Discrete Applied Mathematics 25 (1989) 19-26
North-Holland

A NEW COMPLETE LANGUAGE FOR DSPACE($\log n$)*

Jian-er CHEN
Department of Mathematics, Columbia University, New York, NY 10027, USA

Received 9 January 1989

An important open problem relating sequential and parallel computations is whether the space complexity on Turing machines is linearly related to the depth complexity on uniform circuits. Some graph problems have been successfully proved to be complete for DSPACE($\log n$) under ($\log n$)-depth Turing reducibility [3]. In this paper, we discuss ($\log n$)-depth many-one reducibility which is proved to be weaker than ($\log n$)-depth Turing reducibility. A new complete language for DSPACE($\log n$) under our reducibility is presented.

1. Introduction and definitior

An important open problem which relates sequential and parallel computations is to determine if the space complexity on Turing machines is linearly related to the depth complexity on uniform circuits. The best result known is by Borodin [1]:

$$\text{DEPTH}(s(n)) \subseteq \text{DSPACE}(s(n)) \subseteq \text{DEPTH}((s(n))^2).$$

Any improvement to this is a breakthrough. One approach is to find a "hardest" problem in DSPACE($s(n)$) and see if it can be accepted by a uniform circuit family of subquadratic depth. Some attempts have been made and some graph problems have been successfully proved to be complete for DSPACE($\log n$) under ($\log n$)-depth Turing reducibility (will be defined later) [3]. In this paper, we discuss ($\log n$)-depth many-one reducibility which is proved to be weaker than the $\log n$-depth Turing reducibility. We present a new problem which is complete for DSPACE($\log n$) under our reducibility.

Some basic definitions are given below.

A (Boolean) *circuit* C with n inputs and m outputs is a finite directed acyclic graph with nodes (called *gates*) labelled as follows. The circuit C has n "input gates" with indegree zero labelled $x_1, \dots, x_n$, respectively. All other gates of indegree zero are labelled either 0 or 1. All gates of indegree one are labelled $\neg$. All other gates have indegree two and are labelled either $\vee$ or $\wedge$. Exactly m gates are labelled output gates and have labels $y_1, \dots, y_m$, respectively. The *size* of C is the number of gates of C, and the *depth* of C is the length of the longest path from some input to some output. The circuit C computes a function $f: \{0,1\}^n \to \{0,1\}^m$ in the obvious way.

* This work is supported in part by NSF grants DCR-84-01898 and DCR-84-01633.

A *circuit family* with input size $g(n)$ and output size $h(n)$ is a sequence $\{C_n \mid n \geq 1\}$ of circuits, where C_n is a circuit with $g(n)$ inputs and $h(n)$ outputs. Let $f = \{f_n \mid n \geq 1\}$ be a sequence of Boolean functions, where $f_n : \{0,1\}^{g(n)} \to \{0,1\}^{h(n)}$. We say that the circuit family $\{C_n \mid n \geq 1\}$ *computes* f iff C_n computes f_n for all n. A circuit family α can also be used to *accept* a set $S \in \{0,1\}^*$ if α computes the characteristic function of S.

A circuit family $\alpha = \{C_n \mid n \geq 1\}$ is *uniform* if the transformation $1^n \to \bar{C}_n$ (where $\bar{C}_n$ is an encoding of the circuit C_n) can be performed in $O(\log n)$ space on a deterministic Turing machine.

A circuit family $\alpha = \{C_n \mid n \geq 1\}$ is in $\text{DEPTH}(d(n))$ if α is uniform and there is a constant c such that for all n the circuit C_n has depth at most $c \cdot d(n)$. Note that by the uniformity condition, if $\alpha = \{C_n \mid n \geq 1\}$ is in $\text{DEPTH}(d(n))$, then there is a polynomial p such that the size of C_n is bounded by $p(n)$ for all n.

We also say that a function $f : \{0,1\}^* \to \{0,1\}^*$ is in $\text{DEPTH}(d(n))$ if f is computed by a circuit family α in $\text{DEPTH}(d(n))$, and a language L is in $\text{DEPTH}(d(n))$ if L is accepted by a circuit family α in $\text{DEPTH}(d(n))$. The context will always make our meaning clear.

2. Logarithmic depth reducibilities

In this section, we will discuss two different kinds of $(\log n)$-depth reducibilities: The $(\log n)$-depth Turing reducibility and the $(\log n)$-depth many-one reducibility.

To define the $(\log n)$-depth Turing reducibility, we need the notion of an oracle gate. An *oracle gate* in a circuit is a k inputs, one output gate which, on an input x of length k, will produce the value 1 on its output edge iff x is in the specified oracle set. The contribution of this node to the depth of the path on which it lies in the circuit is $\lceil \log k \rceil$. An oracle circuit C^A relative to an oracle set $A \in \{0,1\}^*$ is a circuit whose gates can be oracle gates computing the characteristic function of the oracle set A (more precisely, if g is an oracle gate with k inputs in C^A, then g computes the characteristic function of $\{0,1\}^k \cap A$). An oracle circuit family $\alpha^A = \{C_n^A \mid n \geq 1\}$ relative to A is then a sequence of circuits such that C_n^A is an oracle circuit relative to A for all $n \geq 1$.

Given the discussion above, it now makes sense to define *uniform oracle circuit families*, and to define the relative class $\text{DEPTH}^A(d(n))$ (for full discussion, the reader is referred to [2, 10]).

Definition 2.1. A language L is $\log n$*-depth Turing reducible to* a language A (written $L \leq_{\text{T}}^{\text{LD}} A$) if and only if there is a uniform oracle circuit family $\alpha^A = \{C_n^A \mid n \geq 1\}$ in $\text{DEPTH}^A(\log n)$ which accepts L.

We are more interested in the following "weaker" reducibility.

Definition 2.2. A language A is (log n)*-depth many-one reducible to* a language B if and only if there is a function $f: \{0,1\}^* \to \{0,1\}^*$ which can be computed by a circuit family $\{C_n \mid n \geq 1\}$ in DEPTH(log n), such that for all $x \in \{0,1\}^*$, $x \in A$ if and only if $f(x) \in B$. We will write this relation as $A \leq_{\mathrm{m}}^{\mathrm{LD}} B$.

The following theorem shows that the (log n)-depth many-one reducibility is "weaker" than the log n-depth Turing reducibility in a natural sense.

Definition 2.3. Let $\leq_A$ and $\leq_B$ be two reducibilities. $\leq_A$ is *strictly stronger than* $\leq_B$ if

(1) for any pair of languages L_1 and L_2, $L_1 \leq_B L_2$ implies $L_1 \leq_A L_2$;

(2) there is a pair of languages S_1 and S_2 such that $S_1 \leq_A S_2$ holds but $S_1 \leq_B S_2$ does not hold.

Theorem 2.4. *The reducibility* $\leq_{\mathrm{T}}^{\mathrm{LD}}$ *is strictly stronger than the reducibility* $\leq_{\mathrm{m}}^{\mathrm{LD}}$.

Proof. Ladner, Lynch and Selman [7] have given a language $L \in$ DEXPTIME which is not reducible to its complement co-L under the Karp reducibility. Thus this language is not reducible to its complement co-L under the reducibility $\leq_{\mathrm{m}}^{\mathrm{LD}}$ since the circuit families in DEPTH(log n) are of polynomial size (see [4]). However, the language L is obviously reducible to its complement co-L under the $\leq_{\mathrm{T}}^{\mathrm{LD}}$ reducibility. □

3. A complete language for DSPACE(log n) under $\leq_{\mathrm{m}}^{\mathrm{LD}}$

Definition 3.1. Let $\leq$ be a reducibility. A language L is $\leq$*-hard* for the class C if and only if $S \leq L$ for all S in C. Further, L is $\leq$*-complete* for C if and only if L is $\leq$-hard for C and $L \in C$.

Recently, some graph problems have been successfully proved to be $\leq_{\mathrm{T}}^{\mathrm{LD}}$-complete for DSPACE(log n) [3] (a closer examination reveals that these problems are also $\leq_{\mathrm{m}}^{\mathrm{LD}}$-complete for DSPACE(log n)). Here we present a new $\leq_{\mathrm{m}}^{\mathrm{LD}}$-complete language for DSPACE(log n).

A *k-head finite automaton* is a Turing machine with only one read-only tape and k heads on this tape. We call a k-head finite automaton *one-way* if all heads of it can move only to the right but cannot move to the left (it is allowed for the heads to make stationary moves), otherwise we call it *two-way*.

Define a language *i-k-DFA* as follows:

$$i\text{-}k\text{-DFA} = \{x \# M \mid M \text{ is a } i\text{-way } k\text{-head deterministic finite automaton accepting the string } x\}.$$

We are going to prove that 1-2-DFA is $\leq_{\mathrm{m}}^{\mathrm{LD}}$-complete for DSPACE($\log n$). Before doing this, we first need some lemmas.

The first lemma is due to Hartmanis [5].

Lemma 3.2. *If a language L is in* DSPACE($\log n$), *then there exist an integer k and a two-way k-head deterministic finite automaton F accepting L.*

Lemma 3.3. *Given a two-way k-head deterministic finite automaton M_1, there exist a function f_1, which can be computed by a circuit family in* DEPTH($\log n$), *and a one-way $(k+1)$-head deterministic finite automaton M_2 such that for all x, M_2 accepts $f_1(x)$ if and only if M_1 accepts x.*

Proof. Given a two-way k-head deterministic finite automaton M_1, without loss of generality, we assume that the running time of M_1 is not greater than $a \cdot n^k$, where n is the length of the input and a is a constant. Then we define

$$f_1(x) = (x\#)^{k \cdot a \cdot n^k}.$$

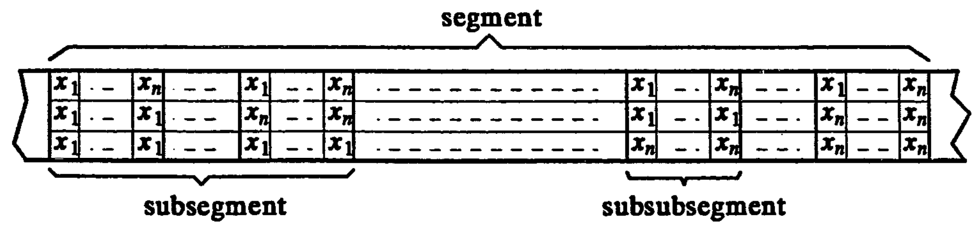
We construct a one-way $(k+1)$-head deterministic finite automaton M_2 such that for all x, M_2 accepts $f_1(x)$ if and only if M_1 accepts x.

M_2 will use k of its heads to simulate the k heads of M_1, respectively, and use the extra head as a counter. More precisely, suppose that the k heads of M_1 are $H_1, \ldots, H_k$, then let the $k+1$ heads of M_2 be $S_1, \ldots, S_k, S_{k+1}$. At the beginning of the computation of the automata, suppose all heads are placed at the left end of the inputs. Inductively, if after step i, the head H_j of M_1 is pointing to the tth symbol of the input, then the head S_j of M_2 will be pointing to the tth symbol of the ith copy of the $x\#$ in the string $f_1(x)$, i.e., the symbols being pointed by H_j and S_j are the same, $1 \leq i \leq a \cdot n^k$, $1 \leq j \leq k$. M_2 also remembers in its finite control the current state of M_1. Therefore in this configuration, M_2 can determine completely what M_1 will do in next step. If in next step M_1 moves its head H_j one square to right, then M_2 moves its corresponding head S_j $n+2$ squares to right; if H_j is moved one square to left, then S_j is moved n squares to right; finally, if H_j makes a stationary move, then S_j is moved $n+1$ squares to right. The extra head S_{k+1} of M_2 is used to count these n, $n+1$ and $n+2$. It is easy to see that after these modifications, M_2 is in the configuration corresponding to the configuration of M_1 at the end of the $(i+1)$st step. M_2 accepts $f_1(x)$ when it finds that M_1 is in an accepting state. For each step of M_1, the heads of M_2 cross at most k copies of $x\#$. Since M_1 runs in time $a \cdot n^k$, there are enough copies ($k \cdot a \cdot n^k$ copies) of $x\#$ in $f_1(x)$ for M_2 to complete the simulation of the whole computation of M_1. It is obvious that the function $f_1(x)$ can be computed by a circuit family in DEPTH($\log n$) and M_2 accepts $f_1(x)$ if and only if M_1 accepts x. $\square$

Lemma 3.4. *Given a one-way k-head deterministic finite automaton M_2, there exist a function f_2, which can be computed by a circuit family in* DEPTH($\log n$), *and* *a one-way two-head deterministic finite automaton M_3 such that for all x, M_3 accepts $f_2(x)$ if and only if M_2 accepts x.*

Proof. We only prove Lemma 3.4 for the case $k=3$. The proof can be easily generalized for any integer k.

Given a one-way three-head deterministic finite automaton M_2. Without loss of generality, we can assume that the running time of M_2 is at most $3n$. Given an input $x=x_1 \dots x_n \in \{0,1\}^*$, the value of the function $f_2(x)$ will be a string uuu where each segment u is a three-track string of the following form:



More precisely, $f_2(x)$ will be a three-track string, on the first track the track string is $(x_1x_2 \dots x_n)^{3n^2}$; on the second track the track string is $(x_1^n x_2^n \dots x_n^n)^{3n}$; and on the third track the track string is $(x_1^{n^2} x_2^{n^2} \dots x_n^{n^2})^3$. Now each position in $f_2(x)$ will be a possible tape head configuration of M_2. For example, the square of $f_2(x)$ which contains x_i, x_j and x_h on its first, second and third tracks, respectively, would indicate that the first, second and third heads of M_2 are pointing to the ith, jth and hth symbols of the input x, respectively. Now it is easy to see how our one-way two-head deterministic finite automaton M_3 simulates M_2: M_3 uses one head to simulate the movements of the heads of M_2 and uses another head as a counter. Suppose that the heads of M_2 are S_1, S_2 and S_3 and the heads of M_3 are T_1, T_2. At the beginning of the computations, all heads are placed on the left end of the inputs. Inductively, suppose that after the ith step, the heads S_1, S_2 and S_3 of M_2 are pointing to the i_1th, i_2th and i_3th symbols of the input x, respectively, then the head T_1 of M_3 will be pointing to the i_1th symbol of the i_2th subsubsegment of the i_3th subsegment of $f_2(x)$ (this symbol contains x_{i_1}, x_{i_2} and x_{i_3} on its first, second and third tracks, respectively). To simulate one step of M_2, M_3 works as follows: If M_2 moves its head S_1 (S_2, S_3) one square to the right, then M_3 moves its head T_1 one square (n squares, n^2 squares) to the right. The head T_2 is used as a counter to count these numbers n and n^2. It is easy to check that M_3 simulates M_2 correctly. Moreover, the function $f_2(x)$ can be computed by a circuit family in DEPTH(log n). □

Now we can prove our main theorem.

Theorem 3.5. 1-2-*DFA is* $\le_{\mathrm{m}}^{\mathrm{LD}}$*-complete for* DSPACE(log n).

Proof. To prove that 1-2-DFA is in DSPACE($\log n$) is easy: Given an input $x\#M$, the Turing machine T first checks whether M encodes a one-way two-head deterministic finite automaton, then simulates M on input x step by step. T only needs to store a triple (s, p_1, p_2) in the simulation, where s is the current state of M and p_1 and p_2 are the current positions of the two heads of M, respectively. Under reasonable assumptions, it can be seen that T runs in $O(\log n)$ space.

Let L be a language in DSPACE($\log n$). By our Lemmas 3.2–3.4, there exist a function f_3 ($f_3 = f_2 \circ f_1$), which can be computed by a circuit family in DEPTH($\log n$), and a one-way two-head deterministic finite automaton M such that for any x, M accepts $f_3(x)$ if and only if $x \in L$. Thus the function $f: x \to f_3(x)\#M$ is a witness to the fact $L \le_m^{LD}$ 1-2-DFA. □

It is interesting to note that there is a fixed one-way two-head deterministic finite automaton M whose computation is the "hardest" with respect to the corresponding parallel implementation.

Theorem 3.6. *There is a one-way two-head deterministic finite automaton M_d such that if the language accepted by M_d is in* DEPTH($\log^{1+\varepsilon}(n)$) *for some $\varepsilon \ge 0$, then*

$$\text{DSPACE}(\log n) \subseteq \text{DEPTH}(\log^{1+\varepsilon}(n)).$$

Proof. Since the 1-2-DFA is in DSPACE($\log n$), there exist a function F_1, which can be computed by a circuit family in DEPTH($\log n$), and a one-way two-head deterministic finite automaton M_d such that for any x, M_d accepts $F_1(x)$ if and only if $x \in$ 1-2-DFA. Now for a given language L in DSPACE($\log n$), since 1-2-DFA is $\le_m^{LD}$-complete for DSPACE($\log n$), there is a function F_2, which can be computed by a circuit family in DEPTH($\log n$), such that for any x, $x \in L$ if and only if $F_2(x) \in$ 1-2-DFA. Combining the functions F_1 and F_2, we conclude that for any x, $x \in L$ if and only if $F_1(F_2(x))$ is accepted by the one-way two-head deterministic finite automaton M_d. It is obvious that the function $F = F_1 \circ F_2$ can be computed by a circuit family in DEPTH($\log n$). □

We will call the M_d in Theorem 3.6 *basic one-way two-head deterministic finite automaton.*

From Theorem 3.6, we can easily show a graph problem which is $\le_m^{LD}$-complete for DSPACE($\log n$).

We suppose that graphs are represented by adjacency matrices. An acyclic directed graph is a *tree* if it is connected and each node of it has outdegree at most 1. A *forest* is a directed graph such that all of its connected components are trees. We define the *Forest accessibility problem* (*FAP*) as follows:

$$\text{FAP} = \{F\#n_1\#n_2 \mid \text{in the forest } F \text{ there is a directed path from node } n_1 \text{ to node } n_2\}.$$

It is easy to prove the following lemma.

Lemma 3.7. *A directed graph G is a forest if and only if the following two conditions hold*:

(1) *each node in G has outdegree at most* 1;
(2) *no cycles exist in G.*

Theorem 3.8. *FAP is* $\leq_{\mathrm{m}}^{\mathrm{LD}}$*-complete for* DSPACE(log *n*).

Proof. To see that FAP is in DSPACE(log n), we use the following algorithm: Given an input $F\#n_1\#n_2$, we can test if F satisfies conditions (1) and (2) in Lemma 3.7 in O(log n) space: condition (1) is easy. To make sure condition (2) is satisfied assuming (1) is satisfied, we go from each node N in F at most $n+1$ steps (where n is the number of nodes in F) and see if N can be reached again. If there is a cycle in F, then clearly there is a node N in F from which we can reach it again in at most $n+1$ steps. When F is a forest, we go from node n_1 and see if we can reach node n_2, then accept or reject accordingly.

Now we prove that FAP is $\leq_{\mathrm{m}}^{\mathrm{LD}}$-hard for DSPACE(log n). Let M_d be the basic one-way two-head deterministic finite automaton and let $L\in$ DSPACE(log n). By Theorem 3.6, there is a function F, which can be computed by a circuit family in DEPTH(log n) such that $x\in L$ if and only if $F(x)$ is accepted by M_d. Each configuration C of M_d can be represented as (q,p_1,p_2) where q is a state of M_d, p_1 and p_2 are the head positions of head 1 and head 2, respectively. Given an input $F(x)$, the connections of configurations of M_d are fixed: There is an edge from configuration C_1 to configuration C_2 if and only if C_1 is a succeeding configuration of C_2. For this fixed input, all the configurations and the connections above form a forest FOREST since M_d is one-way and deterministic. M_d accepts $F(x)$ if and only if there is a directed path in FOREST from the accepting configuration to the initial configuration. Moreover, the function $F_1:F(x)\to$ FOREST can be computed by a circuit family in DEPTH(log n). This completes our proof. □

Remark. In [3] it is independently proved that FAP is $\leq_{\mathrm{T}}^{\mathrm{LD}}$-complete for DSPACE(log n).

4. Remarks

We have shown that the computation performed by one-way two-head finite automata is the "hardest" computation in DSPACE(log n) with respect to parallel implementation. This fact gives us an evidence that there is an intrinsic difference between one-head finite automata and multihead finite automata, as we will discuss in next few paragraphes.

In 1965, Rosenberg [9] claimed that for any integer $k\geq 1$, one-way $(k+1)$-head finite automata are strictly stronger than one-way k-head finite automata, i.e., for each k, there is a language L_k which can be accepted by a one-way $(k+1)$-head finite automaton but cannot be accepted by any one-way k-head finite automata. Rosenberg's conjecture was finally proved by Yao and Rivest [11]. The similar results were also obtained for two-way finite automata by Ibarra [6]. However, our Theorem 3.6 and Lemma 3.2 tell us that there is no intrinsic difference between two-head finite automata and k-head finite automata for $k \geq 2$ with respect to their parallel implementations.

On the other hand, Ladner and Fischer have given a $(\log n)$-depth parallel implementation for one-head finite automata [8]. If we believe that DSPACE$(\log n) \neq$ DEPTH$(\log n)$, then our Theorem 3.6 shows that it takes more parallel time to simulate two-head finite automata than to simulate one-head finite automata.

References

[1] A.E. Borodin, On relating time and space to size and depth, SIAM J. Comput. 6 (1977) 733–743.
[2] S.A. Cook, A taxonomy of problems with fast parallel algorithms, Inform. and Control 64 (1985) 2–22.
[3] S.A. Cook and P. McKenzie, Problems complete for deterministic logarithmic space, J. Algorithms 8(3) (1987) 385–394.
[4] M.J. Fischer and N.J. Pippenger, Relations among complexity measures, J. ACM 26(2) (1979) 361–381.
[5] J. Hartmanis, On non-determinacy in simple computing devices, Acta Inform. 1 (1972) 336–344.
[6] O.H. Ibarra, On two-way multihead automata, J. Comput. System Sci. 7 (1973) 28–36.
[7] R.E. Ladner, N.A. Lynch and A.L. Selman, A comparison of polynomial time reducibilities, Theoret. Comput. Sci. 1 (1975) 103–123.
[8] R.E. Ladner and M.J. Fischer, Parallel prefix computation, J.ACM 27 (1980) 831–838.
[9] A.L. Rosenberg, On multihead finite automata, IBM J. Res. Develop. 10 (1966) 388–394.
[10] C.B. Wilson, Parallel computation and the NC hierarchy relativized, in: Proceedings First Structure in Complexity Theory Conference (1986) 362–382.
[11] A.C. Yao and R.L. Rivest, $k+1$ heads are better than k, J. ACM 25 (1978) 337–340.