik d. Universität Hamburg 71/1.
[13] D. H. YOUNGER, Recognition and parsing of context-free languages in time n^3, *Information and Control* **10** (1967), 189–208.

(*Received 17 May 1972,
and in revised form 4 December 1974*)