SIAM J. COMPUT.
Vol. 11, No. 3, August 1982


0097-5397/82/1103-0019 $01.00/0



# ON SOME DETERMINISTIC SPACE COMPLEXITY PROBLEMS*

JIA-WEI HONG†

**Abstract.** In this paper we give a complete problem in DSPACE($n$). The problem is whether there exists a cycle in the connected component containing $(0, 0, \cdots, 0)$ in the graph $G_p$ of the zeros of a polynomial $P$ over $GF(2)$ under a suitable natural coding. Hence the deterministic space complexity of this problem is $O(n)$ but not $o(n)$. We give as well several problems for which we can obtain very close upper and lower deterministic space bounds. For example, the deterministic space complexity to determine whether there exists a cycle in the graph of the set of assignments satisfying a Boolean formula is $O(n/\log n)$ but not $o(n/\log^2 n)$.

**Key words.** space complexity, deterministic space complete problem, lower bounds, cycle-free problem, set of assignments, Boolean expression, polynomial over $GF(2)$

**1. A typical theorem.** The purpose of this paper is to find some "natural problems" which are complete in DSPACE ($n$), or at least for which we can obtain very close upper and lower bounds of deterministic space complexity. It is well known that there exists a hardest language in NSPACE ($n$) [1], but we do not know any complete language in DSPACE ($n$), except the universal one. In this paper, we will give a "natural problem" which is complete in DSPACE ($n$). People have obtained several results about lower space bounds, but the bounds apply not only to deterministic Turing machines but also to nondeterministic Turing machines (see [2]–[4]). So the upper bounds are the squares of the lower bounds. In order to obtain close upper and lower space complexity bounds, we have to use some methods that can only be used in a deterministic situation.

Consider the following problem: Let $F$ be a Boolean formula constructed from $m$ variables $x_1, x_2, \cdots, x_m$. The distance of two assignments $Y = (y_1, \cdots, y_m)$ and $Z = (z_1, \cdots, z_m)$ (two $m$-tuples of zeros and ones) is defined as

$$d(Y, Z) = \sum_{i=1}^{m} |y_i - z_i|,$$

that is, the number of indices at which they differ. Set

$$V_F = \{(x_1, \cdots, x_m) | F(x_1, \cdots, x_m) = 1\},$$
$$E_F = \{(Y, Z) \mid Y, Z \in V_F, d(Y, Z) = 1\}.$$

Given a Boolean formula $F$, there is a graph $G_F = \{V_F, E_F\}$. We want to determine whether there exists a cycle in $G_F$, and to determine the space complexity of solving this problem. A typical result is the following.

THEOREM 1. *The deterministic space complexity to determine whether there exists a cycle in graph $G_F$ is $O(n/\log n)$ but not $o(n/\log^2 n)$, where $n$ is the length of the binary expression of $F$.*

**2. Proof of the theorem.** The theorem depends on the following

LEMMA. *For every Turing machine $M$, there exists a Turing machine $R$, whose input is a binary string $W = w_1 w_2 \cdots w_l$, whose output is a binary coding $F^*$ of a*

* Received by the editors April 30, 1980, and in final form July 15, 1981. Significant portions of this paper are reprinted with permission from "On Some Deterministic Space Complexity Problems" by Hong Jia-wei published in Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing, Copyright 1980, Association for Computing Machinery, Inc.

† Peking Municipal Computing Centre, Peking, China, and Department of Computer Science, University of Toronto, Toronto, Ontario, Canada.

*Boolean formula $F$ in $t$ variables, such that*

1) *The length of the work tape of $R$ is $o(l)$.*

2) *$M$ accepts $W$ in space $l$ iff there is a cycle in $G_F$.*

3) *The length of $F^*$ is $O(l \log^2 l)$.*

Using this lemma, we can prove the theorem as follows. Suppose that $g$ is a space-constructable function such that $g(n \log^2 n) = o(n)$. Suppose that there were a Turing machine $T$ which can determine whether there exists a cycle in graph $G_F$ in space $g(n)$. Let $M$ be an arbitrary space linear bounded Turing machine. We construct a Turing machine $S$ as follows:

1) The input of $S$ is $W = w_1 w_2 \cdots w_l$, placed on a read-only tape.

2) Using $W$ as input, simulate $R$ on the work tape; calculating the coding $F^*$ of the formula $F$, the work space is $o(l)$.

3) Simulate $T$, using $F^*$ as input, determining whether there exists a cycle in graph $G_F$.

The work space is $g(n)$, where $n$ is the length of $F^*$. Therefore, $g(n) \leqq g(cl \log^2 l) \leqq g(cl \log^2 (cl)) = o(cl) = o(l)$ for some constant $c$. Although the length of $F^*$ is $cl \log^2 l$, we do not store $F^*$ in step 2), but calculate every digit of $F^*$ from the very beginning. Hence the work space is $o(l)$.

Now, $S$ would accept in space $o(l)$ a language that is accepted by $M$ in space $l$. This is impossible because $M$ is an arbitrary space linear bounded Turing machine (see [5]).

If we take $g(n) = o(n/\log^2 n)$, then $g(n) \log^2 n/n \to 0$ $(n \to \infty)$. Hence, substituting $n \log^2 n$ for $n$, we obtain

$$\frac{g(n \log^2 n) \log^2 (n \log^2 n)}{n \log^2 n} \to 0 \qquad (n \to \infty).$$

Because of $\log^2 (n \log^2 n) \geqq \log^2 n$, we must have $g(n \log^2 n)/n \to 0$ $(n \to \infty)$; that is, $g(n \log^2 n) = o(n)$. This completes the proof of the lower bound.

Suppose $G$ is a graph and the number of vertices of $G$ is not more than $2^r$, so every vertex in $G$ has a coding whose length is not more than $r$. If vertex $Y$ has $k$ neighbors, we can define which one is its first neighbor, which is the second and which is the last, according to the coding's order. We define a Cycle-Search-Procedure as follows: When we come to a vertex $Y$, if the vertex $Y_1$ we just come from is the $i$th neighbor of $Y$, then we go to the $(i+1)$th neighbor of $Y$; if $Y_1$ is $Y$'s last neighbor, then we go to the first neighbor of $Y$. Using two pebbles, we can do a Cycle-Search on an undirected graph. We start from vertex $Y$, and go to its first neighbor $Y_1$. Then, if $Y$ is $Y_1$'s $i$th neighbor, we go to $Y_1$'s $(i+1)$th neighbor, and so forth. Now, if the connected area containing $Y$ is a tree, then 1) we must eventually come back to $Y$ from its last neighbor and begin another period, and 2) in one period, from every vertex we visit, we go to its every neighbor exactly once according to a cyclic order. And, it is not difficult to prove that these two conditions are sufficient to guarantee that the connected area is a tree.

The algorithm is from a Chinese story. There are three Chinese, the father, the son and the grandfather, in a large maze. They have only a clock, but want to determine whether there is a cycle in the maze. Standing at a point, the father lets his son do the Cycle-Search. If his son always comes back to the point from the direction the son goes, the father knows that there is no problem with this point. Then he will go one step ahead according to the Cycle-Search and let his son do the whole Cycle-Search again. Therefore the son is very busy. The grandfather is too old to move. He just watches the clock. If his son and grandson have spent too much time, he knows there must be a cycle. To simulate these three Chinese, logarithmic space is enough.

In the following program, $P_1$ and $P_2$ are pairs of pebbles; CSP($P$) means to put the pair $P$ of pebbles a step ahead according to Cycle-Search-Procedure. This program can determine whether there exists a cycle in graph $G$.

```
for every X in G do {every time change X according to the order of its coding}
  begin put P1 at X; {use P1 to check condition 2) at different places}
    while P1 has not come back to X from its last neighbor do
    begin put P2 at X;
      0 → j;
    while P2 has not come back to X from its last neighbor do
        begin j+1 → j;
          if j > 2^r then go to Cyc; {there must be a cycle}
        if P2 coincides with P1, then check condition 2) if is not satisfied then
             go to Cyc; {use another pebble to do so}
        CSP(P2)
        end
        CSP(P1)
    end
  end stop; (no cycle)
Cyc: stop (cycle);
```

For the first line, one pebble is enough. In order to check condition 2), another pebble will do. Therefore, we need 6 pebbles altogether, and each pebble uses space $\log 2^r = r$. For counting, we need space $r$. Hence the total work space is $O(r)$. As to our problem, the length of the binary expression of the formula is $n$, so we have $r \log r \leqq n$, $r \log n/n = (r \log r/n)(\log n/\log r) = O(1)$. Therefore $r = O(n/\log n)$; this is the upper space bound.

**3. The construction of the admissable set.** Without loss of generality, we can suppose that the input of the linear bounded Turing machine $M$ is $W = w_1 \cdots w_l$, that the head will never move to the outside, that the head moves either right or left in every step and that there is only one acceptable instantaneous description $I_t$. Suppose $\delta$ is the transition function of $M$ and $I = a_1 a_2 \cdots qa_i \cdots a_l$ is an instaneous description (I.D.) saying that machine $M$ is in the state $q$, scanning the $i$th square. If $\delta(q, a_i) = (q', a_i', -1)$, we should substitute $q'a_{i-1}a_i'$ for $a_{i-1}qa_i$ in $I$; if $\delta(q, a_i) = (q', a_i', 1)$, we should substitute $a_{i-1}a_i'q'$ for $a_{i-1}qa_i$.

In the following, if the number of 1's in a coding is even, we say the coding is even; otherwise the coding is odd. Suppose that the number of different states and the number of different letters in the alphabet are all less than or equal to $2^{s-4}$, so we can use an $s$-digit binary number to express the states and letters such that the last three digits of the letter's coding are 000 and the last three digits of the state's coding are 111, and every coding is even. These three last digits are called character codings, and the first $s-3$ digits are called information codings. To obtain the coding of the I.D. $I$, for every letter and state in $I$, we substitute its coding, and then add $d$ zeros at the end (called complement coding), where $d$ is the number of different substitutions in $M$. The length of the coding $I^*$ of $I$ is $(l+1)s + d$.

For example, suppose there is a substitution $aqb \to q'ab'$, and the codings of $a$, $b$, $b'$, $q'$, $q$ are 011000, 000000, 110000, 100111, 010111 respectively. Then the substitution becomes

$$\left|\begin{matrix} 011000 & 010111 & 000000 \\ 100111 & 011000 & 110000 \end{matrix}\right|.$$

In $G_F$, this substitution is realized by the "path" in Table 1.

TABLE 1

| | | | | $s_1$ | $s_2$ | $s_3$ |
|---|---|---|---|---|---|---|
| $L_{ij1}$ | 011000 | 010111 | 000000 | 0 | 0 | 0 |
| $L_{ij2}$ | 011000 | 010111 | 000000 | 1 | 0 | 0 |
| $L_{ij3}$ | 111000 | 010111 | 000000 | 1 | 0 | 0 |
| $L_{ij4}$ | 101000 | 010111 | 000000 | 1 | 0 | 0 |
| $L_{ij5}$ | 100000 | 010111 | 000000 | 1 | 0 | 0 |
| $L_{ij6}$ | 100000 | 011111 | 000000 | 1 | 0 | 0 |
| $L_{ij7}$ | 100000 | 011111 | 100000 | 1 | 0 | 0 |
| $L_{ij8}$ | 100000 | 011111 | 110000 | 1 | 0 | 0 |
| $L_{ij9}$ | 100001 | 011111 | 110000 | 1 | 0 | 0 |
| $L_{ij10}$ | 100011 | 011111 | 110000 | 1 | 0 | 0 |
| $L_{ij11}$ | 100111 | 011111 | 110000 | 1 | 0 | 0 |
| $L_{ij12}$ | 100111 | 011110 | 110000 | 1 | 0 | 0 |
| $L_{ij13}$ | 100111 | 011110 | 110000 | 0 | 0 | 0 |
| $L_{ij14}$ | 100111 | 011100 | 110000 | 0 | 0 | 0 |

In this table, two successive lines are different from each other only in one digit. It realizes the substitution by the following steps:

1) There are $d$ digits $(s_1, s_2, \cdots, s_d)$ corresponding to the substitutions in machine $M$. Suppose this is the $j$th substitution; then we change the value of $s_j$ from 0 to 1, so that we can separate this path from the paths corresponding to other substitutions. In this example we have supposed that $d=3$ and $j=1$.

2) From left to right, change the digits one by one. Leave the character codings unchanged. In this example it takes 6 lines to accomplish this procedure.

3) Change the character codings of $q'$ from 000 to 001 to 011 to 111, and then change the original character coding of $q$ from 111 to 110.

4) Abolish the 1 at the position $s_j$.

5) Change the character coding of $q$ from 110 to 100, so the last line can be connected with the first line of the next substitution. Notice the position changed is $x_{is-1}$.

According to this procedure, the string is changed from an even coding to an odd coding, then to an even coding again, $\cdots$ and so forth. Generally speaking, there are $k$ lines in the table, $k$ is even and is a function of $j$. To simplify the discussion below, we suppose $k$ is a constant.

There are $(l+1)s+d$ digits in the coding of an instantaneous description. We use variables $x_1, x_2, \cdots x_{(l+1)s}$ and $s_1, \cdots, s_d$ to express their values. Suppose the head is scanning the $i$th square. Then the columns of the table correspond to $x_{(i-2)s+1}, \cdots, x_{(i+1)s}, s_1, \cdots, s_d$. Hence for every position $i$ and every substitution $j$ there is a table. For every line $L_{iju}$, we construct a conjunction $C'_{iju}$ as follows: it is a conjunction of $3s+d$ variables $x_{(i-2)s+1}, \cdots, x_{(i+1)s}, s_1, \cdots, s_d$ (later we call them the group i); but if the value of the variable in that line $L_{iju}$ is 0, we should put a negation sign on it. In this example, we have $C'_{311}=\bar{x}_7x_8x_9\bar{x}_{10}\bar{x}_{11}\bar{x}_{12}\bar{x}_{13}x_{14}\bar{x}_{15}x_{16}x_{17}x_{18}\bar{x}_{19}\bar{x}_{20}\bar{x}_{21}\bar{x}_{22}\bar{x}_{23}\bar{x}_{24}\bar{S}_1\bar{S}_2\bar{S}_3$. We define $C_{iju}=\bar{x}_{(i-2)s-2}\bar{x}_{(i-2)s-1}\bar{x}_{(i-2)s}C'_{iju}\bar{x}_{(i+2)s-2}\bar{x}_{(i+2)s-1}\bar{x}_{(i+2)s}$. These 6 variables correspond to the $(i-2)$ and $(i+2)$th character codings. Set

$$D_u=\begin{cases} x_1\oplus\cdots\oplus x_{(l+1)s}\oplus s_1\oplus\cdots\oplus s_d & \text{if } u \text{ is odd,}\\ x_1\oplus\cdots\oplus x_{(l+1)s}\oplus s_1\oplus\cdots\oplus s_d\oplus 1 & \text{if } u \text{ is even,}\end{cases}$$

where $\oplus$ means exclusive or. $D_u=1$ means the coding is even, iff $u$ is even. Set

$$D_{iju}=C_{iju}D_u,$$

$$D=\bigcup_{iju} D_{iju}=\bigcup_u D_u\Big(\bigcup_{ij} C_{iju}\Big)=\Big(D_1\Big(\bigcup_{\substack{ij\\ 2\nmid u}} C_{iju}\Big)\Big)\cup\Big(D_2\Big(\bigcup_{\substack{ij\\ 2\mid u}} C_{iju}\Big)\Big).$$

The binary length of formula $D$ is $O(l\log l)$.

*Remark.* Here we use three operations $\vee$, $\wedge$, $\oplus$ to construct the formula. We cannot express $D_u$ in length $O(l\log l)$ if we only use the usual operations $\neg$, $\vee$, $\wedge$. Anyway, it is easy to see that in this procedure at most one coding of a group is odd and all the others are even. Hence when $\bigoplus_{i=1}^{(l+1)s} x_i$ is even, we can use

$$E=\bigcap_{i=1}^{l}\Big(\neg\bigoplus_{h=1-s}^{2s} x_{is+h}\Big)$$

instead of it. The length of $E$ is $O(l\log l)$. When $\bigoplus_{i=1}^{(l+1)s} x_i$ is odd, we can use

$$E'=\bigcup_{g=1}^{l}\Big(\bigcap_{\substack{i=1\\ i\neq g}}^{l}\Big(\neg\bigoplus_{h=1-s}^{2s} x_{is+h}\Big)\cap\Big(\bigoplus_{h=1-s}^{2s} x_{gs+h}\Big)\Big)$$

instead. Its binary length is $O(l^2\log l)$. Here $s$ is a constant. Using "divide and conquer" (see [4, Chapt. 2]), after collecting common factors, its length can be reduced to $O(l\log^2 l)$.

In order to get rid of meaningless binary strings, we define an admissible set $A$. A binary string $Y$ belongs to $A$ iff

1) There is at most one complement coding digit which equals 1, and

2) $Y\in D_{iju}$ for some $u=1,2,\cdots,k-6$ and there is only one character coding which equals 111 (all the others equal 000); or $Y\in D_{ijk-5}$ ($Y\in D_{ijk-4}$, $Y\in D_{ijk-3}$, $Y\in D_{ijk-2}\cup D_{ijk-1}$, $Y\in D_{ijk}$ respectively) and there is only one pair of successive character codings which equal 001 and 111, (011 and 111, 111 and 111, 111 and 110, 111 and 100, respectively), and the others are equal to 000.

We can express set $A$ with a Boolean formula whose binary length is $O(l\log^2 l)$. For example, the sentence, "There is only one character coding which equals 111, all the others equal 000", can be expressed with a formula of binary length $O(l^2\log l)$. Using "divide and conquer", after collecting common factors, its length can be reduced to $O(l\log^2 l)$.

**4. The proof of the lemma.** Because $M$ is deterministic, for every admissible I.D. (i.e., whose coding is in $A$) $I_1$, we can use at most one substitution of $M$ such that $I_1\to I_2$, and this defines a directed path in set $A$. All these paths make set $A$ a directed graph $\vec{A}$.

More precisely, we say $\alpha\to\beta$ iff 1) there exists $D_{iju}$ such that $\alpha\in D_{iju}$, $\beta\in D_{iju+1}$, $\alpha$ and $\beta$ are adjacent in $A$; or 2) there exists $D_{ijk}$ such that $\alpha\in D_{ijk}\cap A$, and $\beta$ is obtained by changing the value of the position $x_{is-2}$ of $\alpha$ from 1 to 0.

This directed graph $\vec{A}$ satisfies

1) For every point in $\vec{A}$, the fan-out number is at most one. This is because $M$ is deterministic and all the paths realizing these substitutions are separated by the design of the set $A$.

We are going to prove this statement. (We suggest the reader ignore this paragraph at the first reading.) Suppose $\alpha \to \beta_1$, $\alpha \to \beta_2$, then there exist $D_{iju}$ and $D_{abc}$ satisfying the above condition. If $u \in \{2, 3, \cdots, k-2\}$, then $j = b$. Hence $i = a$ and $u = c$. In this case, $D_{iju+1}$ is different from $D_{iju}$ in one position, and $\beta_1$, $\beta_2$ are different from $\alpha$ at one position, so $\beta_1$ and $\beta_2$ are different from $\alpha$ at the same position; therefore $\beta_1 = \beta_2$. If $u = 1$, then $i = a$. In this case, because the Turing machine is deterministic, there is only one substitution we can use. Hence $\beta_1 = \beta_2$. If $u = k-1$, then $i = a$. $\beta_1$ and $\beta_2$ are different from $\alpha$ at position $x_{i_s-1}$, therefore $\beta_1 = \beta_2$. If $u = k$, then $i = a$, $c = u = k$. There exists $D_{ijk}$ such that $\alpha \in A \cap D_{ijk}$ and $\beta_1$, $\beta_2$ are obtained by changing the value of the position $x_{i_s-2}$ from 1 to 0; hence $\beta_1 = \beta_2$.

2) An I.D. $I_1$ leads to an I.D. $I_2$ iff there is a path from the coding of $I_1$ to the coding of $I_2$ in $\vec{A}$.

Now, if $\alpha, \beta \in \vec{A}$ and there is an arrow from $\alpha$ to $\beta$, then we write $\alpha \to \beta$ or $\beta \leftarrow \alpha$. In the same time, $A$ is a set of codings. Two codings are adjacent if they are different in only one digit. Hence $A$ is a undirected graph. What is the relation between $A$ and $\vec{A}$? We have

PROPOSITION 1. *$\alpha$ and $\beta$ are adjacent in $A$ iff $\alpha \to \beta$ or $\beta \to \alpha$ in $\vec{A}$.*

*Proof.* In fact, we are going to prove the following stronger proposition: If $\alpha \in A$, $\beta \in D$, $d(\alpha, \beta) = 1$, then $\beta \in A$ and $\alpha \to \beta$ or $\beta \to \alpha$ in $\vec{A}$. Hence $A$ is an isolated part of graph $D$. (We suggest the reader ignore this proof at the first reading.)

Suppose $\alpha \in A$, $\beta \in D$, $d(\alpha, \beta) = 1$, $\alpha \in D_{iju}$, $\beta \in D_{abc}$. If $|i-a| \geqq 2$, then $d(\alpha, \beta) \geqq 2$. Therefore, we need only consider the following two cases.

*Case* 1. $i = a$. Because one of $\alpha$ and $\beta$ is even and the other is odd, $u \neq c$. Therefore, they are different at just one position in the group $i$.

If $j = b$, then $d(\alpha, \beta) \geqq \min\{|u-c|, 2\}$ by the structure of the table. Hence $|u-c| = 1$ and $\alpha \to \beta$ or $\beta \to \alpha$, $\beta \in A$.

Now suppose $j \neq b$. If $u$ and $c$ both belong to $\{2, 3, \cdots, k-2\}$, then $\alpha$ and $\beta$ are different at position $s_j$ and $s_b$. This is impossible, so we can assume $u = 1$, $k-1$, $k$. (If $c = 1$, $k-1$, $k$, we can treat it the same way.)

If $c \in \{2, 3, \cdots, k-2\}$, $\alpha$ and $\beta$ are different at $s_h$ $(h = 1, 2, \cdots, d)$, so they are the same at other positions. Hence $u \neq k$. Therefore $u = k-1$ or 1. If $u = k-1$, we must have $c = k-2$ and $\beta \to \alpha$, $\beta \in A$. If $u = 1$, then because $\alpha$ and $\beta$ are the same at all other positions, $\beta \in D_{ib2}$. Because of $\alpha \in D_{ij1} = D_{ib1}$, $\alpha \to \beta$ and $\beta \in A$.

Now suppose both $u$ and $c$ belong to $\{1, k-1, k\}$. In this case, we must have $u = k$, $c = k-1$, or $c = k$, $u = k-1$. Hence $\alpha \to \beta$ or $\beta \to \alpha$. $\beta \in A$.

*Case* 2. $|i-a| = 1$. Suppose $i = a+1$.

*Subcase* 1. $u, c \in \{2, 3, \cdots, k-2\}$. We must have $b = j$. That means the corresponding substitutions are the same; especially, the heads move in the same direction (say left). Then at the positions of the $i$th character coding, $\alpha$ has 2 1's at least, but $\beta$ has none. This is impossible.

*Subcase* 2. $u \in \{1, k-1, k\}$, $c \in \{2, 3, \cdots, k-2\}$. Then $\alpha$ and $\beta$ should be the same at the position $x$'s. Compare the pairs of two successive character codings (the $i$th and the $a$th) of $\alpha$ and $\beta$. They cannot be the same.

*Subcase* 3. $u \in \{2, 3, \cdots, k-2\}$, $c \in \{1, k-1, k\}$. This subcase is similar to subcase 2.

*Subcase* 4. $u, c \in \{1, k-1, k\}$. Compare the pairs of two successive character codings (the $i$th and $a$th) of $\alpha$ and $\beta$. They have at most one position different. We must have $u, c \neq k-1$. Hence $u = k$, $c = 1$, $\alpha \to \beta$ or $c = k$, $u = 1$, $\beta \to \alpha$, $\beta \in A$.

PROPOSITION 2. *Suppose $G$ is an undirected graph. If we can define a direction on every edge of $G$ such that the fan-out number of any point of the directed graph $\vec{G}$*

*is at most one, and if $\beta$ is a terminal point of $\vec{G}$, $\alpha \in \vec{G}$, then $\alpha$ is connected with $\beta$ in $G$ iff there is a directed path from $\alpha$ to $\beta$ in $\vec{G}$. Furthermore, there is a cycle in $G$ iff there is a cycle in $\vec{G}$.*

*Proof.* If $\alpha$ is connected with $\beta$, then there is an undirected chain from $\alpha$ to $\beta$; $\alpha = \alpha_1, \alpha_2, \cdots, \alpha_n = \beta$. Because $\beta$ is a terminal point, we must have $\alpha_{n-1} \to \alpha_n$. If there were a $\alpha_i$ such that $\alpha_{i-1} \leftarrow \alpha_i$, then we can suppose $i$ is the maximum number such that $\alpha_{i-1} \leftarrow \alpha_i$. Now we would have both $\alpha_{i-1} \leftarrow \alpha_i$ and $\alpha_i \to \alpha_{i+1}$. The fan-out number of $\alpha_i$ is 2 at least. The second part can be proved in the same way.

For every linear bounded Turing machine $M$, we can construct another linear bounded Turing machine $M_1$, which simulates $M$ on the one hand and counts the number of steps simultaneously on the other hand. If the number of steps exceeds the number of states that $M$ and its tape can express, then $M_1$ refuses the input and stops. With a little skill, the reader can construct $M_1$ such that no matter which admissible instantaneous description it starts from (this I.D. may never be reached, if $M_1$ starts from $I_0 = q_0 w_1 \cdots w_l$), $M_1$ will stop eventually. Therefore, we can assume that $M$ itself has this property.

PROPOSITION 3. *There is no cycle in graph $A$.*

*Proof.* If there is a cycle in $A$, then there is a cycle in the directed graph $\vec{A}$ by Proposition 2. Hence there is a cycle in the computation of $M$, which is impossible.

Without loss of generality, we can suppose there is a unique accepting I.D. $I_t$, and $I_0 = q_0 w_1 \cdots w_l$ is the initial I.D. Let $I_0^*$ and $I_t^*$ be the codings of $I_0$ and $I_t$. From the discussion above, we obtain

PROPOSITION 4. *The linear bounded Turing machine $M$ accepts input $W = w_1 \cdots w_l$ iff $I_0^*$ is connected with $I_t^*$ in the graph $A$.*

Now, we should construct a path connecting $I_0^*$ and $I_t^*$. Instead, we construct a path connecting $I_0^*$ with $(1, 1, \cdots, 1)$ and another path connecting $I_t^*$ with $(1, 1, \cdots, 1)$. We use $C_1 = x_1 x_2 \cdots x_{(l+1)s} s_1 \cdots s_d$ to express point $(1, 1, \cdots, 1)$. In the same way, suppose the conjunctions to express $I_0^*$ and $I_t^*$ are $C(I_0^*)$ and $C(I_t^*)$, respectively.

First, we design a path $G_1$ connecting $I_0^*$ and $(1, 1, \cdots, 1)$ satisfying that there is no cycle in $G_1$ and that the number of total variables in $G_1$ is linear in $l$.

We use the following logical symbol $\geq$ to express "not less than": $x \geq y$ means "not $x$ is false and $y$ is true".

$$x \geq y \equiv 1 \oplus (1 \oplus x) y.$$

Assume $y_1, y_2, \cdots, y_e$ are the variables in $\{x_1, \cdots, x_{(l+1)s}, s_1, \cdots, s_d\}$ which have a negation sign in $C(I_0^*)$; $z_1, z_2, \cdots, z_f$ are those without a negation sign in $C(I_0^*)$. Then

$$G_1 = z_1 z_2 \cdots z_f \bigcap_{i=1}^{e-1} (y_i \geq y_{i+1}).$$

In the set $G_1$, every point has the property that $y_1 \geq y_2 \geq y_3 \geq \cdots \geq y_e$. So there is no cycle in $G_1$. The binary length of $G_1$ is $O(l \log l)$. Notice that the $y_i$'s are $x_i$'s whose value is 0 in $I_0^*$, and nearly every character coding in $I_0^*$ is 000. Hence the distance between the positions of $y_i$ and $y_{i+1}$ in the list $\{x_1, x_2, \cdots, x_{(l+1)s}\}$ is not more than a constant. We will use this property later.

Using the same technique we can design another path $G_2$ connecting $I_t^*$ with $(1, 1, \cdots, 1)$. Set

$$F = A \bar{t}_1 \bar{t}_2 \bar{t}_3 \bar{t}_4 \cup C(I_0^*) t_1 \bar{t}_2 \bar{t}_3 \bar{t}_4 \cup G_1 t_1 t_2 \bar{t}_3 \bar{t}_4$$
$$\cup C_1 (t_1 t_2 t_3 \bar{t}_4 \cup t_1 t_2 t_3 t_4 \cup \bar{t}_1 t_2 t_3 t_4) \cup G_2 \bar{t}_1 \bar{t}_2 t_3 t_4 \cup C(I_t^*) \bar{t}_1 \bar{t}_2 \bar{t}_3 t_4.$$

We have

PROPOSITION 5. *M accepts $W = w_1 \cdots w_l$, iff there is a cycle in the graph $G_F$.*

It is easy to construct the coding $F^*$ of $F$ in space $o(l)$. This completes the proof of the lemma.

**5. The main results.** In this section, we want to improve the result. The key is to reduce the length of $F^*$, which is $O(l \log^2 l)$ in the lemma. The square on the log $l$ comes from two places.

One of them is that if we use only the logical operations $\vee$, $\wedge$, $\neg$, we cannot express $x_1 \oplus x_2 \oplus \cdots \oplus x_l$ in a short form. Therefore we have to use the technique mentioned in the remark. The length becomes $O(l \log^2 l)$. But, if we use the formulae

$$\bar{x} = 1 \oplus x,$$

$$x_1 \vee x_2 \vee \cdots \vee x_n = (x_1 \oplus 1)(x_2 \oplus 1) \cdots (x_n \oplus 1) \oplus 1,$$

we can express $F$ as a polynomial over $GF(2)$. Then the length of $D$ is only $O(l \log l)$.

Another trouble comes from the expression of $A$. But in Proposition 1 we proved that $A$ is an isolated part of $D$. Therefore there is a cycle in the graph $F$ iff there is a cycle in the connected component containing $(1, 1, \cdots, 1)$ (notice that there are many cycles in $D \backslash A$) in the graph

$$\begin{aligned} F_1 = D\bar{t}_1\bar{t}_2\bar{t}_3\bar{t}_4 \cup C(I_0^*)t_1\bar{t}_2\bar{t}_3\bar{t}_4 \cup G_1 t_1 t_2 \bar{t}_3 \bar{t}_4 \\ \cup C_1(t_1t_2t_3\bar{t}_4 \cup t_1t_2t_3t_4 \cup \bar{t}_1t_2t_3t_4) \cup G_2\bar{t}_1\bar{t}_2t_3t_4 \cup C(I_t^*)\bar{t}_1\bar{t}_2\bar{t}_3t_4, \end{aligned}$$

whose binary length is $O(l \log l)$ as a polynomial over $GF(2)$. Therefore, we can remove the square of the $\log l$ from the lower bound. As to the upper bound, we need only to test whether there is a cycle in the connected component containing $(1, 1, \cdots, 1)$, so it is even easier. We exchange 0 and 1 in the coding to get the following

THEOREM 2. *The deterministic space complexity to determine whether there is a cycle in the connected component containing $(0, 0, \cdots, 0)$ in the graph $G_p$ of the zeros of a polynomial $P$ over $GF(2)$ is $O(n/\log n)$ but not $o(n/\log n)$, where $n$ is the binary length of the polynomial $P$.*

Because of the definition of $D$, we have

$$D = D_1\Big(\bigcup_i \Big(\bigcup_{\substack{j \\ 2\nmid u}} C_{iju}\Big)\Big) \cup D_2\Big(\bigcup_i \Big(\bigcup_{\substack{j \\ 2\mid u}} C_{iju}\Big)\Big).$$

So the formula $F_1$ consists of nine "main" parts, that is,

$$D_1, D_2, \bigcup_i \Big(\bigcup_{\substack{j \\ 2\nmid u}} C_{iju}\Big), \quad \bigcup_i \Big(\bigcup_{\substack{j \\ 2\mid u}} C_{iju}\Big), \quad C(I_0^*), \quad C(I_t^*), \quad C_1, \quad G_1, \quad G_2.$$

Only $C(I_0^*)$ and $G_1$ depend on the context of the input $W = w_1 \cdots w_l$; the other parts only depend on the length $l$. To output $F_1$, machine $R$ needs only 9 reversals of its input head. At every reversal, $R$ outputs one part. Machine $R$ need only remember the subscript of the variable output at that time. Therefore, the work space is $O(\log l)$.

Although Theorem 2 is better, we still cannot get a complete problem in DSPACE$(n)$, because the length of the binary expression of $P$ is $O(l \log l)$. We want to reduce the length to linear. Notice that the polynomial has to reflect every bit of the input, so there are $l$ variables in $P$ at least, and the length of the coding of every variable is $\log l$ at least. What can we do? We have to invent a new kind of coding.

Imagine that there is a list of infinite many variables: $L=\{x_1, x_2, \cdots\}$. We use the following three kinds of words to express a string of variables in $L$.

$\Delta$: the first variable $x_1$ in $L$. After the use of this symbol, every variable in $L$ becomes a "new" variable again.

$\Delta 0$: a "new" variable; after the use of it, this variable becomes a "used" one.

$\Delta n$: the same variable as its $n$th left neighbor.

For example, the coding

$$\Delta(\Delta 0 \oplus \Delta 2)(\Delta 2 \oplus \Delta 2)(\Delta \oplus \Delta 0) \oplus \Delta 0 \oplus \Delta 0 \oplus \Delta 0 \oplus \Delta 2$$

has the meaning

$$x_1(x_2 \oplus x_1)(x_2 \oplus x_1)(x_1 \oplus x_2) \oplus x_3 \oplus x_4 \oplus x_5 \oplus x_4.$$

After assuming $L=\{s_1, \cdots, s_d, x_1, \cdots, x_{(l+1)s}, t_1, t_2, t_3, t_4\}$ and rearranging the variables in $F_1$ suitably, it is easy to see that, in every part of the expression $F_1$,

1) the leftmost variable is $s_1$,

2) the distance between two adjacent occurrences of the same variable is not more than a fixed constant.

Therefore, we have a linear length expression of $F_1$ in this $\Delta$ coding system (use 9 $\Delta$'s). Furthermore, when we use machine $R$ to output the coding of $F_1$, we need not remember the subscript used. We only need a finite memory. Now the machine $R$ can be a finite automaton, its input head does only nine reversals and the whole time is linear in $l$. We obtain

THEOREM 3. *Under the $\Delta$ coding system, the problem whether there exists a cycle in the connected component containing $(0, 0, \cdots, 0)$ in the graph $G_p$ of the zeros of a polynomial over $GF(2)$ is complete in* DSPACE($n$) *in the sense that*

1) *This problem is in* DSPACE($n$).
2) *Every problem in* DSPACE($n$) *can be reduced to this problem using an oblivious Turing machine within a constant space, linear time and nine reversals of the input head.*

*Therefore, the deterministic space complexity is $O(n)$ but not $o(n)$.*

The $\Delta$-coding system seems a little bit strange, because we are not used to it. We have seen that if we use ordinary coding in the above problem, we have to use $\log n$ space and $n \log n$ time, and we cannot get a complete problem. That means that the $\Delta$ coding system is even more "natural" than the ordinary one.

**6. Further discussion.** If we discuss the connectivity of two points in a graph, we can get the following lower bounds, but we fail to obtain a close upper bound. The best known upper bounds are the squares of the lower bounds [7].

THEOREM 4. *The deterministic space complexity to determine whether two points $(0, 0, \cdots, 0)$ and $(1, 1, \cdots, 1)$ are connected in a graph $G_p$ of the zeros of a polynomial $P$ over $GF(2)$ is not $o(n/\log n)$ (in the ordinary coding) and is not $o(n)$ (in the $\Delta$ coding system).*

In the situation of DSPACE($\log n$), we can use an adjacency matrix to express an undirected graph. Using the same algorithm, we can show that the cycle-free problem (CFP) for undirected graphs belongs to DSPACE($\log n$). The lower bound is not very interesting, because $\log n$ space is needed to determine whether a coding is legitimate. Regardless, using the same method, under Cook's definition and formation [8], we can prove that the CFP is log depth complete for DSPACE($\log n$). We know Depth($\log n$) $\subseteq$ DSPACE($\log n$) by a theorem of Borodin [9], but we do not know whether Depth($\log n$) $=$ DSPACE($\log n$). Now it holds iff CFP $\in$ Depth($\log n$).

At the end of this paper, we discuss some higher space complexity problems briefly. Let $f(n) \geqq n$ be a space constructable function. We insert several pairs of [ , ] and $\langle\, , \rangle$ into a $\Delta$ coding string. These pairs cannot be nested inside each other, every pair of [ , ] can only contain one variable $\Delta 0$ and there exists at least one variable symbol $\Delta$ between every two adjacent pairs of [ , ]. In this coding system, we only use the following 11 symbols: $\Delta$, $\bar{\Delta}$, $\oplus$, 0, 1, ( , ), $\langle\, , \rangle$, [ , ]. Four binary bits are enough to encode them. Suppose the total number of characters inside all the pairs $\langle\, , \rangle$ is $l$; the meaning of [ , ] is that the context inside each [ , ] should be repeated $f(l)$ times. For example, if $f(1) = 4$, we have

$$\Delta[\oplus\bar{\Delta}0]\oplus\Delta[\Delta 0]\oplus\langle\Delta\rangle = \Delta\oplus\bar{\Delta}0\oplus\bar{\Delta}0\oplus\bar{\Delta}0\oplus\bar{\Delta}0\oplus\Delta\Delta0\Delta0\Delta0\Delta0\oplus\Delta$$

$$= x_1\oplus\bar{x}_2\oplus\bar{x}_3\oplus\bar{x}_4\oplus\bar{x}_5\oplus x_1x_2x_3x_4x_5\oplus x_1.$$

Because the length of $\Delta$ is 1, $f(1) = 4$, the context inside [ , ] should be repeated 4 times. We call this the $\Delta(f)$ coding system.

Suppose

$$\alpha_i \in \{\Delta, \bar{\Delta}, 0, 1, (\,,), [\,,], \oplus\}^{\dagger}, \qquad \beta_i \in \{\Delta, \bar{\Delta}, 0, 1, (\,,), \oplus\}^{\dagger}.$$

We call the word $\alpha_1\langle\beta_1\rangle\alpha_2\langle\beta_2\rangle \cdots \alpha_n\langle\beta_n\rangle$ the $\alpha-\beta$ problem: whether there exists a cycle in the connected component containing $(0, 0, \cdots, 0)$ in the graph $G_p$ of the zeros of $P$, whose $\Delta(f)$ coding is $\alpha_1\langle\beta_1\rangle\alpha_2\langle\beta_2\rangle \cdots \alpha_n\langle\beta_n\rangle$.

Notice that if we write any "main" part of $F_1$ in the $\Delta$ coding system, the same segment will repeat again and again. Therefore, if we use the $\Delta(f)$ coding system, except for $C(I_0^*)$ and $G_1$, all the main parts will become fixed words. If $f(n)$ is the space used, no matter how big $f(n)$ is, or how long the polynomial $P$ is, we can always get a short coding of $P$ linear in $n$. With this in mind, we can prove

PROPOSITION 6. *Suppose $f(n) \geqq n$ is a space constructable function, $M$ is a Turing machine, $W = w_1w_2 \cdots w_n$ is the input. Then there exist words $\alpha_1$, $\alpha_2$, $\alpha_3$, $\beta_1$, $\beta_2$, $\beta_3$ such that*

1) $\alpha_i \in \{\Delta, \bar{\Delta}, 0, 1, (\,,), [\,,], \oplus\}^*$, *$\alpha_i$ only depend on $M$.*
2) $\beta_i \in \{\Delta, \bar{\Delta}, 0, 1, (\,,), \oplus\}^*$, *$\beta_i$ only depend on $W$, and can be obtained by an automaton with input $W$, $|\beta_i| \leqq c|W|$, $c$ is a constant.*
3) *$\alpha_1\langle\beta_1\rangle\alpha_2\langle\beta_2\rangle\alpha_3\langle\beta_3\rangle$ is a $\Delta(f)$ coding of a polynomial $P$.*
4) *$M$ accepts $W$ in* DSPACE$(f(n))$ *iff there exists a cycle in the connected component containing $(0, 0, \cdots, 0)$ in the graph $G_p$ of the zeros of $P$.*

THEOREM 5. *Suppose $f(n)$ is space constructable such that $f(m+n) \geqq f(m) + f(n)$. Then under the $\Delta(f)$ coding system, the $\alpha$–$\beta$ problem is complete in the following sense:*

1) *The $\alpha$–$\beta$ problem belongs to* DSPACE$(f(n))$.
2) *Every problem in* DSPACE$(f(n))$ *can be reduced to an $\alpha$–$\beta$ problem of linear length by a finite automaton within* 3 *reversals.*

*Proof.* 1) Suppose the binary length of $\alpha_1\langle\beta_1\rangle\alpha_2\langle\beta_2\rangle \cdots \alpha_i\langle\beta_i\rangle$ is $n$. $l = |\beta_1| + |\beta_2| + \cdots + |\beta_i| \leqq n$. There are at most $n + f(l) \leqq n + f(n) \leqq 2f(n)$ different variables in the polynomial $P$ whose $\Delta(f)$ coding is $\alpha_1\langle\beta_1\rangle \cdots \alpha_i\langle\beta_i\rangle$. We can construct $f(n)$ and calculate the value of $P$ in space $O(f(n))$. Using the Cycle-Search-Procedure, we can determine the $\alpha$–$\beta$ problem in space $O(f(n))$.

2) Suppose machine $M$ accepts a language in DSPACE$(f(n))$. According to Proposition 6, there exist $\alpha_1$, $\alpha_2$, $\alpha_3$, $\beta_1$, $\beta_2$, $\beta_3$, satisfying the condition, $|\alpha_1\langle\beta_1\rangle\alpha_2\langle\beta_2\rangle\alpha_3\langle\beta_3\rangle| \leqq 3c|W| + c_1$. We can realize this reduction by a finite automaton within 3 reversals of the input head.

COROLLARY. *Under the $\Delta(n^t)$ coding system, the deterministic space complexity of the $\alpha$–$\beta$ problem is $O(n^t)$ but not $o(n^t)$.*

**Acknowledgments.** The author thanks Professor Hao Wang for his kind help, and Professor S. A. Cook for his kind help, his valuable suggestions and careful examination. He also thanks Professor C. Rackoff for his references and discussion.

## REFERENCES

[1] A. R. MEYER, AND L. J. STOCKMEYER, *The equivalence problem for regular expression with squaring requires exponential space*, Proc. 13th IEEE Symposium on Switching and Automata Theory, pp. 125–129.

[2] H. B. HUNT, III, *The equivalence problem for regular expressions with intersection is not polynomial in tape*, TR73-156, Dept. Computer Science, Cornell University, Ithaca, NY, 1973.

[3] L. J. STOCKMEYER, *The complexity of decision problems in automata theory and logic*, Massachusetts Institute of Technology Project MAC, Cambridge, MA, 1974.

[4] J. FERRANTE AND C. W. RACKOFF, *The computational complexity of logical theories*, Lecture Notes in Mathematics 718, Springer-Verlag, New York, 1979.

[5] A. V. AHO, J. E. HOPCROFT AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1975.

[6] J. E. HOPCROFT AND J. D. ULLMAN, *Formal Languages and Their Relation to Automata*, Addison-Wesley, Reading, MA, 1969.

[7] S. A. COOK AND C. W. RACKOFF, *Space lower bounds for maze threadability on restricted machines*, this Journal, 9 (1980), pp. 636–652.

[8] S. A. COOK, *Towards a complexity theory of synchronous parallel computation*, presented at Internationales Symposium über Logik und Algorithmik zu Ehren von Professor Ernst Specker, Zurich, Switzerland, 1980.

[9] A. B. BORODIN, *On relating time and space to size and depth*, this Journal, 6 (1977), pp. 733–744.