$StUSPACE(\log n) \subseteq DSPACE(\log^2 n / \log\log n)$

Eric Allender*[1] and Klaus-Jörn Lange**[2]

[1] Department of Computer Science
Rutgers University
P.O. Box 1179
Piscataway, NJ 08855-1179
USA
allender@cs.rutgers.edu

[2] Wilhelm-Schickard Institut für Informatik
Universität Tübingen
Sand 13
D-72076 Tübingen
Germany
lange@informatik.uni-tuebingen.de

Abstract

We present a deterministic algorithm running in space $O\left(\log^2 n / \log\log n\right)$ solving the connectivity problem on strongly unambiguous graphs. In addition, we present an $O(\log n)$ time-bounded algorithm for this problem running on a parallel pointer machine.

1 Introduction

One of the most central questions of complexity theory is to relate determinism and nondeterminism. Our inability to exhibit the precise relationship between these two notions motivates the investigation of intermediate notions such as symmetry or unambiguity. In this paper we concentrate on unambiguity in space-bounded computation, and present improved deterministic and parallel simulations.

Recently, surprising results have indicated that "symmetric" space bounded computation is weaker than nondeterminism. In particular, symmetric logspace has been shown to be contained in parity logspace [12], in SC^2 [18], and in $DSPACE(\log^{1.5} n)$ [19]. None of these upper bounds is known to hold in the nondeterministic case. If we consider these questions for space bounded unambiguous classes, we are confronted with the fact that there are several ways to define notions of unambiguity that apparently do not coincide [3]. In this paper we will concentrate on the notion of unambiguity (in the sense of unique existence of computation paths), and of strong unambiguity (in the sense of

* Supported in part by NSF grant CCR-9509603.
** Supported in part by NSF grant CCR-9509603.

uniqueness of computations between any pair of configurations). This yields the two classes $USPACE(\log n)$ and $StUSPACE(\log n)$.

By definition, $USPACE(\log n)$ is a subclass of parity logspace; this is not known to hold for $NSPACE(\log n)$ (although see [23]); there is no additional nontrivial containment known for $USPACE(\log n)$. However, $StUSPACE(\log n)$ is contained in SC^2, since strongly unambiguous logspace languages can be accepted by deterministic auxiliary pushdown automata in polynomial time [3,5]. Still, it was unknown whether there are $o(\log^2 n)$ space algorithms for strongly unambiguous logspace languages. We answer this question affirmatively by showing that $StUSPACE(\log n)$ is contained in $DSPACE(\log^2 n/\log\log n)$.

Since $StUSPACE(\log n)$ is a subclass of $DAuxPDA\text{-}TIME\big(n^{O(1)}\big)$ we know that there are logtime *CROW*-algorithms for the elements of $StUSPACE(\log n)$ [7]. To give better relative upper bounds on the complexity of $StUSPACE(\log n)$ it is interesting to consider intermediate classes between $DSPACE(\log n)$ and $DAuxPDA\text{-}TIME\big(n^{O(1)}\big)$.

Trying to find these classes with sequential models could be difficult, since the usual restrictions of a pushdown store (e.g. the one-turn property, or using a counter instead of a push down) all collapse to logspace. Here, parallel machine models seem helpful, leading to two intermediate classes: the parallel pointer machine [6,14] and the *OROW*-PRAM [20]. As a consequence of our main result we get *OROW* algorithms for the elements of $StUSPACE(\log n)$ taking time $O(\log^2 n/\log\log n)$. We are also able to show that all sets in $StUSPACE(\log n)$ are accepted in logarithmic time on a parallel pointer machine. This latter containment is somewhat surprising, because there are characterizations in terms of parallel programs indicating that the class $PPM\text{-}TIME(\log n)$ is rather close to $DSPACE(\log n)$ [16].

2 Preliminaries

We assume the reader to be familiar with the basic notions of complexity theory (e.g. [10]). In addition, let $DTISP(f, g)$ be the set of all languages accepted by $O(g)$ space-bounded Turing machines in time $O(f)$. $NTISP(f, g)$ denotes the corresponding nondeterministic class.

We refer the reader to the survey article of Karp and Ramachandran [13] for coverage of the many varieties of parallel random access machines and their relationship to sequential classes. Let us remark here, that we deal in this paper only with algorithms and classes using PRAMs with a polynomial number of processors. The notion of a *CROW*-PRAM was introduced by Dymond and Ruzzo [7] and provides the tightest possible connections to deterministic machines [9].

CROW-PRAMs need only logarithmic time to recognize any given language in $DSPACE(\log n)$; There are two important ways to restrict *CROW*-PRAMs and still maintain this property. One way is to restrict the concurrent read access to the global memory, which leads to the *OROW*-PRAMs of Rossmanith [20]. The other way is to restrict the arithmetical capabilities of the instruction set leading to *rCROW*-PRAMs and to parallel pointer machines [6,14].

3 Unambiguity

A concept intermediate in power between determinism and nondeterminism is *Unambiguity.* A nondeterministic machine is said to be unambiguous, if for every input there exists at most one accepting computation. This leads to the classes *UP* and *USPACE*$(\log n)$; we have $P \subseteq UP \subseteq NP$ and $DSPACE(\log n) \subseteq USPACE(\log n) \subseteq NSPACE(\log n)$. The notion of ambiguity should be distinguished from that of *Uniqueness*, which uses the unique existence of an accepting path not as a restriction but as a tool. The resulting language classes $1NSPACE(\log n)$ and $1NP$ consists of languages defined by machines that accept their inputs if there is exactly one accepting path. Thus, the existence of two or more accepting computations is not forbidden, but simply leads to rejection. In the polynomial time case we have $Co\text{–}NP \subseteq 1NP$ [2]. In the logspace case inductive counting [11,22] shows $1NSPACE(\log n) = NSPACE(\log n)$.

A more restrictive form of unambiguity is *Strong Unambiguity.* A nondeterministic machine is said to be strongly unambiguous, if for *every* pair of configurations there exits at most one computational path connecting these configurations. An ordinary unambiguous machine makes this restriction only for the initial and the accepting configurations of the machine.

While these two concepts coincide (yielding the class *UP*) in the case of time bounded computations, this is not known to be true in the space bounded case. There we end up with an additional class $StUSPACE(\log n)$ which is located between $DSPACE(\log n)$ and $USPACE(\log n)$. Correspondingly, $StUTISP(f, g)$ is the class of all languages accepted by strongly unambiguous Turing machines that are simultaneously $O(g)$ space- and $O(f)$ time-bounded. In fact, there are several more versions of unambiguity that are not known to coincide (depending for instance on whether or not there can be more than one accepting configuration, see [3]), but in this work we consider only the two classes $USPACE(\log n)$ and $StUSPACE(\log n)$. Our algorithms work for all unambiguous classes for which the unfoldings of the configuration graphs to trees are of polynomial size. This is the case for $StUSPACE(\log n)$ but not for $USPACE(\log n)$.

In the time-bounded setting, problems such as factoring and primality have efficient unambiguous algorithms but are not known to possess deterministic algorithms with a comparable running time [8]. Recently, Lange presented a problem that is complete for a subclass of $USPACE(\log n)$ [15]; this is the first explicit presentation of a problem in $USPACE(\log n)$ that is not known to be in $DSPACE(\log n)$. (Completeness is a tool that is not often available in studying unambiguous classes. None of *UP*, $USPACE(\log n)$, or $StUSPACE(\log n)$ is known or believed to have complete sets.)

No problem is known to be in $StUSPACE(\log n)$ that is not known to be in $DSPACE(\log n)$. Nonetheless, there is an important *class* of languages with this property: The class of unambiguous linear languages, *ULIN*, is contained in $StUSPACE(\log n)$ [3], and it is not known to be contained in $DSPACE(\log n)$. To date, however, no explicit example of a language in *ULIN* has been exhibited for which a $DSPACE(\log n)$ algorithm is not known. (Note in this regard that the linear context free languages are $NSPACE(\log n)$–complete.)

Logspace classes are closely connected to graph accessibility problems. These connectivity problems in directed graphs, undirected graphs, and in trees are complete for $NSPACE(\log n)$, $SSPACE(\log n)$, resp. $DSPACE(\log n)$. For unambiguous classes the corresponding connectivity problems do not seem to be complete. Nevertheless it is useful to explain these notions in terms of directed graphs, since this will make the demonstration of our algorithms easier.

Let $G = (V, E)$ be a directed acyclic graph. If G has n nodes we assume V to be $\{1, 2, \cdots, n\}$ and we will be interested in the existence of a path leading from 1 to n. For each pair of nodes (x, y) let $d(x, y)$ be the length of the shortest path between x and y. If x and y are not connected, $d(x, y)$ is infinite; $d(x, x)$ is 0. In the following we will work with complete binary graphs; that is, each node of G is either a leaf with no outgoing edges, or an inner node with two outgoing edges. We assume the leaves to be accepting or rejecting. This is determined by a mapping $\phi : V \longrightarrow \{+, -, i\}$, which takes the value $+$ for accepting leaves, $-$ for rejecting leaves and i otherwise. In particular, 1 is an inner node (unless the graph has only one vertex) and we assume n to be the only accepting leaf, since we are only interested in paths from 1 to n. The two successors of an inner node x will be denoted by $L(x)$ and $R(x)$.

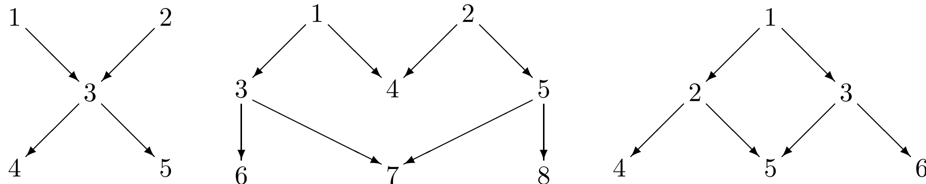
Let $P_E := \{(a_0, a_1)(a_1, a_2) \cdots (a_{k-1}, a_k) | k \geq 1, (a_{i-1}, a_i) \in E\}$ be the set of all paths in G. We define a mapping $C : P_E \longrightarrow V \times V$ by mapping a path leading from node x to node y to the pair (x, y), i.e.: we set

$$C((a_0, a_1)(a_1, a_2) \cdots (a_{k-1}, a_k)) := (a_0, a_k).$$

For $x \in V$ let $T(x) := \left\{y \in V | C^{-1}((x, y)) \neq \emptyset\right\} \cup \{x\}$ be the set of all nodes reachable from x.

G is called *unambiguous* if there is at most one path from 1 to n, i.e.: if $\left|C^{-1}((1, n))\right| \leq 1$. G is called *strongly unambiguous* or a *Mangrove* if for any pair (x, y) of nodes there is at most one path leading from x to y, i.e.: if $\forall_{x,y \in V} \left|C^{-1}((x, y))\right| \leq 1$. Although a mangrove does not need to be a tree, for each x the subgraph of G induced by $T(x)$ is indeed a tree and the same is true for the set of all nodes from which x can be reached.

The following three examples show different version of unambiguous graphs. The first one is a very simple mangrove with a positive result since there is a path from node 1 to node 5. The second one is also a mangrove, but a negative one, since there is no path from 1 to 8. The third graph is unambiguous and positive, since there is exactly one path from 1 to 6, but it is not a mangrove, since there are two different paths from node 1 to node 5.



Typical examples of mangroves are butterfly graphs or the *Reinhardt Graphs* $R_n := (V_n^r, E_n^r)$ with $V_n^r := \{1, 2, \cdots, n\}^2$ and E_n^r is $\{(\langle i, j\rangle, \langle i, j+1\rangle) | 1 \leq i \leq$

$n, 1 \leq j < n\} \cup \{(\langle i,j\rangle, \langle i+j \bmod n, n\rangle) | 1 \leq i \leq n, 1 \leq j < n\}$. (Observe that the Reinhardt Graph is not a complete binary graph.)

By definition a Turing machine is unambiguous if for each input word the reachability graph of its configurations is unambiguous. It is strongly unambiguous if for every input this graph forms a mangrove.

4 Contracting Mangroves

The class *StUSPACE*($\log n$) has been shown to be contained in *DAuxPDA-TIME*(pol) by unfolding the reachability graph of a mangrove to a tree of polynomial size [3]. This tree can be searched with the help of a push-down store in polynomial time. The *DAuxPDA-TIME*$(n^{O(1)})$-completeness of *DCFL*[21] together with the algorithm of Cook for deterministic context free languages [5] implies *StUSPACE*($\log n$) $\subseteq SC^2$. Elements of Cook's algorithm were later used by Dymond and Ruzzo [7] and independently by Monien et al [17] to show *DCFL* $\subseteq$ *CROW-TIME*($\log n$) (see also [9]). As a consequence, the elements of *StUSPACE*($\log n$) possess logtime algorithms running on a CROW–PRAM. But these algorithms for mangroves, generated by composing the constructions in [3] with those of [7,9,17], are rather complicated. Below, we give a very simple logtime algorithm that runs on a parallel pointer machine (which is a restricted CROW-PRAM).

Theorem 1.

$$StUSPACE(\log n) \subseteq PPM\text{-}TIME(\log n)$$

Proof: We first describe the parallel algorithm and then indicate how to run it on a parallel pointer machine. Let A be a strongly unambiguous logspace machine and $G = (V, \phi, L, R)$ be its configuration mangrove induced by some input w of size n. For each $x \in V$ the tree $T(x)$ is of polynomial size in n. What we would like to do on each $T(x)$ is pointer jumping. Unfortunately its pointers are directed towards the leaves instead of towards the root. The alternative is to perform the shunt operation (see e.g. [13]). This would need backpointers directed towards the root, which are usually provided by establishing an Euler tour through a tree. This is also not possible in a mangrove. The simple way out of this is to perform the shunt operation not by the parent node of a leaf but by its grandparent node, in parallel in each tree. Of course, these operations destroy any tree structure unless they are synchronized. But obviously the property of being a mangrove is invariant under the parallel application of shunt operations. Hence there is no need to do some sophisticated arrangement of the working processors as in [1,4]. (It should be remarked that the shunt operation preserves the outdegrees of the nodes to be either zero or two. This is not true for indegrees.) This is explained in more detail below.

For each $x \in V$ that is not a leaf, i.e. $\phi(x) = i$, we first check whether one of its children is an accepting leaf, in this case we set $\phi(x) := +$. If both children are rejecting we set $\phi(x) := -$. Otherwise, one of the children has to be an inner node. If $\phi(L(x)) = i$ we check if the left grandson via the left subtree is

a rejecting leaf. In that case we set $L(x) := R(L(x))$. If this was not the case we see whether the right grandson is a rejecting leaf which would lead to the assignment $L(x) := L(L(x))$. This is then repeated for the right son of x. After performing this parallel transformation $O(\log n)$ times, there is no inner node left; for each $x \in V$ we have either $\phi(x) = +$ or $\phi(x) = -$. The existence of an accepting path is then equivalent to $\phi(1) = +$.

Formally, the algorithm is expressed by the following statements:

for $j = 1, 2, \cdots, O(\log n)$ **do**
  for all $x \in V$ **do in parallel**
    if $\phi(x) = i$ **then**
      if $\phi(L(x)) = +$ **or** $\phi(R(x)) = +$ **then** $\phi(x) := +$;
      if $\phi(L(x)) = -$ **and** $\phi(R(x)) = -$ **then** $\phi(x) := -$;
      if $\phi(L(x)) = i$ **then**
        if $\phi(L(L(x))) = -$ **then** $L(x) := R(L(x))$
        else
          if $\phi(R(L(x))) = -$ **then** $L(x) := L(L(x))$;
      if $\phi(R(x)) = i$ **then**
        if $\phi(L(R(x))) = -$ **then** $R(x) := R(R(x))$
        else
          if $\phi(R(R(x))) = -$ **then** $R(x) := L(R(x))$;
if $\phi(1) = +$ **then accept**
else reject

We now shortly indicate how to put all this on a parallel pointer machine. Obviously, the algorithm itself is already suitable for a PPM. It remains only to show how to build the mangrove of configurations in logarithmic time, given the input word. As in the corresponding construction of Cook and Dymond [6] the initial PPM unit starts to build in logarithmic time a tree of logarithmic depth such that each leaf unit corresponds to a configuration of A on w. Then the leaves are interconnected according to the successor relation of A. But instead of one pointer leading to a successor, each leaf unit will now have two pointers L and R representing the two subtrees hanging below an inner node of a mangrove. □

Let us remark here that the number of PPM units, i.e. processors, is linear in the size of the mangrove. For the special case of *ULIN*, this leads to a quadratic number of processors. The best known sequential algorithm for solving membership questions in *ULIN* needs quadratic time.

Our algorithm makes intensive use of concurrent reads and therefore does not pertain to owner read PRAMs. It works via $O(\log n)$ applications of the operations of searching and pruning trees of depth 2. Increasing this depth to $O(\log n)$ is the first step in reducing the number of iterations to $O(\log n/ \log\log n)$. This yields an algorithm using $o(\log^2 n)$ space (or parallel time).

Theorem 2.

$$StUSPACE(\log n) \subseteq DSPACE(\log^2 n/ \log\log n)$$

PROOF: Let A be a strongly unambiguous logspace machine and let w be an input of length n. This yields a reachability problem in a mangrove $G = (V, \phi, L, R)$ of polynomial size.

We will now construct a mangrove $G' = (V, \phi', L', R')$ of smaller size. Let $t \geq 1$ be an integer that determines the rate of contraction of G' compared to G. We will fix the value of t later. First we map each node x to one of its successors which we call $f(x)$.

If in the following procedures an accepting leaf (i.e. a node y with $\phi(y) = +$) is found, the search is stopped, and we set $\phi'(x) := +$ and $f(x) := x$. We search for each $x \in V$ the tree $T_t(x) := \{y \in V | d(x, y) \leq t\}$ of all nodes of distance to x not greater than t.

Let z be a pointer initialized to x. Consider the set M of all nodes y in $T_t(z)$ with $d(z, y) = t$ and $\phi(y) = i$, i.e. of all leaves of $T_t(z)$ that are not leaves in $T(z)$. If M is empty, that is if $T(z) = T_t(z)$ then set $\phi'(x) := -$ and $f(x) := x$. If M is nonempty we replace z by the least common ancestor z' of M in $T_t(z)$. If $z = z'$ we stop the procedure for x and set $f(x) := z$ and $\phi'(x) := i$. Otherwise we replace z by z' and continue the process.

This algorithm computing f and ϕ' in a more formal way looks as follows, where $LCA(M)$ denotes the least common ancestor of a set M of nodes in a tree:

$z := x;$
$M := \{y \in T_t(z) \,|d(z, y) = t, \phi(y) = i\,\};$
while $+ \notin \phi(T_t(z))$ **and** $M \neq \emptyset$ **and** $LCA(M) \neq z$ **do**
begin
    $z := LCA(M);$
    $M := \{y \in T_t(z) \,|d(z, y) = t, \phi(y) = i\,\}$
end
if $+ \in \phi(T_t(z))$
  then $\phi'(x) := +; f(x) := x$
  else if $M = \emptyset$
    then $\phi'(x) := -; f(x) := x$
    else $\phi'(x) := i; f(x) := z$

After this process, for each $x \in V$ either $\phi'(x) = +$, i.e. an accepting leaf has been found in $T(x)$, or $\phi'(x) = -$, i.e. $T(x)$ has been totally searched and contains rejecting leaves, only, or $\phi'(x) = i$. In this case x points to $f(x)$ which is a node such that both subtrees of $f(x)$ contain nodes of a distance larger than t to $f(x)$. That is, both subtrees are of height not smaller than t, and, since both are complete binary trees, both of them are of a size not smaller than $2t + 1$.

Clearly, the computation of f can be done in space $t + O(\log n)$. (Note that M need not be stored explicitly.) We now construct G' by setting $L'(x) := f(L(f(x)))$ and $R'(x) := f(R(f(x)))$ for each x with $\phi'(x) = i$. We have for each $x \in V$ with $\phi'(x) = i$:

$$(P1)\ \ |T(L(f(x)))| \geq 2t + 1 \text{ and } |T(R(f(x)))| \geq 2t + 1.$$

Furthermore, the construction implies

$$(P2)\;\; f(f(x)) = f(x) \text{ and } (P3)\;\; \phi'(x) = \phi'(f(x))$$

for every $x \in V$.

Now let $T'(x)$ be the tree below x in G'. Although there may be nodes in V that are not finished (i.e. $\phi'(x) = i$) and that are not contracted, (i.e. $f(x) = x$) the mangrove in total is smaller by a factor of t; we claim for each $x \in V$ with $\phi'(x) = i$:

$$|T'(x)| \leq |T(x)| / t$$

> Proof of the claim: Let $L'(z)$ be a left leaf in the tree $T'(x)$. That is, $\phi'(z) = i$ and $\phi'(L'(z)) \neq i$. Then by (P3), we have that $\phi'(L(f(z))) = \phi'(f(L(f(z))))$, and by definition of L' this is equal to $\phi'(L'(z)) \neq i$. Thus $L'(z) = f(L(f(z))) = L(f(z))$ by the construction of f. Hence by (P1), $|T(L'(z))| = |T(L(f(z)))| \geq 2t + 1$. Hence for each left leaf $L'(z)$ in $T'(x)$ there are at least $2t$ nodes that are removed from $T(x)$. And the same holds for right leaves. Hence $T(x)$ is not smaller than $2t$ times the number of leaves of $T'(x)$. Since at least half of the elements of a complete binary tree are leaves the result follows.

Repeating this process of shrinking G ends in a mangrove of depth $O(1)$ after $O(\log_t n)$ phases. Setting $t := \log n$ we obtain $O(\log n / \log\log n)$ phases, each of which uses $O(\log n)$ space, to obtain a procedure to search a mangrove using $O(\log^2 n / \log\log n)$ space. □

We remark here, that the resulting algorithm is in general not polynomial time-bounded.

Corollary 3.

$$\mathit{ULIN} \subseteq \mathit{DSPACE}(\log^2 n / \log\log n)$$

All of $\mathit{DSPACE}(f)$ can be recognized by *OROW*-PRAMs in $O(f)$ steps. But generally these algorithms need $c^{O(f)}$ many processors, which in our case would be superpolynomial. But due to the recursive or iterative structure of our algorithm a polynomial number suffices, as we now show.

Corollary 4.

$$\mathit{StUSPACE}(\log n) \subseteq \mathit{OROW}\text{-}\mathit{TIME}\left(\log^2 n / \log\log n\right)$$

Proof: The PRAM works in $\log n / \log\log n$ phases, each taking $O(\log n)$ steps. In each phase the work of a logarithmically space bounded TM is simulated in logarithmic time by an *OROW*-PRAM with a polynomial number of processors [20]. The difference is here that we don't simulate an accepting machine, but one producing some output. This output is stored in global memory. In order to use this output as input of the next phase, we first have to distribute it to polynomially many processors in a tree-like way in $O(\log n)$ steps. In total this costs $O(\log^2 n / \log\log n)$ steps on an *OROW*-PRAM. The number of processors

corresponds to the number of configurations of the simulated logspace machine and hence is polynomial. □

We mention in passing that this result, applied to the unambiguous linear languages, leads to algorithms using $O(n^2)$ processors since this is the size of the corresponding mangrove.

We end this section by remarking that our main theorem generalizes in the following way: It is well-known that $\mathit{NTISP}(f,g)$ is a subset of $\mathit{DSPACE}(g \log f)$. If we restrict the nondeterminism to be strongly unambiguous we get the sharper bound $\mathit{StUTISP}(f,g) \subseteq \mathit{DSPACE}(g \cdot \log f / \log g)$.

5 Discussion and open questions

The most basic open question is, of course, to clarify the relationship between $\mathit{StUSPACE}(\log n)$ and $\mathit{DSPACE}(\log n)$. It might very well be that these two classes coincide. Indeed, there seem to be no drastic consequences implied by such a collapse. A first step in this direction would be to exhibit a logtime OROW-algorithm for $\mathit{StUSPACE}(\log n)$ (or even for *ULIN*).

Another open issue concerns the class $\mathit{USPACE}(\log n)$. Is it possible to place it in SC^2 or to give an $o(\log^2 n)$ space algorithm as it has been possible for symmetric logspace and the class $\mathit{StUSPACE}(\log n)$?

A third line of investigation is to consider these questions for polynomial time bounded auxiliary pushdown automata where results like our theorems 1 and 2 are probably harder to obtain. The complexity of *UCFL*, the class of unambiguous context free languages, is uncertain since we do not know whether they are complete for the strongly unambiguous auxiliary pushdown class, just as we don't know whether *ULIN* is complete for $\mathit{StUSPACE}(\log n)$. Since $\mathit{StUSPACE}(\log n) \subseteq \mathit{DAuxPDA\text{-}TIME}(n^{O(1)})$ and since *DCFL* is complete for the later class, we know $\mathit{StUSPACE}(\log n) \subseteq \mathit{LOG}(\mathit{UCFL})$. It is not known whether *UCFL* is also hard for $\mathit{USPACE}(\log n)$. On the other hand it might well be the case that *UCFL* is contained in SC^2.

References

1. R. J. Anderson and G. L. Miller. Deterministic parallel list ranking. In *VLSI Algorithms and Architectures, Proc. 3rd Aegean Workshop on Computing*, number 319 in LNCS, pages 81–90. Springer, 1988.
2. A. Blass and Y. Gurevich. On the unique satisfiability problem. *Inform. and Control*, 55:80–88, 1982.
3. G. Buntrock, B. Jenner, K.-J. Lange, and P. Rossmanith. Unambiguity and fewness for logarithmic space. In *Proc. of the 8th Conference on Fundamentals of Computation Theory*, number 529 in LNCS, pages 168–179, 1991.
4. R. Cole and U. Vishkin. Approximate parallel scheduling, part i: the basic technique with applications to optimal parallel list ranking in logarithmic time. *SIAM J. Comp.*, 17:128–142, 1988.

5. S. Cook. Deterministic CFL's are accepted simultaneously in polynomial time and log squared space. In *Proc. of the 11th Annual ACM Symp. on Theory of Computing*, pages 338–345, 1979.
6. S. Cook and P. Dymond. Parallel pointer machines. *Computational Complexity*, 3:19–30, 1993.
7. P. Dymond and W. Ruzzo. Parallel RAMs with owned global memory and deterministic context-free language recoginition. In *Proc. of 13th International Colloquium on Automata, Languages and Programming*, number 226 in LNCS, pages 95–104. Springer, 1986.
8. M. Fellows and N. Koblitz. Self-witnessing polynomial-time complexity and prime factorization. In *Proc. of the 7th IEEE Structure in Complexity Conference*, pages 107–110, 1992.
9. H. Fernau, K.-J. Lange, and K. Reinhardt. Advocating ownership. In *Proc. of the 17th FST&TCS*, 1996. to be published.
10. J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Language, and Computation*. Addison-Wesley, Reading Mass., 1979.
11. N. Immerman. Nondeterministic space is closed under complementation. *SIAM J. Comp.*, 17:935–938, 1988.
12. M. Karchmer and A. Wigderson. On span programs. In *Proc. of the 8th IEEE Structure in Complexity Theory Conference*, pages 102–111, 1993.
13. R.M. Karp and V. Ramachandran. A Survey of Parallel Algorithms for Shared-Memory Machines. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Vol. A*, pages 869–941. Elsevier, Amsterdam, 1990.
14. T. Lam and W. Ruzzo. The power of parallel pointer manipulation. In *Proc. of the 1st ACM Symposium on Parallel Algorithms and Architectures (SPAA'89)*, pages 92–102, 1989.
15. K.-J. Lange. An unambiguous class possessing a complete set. Manuscript, 1996.
16. K.-J. Lange and R. Niedermeier. Data-independences of parallel random access machines. In *Proc. of 13th Conference on Foundations of Software Technology and Theoretical Computer Science*, number 761 in LNCS, pages 104–113. Springer, 1993.
17. B. Monien, W. Rytter, and H. Schäpers. Fast recognition of deterministic cfl's with a smaller number of processors. *Theoret. Comput. Sci.*, 116:421–429, 1993. Corrigendum, 123:427,1993.
18. N. Nisan. $RL \subseteq SC$. In *Proc. of the 24th Annual ACM Symposium on Theory of Computing*, pages 619–623, 1992.
19. N. Nisan, E. Szemeredi, and A. Wigderson. Undirected connectivity in $O(\log^{1.5} n)$ space. In *Proc. of 33th Annual IEEE Symposium on Foundations of Computer Science*, pages 24–29, 1992.
20. P. Rossmanith. The owner concept for PRAMs. In *Proc. of the 8th STACS*, number 480 in LNCS, pages 172–183. Springer, 1991.
21. I. Sudborough. On the tape complexity of deterministic context-free languages. *J. Assoc. Comp. Mach.*, 25:405–414, 1978.
22. R. Szelepcsényi. The method of forcing for nondeterministic automata. *Acta Informatica*, 26:279–284, 1988.
23. A. Wigderson. NL/poly $\subseteq$ $\oplus$L/poly. In *Proc. of the 9th IEEE Structure in Complexity Theory Conference*, pages 59–62, 1994.