About the derivation languages of grammars and machines

Burkhard Monien

Gesamthochschule Paderborn

Fachbereich Mathematik - Informatik

In the same way as for grammars (see f.e. [10]) we define derivation languages also for machines. We show:

1. For each phrase structure grammar $G = (V_N, V_T, S, P)$ with $P \subset V_N^* \times (V_N \cup V_T)^*$ the language of all left derivations is contained in $\mathcal{C}$. [$\mathcal{C}$ is the family of all languages which are acceptable by deterministic two-way counter automata.]

2. For each nondeterministic multi-tape Turing machine the derivation language is contained in $\mathcal{C}$. Especially to each $L \in$ NTIME (n) there exists a language $L_1 \in \mathcal{C}$ and a length preserving homomorphism h with $L = h(L_1)$. [TIME (..) and NTIME (..), respectively, denote the time complexity classes defined by deterministic and nondeterministic multi-tape Turing machines.]

3. For each nondeterministic successor RAM (we define this machine in such a way that it can guess in one step the content of an arbitrary register) the derivation language is contained in TIME $(n \cdot \log n)$ and its complement in NTIME (n) . Especially for the time complexity classes NRAM (..) defined by this machine and for the time complexity classes Nk DIM (..) defined by nondeterministic k-dimensional Turing machines the following holds:

a. $\bigcup_{d \in \mathbb{N}} \text{NRAM}(d \cdot n) \subset \text{NTIME}(n \cdot (\log n)^2)$

b. NTIME (n) closed under complement $\Rightarrow \bigcup_{d \in \mathbb{N}}$ NRAM $(d \cdot n) \subset$ NTIME $(n \cdot \log n)$

c. $\bigcup_{d \in \mathbb{N}}$ N k D I M $(d \cdot n) \subset$ NTIME $(n \cdot (\log n)^2)$.

4. With methods which are similar to those used in [6], we show that NTIME $(n \cdot \log n) \subset \bigcup_{d \in \mathbb{N}}$ NRAM $(d \cdot n)$.

This result and the result from 3. imply:

a. NTIME (n) closed under complement $\Rightarrow \bigcup_{d \in \mathbb{N}}$ NRAM $(d \cdot n)$ closed under complement

b. There exists a simple string matching problem L such that $L \in$ NTIME (n) implies $\bigcup_{d \in \mathbb{N}}$ NRAM $(d \cdot n) =$ NTIME $(n \cdot \log n)$.

The results out of 1. are proved in section 1., the results out of 2. in section 2 and the results out of 3. and 4. in section 3. Some of the proofs can be found in more detail in [9].

1. Left derivations of phrase structure grammars

We use the same notations as in [10] .

Definition: A phrase structure grammar (PSG) is a 4-tuple $G = (V_N, V_T, S, P)$ where V_N, V_T are the alphabets of non-terminal and terminal symbols, $S \in V_N$ is the start symbol and $P \subset ((V_N \cup V_T)^* - V_T^*) \times (V_N \cup V_T)^*$ is the set of productions.

A derivation is a sequence $\varphi_1, .., \varphi_k \in V^*$ $(V = V_N \cup V_T)$ such that for all $i = 1,...,k-1$ there exist $R_1, R_2, Q_1, Q_2 \in V^*$ with $\varphi_i = R_1 Q_1 R_2$, $\varphi_{i+1} = R_1 Q_2 R_2$ and $(Q_1, Q_2) \in P$. Now let M be another set and $\alpha: M \to P$ a bijective mapping. Then to each derivation of G there corresponds a string $\nu \in M^*$ in a natural way. We write: $\varphi_1 \overset{*}{\Rightarrow}_\nu \varphi_k$. The derivation language (Szilard language) of G is defined by $Sz(G) = \{\nu \in M^* \mid \exists \varphi \in V_T^*: S \overset{*}{\Rightarrow}_\nu \varphi\}$

A left derivation is a derivation where $R_1 \in V_T^*$ holds in each step. In this case we write: $\varphi_1 \overset{*}{\Rightarrow}_\nu^{\text{left}} \varphi_k$.

$Sz_{\text{left}}(G) = \{\nu \in M^* \mid \exists \varphi \in V_T^*: S \overset{*}{\Rightarrow}_\nu^{\text{left}} \varphi\}$.

It is easy to see that Sz_{left} (G) is a context-free language for every PSG G. Under the assumption $P \subset V_N^* \times V^*$ Y. Igarashi [7] proved that Sz_{left} (G) $\in$ TAPE (log n). (TAPE (log n) is the class of all languages which are acceptable by a Turing machine operating with a two-way head on its input tape and with the tape bound log n.) We will improve this result in this section.

Definition: A two-way counter automaton $M = (S, X, \delta, s_o, F)$ consists of a finite memory (S-set of states, $s_o \in S$, $F \subset S$), a counter and an input tape (X-set of input symbols) with a two-way read only head. $\delta: S \times X \times \{0,1\} \to S \times \{-1,0,+1\} \times \{-1,0,+1\}$ is the transition function.

A configuration is a 4-tuple (s,w,i,z) with $s \in S$ (state), $w \in X^*$ (input word), $i \in \{1,\dots,l(w)\}$ (head position) and $z \in \mathbb{N}_o$ (number stored by the counter). δ defines in the usual way a mapping $\to$ on the set of configurations. $\overset{*}{\to}$ is the transitive closure of $\to$. M accepts $w \in X^*$ if there exist $t \in F$, $i \in \{1,\dots,l(w)\}$ and $z \in \mathbb{N}_o$ such that $(s_o,w,1,0) \overset{*}{\to} (t,w,i,z)$.

Let C denote the class of all languages which are acceptable by two-way counter automata (notice that these automata work deterministically). It is easy to show that $C \subset$ TAPE (log n).

Now we prove the main result of this section.

Theorem 1: Let $G = (V_N,V_T,P,S)$ be a phrase structure grammar such that $P \subset V_N^* \times V^*$. Then Sz_{left} (G) $\in C$.

Proof: Again let M be the set of markers and $\alpha: M \to P$ the bijective mapping. Then a string $b_1 \dots b_n \in M^*$ belongs to Sz_{left} (G) if and only if $Q_1 = S$ (we set $\alpha(b_i) = (Q_i,R_i)$ $\forall i = 1,\dots,n$) and the two following conditions hold.

(1) For all $i = 2,\dots,n$ there exist $\gamma_i \in V_T^*$ and $\delta_i \in V^*$ such that $S \overset{*}{\underset{b_1 \dots b_{i-1}}{\xrightarrow{left}}} \gamma_i Q_i \delta_i$. (This guarantees that $\alpha(b_i)$ can be applied in the i-th step.)

(2) $\exists w \in V_T^*: S \overset{*}{\underset{b_1 \dots b_n}{\xrightarrow{left}}} w$. (Note that w is determined uniquely if it exists.)

We will show in I how a counter automaton tests (1) and in II how it tests (2).

(I) For all $i = 2,\dots,n$ we will test the correct occurence of Q_i symbol for symbol. We will investigate for all $j = \{1,\dots,l(Q_i)\}$ whether in the string x_i, gained by the left derivation $b_1 \dots b_{i-1}$, (i) the j-th nonterminal symbol of x_i is equal to the j-th symbol of Q_i and (ii) whether for $j > 1$ the symbol preceeding the j-th nonterminal symbol is also a nonterminal symbol.

In order to verify these two conditicus we use the following simple relation

$$S \overset{*}{\underset{b_1 \dots b_k}{\xrightarrow{\text{left}}}} u\, A\, v \text{ with } u,v \in V^*,\ A \in V_N$$

$\Longleftrightarrow \exists\, \lambda \in \{1,\dots,k\},\ \beta \in V_N^*$ and $u_1,u_2,v_1,\beta_1,\beta_2 \in V^*$ with $u = u_1\, u_2$, $v = \beta_2\, v_1$ and $S \overset{*}{\underset{b_1 \dots b_{\lambda-1}}{\xrightarrow{\text{left}}}} u_1\, \beta\, v_1$, $\beta \underset{b_\lambda}{\xrightarrow{\text{left}}} \beta_1\, A\, \beta_2$, $\beta_1 \overset{*}{\underset{b_{\lambda+1} \dots b_k}{\xrightarrow{\text{left}}}} u_2$

Since we consider only left derivationes $u_1 \in V_T^*$. Furthermore $\beta_1 \overset{*}{\underset{b_{\lambda+1} \dots b_k}{\xrightarrow{\text{left}}}} u_2$ holds if and only if there exist $\varphi_\lambda, \dots \varphi_k \in V^*$ such that $\varphi_\lambda = \beta_1$, $\varphi_k = u_2$ and $\varphi_\mu \underset{b_{\mu+1}}{\xrightarrow{\text{left}}} \varphi_{\mu+1}$ $\forall\, \mu = \lambda,\dots,k-1$.

Now suppose $i \in \{2,\dots,n\}$ and suppose our counter automaton $\tilde{M}$ has tested already that the left derivation $b_1 \dots b_{i-1}$ can be carried out. Consider some $j \in \{1,\dots,l(Q_i)\}$. If $\|x_i\| \geq j$ (for arbitrary $v \in V^*$ $\|v\|$ is the number of nonterminals occuring in v) then there exists a decomposition $x_i = u\, A\, v$ with $\|u\| = j-1$ and $A \in V_N$. Let λ, β_1, $\varphi_\lambda,\dots,\varphi_{i-1}$ be determined by the above relation (with $k = i-1$).

In order to determine the j-th nonterminal symbol of x_i, our automaton $\tilde{M}$ starts with the number $j-1 = \|\varphi_{i-1}\|$ and computes (going to the left with its head) successively $\|\varphi_{i-2}\|$, $\|\varphi_{i-3}\|$,.. and stores them on its counter. Note that for all $\mu \in \{\lambda+1,\dots,i-1\}$

$$\varphi_{\mu-1} \underset{b_\mu}{\xrightarrow{\text{left}}} \varphi_\mu \text{ holds and therefore}$$

$\|R_\mu\| \leq \|\varphi_\mu\|$ and $\|\varphi_{\mu-1}\| = \|\varphi_\mu\| - \|R_\mu\| + \|Q_\mu\|$. In the case $\mu = \lambda$ $\beta \underset{b_\lambda}{\xrightarrow{\text{left}}} \varphi_\lambda\, A\, \beta_2$ and therefore $\|R_\lambda\| > \|\varphi_\lambda\|$. In this way a counter automaton which moves its head to the left and uses $\|R_\mu\| \leq \|\varphi_\mu\|$ as interrupt condition can determine the head position λ. Note that the $(\|\varphi_\lambda\|+1)$-st nonterminal of R_λ is the j-th nonterminal symbol of χ_i.

Now we will write down this algorithm for some fixed $i \in \{2,\dots,n\}$ and some fixed $j \in \{1,\dots,l(Q_i)\}$ formally. The algorithm starts with $h = i$ and $z = 0$ (h denotes the head position and z the content of the counter).

```
A 1     $z = j-1$
A 2     If $h = 1$ then STOP
A 3     $h = h-1$
A 4     If $z < \|R_h\|$ then goto A 6
A 5     $z = z - \|R_h\| + \|Q_h\|$, Goto A 2
A 6     If the $(z+1)$-st nonterminal symbol of $R_h$ is equal to the j-th
        symbol of $Q_i$ then RETURN else STOP
```

It should be clear that condition (i) is true if and only if the above algorithm uses its RETURN exit. Now $\tilde{M}$ moves its head back to the i-th cell just by reversing the above process. During these operations $\tilde{M}$ computes (in order to verify (ii)) for each actual head position $\mu \in \{\lambda,\dots,i-1\}$ whether the rightmost symbol of φ_μ is a nonterminal. This can be determined easily in the case $\mu = \lambda$. Now suppose $\varphi_\mu \in V^*V_N$. Then $\varphi_{\mu+1} \in V^*V_N$ iff $\|Q_{\mu+1}\| < \|\varphi_\mu\|$ or $\|Q_{\mu+1}\| = \|\varphi_\mu\|$ and $R_{\mu+1} \in V^*V_N$. Condition (ii) holds for these fixed values i and j iff $\varphi_{i-1} \in V^*V_N$.

(II) We now have to show how a counter automaton verifies that $S \overset{*}{\Rightarrow}{}^{\text{left}}_{b_1\dots b_n} w$ for some $w \in V_T^*$. Consider again the strings $\gamma_\mu \in V_T^*$, $\delta_\mu \in V^*$, $\mu = 1,\dots,n$ defined by $S \Rightarrow^{\text{left}}_{b_1\dots b_{\mu-1}} \gamma_\mu Q_\mu \delta_\mu$. Then $S \Rightarrow^{\text{left}}_{b_1\dots b_n} w$ holds for some $w \in V_T^*$ iff $Q_n \in V_T^*$ and $\delta_n \in V_T^*$.

We will define now a counter automaton which decides, starting from an arbitrary head position $k \in \{2,\dots,n\}$ whether $\delta_k \notin V_T^*$ or computes a head position $\lambda < k$ such that $\delta_k \in V_T^* \Leftrightarrow \delta_\lambda \in V_T^*$. In order to do this let $A \in V_N$ be defined by $Q_k = QA$ and let λ be the number which is defined as in I. by $S \overset{*}{\Rightarrow}{}^{\text{left}}_{b_1\dots b_{\lambda-1}} \gamma_\lambda Q_\lambda \delta_\lambda$, $R_\lambda = \beta_1 A \beta_2$ and $\beta_1 \overset{*}{\Rightarrow}{}^{\text{left}}_{b_{\lambda+1}\dots b_{k-1}} u_2$ with $\gamma_\lambda u_2 = \gamma_k Q$, $\beta_2 \delta_\lambda = \delta_k$.

Note that $\delta_k \in V_T^*$ if and only if $\beta_2 \in V_T^*$ and $\delta_\lambda \in V_T^*$.

We have shown in I how a counter automaton can compute the head position λ and how it computes simultaneously the number $\|\beta_1\|$ on its counter. Therefere it can decide whether $\beta_2 \notin V_T{}^*$. In this case $\delta_k \notin V_T{}^*$, otherwise $\delta_k \in V_T{}^* \iff \delta_\lambda \in V_T{}^*$.

Since $\delta_1 = \varepsilon \in V_T{}^*$ it is clear that by iterating this process a counter automaton can decide whether $\delta_n \in V_T{}^*$. □

2. Derivation languages of Turing machines

Now we will define the notion of derivation languages also for Turing machines. We use notations similar to those in [5].

Definition: A nondeterministic k-tape Turing machine $M = (S,X,T,\delta,s_o,F)$ consists of a finite memory (S-set of states, $s_o \in S$, $F \subset S$) and k tapes (T-set of tape symbols, X-set of input symbols, $X \subset T$) which are bounded to the left and on which a read-write head is moving in both directions. $\delta: S \times T^k \to 2^{S \times (T \times \{-1,0,+1\})^k}$ is the transition function.

A configuration is a(2k+1)-tuple $(s,u_1,i_1,\dots,u_k,i_k)$ with $s \in S$ (state); $u_1,\dots,u_k \in T^*$ (tape inscriptions) and $i_1,\dots,i_k \in \mathbb{N}$ (head positions). δ defines in the usual way a relation $\to$ on the set on configurations. $\overset{*}{\to}$ is the transitive closure of $\to$. M accepts $w \in X$ iff there exist $t \in F$; $u_1,\dots,u_k \in T^*$ and $i_1,\dots,i_k \in \mathbb{N}$ such that $(s_o,w,1,\varepsilon,1,\dots,\varepsilon,1) \overset{*}{\to} (t,u_1,i_1,\dots,u_k,i_k)$.

In order to define a derivation language in analogy to section 1, we set

$$P = \{(\beta,\gamma) \mid \beta \in S \times T^k,\ \gamma \in S \times (T \times \{-1,0,+1\})^k \text{ and } \gamma \in \delta(\beta)\}.$$

Again we could consider also a set M_o of markers and a bijektive mapping $\alpha: M_o \to P$, but because of simplicity we will identify M_o with P.

To each computation of M there is associated in a natural way a $\nu \in P^*$. Let K_1,K_2 be two configurations, let $K_1 \overset{*}{\to} K_2$ hold and let $\nu \in P^*$ be the string describing this computation, then we write $K_1 \overset{*}{\to}_\nu K_2$.

The derivation language of M is defined by

$$Sz(M) = \{\nu \in P^* \mid \exists\ t \in F;\ w \in X^*;\ u_1,\dots,u_k \in T^*;\ i_1,\dots,i_k \in \mathbb{N} \text{ with } (s_o,w,1,\varepsilon,1,\dots,\varepsilon,1) \overset{*}{\to}_{\nu} (t,u_1,i_1,\dots,u_k,i_k)\}$$

Turing machines and phrase structure grammars both accept (generate) just all recursively enumerable sets but nevertheless the derivation languages of Turing machines are in some sense much simpler. This is because a Turing machine changes its configuration in each step only at uniquely determined positions (this is similar to the left derivations in section 1). More over the "local past" of a cell (these are the operations performed by M when it reached this cell for the last time) can be determined quite easily.

Theorem 2: Let M be a nondeterministic k-tape Turing machine. Then $Sz(M) \in C$.

Proof: Let $b_1 \dots b_n \in P^*$ be some string and consider some number $i \in \{2,\dots,n\}$. Let us assume that we already constructed a counter automaton which verified that $b_1 \dots b_{i-1}$ is a correct string (that means that $b_1 \dots b_{i-1}$ describes a computation of M starting from some configuration $(s_o,v,1,\varepsilon,1,\dots,\varepsilon,1)$, $v \in X^*$).

In order to verify that also $b_1 \dots b_i$ is a correct string we have to test whether (1) the state given by the first component of b_i is equal to the state given by the second component of b_{i-1} and (2) whether the tape symbols given by the first component of b_i are equal to the tape symbols which are scanned by the heads of M after M has performed the operations $b_1 \dots b_{i-1}$.

Condition (1) can be tested easily. Condition (2) is tested for each tape separately. Let $H_j(l)$ be the position of the j-th head $(1 \le j \le k)$ after l steps. Our counter automaton moves its head (starting from the i-th cell) back on the input string and computes in each step the distance $|H_j(l) - H_j(i)|$ on its counter (where l is the actual head position). This is controlled in the same way as in the proof of Theorem 1 by the input symbols. Let $\lambda < i$ be the greatest number such that $|H_j(\lambda) - H_j(i)| = 0$, then b_λ gives us the tape symbol we looked for. If such a number λ does not exist then M scans in its i-th step on its j-th tape a cell which has never been scanned before. In this case the corresponding symbol in b_i must be the blank symbol or an input symbol, respectively. It is clear that after performing these operations our counter automaton can put its head in a reverse process on the position i again.

In this way our automaton tests for each $i \in \{2,\dots,n\}$ whether $b_1 \dots b_i$ is correct. A more formal proof is given in [9]. $\square$

It is not difficult to see that from theorem 2 we get a homomorphic descpription of NTIME (n). Remember that
$\mathrm{NTIME}\ (n) = \bigcup_{d \in \mathbb{N}} \mathrm{NTIME}\ (d.n)$, [2].

Corollary 1: For any $L \in \mathrm{NTIME}\ (n)$ there exist $L \in C$ and a length preserving homomorphism h such that $L = h\ (L_1)$.

3. Nondeterministic random access machines

We will prove in this section relations between the complexity classes of nondeterministic random access machines and Turing machines. Again we will use the notion of a derivation language.

Definition: A random access machine (RAM) consists of a finite program which operates on an infinite sequence of registers. Each register can store a natural number. The contents of the registers are denoted by the sequence $x_0, x_1, x_2, \dots$ The program consists of instructions of the type

(1) $X_i \leftarrow X_{X_j}$, (2) $X_{X_i} \leftarrow X_j$, (3) $X_i \leftarrow X_i + 1$,

(4) Goto m if $X_j > 0$, (5) Read X_i, (6) Accept

At the begin of a computation all registers store the number zero. The meaning of the instructions should be clear. The program is written line by line. At the beginning the program counter points to the first line, it is increased by one after the execution of the corresponding instruction unless a jump is caused by instruction (4). Because of more details concerning the definitions of RAMs see [3] or [8].

A configuration is a 3-tuple (m,i,n) with $m \in \mathbb{N}$ (program counter), $i \in \mathbb{N}^*$ (part of the input sequence which has not been read in to this time) and $u \in \mathbb{N}^*$ (sequence of the contents of the registers). The program defines a mapping $\to$ on the set of configurations. $\overset{*}{\to}$ is the transitive closure of $\to$. M accepts $i \in \mathbb{N}^*$ iff there exist $m \in \mathbb{N}$ and $u \in \mathbb{N}^*$ such that $(1,i,\varepsilon) \overset{*}{\to} (m,\varepsilon,u)$ and the instruction Accept stands in the m-th line of the program.

Let X be an alphabet and α: $X \to \mathbb{N}$ a bijective mapping. Let α^*: $X^* \to \mathbb{N}^*$ be the corresponding monoid homomorphism. We say that M accepts $w \in X^*$ iff M accepts $\alpha^*(w)$.

Let T: $\mathbb{N} \to \mathbb{N}$ be a function. Then we denote by RAM (T(n)) the class of all languages over finite alphabets which are acceptable by a RAM which stops for any input of length n, $n \in \mathbb{N}$, after at most T(n) steps. Here a step is the execution of one of the instructions (1) - (6).

Since the only arithmetical operation of our RAM is the operation + 1, a RAM can generate in t steps only numbers which are bounded by t. Therefore RAM (T(n)) $\subset$ TIME $(T(n)^2 \cdot \log T(n))$ holds for every "honest" function T. It is not known whether there exists a faster simulation of deterministic random access machines by Turing machines.

In analogy to the definition of nondeterministic Turing machines we can define nondeterministic RAMs by allowing non-deterministic jumps

(7) Goto m_1 or Goto m_2

We will make our machine even more efficient by allowing that this machine can transfer in one step the content of an arbitrarely chosen register

(8) $X_i \leftarrow \bigvee_{n \in \mathbb{N}} X_n$

Let NRAM (T(n)) be the complexity classes defined in this way. It was shown in [6] that every $L \in$ TIME $(n \cdot \log n)$ is accepted in linear time by a random access machine which can apply the arithmetic operations $x + y$ and $x \dot{-} y$ in one step. We will show that this result holds in the nondeterministic case also for our RAM.

<u>Theorem 3:</u> NTIME $(n \cdot \log n) \subseteq \bigcup_{d \in \mathbb{N}}$ NRAM $(d \cdot n)$

<u>Proof:</u> Consider $L \in$ NTIME $(n \cdot \log n)$. Then there exists ([2]) a

nondeterministic 2-tape Turing machine $M = (S,X,T,\delta,s_0,F)$ which accepts L with the time bound $n \cdot \log_2 n$. We split the tapes of M up into segments of the length $x \cdot \log_2 n$ with some $x \in \mathbb{N}$ which still is to determine. A nondeterministic RAM $\tilde{M}$ is defined in such a way that after an initial phase it simulates $x \cdot \log_2 n$ steps of M in a bounded number of steps (see also [5], proof of Theorem 10.3). We use in this proof that a RAM has direct access to any one-parametric function which has been precomputed and whose table is stored by the RAM.

We use a relation R which relates two pairs $(\varphi_1\, s\, \varphi_2, \varphi_3\, s\, \varphi_4)$, $(\psi_1\, s'\, \psi_2, \psi_3\, s'\, \psi_4)$ with $s,s' \in S$ and $\varphi_i, \psi_i \in T^*$, $l(\varphi_i), l(\psi_i) \geq x \cdot \log_2 n \;\; \forall\, i = 1,\dots,4$ iff $l(\varphi_i\, \varphi_{i+1}) = l(\psi_i\, \psi_{i+1}) = 3\, x \cdot \log_2 n$ for $i = 1,3$ and M can perform the computation $(s, \varphi_1\, \varphi_2, l(\varphi_1) + 1, \varphi_3\, \varphi_4, l(\varphi_3) + 1) \overset{*}{\rightarrow} (s', \psi_1\, \psi_2, l(\psi_1) + 1, \psi_3\, \psi_4, l(\psi_3) + 1)$ in $x \cdot \log_2 n$ steps.

Let us assume first that we are operating on a random access machine which has multidimensional access. Then we use an injective encoding $\gamma: T^*ST^* \rightarrow \mathbb{N}$ and store the relation R on a 4-dimensional array A in such a way that $A(x_1,x_2,y_1,y_2) = 1$ iff

$(\gamma^{-1}(x_1), \gamma^{-1}(x_2))\; R\; (\gamma^{-1}(y_1), \gamma^{-1}(y_2))$ holds.

Now our machine has only onedimensional access but we can overcome this difficulty by a trick.

To encode a multidimensional array by a one dimensional array normally a bijective pairing function $P: \mathbb{N}^2 \rightarrow \mathbb{N}$ is used:
Let $I,J: \mathbb{N} \rightarrow \mathbb{N}$ be the "inverse functions" of P, that means $I(P(x,y)) = x$ and $J(P(x,y)) = y \;\; \forall\, x,y \in \mathbb{N}$. Now let $x,y \in \mathbb{N}$ be some numbers and suppose our RAM has to compute $P(x,y)$. Then it just guesses some $z \in \mathbb{N}$ and verifies afterwards that $I(z) = x$ and $J(z) = y$. So we have only to find a pairing function P such that the corresponding functions I and J can be precomputed quickly on a RAM.

Of course it remains to show that the precomputation can be performed in linear time by our RAM, provided that x is small enough. But this involves mainly technical problems. A detailed proof is given in [9]. □

Now we will show that our nondeterministic RAM is not "much faster" than a nondeterministic Turing machine. We will use again the notion of a derivation language. Note that the set of markers was always chosen in such a form that it encodes in a suitable manner all the information which is necessary in order to perform one step and the alternations induced by this step. In the case of the RAM we choose $M \subset 2^{\mathbb{N} \cup \mathbb{N}^2} \times 2^{\mathbb{N} \cup \mathbb{N}^2}$. M is an infinite set because each natural number may be stored by M in a register during some computation.

The set M belonging to a RAM which uses only numbers $x \leq p$ for some $p \in \mathbb{N}$ as input numbers is defined in the following way.

$$(\varphi, \psi) \in M$$

$\Longleftrightarrow$ There exists exactly one $r \in \mathbb{N} \cap \varphi$ and this r determines (φ,ψ) according to the following schedule. (In this schedule $x,y \in \mathbb{N}$ are arbitrary numbers and $z \leq p$.)

instruction in the r-th line of the program	φ	ψ
$x_i \leftarrow x_{x_j}$	$\{r,(j,x), (x,y)\}$	$\{r+1, (i,y)\}$
$x_{x_i} \leftarrow x_j$	$\{r,(i,x), (j,y)\}$	$\{r+1, (x,y)\}$
$x_i \leftarrow x_i + 1$	$\{r,(i,x)\}$	$\{r+1, (i,x+1)\}$
Goto m if $x_j > 0$	$\{r,(j,x)\}$	$\begin{cases} \{r+1\}, & \text{if } x = 0 \\ \{m\}, & \text{otherwise} \end{cases}$
Read x_i	$\{r\}$	$\{r+1, (i,z)\}$
Accept	$\{r\}$	$\emptyset$
Goto m_1 or Goto m_2	$\{r\}$	$\{m_1\}$ or $\{m_2\}$
$x_i \leftarrow \bigvee_{n \in \mathbb{N}} x_n$	$\{r,(x,y)\}$	$\{r+1, (i,y)\}$
no instruction	$\{r\}$	$\{r\}$

Note that φ stores the content of the registers necessary in order to apply the instruction in the r-th line of the program. [(x,y) encodes: y is the content of register x.] ψ stores the alterations induced by the application of the instruction.

It is clear that each $(\varphi,\psi) \in M$ determines one step of M. If $\nu \in M^*$ is some string and if $K_1 \overset{*}{\to} K_2$ is induced by this string, then we write $K_1 \overset{*}{\to}_\nu K_2$.

$Sz(M) = \{\nu \in M^* | \exists\ i \in \{0,\dots,p\}^*,\ m \in \mathbb{N},\ u \in \mathbb{N}^*$:

$(1, i, \varepsilon) \overset{*}{\to}_\nu (m, \varepsilon, u)$ and "Accept" is the

instruction in the m-th line of the program}.

Now let us denote by Q,Z the two projections from $(2^{\mathbb{N} \cup \mathbb{N}^2})^2$ on its first and second component, respectively. Then the following lemma holds:

<u>Lemma 1:</u> For any nondeterministic RAM M $b_1..b_n \in Sz$ (M)

$\leftrightarrow$ 1. $b_i \in M \quad \forall\ i = 1,\dots,n$

2. $1 \in Q(b_1)$ and $[r_1 \in Z(b_{i-1}) \wedge r_2 \in Q(b_i) \Rightarrow r_1=r_2\ \forall\ i=2,..,n]$

3. For all $i = 1,\dots,n$ the following holds: If $(x,y) \in Q(b_i)$ and if $j = \max\ \{j < i | \exists\ z \in \mathbb{N}: (x,z) \in Z(b_j)\}$ then $y = z$ if $j > 0$ and $y = 0$ otherwise

4. $Z(b_n) = \emptyset$

The proof of this lemma is straight forward. Note that condition 3 guarantees that always the correct contents of the registers are used.

We will construct in the following a Turing machine which accepts Sz (M). First we have to encode the elements of M over a finite alphabet. Let us assume that the elements in the sets φ and ψ are ordered in some way (for example take the order used in the schedule) and let us define a mapping

$\alpha: \mathbb{N} \cup \{(,), \text{","}, \text{"\{"}, \text{"\}"}\} \to \{0,1,2,3,4\}^*$ by: $\alpha(n)$ is the binary notation of n for all $n \in \mathbb{N}$, $\alpha(\text{"("}) = \alpha(\text{")"}) = 2$, $\alpha(\text{","}) = 3$ and $\alpha(\text{"\{"}) = \alpha(\text{"\}"}) = 4$. It is clear that α induces a mapping $\alpha: M \to \{0,\dots,4\}^*$ and let $\alpha^*: M^* \to \{0,\dots,4\}^*$ be the monoid homomorphism induced by α.

Furthermore let L_0 be the following "string-matching" problem:

$L_o = \{d\ \varphi_1\ c..c\ \varphi_q\ d\ \psi_1\ c..c\ \psi_r\ d \mid q,\ r \in \mathbb{N}$ and $\varphi_i,\ \psi_j \in \{0,1,2\}^*$
for all $1 \leq i \leq q$, $1 \leq j \leq r$ and for every $j \in \{1,\ldots,r\}$
there exists a $i \in \{1,\ldots,q\}$ such that $\psi_j = \varphi_i\}$.

Theorem 4: Let M be any nondeterministic RAM. Then

1. $\overline{\alpha^*(Sz\ (M))} \in NTIME\ (n)$
2. Let $T_o: \mathbb{N} \to \mathbb{N}$ be some function such that $L_o \in NTIME\ ((T_o(n))$
   Then $\alpha^*\ (Sz(M)) \in NTIME\ (T_o(n))$

Proof: It is clear that a deterministic Turing machine can decide in linear time for every input $\nu \in \{0,\ldots,4\}^*$ whether there exists $b_1\ldots b_n \in M^*$ such that $\nu = \alpha^*(b_1\ldots b_n)$ and whether the conditions 2. and 4. of lemma 1 hold. Therefore we have to consider in the following only condition 3.

In order to prove 1. note that a nondeterministic Turing machine can guess some $i \in \{1,..,n\}$ and some $(x,y) \in Q(b_i)$, then it can compute the number j and afterwards it can test whether y is the correct value. These operations can be performed in linear time.

In order to prove 2. let a Turing machine transform for each $i \in \{1,\ldots,n\}$ each $r \in (Q(b_i) \cup Z(b_i)) \cap \mathbb{N}$ into ε, each pair $(x,y) \in Q(b_i)$ into $2\alpha(x)2\alpha(y)2$ and each $(x,y) \in Z(b_i)$ into $2\alpha(x)22\alpha(y)2$ and separate the pairs by the symbol c. Let $v = w_1c..cw_m$ be the string generated in this way. Then condition 3 holds if and only if for all $i = 1,\ldots,m$ such that $w_i = 2u_1\ 2u_2\ 2$ with $u_1\ u_2 \in (0,1)^*$ the following holds:
If $j = \max\ \{j < i \mid \exists\ u_3 \in \{0,1\}^*, \nu \in \{2,22\}: w_j = 2u_1\ \nu\ u_3\ 2\}$
then $u_2 = u_3$ if $j > 0$ and $u_2 = 0$ otherwise.

It is clear that this condition can be verified easily if we have a copy of $v = w_1\ c\ldots c\ w_m$ which is ordered according to the first component in such a way that the order of succession for pairs with equal first component is the same as in v. To get this copy we use the language L_o.

First set $w_i' = w_i\ \alpha(i)\ \ \forall i = 1,\ldots,n$. Now our nondeterministic

Turing machine operates in the following way: 1. It computes the string $w_1' \, c \ldots c w_m'$; 2. it guesses some string $\varphi \in \{0,1,2,c\}^*$ and verifies that $d \, w_1' \, c \ldots c \, w_m' \, d \, \varphi \, d \in L_0$ holds; 3. it verifies that $\varphi = u_1 \, c \ldots c \, u_m$ is lexicographically ordered according to the first and third component (Note that $w'_i \neq w_j'$ for $i \neq j$ and therefore $u_1 \, c \ldots c \, u_m$ is just a copy of $w_1' \, c \ldots c \, w_m'$ in a different arrangement) and 4. it verifies that the modification of condition 3 holds.

It is clear that all these operations (with the exception of the membership problem in 2.) can be performed in linear time. □

<u>Corollary:</u>

1. $\bigcup_{d \in \mathbb{N}} \text{NRAM}(d \cdot n) < \text{NTIME}(n \cdot (\log n)^2)$

2. $L_0 \in \text{NTIME}(n) \Rightarrow \bigcup_{d \in \mathbb{N}} \text{NRAM}(d \cdot n) = \text{NTIME}(n \cdot \log n)$

3. If NTIME (n) is closed under complement then $\bigcup_{d \in \mathbb{N}} \text{NRAM}(d \cdot n) =$ NTIME $(n . \log n)$ and $\bigcup_{d \in \mathbb{N}} \text{NRAM}(d \cdot n)$ is closed under complement

<u>Proof:</u> Let $L \in \text{NRAM}(d \cdot n)$ be some language and let M be a RAM which accepts L with the time bound $d \cdot n$. Let $T_1 : \mathbb{N} \to \mathbb{N}$ be some function such that $\alpha^*(Sz(M) \in \text{NTIME}(T_1(n))$.
Then $L \in \text{NTIME}(T_1(d \cdot n \cdot \log_2 n))$.[A nondeterministic Turing machine can guess a derivation string (this string has the length $d \cdot n \cdot \log_2 n$) and afterwards it has only to verify that this string describes a computation of M with the given input.]

If NTIME (n) is closed under complement then $T_1(n) = n$ and therefore $\bigcup_{d \in \mathbb{N}} \text{NRAM}(d \cdot n) = \text{NTIME}(n \cdot \log_2 n)$ follows from Theorem 3. The second statement in 3. follows immediately by using transformational methods.

Furthermore we have proved in Theorem 4 that $T_1(n) \leq T_0(n)$. Therefore 2. follows immediately. In order to prove 1. we have to show that $T_1(n) \leq n \cdot \log_2 n$. Remember that we can sort any sequence

$0\varphi_1,\dots,0\varphi_q,\ 1\psi_1,\dots,1\psi_r \in \{0,1,2\}^*$ in $O(m\cdot\log_2 m)$ steps where $m = \sum_i l(0\varphi_i) + \sum_j l(1\psi_j)$ [see f.e. Mergesort, [1], page 66]. If the sequence is sorted then we can test easily whether $d\varphi_1\ c\dots c\ \varphi_q d\psi_1\ c\dots c\ \psi_r d \in L_o$. □

With just the same methods used in the proof of theorem 4 we can relate also the complexity classes of multidimensional and one-dimensional Turing machines.

Theorem 5: $\bigcup_{k\in\mathbb{N}}$ Nk DIM (2n) $\subset$ NTIME $(n.(\log n)^2)$.

This is considerably different from the deterministic case where we know only [11] that

$$k\ \text{DIM}\ (2n) \subset \text{TIME}\ (n^{2-1/k}.\ \log n).$$

References

1. Aho, A.V., Hopcroft, J.E. and Ullman, J.D.: The Design and Analysis of Computer Algorithms, Addison-Wesley Publ. Comp., Reading - Massachusetts, 1974

2. Bock, R.V. and Greibach, S.A.: Quasi - Realtime Languages, Math. Systems Theory 4, 97-111 (1970).

3. Cook, S.A. and Reckhow, R.A.: Time Bounded Random Access Machines, J. Comp. Syst. Sciences 7, 354-375 (1973).

4. Gray, J.N., Harrison, M.A. and Ibarra, O.H.: Two-Way Pushdown Automata, Information and Control 11, 30-70 (1967).

5. Hopcroft, J.E., and Ullman, J.D.: Formal Languages and their Relation to Automata, Addison-Wesley Publ. Comp., Reading - Massachusetts (1969).

6. Hopcroft, J.E., Paul, W. and Valiant, L.: On time versus space and related problems, 8-th Ann.Symp. Th. Computing, 57-64 (1967).

7. Igarashi, Y.: The Tape-Complexity of Some Classes of Szilard Languages, Report Nr. 81, University of Leeds, England.

8. Monien, B.: Characterizations of time-bounded computations by limited primitive recursion, 2nd colloquium, Automata Languages and Programming, 280-293 (1974).

9. Monien, B.: Über die Komplexität der Ableitungssprachen von Grammatiken und Maschinen, Bericht Nr. 35, Abtl. Informatik, Universität Dortmund (1976).

10. Salomaa, A.: Formal Languages, Academic Press, New York and London, 1973.

11. Stoß, H.J.: Zwei-Band Simulation von Turingmaschinen. Computing 7, 222-235 (1971).