Theoretical Computer Science 132 (1994) 319–336
Elsevier

Deterministic versus nondeterministic space in terms of synchronized alternating machines

Juraj Hromkovič[†]

Department of Mathematics and Informatics, University of Paderborn, 33095 Paderborn, Germany

Branislav Rovan[‡]

Department of Computer Science, Comenius University, 842 15 Bratislava, Slovak Republic

Anna Slobodova[‡]

Institute of Computer Science, Comenius University, 842 15 Bratislava, Slovak Republic

Communicated by A. Salomaa
Received September 1992
Revised June 1993

Abstract

Hromkovič, J. et al., Deterministic versus nondeterministic space in terms of synchronized alternating machines, Theoretical Computer Science 132 (1994) 319–336.

The study of synchronized alternating machines has enabled to characterize several natural complexity classes. It is known that synchronized alternating space $\mathrm{SASPACE}(S(n)) = \bigcup_{c>0}\mathrm{NSPACE}(nc^{S(n)})$ for any (space-constructible) function $S(n)$ [Hromkovič et al. (1991)]. In particular, context-sensitive languages are characterized by two-way synchronized alternating finite automata. Furthermore, PSPACE is characterized by synchronized alternating multihead finite automata and NLOG by synchronized alternating two-way finite automata with parallelism bounded by a constant. In the present paper we prove analogous characterizations for deterministic space classes using a restricted form of synchronization — globally deterministic synchronization. This enables to study the well-known open problems concerning nondeterminism versus determinism as problems about synchronization. We also show that globally deterministic synchronization is strictly more powerful than deterministic synchronization.

Correspondence to: J. Hromkovič, Fachbereich Mathematik-Informatik, Universität Paderborn, 33095 Paderborn, Germany.

[†] This author was partially supported by DFG-Grant Di 412/2-1.

[‡] These authors were partially supported by MŠV SR and EC Cooperative Action IC 1000 project ALTEC.

SSDI 0304-3975(93)E0164-Y

1. Introduction

The synchronized alternating machines (SAM's) have been introduced in [7] as a generalization of alternating machines [1] enabling a restricted form of communication among parallel processes of alternating machines. The goal was to obtain a generalized model of alternating computations, accounting for the possibility of communication among deterministic or nondeterministic parallel processes.

First results have appeared in [20, 21] showing that two-way synchronized alternating finite automata (2SAFA) with constant number of parallel processes recognize exactly languages in NLOG and that synchronized alternating time $\mathrm{SATIME}(T(n)) = \mathrm{ATIME}(T(n))$. A number of characterizations of nondeterministic space classes followed. In [2, 12] it is shown that synchronized alternating space $\mathrm{SASPACE}(S(n)) = \bigcup_{c>0} \mathrm{NSPACE}(nc^{S(n)})$, for any space-constructable function $S(n)$, which for constant $S(n)$ implies a characterization of the context-sensitive languages by 2SAFA. This result was extended in [11] by showing that both one-way and two-way k-head synchronized alternating finite automata ($1\mathrm{SAFA}(k), 2\mathrm{SAFA}(k)$) recognize exactly languages in $\mathrm{NSPACE}(n^k)$, for $k \geqslant 1$. The fact that $\mathrm{SASPACE}(S(n)) = \bigcup_{c>0} \mathrm{SATIME}(c^{S(n)})$, for any space-constructible function $S(n) \geqslant \log n$, [12], can be interpreted so that synchronized alternating machines use space in an optimal way. Analogous results for alternating, nondeterministic and deterministic machines would yield collapse of some fundamental complexity classes. These and other results in [20–24, 2, 9–12, 27] make synchronized alternation an interesting concept in the study of the fundamental complexity classes.

In the present paper we turn our attention to the characterizations of deterministic space classes. We introduce the concept of globally deterministic synchronization. Unlike deterministic synchronization [12, 22] which requires all parallel processes of the synchronized alternating machine to be deterministic, globally deterministic synchronization allows certain degree of "local" nondeterminism. We give the precise definition in the next section. In Section 4 we prove that globally deterministic synchronization is strictly more powerful than deterministic synchronization showing that $\mathrm{DSASPACE}(S(n)) = \mathrm{NSPACE}(S(n)) \subset \mathrm{GDSASPACE}(S(n))$ for any space-constructible function $S(n) \geqslant \log n$. The questions about the relative power of synchronization and globally deterministic synchronization are equivalent to the questions about the relative power of nondeterminism and determinism for space classes ($\mathscr{L}(\mathrm{DCS})?\mathscr{L}(\mathrm{CS})$, $\mathrm{DSPACE}(n^k)?\mathrm{NSPACE}(n^k)$, DLOG?NLOG) as implied by [2, 12] and the main results of this paper.

(1) The globally deterministic synchronized alternating space $\mathrm{GDSASPACE}(S(n)) = \bigcup_{c>0} \mathrm{DSPACE}(nc^{S(n)})$ for any (deterministic space-constructible) function $S(n)$.

(2) The two-way k-head globally deterministic synchronized alternating finite automata ($2\mathrm{GDSAFA}(k)$) recognize exactly $\mathrm{DSPACE}(n^k)$, for $k \geqslant 1$.

(3) The two-way globally deterministic synchronized alternating finite automata (2GDSAFA) with constant number of parallel processes recognize exactly DLOG.

A separation of synchronization from globally deterministic synchronization for (multihead) finite automata would separate the corresponding nondeterministic classes from the deterministic ones. Here, we have proved the separation for one-way finite automata with constant number of parallel processes only.

2. Definitions

Let us first present the basic concept of synchronized alternation as formally described in [12]. We refer to [1] or [8, 16, 17] for a more formal introduction to alternation and stress here only notions important for the following arguments.

Since we want to use the synchronized alternation concept for several types of Turing machines and finite automata, we need to describe this concept for a general machine model. So, let A be a machine whose global state in any moment can be described as a *configuration*(p, α), where p is a state of a finite state control, and α is a finite description of the internal state of A (For instance, α describes the positions of all heads and the contents of all tapes by multitape Turing machines). Further, the formal description of A unambiguously defines a relation $\vdash_A$ on configurations of A corresponding to the move of the machine A from one configuration to another one in one computational step. In what follows, we consider A to be an alternating machine of the above described type, whose states are partitioned into universal, existential, accepting and rejecting states with the usual meaning.

Now, let us introduce synchronized alternation as a generalization of the alternation concept. Informally, we add a finite set of synchronizing symbols to A, and we consider that the state of A is either a state of the finite control or a pair (state of A, synchronizing symbol) called a synchronizing state. The idea of the use of synchronizing state is as follows. When a process (a copy of A) enters a synchronizing state with a synchronizing symbol S, it stops and waits until all processes (copies of A) running in parallel either enter states with the same synchronizing symbol S or stop in final states. After all active processes have reached the same synchronizing symbol they may continue in their work.

To carefully describe the new concept of globally deterministic synchronization, we start by giving the precise definition of an accepting computation of a synchronized alternating machine.

Definition 2.1. The *full configuration tree* of a synchronized alternating machine SAM A on an input word w is a (possibly infinite) labelled tree T_w^A such that,

(i) each node t of T_w^A is labelled by some configuration $c(t)$ of A,

(ii) for the root t_0 of T_w^A, $c(t_0)$ is the initial configuration of A on w, and

(iii) if t_2 is a direct descendant of t_1 in T_w^A, then $c(t_1) \vdash_A c(t_2)$, and if $c(t_1) \vdash_A c$, then there is a unique direct descendant t_2 of t_1 such that $c(t_2) = c$.

Taking all descendants of universal configurations and exactly one of existential configurations gives a subtree representing a computation of an alternating machine as considered usually. It can be viewed as a set of computations of independent "copies" of the original machine (called "processes" in what follows), working in parallel and splitting in universal configurations. The computations of synchronized alternating machines fulfills the following additional condition. Each time one of the machines working in parallel enters a synchronizing state it must wait until all other machines working in parallel either enter an accepting state or a synchronizing state with the same synchronizing symbol. When this happens all the machines are allowed to move from the synchronization states. We shall make this more precise now.

Definition 2.2. The *synchronizing sequence of a node t*, $s_w^A(t)$ [*shortly* $s(t)$] in a full configuration tree T_w^A with the root t_0 is the sequence of synchronizing symbols occuring in labels of the nodes on the path from t_0 to t. The *synchronizing depth of a node t*, $|s(t)|$, is the length of $s(t)$.

In what follows, a configuration (p, α) is called *existential, universal, synchronizing* resp. if p is an existential, universal, synchronizing state respectively. Note, that without loss of generality, one may assume that no SAM A contains existential synchronizing states, i.e. that all synchronizing states are universal states.

Definition 2.3. A *computation tree* of a SAM A on an input word w is a maximal subtree T' of the full configuration tree T_w^A of A on w such that

(i) the root of T' is the root of T_w^A,

(ii) each node in T' labelled by a universal configuration has the same direct descendants as in T,

(iii) each node in T' labelled by an existential configuration has at most one direct descendant,

(iv) for arbitrary nodes t_1 and t_2 the synchronizing sequence of t_1 is an initial subsequence of the synchronizing sequence of t_2 or vice versa.

For machines without existential states the full configuration tree satisfies (i) and (ii) of the above definition. If it happens to satisfy (iv) as well it is the unique computation tree of A on w. Having all parallel processes deterministic (i.e., A is without existential states) makes SAM in this case a natural model of practical parallel computations with restricted type of communication among the parallel processes. We shall call such a machine a *deterministic synchronized alternating machine* (DSAM).

Definition 2.4. An *accepting computation* of a SAM A on an input word w is a finite computation tree of A on w such that each leaf node is labelled by an accepting configuration.

We shall now introduce three technical notions used in the proofs later on. They are meant to capture the fact that unlike in the case of alternating machines in the case of

synchronized alternating machines arbitrary two configurations on parallel branches of the full configuration tree are not necessarily reachable "in the same instant of time".

Definition 2.5. A *meaningful cut* of a computation tree T_w^A is a set Z of nodes in T having the following properties.

(i) There is a positive integer d such that all nodes in Z have depth d or $d-1$,

(ii) all nodes of Z labelled by a synchronizing configuration have the same depth d, and

(iii) every infinite path of T_w^A from the root and every path of T_w^A from the root to a leaf node with synchronization depth greater than $d-1$ contains exactly one node of Z.

Let Z_1 and Z_2 be two meaningful cuts of a computation tree T. We say that Z_1 is a *successor* of Z_2 iff for every $v \in Z_1 - Z_2$ there exists a node u in Z_2 such that v is a direct descendant of u in T.

Definition 2.6. A *synchronization cut* of a computation tree T is a meaningful cut containing nodes labelled by synchronizing configurations only.

Definition 2.7. For a given meaningful cut Z, let $cm(Z)$ be the multiset of the labels of the nodes in Z — *the meaningful cut configurations multiset* (MCCM), and let cs(Z) be the set of the labels of nodes in Z — *the meaningful cut configurations set* (MCCS). Let Z_1 be a successor of Z_2 in a computation tree T. Then the MCCS $\mathrm{cs}(Z_1)$ is called a *successor* of the MCCS $\mathrm{cs}(Z_2)$ in T.

Let us now define the notion of globally deterministic synchronization. It will involve two conditions, one to be fulfilled by the machine itself, the other to be fulfilled by the computations.

Definition 2.8. A synchronized alternating machine A is said to be *globally deterministic* (GDSAM), if

(i) for every existential configuration (p, α) of A the set $S(p,\alpha) = \{(r,\beta) \mid (p,\alpha) \vdash_A (r,\beta)\}$ is equal to a set $\{(r_1,\beta_1),(r_2,\beta_2), \ldots,(r_k,\beta_k)\}$ consisting of synchronizing universal configurations only, where $\alpha = \beta_i$ for every $i \in \{1, \ldots, k\}$, $r_i = (p_i, Q_i)$ for a synchronizing symbol Q_i, and $Q_i \neq Q_j$ for $i \neq j$, $i,j \in \{1, \ldots, k\}$

(ii) for each w, each synchronization cut of a computation on w contains a node which is the direct descendant of a node labelled by a universal configuration.

The condition (i) is a condition on the transition function of A and implies that different nondeterministic decisions must be connected with a choice of different synchronizing symbols. The condition (ii) is a condition on the computation tree assuring the unique choice of descendants for existential nodes given by the synchronizing descendant of a universal configuration. Note that for different synchronizing cuts the synchronizing symbol may be enforced by different processes.

Observation 2.9. *For every globally deterministic synchronized alternating machine A and every word w in $L(A)$, there is a unique accepting computation tree of A on w.*

Proof. Let A be a GDSAM. Without loss of generality we assume that no synchronizing state of A is existential (this can be achieved by introducing new universal counterparts of existential synchronizing states and a suitable extension of the $\vdash_A$ relation). Let T_w^A be a full configuration tree of A on an input word $w \in L(A)$. We show that T_w^A contains exactly one computation tree T by building T step by step. At the beginning T contains the root r of T_w^A labelled by a universal configuration, and we must unambiguously take all descendant of r in T_w^A to T. We continue to build T in such a way, that all leaves of the constructed part T' of T create a meaningful cut.

Generally, let after a number of steps a subtree T' be unambiguously constructed, and let Z contain all leaves of T'. If all nodes in Z are labelled by universal configurations we add all descendants of nodes of Z labelled by non-synchronizing configurations to T. If Z contains both existential and universal configurations and also a synchronizing configuration with a synchronizing symbol S, then following (ii) of Definition 2.3, T must contain all direct descendants of all nodes labelled by universal configurations, and following (i) and (ii) of Definition 2.8, the descendant of every node v labelled by an existential configuration must unambiguously be the only direct descendant of v labelled by a synchronizing configuration with the synchronizing symbol S. If cs (Z) contains existential configurations as well as universal ones, and no synchronizing configuration, then we recursively take all direct descendants of nodes of Z labelled by universal configurations until a synchronizing configuration is added to T' (note that this must happen because of (ii) of Definition 2.8). Then, the next step of the construction of T is already described above. Since (ii) of Definition 2.8 holds there exists no meaningful cut Z containing nodes labelled by existential configurations only.

As we have seen there is no possibility of a nondeterministic choice in the construction of T, and so T is given unambigously. □

In what follows we shall consider some complexity measures of SAM's. We shall study the usual parallel complexity measure P as considered for alternating devices [16, 6, 8]. The parallel complexity $P_A(n)$ of a synchronized alternating multitape Turing machine A on inputs of length n is the maximal number of leaves of all accepting computations of A on words of length n. The space complexity $S_A(n)$ of a synchronized alternating multitape Turing machine A on inputs of length n is the maximum number of tape cells visited by the storage head in all accepting computations of A on words of length n.

We use the usual notation $X\text{TIME}(f(n))$ and $X\text{SPACE}(f(n))$ with $X \in \{\text{D}, \text{N}, \text{A}, \text{SA}, \text{DSA}, \text{GDSA}\}$ for deterministic, nondeterministic, alternating, synchronized alternating, and globally deterministic synchronized alternating complexity classes. Also $Y\text{LOG}$ and $Y\text{P}$ denote logarithmic space and polynomial time for

$Y \in \{D, N, A, SA, DSA, GDSA\}$. The machines considered for the definition of the above complexity classes are off-line multitape Turing machines as usual.

We shall consider the off-line multitape Turing machine (TM), finite automata (FA), and multihead finite automata as computation models. To denote R-way k-head X finite automata for $X \in \{D, N, A, SA, DSA, GDSA\}$ and $R \in \{1, 2\}$, we use the notation $RX\text{FA}(k)$. If $k=1$, we write briefly $RX\text{FA}$. For a family of automata from M, $\mathscr{L}(M)$ denotes the family of languages recognized by automata from M. For any computing device B, $S_B(n)$ and $T_B(n)$ denote space and time complexity of B respectively.

3. A Characterization of deterministic complexity classes by globally deterministic synchronization

The nondeterministic complexity classes were characterized by the equality $\text{SASPACE}(S(n)) = \bigcup_{c>0} \text{NSPACE}(nc^{S(n)})$ for any space-constructible function $S(n)$ in [12]. Moreover, Slobodová [2] has shown that the languages in NLOG are recognized exactly by 2SAFA with constant parallel complexity, and Hromkovič et al. [11] have proved that $\text{NSPACE}(n^k) = \mathscr{L}(1\text{SAFA}(k)) = \mathscr{L}(2\text{SAFA}(k))$ for any $k \geqslant 1$. First, we shall establish analogous characterization results for deterministic space complexity classes using globally deterministic synchronized alternation.

We start by showing that $\text{DLOG} = \bigcup_{c \in \mathbb{N}} \mathscr{L}(2\text{GDA}(c)\text{FA})$, where $2\text{GDSA}(c)\text{FA}$ denotes a 2GDSAFA whose all computation trees have at most c leaves (independently of the length of inputs). The proof of this assertion shows how synchronization can be used to exchange information (communicate) between processes of DGSAMs working in parallel.

Theorem 3.1. $\text{DLOG} = \bigcup_{c \in \mathbb{N}} \mathscr{L}(2\text{GDSA}(c)\text{FA})$.

Proof. It is well-known fact that $\text{DLOG} = \bigcup_{c \in \mathbb{N}} \mathscr{L}(2\text{DFA}(c))$ [26]. So, it suffices to prove $\mathscr{L}(2\text{DFA}(c)) \subseteq \mathscr{L}(2\text{GDSA}(c)\text{FA})$ and $\mathscr{L}(2\text{GDSA}(c)\text{FA}) \subseteq \mathscr{L}(2\text{DFA}(2^c))$ for every $c \in N$.

First, we prove $\mathscr{L}(2\text{DFA}(c)) \subseteq \mathscr{L}(2\text{GDSA}(c)\text{FA})$ for every $c \in \mathbb{N}$. Let A be a $2\text{DFA}(k)$ with k read-only heads $H_1, H_2, \ldots, H_k$ for a $k \in N$. Let Σ and Q be the input alphabet and the set of states of A. We shall construct a $2\text{GDSA}(k)\text{FA}$ B with $L(B) = L(A)$. The idea of the simulation of A by B is to use one copy (process) of B to simulate the movement of one head of A, and to use the globally deterministic synchronization to exchange the information about the symbols read (from the input tape) between the parallel processes of B.

Formally, let Σ be the set of synchronizing symbols of B. We set the set of states of B as $Q_B = \{\bar{q}_0\} \cup Q_u \cup Q_c \cup Q_s$, where

(i) $\bar{q}_0 \notin Q$ is a new initial state,

(ii) $Q_u = Q \cup \bigcup_{j=1}^{k} \{q\langle a_1, a_2, \ldots, a_j\rangle \mid q \in Q, a_i \in \Sigma \text{ for } i = 1, \ldots, j\}$ is the set of universal nonsynchronizing states,

(iii) $Q_e = \{q' \mid q \in Q\} \cup \bigcup_{j=1}^{k} \{q'\langle a_1, a_2, \ldots, a_j\rangle \mid q \in Q,\ a_i \in \Sigma \text{ for } i = 1, \ldots, j\}$ is the set of existential nonsynchronizing states, and

(iv) $Q_s = \{(q, a) \mid q \in Q,\ a \in \Sigma\} \cup \bigcup_{j=1}^{k-1} \{q\langle a_1, \ldots, a_j\rangle \mid q \in Q,\ b \in \Sigma,\ a_1 \in \Sigma \text{ for } i = 1, \ldots, j\}$ is the set of synchronizing universal states.

At the beginning of the simulation of the work of A on an input $w = w_1 w_2 \ldots w_n \in \Sigma^n$, B universally splits into k copies $B_1, B_2, \ldots, B_k$, and this is the only one universal branching in the T_w^B. From now on each configuration $(q, i_1, \ldots, i_k)$ of A, where i_j is the position of the head H_j on the input tape ($j = 1, \ldots, k$), is represented by a MCCM $\{(q, i_1), \ldots, (q, i_k)\}$ (i.e. the jth process B_j is on the same position of the input tape as H_j is). Now, it remains to explain how one step $(q_1, r_1, \ldots, r_k) \vdash_A (q_2, s_1, \ldots, s_k)$ of A is simulated by B. Obviously, the arguments of the transition function of A in this step are $q_1, w_{r_1}, \ldots, w_{r_k}$, but each process B_j knows only the part q_1, w_{r_j} of this argument for $j = 1, \ldots, k$. In order to get the complete argument to each process of B, B starts the following procedure.

Step 1: B_1 deterministically enters the synchronizing configuration $((q_1\langle w_{r_1}\rangle, w_{r_1}), r_1)$, i.e. B_1 stores the actual state of A and the symbol read by both H_1 and B_1 in its state $(q_1, \langle w_{r_1}\rangle, w_{r_1}\rangle$, and using synchronization informs other processes about the fact that H_1 reads $w_{r_1} \in \Sigma$. For every $j \in \{2, \ldots, k\}$, B_j first makes a deterministic step $(q_1, r_j) \vdash_B (q_1', r_j)$, where $q_1' \in Q_e$ is the existential counterpart of q_1. Then B_j nondeterministically chooses a successor from $|\Sigma|$ potential candidates of $\{((q_1\langle a\rangle, a), r_j) \mid a \in \Sigma\}$. [Note, that the only possibility to get an accepting computation subtree of T_w^B is to choose $((q_1\langle w_{r_1}\rangle, w_{r_1}), r_j)$ because the synchronizing symbol w_{r_1} is unambiguously prescribed by B_1 reading w_{r_1}.]

Thus after Step 1 all k processes $B_1, \ldots, B_k$ know that H_1 reads w_{r_1}.

Step i (for $i = 2, \ldots, k$): After the $(i-1)$th step B_j is in the universal configuration $((q_1\langle w_{r_1}, \ldots, w_{r_{i-1}}\rangle, w_{r_{i-1}}), r_j)$ for $j = 1, \ldots, k$, i.e. each process stores in its state the symbols $w_{r_1}, \ldots, w_{r_{i-1}}$ read by the heads $H_1, H_2, \ldots, H_{i-1}$ in the configuration $(q_1, r_1, r_2, \ldots, r_k)$ of A.

Now, B_i deterministically enters the synchronizing configuration $((q_1\langle w_{r_1}, \ldots, w_{r_{i-1}}, w_{r_i}\rangle, w_{r_i}), r_i)$ unambiguously given by the symbol w_{r_i} read by B_i. For every $j \in \{1, \ldots, k\} - \{i\}$, B_j first enters the configuration $(q_1'\langle w_{r_i}, \ldots, w_{r_{i-1}}\rangle, r_j)$, and then nondeterministically chooses a successor from $|\Sigma|$ potential candidates of $\{((q_1\langle w_{r_1}, \ldots, w_{r_{i-1}}, a\rangle, a), r_j) \mid a \in \Sigma\}$.

Obviously, after the kth step B_j is in the configuration $((q_1\langle w_{r_1}, \ldots, w_{r_k}\rangle, w_{r_k}), r_j)$. This means that each process knows the complete argument $q_1, w_{r_1}, \ldots, w_{r_k}$ of the transition function of A. Thus, if $\delta(q_1, w_{r_1}, \ldots, w_{r_k}) = (q_2, d_1, \ldots, d_k)$ for some $d_1, \ldots, d_k \in \{-1, 0, 1\}$ determining the movement of the heads ($s_j = r_j + d_j$, for $j = 1, \ldots, k$), then every B_j deterministically enters the configuration $(q_2, r_j + d_j)$ for $j = 1, \ldots, k$. This completes the simulation of one step of A by B.

Now, we shall show that $\mathscr{L}(2\text{GDSA}(c)\text{FA}) \subseteq \mathscr{L}(2\text{DFA}(2^c))$ for every $c \in \mathbb{N}$. The informal idea behind is that each 2DFA(k) A' can be viewed as k independ processes,

each process corresponding to one head of A', with a total information exchange between them via a common finite state control. So, in the following simulation each process of a synchronized automaton is simulated by a nonempty group of heads of the multihead deterministic automaton.

Let B' be a 2GDSA(c)FA. We describe a 2DFA(m) A' with $m=2^c$ and $L(A')=L(B')$. If Q is the set of states of B' and F is the set of final states of B', then the set of states of A' is Q^m and F^m is the set of final states of A'. Let $H_1, H_2, \ldots, H_m$ be the heads of A'. A computation tree of B' is simulated by A' by passing the computation tree from one MCCM to a next MCCM. Each MCCM $\{(q_1, r_1), (q_2, r_2), \ldots, (q_d, r_d)\}$ for some $d \leqslant c$ is stored as a configuration

$$(\underbrace{(q_1, q_1, \ldots, q_1}_{b_1 \text{ times}} \underbrace{q_2, \ldots, q_2}_{b_2 \text{ times}}, \ldots, \underbrace{q_d, \ldots, q_d)}_{b_d \text{ times}}, \underbrace{r_1, \ldots, r_1}_{b_1 \text{ times}}, \ldots, \underbrace{r_d, \ldots, r_d)}_{b_d \text{ times}},$$

where $\sum_{i=1}^{d} b_i = m$ and $b_i \geqslant 2^{c-d+1}$ for every $i \in \{1, \ldots, d\}$. At the beginning of the simulation each element of the initial state of A' is the initial state of B' and all heads follow the movement of the only one process of B'. If a MCCM contains a process in a universal nonsynchronizing configuration, then in one step A' always simulate one step of one of the active processes of B' in a universal nonsynchronizing configuration. If a MCCM contains only synchronizing configurations and existential ones, A' deterministically chooses the only one successor of each existential configuration given by the synchronizing symbol (see Observation 2.9). If MCCM contains only universal synchronizing configurations, A' change all these configurations by their direct descendants in the computation tree.

It remains to explain the way how A' simulates a universal split of one process $\bar{B}$ of B' into k copies $\bar{B}_1, \ldots, \bar{B}_k$ for some $k \in \{2, \ldots, c\}$. If $\bar{B}$ is simulated by the heads $H_i, H_{i+1}, \ldots, H_{i+j}$ for $1 \leqslant i, i+j \leqslant c$ up till now, then $\{H_i, \ldots, H_{i+j}\}$ is partitioned into k disjoint sets $M_1, \ldots, M_k, |M_1| \leqslant \ldots \leqslant |M_k|, |M_k| - |M_1| \leqslant 1$. In what follows the set of heads in M_z with the corresponding $|M_z|$ elements of the state of Q^m simulates the work of the process $\bar{B}_z$ for $z = 1, \ldots, k$. Now, we see why we use a set of heads to simulate one process of B'. We must always have enough heads in each group in order to simulate any possible split of the process. Clearly, $m=2^c$ secures that we never get an empty group of heads by simulating a universal split of a process of B'. □

In what follows we shall need to simulate a globally deterministic synchronized alternating machine A by some deterministic machine B. For this purpose we shall use a slightly modified procedure SIMULATION from [12]. The details (including the organization of the memory) will depend on the particular types of A and B.

Note: To avoid cluttering up the following simulation procedure with inessential special case considerations we shall assume that no existential state is synchronizing. This can be achieved by introducing new states and means, that each synchronizing cut contains universal and accepting configurations only.

Procedure SIMULATION

{cs(Z) will contain consecutive meaningful cut configurations sets of the computation of A}

1. Initialize the meaningful cut configuration set cs(Z) to contain the initial configurations of A on w.
2. **repeat** forever
3. **if** cs(Z) contains a rejecting configuration or two synchronizing configurations with different synchronizing symbols **then** REJECT and HALT **fi**.
4. **if** cs(Z) contains accepting configurations only **then** ACCEPT and HALT **fi**
5. **if** cs(Z) contains accepting and synchronizing configurations with the same synchronizing symbols only **then**
6. **delete** from cs(Z) all accepting configurations
7. **replace** each universal configuration in cs(Z) by all its direct descendants **fi**
8. **if** cs(Z) contains a universal nonsynchronizing configuration C **then** **replace** C by all its direct descendants **fi**
9. **if** cs(Z) contains an existential (non-synchronizing) configuration C **then**
10. **if** cs(Z) contains synchronizing configurations C' **then** **replace** C by the descendant with the same synchronizing symbol as the one occuring in C' (REJECT if it does not exist) **else** skip **fi**
11. **fi**
12. **endrepeat**

Note that due to the conditions (i) and (ii) of Definition 2.8 if accepting computation exists then whenever the condition in line 9 is true the condition in line 10 will eventually be satisfied and C deterministically replaced.

The machine B can check for the existence of an accepting computation of A on w by scanning T_w^A in an essentially breadth-first manner as done in the proof of Observation 2.9. B shall keep track (in cs(Z)) of the successive MCCM's of T_w^A. It actually suffices to use MCCS's if we are interested in acceptance only. (Indeed, if we find a suitable subtree of a node in a meaningful cut labelled by a configuration C we can use this subtree for all nodes labelled by C in this meaningful cut.)

Obviously, if A accepts w, then B (following Procedure SIMULATION) runs via the accepting computation subtree of T_w^A from one synchronizing cut to the next one until a meaningful cut consisting from accepting configurations only is reached. If A does not accept w, B fails in searching for an accepting computation subtree and rejects.

Lemma 3.2. *For any space-constructible function* $S: \mathbb{N} \to \mathbb{N}$

$$\mathrm{GDSASPACE}(S(n)) \subseteq \bigcup_{c>0} \mathrm{DSPACE}(nc^{S(n)}).$$

Proof. To prove this result it is sufficient to simulate an off-line globally deterministic synchronized alternating Turing machine (GDSATM) A with one working tape by an off-line deterministic multitape Turing machine (DTM) B with $S_B(n) \leqslant nd^{S_A(n)}$ for a suitable constant d.

Let w be the input and T_w^A be the full configuration tree of A working on w. B can simulate the work of A on w by using the procedure SIMULATION, keeping track of consecutive MCCS's in cs(Z). It thus suffices to show that B can store each MCCS in space $nd^{S_A(n)}$ and that this space suffices to perform the simulation.

Let k be the number of internal states of A, n, the length of the input word w and m, the cardinality of the working alphabet of A, including the blank. Then the number of distinct configurations of A on input w is at most $n \cdot k \cdot m^{S_A(n)} \cdot S_A(n) \leqslant n \cdot (2km)^{S_A(n)}$.

This is the upper bound on the number of configurations in any MCCS B will ever have to record. B can use one tape for recording a current MCCS cs(Z) and another tape to produce a successive MCCS. A MCCS cs(Z) is stored on the tape of length $r_n = n \cdot (2km)^{S_A(n)}$ in the following way. The ith cell of the tape contains "1" if the ith configuration is in cs(Z) and "0" otherwise. (We assume arbitrary fixed effective order on configurations on which writing down the ith configuration and finding the index of a given configuration requires space at most r_n, e.g., order first by the input head position and then lexicographically by states and tape contents.) Clearly, $\log_2 r_n$ space suffices to write down a configuration of length $S_A(n) + \log_2 n$ and to find its immediate successor configurations in the sense of the $\vdash_A$ relation.

Clearly, B is able to follow the procedure SIMULATION by producing consequently all configurations of A on the third tape in the given order. The space used on the third tape is at most $\log_2 r_n \leqslant r_n$. □

Despite the fact that the definitions suggest that all active parallel processes must take part in each synchronization, we can achieve that in fact we synchronize only specific processes with the rest in effect idling. This "idling technique" for constructing synchronized alternating machines was introduced in [12]. Here, we describe a version of it useful for the simulation of deterministic devices.

The idea of "idling techniques" is based on adding for each state s of a synchronized alternating machine a special state s' called idling counterpart of s.

We introduce the simplest version of "idling", so called "deterministic idling". This version of idling is called deterministic because each process decides deterministically whether it will be active or idling in the synchronizing period. Suppose that we have three processes A, B and C (one may consider a group of processes instead of B or C) and we want A and B to synchronize by some sequence of synchronizing states. Let us assume that both A and B know that they want to synchronize with each other, and the process C knows that it has to be idling because the other processes want to

synchronize. So, all the processes A, B and C deterministically produce a special synchronizing symbol S_B (which denotes the beginning of the synchronizing period), and after that A and B are engaged in the synchronization, and C deterministically enters the idling counterpart of its current state. In this idling state, C keeps on guessing the sequence of synchronizing symbols used by A and B (entering synchronizing states with the given idling state and corresponding synchronizing symbols). When the synchronizing period of A and B is over, A and B deterministically produce a special synchronizing symbol S_E (which denotes the end of the synchronization period). C nondeterministically guesses the synchronizing symbol S_E, leaves its idling state, and enters its "active" counterpart.

Now, we are prepared to present the simulation of deterministic space by globally deterministic synchronization. Note, that the proof of the next lemma includes a new simulation technique essentially differing from the proof of $\mathrm{NSPACE}(nc^{S(n)}) \subseteq \mathrm{SASPACE}(S(n))$ in [12].

Lemma 3.3. $\mathrm{DSPACE}(nc^{S(n)}) \subseteq \mathrm{GDSASPACE}(S(n))$ *for any space-constructible function* $S: N \to N$ *and any positive integer* c.

Proof. Let M be an off-line deterministic TM with one input tape and one working tape having space complexity $S_M(n) \leqslant nc^{S(n)}$, where c is a constant. We shall construct a globally deterministic synchronized alternating multitape Turing machine F having space complexity

$$S_F(n) \leqslant \log(nc^{S(n)}/n)$$

simulating M as follows.

We shall use one parallel process for each working tape square of M (the "signatures" of these processes will change during the computation). Each configuration of M will be represented by a special synchronizing cut of the computation tree of F. One step of M will correspond to a transition (consisting of several steps) from one special synchronizing cut to another special synchronizing cut of the computation tree of F. In each special synchronizing cut there are exactly $nc^{S(n)}+2$ nodes, corresponding to parallel processes A and D_{ij}^k for some $k \in \{-1, 0, 1\}$, $i \in \{1, \ldots, n\}$, and $j \in \{0, \ldots, c^{S(n)-1}\}$. Each process D_{ij}^k is an $S(n)$-space bounded multitape Turing machine that

(a) has the input head on the ith position of the input tape,

(b) stores the number $j \in \{0, 1, \ldots, c^{S(n)}-1\}$ on its working tape,

(c) stores $k \in \{-1, 0, 1\}$ in its finite memory,

(d) stores the current state of the machine M in its finite memory, and

(e) stores the current symbol from the square of the working tape of M which is in the distance $k(c^{S(n)}(i-1)+j+1)$, from the current position of the working head of M (negative distance means to the left).

The process A is a finite automaton that

(f) has its input head on the same position as M, and

(g) stores the current state of M in its finite control.

Since the length of the M's working tape is bounded by $nc^{S(n)}+1$ (including the left endmarker) it is clear that A and appropriate (depending on the position of the working head of M)$nc^{S(n)}+1$ processes D_{ij}^{k} can code unambiguously any configuration of M. During the whole computation of F each of the D_{ij}^{k} processes will store the contents of one fixed square of the working tape. It will only change its own signature by changing the stored values for i, j and k in order to follow the relative position of the square to the position of the working head of M.

Now, let us show the simulation of M by F using exactly $nc^{S(n)}+2$ processes working in parallel. Let $w=a_1 \ldots a_n$ be the input word and $\mathscr{C}=C_0 \vdash C_1 \vdash \ldots C_t$ be the computation of M on the input w. We shall first show in which way F reaches the meaningful cut representing the initial configuration C_0 and then the simulation of the step $C_i \vdash C_{i+1}$ for $i \in \{0, \ldots, t-1\}$.

The GDSATM F first builds up the representation of the working tape of M. It starts by writing $0^{S(n)}$ on its working tape and adjusting both its heads on the left endmarkers of its tapes. Afterwards F splits (universally) into processes A and D_{00}^{0}. Both A and D_{00}^{0} store the initial state of M in their finite controls. Moreover, D_{00}^{0} stores the left endmarker $\not\in$ in its finite control. Next D_{00}^{0} splits into D_{00}^{0} and D_{10}^{1}, and D_{10}^{1} moves its input head one square to the right on the input tape. Now, D_{10}^{1} splits in to D_{10}^{1} and D_{20}^{1}, etc., until we have n processes $D_{10}^{1}, D_{20}^{1}, \ldots, D_{n0}^{1}$, each D_{i0}^{1} having its input heads on the ith square of the input tape. Now, each process D_{i0}^{1} splits successively into $c^{S(n)}$ processes D_{ij}^{1} for $j \in \{0, \ldots, c^{S(n)}-1\}$ in such a way that each D_{ij}^{1} stores j in c-ary notation on its working tape, stores the symbol "blank" (the current symbol on the position $c^{S(n)}(i-1)+j$ of M's working tape) and the initial state of M in its finite control, and positions its input head on the ith square of the input tape. Since the whole process of creating the processes D_{ij}^{1} is fully deterministic, we can assume that all processes (including A and D_{00}^{0}) enter a synchronizing state with a special synchronizing symbol S_b (beginning of the simulating of one step of M). We have thus obtained the synchronizing cut of F coding the initial configuration of M on w.

Now let us assume that F is in a synchronizing cut $\bar{C}_i$ representing the configuration C_i for some $i \in \{0, \ldots, t-1\}$. Let the nodes in $\bar{C}_i$ be labelled by configurations with the synchronizing symbol S_b.

(1) All processes except A and D_{00}^{0} enter deterministically the idling counterparts of their current states.

(2) A enters a state with a synchronizing symbol a, where a is the symbol read by A on the input tape. D_{00}^{0} nondeterministically enters the state with the same synchronizing symbol a. Both A and D_{00}^{0} store the symbol in their finite controls.

(3) D_{00}^{0} deterministically enters a state with a synchronizing symbol d, where d is the symbol read by the M's working head and stored in the finite control of D_{00}^{0}. Process A nondeterministically enters a state with the same synchronizing symbol d. Both A and D_{00}^{0} store the symbol d in their finite controls.

(4) Now both A and D_{00}^{0} know the argument (q, a, d) of the transition function of M, where q is the current state of M in C_i stored by both A and D_{00}^{0}. If $\delta_M(q, a, d)=(p, b, z_1, z_2)$ [p being the new state, b the new tape symbol, and $z_1, z_2 \in \{-1, 0, 1\}$

indicate the input and working head movement resp.] then both A and D_{00}^0 enter deterministically a state with the synchronizing symbol (p, b, z_1, z_2).

(5) All processes become active again and store (p, b, z_1, z_2) in their finite controls. Each process stores p as the current state of M instead of q [cf. (d) and (g)]. D_{00}^0 stores the symbol b instead of d [cf. (e)]. A moves its input head according to z_1 [cf. (f)]. If $z_2 = 0$ the simulation of one step of M is over and all processes enter synchronization states with the special synchronizing symbol S_e (the end of the simulation of one step of M). *If* $z_2 = 1$ $[-1]$ each of the $nc^{S(n)}+1$ processes D_{ij}^k "changes its signature" by storing the number $k(c^{S(n)}(i-1)+j+1)+1$ [resp. $k(c^{S(n)}(i-1)+j+1)-1$] instead of the original $k(c^{S(n)}i-1)+j+1$. Obviously, each process is able to change the representation of i, k and j correctly in a deterministic way. Afterwards all processes enter a state with the synchronizing symbol S_e.

Obviously, the synchronizing cut labelled by S_e codes the next configuration C_{i+1} and the work of F can continue by simulating the next step of M. In case the state p of M is an accepting state of M all processes of F working in parallel enter the accepting state. □

Corollary 3.4. $\mathrm{DSPACE}(nc^{S(n)}) \subseteq \mathrm{1GDSASPACE}(S(n))$ *for any space-constructible function* $S: \mathbb{N} \to \mathbb{N}$, $S(n) \geqslant \log n$, *and any positive integer* c.

Proof. If $S(n) \geqslant \log_2 n$ we have $nc^{S(n)} \leqslant d^{S(n)}$ for suitable d. Thus we do not need to use the head on the input to store the signatures of processes according to the virtual position of the head on the working tape. So, only the process A and the processes D_j^k for $k \in \{-1, 0, 1\}$ and $j \in \{1, \ldots, d^{S(n)}\}$ are used. Since the only left-movement of the head of a D_{ij}^k on the input tape has appeared (in the proof of Lemma 3.3) when the signature of a process has been changed, no D_j^k moves its head on the input tape to the left during the simulation procedure described in the proof of Lemma 3.3. Since $n \cdot c^{S(n)} \geqslant n$ one may assume without loss of generality that the simulated deterministic TM M is on-line, i.e. the head of M on the input tape can move only to the right. Thus, the process A simulating the movement of the input head of M does not need to move to the left during the simulation procedure. □

As a consequence of Lemmas 3.2, 3.3 and Corollary 3.4 we obtain the main result of our paper.

Theorem 3.5. *For any space-constructible function* $S: \mathbb{N} \to \mathbb{N}$

$$\mathrm{GDSASPACE}(S(n)) = \bigcup_{c>0} \mathrm{DSPACE}(nc^{S(n)}),$$

and if $S(n) \geqslant \log n$, *then*

$$\mathrm{1GDSASPACE}(S(n)) = \mathrm{GDSASPACE}(S(n)) = \bigcup_{c>0} \mathrm{DSPACE}(c^{S(n)})$$

In the special case of $S(n)$ constant Theorem 3.1 gives the following characterization of the family of deterministic context-sensitive languages — $\mathscr{L}$(DCS). The last equality has been already proved in [23].

Corollary 3.6. $\mathscr{L}(\mathrm{DCS}) = \mathrm{DSPACE}(n) = \mathscr{L}(\mathrm{2GDSAFA}) = \mathscr{L}(\mathrm{1GDSAFA})$.

So, we see that the famous open problem $\mathscr{L}(\mathrm{DCS}) ? \mathscr{L}(\mathrm{CS})$ can be formulated as a problem whether two-way globally deterministic synchronized alternating finite automata are as powerful as two-way (one-way) synchronized alternating finite automata. (Note that $\mathscr{L}(\mathrm{1SAFA}) = \mathscr{L}(\mathrm{2SAFA}) = \mathrm{NSPACE}(n)$ has been proved in [12] and [11]).

By a little more elaborated technique (used in Lemma 3.3 and in [11]) we are able to prove the following result which transforms $\mathrm{DSPACE}(n^k)$ versus $\mathrm{NSPACE}(n^k)$ to the problem whether synchronized alternation is more powerful than globally deterministic synchronized alternation for k-head finite automata.

Theorem 3.7. $\mathscr{L}(\mathrm{2GDSAFA}(k)) = \mathrm{DSPACE}(n^k)$, *for any* $k \geqslant 1$.

Proof (*sketch*). Let M be 2GDSAFA(k). Using procedure SIMULATION we are able to simulate M by a deterministic Turing machine N with an amount of space that suffices to store any MCCS of a computation of M. N divides its working tape into n^k blocks of the constant length. If a configuration C belongs to an MCCS cs(Z) for a meaningful cut Z that we want to store, N writes the state from C into the ith block of the working tape, where i is the number coded by the heads of M in C. Since the number cannot be greater than n^k, $O(n^k)$ space suffices to do it.

The reverse inclusion can be proved by combining the method from the proof of Lemma 3.3 with an appropriate coding of the content of the working tape by the positions of the heads (see e.g. [11]). Let N be an off-line deterministic Turing machine with one working tape and space complexity not greater than n^k. We construct 2GDSAFA(k) M simulating N. We use one process to tape cell of N (to store its content). Like in the proof of Lemma 3.3, all processes will store also the current state of N. The only difference from the proof mentioned above is the manner in which the latter processes store their "signatures" (which are the distance of the respective tape cells from the current position of the working head of N, hence they are the integers from the interval $[-n^k, n^k]$). Although the processes are in fact multihead deterministic finite automata and have only constant memory, they can use their heads and finite state control to store any number from this interval. □

Corollary 3.8. *For any positive integer* k,

$$\mathscr{L}(\mathrm{2GDSAFA}(k)) \subset \mathscr{L}(\mathrm{2GDSAFA}(k+1)).$$

Corollary 3.9. *For any positive integer* k,

$$\mathscr{L}(\mathrm{1SAFA}(k)) = \mathscr{L}(\mathrm{2SAFA}(k)) \subseteq \mathscr{L}(\mathrm{2GDSAFA}(2k)).$$

Proof. The result follows directly from the results $\mathscr{L}(1SAFA(k)) = \mathscr{L}(2SAFA(k)) = NSPACE(n^k)$ proved in [11] and from the Savitch's Theorem. □

From the above characterization of $DSPACE(n^k)$ and from [12, 11] we get the following characterizations of PSPACE.

Theorem 3.10.

$$PSPACE = GDSALOG = SALOG = SAP = \bigcup_{k \in \mathbb{N}} \mathscr{L}(2GDSAFA(k))$$

$$= \bigcup_{k \in \mathbb{N}} (\mathscr{L}(2SAFA(k)) = \bigcup_{k \in \mathbb{N}} \mathscr{L}(1SAFA(k))$$

Proof. PSPACE = GDSALOG follows from Theorem 3.5. $PSPACE = SAP = SALOG = \bigcup_{k \in \mathbb{N}} \mathscr{L}((2SAFA(k))$ has been proved in [12], and $\mathscr{L}(2SAFA(k)) = \mathscr{L}(1SAFA(k))$ for every $k \in \mathbb{N}$ is established in [11]. The remaining equality $PSPACE = \bigcup_{k \in \mathbb{N}} \mathscr{L}(2GDSAFA(k))$ follows from Theorem 3.7. □

We conclude this section by formulating hierarchy result directly following from Theorem 3.1 and the deterministic space hierarchy [4].

Theorem 3.11. *Let* $S_1, S_2: \mathbb{N} \to \mathbb{N}$ *be some nondecreasing space-constructible functions, and let* $\lim_{n \to \infty} S_1(n)/S_2(n) = 0$. *Then,* $GDSASPACE(S_1(n)) \subset GDSASPACE(S_2(n))$.

4. Comparing the three types of synchronization

In this section we shall consider relations among the three types of synchronization — deterministic, globally deterministic and unrestricted — for various types of underlying machines.

Concerning the relation between deterministic and globally deterministic synchronization we present two results. First using a result of [22] we show that, for $S(n) \geqslant \log n$, the globally deterministic synchronization is substantially more powerful than the deterministic synchronization for $S(n)$-space bounded alternating Turing machines. Let $DASPACE(S(n))$ be the class of languages accepted by $S(n)$-space bounded off-line alternating Turing machines with universal states only (i.e., without nondeterminism).

Theorem 4.1. *For all space-constructible functions* $S: \mathbb{N} \to \mathbb{N}$ *such that* $s(n) \geqslant \log n$,

$$DSASPACE(S(n)) = NSPACE(S(n)).$$

Proof (*idea*). Clearly, $\mathrm{DASPACE}(S(n)) \subseteq \mathrm{DSASPACE}(S(n))$. By a rather technical proof the reverse inclusion can be shown to hold as well [22]. Since $\mathrm{DASPACE}(S(n)) = \mathrm{coNSPACE}(S(n))$ and for, $S(n) \geqslant \log n$, $\mathrm{NSPACE}(S(n)) = \mathrm{coNSPACE}(S(n))$ [15, 25], we have $\mathrm{DSASPACE}(S(n)) = \mathrm{NSPACE}(S(n))$. □

Corollary 4.2. *For all space-constructible functions* $S: \mathbb{N} \to \mathbb{N}$ *such that* $S(n) \geqslant \log n$, $\mathrm{DSASPACE}(S(n)) \subset \mathrm{GDSASPACE}(S(n))$.

Proof. For any $S(n) \geqslant \log_2 n$,

$$\mathrm{DSASPACE}(S(n)) = \mathrm{NSPACE}(S(n)) \subset \bigcup_{c>0} \mathrm{NSPACE}(c^{S(n)})$$

$$= \bigcup_{c>0} \mathrm{DSPACE}(c^{S(n)}) = \mathrm{GDSASPACE}(S(n)). \qquad \square$$

The second result comparing the deterministic and the globally deterministic synchronization holds for the case of one-way alternating finite automata with constant number of parallel processes.

Theorem 4.3. $\mathscr{L}(1\mathrm{GDSA}(2)\mathrm{FA}) - \bigcup_{k \in \mathbb{N}} \mathscr{L}(1\mathrm{DSA}(k)\mathrm{FA}) \neq \emptyset$.

Proof. Let us consider the language $L = \{u2v \mid u, v \in \{0,1\}^+, u \neq v\}$. One can easily find a 1GDSA(2)FA recognizing L. Slobodová [22] has shown that L cannot be recognized by on-line DSATM with sublinear space complexity which directly implies $L \notin \bigcup_{k \in \mathbb{N}} \mathscr{L}(1\mathrm{DSA}(k)\mathrm{FA})$. □

Concerning the relation between the globally deterministic and the unrestricted synchronization for alternating space-bounded Turing machines we have seen that they are equally powerful for space bounds $S(n) \geqslant \log n$. We do not know the relation for space bounds below $\log n$. In fact, for $S(n) =$ constant this problem is equivalent to the well-known first lba problem.

Acknowledgment

We would like to thank the unknown referees for their comments and suggestions which improved the presentation of this paper.

References

[1] A.K. Chandra, D.K. Kozen and Stockmeyer, Alternation, *J. ACM* **28** (1) (1981) 114–133.

[2] J. Dassow, J. Hromkovič, J. Karhumäki, B. Rovan and A. Slobodová, On the power of synchronization in parallel computations, in: *Proc. 14th MFCS '89*, Lecture Notes in Computer Science, Vol. 379 (Springer, Berlin 1989) 196–206.

[3] D. Geidmanis, On possibilities of one-way synchronized alternating and alternating finite automata, in: *Proc. 15th MFCS '90*, Lecture Notes in Computer Science, Vol. 459 (Springer, Berlin, 1990) 292–299.

[4] J. Hartmanis, P.M. Lewis II and R.E. Stearns, Hiearchies of memory limited computations, in: *Proc. 6th Ann. IEEE Symp. on Switching Circuit Theory and Logical Design* (1965) 179–190.

[5] J. Hromkovič, One-way multihead deterministic finite automata, *Acta Inform.* **19** (1983) 377–384.

[6] J. Hromkovič, On the power of alternation in automata theory, *J. Comput. Sci.* **31** (1985) 28–39.

[7] J. Hromkovič, How to organize the communication among parallel processes in alternating computations, Unpublished manuscript, Comenius University, 1986.

[8] J. Hromkovič, Tradeoffs for language recognition on alternating machines, *Theoret. Comput. Sci.* **63** (1989) 203–221.

[9] J. Hromkovič and K. Inoue, A note on realtime one-way synchronized alternating one-counter automata, *Theoret. Comput. Sci.* **108** (1993) 393–400.

[10] J. Hromkovič, K. Inoue, A. Ito and I. Takanami, On the power of two-dimensional synchronized alternating finite automata, *Fund. Inform.* **15** (1991) 90–98.

[11] J. Hromkovič, K. Inoue, B. Rovan, A. Slobodová, I. Takanami and K. Wagner, On the power of one-way synchronized alternating machines with small space, *Internat. Found. Comput. Sci.* **3** (1992) 65–79.

[12] J. Hromkovič, J. Karhumäki, B. Rovan and A. Slobodová, On the power of synchronization in parallel computations, *Discrete Appl. Math.* **32** (1991) 155–182.

[13] O.H. Ibarra and N.Q. Tran, On space-bounded synchronized alternating Turing machines, in: *Proc. 8th FCT '91*, Lecture Notes in Computer Science Vol. 529 (Springer, Berlin, 1991) 248–257.

[14] O.H. Ibarra and N.Q. Tran, New results concerning synchronized finite automata, in: *Proc. 19th ICALP '92*, Lecture Notes in Computer Science, Vol. 623 (Springer, Berlin, 1992) 126–137.

[15] N. Immerman, Nondeterministic space is closed under complementation, SIAM *J. Comput.* **17** (5) (1988) 935–938.

[16] K.N. King, Alternating finite automata, Ph.D. Dissertation, University of California, Berkeley, 1981.

[17] K.N. King, Alternating multihead finite automata, in: *Proc. 8th ICALP '81*, Lecture Notes in Computer Science, Vol. 115 (Springer Berlin 1981) 506–520.

[18] A.L. Rosenberg, On multihead finite automata, *IBM J. Res. Develop.* **25** (1966) 388–394.

[19] R.L. Rivest and A.C. Yao, $k+1$ heads are better than k, *J. ACM* **25** (1978) 337–340.

[20] A. Slobodová, On the power of communication in alternating computations, Student Research Papers Competition, Section Computer Science, Comenius University, Bratislava, April 1987 (in Slovak).

[21] A. Slobodová, On the power of communication in alternating machines, in: *Proc. 13th MFCS'88*, Lecture Notes in Computer Science, Vol. 324 (Springer, Berlin, 1988) 518–528; extended version: Communication for alternating machines, *Acta Inform.* **29** (1992) 425–441.

[22] A. Slobodová, Some properties of space-bounded synchronized alternating Turing machines with only universal states, in: *Proc. 5th IMYCS '88*, Hungarian Academy of Sciences, Budapest (1988) 209–218; (Extended version: *Theoret. Comput. Sci.* **96** (1992) 411–419).

[23] A. Slobodová, One-way globally deterministic synchronized alternating finite automata recognize exactly deterministic context-sensitive languages, *Inform. Process. Lett.* **36** (1990) 69–72.

[24] A. Slobodová, Some properties of space-bounded synchronized alternating Turing machines with universal states only, *Theoret. Comput. Sci.* **96** (1992) 411–419.

[25] R. Szelepcsényi, The method of forced enumeration for nondeterministic automata, *Acta Inform.* **26** (1988) 279–284.

[26] K. Wagner and G. Wechsung, *Computational Complexity*, Mathematische Monographien, Vol. 19 (Deutscher Verlag der Wissenschaften, Berlin, 1986).

[27] J. Wiedermann, On the power of synchronization, *J. Inform. Process. Cybern. EIK* **25** (10) (1989) 499–506.