On the power of synchronization in parallel computations*

Juraj Hromkovič

Department of Theoretical Cybernetics and Institute of Computer Science, Comenius University, 842 15 Bratislava, Czechoslovakia

Juhani Karhumäki

Department of Mathematics, University of Turku, 20500 Turku 50, Finland

Branislav Rovan and Anna Slobodová

Department of Theoretical Cybernetics and Institute of Computer Science, Comenius University, 842 15 Bratislava, Czechoslovakia

Received 15 June 1988
Revised 15 September 1989

Abstract


This paper continues investigations of synchronized alternating machines which provide a natural model of communication of parallel processes. It is proved that synchronized alternating space

$$\text{SASPACE}(S(n)) = \bigcup_{c \in C} \text{NSPACE}(n \cdot c^{S(n)})$$

Using this characterization of synchronized alternating space the following new characterizations of fundamental complexity classes are established:

1. $$\bigcup_{c \in C} \text{DSPACE}(c^{S(n)}) = \bigcup_{c \in C} \text{ATIME}(c^{S(n)}) = \bigcup_{c \in C} \text{SATIME}(c^{S(n)}) = \text{SASPACE}(S(n))$$ for $$S(n) > \log_2 n$$.

2. $$\text{NSPACE}(n) = \mathcal{L}(2 \text{SAFA})$$, i.e. two-way synchronized alternating finite automata recognize exactly context-sensitive languages.

3. $$\text{PSPACE} = \text{SALOGSPACE}$$ is exactly the class of languages recognized by two-way synchronized alternating multihead finite automata.

Further, the parallel complexity of synchronized alternating finite automata is investigated and some hierarchy results are established. We also investigate the decidability problems for multihead and multitape automata from the new point of view. Instead of having one common finite state control enabling "full communication" of the heads as usual we consider $$k$$ independent finite automata communicating only by synchronization. This represents a natural intermediate case between "full communication" and "no communication" and we present several new results and open problems for this form of communication of parallel processes.

* This research was supported in part by the grant ŠPZV I-1.5/8.
1. Introduction

A number of models has been introduced in order to study parallelism [4]. The model used in this paper is based on that of an alternating machine [3]. An alternating machine can be viewed as a number of "ordinary" machines working independently (in parallel) on the same input. Each of these machines may split into several copies of itself (distinguished by the current internal state). Though useful for studying some properties this model does not capture the fact that in practical parallel computations parallel processes often communicate with each other. In order to study the influence of the limited form of communication on the computational power and complexity of language recognition the notion of synchronized alternating machine was introduced in [6] and investigated in [14–16]. The results obtained show that synchronization increases the power of underlying machines. Besides, a new characterization of the class NLOGSPACE by simple devices was obtained. A special property of synchronization [6,14] enabling to synchronize nondeterministic decisions was investigated in [18], where it is shown that the alternating Turing machines with this type of synchronization are in the Second Machine Class [4].

In the present paper we continue the study of synchronized alternating machines. In Section 3 we characterize the power of synchronization by nondeterministic and deterministic space classes. We show that synchronized alternating space \( \text{SASPACE}(S(n)) \) is equal to \( \bigcup_{c>0} \text{NSPACE}(nc^{S(n)}) \) for any space constructible function \( S: \mathbb{N} \rightarrow \mathbb{N} \). In particular we obtain a new characterization of context-sensitive languages (\( \text{NSPACE}(n) \)) by two-way synchronized alternating finite automata and the equality

\[
\text{SASPACE}(S(n)) = \bigcup_{c>0} \text{DSPACF}(c^{S(n)}), \quad \text{for } S(n) > \log_2 n.
\]

Further, we give a new characterization of PSPACE by \( \text{SALOGSPACE} \) (synchronized alternating logspace) which is characterized by synchronized alternating multihead finite automata. This extends the well-known characterization of the fundamental complexity classes \( \text{DLOGSPACE} \subseteq \text{NLOGSPACE} \subseteq \text{P} \) by deterministic, nondeterministic and alternating multihead automata, respectively. We also show

\[
\text{SASPACE}^F(S(n)) = \bigcup_{c>0} \text{SATIME}(c^{S(n)}).
\]

This means that our machine model seems to be the first one that uses space in an optimal way. In case alternating machines would have this property it would follow that \( \text{P} = \text{NP} = \text{PSPACE} \). In case nondeterministic machines would have this property then \( \text{NLOGSPACE} = \text{P} = \text{NP} \), and in case deterministic machines would have this property, then \( \text{DLOGSPACE} = \text{NLOGSPACE} = \text{P} \).
In Section 4 we turn our attention to the simplest synchronized machine, the synchronized alternating finite automaton. While nondeterminism and alternation do not help to increase the power of two-way finite automata [3] the family of languages recognized by two-way synchronized alternating finite automata is equal to NSPACE(n) as follows from our results mentioned above. Slobodová [14] has proved that two-way synchronized alternating finite automata with a bounded number of universal branchings recognize exactly languages in NLOGSPACE.

We investigate here the influence of limiting the number of parallel processes on the computational power of synchronized alternating finite automata, i.e. we study the parallel complexity classes. First we show that limiting the parallelism to a constant k reduces the power to that of k-head nondeterministic finite automata. Besides supporting the common view that the number of heads is a reasonable measure of parallelism, this result enables us to translate the hierarchy results to synchronized automata. Next we give an upper bound NSPACE(f(n)log₂ n) for the power of synchronized alternating finite automata with parallelism limited by f(n). We conclude this section by developing a lower bound proof technique that enables us to obtain hierarchy results for parallel complexity classes when \( f(n) \leq n^{1/4}/\log_2 n \).

In Section 5 we consider decidability questions and find a surprising connection between synchronization and transducers. First we prove there is a little hope to have interesting decidability properties for synchronized machines since the emptiness problem is undecidable already for one-way synchronized alternating finite automata without existential states (i.e., without nondeterminism) and with parallelism two. We can prove however that in case of restricting the synchronization alphabet to one symbol only the equivalence problem for one-way synchronized alternating automata without existential states and with constant parallelism is decidable. We then turn our attention to recognition of relations (i.e., tuples of words) instead of languages. The finite automata with k input tapes were studied in \([5,2]\). Instead of having one common finite state control enabling "full communication" among the heads we shall consider k independent finite automata (one for each input tape) communicating by synchronization. This presents a natural intermediary case between "full communication" and "no communication" for which some problems undecidable for the full communication case may be decidable. It also presents a new way of attacking the known open decidability problems for the full communication case (like the equivalence problem for deterministic multitape automata \([5,2]\)). Indeed we can show equivalence to be decidable for one-way "deterministic" synchronized finite automata on k tapes and undecidable in the nondeterministic case. In doing so we reveal a close relation between synchronization and finite transducers. This suggests another possibility of defining acceptance for synchronized machines and a number of open problems. Finally we prove some hierarchy results by comparing the "full communication" case with different types of synchronizations.
2. Basic notions

We refer to [3, 7] for a more formal introduction of the alternation and stress here only the notions important for following arguments. Given any machine type $M$ we shall augment it by a finite synchronization alphabet. An internal state of such an augmented (synchronized) machine can be either an internal state of $M$ or a pair (internal state of $M$, synchronizing symbol). The latter is called a synchronizing state. As usual for alternating machines we consider the states of $M$ partitioned into universal, existential, accepting and rejecting states. We use the usual notion of a configuration and the computation step relation $\rightarrow_A$ for the machine and call the configuration universal, existential, or synchronizing in correspondence to the type of state. Initial and accepting configurations are defined as usual for the particular type of the machine. To avoid misunderstandings we give a precise definition of accepting computation of a synchronized alternating machine. It is a suitable subtree of the full configuration tree.

**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 $t_0$, $c(t_0)$ is an initial configuration of $A$ on $w$;
(iii) $t_2$ is a direct descendant of $t_1$ iff $c(t_1) \not\rightarrow_A c(t_2)$.

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 (sometimes called "processes" in what follows), working in parallel and splitting in universal configurations. An informal description of the use of synchronization is the following. 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 synchronizing states. We shall make this more precise now.

**Definition 2.2.** The synchronizing sequence of a node $t$ in a full configuration tree $T$ with the root $t_0$ is the sequence of synchronizing symbols occurring in labels of the nodes on the path from $t_0$ to $t$.

**Definition 2.3.** A computation tree of a SAM $A$ on an input word $w$ is a (possibly infinite) subtree $T'$ of the full configuration tree $T$ of $A$ on $w$ such that

(i) each node in $T'$ labelled by a universal configuration has the same direct descendants as in $T$;
(ii) each node in $T'$ labelled by an existential configuration has at most one direct descendant;
(iii) 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 with deterministic transition function the full configuration tree satisfies (i) and (ii) of the above definition. If it happens to satisfy (iii) as well it is a 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 case of alternating machines, in 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.** The *synchronizing depth* of a node \( t \) of a full configuration tree \( T \) is the number of synchronizing configurations on the path from the root to \( t \) (excluding the configuration which is the label of \( t \)).

**Definition 2.6.** A *meaningful cut* (MC) of a computation tree \( T \) is a set \( Z \) of nodes in \( T \) having the same synchronizing depth \( d \) such that every infinite path from the root and every path from the root to a leaf node with synchronization depth greater than \( d \) contains exactly one node from \( Z \).

**Definition 2.7.** A *synchronization cut* of a computation tree \( T \) is a meaningful cut containing nodes labelled by synchronizing configurations only.

**Notation 2.8.** For a given MC \( Z \) 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 the nodes in \( Z \)—the *meaningful cut configurations set* (MCCS).

In Sections 3 and 4 we shall deal with complexity measures. The space complexity measure \( S \) is considered as for alternating machines [3]. The parallel complexity \( P_A(n) \) of a SAM \( A \) on inputs of length \( n \) is the maximal number of leaves of all accepting computations of \( A \) on words of length \( n \).

We use the usual notation \( XTIME(f(n)) \) and \( XSPACE(f(n)) \) with \( X \in \{D,N,A\} \) for deterministic, nondeterministic and alternating complexity classes. For syn-
chronized alternation we let \( X = \text{SA} \). Also \( \text{SA} \text{LOGSPACE} \) and \( \text{SA} \text{P} \) denote synchronized alternating logspace and synchronized alternating polynomial time respectively. To denote \( R \)-way \( k \)-head \( X \) finite automata, for \( X \in \{ \text{D}, \text{N}, \text{A}, \text{SA} \} \) and \( R \in \{1,2\} \) we use the notation \( RXFA(k) \). If \( k = 1 \) we write briefly \( RXFA \). For a family of automata \( M \), \( \mathcal{F}(M) \) 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. Characterization of the power of synchronized alternation in terms of nondeterministic and deterministic space classes

In this section we show that synchronized alternation is very powerful because its simulation by nondeterministic machines requires an exponential increase in space complexity. We also show that an exponential increase in space complexity suffices for a deterministic simulation. This enables us to view the hierarchy of the fundamental complexity classes from a new perspective. Further we give new characterizations of context-sensitive languages and \( \text{PSPACE} \) by two-way synchronized alternating finite automata and two-way synchronized multihead finite automata respectively.

In what follows we shall repeatedly need to simulate a synchronized alternating machine \( A \) by some nondeterministic machine \( B \). We shall give here a general procedure for such simulation. (The details including the organization of the memory will then depend on the particular type of \( A \) and \( B \).)

**Procedure SIMULATION**

1. Initialize \( Z \) to contain the initial configuration of \( A \) on \( w \).
2. **Repeat** forever
3. **if** \( Z \) contains a rejecting configuration or two synchronizing configurations with different synchronizing symbols
   - **then** REJECT and HALT
4. **if** \( Z \) contains accepting configurations only
   - **then** ACCEPT and HALT
5. **if** \( Z \) contains accepting and synchronizing configurations with the same synchronizing symbols only
   - **then**
     - delete from \( Z \) all accepting configurations
   - replace each universal configuration in \( Z \) by all its direct descendants
6. replace each existential configuration in \( Z \) nondeterministically by exactly one descendant.
Synchronization in parallel computations

9 \text{fi.}
10 \text{if } Z \text{ contains a nonaccepting nonsynchronizing configuration } C
11 \quad \text{then}
12 \quad \text{if } C \text{ is universal}
13 \quad \quad \text{replace } C \text{ by all its direct descendants}
14 \quad \text{fi}
15 \text{if } C \text{ is existential}
16 \quad \text{replace } C \text{ nondeterministically by exactly one direct descendant}
17 \text{fi}
18 \text{endrepeat}

The machine \( B \) can check for the existence of an accepting computation of \( A \) on \( w \) by scanning \( T^A_w \) in an essentially breadth-first manner. \( B \) shall keep track (in \( Z \)) of the successive MCCM's of \( T^A_w \). It actually suffices to use MCCS's if we are interested in acceptance only. (Indeed, if we find a suitable subtree for a node in MC labelled by a configuration \( C \) we can use this subtree for all nodes in MC labelled by \( C \).)

In case \( A \) accepts \( w \), \( B \) can make correct choices in lines 8 and 12 so that it will find the accepting computation tree of \( A \) on \( w \) with \( Z \) eventually containing accepting configurations only and accepting. Vice versa, if \( B \) ever accepts, then choices in lines 8 and 12 indicate how to pick up an accepting computation from \( T^A_w \).

\textbf{Lemma 3.1.} \( \text{SASPACE}(S(n)) \subseteq \bigcup_{c>0} \text{NSPACE}(n \cdot c^{S(n)}) \) for any space-construc-
tible function \( S: \mathbb{N} \to \mathbb{N} \).

\textbf{Proof.} To prove this result it is sufficient to simulate an off-line synchronized alternating Turing machine (SATM) \( A \) with one working tape by an off-line nondeterministic multitape Turing machine (NTM) \( B \) with \( S_B(n) \leq n \cdot d^{S_A(n)} \) for a suitable constant \( d \).

Let \( w \) be an input and \( T^A_w \) be the full configuration tree of \( A \) working on \( w \). \( B \) can simulate the work of \( A \) on \( w \) by using Procedure SIMULATION, keeping track of consecutive MCCS's in \( Z \). It thus suffices to show that \( B \) can store each MCCS in space \( n \cdot d^{S_A(n)} \) 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 \( n \cdot k \cdot m^{S_A(n)} \cdot S_A(n) \leq n \cdot (2km)^{S_A(n)} \).

This is then 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 \( Z \) and another tape to produce a successive MCCS. A MCCS \( Z \) is stored on the tape of
the length \( r_n = n \cdot (2km)^{S_3(n)} \) in the following way. The \( i \)th cell of the tape contains "1" if the \( i \)th configuration is in \( Z \) and "0" otherwise. (We assume arbitrary fixed effective order on configurations on which writing down the \( i \)th 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 the 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 a sense of the \( \sqsubseteq \) relation.

Clearly \( B \) is able to follow Procedure SIMULATION by producing all configurations of \( A \) on the third tape in the given order. The space used on the third tape is at most \( \log_* r_n \). □

In order to explain the proof of the reverse inclusion in a readable form, we first have to explain some techniques used for construction of synchronized alternating devices. Despite the fact that the definition suggests that synchronization is uniform, i.e., all parallel processes must take part, we can achieve that in fact we synchronize only specific processes with the rest in effect idling. It can be achieved as follows. For some internal states we shall have their idling counterpart. Suppose we have three processes \( A, B, \) and \( C \) and we want \( A \) and \( B \) to synchronize by some sequence of synchronizing states. While \( A \) and \( B \) engage in the synchronization, \( C \) enters nondeterministically 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 symbol). When the synchronization period of \( A \) and \( B \) is over \( C \) nondeterministically leaves its idling state and enters its "active" counterpart. Note that when actually using this technique, the beginning and the end of the synchronization of \( A \) and \( B \) will have to be clearly marked by some synchronizing cuts.

Next we show that it is possible to check for synchronized alternating devices whether two parallel processes \( A \) and \( B \) scan the same position on the input tape. To do so, both \( A \) and \( B \) split off one copy of itself, say \( A' \) and \( B' \), in special "checking position" states. After that all processes including \( A, B, A' \) and \( B' \) enter a special synchronizing state and all parallel processes besides \( A' \) and \( B' \) nondeterministically enter the idling part of their current state. Now both \( A' \) and \( B' \) start moving right synchronizing with each other at each step. They finish by synchronizing at the right end of the tape and entering accepting states.

Now we are ready to prove the following.

**Lemma 3.2.** \( \text{NSPACE}(n \cdot c^{S(n)}) \subseteq \text{SASPACE}(S(n)) \) for any \( c \in N \) and any space-constructible function \( S : N \to N \).

**Proof.** Let \( M \) be an off-line nondeterministic Turing machine with one working tape and with space complexity \( S_M(n) = n \cdot c^{S(n)} \). Without loss of generality we can
assume $c$ is a power of two. We shall construct a synchronized alternating multitape Turing machine $F$ with space complexity $S_F(n) = \log_2(S_M(n)/n) \leq S(n)$ simulating $M$ as follows.

Given an input word $a_1 \ldots a_n$, $F$ universally splits itself into $n+4$ parallel processes $A, B, D_0, D_1, \ldots, D_{n+1}$ in such a way that the head of $D_i$ is on the $i$th position of the input tape and both $A$ and $B$ have their input heads on the first position. Since $S$ is a space constructible function we may assume that each of these $n+4$ parallel processes has the word $0^{S(n)}$ on its first working tape. Clearly, this whole starting situation can be achieved in one sweep of $F$.

Each $D_i$ using the contents $0^{S(n)}$ of its first working tape splits itself into $c^{S(n)}$ parallel processes $D_i^1, D_i^2, \ldots, D_i^{c^{S(n)}}$. During the computation of $F$ each process $D_i^j$ for $i \in \{0, 1, \ldots, n+1\}, j \in \{1, 2, \ldots, c^{S(n)}\}$ will have its input head stationary on the $i$th position of the input tape. We assume that the cardinality of the working alphabet of $F$ is large enough to store the number $c^{S(n)}$ in space $S(n)$. So, we can organize the splitting of $D_i$ in such a way that each $D_i^j$ remembers the number $j$ on the second working tape ($D_i^j$ splits itself into $D_i^{j0}, D_i^{j0+1}, \ldots, D_i^{j2c^{S(n)}}$, where $D_i^j$ will recursively split itself into $D_i^{j0}, \ldots, D_i^{j2c^{S(n)}}$, and $D_i^{j2c^{S(n)}}$ will split itself into $D_i^{j2c^{S(n)}}$, $\ldots, D_i^{j2c^{S(n)}}$).

To be ready to describe the simulation of one step of $M$ by $F$ we still have to show how a configuration $C$ of $M$ is represented by a MC $Z$ of $F$. The process $A$ has its input head at the same position as $M$ has and $A$ also stores the state of $M$ in its state. Let the head of $M$ be on the $d$th position of the working tape. Since $d \leq n \cdot c^{S(n)}$ we can write $d = (b-1) \cdot c^{S(n)} + m$ for $b \in \{1, \ldots, n+1\}, m \in \{0, \ldots, c^{S(n)}-1\}$. To store this information the process $B$ is on the $b$th input position and has the number $m$ on the second working tape. The contents of the first $n \cdot c^{S(n)}$ positions of the working tape of $M$ (including blanks) is stored by $D_i^j$ in such a way that $D_i^j$ stores the symbol on the $(c^{S(n)} \cdot (i-1)+j)$th position.

Now, let us describe the simulation of one step of $M$ by $F$.

(1) The simulation of one step of $M$ starts by synchronizing all processes with a special synchronizing symbol $S_0$. First, one of the $D_i^j$ must find out (for sure) that it is representing the working tape square currently scanned by $M$. To do so, $B$ and each of the $D_i^j$ split off "checking position" processes $'B'$ and $'D_i^j'$ or $"D_i^j"$. The process $'D_i^j'$ will succeed if $D_i^j$ stores the same tape position as $B$, the $"D_i^j"$ will succeed if the positions are different. Now each $D_i^j$ stores in its state whether it splitted off a primed or double primed process. Now the processes $'B'$, $'D_i^j'$ and $"D_i^j"$ will check their positions by count down synchronizing with each other at each step. All the other processes are idling. This phase will end by a special synchronizing symbol $S_1$. Clearly, if none or more than one $D_i^j$ split off the primed version, the checking process will fail. If there is exactly one primed version $'D_i^j'$ but not from the "correct" $D_i^j$, the checking process will also fail. Thus at the $S_1$ synchronizing cut there is exactly one $D_i^j$ that "knows" it is representing the active working tape square. Let it be $D_i^m$.
(2) All $D^f_i$ except $D^m_0$ are now supposed to enter their idling states. Then all $A$, $B$ and $D^m_0$ synchronize themselves by synchronizing symbol $(q, a, h)$, where $q$ is the state of $M$, $a$ and $h$ are the currently scanned symbols on the input and on the working tape respectively. Clearly, $A$ knows $q$ and $a$, and it nondeterministically guesses $h$, $D^m_0$ knows $h$, and it nondeterministically guesses $q$ and $a$, and $B$ guesses all three $q$, $a$ and $h$. Now $A$, $B$ and $D^m_0$ store $(q, a, h)$ in their states in order to know the whole information used by $M$ in making one step and start to simulate it. Since $M$ is a nondeterministic device there is a finite set $M(q, a, h)$ of possible activities of $M$ in state $q$ by reading $a$ and $h$. Clearly, we have to arrange that all processes $A$, $B$ and $D^m_0$ will consider the same nondeterministic decision of $M$. Let $(p, h', z_1, z_2) \in M(q, a, h)$, where $p$ is the new state of $M$, $h'$ is the symbol written on the $d$th position and $z_1, z_2 \in \{-1, 0, 1\}$ indicate the movement of the head on the input and working tape respectively. $A$, $B$ and $D^m_0$ guess nondeterministically $(p, h', z_1, z_2)$ and use $(p, h', z_1, z_2)$ as synchronizing symbol.

Now we are sure that $A$, $B$ and $D^m_0$ have chosen the same action and we can start to change the representation of the original configuration of $M$. $A$ stores the new state $p$ in its state and moves the head according to $z_1$. $D^m_0$ stores the symbol $h'$ in its state. If $z_2 = 0$, $B$ does nothing. If $z_1 = 1$ (respectively $-1$), then $B$ tries to add (subtract) 1 to (from) the contents of its second working tape. If it is possible $B$ does it and stops. If it is impossible because the stored number on the second tape is $c^S(n)$ (respectively $1$), then $B$ moves its head on the input tape one square to the right (left) and writes $1$ ($c^S(n)$) on its second working tape.

(3) Now all processes produce a special synchronizing symbol $S_2$. Following this synchronization cut all idling processes become active again. In case state $p$ stored by $A$ is a final state of $M$, $A$ produces a special synchronizing symbol $S_3$ and stops in a final state. All other processes guess nondeterministically symbol $S_3$ and stop in a final state. If $p$ is not a final state all processes synchronize themselves by the synchronizing symbol $S_0$ and start to simulate the next step of $M$ as in 1.

To check the correctness of the above construction it suffices to check that in (1) exactly one $D^f_i$ decides to stay active. It is crucial that the $D^f_i$ can enter idling states only immediately following the $S_0$ synchronization. Thus $B$ can synchronize itself with at most one of the $D^f_i$ (the others cannot make up either for the "wrong" head position or for the "wrong" contents of the second working tape by joining the checking process later).

Applying Lemmas 3.1 and 3.2 we obtain the following result.

**Theorem 3.3.** $\bigcup_{c > 0} \text{NSPACE}(n \cdot c^{S(n)}) = \text{SASPACE}(S(n))$ for any space-constructible function $S : \mathbb{N} \to \mathbb{N}$.

The characterization of synchronized alternating space by nondeterministic space
provides several new interesting consequences. Symbol $\subset$ is used for the proper inclusion in what follows.

**Corollary 3.4.** Let $S_1, S_2 : \mathbb{N} \to \mathbb{N}$ be such nondecreasing functions that $S_1(n) \geq S_2(n)$ for all $n \in \mathbb{N}$ and $\lim_{n \to \infty} S_1(n)/S_2(n) = \infty$. Then $SASPACE(S_2(n)) \subset SASPACE(S_1(n))$.

**Corollary 3.5.** For any space-constructible function $S(n) \geq \log_2 n$

$$SASPACE(S(n)) = \bigcup_{c > 0} DSATIME(c^{S(n)}) = \bigcup_{c > 0} ATIME(c^{S(n)}) = \bigcup_{c > 0} SATIME(c^{S(n)})$$

**Proof.** The first equality follows from Savitch’s Theorem because $NSPACE(n \cdot c^{S(n)}) \subset DSATIME(c^{2S(n) + \log n}) \subset DSATIME(c^{2S(n)})$, for $S(n) \geq \log_2 n$. Slobodova proved $SATIME(f(n)) \subset ATIME(2f(n))$ in [15] hence the last equality. The second equality has been proved in [3].

Clearly, the equality

$$SASPACE(S(n)) = \bigcup_{c > 0} SATIME(c^{S(n)})$$

implies that synchronized alternating machines are able to use the space in an “optimal way”. It seems that deterministic, nondeterministic and alternating computing devices do not have this property because if they would have this property then some fundamental complexity hierarchies collapses as mentioned already in the Introduction.

We note that Wiedermann started to investigate a new type of synchronization [18] that enables to synchronize nondeterministic decisions only, and he proved in another way for his synchronization a result similar to our Corollary 3.5. Wiedermann’s result seems to show that the crucial power of our model of synchronized computations comes from the synchronization of nondeterministic decisions (although from the point of view of our model we have considered the synchronization of nondeterministic decisions only as one of special techniques of synchronized alternating computations).

Now considering $S$ as a constant function in Theorem 3.3 we obtain a new characterization of the family $L_{CS}$ of context-sensitive languages.

**Corollary 3.6.** $L(2SAFA) = L_{CS} = NSPACE(n)$.

The most interesting fact on this result is that neither nondeterminism nor alternation were able to increase the computational power of finite automata [3] but synchronized alternation brings so much computational power that finite automata jump two steps in Chomsky hierarchy.
The last result presented in this section provides new characterization of PSPACE.

**Theorem 3.7.** \( \text{PSPACE} = \text{AP} = \text{SALOGSPACE} = \bigcup_{c>0} \mathcal{L}(2\text{SAFA}(k)) \).

**Proof.** \( \text{PSPACE} = \text{SALOGSPACE} \) follows directly from Theorem 3.3. Corollary 3.5 implies \( \text{AP} = \text{SALOGSPACE} \). Thus it suffices to show that

\[
\text{PSPACE} = \bigcup_{k \in \mathbb{N}} \mathcal{L}(2\text{SAFA}(k)).
\]

To prove \( \mathcal{L}(2\text{SAFA}(k)) \subseteq \text{NPSPACE} = \text{PSPACE} \), for any \( k \in \mathbb{N} \), one can use Procedure SIMULATION in the same way as in Lemma 3.1. Since the number of distinct configurations of any \( k \)-head finite automaton can be bounded by \( d \cdot n^d \), for a suitable constant \( d \), and each configuration of a \( k \)-head finite automaton can be stored in space \( O(\log_2 n) \) it is clear that \( \mathcal{L}(2\text{SAFA}(k)) \subseteq \text{NSPACE}(n^d \log_2 n) \subseteq \text{PSPACE} \).

To prove \( \text{PSPACE} \subseteq \bigcup_{k \in \mathbb{N}} \mathcal{L}(2\text{SAFA}(k)) \) let us consider an off-line deterministic Turing machine \( M \) working in space \( d \cdot n^d \), for a suitable constant \( d \). Without loss of generality we may assume that \( M \) has only one working tape. We construct a 2SAFA\((3d + 4)\) automaton \( A \) simulating \( M \). \( A \) will work by using \( d \cdot n^d = r_n \) parallel processes \( B_1, \ldots, B_{r_n} \), each process for the simulation of the work of \( M \) on one square of the working tape.

Let the heads of \( A \) be \( H_0, H_1, H_2, \ldots, H_{3d+3} \). \( A \) uses its heads to store numbers. It is clear that any number \( j \in \{1, 2, \ldots, d \cdot n^d\} \), for \( n > d \), can be written as

\[
j = j_1 + j_2 n + j_3 n^2 + \cdots + j_{d+1} n^{d-1} + j_{d+2} n^d,
\]

each \( j_i < n \). To store \( j \), \( A \) uses \( d + 1 \) heads \( H_1, \ldots, H_{d+1} \) positioning \( H_i \) on the \( j_i \)th position of the input tape, for any \( i \in \{1, \ldots, d+1\} \).

For the \( i \)th computation step of \( M \) on an input \( w \), \( |w| = n \), let \( c_i \) (\( i \in \{1, 2, \ldots, T_M(n)\} \)) be the number of squares of the working tape used until the \( i \)th step. Simulating the \( i \)th step of \( M \), \( A \) makes use of \( c_i \) copies of itself: \( B_1, \ldots, B_{c_i} \), working in parallel. For each \( j \in \{1, \ldots, c_i\} \), \( B_j \) stores

(a) the number \( j \) by using the heads \( H_1, \ldots, H_{d+1} \);
(b) the position of the head of \( M \) on the working tape by using the heads \( H_{d+2}, \ldots, H_{2d+2} \);
(c) the number \( c_i \) by using the heads \( H_{2d+3}, \ldots, H_{3d+3} \);
(d) the actual state of \( M \) in its final control;
(e) the actual symbol \( b'_j \) on the \( j \)th position of the working tape of \( M \), in its final control.

All \( B_j \) have their input head on the same position of the input tape as \( M \) has. Clearly, there exists exactly one \( k \in \{1, \ldots, c_i\} \) such that \( B_k \) stores the same number by the heads \( H_1, \ldots, H_{d+1} \) and by the heads \( H_{d+2}, \ldots, H_{2d+2} \) (i.e., the cur-
rent position of $H_i$ is equal to the position of $H_{d+i+1}$ for each $i \in \{1, \ldots, d+1\}$. $B_k$ nondeterministically guesses this coincidence of its heads (in the meaning described above) and all other $B_i$ nondeterministically guess that they store different numbers by the heads $H_1, \ldots, H_{d+1}$ and $H_{d+2}, \ldots, H_{2d+2}$ (and check the correctness of their version similarly as described for $B_k$ in what follows). Now $B_k$ (or still any other processes guessing the coincidence of the heads) starts to be in a special checking state in which it branches into two processes $B_k$ and $B_k'$ in the universal way. Then $B_k'$ moves simultaneously the heads $H_1, \ldots, H_{d+1}, \ldots, H_{2d+2}$ to the right and stops in a final state iff the head $H_i$ reaches the right endmarker in the same moment as $H_{d+i+1}$ does, for all $i \in \{1, \ldots, d+1\}$. Obviously if a process different from $B_k$ guesses the coincidence of the heads then its copy will never reach an accepting state and the whole computation cannot be an accepting one.

Process $B_k$ enters a state with $b'_k$ as a synchronizing symbol. Every other $B_j$ ($j \neq k$) nondeterministically guesses a symbol from the working alphabet of $M$ and uses it as the synchronizing symbol. Obviously, only in the case all processes guess the symbol $b'_k$ read by $M$ on the $k$th position of the working tape, the computation can continue.

Now, all parallel processes know the state of $M$, the symbol read from the input tape and the symbol read from the working tape, i.e., all processes know the action $(q, b, z, r)$ of $M$, where $q$ is the new state, $b$ is the new symbol on the $k$th position of the working tape, $z, r \in \{0, 1, -1\}$ indicate the movement of the heads. So, all processes store $q$ in their final control (cf. (d)) and move $H_0$ according to $z$. The process $B_k$ stores $b$ instead of $b'_k$ in its final control (cf. (e)). All processes move their heads $H_{d+2}, \ldots, H_{2d+2}$ according to $r$ in order to store the actual position of the head on the working tape of $M$ (cf. (b)).

Moreover, in case $k = c_i$ and $r = 1$, $B_k$ universally branches into two processes $B_k$ and $B_{k+1}$. $B_{k+1}$ is the copy of $B_k$ which adds one to the number stored by $H_1, \ldots, H_{d+1}$ (cf. (a)) and stores blank as the actual symbol of the $(k+1)$th position. Finally $B_{k+1}$ enters a state with the synchronizing symbol 1 and all other processes guess nondeterministically one of two symbols 0 and 1 and enter a state with the guessed synchronizing symbol. Obviously the computation can continue only if all processes have guessed 1 which is interpreted for each process to add one to the number stored by the heads $H_{2d+3}, \ldots, H_{3d+3}$ (cf. (c)). In case $k < c_i$, $B_k$ enters a state with synchronizing symbol 0 and all other processes do so without changing the positions of their heads. Clearly $B_k$ is able to recognize whether $k = c_i$ or $k < c_i$ as was already described above.

Note that the proof of $\text{PSPACE} = \bigcup_{k \in \mathbb{N}} \mathcal{L}(2\text{SAFA}(k))$ implies that there is a strong infinite hierarchy according to the number of heads used by two-way multihead synchronized alternating finite automata. The open problem remains whether the result can be improved to $\mathcal{L}(2\text{SAFA}(k)) \subset \mathcal{L}(2\text{SAFA}(k+1))$ for every $k \geq 1$, i.e., whether it holds that already one additional head increases the computational power of two-way synchronized alternating finite automata.
4. Synchronized alternating finite automata

In this section we shall study the simplest synchronized alternating devices—the synchronized alternating finite automata. Despite the fact that two-way alternating finite automata recognize only regular sets [3] we have proved \( \mathcal{L}(2\text{SAFA}) = \mathcal{L}_{CS} \) in the previous sections.

We devote this section to the study of parallel complexity classes of \( X\text{SAFA} \), for \( X \in \{1,2\} \). First we shall consider the constant case, i.e., \( X\text{SAFA} \) with the number of parallel branches in the computation tree bounded by some constant \( k \). We prove that \( X\text{SAFA} \) with this restriction are equivalent to multithread finite automata. This supports the view that the number of input heads may be considered as parallel complexity measure for finite automata.

Before starting the presentation of results let us introduce some new notation. Let \( XR(f(n))\text{FA} \) denote \( X \)-way \( R \) finite automaton with parallelism bounded by \( f(n) \) (it means any computation tree of \( XR(f(n))\text{FA} \) working on an input of length \( n \) has at most \( f(n) \) leaves) for \( X \in \{1,2\} \) and \( R \in \{A, \text{SA}, \text{DSA}\} \), where DSA stands for “deterministic” synchronized alternation, i.e., for SA without existential states. The model \( X\text{DSA}(f(n))\text{FA} \) seems to be a very interesting one because it represents real parallel computing with \( f(n) \) deterministic finite processes, each having the access to the input and each able to communicate with others via synchronization. In what follows we shall simply write \( m \) instead of a constant function \( f_m(n) = m, \) for all \( m \), in our notation (for instance, \( XR(m)\text{FA} \)).

**Theorem 4.1.** For any \( k \in \mathbb{N} - \{0\} \) and \( X \in \{1,2\} \):

\[ \mathcal{L}(X\text{SA}(k)\text{FA}) = \mathcal{L}(X\text{NFA}(k)). \]

**Proof.** An \( X\text{NFA}(k) \) can be considered as \( k \) parallel nondeterministic finite automata with total information exchange among them. This clearly implies \( \mathcal{L}(X\text{SA}(k)\text{FA}) \subseteq \mathcal{L}(X\text{NFA}(k)) \).

To prove the opposite inclusion we shall use the technique developed in [15] and used for the more general result. An \( X\text{NFA}(k) \) automaton \( A \) can be simulated by an \( X\text{SA}(k)\text{FA} \) \( B \) which starts by universal branching into \( k \) automata \( B_1, \ldots, B_k \) and this is the only universal branching of \( B \) in the whole full configuration tree. Simulating one step of \( A \) each process \( B_i \) enters a special state with synchronizing symbol \( (a_1, \ldots, a_k) \), where \( a_i \) is the symbol currently scanned by \( B_i \) and \( a_{i+1}, \ldots, a_{i-1}, a_{i+1}, \ldots, a_k \) are symbols from the input alphabet, nondeterministically guessed by \( B_i \). Since each \( B_i \) stores in its state also the current state of \( A \) each \( B_i \) has the complete information needed to simulate the transition function of \( A \) after this synchronization. Again by using further synchronization all \( B_i \) choose nondeterministically the same action from the finite set of possibilities and the simulation continues by simulating the next step. \( \square \)
Corollary 4.2. For any \( k \in \mathbb{N} - \{0\} \) and \( X \in \{1, 2\} \):
\[
\mathcal{Q}(XSA(k)FA) \subseteq \mathcal{Q}(XSA(k+1)FA).
\]

Proof. The proof that \( k+1 \) heads are more powerful than \( k \) heads for one-way and two-way nondeterministic multihead automata can be found in [12, 11], respectively. □

Note Theorem 4.1 also implies the result \( \text{NLOGSPACE} = \bigcup_{k \in \mathbb{N}} \mathcal{Q}(2SA(k)FA) \) proved already by Slobodová [14], and \( \mathcal{Q}(1SA(k)FA) \subseteq \mathcal{Q}(2SA(k)FA) \) for any \( k \geq 1 \).

Clearly, considering \( XDFA(k) \) automata as \( k \) automata with total information exchange among them we have also \( \mathcal{Q}(XDSA(k)FA) \subseteq \mathcal{Q}(XDFA(k)) \), for any \( k \geq 1 \) and any \( X \in \{1, 2\} \). We conjecture that this inclusion is proper (we expected even more, namely that \( \mathcal{Q}(XDFA(2)) = \bigcup_{k \in \mathbb{N}} \mathcal{Q}(XDSA(k)FA) \neq \emptyset \)) but we were not able to prove it and state it as an open problem here.

Now, let us look for a hierarchy based on the number of deterministic parallel processes (finite automata). To do it we shall consider the following languages
\[
L_m = \{ w_1cw_2c\ldots cw_mcw_mc\ldots cw_2cw_1 \mid w_i \in \{0, 1\}^* \text{ for each } i \in \{1, \ldots, m\} \}
\]
for any \( m \in \mathbb{N} - \{0\} \).

Slobodová [16] already proved \( L_1 \in \mathcal{Q}(1DSA(2)FA) - \mathcal{Q}(2DSA(1)FA) \) (the later being \( \mathcal{R} \)). We shall investigate the hierarchy starting with this result.

Theorem 4.3. Let \( k \geq 2 \) be a positive integer. Then
\[
\mathcal{Q}(1DSA(k - 1)FA) \subseteq \mathcal{Q}\left(1DSA\left(\binom{k}{2} + 1\right)FA\right).
\]

Proof. Obviously \( \mathcal{Q}(1DSA(k - 1)FA) \subseteq \mathcal{Q}(1DSA(\binom{k}{2} + 1)FA) \).

To prove that this inclusion is proper we show that
\[
L_{c'}(i) \in \mathcal{Q}\left(1DSA\left(\binom{k}{2} + 1\right)FA\right) - \mathcal{Q}(1DSA(k - 1)FA).
\]
Rivest and Yao proved
\[
L_{c'}(i) \notin \mathcal{Q}(1DFA(k - 1))
\]
in [12] which implies
\[
L_{c'}(i) \notin \mathcal{Q}(1DSA(k - 1)FA).
\]
Now it is sufficient to show that \( L_m \in \mathcal{Q}(1DSA(m + 1)FA) \), for any positive integer \( m \). Let us prove this fact by induction.

Obviously \( L_1 = \{ wcw \mid w \in \{0, 1\}^* \} \in \mathcal{Q}(1DSA(2)FA) \). Now, let us assume that
We show that there is a lDSA(m + 1)FA $A$ recognizing $L_m$. Let $A$ starts by splitting itself into two processes $A$ and $B$. They work on an input

$$x = w_1 c w_2 c \ldots c w_m c u_1 c \ldots c u_i c _1$$

as follows. $A$ crosses $w_1$ producing $w_1$ as its synchronizing sequence and stops in a special state when it reaches the symbol $c$. $B$ moves to the right until it crosses $2m$ symbols $c$. Then $B$ crosses $u_1$ producing $u_1$ as its synchronizing sequence and stops in a final state when it reaches the right endmarker. Afterwards $A$ can continue as an lDSA(m − 1)FA automaton recognizing $L_{m-1}$.

We have proved above an infinite hierarchy on the number of deterministic parallel processes communicating by synchronization. We conjecture that a much stronger hierarchy may be proved, namely, that $k + 1$ deterministic parallel processes are more powerful than $k$ ones. It is another of our open problems.

Turning to the nonconstant case of parallel complexity we first give an upper bound on the power of such machines in terms of nondeterministic space classes.

**Theorem 4.4.** For any function $f: \mathbb{N} \rightarrow \mathbb{N}$:

$$\mathcal{L}(2SA(f(n)FA) \subseteq \text{NSPACE}(f(n)\log_2 n).$$

**Proof.** To prove this result it is sufficient to use Procedure SIMULATION. Finite automata use constant space, storing the input head position requires $\log_2 n$ space, and there are at most $f(n)$ configurations to be stored for each meaningful cut. Thus the result follows.

The last result we want to prove in this section is a hierarchy result for the nonconstant case of parallel complexity. We prove the result for the modified version of acceptance only. The difficulty is that we do not see the way to force the synchronized alternating finite automaton to a nonaccepting configuration in case the number of parallel processes in a computation on an input of length $n$ exceeds $f(n)$ for a given nonconstant function $f$. To overcome this difficulty we assume that there is an external control that stops any computation that contains more parallel processes than the upper bound is. This means that we will work with the parallel complexity classes $\mathcal{L}'(XSA(f(n)FA)$ containing all languages accepted by XSAFA's in the following way. An XSAFA $A$ accepts the language $L'(A)$ iff for each input word $w \in L'(A)$ there is an accepting computation of $A$ on $w$ having at most $f(|w|)$ leaves, i.e., $L'(A) \subseteq L(A)$ and this inclusion may be proper.

Now let us start to prove some hierarchy results by proving lower bounds for the recognition of languages

$$L(f) = \{w = w_1 c w_2 c \ldots c w_f(n)c w_f(n)c \ldots c w_2 c w_1 c^{r(n)} |$$

$$|w| = n, w_i \in \{0, 1\}^{f(n)}, \text{ for } i \in \{1, \ldots, f(n)\},$$

$$r(n) = n - 2f(n) \cdot (f(n) + 1)\}$$
and

\[ SL(f) = \{ w \in \{0, 1, c\}^* \mid w \in L(h) \text{ for a function } h: \mathbb{N} \to \mathbb{N} \text{ such that} \]
\[ h(n) \leq f(n), \text{ for all } n \in \mathbb{N} \}, \]

\( f \) being an arbitrary function \( f: \mathbb{N} \to \mathbb{N} \) such that \( 2f(n) \cdot (f(n) + 1) \leq n \).

To prove our lower bounds we generalize the crossing sequence lower bound technique used by Rivest and Yao [12] for multihead automata and developed for alternating machines in [7]. In order to do so we need to study computations of SAM's in much greater detail than before. We have already introduced the notion of meaningful cut to reflect the fact that when "growing" the computation tree from the root to the leaves not all "frontiers" are achievable (i.e., synchronization prevents some branches to "outgrow" the others too much). Now we need to point out the fact, that some pairs \((Z_1, Z_2)\) of MC's are not achievable "in the same process of growing the computation tree" (cf. Fig. 1). To avoid unnecessary ambiguity in describing (the process of "growing") the computation tree by a sequence of meaningful cuts we shall assume the full configuration tree is ordered (i.e., some left to right order for descendants of each node is given, i.e., by the lexicographic order on the configurations labeling these nodes). For a MC \( Z \) this implies a natural order for the MCCM \( \text{cm}(Z) \), thus each MCCM can be viewed as a sequence. If we now consider Procedure SIMULATION modified so that it outputs the sequence of consecutive contents of \( Z \) we obtain a description of a particular computation tree in terms of the consecutive MCCM's representing the consecutive "frontiers of the growing computation tree". To make this description unique we modify lines 10-12 to use the leftmost \( C \) in the current ordered MCCM \( Z \). We shall call such a procedure MODIFIED SIMULATION.

**Definition 4.5.** Let \( T \) be a computation tree of \( A \). A canonical sequence of MCCM's for \( T \) is the sequence \( \text{can}(T) \) of consecutive contents of \( Z \) in the run of MODIFIED SIMULATION that produces \( T \).
Note, that for a given \( A \) and \( w \) there may be several distinct canonical sequences (some even infinite). This is due to the nondeterministic choices in lines 8 and 12 and merely reflects the fact that there may be several computation trees for \( A \) on \( w \). However, it is easy to see, that the following two propositions hold.

**Proposition 4.6.** For a given computation tree \( T \) of \( A \) on \( w \), \( \text{can}(T) \) is unique.

**Proposition 4.7.** Given an automaton \( A \) and a finite sequence of finite sequences of configurations of \( A \) it is decidable whether it forms a canonical sequence for some initial subtree \( T' \) of some computation tree \( T \) of \( A \). Moreover, in case it does, \( T' \) is uniquely determined by \( \text{can}(T') \).

We are now ready for our lower bound lemma.

**Lemma 4.8.** Let \( A \) be a \( 1SA(g(n))FA \) for a function \( g : \mathbb{N} \rightarrow \mathbb{N} \). If \( \text{SL}(f) \supseteq L'(A) \supseteq L(f) \), for an unbounded nondecreasing function \( f : \mathbb{N} \rightarrow \mathbb{N} \), then \( g(n) \in \Omega((f(n)/\log_2 n)^{1/2}) \).

**Proof.** Let \( A \) be a \( 1SA(g(n))FA \) automaton such that \( \text{SL}(f) \supseteq L'(A) \supseteq L(f) \) and let \( A \) have \( s \) states. Let \( m \) be such that \( m \geq s \) and for all \( n \geq m \), \( f(n) \geq 2\log_2 f(n) \) (such \( m \) has to exist because \( f \) is an unbounded nondecreasing function). We shall prove by contradiction that for all \( n \geq m \)

\[
g(n) \geq (f(n)/\log_2 n)^{1/2}/8.
\]

Let us assume that there exists \( d \geq m \) such that

\[
g(d) < (f(d)/\log_2 d)^{1/2}/8.
\]

We shall show that (1) implies that there is a word \( y \in L'(A) - \text{SL}(f) \), which will be a contradiction.

Let \( L_d(f) = \{ w \in L(f) \mid |w| = d \} \). Let us consider for each \( x \in L_d(f) \) a fixed accepting computation (tree) \( D_x \) on \( x \) with a minimal number of leaves.

Let \( x = w_1cw_2c...cw_{(\log_2 f(n))}c...cw_2cw_1c^{(n)} \in L_d(f) \). We say that the \( i \)th pair of subwords \( w_i \) is compared in \( \text{can}(D_x) \) if there exists a MCCM in \( \text{can}(D_x) \) in which one finite automaton (say \( B_1 \)) is positioned on the first subword \( w_i \) and another finite automaton (say \( B_2 \)) is positioned on the second \( w_i \). We also say that finite automata \( B_1 \) and \( B_2 \) compare the \( i \)th pair of subwords in \( \text{can}(D_x) \).

Let us consider a pair of finite automata \( (B_1, B_2) \) such that \( B_1 \) and \( B_2 \) compare the \( j \)th pair of subwords in a MCCM in \( \text{can}(D_x) \). Since \( B_1 \) and \( B_2 \) are one-way automata there cannot exist an MCCM in \( D_x \) in which the pair \( (B_1, B_2) \) compares the \( b \)th pair of subwords for some \( b, b \neq j \). So each pair of automata can compare at most one pair of subwords of \( x \) in the representation \( \text{can}(D_x) \) of the accepting computation \( D_x \). Since the number of finite automata working in parallel is bounded by \( g(d) \), there are at most
pairs of finite automata. Thus, for each \( x \in L_d(f) \) there exists \( j \in \{1, \ldots, f(d)\} \) such that the \( j \)th pair of subwords \( w_j \) is not compared in \( \text{can}(D_x) \).

Note how important it is for our argumentation that we have based the definition of pairs of subwords being compared on a particular fixed representation \( \text{can}(D_x) \) of the computation tree \( D_x \). Using arbitrary MCCM of \( D_x \) instead of MCCM in \( \text{can}(D_x) \) would render this notion useless due to situations like that in Fig. 1.

Let \( L_{b,d}(f) \subseteq L_d(f) \) be the set of those words
\[
x = w_1 c w_2 c \ldots c w_{f(d)} c c w_{f(d)} c \ldots c w_2 c w_1 c^{(n)}
\]
for which the \( i \)th subwords \( w_i \) are not compared in \( \text{can}(D_x) \). Since \( |L_d(f)| = 2^{(f(d))^2} \) and there are \( f(d) \) sets \( L_{i,d}(f) \), there exists \( b \in \{1, \ldots, f(n)\} \) such that \( |L_{b,d}(f)| \geq 2^{(f(d))^2} / f(d) \). Since there are \( 2^{(f(d))^2} / f(d) \) ways to choose \( w_1, w_2, \ldots, w_{b-1}, w_{b+1}, \ldots, w_{f(d)} \), there exist \( u_1, u_2, \ldots, u_{b-1}, u_{b+1}, \ldots, u_{f(d)} \) such that for
\[
L_{b,d}(f) = L_{b,d}(f) \cap u_1 c \ldots c u_{b-1} c \{0,1\}^* c u_{b+1} c \ldots c u_{f(d)} c c u_{f(d)} c \ldots c u_1 c^{(n)}
\]
we have
\[
|L_{b,d}(f)| \geq |L_{b,d}(f)| / 2^{(f(d))^2} / f(d) \geq 2^{f(d)} / f(d).
\]

We shall now consider \( \text{can}(D_x) \) for words \( x \in L_{b,d}(f) \). We shall say that an MCCM \( Z \) in \( \text{can}(D_x) \) is an important MCCM if
(a) in each configuration in \( Z \) the input head scans a position outside the \( b \)th subwords; and
(b) either the succeeding or the preceding MCCM in \( \text{can}(D_x) \) contains a configuration in which one of the \( b \)th subwords is scanned.

Note that (a) and (b) imply that an important MCCM must contain a configuration in which a symbol \( c \) adjacent to the \( b \)th subword is scanned. Thus there are at most \( 4g(d) \) important MCCM’s in \( \text{can}(D_x) \) for \( x \in L_{b,d}(f) \), since there are at most \( g(d) \) parallel processes.

We shall now need to extract from configuration \( C \) the internal state and head position only (thus forgetting the input). We shall call such a pair the base configuration associated with the configuration \( C \), denoted base(C). We extend this to base MCCM’s having base(Z) to mean the sequence of base configurations of the MCCM \( Z \).

Let \( \text{can}(D_x) = X_1, Z_1, X_2, \ldots, X_m, Z_m, X_{n+1} \), where each \( X_i \) is a sequence of MCCM’s and \( Z_1, \ldots, Z_n \) are all important MCCM’s occurring in \( \text{can}(D_x) \). The sequence \( P(D_x) = \text{base}(Z_1), \ldots, \text{base}(Z_n) \) is called a pattern of \( x \). Since the number of different base configurations is at most \( s \cdot d \) and the number of parallel processes is at most \( g(d) \), the number of different base MCCM’s is at most \( (sd)g(d) \). Since
J. Hromkovič et al.

each pattern contains at most $4g(d)$ base MCCM's there are at most \((sd)^{4g(d)}\) different patterns of words in $L'_{b,d}(f)$.

Applying this fact, (1) and the facts $d \geq s$ and $f(d) \geq 2\log_2(f(d))$ we obtain

\[
|L'_{b,d}(f)| \geq 2^{f(d) / f(d)} > (sd)^{4g(d)^2} + 1.
\]  

(To prove (2) one has to start from (1) and to derive the following sequence of implications

\[
g(d) < \frac{1}{4}(f(d)/\log_2 d)^{1/2},
\]

\[
16(g(d))^2 < \frac{1}{4}(f(d)/\log_2 d).
\]

\[
16(g(d))^2 < f(d)/\log_2 d - \log_2 f(d)/\log_2 (sd),
\]

\[
8(g(d))^2 < f(d)/\log_2 (sd) - \log_2 f(d)/\log_2 (sd),
\]

\[
(sd)^{8g(d)^2} < 2^{f(d)/f(d)},
\]

which directly implies (2).)

So, there are two words:

\[
y_1 = x_1c \ldots cx_{b-1}Cu_{b+1}c \ldots cx_{f(d)}ccx_{f(d)}c \ldots cx_{b+1}Cu_{b}c \ldots cx_1
\]

\[
y_2 = x_1c \ldots cx_{b-1}Cu_{b}c \ldots cx_{f(d)}ccx_{f(d)}c \ldots cx_{b+1}Cu_{b}c \ldots cx_1
\]

in $L'_{b,d}(f)$ such that

(i) $u_b \neq v_b$,

(ii) $P(D_u) = P(D_v)$,

(iii) there is no MC in $can(D_u)$ (respectively $can(D_v)$), in which subwords $u_b$ (respectively $v_b$) are compared.

We claim that at least one of

\[
y_1 = x_1c \ldots cx_{b-1}Cu_{b+1}c \ldots cx_{f(d)}ccx_{f(d)}c \ldots cx_{b+1}Cu_{b}c \ldots cx_1
\]

\[
y_2 = x_1c \ldots cx_{b-1}Cu_{b}c \ldots cx_{f(d)}ccx_{f(d)}c \ldots cx_{b+1}Cu_{b}c \ldots cx_1
\]

belongs to $L'(A)$ which implies a contradiction because $y_1$ and $y_2$ do not belong to $SL(f)$ due to (i).

First, let us construct some accepting computations of $A$ on $y_1$ and $y_2$ to show that both words belong to $L(A)$. Let

- $can(D_u) = X_1^u, Z_1^u, X_2^u, Z_2^u, \ldots, Z_k^u, X_{k+1}^u$,
- $can(D_v) = X_1^v, Z_1^v, X_2^v, Z_2^v, \ldots, Z_k^v, X_{k+1}^v$,

where $\text{base}(Z_i^u), \ldots, \text{base}(Z_k^u) = P(D_u) = P(D_v) = \text{base}(Z_i^v), \ldots, \text{base}(Z_k^v)$, and $X_i^u$ (respectively $X_i^v$) is a sequence of MCCM's for $i \in \{1, 2, \ldots, k+1\}$.

Note that from the definition of important configurations and the fact the $b$th...
subwords are not compared in \(\text{can}(D_u)\) and \(\text{can}(D_v)\) it follows, that the sub-
sequences \(X_i^u\) (respectively \(X_i^v\)) are of three types only:

T1. some processes read \(u_b\) (respectively \(v_b\)) in the first half of \(u\) (respectively \(v\))
within \(X_i^u\) (respectively \(X_i^v\));
T2. some processes read \(u_b\) (respectively \(v_b\)) in the second half of \(u\) (respectively \(v\))
within \(X_i^u\) (respectively \(X_i^v\));
T3. no process reads \(u_b\) (respectively \(v_b\)) within \(X_i^u\) (respectively \(X_i^v\)).

By Definition 4.5 of the canonical sequence and the fact the choice of the leftmost
configuration in MODIFIED SIMULATION depends on the internal state (hence
base configuration) only, \(X_i^u\) and \(X_i^v\) are of the same of the above three types for
each \(i \in \{1, \ldots, k+1\}\).

We can thus construct

\[
\text{can}(D_u) = Y_1^1, Z_1^1, \ldots, Y_{k+1}^1,
\]
\[
\text{can}(D_v) = Y_1^2, Z_1^2, \ldots, Y_{k+1}^2,
\]

choosing \(Y_i^j\) and \(Z_i^j\) so that for each \(i\)

\[
\text{base}(Z_i^j) = \text{base}(Z_i^j) = \text{base}(Z_i^j) = \text{base}(Z_i^j),
\]

\[
\text{base}(Y_i^j) = \begin{cases} 
\text{base}(X_i^u), & \text{if } X_i^u \text{ is of the type T1}, \\
\text{base}(X_i^v), & \text{if } X_i^v \text{ is of the type T2}, \\
\text{base}(X_i^w), & \text{if } X_i^w \text{ is of the type T3, with } w \in \{u, v\} \\
\text{minimizing the number of leaves in MCCM's of } X_i^w
\end{cases}
\]

and

\[
\text{base}(Y_i^2) = \begin{cases} 
\text{base}(X_i^u), & \text{if } X_i^u \text{ is of the type T1}, \\
\text{base}(X_i^v), & \text{if } X_i^v \text{ is of the type T2}, \\
\text{base}(Y_i^1), & \text{if } X_i^v \text{ is of the type T3}.
\end{cases}
\]

It is easy to check, that \(\text{can}(D_u)\) and \(\text{can}(D_v)\) so constructed are indeed canonical
sequences of MCCM's and the corresponding computation trees are accepting com-
putations.

To complete the proof it is sufficient to note that the minimum of the number
of leaves of the two computations \(D_{y_1}\) and \(D_{y_2}\) is smaller and equal to the max-
imum of the numbers of leaves of the computations \(D_u\) and \(D_v\). So at least one of
the computations \(D_{y_1}\) and \(D_{y_2}\) is an accepting computation with at most \(g(n)\)
leaves, i.e., \(\{y_1, y_2\} \cap (L'(A) - \text{SL}(f)) \neq \emptyset\). \(\square\)

**Corollary 4.9.** Let \(A\) be a \(\text{1SA}(g(n))\)FA such that \(L'(A) = \text{SL}(n^{1/2}/2)\). Then
\(g(n) \in \Omega(n^{1/4}/(\log n)^{1/2})\).
Now, let us prove an upper bound on the parallel complexity of the acceptance of the languages $\text{SL}(f)$.

**Lemma 4.10.** Let $f$ be a function from $\mathbb{N}$ to $\mathbb{N}$. Then there exists a $1\text{SA}(3f(n) + 7)\text{FA}$ $A$ such that $L'(A) = \text{SL}(f)$.

**Proof.** Let $A$ start the work on an input $w_1c w_2c \ldots c w_k c c u_k \ldots c u_2 c u_1 c^{r(n)}$ by splitting into four automata $A_0, A_1, A_2$ and $A_3$. Then $A_0$ enters an idling state and $A_1, A_2$ and $A_3$ check in a deterministic way, synchronizing with each other, whether the number of occurrences of symbols $c$ in the prefix $w_1 c w_2 c \ldots c w_k c c u_k \ldots c u_2 c u_1$ is the same as the length of $w_1 c$ and as the number of the occurrences of symbol $c$ in $c c u_k \ldots c u_2 c u_1$. Then $A_1, A_2$ and $A_3$ stop in an accepting state. Next, $A_0$ produces two finite automata $A_4$ and $A_5$ that check whether $|w_1| = |w_2| = \ldots = |w_k| = |u_k| = \ldots = |u_2| = |u_1|$, and stops in an accepting state. Now $A_0$ splits itself into $A_0$ and $B$. $A_0$ remains on the first symbol of the input tape and $B$ crosses $w_1 c w_2 c \ldots c w_k c c$ splitting itself into two processes after reading a symbol from $\{0, 1\}$ laying immediately after each symbol $c$ and on the first symbol of $w_j$. So we have now $k + 2$ processes: $A_0, B, H_1, \ldots, H_k$, each $H_i$ is positioned on the first symbol of $w_j$.

In what follows $B$ shall move right and while traversing $u_i$ checks for coincidence of $u_i$ and $w_i$ by synchronizing itself with $H_i$. Since $H_i$ becomes active nondeterministically it is then necessary to check whether $B$ synchronized itself with the correct $H_i$. Thus to check $u_k = w_k$ all processes $A_0, B, H_1, \ldots, H_k$ enter states with special synchronizing symbol $S_1$. Afterwards $A_0$ starts idling, $B$ remains active and each of the $H_i$ decide whether to be active or idling. (In fact we want only $H_k$ to be active.) The active $H_i$ (presumably $H_k$ only) and $B$ produce synchronizing symbol $S_2$ and $A_0$ and $B$ split a copy $A_0'$ and $B'$ of themselves.

Now $A_0, B$ and $H_k$ (all active $H_i$) decide deterministically to be idling in another idling state than the already idling $H_i$ are. Then $A_0'$ and $B'$ produce a special synchronizing symbol $S_2$ and move to the right producing the synchronizing symbol $c$ for any $c$ scanned by them. $B'$ nondeterministically stops this synchronization when it reaches the first symbol $c$ of $c^{r(n)}S$ or directly the symbol $S$. $B'$ deterministically checks the correctness of its decision, produces the special synchronizing symbol $S_1$ and halts in an accepting state. $A_0'$ nondeterministically stops when its head coincides with the head of $H_k$, and it produces also the synchronizing symbol $S_3$. Now $H_k$ (and all $H_i$ active before the synchronizing cut with the synchronizing symbol $S_2$) starts to be active by splitting itself into $H_k$ and $H_k'$. Now all processes produce a new synchronizing symbol $S_4$ and after that $H_k$ starts to be idling again. $H_k'$ with $A_0'$ check by synchronization whether they coincide. This checking is finished by synchronizing with a special symbol $S_2$. Afterwards $B$ and $H_k$ start to be active and check whether $w_k = u_k$ by synchronization. Then $H_k$ stops in an accepting state. Clearly, if any other process from $H_1, \ldots, H_{k-1}$ was active between the synchronizing cuts with synchronizing symbols $S_1$ and $S_2$ (and so also between $S_3$ and $S_4$) then the computation cannot be accepting. (We simplified the presentation to avoid clut-
Afterwards the computation continues by checking whether $w_{k-1} = u_{k-1}$ in the same way as described above, and so on. Obviously, ISAFA $A$ produces an accepting computation with exactly $3k + 7$ parallel processes for an input word

$$w_1 c w_2 c \ldots c w_k c c w_k c \ldots c w_2 c w_1 \in \text{SL}(f).$$

Following Lemma 4.5 and Lemma 4.7 we can establish the following hierarchy.

**Theorem 4.8.** For any unbounded, nondecreasing function $f: \mathbb{N} \to \mathbb{N}$ such that

$$f(n) \leq (n^{1/2} - 1)/2,$$

we have

$$\mathcal{P}'(1\text{SA}((f(n))^{1/2}/\log_2 n)\text{FA}) \subset \mathcal{P}'(1\text{SA}(3f(n) + 7)\text{FA}).$$

We conjecture that there is a much stronger hierarchy and that it can be extended also for $f(n) \geq (n^{1/2} - 1)/2$. To improve or to extend the result of Theorem 4.8 is left as a motivation for further research in this area. May be that the simplest way to do it is to improve the acceptance procedure from Lemma 4.7. Still more interesting research idea is to prove a hierarchy for the two-way case. One can expect that the crucial problem is in proving a lower bound on the parallel complexity of 2SAFA's.

5. Synchronization and decidability

In this section we consider decidability questions for synchronized (alternating) finite (one-way) automata, in particular the decidabilities of the equivalence problem. Our considerations are based on an observation that synchronized automata are closely related to finite transducers—the synchronizing symbols correspond to outputs of transducers.

In order to make this connection clearer we restrict our considerations to ordinary finite automata without alternation. Moreover, we slightly modify the notion of synchronization (therefore we will use a different abbreviation for the automata). Indeed, instead of requiring that the synchronizing sequences in accepting computations are in the prefix relation we now require that they coincide. Hence, the languages accepted by, e.g., a one-way deterministic synchronized automata with two parallel processes, i.e., a 1DSFA(2) (or equivalently 1DSA(2)FA in the previous section) $A$, and one-way nondeterministic synchronized finite automata with two parallel processes, i.e., a 1NSFA(2) $B$, can be expressed in the forms:

1. $L(A) = \{x \mid s_1(x) = s_2(x)\}$,
2. $L(B) = \{x \mid t_1(x) \cap t_2(x) \neq \emptyset\},$

where $s_i$ are deterministic generalized sequential machines (DGSM's) mapping in-
put words to sequences of synchronizing symbols, and \( t_i \) are finite transducers mapping input words to finite sets of sequences of synchronizing symbols. Observe that in this formalism the transducers are not \( \lambda \)-free—outputs of the transducers are empty whenever the original automaton enters a nonsynchronizing state.

For this model of synchronization we use an abbreviation \( S_c \) instead of \( S \) alone. If the synchronizing alphabet is unary we use an superscript \( u \), e.g. \( S^u \). By (1) and (2) the acceptance of words is defined via an equality mechanism of two mappings (see, e.g. [13]). Hence, in the case of many-valued mappings, that is to say, in the case of nondeterministic automata, the acceptance can be defined in several different ways, cf. [10]. We can ask that, for an input word, a common synchronizing sequence is produced, as in (2), or we can ask that, for a given input word, the sets of all synchronizing sequences are the same, or even that they are the same as multisets. We will consider here the two first possibilities and consider these two as existential and universal synchronization mechanisms; abbreviated \( S_{e3} \) and \( S_{ev} \).

Observe, as we already noted, that \( S_{e3} \) equals to \( S_c \) in our terminology and in the previous sections.

After fixing our definitions and notations a few remarks are in order.

(i) Our transducer approach to define synchronization is natural only in connection with ordinary automata without alternation.

(ii) When defining synchronized automata via transducers we can, without loss of generality, assume that transducers are normalized, i.e., the outputs are of the length at most 1 for each transition. This makes our original definition and the newer one directly translatable to both directions.

(iii) For deterministic, as well as for nondeterministic, automata \( S_c \)-synchronization can be simulated by \( S \)-synchronization as follows: If \( L \) is accepted by a synchronized automaton in the equality sense, then \( L_S \) is accepted by a similar automaton in the prefix sense. This is achieved by synchronizing the computations at the end by a new synchronizing symbol.

(iv) For nondeterministic automata the synchronization models \( S \) and \( S_c \) are equivalent. This is because of (iii) and the fact that a nondeterministic automaton can always continue a prefix of synchronizing sequence to the whole sequence by guessing.

Now we turn to our results on equivalence problems for synchronized automata. Our first result emphasizes the power of the synchronization mechanism.

**Theorem 5.1.** *The emptiness problem for \text{IDS}_c\text{FA}(2) is undecidable.*

**Proof.** Theorem 5.1 follows immediately from the fact that the synchronization is defined via the equality mechanism, cf. (1). Indeed, for two morphisms \( h, g : \Sigma^* \to \Delta^* \) their equality language

\[
E(h, g) = \{ x \in \Sigma^* | h(x) = g(x) \}
\]
Synchronization in parallel computations

is in 1DS\(_n\)FA(2), and the emptiness problem for equality languages is exactly the Post Correspondence Problem. □

**Corollary 5.2.** The equivalence problem for 1DS\(_n\)FA(2) is undecidable.

The cardinality of the synchronizing alphabet is decisive in the above corollary. Clearly, two is enough to make the problem undecidable. On the other hand we have

**Theorem 5.3.** The equivalence problem for 1DS\(_n\)FA(k) is decidable.

**Proof.** Let \(A_1\) and \(A_2\) be automata in 1DS\(_n\)FA(k), for some \(k \geq 1\). It is straightforward to construct a deterministic 2k-counter automaton which accepts the symmetric difference of \(L(A_1)\) and \(L(A_2)\) by changing the direction in each counter only once. For such automata the emptiness problem is shown decidable in [8]. □

Corollary 5.2 and Theorem 5.3 settle the equivalence problems for deterministic synchronized one-way finite automata working on one tape in parallel way. (Recall as we already noted that 1DS\(_n\)FA(k) = 1DSA(k)FA.) For corresponding nondeterministic automata the question of whether or not the result of Theorem 5.3 holds true remains open: Is the equivalence problem for 1NS\(_n\)FA(k) decidable?

We have the following related result:

**Theorem 5.4.** The equivalence problem for 1NS\(_n\)FA(k) is undecidable.

**Proof.** Let \(A_1\) and \(A_2\) be two finite transducers with a unary output alphabet and equal domains. Clearly, the language

\[
\{ x \mid t_1(x) = t_2(x) \}
\]

can be recognized by an automaton in NS\(_n\)FA(k), say \(A_1\). Let \(A_2\) be a finite automaton accepting the common domain of \(A_1\) and \(A_2\).

Then \(A_1\) and \(A_2\) are equivalent iff the transducers \(t_1\) and \(t_2\) are so. However, a deep result of [9] shows that the equivalence for transducers with a unary output alphabet is undecidable. □

Next we turn our attention to automata with multiple input tapes, recognizing relations rather than languages, see [2,5]. Let us denote by 1DT(k)FA and 1NT(k)FA the deterministic and nondeterministic k-input tape automata, respectively. The equivalence problem for 1NT(k)FA, i.e., the equivalence problem for finite transducers, is a well-known undecidable problem, cf. [1], while the equivalence problem for 1DT(2)FA is decidable, cf. [2]. To extend the latter decidability result for arbitrary \(k\) is a well-known open problem. We suggest to attack this problem by weakening the machine model. The original model can be view-
ed as a $k$ finite automata each working on its own tape but having a "full communication" possibility via a common control. In our model we allow communication via synchronization only. This provides a natural intermediate case between "full" and "no" communication of finite automata.

A one-way deterministic synchronized $k$-tape finite automaton, abbreviated $\text{1DS}_k \mathsf{T}(k) \text{FA}$, accepts a $k$-tuple of words on its $k$ tapes iff all FA's produce the same synchronizing sequence (and stop in an accepting state). For nondeterministic finite automata we again have at least two possibilities for defining the acceptance—existential and universal. A one-way nondeterministic existential synchronized $k$-tape FA, abbreviated $\text{1NS}_{\mathsf{ex}} \mathsf{T}(k) \text{FA}$, accepts the contents of its $k$ tapes $(x_1, \ldots, x_k)$ iff $S_1(x_1) \cap \cdots \cap S_k(x_k) \neq \emptyset$, where $S_i(x_i)$ is the set of all possible synchronizing sequences produced by the $i$th FA on the input $x_i$. The corresponding universal automaton, abbreviated $\text{1NS}_{\mathsf{un}} \mathsf{T}(k) \text{FA}$, accepts $(x_1, \ldots, x_k)$ iff $S_1(x_1) = \cdots = S_k(x_k)$.

**Theorem 5.4.** The equivalence problem for $\text{1DS}_k \mathsf{T}(k) \text{FA}$ is decidable.

**Proof.** Let $A$ be an automaton $\text{1DS}_k \mathsf{T}(k) \text{FA}$. The relation it accepts is of the form:

$$\{ (x_1, \ldots, x_k) \mid s_1(x_1) = \cdots = s_k(x_k) \}$$

$$= \{ (x_1, \ldots, x_k) \mid S_1(x_1) = S_2(x_2), \ldots, s_1(x_1) = s_k(x_k) \}$$

$$= \{ (x_1, \ldots, x_k) \mid x_1 \in S_1^{-1}S_2(x_2), \ldots, x_k \in S_k^{-1}S_1(x_1) \},$$

where each $S_i$ is a DGSM.

Now, Theorem 5.4 follows since it is decidable whether two many-valued mappings of the form $s^{-1}r$, where $s$ and $r$ are DGSM's, are equivalent on a given regular language, cf. [17]. (Actually, Theorem 5.4 follows also from the rather difficult result of [2], solving the equivalence for $\text{1DT}(2) \text{FA}$.) □

From the equality $\text{1NS}_{\mathsf{ex}} \mathsf{T}(k) \text{FA} = \text{1NT}(k) \text{FA}$ (cf. the proof of Theorem 4.1) we conclude a nondeterministic variant of Theorem 5.4.

**Theorem 5.5.** The equivalence problem for $\text{1NS}_{\mathsf{ex}} \mathsf{T}(2) \text{FA}$ is undecidable.

We do not know whether the undecidability of this result can be extended to a unary synchronizing alphabet, i.e. we have an open problem: Is the equivalence problem for $\text{1NS}_{\mathsf{ex}} \mathsf{T}(k) \text{FA}$ decidable?

We also can solve the universal variant of the problem of Theorem 5.5.

**Theorem 5.6.** The equivalence problem for $\text{1NS}_{\mathsf{ex}} \mathsf{T}(2) \text{FA}$ is undecidable.

**Proof.** Let $t_1$ and $t_2$ be two finite transducers such that they possess a common in-
put alphabet $\Sigma$ and their output alphabets are disjoint from $\Sigma$. We define transducers $\tilde{t}_i$, for $i = 1, 2$, by setting

$$\tilde{t}_i(x) = \{x\} \cup t_i(x), \quad \text{for } x \in \Sigma^*.$$ 

Obviously, we can construct an automaton $A(t_1, t_2)$ in $1\text{NS}_{ev}T(2)\text{FA}$ which accepts the relation

$$\{(x, y) \mid \tilde{t}_1(x) = \tilde{t}_2(y)\}.$$ 

Further let $A_{\text{diag}}$ be an automaton in $1\text{NS}_{ev}T(2)\text{FA}$ accepting the relation

$$\{(x, x) \mid x \in \Sigma^*\}.$$ 

It follows from the construction that $A_{\text{diag}}$ and $A(t_1, t_2)$ are equivalent iff $t_1$ and $t_2$ are so. Hence, Theorem 5.6 follows from the undecidability of the equivalence problem for finite transducers, cf. [1].

It is interesting to note that, although the equivalence problem for finite transducers remains undecidable also in the case of unary output alphabet, cf. [9], we cannot extend Theorem 5.5 to a unary synchronizing alphabet. Indeed, our construction increases the size of that alphabet. Consequently, we again have an open problem: Is the equivalence problem for $1\text{NS}_{ev}T(k)\text{FA}$ decidable?

We have considered the equivalence problem for synchronized one-way finite automata. The automata have been deterministic or nondeterministic as well as multitape or multihead automata. It turned out the equivalence problem is always undecidable (cf. however Theorem 5.4) if the synchronization alphabet is at least of the cardinality two. This is a clear evidence of the power of the synchronization mechanism.

On the other hand, if the synchronization alphabet is unary then we have both decidable and undecidable cases, as well as their open problems, namely the equivalence problems for the classes $1\text{NS}_{ev}T(k)$FA, $1\text{NS}_{ev}U(k)$FA and $1\text{NS}_{ev}T(k)$FA.

Acknowledgement

We acknowledge Jürgen Dassow for taking part in the initial discussions starting the investigation of the power of synchronization in parallel computations.

References