Theoretical Computer Science 99 (1992) 243–264
Elsevier

On space-bounded synchronized alternating Turing machines*

Oscar H. Ibarra and Nicholas Q. Trân

Department of Computer Science, University of California at Santa Barbara, Santa Barbara, CA 93106, USA

Communicated by A. Salomaa
Received February 1990
Revised September 1990

Abstract

Ibarra, O.H. and N.Q. Trân, On space-bounded synchronized alternating Turing machines, Theoretical Computer Science 99 (1992) 243–264.

We continue the study of the computational power of synchronized alternating Turing machines ($SATM$) introduced in (Hromkovič 1986, Slobodová 1987, 1988a, b) to allow communication via synchronization among processes of alternating Turing machines. We are interested in comparing the four main classes of space-bounded synchronized alternating Turing machines obtained by adding or removing off-line capability and nondeterminism ($1SUTM(S(n))$, $SUTM(S(n))$, $1SATM(S(n))$, and $SATM(S(n))$) against one another and against other variants of alternating Turing machines. Denoting the class of languages accepted by machines in C by $\mathscr{L}(C)$, we show as our main results that $\mathscr{L}(1SUTM(S(n)))\subset\mathscr{L}(SUTM(S(n)))\subset\mathscr{L}(1SATM(S(n)))=\mathscr{L}(SATM(S(n)))$ for all space-bounded functions $S(n)\in o(n)$, and $\mathscr{L}(1SUTM(S(n)))=\mathscr{L}(SUTM(S(n)))\subset\mathscr{L}(1SATM(S(n)))=\mathscr{L}(SATM(S(n)))$ for $S(n)\geqslant n$. Furthermore, we show that for $\log\log(n)\leqslant S(n)\in o(\log(n))$, $\mathscr{L}(1SUTM(S(n)))$ is incomparable to $\mathscr{L}([1]ATM(S(n)))$, $\mathscr{L}(UTM(S(n)))$, $\mathscr{L}(1MUTM(S(n)))$, and $\mathscr{L}(MUTM(S(n)))$, where $MATM$s are alternating Turing machines with modified acceptance proposed in (Inoue et al. 1989); in contrast, we show that these relationships become proper inclusions when $\log(n)\leqslant S(n)\in o(n)$.

For deterministic synchronized alternating finite automata with at most k processes ($1DSA(k)FA$ and $DSA(k)FA$) we establish a tight hierarchy on the number of processes for the one-way case, namely, $\mathscr{L}(1DSA(n)FA)\subset\mathscr{L}(1DSA(n+1)FA)$ for all $n>0$, and show that $\mathscr{L}(1DFA(2))-\bigcup_{k=1}^{\prime}\mathscr{L}(DSA(k)FA)\neq\emptyset$, where $DFA(k)$ denotes deterministic k-head finite automata. Finally we investigate closure properties under Boolean operations for some of these classes of languages.

1. Introduction

Alternating Turing machines (ATM) were proposed in [1] to model parallel computation. Informally, an alternating Turing machine is a generalization of a

* Research supported in part by NSF Grants DCR89-18409 and DCR90-96221.

non-deterministic Turing machine which can, at some point during a computation, split into several processes working in parallel and independently; an input is accepted iff all parallel processes finish in accepting configurations. The power of alternation on various machine models has since been investigated, and many interesting results have been established in [18, 17, 4, 11, 5, 6, 12, 13, 14, 15, 2]. However, the alternating Turing machine is not a realistic model for real-world parallel computers, because it does not allow any communication among its processes. To remedy the situation, two modifications have been proposed recently to study the effect of communication among processes on the computational power of alternating Turing machines.

The first modification, alternating Turing machines with modified acceptance ($MATM$), was proposed in [10]. In this model, an input is accepted iff the internal states of the leaves of its computation tree form an accepting state set. It was shown that although for any space-bounded function $S(n)$ $\mathscr{L}(MATM(S(n)))=\mathscr{L}(ATM(S(n)))$ and $\mathscr{L}(1MATM(S(n)))=\mathscr{L}(1ATM(S(n)))$, the modified model becomes more powerful when only universal states are allowed and the space bound is small.

The second modification, synchronized alternating Turing machines ($SATM$), was introduced in [7] and investigated in [25, 22 24]. A synchronized alternating machine is an alternating machine with a special subset of internal states called synchronizing states. Each synchronizing state is associated with a synchronizing symbol. If during the course of computation some process enters a synchronizing state, then it has to wait until all other processes enter either an accepting state or a synchronizing state with the same synchronizing symbol. When this happens, all processes are allowed to continue their computation. It was shown in [8] that $SASPACE(S(n))=\bigcup_{c>0} NSPACE(nc^{S(n)})$ for all space-constructible $S(n)$, and hence $SASPACE(S(n))=\bigcup_{c>0} SATIME(c^{S(n)})=\bigcup_{c>0} ATIME(c^{S(n)})$. This result establishes the synchronized alternating Turing machine as the first model known to use space optimally. As another corollary, $NSPACE(n)=\mathscr{L}(2SAFA)$, i.e. two-way synchronized alternating finite automata recognize exactly the class of context-sensitive languages.

The parallel complexity of synchronized alternating finite automata was also investigated in [8], where several results were given for synchronized alternating finite automata bounded in the number of processes ($SA(k)FA$). For example, it was shown that for any $k>0$ $\mathscr{L}([1]SA(k)FA)=\mathscr{L}([1]NFA(k))$ and therefore $\mathscr{L}([1]SA(k)FA)\subset\mathscr{L}([1]SA(k+1)FA)$, where $[1]SA(k)FA$ denotes [one-way] synchronized alternating finite automata with at most k processes, and $[1]NFA(k)$ denotes [one-way] k-head finite automata. It was asked in [8] whether the same relations hold true when these synchronized automata have only universal states ($DSA(k)FA$), i.e.

(i) $\mathscr{L}([1]DSA(k)FA)\subset\mathscr{L}([1]DFA(k))$ for any $k>1$?, and

(ii) $\mathscr{L}(1DSA(k)FA)\subset\mathscr{L}(1DSA(k+1)FA)$ for any $k>1$?

The second question was solved partially in [8], where it was stated that for $k>1$, $\mathscr{L}(1DSA(k-1)FA)\subset\mathscr{L}(1DSA[(\frac{k}{2})+1]FA)$, and the positive answer was given without proof as a corollary in [3]. The first question remained open until now.

In [24] some properties of space-bounded synchronized alternating Turing machines with only universal states were established. In particular, it was shown that $\mathscr{L}(1SUTM(S(n))) \subset \mathscr{L}(1SATM(S(n)))$ for $S(n) \in o(n)$. Furthermore, if $\log(n) \leqslant S(n) \in o(n)$, then $\mathscr{L}(1SUTM(S(n))) \subset \mathscr{L}(SUTM(S(n)))$. As a corollary, $\mathscr{L}(1SUTM(S(n)))$ is not closed under complementation for any $S(n) \in o(n)$.

In this paper we continue the investigation of properties of space-bounded synchronized alternating Turing machines using the powerful Kolmogorov complexity theory as our main tool [19, 20]. First we study the relationships among the four classes of synchronized alternating Turing machines: one-way and with universal states only ($1SUTM$), with universal states only ($SUTM$), one-way (1SATM), and general ($SATM$). We also investigate their relationships with the corresponding plain and modified variants of alternating Turing machines (1UTM, UTM, 1ATM, ATM, 1MUTM, MUTM, 1MATM, MATM). Our main result shows that

(i) $\mathscr{L}(1SUTM(S(n))) \subset \mathscr{L}(SUTM(S(n))) \subset \mathscr{L}(1SATM(S(n))) = \mathscr{L}(SATM(S(n)))$ for $S(n) \in o(n)$,

(ii) $\mathscr{L}(1SUTM(S(n))) = \mathscr{L}(SUTM(S(n))) \subset \mathscr{L}(1SATM(S(n))) = \mathscr{L}(SATM(S(n)))$ for $S(n) \geqslant n$,

(iii) $\mathscr{L}(1SUTM(S(n)))$ is incomparable to $\mathscr{L}(1ATM(S(n)))$, $\mathscr{L}(ATM(S(n)))$, $\mathscr{L}(UTM(S(n)))$, $\mathscr{L}(1MUTM(S(n)))$, and $\mathscr{L}(MUTM(S(n)))$ for $\log\log(n) \leqslant S(n) \in o(\log(n))$, and

(iv) $\mathscr{L}(1SUTM(S(n))) \subset \mathscr{L}(1ATM(S(n)))$, $\mathscr{L}(UTM(S(n)))$, $\mathscr{L}(1MUTM(S(n)))$, and $\mathscr{L}(MUTM(S(n)))$ for $\log(n) \leqslant S(n) \in o(n)$.

Next we prove a more general result to answer positively the first open question mentioned above, namely, $\mathscr{L}(1DFA(2)) - \bigcup_{k=1}^{\infty} \mathscr{L}(DSA(k)FA) \neq \emptyset$. We also give a different proof for the second open question ([3] suggested a solution which appeals to results concerning the hierarchy of multihead automata in [26, 16]). Finally, we show various (non-)closure properties under Boolean operations for some of these classes of languages. For example, we show that $\mathscr{L}(1DSA(k)FA)$ is not closed under complementation, union, or intersection for any $k > 1$; $\mathscr{L}(DSA(k)FA)$ is not closed under complementation for any $k > 1$; $\mathscr{L}(1SATM(S(n)))$ is closed under complementation, intersection, union, concatenation, and Kleene closure for all $S(n)$; and $\mathscr{L}(SUTM(S(n)))$ is not closed under complementation for $S(n) \in o(\log(n))$. We also give some partial answers for some of the remaining separation problems involving different variants of alternating Turing machines.

The rest of this paper is organized as follows. Section 2 gives definitions relating to different variants of alternating Turing machines and to Kolmogorov complexity. Section 3 establishes our main result and gives our solutions to the two open problems mentioned above. Section 4 gives various corollaries and related results, and Section 5 concludes the paper with a discussion of some open problems.

2. Preliminaries

First we give precise definitions of operation and acceptance for synchronized alternating machines. Our definitions use straightforward notions (instantaneous description, $\vdash_M$, configuration tree) and are based on those given in [8] (see also [1].)

Definition 2.1. A synchronized alternating machine (denoted by *SATM*) is a 12-tuple $M=(Q, U, E, S, \Sigma, ¢, \$, \Pi, \Gamma, \delta, q_0, F)$, where

- Σ is a finite *input alphabet*,
- $¢, \$ \notin \Sigma$ are the *left* and *right markers* respectively,
- Γ is a finite *storage tape alphabet* containing the special *blank symbol* B,
- Π is a finite *alphabet of synchronizing symbols*,
- U is the set of *universal states*,
- E is the set of *existential states*,
- $S \subset \{(q, s)\colon q \in U \cup E, s \in \Pi\}$ is the set of *synchronizing states* (*s-states*),
- $Q = U \cup E \cup S$ is a finite set of *states*,
- $q_0 \in Q$ is the *initial state*,
- $F \subseteq Q$ is the set of *accepting states*,
- $\delta \subseteq (Q \times (\Sigma \cup \{¢, \$\}) \times \Gamma) \times (Q \times (\Gamma - \{B\}) \times \{\text{left, stationary, right}\}^2)$ is the *next move relation*.

M has a read-only input tape with the left and right markers ¢ and \$, and one semi-infinite storage tape, initially filled with the blank symbols. M begins in state q_0. A *step* of M consists of reading one symbol from each tape, writing a symbol on the storage tape, moving the input and storage tape heads in specified directions, and entering a new state, according to the next move relation δ.

Definiton 2.2. An *instantaneous description* (ID) of M is an element $(w, p_i, q, z, p_s) \in \Sigma^* \times (N \cup \{0\}) \times Q \times (\Gamma - \{B\})^* \times N$, where w is the content of the input tape (excluding ¢ and \$), p_i is the position of input tape head, q is the internal state, z is the nonblank content of the storage tape, and p_s is the position of storage tape head of M.

An ID is *universal* (*existential, synchronizing, accepting*) depending on the type of its internal state. The *initial* ID of M on input x is $(x, 0, q_0, \varepsilon, 1)$, where ε is the null word.

Definition 2.3. Suppose I_1 and I_2 are two IDs of M and I_2 follows from I_1 in one step according to the next move relation δ. Then we write $I_1 \vdash_M I_2$ and say that I_2 is a *successor* of I_1.

Definition 2.4. The *full configuration tree* of M on an input word w is a (possibly infinite) labeled tree T_w^M such that

(i) each node v is labeled by some ID c_v of M;
(ii) the root is labeled by the initial ID of M on w;
(iii) v_2 is a direct descendant of v_1 iff $c_{v_1} \vdash_M c_{v_2}$.

Each branch of T_w^M is called a *process*.

Definition 2.5. The *synchronizing sequence* (*s-sequence*) *of a node* v in a full configuration tree T with root v_0 is the sequence of synchronizing symbols occurring in labels of the nodes on the path from v_0 to v. Two s-sequences are *compatible* if one is a prefix of the other. If s_1 and s_2 are two compatible s-sequences, and s_2 is longer than s_1, then we use $s_2 - s_1$ to denote their difference.

Definition 2.6. A *computation tree* of M on an input w is a (possibly infinite) subtree T' of the full configuration tree T_w^M such that

(i) each node in T' labeled by a universal ID has the same direct descendants as in T;
(ii) each node in T' labeled by an existential ID has at most one direct descendant;
(iii) for arbitrary nodes v_1 and v_2 in T', the s-sequences of v_1 and v_2 are compatible.

If M on input w has no computation trees, then any subtree of T_w^M that satisfies the first two conditions above must have two processes with incompatible s-sequences. In this case, we say M *deadlocks* on w. The two processes with incompatible s-sequences are called *deadlock processes* and the nonmatching synchronizing states causing the deadlock are called *deadlock states*.

Definition 2.7. An *accepting computation tree* of M on an input w is a finite computation tree of M on w such that each leaf node is labeled by an accepting configuration.

Definition 2.8. The *set of all internal configurations of M on inputs of size n with space-bound $S(n)$* is given by $I_M^n = \{(q, i, w)\colon q \in Q,\ 0 \leqslant i,\ |w| \leqslant |S(n)|,\ w \in \Gamma^{|S(n)|}\}$. The following inequality holds for the size of I_M^n: $|I_M^n| \leqslant |Q| \cdot |S(n)| \cdot |\Gamma|^{S(n)} \leqslant c^{S(n)}$ for some constant c depending only on M.

Next we define alternating Turing machines with modified acceptance. Since there is no risk of ambiguity, we only give an informal definition; the full definition was given in detail in [10].

Definition 2.9. An alternating Turing machine with modified acceptance ($MATM$) is an alternating Turing machine some of whose internal states are called *halting states*, and whose set of accepting states F is replaced by a collection of accepting state sets $C \subset 2^Q$ called *accepting state sets*. An $MATM$ M works in the same way an ATM does, except that if M enters a halting state then it can make no further moves. Acceptance of an input w by M is defined by the set of internal states of the leaves L_w^M of its computation tree: M accepts w iff $L_w^M \in C$.

Definition 2.10. Let M be an alternating Turing machine of any kind. We say M is (*weakly*) *space-bounded by* $S(n)$ if for each input x of length n, if M accepts x, then there is an accepting computation tree of M on x such that the space used by each node is at most $S(n)$.

Definition 2.11. Let $S(n)$, $L(n)$, and $U(n)$ be functions of n. We say $S(n) \in [L(n), U(n)]$ if $S(n) \geq L(n)$ and $S(n) \in o(U(n))$.

Specifically, we are interested in the following classes of machines.

Definition 2.12. We use $[1]DFA(k)$ ($[1]NFA(k)$) to denote [one-way] k-head deterministic (nondeterministic) finite automata.

We call an *SATM* M *deterministic* if it has only universal states. In this case, there is a unique computation tree of M on every input $w \in \{0,1\}^*$.

We use $[1]\{UTM, MUTM, SUTM\}(S(n))$ and $[1]\{ATM, MATM, SATM\}(S(n))$ to denote [one-way] {plain, with modified acceptance, synchronized} alternating Turing machines with universal states only and [one-way] {plain, with modified acceptance, synchronized} alternating Turing machines space-bounded by function $S(n)$, respectively.

We use $[1]DSA(k)FA$ ($[1]SA(k)FA$) to denote [one-way] deterministic (general) synchronized alternating finite automata such that any computation tree of M on any input w has at most k leaves.

We use the symbol $\subset$ to denote proper inclusion for classes of languages.

We now define some important concepts in Kolmogorov (descriptional) complexity theory. Informally, the Kolmogorov complexity of a binary string w measures how succinctly w can be described or coded (using a common coding method for all binary strings.) We follow the approach in [21].

Definition 2.13. A *relative coding scheme* is any computable partial function $F:\{0,1,\#\}^* \to \{0,1,\#\}^*$. We define *relative descriptional complexity* $K_F:\{0,1,\#\}^* \times \{0,1,\#\}^* \to N \cup \{\infty\}$ by

$$K_F(x \mid y) = \min\{|d| : d \in \{0,1\}^* \;\&\; F(d\#y) = x\}.$$

Because there is a universal computable partial function, there is some F_u for which

$$\forall F \exists c_F \forall x, y\,[K_{F_u}(x \mid y) \leq K_F(x \mid y) + c_F]$$

since F_u can simulate any computable function F given the code of a Turing machine M_F for F. We define the *relative descriptional complexity* $K:\{0,1,\#\}^* \times \{0,1,\#\}^* \to N$ by $K(x \mid y) \overset{\text{def}}{=} K_{F_u}(x \mid y)$. We define $K(x) \overset{\text{def}}{=} K(x \mid \varepsilon)$, where ε is the null string.

Definition 2.14. A string $w \in \{0,1\}^*$ is *incompressible* or *Kolmogorov random* (or just *random*) if $K(x) \geq |x|$. Since there are 2^n binary strings of length n but only $2^n - 1$ possible shorter descriptions, there is a random string r of each length. Similarly, for each y, $K(x \mid y) \geq |x|$ holds for some binary string x of each length.

The fact that random strings exist in all lengths gives rise to a powerful proof technique in proving lower bounds, which was popularized by [21]. While combinatorial proofs of lower bounds must usually deal with aggregates of inputs, typical Kolmogorov proofs establish lower bounds by showing that *some* random strings are not really random if the lower bounds fail. Since the proofs only deal with certain random strings, they are usually much shorter and more intuitive.

3. Main results

First, we establish the relationships among different variants of space-bounded synchronized alternating Turing machines. We start out by proving a few lemmas. In the following proofs we say *a process p of an SATM M measures out the length n with an s-sequence* to mean that p enters an s-sequence of length $n+1$ of the form $s_0^n s_1$, where s_0 and s_1 are special s-symbols reserved for this counting process.

The first lemma shows that synchronization cannot be replaced by nondeterminism, modified acceptance, off-line capability and sublogarithmic space *combined.*

Lemma 3.1. *$\mathscr{L}(1DSA(2)FA)-\mathscr{L}(MATM(S(n)))\neq\emptyset$ for any function $S(n)\in o(\log(n))$.*

Proof. Let $L_1=\{x\#x\colon x\in\{0,1\}^+\}$. There is a $1DSA(2)FA$ M to accept L_1 as follows: on input $x\#y$, M splits universally into two processes p_1 and p_2. Process p_2 moves onto the symbol $\#$ and then synchronizes with process p_1 to compare x and y. M accepts $x\#y$ iff $x=y$. Note that M uses only constant space.

On the other hand, it was shown in [10] that $L_1\notin\mathscr{L}(MATM(S(n)))$ for $S(n)\in o(\log(n))$. □

It was shown in [24] that $\mathscr{L}(1SUTM(S(n)))\subset\mathscr{L}(SUTM(S(n)))$ for $S(n)\in[\log(n), o(n)]$. The next lemma shows the same relation holds for $S(n)\in o(\log(n))$.

Lemma 3.2. *$\mathscr{L}(DSA(2)FA)-\mathscr{L}(1SUTM(S(n)))\neq\emptyset$ for any function $S(n)\in o(\log(n))$.*

Proof. Let $L_2=\{x\in\{0,1\}^+\colon x \text{ is a palindrome}\}$. There is an $DSA(2)FA$ M that accepts L_2 as follows: on input x, M splits universally into two processes p_1 and p_2. Process p_2 moves onto the right marker \$ and then synchronizes with process p_1 to compare x and x^R. M accepts x iff x is a palindrome. M uses only constant space.

Now suppose there is some $1SUTM(S(n))$ N that accepts L_2 where $S(n)\in o(\log(n))$. Let x be a random string, $n=2\cdot|x|$, and consider the computation tree of N on xx^R with its processes labeled in some fixed order. We say that processes p and q are *in the same class i* if at the time their input heads move off the prefix x they have the same internal configuration i. Let k_i and l_i be the largest and smallest numbers of s-states that processes in class i enter while reading x. We say that process p in class i is *representative* if it enters k_i s-states while reading x. Let T_i denote the ordered list of all s-states a process p in class i will enter while reading the right marker \$. It is easy to see that the following assertions hold for any processes p and q of N on xx^R:

(i) if p and q are in the same class and both are representative, then the s-sequences they generate on xx^R are the same;

(ii) suppose p and q are in the same class i, and p is representative whereas q is not. Denote the difference of the two s-sequences by the string of s-symbols d. Then the s-sequence that process p will make on reading the remaining input string x^R has the form $d^f e$, where e is a prefix of d and f is some integer. Let d_i denote the longest such string for class i, ($|d_i| = k_i - l_i$) and e_i and f_i be the corresponding values.

Next, we note that if two processes p_1 and p_2 of N deadlock on xw for some string w of length $|x|$, then one deadlock process must have reached the right marker \$, else N rejects the string xww^Rx^R. Furthermore, it is easy to see that either both p_1 and p_2 are representative of their classes, or p_1 and p_2 are in the same class and one of them is representative.

Let $C = \{(i, k_i, l_i, T_i, |d_i|, |e_i|)\colon\ i \in I_X^n\}$. Then the following program P uses C to determine x:

```
let C be defined as above
for all $w \in \{0, 1\}^+$ of length $|x|$ do
  for each $i \in I_X^n$ do
    use C to continue the computation of N on xw starting at w for a
      representative process in class i; also update its counter of s-states
    if $l_i < k_i$ generate $d_i$ and $e_i$ (and from those the correct s-sequence
      of the representative process of class i of N on reading $x^R$)
  endfor
  print $w^R$ if all representative processes end up in accepting states and
    no deadlock occurs (this can be checked using $k_i$'s, $T_i$'s, $d_i$'s and $e_i$'s)
endfor
```

We show the correctness of P. If $w^R = x$ then all representative processes end up in accepting states and there is no deadlock, so P prints x. Conversely, if P prints w^R, then all representative processes of N on xw finish in accepting states, and there are no deadlocks, because P makes sure that no two representative processes deadlock with each other (using k_i and T_i's), and that no two processes in the same class deadlock with each other (using d_i's, e_i's, k_i's, and T_i's to verify the s-sequence generated by each representative process). Hence, xw is a palindrome, i.e. $w^R = x$.

We have $|P| = c + \log(|x|) + |C|$, where c is some constant. C contains at most $|I_X^n| = h_1^{S(n)}$ vectors, and the size of each vector is at most $(h_2 \cdot S(n) + h_3 \cdot \log(n) + h_4 \cdot S(n) + h_5^{S(n)})$, where h_1, h_2, h_3, h_4, h_5 are all constants, so $|C| \leqslant h_6^{2S(n)}$ for some constant h_6. Since $S(n) \in o(\log(n))$, for sufficiently large x we have $|P| < |x|$. This contradicts the randomness of x.

Remark. L_2 can be accepted by a 1*SAFA* N' as follows: on input w, N' splits universally into two processes p_1 and p_2. Process p_1 makes sure that the first and the last symbols are the same, and in the process measures out the length of w with an s-sequence. Using this s-sequence as the yardstick, process p_2 then compares the corresponding bits of the rest of the string w. For example, to compare the ith bit, p_2 measures out the length $2\cdot(i-1)$, reads the ith bit, and guesses the position of the corresponding bit to verify that they are the same, marking out the length $|w|-2\cdot(i-1)$ in the process. At some point p_2 nondeterministically decides that it has finished comparing and quits.

This is not surprising, because later on we will show that for *SATM*s bounded in space by any function $S(n)$ the one-way and two-way models are equivalent.

The next lemma improves Lemma 3.2 by showing that the off-line capability cannot be replaced by synchronization for any $S(n)\in[\log\log(n), \log(n)|$.

Lemma 3.3. *$\mathscr{L}(UTM(S(n)))-\mathscr{L}(1SUTM(S(n)))\neq\emptyset$ for any function $S(n)\in[\log\log(n), \log(n)|$.*

Proof. Let $L_3=\{B(n)\natural x\#y\colon\ B(n)=\text{bin}(1)\#\text{bin}(2)\#\cdots\#\text{bin}(n)\ \ \&\ \ |x|=\log(n)\ \&\ x\neq y\}$, where bin$(i)$ is the shortest binary representation of the integer i. That $L_3\in\mathscr{L}(UTM(\log\log(n)))$ was shown in [14]. We show that L_3 cannot be accepted by any $1SUTM(S(n))$ for $S(n)\in o(\log(n))$.

Suppose L_3 is accepted by some $1SUTM(S(n))$ M with $S(n)\in[\log\log(n), \log(n)|$. Let x be a random string of length $\log(n)$ relative to $B(n)\natural$, i.e. $K(x|B(n)\natural)\geqslant|x|$, and consider the computation tree of M on $y=B(n)\natural x\#x$. In general if M deadlocks on $B(n)\natural x\#x$, then one deadlock process must have read the right marker \$, else M rejects $B(n)\natural x\#xx$. Since $B(n)\natural x\#x\notin L_3$, there are three cases: there must be a process that loops or finishes in a nonaccepting state, or two processes that deadlock with each other.

In the first two cases, let $C=(1, i)$, where $i\in I_N^{|y|}$ is the internal configuration of a process p_1 of N right after it finishes reading $B(n)\natural x\#$, and one of whose children either loops or ends up in a nonaccepting state. In the third case, there are two processes that deadlock with each other, p_1 and p_2. One of them must deadlock while reading the right marker \$, say p_2, without loss of generality. If p_1 deadlocks before it finishes reading the prefix $B(n)\natural x\#$ with s-state s_1 then let $C=(2, i_2, k_2, s_1)$, where i_2 is the internal configuration of p_2 right after it finishes reading $B(n)\natural x\#$, k_2 is the number of s-states p_2 will enter before it deadlocks with p_1. Else if p_1 and p_2 deadlock after they finish reading $B(n)\natural x\#$, let $C=(3, i_1, k_1, i_2, k_2)$, where i_1, i_2 are the internal configurations of p_1, p_2 right after they finish reading $B(n)\natural x\#$, k_1, k_2 are the number of s-states p_1, p_2 will, respectively, enter before they deadlock.

Then the following program P uses C to determine x:

```
let C be defined as above
for all $w\in\{0,1\}^+$ of length $|x|=\log(n)$ do
  use C to continue simulating faithfully the computation of M
    on $B(n)\natural x\#w$ starting at $w$ for processes $p_1$ and/or $p_2$
    print w if some configuration rejects, loops or
      some deadlock occurs (this can be checked using $k_1$ and $k_2$)
endfor
```

P prints w iff there is a process which loops, or ends in a nonaccepting state, or two processes that deadlock with each other, iff $w = x$. Not counting the string $B(n)\natural$, we have $|P| \leqslant c + \log\log(n) + |C|$ for some constant c, and $|C| \leqslant d_1 \cdot S(|w|) \leqslant d_2 \cdot S(n)$ for some constants d_1 and d_2. For sufficiently large x, we have $|P| < \log(n) = |x|$ since $S(n) \in o(\log(n))$. But then $K(x|B(n)\natural) < |x|$, and this contradicts the randomness of x.

It was shown in [24] that $\mathscr{L}(1SUTM(S(n))) \subset \mathscr{L}(1SATM(S(n)))$ for $S(n) \in o(n)$. The next lemma improves this result for $S(n) \in o(\log(n))$.

Lemma 3.4. $\mathscr{L}(1SA(2)FA) - \mathscr{L}(SUTM(S(n))) \neq \emptyset$ *for any function* $S(n) \in o(\log(n))$.

Proof. Let $L_4 = \{x \# y: x, y \in \{0, 1\}^* \ \& \ x \neq y\}$. There is a $1SA(2)FA$ M that accepts L_4 as follows: on input $x \# y$, M first guesses whether x and y have different lengths or the kth symbols of x and y differ for some $k \leqslant \min\{|x|, |y|\}$. In the first case where it decides x and y have different lengths, M first splits universally into two processes p_1 and p_2. Process p_1 deterministically measures out the length of x with an s-sequence. Process p_2 first guesses that y is shorter (longer) than x and then verifies its guess by measuring out the same s-sequence as p_1 does and noting that y has less (more) symbols than the s-sequence.

In the second case where it decides x and y differ at the kth symbols for some $k \leqslant \min\{|x|, |y|\}$, M again splits universally into two processes p_1 and p_2. Process p_1 picks a symbol of x, and process p_2 picks a symbol of y, and then both verify that they pick two symbols of the same position by measuring out the same s-sequence. Finally, p_1 and p_2 verify that the two symbols are different by guessing each other's symbol. Note that M is one-way and uses only constant space.

Suppose L_4 is accepted by some $SUTM(S(n))$ N, where $S(n) \in o(\log(n))$. We assume, without loss of generality, that N only halts on the right marker \$. Let x be a random string, $n = 2|x| + 1$, and consider the computation tree of N on $x \# x$. Since $x \# x \notin L_4$, there are three cases: there is some process p_1 of N on $x \# x$ that halts in a nonaccepting state, that loops, or there are two processes p_1 and p_2 that deadlock with each other.

In the first case, where there is a nonaccepting process p, let c_p be the internal configuration of p when it first moves right from the prefix $x\#$, and let $E = (1, c_p)$. During the computation, p visits the prefix $x\#$ at most $|I_N^n| = c^{S(n)}$ times for some constant c. Let C be the set of all triples (k, i_k, o_k), where i_k and o_k are the internal configurations of p when it enters and exits $x\#$ in the kth visit.

In the second case, where there is a looping process p, if p never exits from the prefix $x \# x$ then let $E = (2, -1)$; otherwise let $E = (2, c_p)$, where c_p is defined as in the previous case. During the computation, p visits the prefix $x\#$ at most $|I_N^n|=c^{S(n)}$ times, for some constant c, before looping occurs. Let C be the set of the first $c^{S(n)}$ triples (k, i_k, o_k), where i_k and o_k are the internal configurations of p when it enters and exits $x\#$ in the kth visit. If p never exits from $x\#$ after the kth visit then let $o_k=-1$.

In the last case there are two processes p_1 and p_2 that deadlock with each other. Let C be the set of all tuples $(k, i_k, o_k, t_k, I_k, O_k, T_k)$ where i_k and o_k are the internal configurations of p_1 when it enters and exits $x\#$ in the kth visit, and t_k is the number of s-states p_1 enters during this period. Similarly, I_k, O_k, and T_k are the corresponding data for p_2. Again, there can be at most $c^{S(n)}$ such tuples.

Next, let c_1 and c_2 be the internal configurations of p_1 and p_2 when they first move right from $x\#$, and d_1 and d_2 be the numbers of s-states they have entered at that point. Let m be the number of s-states p_1 and p_2 enter before deadlocking. If p_1 deadlocks while it is reading the prefix $x\#$, then let l_1 be the number of s-states p_1 enters since the last time it crosses the symbol $\#$ (or since the beginning if p_1 has not crossed $\#$) before it deadlocks and s_1 be the deadlock s-state of p_1; else, let $l_1=-1$ and $s_1=-1$. Let l_2 be defined in a similar fashion, and let $E=(3, m, l_1, s_1, l_2, s_2, c_1, d_1, c_2, d_2)$.

Then the following program P uses C and E to determine x:

```
let C and E be defined as above
for all w of length $|x| \in \{0,1\}^+$ do
    use C and E to continue simulating faithfully p (or $p_1$ and $p_2$) on $x\#w$,
        starting at w, and updating the counters of s-states in the process
    print w if some configuration rejects, loops or
        some deadlock occurs (this can be checked using E and C)
endfor
```

We show the correctness of P. If $w=x$ then N rejects $x\#w$ and so P prints w, because P is able to detect a looping process, a nonaccepting process, or two processes that deadlock with each other. Conversely, if P prints w then N rejects $x\#w$, so $w=x$.

We have $|P|=c+\log(|x|)+|C|+|E|$, where c is some constant. There can be at most $e_1^{S(n)}$ elements in C, and the size of each element is at most $e_1^{S(n)}\cdot(e_2\cdot S(n)+\log(n))$, so $|C|\leqslant e_3^{3\cdot S(n)}$, where e_1, e_2, and e_3 are constants. We also have $|E|\leqslant(e_4\cdot S(n)+e_5\cdot\log(n))$, where e_4 and e_5 are constants. Since $S(n)\in[\log\log(n), \log(n)]$, for sufficiently large x, we have $|P|<|x|$. This contradicts the randomness of x. □

The next lemma shows that when the space bound is at least $\log(n)$, synchronization does not increase the power of one-way alternating Turing machines with only universal states. In contrast, modified acceptance *does* increase the power of this class [10].

Lemma 3.5. $\mathscr{L}(1SUTM(S(n)))=\mathscr{L}(1UTM(S(n)))$ *for any* $S(n)\geqslant\log(n)$.

Proof. Given an $1SUTM(S(n))$ M we construct a $1UTM(S(n))$ M' to accept the same language as follows. First we describe the simulation for the case when the computation tree of M on its input x has only two processes. M' stores the internal configurations of both processes p_1 and p_2 of M in the worktape of a process p', and it keeps track of the number of s-states each process has entered with a counter. Process p' must simulate both p_1 and p_2 one step at a time, and it will move the input head only when both p_1 and p_2 are finished with the current input square. Suppose during the simulation process p_1 enters an s-state, and the counts of s-states entered so far are k_1 for p_1 and k_2 for p_2. If $k_1 < k_2$ then p' continues the simulation. Else if $k_1 \geq k_2$ then p' spawns off a child process r' to make sure that the k_1th s-state that p_2 enters has the same s-symbol. If they are not the same, r' rejects. A symmetric procedure is carried out if p_2 enters an s-state. It is easy to see that M' accepts the same language as M does, and that M' uses the same amount of space, since the counters take at most $S(n)$ space.

We now generalize the simulation given above to the general case, where an ID of M may have many direct descendants. Again, every process p' of M' is used to simulate two processes of M. To do that, each p' stores in its worktape the internal configurations of the two processes p_1 and p_2 of M that it is to simulate, along with two counters to keep track of the numbers of s-states these processes have entered so far. During the simulation p' will move its input head only if both p_1 and p_2 are finished with the current input square, and p' follows the procedure described above to make sure that p_1 and p_2 will not deadlock with each other. For the sake of uniformity, we assume that initially M' simulates two copies of the initial process of M.

Suppose during the simulation, process p_1 enters a universal state and splits into d descendant processes, where d is bounded by a constant depending only on M. Then p' will split into $\binom{d+1}{2}$ descendant processes, each simulating a pair of processes chosen from p_2 and the descendants of p_1. Finally, suppose a process r' of M' is used to verify the kth s-symbol of some process r of M, and during the process it finds that process r enters a universal state and splits into d descendants. Then r' also splits into d descendants to verify the kth s-symbol of each descendant of r.

It is easy to see that M' accepts its input $x \Leftrightarrow$ every process of M finishes in an accepting state, and there are no deadlocks $\Leftrightarrow$ M accepts x. Also, M' uses the same amount of space as M does because the counters take up at most $S(n)$ space.

When off-line capability is present, the situation is slightly different. In the next three lemmas we show that when the space bound is at least $\log(n)$, neither synchronization nor modified acceptance add to the computational power of two-way alternating Turing machines with only universal states.

Lemma 3.6. *$\mathscr{L}(SUTM(S(n))) = \mathscr{L}(UTM(S(n)))$ for any $S(n) \geq \log(n)$.*

Proof. We modify the simulation technique given in [24] to prove the lemma. Given an SUTM$(S(n))$ M where $S(n) \geq \log(n)$, we construct a $UTM(S(n))$ M' to accept the same language as follows. On input w, M' simulates each process of M' with a process of its own. When the current internal state of some process p of M is an s-state, the corresponding process p' of M' spawns off a process c whose worktape contains the s-symbol associated with the s-state and the number of s-states p has entered so far. Since each process makes at most $d^{S(n)}$ moves, d is a constant, and $S(n) \geq \log(n)$, there is enough space to store them. Process c restarts the computation of M on w and verifies that the corresponding s-symbols in other processes match with the one stored on its worktape. If a discrepancy occurs, M' rejects. It is easy to see that M and M' accept the same language. □

To obtain the same result for $MUTM$s, we first show that $MUTM(S(n))$ is closed under complementation for $S(n) \geq \log(n)$.

Lemma 3.7. *$\mathscr{L}(MUTM(S(n)))$ is closed under complementation for any $S(n) \geq \log(n)$.*

Proof. Given a $MUTM(S(n))$ M where $S(n) \geq \log(n)$, we construct a $MUTM(S(n))$ M' to accept the complement language using the technique used in [9] to show nondeterministic space is closed under complementation.

The idea is to cycle through all configurations to find the set of internal states of the leaves of the computation tree of M on some input w. Since there are at most $d^{S(n)}$ configurations, d is a constant, and $S(n) \geq \log(n)$, there is enough space to do this. Let $R(n)$ denote the number of configurations reachable from the initial configuration in at most n steps. We use two counters and $R(1)$ to identify all leaf configurations in the computation tree of M on w as follows:

```
let LEAVES = $\emptyset$
let $R(1)$ = the number of configurations reachable in one step from
  the initial configuration
repeat until $R(n) = R(n+1)$
  for each configuration $c_1$ in counter 1 do
    for each configuration $c_2$ in counter 2 do
      branch
        a) do nothing
        b) verify that $c_2$ is reachable from the initial configuration in
          at most $n$ steps (with more branching). For each branch:
            if it is not halt in state $q_R$
            else if $c_1$ is reachable from $c_2$ in at most one step then
              increment $R(n+1)$
            if $c_1$ is a halting configuration then
              add its state to LEAVES
            endif
        endbranch
      endfor
      if not all $R(n)$ configurations have been found then enter state $q_R$
    endfor
endrepeat
enter state $q_A$ iff LEAVES is not an accepting state set
```

M' has only two halting states: q_A and q_R, and the accepting state sets of M' are $\{q_A\}$ and $\{q_A, q_R\}$. It is easy to see that M' uses as much space as M does, and that $L(M') = \overline{L(M)}$.

Corollary 3.8. $\mathcal{L}(MUTM(S(n))) = \mathcal{L}(UTM(S(n)))$ *for any* $S(n) \geq \log(n)$.

Proof. Given a $MUTM(S(n))$ M where $S(n) \geq \log(n)$, we show how to construct a $UTM(S(n))$ M' which accepts $L(M)$. By the proof of Lemma 3.7 there is some $MUTM(S(n))$ N that accepts $\overline{L(M)}$ with only 2 halting states q_A and q_R, and whose accepting state sets are $\{q_A\}$ and $\{q_A, q_R\}$. Now let N' be a $UTM(S(n))$ obtained from the $MUTM(S(n))$ N by defining the set of accepting states to be $\{q_R\}$. It is clear that N' uses only $S(n)$ space, and that $x \in L(M) \Leftrightarrow x$ is accepted by $M \Leftrightarrow x$ is rejected by $N \Leftrightarrow$ the internal states of all leaves of the computation tree of N on x are $q_R \Leftrightarrow N'$ accepts x.

In the next two lemmas we sharpen a main result in [8] which states that $\bigcup_{c>0} \mathcal{L}(NTM(n \cdot c^{S(n)})) = \mathcal{L}(SATM(S(n)))$ for any space-constructible function $S(n)$. We will modify the proofs leading to this result to remove the requirement of space-constructibility for $S(n)$ and the off-line capability of the $SATM$s.

Lemma 3.9. $\bigcup_{c>0} \mathcal{L}(NTM(n \cdot c^{S(n)})) = \mathcal{L}(1SATM(S(n)))$ *for any function* $S(n)$.

Proof. We will show that $\mathcal{L}(SATM(S(n))) \subseteq \bigcup_{c>0} \mathcal{L}(NTM(n \cdot c^{S(n)})) \subseteq \mathcal{L}(1SATM(S(n)))$ for any function $S(n)$. The first ($\subseteq$) relation was shown in [8, Lemma 3.1] for space-constructible $S(n)$. We reproduce the proof below and show how to remove the space-constructibility requirement. Given an $1SATM(S(n))$ M, we can construct an NTM M' to simulate M by doing a breadth-first-like traversal of the computation tree of M on its input w of size n. Each process of M is simulated until it enters an s-state; M' will compare the corresponding s-states to make sure that no deadlock occurs before continuing the simulation. Since there are at most $n \cdot d^{S(n)}$ distinct configurations of M on an input w of size n, M' needs at most $n \cdot c^{S(n)}$ space, for some constants d and c, at any time to maintain the current IDs of all processes of M on w. It is easy to see that the space-constructibility requirement is not necessary: M' keeps track of the length l of the longest worktape used during the simulation, and at the end of the simulation uses up $n \cdot e^l$ squares of tape. Then on any input w of length n, M uses at most $S(n)$ space iff M' uses at most $n \cdot e^{S(n)}$ space.

The second ($\subseteq$) relation was shown in [8, Lemma 3.2] for space-constructible $S(n)$ and for off-line $SATM$s. We reproduce the proof here and show how to remove both conditions. First assume that $S(n)$ is space-constructible. Given an $NTM(n \cdot c^{S(n)})$ M, we construct a $1SATM(S(n))$ M' to simulate M as follows. On some input w of length n, M' first splits into $n+4$ processes $A, B, D_0, D_1, \ldots, D_{n+1}$ in such a way that the head of D_i is on the ith position of the input tape and both A and B have their input heads at the first position. Since $S(n)$ is space-constructible, we assume that each of these $n+4$ processes has the word $0^{S(n)}$ in its worktape. Each process D_i then splits into $c^{S(n)}$ copies $D_i^1, \ldots, D_i^{c^{S(n)}}$. The significance of these processes are as follows: M' uses A to represent the position of the input head and its state, each D_i^j to represent the $(c^{S(n)} \cdot (i-1)+j)$th square of the worktape, and B to represent the position of the worktape head (suppose the input head of process B is on hth square and its worktape contains the number $0 \leqslant m \leqslant c^{S(n)}$; then the worktape head of M is at position $(h-1) \cdot c^{S(n)} + m$).

To simulate a step of M, M' performs the following steps: first M' synchronizes all of its processes with a special s-symbol S_0. Then some process D_i^j has to decide that the worktape head is on the square it represents. To do that, B spawns a copy B' to measure out deterministically the lengths h and m with two s-sequences. Each process D_i^j decides either to verify that $(h, m)=(i, j)$ or to verify that $(h, m) \neq (i, j)$ by spawning a copy $(D')_i^j$ and the technique described in Lemma 3.4. All other processes guess along with process B. M' concludes this phase with a special s-symbol S_1. At the end of this phase, exactly one unique process D_k^l satisfies the relations $(h, m)=(k, l)$.

In the next phase, processes B, A, and D_k^l synchronize among themselves (by guessing each other's symbol) to determine the next move of M. All other processes guess along with them. Next A updates its state and the position of its input head to reflect the change of the input head position and state of M; process B updates its input head position and worktape content to reflect the change in the position of the worktape head of M; process D_k^l updates the symbol at the square it represents. M' ends this phase with a special s-symbol S_2.

In the last phase, if process A is in a final state of M, A deterministically produces the special s-symbol S_3 and stops; else it deterministically produces the s-symbol S_0 to restart the process. Other processes guess along with A. This concludes the description of the simulation of M by M'.

Now we show how to make M' an on-line $SATM$. We note that only processes A and B need the off-line capability; all processes D_i^j stay stationary throughout the simulation, and their copies $(D')_i^j$ move only to the right. First we show that B is not necessary, and then we show how to replace A with n on-line processes $E_1, \ldots, E_n$.

The position of the worktape head can be maintained by marking the state of the process D_i^j representing the square the head is currently on. Initially, the state of process D_1^0 is marked. To move the head, process D_i^j needs only to identify its successor $D_{i'}^{j'}$ by entering two s-sequences of lengths i' and j'. Only one unique process

$D_i^{j'}$ can match this sequence, so the position of the worktape head will be correctly maintained at all times. Hence, process B can be removed. Note also that processes D_i^j are still on-line processes. Similarly, the position of the input tape head of M can be maintained with n on-line processes $E_1, \ldots, E_n$, so process A can be removed also.

Now we show how to remove the requirement of space-constructibility. Before spawning off the processes D_i^j's, M' guesses the amount of worktape of M needed by input w and marks the amount on the worktapes of all of its children processes. If during the computation, M attempts to use more space than allowed then M' rejects. Clearly, M' accepts w within space $S(n)$ iff M accepts w within space $(n \cdot c^{S(n)})$. This removes the requirement of space-constructibility for $S(n)$. □

We are now ready to establish the relationships among different variants of synchronized alternating Turing machines.

Theorem 3.10. *$\mathscr{L}(1SATM(S(n))) = \mathscr{L}(SATM(S(n)))$ for all functions $S(n)$.*

Proof. Follows immediately from Lemma 3.9 and [8]. □

Corollary 3.11. *$\mathscr{L}(SUTM(S(n))) \subset \mathscr{L}(1SATM(S(n)))$ for any $S(n) \geq \log(n)$.*

Proof. Follows from the fact that $\mathscr{L}(UTM(S(n))) = \mathscr{L}(NTM(S(n)))$ for $S(n) \geq \log(n)$, Lemmas 3.6 and 3.9.

Theorem 3.12. (i) *$\mathscr{L}(1SUTM(S(n))) \subset \mathscr{L}(SUTM(S(n))) \subset \mathscr{L}(1SATM(S(n))) = \mathscr{L}(SATM(S(n)))$ for any function $S(n) \in o(n)$.*

(ii) *$\mathscr{L}(1SUTM(S(n))) = \mathscr{L}(SUTM(S(n))) \subset \mathscr{L}(1SATM(S(n))) = \mathscr{L}(SATM(S(n)))$ for any $S(n) \geq n$.*

Proof. The relations between $\mathscr{L}(1SUTM(S(n)))$ and $\mathscr{L}(SUTM(S(n)))$ follow from Lemma 3.2 and a result in [24]. The relation between $\mathscr{L}(SUTM(S(n)))$ and $\mathscr{L}(1SATM(S(n)))$ follows from Lemma 3.4 and Corollary 3.11, and the relation between $\mathscr{L}(1SATM(S(n)))$ and $\mathscr{L}(SATM(S(n)))$ follows from Theorem 3.10.

Considering the results of the lemmas above, we expect no surprise from the next theorem.

Theorem 3.13. (i) *$\mathscr{L}(1SUTM(S(n)))$ is incomparable to $\mathscr{L}(1ATM(S(n)))$, $\mathscr{L}(ATM(S(n)))$, $\mathscr{L}(UTM(S(n)))$, $\mathscr{L}(1MUTM(S(n)))$, and $\mathscr{L}(MUTM(S(n)))$ for $S(n) \in [\log\log(n), \log(n)|$.*

(ii) *$\mathscr{L}(1SUTM(S(n))) \subset \mathscr{L}([1]ATM(S(n)))$, $\mathscr{L}(UTM(S(n)))$, $\mathscr{L}(1MUTM(S(n)))$, and $\mathscr{L}(MUTM(S(n)))$ for $S(n) \in [\log(n), n|$.*

Proof. The first part follows from Lemmas 3.1, 3.3, and the fact that $L_3 \in \mathscr{L}(1MUTM(S(n)))$ for $S(n) \geqslant \log\log(n)$. The second part follows from Lemma 3.5, [10] and Corollary 3.8. □

Below we give our solutions to the open problems posed in [8]. The first result shows that for all $k>0$ $\mathscr{L}([1]DSA(k)FA) \subset \mathscr{L}([1]DFA(k))$. We begin by stating a corollary to Lemma 3.4.

Corollary 3.14. $\mathscr{L}(1DFA(2)) - \bigcup_{k=1}^{\infty} \mathscr{L}(DSA(k)FA) \neq \emptyset$.

Proof. Follows from Lemma 3.4 and the fact that there is a $1DFA(2)$ M that accepts L_4: on input $x \# y$, one head of M moves onto the # symbol and then works in tandem with the other head to compare x and y. M accepts iff x and y differ in one position or their lengths are different. M rejects its input if no # is found. □

Theorem 3.15. $\mathscr{L}([1]DSA(k)FA) \subset \mathscr{L}([1]DFA(k))$ *for all* $k>1$.

Proof. Follows immediately from Corollary 3.14. □

The second result establishes a tight hierarchy on the number of processes for $1DSA(k)FA$s. We need a few terminologies for our proofs.

Definition 3.16. For $m>0$, $n \geqslant 0$, let

$$L_{m,n} = \{w_1 \# w_2 \# \cdots \# w_m \# w_m \# \cdots \# w_2 \# w_1 \#^n \colon w_i\text{'s} \in \{0,1\}^*\}.$$

Definition 3.17. Suppose M is a $1DSA(k)FA$ which accepts $L_{m,n}$ for some $m>1$, $n \geqslant 0$, $k>0$. Let w be in $L_{m,n}$ and v be a prefix of w, and p be a process in the computation tree of M on w. We use

(i) $SS_{p,v}$ to denote the s-sequence of process p after reading the prefix v,
(ii) SS_v to denote the longest s-sequence among all processes of M after reading v,
(iii) ss_v to denote the shortest s-sequence among all processes of M after reading v.

Also, we call $SS_v - SS_{p,v}$ the *expected s-sequence of process p after reading v*. In other words, to avoid deadlocks, process p must enter these s-states after reading v.

Lemma 3.18. $L_{1,n} \in \mathscr{L}(1DSA(2)FA) - \mathscr{L}(1DSA(1)FA)$ *for all* $n \geqslant 0$.

Proof. By definition, $\mathscr{L}(1DSA(1)FA)$ is just the class of regular languages. It is evident that $L_{1,n}$ are not regular. □

Lemma 3.19. *Suppose* $L_{m,n}$ *is accepted by some* $1DSA(m)FA$ M, $m>1$, $n \geqslant 0$. *Then there is a random string* w_1 *and a process p of M on* $w_1 \#$ *such that for all* $v \in L_{m-1,0}$, *process p on input* $w_1 \# v \#$ *enters no more than d s-states, where d is some constant depending only on* w_1 *and M.*

Proof. Let w_1 be random and let $SS_{p,w_1\#} = ss_{w_1\#}$. Then for all $v \in L_{m-1,0}$, process p on input $w_1\#v\#$ can enter no more than $SS_{w_1\#}$ s-states, or else, the following program P can be used to determine w_1:

```
let $Q=((q_1,l_1),(q_2,l_2),\ldots,(q_m,l_m))$ where
  $q_i$ denotes the state of process $i$ after reading $w_1\#$ and $l_i$ the length of $SS_{i,w_1\#}$
for all $v$ in $L_{m-1,0}$ lexicographically do
  simulate $M$ on $v\#$ with initial states $q_1,q_2,\ldots,q_k$
  until for some $v_0$, $|SS_{p,w_1\#v_0\#}|=SS_{w_1\#}$
endfor
for all $y$ lexicographically do
  simulate $M$ on $\#^{2m-2}y\#^n$ with initial states $q_1,q_2,\ldots,q_k$
  print $y$ iff there is an accepting computation tree of
    $M$ on $x\#^{2m-1}y\#^n$ (this can be checked using $SS_{p,w_1\#v_0\#}$ and $l_1,l_2,\ldots,l_m$)
endfor
```

Clearly, P prints y iff $y=w_1$. We have $|P| \leqslant c_1 \cdot \log(|w_1|)+c_2$ for some constants c_1 and c_2. But then for sufficiently large random w_1, we have $|P'|<|w_1|$. This contradicts the randomness of w_1. Hence there is a random string w_1, a constant d depending only on w_1 and M, and a process p of M such that for all v in $L_{m-1,0}$, $|SS_{p,w_1\#v\#}| \leqslant d$.

Lemma 3.20. *Suppose there is some 1DSA(m)FA M that accepts $L_{m,n}$, $m>1$, $n \geqslant 0$. Then there is some 1DSA(m − 1)FA M' that accepts $L_{m-1,n+k}$, $k \geqslant 0$ is some constant.*

Proof. Suppose M is a 1DSA(m)FA that accepts $L_{m,n}$ for some $m>1$, $n \geqslant 0$. Then by Lemma 3.19, there is a random string w_1, a constant d depending only on w_1 and M, and a process p of M, such that for any $v \in L_{m-1,0}$, $|SS_{p,w_1\#v\#}| \leqslant d$. Without loss of generality, let $p=1$.

We construct a 1DSA(m − 1)FA M' to accept $L_{m-1,n+|w_1|}$ as follows. Let $(p_1,p_2,\ldots,p_m)$ be states of the m processes of M after reading $w_1\#$. Some of them might be null, if the corresponding processes do not exist at the time. M' on input $v \in L_{m-1,n+|w+1|}$ branches into $m-1$ processes with initial states $([p_1,p_2],p_3,\ldots,p_m)$. Note that the first two processes of M are combined into one.

M' continues to simulate M on input $v\#^{n+|w_1|}$, except for some minor modifications:

(i) for $i>1$, process i of M' refrains from entering the next $|SS_{w_1\#}-SS_{i,w_1\#}|$ s-states. Instead, it checks to see whether they form the expected s-sequence for process $i+1$ of M and, if so, it enters some equivalent but nonsynchronizing states. Else, process i rejects the input. Once the expected s-sequence has been confirmed, process i will resume entering s-states.

(ii) The first (combined) process of M' will simulate process 2 of M in the manner described above. It will also verify that process 1 is not involved in a deadlock in the computation of M on $w_1\#v\#w_1\#^n$. Because process 1 of M on

$w_1 \# v \# w_1 \#^n$ does not enter more than $d+e\cdot(|w_1|+n)$ s-states (e is some constant depending only on M), it suffices to keep track of the first $d+e\cdot(|w_1|+n)$ s-states of the other processes of M. This can be done with constant space.

(iii) M' treats the first $|w_1|$ $\#$'s that comes after v as w_1.

Clearly, $v\#^{n+|w_1|}\in L_{m-1,n+|w_1|}$ iff $w_1 \# v \# w_1 \#^n \in L_{m,n}$, iff there is an accepting computation of M on $w_1 \# v \# w_1 \#^n$ iff M' accepts $v\#^{n+|w_1|}$. □

Theorem 3.21. *$L_{m,n}\in \mathscr{L}(1DSA(m+1)FA)-\mathscr{L}(1DSA(m)FA)$ for all $m>0$, $n\geqslant 0$.*

Proof. We use induction on m. The theorem is true when $m=1$ because of Lemma 3.18. Suppose $L_{l-1,n}\in\mathscr{L}(1DSA(l)FA)-\mathscr{L}(1DSA(l-1)FA)$ for some $l>1$ and all $n\geqslant 0$. We show $L_{l,n}\in\mathscr{L}(1DSA(l+1)FA)-\mathscr{L}(1DSA(l)FA)$ for all $n\geqslant 0$.

Suppose for some n, $L_{l,n}\in\mathscr{L}(1DSA(l)FA)$. Then by Lemma 3.20, $L_{l-1,n+k}\in\mathscr{L}(1DSA(l-1)FA)$ for some $k\geqslant 0$. This contradicts the induction hypothesis. On the other hand, the following $1DSA(m+1)FA$ M accepts $L_{m,n}$ in a straightforward manner: a process p_1 of M sequentially outputs as its s-sequence the first m segments of the input delimited by two $\#$'s, or a $\#$ and the left endmarker; at the end of each segment, p_1 spawns off a process which seeks and outputs the corresponding segment as its s-sequence. Also p_1 verifies that the input contains exactly $(2m-1+n)$ $\#$ symbols. It is clear that M accepts x iff $x\in L_{m,n}$. Hence the theorem holds for all $m>0$. □

4. Related results and corollaries

We give a number of corollaries and other results related to our main theorems.

Corollary 4.1. *The following classes are closed under complementation, intersection, union, concatenation, and Kleene closure:*

(i) *$\mathscr{L}([1]SATM(S(n)))$ for any $S(n)$.*
(ii) *$\mathscr{L}(SUTM(S(n)))$ for $S(n)\geqslant\log(n)$.*
(iii) *$\mathscr{L}(1SUTM(S(n)))$ for $S(n)\geqslant n$.*

Also:

(i) *$\mathscr{L}(SUTM(S(n)))$ is not closed under complementation for $S(n)\in o(\log(n))$.*
(ii) *$\mathscr{L}(1SUTM(S(n)))$ is not closed under complementation for $S(n)\in o(n)$.*
(iii) *$\mathscr{L}(1SUTM(S(n)))$ is not closed under union for $S(n)\in o(\log(n))$.*
(iv) *$\mathscr{L}(1SUTM(S(n)))$ is closed under intersection for $S(n)\in[\log(n), n]$.*

Proof. The first part follows from Lemmas 3.9 and 3.6 and Theorem 3.12(ii).

The first two items of the second part follow from Lemma 3.4 and [24]. To prove the third item, consider $L_8=\{x\#x\#y\colon\ x,y\in\{0,1\}^*\}$ and $L_9=\{x\#y\#x\colon\ x,y\in\{0,1\}^*\}$. It is easy to see that L_8 and $L_9\in\mathscr{L}(1SA(2)FA)$; however, by using an argument similar to the one given in Lemma 3.2 we can show that $L_8 \cup L_9 \notin \mathcal{L}(1SUTM(S(n)))$ for any $S(n) \in o(\log(n))$. The fourth item follows from Lemma 3.5.

Corollary 4.2. *$\mathcal{L}([1]DSA(k)FA)$ is not closed under complementation for any $k>1$.*

Proof. We construct a $1DSA(2)FA$ M to accept L_4. On input x, M splits into 2 processes p_1 and p_2. Process p_2 moves to the right until it reaches the symbol # and then synchronizes with p_1 to compare the first half and the second half of the input. M accepts x iff $x = u \# v$, $u, v \in \{0,1\}^+$ and $u = v$, or x does not have the form $u \# v$, $u, v \in \{0,1\}^+$. The corollary then follows from Lemma 3.4.

Corollary 4.3. *$\mathcal{L}(1DSA(k)FA)$ is not closed under union or intersection for any $k>1$.*

Proof. For each $k>0$, let $R_k = \{w_1 \# w_2 \# \cdots \# w_k \# x \# y \# w_k \# \cdots \# w_2 \# w_1 : w_i\text{'s}, x, y \in \{0,1\}^*\}$, and $S_k = \{x \# w_1 \# w_2 \# \cdots \# w_k \# w_k \# \cdots \# w_2 \# w_1 \# y : w_i\text{'s}, x, y \in \{0,1\}^*\}$. Both R_k and S_k are in $\mathcal{L}(1DSA(k+1)FA)$, but $R_k \cap S_k = L_{k+1,0}$ is not, by Theorem 3.21. Hence $\mathcal{L}(1DSA(k)FA)$ is not closed under intersection for all $k>1$.

Now let $T_k = \{w \# x_1 \# x_2 \# \cdots \# x_k \# y_k \# \cdots \# y_2 \# y_1 \# w : x_i\text{'s}, y_i\text{'s}, w \in \{0,1\}^*\}$, which is in $\mathcal{L}(1DSA(k+1)FA)$. We show that $S_k \cup T_k \notin \mathcal{L}(1DSA(k+1)FA)$.

Suppose there is some $1DSA(k+1)FA$ M that accepts $S_k \cup T_k$. Using the same technique shown in Lemma 3.19, we can find a random string w_1 and a process p of M on $w_1 \#$ such that for all $v \in L_{k,0}$, process p on input $w_1 \# v \#$ enters no more than d s-states, where d is a constant depending only on w_1 and M. But then using the simulation technique in Lemma 3.20, we can show that $L_{k,0}$ can be accepted by some $1DSA(k)FA$ M'. This contradicts Theorem 3.21; hence $\mathcal{L}(1DSA(k)FA)$ is not closed under union for all $k>1$.

Corollary 4.4. *$\mathcal{L}(DSA(2)FA) - \bigcup_{k=1}^{\prime} \mathcal{L}(1DFA(k)) \neq 0$.*

Proof. Let $L_5 = \bigcup_{m=1}^{\prime} L_{m,0}$. L_5 can be accepted by a $DSA(2)FA$ M as follows. On input w, M splits into 2 processes p_1 and p_2. Process p_2 moves onto the right marker \$ and then synchronizes with p_1 to compare the w_i's. Note that p_2 moves backward after each comparison, whereas p_1 moves forward. Also, M makes sure that there is an odd number of #'s. M accepts w iff w is in L_5. Suppose L_5 is accepted by some $1DFA(k)$ M' for some k. Then we can construct a $1DFA(k)$ N to accept the language $L_{\binom{k}{2}+1,0}$ by making sure that w has exactly $k\cdot(k-1)+1$ #'s before accepting. This contradicts a result by Yao Rivest [26], which shows $L_{\binom{k}{2}+1,0} \notin \mathcal{L}(1DFA(k))$. Hence the corollary follows.

Corollary 4.5. *$\bigcup_{k>0}^{\prime} \mathcal{L}(DSA(k)FA) \subset \mathcal{L}(DFA(2))$.*

Proof. Given $L \in \mathcal{L}(DSA(k)FA)$ for some $k>0$, we construct a $DFA(2)$ M to accept L: on each input x, M first does a depth-first search with one head to identify the shape of the computation tree of M on x without paying any attention to synchronizing symbols. This can be done in constant space because there are only finitely many different trees with k leaves. M then uses this information to compare all $\binom{k}{2}$ possible pairs of s-sequences of the processes of M on x using two heads. M accepts x iff all k processes finish in accepting states and there are no deadlocks. $\square$

Corollary 4.6. *$\mathcal{L}(SUTM(S(n)))-\mathcal{L}(MUTM(S(n)))\neq\emptyset$ and $\mathcal{L}(SUTM(S(n)))-\mathcal{L}(ATM(S(n)))\neq\emptyset$ for any $S(n)\in[\log\log(n), \log(n)|$.*

Proof. Follows directly from Lemma 3.1. $\square$

5. Concluding remarks

Although the relationships between $\mathcal{L}(1SUTM(S(n)))$ and other classes of alternating Turing machines are well understood, some corresponding relationships concerning $\mathcal{L}(SUTM(S(n)))$ remain open or only partially understood. Below we list some partial results and remaining open questions as directions for further research.

The following relationships are known:

(i) $\mathcal{L}(1ATM(S(n)))-\mathcal{L}(UTM(S(n)))\neq\emptyset$ for $S(n)\in[\log\log(n), \log(n)|$ ([15]).

(ii) $\mathcal{L}(SUTM(S(n)))-\mathcal{L}(ATM(S(n)))\neq\emptyset$ for $S(n)\in[\log\log(n), \log(n)|$ (Corollary 4.6).

The following questions are open:

(i) $\mathcal{L}(UTM(S(n)))-\mathcal{L}(1ATM(S(n)))\neq\emptyset$ for $S(n)\in[\log\log(n), \log(n)|$ ([15])?

(ii) $\mathcal{L}(1ATM(S(n)))-\mathcal{L}(SUTM(S(n)))\neq\emptyset$ for $S(n)\in[\log\log(n), \log(n)|$?

(iii) $\mathcal{L}(DSA(k)FA)\subset\mathcal{L}(DSA(k+1)FA)$ for all $k>0$?

(iv) Characterize $\mathcal{L}(SUTM(S(n)))$ for $S(n)\in o(\log n)$.

(v) $\mathcal{L}(1SATM(S(n)))-\mathcal{L}(1ATM(S(n)))\neq\emptyset$ for $S(n)\in[\log(n), n|$?

Note that if $\mathcal{L}(1SATM(\log(n)))=\mathcal{L}(1ATM(\log(n)))$ then alternating Turing machines use space optimally, and hence P = NP (see [8]). We conjecture that the answer to (iii) is positive and offer the class of languages $T_i=\{x_1 \# x_2 \# \cdots \# x_i$: $x_1, x_2, \ldots, x_i \in \{0, 1\}^*$ are pairwise compatible$\}$ as candidate witness languages.

References

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

[2] J.H. Chang, O.H. Ibarra and B. Ravikumar, Some observations concerning alternating Turing machines using small space, *Inform. Process. Lett.* **25** (1987) 1–9.

[3] 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.

[4] E.M. Gurari and O.H. Ibarra, (Semi-)alternating stack automata, *Math. Systems Theory* **15** (1982) 211–224.

[5] J. Hromkovič, Alternating multicounter machines with constant number of reversals, *Inform. Process. Lett.* **21** (1985) 7–9.

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

[7] J. Hromkovič, How to organize the communication among parallel processes in alternating computations. Manuscript, 1986.

[8] J. Hromkovič, J. Karhumäki, B. Rovan and A. Slobodová, On the power of synchronization in parallel computations, Tech. Report, Comenius University, Bratislava, Czechoslovakia, Department of Theoretical Cybernetics and Institute of Computer Science, 1989.

[9] N. Immerman, Nondeterministic space is closed under complementation, in: *Proc. 3rd IEEE Conf. on Structure in Complexity Theory* (1988) 112–115.

[10] K. Inoue, A. Ito, and I. Takanami, Alternating Turing machines with modified accepting structure, manuscript, 1989.

[11] K. Inoue, A. Ito, I. Takanami and H. Taniguchi, Two-dimensional alternating Turing machines with only universal states, *Inform. and Control* **55** (1982) 193–221.

[12] K. Inoue, A. Ito, I. Takanami and H. Taniguchi, A space-hierarchy results on two-dimensional alternating Turing machines with only universal states, *Inform. Sci.* **35** (1985) 79–90.

[13] K. Inoue, H. Matsuno, I. Takanami and H. Taniguchi, Alternating simple multihead finite automata, *Theoret. Comput. Sci.* **36** (1985) 291–308.

[14] K. Inoue, I. Takanami and R. Vollmar, Alternating on-line Turing machines with only universal states and small space bounds, *Theoret. Comput. Sci.* **41** (1985) 331–339.

[15] A. Ito, K. Inoue and I. Takanami, A note on alternating Turing machines using small space, *Trans. IECE Japan E* **70** (1987) 990–996.

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

[17] K.N. King, Alternating multihead finite automata, *Theoret. Comput. Sci.* **61** (1988) 149–174.

[18] R.E. Ladner, R.J. Lipton and L.J. Stockmeyer, Alternating pushdown automata, in: *Proc. 19th IEEE FOCS* (1978) 92–106.

[19] M. Li and P.M.B. Vitányi, Two decades of applied Kolmogorov complexity in memoriam Andrei Nikolaevich Kolmogorov 1903–1987, in: *Proc. 3rd IEEE Conf. on Structure in Complexity Theory* (1988) 80–101.

[20] M. Li and P.M.B. Vitányi, A new approach to formal language theory by Kolmogorov complexity (preliminary version), in: *Proc. 16th ICALP'89*, Lecture Notes in Computer Science, Vol. 372 (Springer, Berlin, 1989) 506–520.

[21] W.J. Paul, J.I. Seiferas and J. Simon, An information-theoretic approach to time bounds for on-line computation, *J. Comput. System Sci.* **23** (1981) 108–126.

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

[23] 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.

[24] A. Slobodová, Some properties of space-bounded synchronized alternating Turing machines with only universal states, in: *Proc. 5th IMYCS '88*, Lecture Notes in Computer Science, Vol. 381 (Springer, Berlin, 1988) 102–113.

[25] J. Wiedermann, On the power of synchronization, Tech. Report, VUSEI-AR, Bratislava, Czechoslovakia, 1987.

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