

CHARACTERIZATIONS AND COMPUTATIONAL COMPLEXITY OF SYSTOLIC TRELLIS AUTOMATA*

Oscar H. IBARRA and Sam M. KIM

Department of Computer Science, University of Minnesota, Minneapolis, MN 55455, U.S.A.

Communicated by A. Salomaa
Received December 1982
Revised April 1983

Abstract. Systolic trellis automata are simple models for VLSI. We characterize the computing power of these models in terms of Turing machines. The characterizations are useful in proving new results as well as giving simpler proofs of known results. We also derive lower and upper bounds on the computational complexity of the models.

1. Introduction

It is known that systolic (VLSI) systems with simple interconnection networks (e.g., tree-structured, linear, orthogonal, hexagonally interconnected, etc.) can solve nontrivial problems such as language recognitions, matrix manipulations, sorting, etc. [2, 5–8, 14–17]. An important problem in the area of systolic systems design is to develop techniques for increasing systems versatility to cope with larger computing environments. One approach to solve this problem is to design programmable building blocks such that systolic systems based on such components can be programmed to solve large classes of problems and the same building blocks can be used in different systolic architectures. For this approach it is useful to have techniques for analyzing and comparing the computational power of systolic systems built with such programmable devices. Analyzing the power of a systolic system is not so easy because it requires us to think about a large number of simultaneously executing tasks. Moreover, writing a program on such a system is usually quite difficult for largely the same reason. Hence, it would be very useful to have characterizations of these systems in terms of the more familiar sequential machines.

In a series of papers [5–8], Culik et al. introduced and studied some simple programmable models of systolic systems. For the most part, their study concerned systolic triangularly shaped trellis automata (trellis automata, for short) used as language recognizers. The basic structure of a trellis automaton is shown in Fig. 1. The dotted arrows show the control lines for labelling (i.e., programming) the processor nodes in a top-down manner, and the solid lines show the data path.

* This research was supported in part by NSF Grant MCS81-02853.

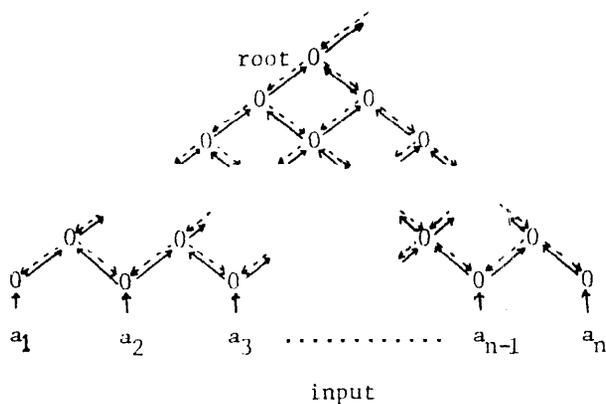


Fig. 1.

Each node computes a finite function on the information from the two input data lines, and outputs the result on its two output lines. (Input nodes use only one of their input lines.) An input string to a trellis automaton is accepted if the root node outputs a distinguished symbol (result). If all nodes have the same label, we call the trellis automaton 'homogenous', otherwise 'regular'.

In this paper we give characterizations of trellis automata in terms of sequential machines which can be easily programmed and analyzed (for their capabilities). For the characterizations, we introduce three Turing machine (TM) models: 'simple TM's', 'oblivious TM's' and 'regular TM's' to characterize variants of trellis automata, i.e., homogeneous trellis automata and regular trellis automata both of deterministic and nondeterministic varieties. (In [5] only deterministic trellis automata were defined.) Using the characterizations, we are able to prove new results as well as give simpler proofs of known results concerning trellis automata. In fact, since the characterizations are such that we can easily convert sequential programs for the TM's into parallel programs for the trellis machines (and vice versa), the TM's can be used as a programming tool for the parallel processors.

An oblivious TM is an on-line TM with a single worktape. The moves are restricted in that the worktape head makes alternate left-to-right and right-to-left sweeps incrementing the range of the sweep by one cell to the right at the end of each left-to-right sweep. An input symbol is read only at the first step of each right-to-left sweep. A simple TM is an oblivious TM with an additional restriction that, during a left-to-right sweep, the machine enters a specific state and the tape contents are not changed. A regular TM is a simple TM augmented by a one-turn pushdown stack on which 'labels' are generated before processing of the input begins. The stack is popped on each right-to-left step.

This paper contains 5 sections including this section. Section 2 formally defines the nondeterministic homogeneous trellis automaton which is a natural extension of the deterministic trellis automaton model defined in [5]. This section also formally defines simple TM and nondeterministic oblivious TM and gives some examples of machine constructions. Section 3 characterizes deterministic and nondeterministic homogeneous trellis automata in terms of deterministic simple TM's and nondeterministic oblivious TM's, respectively. This section also derives lower and upper

bounds on the computational complexities of the trellis models. A result in this section partially answers an open problem in [6]. Section 4 (using the characterization of Section 3) presents new results and gives simple proofs of known results from [5, 6]. Finally, Section 5 formally defines regular trellis automata (deterministic and nondeterministic) and introduces regular TM's which characterize regular trellis automata.

2. Definitions of the models

We begin with the definition of a *nondeterministic homogeneous trellis automaton* (NHTA, for short). In [5] only deterministic trellis automata were considered. Our definition is a natural generalization of the deterministic case.

Definition. An NHTA is a 5-tuple $M = \langle \Sigma, \Gamma, \Delta, f, g \rangle$, where Σ and Γ are the input and internal alphabets, respectively, $\Delta \subseteq \Gamma$ is the output alphabet, $f: \Sigma \rightarrow 2^{\Gamma}$ and $g: \Gamma \times \Gamma \rightarrow 2^{\Delta}$.

A string $x = a_1 \cdots a_n$, $n \geq 1$, each a_i in Σ , is accepted by M if we can construct a weighted directed acyclic graph of the form¹ given in Fig. 2, where

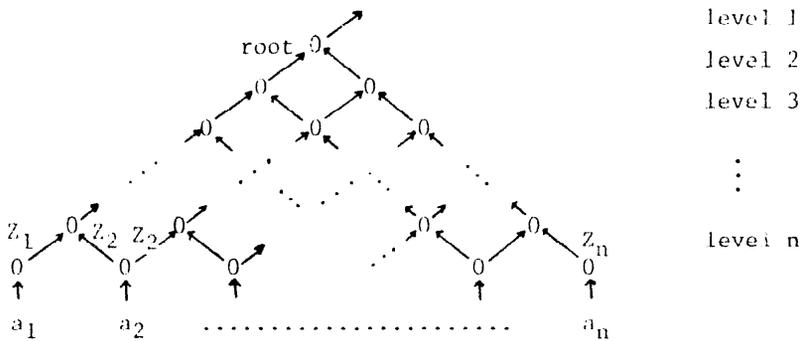
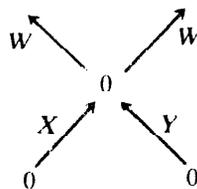


Fig. 2. Computation of an NHTA.

- (1) the edges at level n have weights Z_1, Z_2, \dots, Z_n , where each Z_i is in $f(a_i)$,
- (2) the weights of the other edges are assigned using the mapping g according to the following rule:



is valid if $g(X, Y)$ contains W .

¹ In this paper we mainly deal with the manipulation of the data through the trellis, and we will not show the control lines.

(3) The weight of the edge from the root node is in Δ .

$L(M) = \{x \text{ in } \Sigma^+ \mid x \text{ is accepted by } M\}$ is the language (or set) accepted by M . $L \subseteq \Sigma^+$ is an NHTA-language if there exists an NHTA M such that $L = L(M)$. $\mathcal{L}(\text{NHTA})$ denotes the class of all NHTA-languages.

Definition. An NHTA $M = \langle \Sigma, \Gamma, \Delta, f, g \rangle$ is *deterministic* (DHTA, for short) if f and g are single valued functions, i.e., $f: \Sigma \rightarrow \Gamma$ and $g: \Gamma \times \Gamma \rightarrow \Gamma$.

NHTA's are parallel processors. We now give the definition of sequential processors that will characterize NHTA's and DHTA's.

Definition. A *nondeterministic simple Turing machine* (NSTM, for short) (see Fig. 3) is a 6-tuple $M = \langle Q, \Sigma, \Gamma, \delta, q_0, F \rangle$, where Q, Σ and Γ are the set of states, input alphabet and worktape alphabet, respectively, Γ contains two special symbols $\$$ and λ (for blank). q_0 is the start state and $\delta: Q \times \Gamma \times (\Sigma \cup \{\epsilon\}) \rightarrow 2^{Q \times \Gamma \times \{-1, +1\}}$. Thus M does not write blanks.

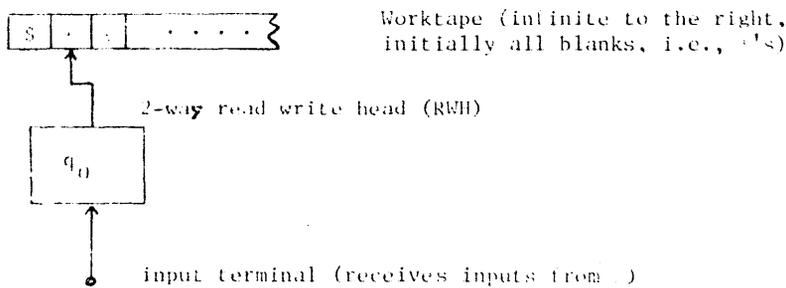


Fig. 3. An NSTM.

δ is restricted as follows. Suppose $\delta(q, Z, a)$ contains (p, Z', d) . Then

- (1) $Z' = \$$ if and only if $Z = \$$.
- (2) If $q = q_0$ and $Z = \lambda$, then $a \neq \epsilon, p \neq q_0, d = -1$.
- (3) If $q = q_0$ and $Z \neq \lambda$, then $a = \epsilon, p = q_0, Z' = Z, d = +1$.
- (4) If $q \neq q_0$ and $Z \neq \$$, then $a = \epsilon, p \neq q_0, d = -1$.
- (5) If $q \neq q_0$ and $Z = \$$, then $a = \epsilon, p = q_0, Z' = \$, d = +1$.

Restrictions (1)–(5) say that M 's read-write head (RWH) operates as follows: the RWH makes alternate sweeps on the worktape (left-to-right and right-to-left between $\$$ and the leftmost λ). Moreover, on any left-to-right sweep, the machine, upon leaving $\$$, must enter state q_0 and remain in this state without altering the contents of the worktape. The machine reads an input symbol if and only if the RWH is on the leftmost blank.

An input string $x = a_1 \cdots a_n, n \geq 1$, each a_i in Σ , is *accepted* if M reads all of x and enters an accepting state with its RWH on $\$$. $L(M)$ denotes the language or set accepted by M and is called an NSTM-language. $\mathcal{L}(\text{NSTM})$ denotes the class of all NSTM-languages.

Definition. An NSTM $M = \langle Q, \Sigma, \Gamma, \delta, q_0, F \rangle$ is *deterministic* (DSTM, for short) if $|\delta(q, Z, a)| \leq 1$ for all q in Q , Z in Γ , and a in $\Sigma \cup \{\epsilon\}$.

We conclude this section with two examples of DSTM constructions and some easy propositions.

Example 2.1. The language $L = \{0^n 1^{2^n} \mid n \geq 1\}$ is in $\mathcal{L}(\text{DSTM})$. L can easily be accepted by a DSTM M which reads the n 0's and writes n 0's on the worktape. Then it uses this worktape area as a binary counter to check that the number of 1's is 2^n . It is easy to show that $\mathcal{L}(\text{DSTM})$ is closed under intersection and contains all ϵ -free regular sets (see Propositions 2.3 and 2.4 below). Hence, it is sufficient to describe the operation of M on inputs of the form $0^n 1^m$, $n, m \geq 1$. Formally, $M = \langle Q, \Sigma, \Gamma, \delta, q_0, F \rangle$, where $Q = \{q_0, q_1, q_2, q_3, q_4\}$, $\Sigma = \{0, 1\}$, $\Gamma = \{\$, \lambda, 0, 1, \#\}$, $F = \{q_4\}$. The transition function δ is given below (where a is in $\{0, 1, \#\}$):

- | | | |
|---|---|--|
| (1) $\delta(q_0, \lambda, 0) = (q_1, 0, -1),$
(2) $\delta(q_1, 0, \epsilon) = (q_1, 0, -1),$
(3) $\delta(q_1, \$, \epsilon) = (q_0, \$, +1),$
(4) $\delta(q_0, a, \epsilon) = (q_0, a, +1),$ | } | read the 0's and put 0's on the worktape, |
| (5) $\delta(q_0, \lambda, 1) = (q_2, \#, -1),$
(6) $\delta(q_2, \#, \epsilon) = (q_2, \#, -1),$ | } | read a 1, put # on the worktape and move left to the first symbol that is not a #, |
| (7) $\delta(q_2, 0, \epsilon) = (q_3, 1, -1),$
(8) $\delta(q_3, a, \epsilon) = (q_3, a, -1),$
(9) $\delta(q_3, \$, \epsilon) = (q_0, \$, +1),$
(10) $\delta(q_2, 1, \epsilon) = (q_4, 0, -1),$
(11) $\delta(q_4, 1, \epsilon) = (q_4, 0, -1),$
(12) $\delta(q_4, 0, \epsilon) = (q_3, 1, -1).$ | } | increment the counter by 1 and accept if, after incrementing, the count is a power of 2. |

The only accepting state is q_4 . One can easily check that for an input $0^n 1^m$, $n, m \geq 1$, $0^n 1^m$ is accepted by M if and only if $m = 2^n$.

Example 2.2. Let Σ be an alphabet. We can construct a DSTM M accepting $L = \{xx^R \mid x \text{ in } \Sigma^+\}$, where x^R is the reverse of x . Let the input be $a_1 \cdots a_n$. The machine M , while copying the input on the worktape, assumes that the current input symbol a_i , $1 \leq i \leq n$, is the first symbol of the input portion of x^R , and compares it with a_{i-1} . If they match, then the machine marks a_{i-1} . On the next input M must compare a_{i+1} with a_{i-2} if a_{i-1} had a mark. If a_{i-1} had a mark and a_{i+1} and a_{i-2} match, the mark is moved to a_{i-2} . On input a_{i+2} , M must compare a_{i+2} with a_{i-3} , if a_{i-2} had a mark, etc. Examples of accepting computations are shown in Fig. 4.

Input: $a b c c b a a b c c b a$

Worktape: $\$ a$
 $\$ a b$
 $\$ a b c$
 $\$ a b \bar{c} c$
 $\$ a \bar{b} c c b$
 $\$ \bar{a} b c c b a$
 $\$ a b c c b \bar{a} a$
 $\$ a b c c \bar{b} a a b$
 $\$ a b c \bar{c} b a a b c$
 $\$ a b \bar{c} c b a a b \bar{c} c$
 $\$ a \bar{b} c c b a a \bar{b} c c b$
 $\$ \bar{a} b c c b a \bar{a} b c c b a$

(a)

Input: $a a a a a a a a$

Worktape: $\$ a$
 $\$ \bar{a} a$
 $\$ a \bar{a} a$
 $\$ \bar{a} a \bar{a} a$
 $\$ a \bar{a} a \bar{a} a$
 $\$ \bar{a} a \bar{a} a \bar{a} a$
 $\$ a \bar{a} a \bar{a} a \bar{a} a$
 $\$ \bar{a} a \bar{a} a \bar{a} a \bar{a} a$

(b)

Fig. 4. Accepting profiles of a DSTM for (a) $a b c c b a a b c c b a$, and (b) $a a a a a a a a$.

We now define M formally:

$$M = \langle Q, \Sigma, \Gamma, \delta, q_0, F \rangle, \text{ where } Q = \{q_0\} \cup \{[q_i, a] \mid i = 1, 2, 3, a \text{ in } \Sigma\},$$

$$\Gamma = \{\$, \lambda\} \cup \Sigma \cup \{\bar{a} \mid a \text{ in } \Sigma\}, F = \{[q_2, a] \mid a \text{ in } \Sigma\},$$

and δ is defined as follows, where a, b, c are in Σ with $a \neq b$:

- (1) $\delta(q_0, \lambda, a) = ([q_1, a], a, -1)$,
- (2) $\delta([q_1, a], a, \varepsilon) = ([q_2, a], \bar{a}, -1)$,
- (3) $\delta([q_1, a], b, \varepsilon) = ([q_3, a], b, -1)$,
- (4) $\delta([q_2, a], c, \varepsilon) = \delta([q_3, a], c, \varepsilon) = ([q_3, a], c, -1)$,
- (5) $\delta([q_2, a], \bar{c}, \varepsilon) = \delta([q_3, a], \bar{c}, \varepsilon) = ([q_1, a], c, -1)$,
- (6) $\delta([q_i, a], \$, \varepsilon) = (q_0, \$, +1)$ for $i = 1, 2, 3$,

$$(7) \quad \delta(q_0, a, \varepsilon) = (q_0, a, +1),$$

$$(8) \quad \delta(q_0, \bar{a}, \varepsilon) = (q_0, \bar{a}, +1).$$

Remark. We shall see in Section 3 that $\mathcal{L}(\text{DSTM}) = \mathcal{L}(\text{DHTA})$ and $\mathcal{L}(\text{NSTM}) = \mathcal{L}(\text{NHTA})$. Hence $\{xx^R \mid x \text{ in } \Sigma^+\}$ and $\{0^n 1^{2^n} \mid n \geq 1\}$ are in $\mathcal{L}(\text{DHTA})$. The language $\{xx^R \mid x \text{ in } \Sigma^+\}$ was already shown to be in $\mathcal{L}(\text{DHTA})$ [5].

Proposition 2.3. *Every ε -free regular set is a DSTM-language.*

Proof. If L is regular, then L^R (reverse of L) is regular. Let A^R be a deterministic finite automaton accepting L^R . Then a DSTM M can be constructed which operates as follows: After reading a symbol in state q_0 , M writes the symbol and scans the worktape simulating A^R . \square

Proposition 2.4. *DSTM-languages are closed under the Boolean operations.*

Proof. Complementation is obvious if we first make the δ function total. Union and intersection easily follow from the fact that a DSTM can simulate two DSTM's by having 2 tracks on its worktape. \square

Corollary 2.5. *NSTM-languages are closed under union and intersection.*

3. Characterizations and complexity of homogeneous trellis automata

Theorem 3.1. $\mathcal{L}(\text{DSTM}) = \mathcal{L}(\text{DHTA})$ and $\mathcal{L}(\text{NSTM}) = \mathcal{L}(\text{NHTA})$.

Proof. We only prove the deterministic case, the nondeterministic case being similar. Let $M = \langle \Sigma, \Gamma, \Delta, f, g \rangle$ be a DHTA. Construct a DSTM $M' = \langle Q, \Sigma, \Gamma \cup \{S, \lambda\}, \delta, q_0, F \rangle$, where $Q = \{q_0\} \cup \Gamma$, $F = \Delta$ and δ is defined below, where a is in Σ , Z_1 and Z_2 in Γ :

$$(1) \quad \delta(q_0, \lambda, a) = (f(a), f(a), -1),$$

$$(2) \quad \delta(Z_2, Z_1, \varepsilon) = (g(Z_1, Z_2), g(Z_1, Z_2), -1),$$

$$(3) \quad \delta(Z_1, S, \varepsilon) = (q_0, S, +1),$$

$$(4) \quad \delta(q_0, Z_1, \varepsilon) = (q_0, Z_1, +1).$$

As an example, consider a computation of a DHTA M as shown in Fig. 5(a), where the nodes are $1, 2, \dots, 10$ and the output from node i is Z_i . Then one can easily check that the worktape profile of M' will be as shown in Fig. 5(b) after each right-to-left sweep, i.e., M' simulates M node by node in the order $1, 2, 3, \dots, 10$. One can easily check that $L(M') = L(M)$.

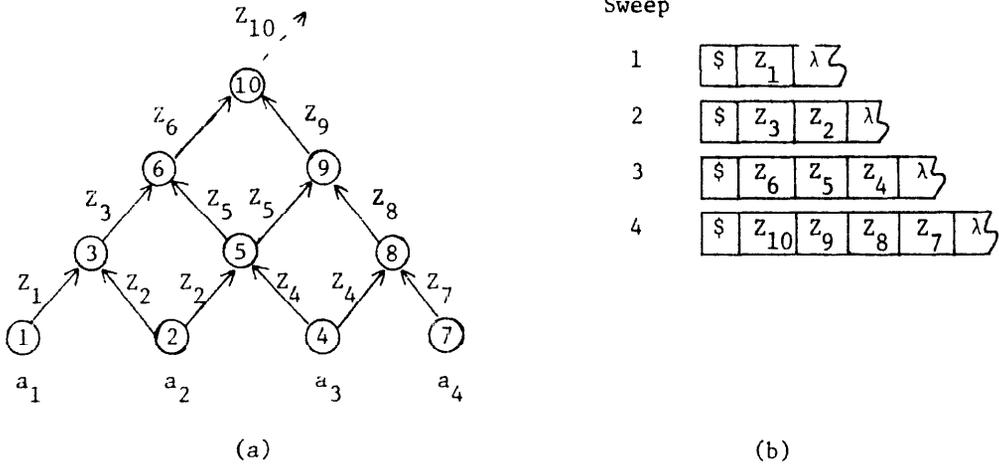


Fig. 5. (a) A computation of a DHTA M . (b) The simulation profile of M' .

Now suppose that $M = \langle Q, \Sigma, I', \delta, q_0, F \rangle$ is a DSTM. Construct

$$M' = \langle \Sigma, I'', \Delta, f, g \rangle, \text{ where } I'' = \{[q, Z] \mid q \text{ in } Q, Z \text{ in } I'\} \cup \{D\},$$

$$\Delta = \{[q, Z] \mid q \text{ in } F, Z \text{ in } I'\}.$$

f and g are defined below, where a is in Σ , q_1 and q_2 in Q , Z_1 and Z_2 in I' :

- (1) $f(a) = \begin{cases} [q_1, Z_1] & \text{if } \delta(q_0, \lambda, a) = (q_1, Z_1, -1), \\ D & \text{if } \delta(q_0, \lambda, a) = \emptyset, \end{cases}$
- (2) $g(D, D) = g([q_1, Z_1], D) = g(D, [q_1, Z_1]) = D,$
- (3) $g([q_1, Z_1], [q_2, Z_2]) = \begin{cases} [q'_1, Z'_1] & \text{if } \delta(q_2, Z_1, \epsilon) = (q'_1, Z'_1, -1), \\ D & \text{if } \delta(q_2, Z_1, \epsilon) = \emptyset. \end{cases}$

It is straightforward to check that $L(M') = L(M)$. \square

Definition. An *NOTM* is a generalization of an *NSTM* in that the machine can also change its state and worktape contents when the worktape head is making a left-to-right sweep. *DOTM* is the deterministic version of *NOTM*.

The next result shows that *NOTM*'s are no more powerful than *NSTM*'s.

Theorem 3.2. $\mathcal{L}(\text{NSTM}) = \mathcal{L}(\text{NOTM})$.

Proof. $\mathcal{L}(\text{NSTM}) \subseteq \mathcal{L}(\text{NOTM})$ is trivial. Given an *NOTM* M_1 we can construct an *NSTM* M_2 which simulates the computation of a full sweep (i.e., left-to-right and right-to-left sweeps) of M_1 by a single right-to-left sweep. This is because M_2 must remain in the same state q_0 and cannot write on the worktape on a left-to-right sweep. Hence, M_2 , when simulating the i th right-to-left sweep of M_1 , must also simultaneously guess and simulate the actions of M_1 on its previous left-to-right sweep of the worktape. We omit the details. \square

Corollary 3.3. $\mathcal{L}(\text{NHTA}) = \mathcal{L}(\text{NSTM}) = \mathcal{L}(\text{NOTM})$.

The next proposition shows that Theorem 3.2 does not hold for the deterministic case.

Proposition 3.4. $\mathcal{L}(\text{DHTA}) = \mathcal{L}(\text{DSTM}) \subsetneq \mathcal{L}(\text{DOTM})$.

Proof. The language $L = \{a^{2^n} \mid n \geq 1\}$ is not in $\mathcal{L}(\text{DHTA})$ [5] but can easily be accepted by a DOTM. \square

Corollary 3.5. $\mathcal{L}(\text{DHTA}) \subsetneq \mathcal{L}(\text{NHTA})$.

Corollary 3.6. Every ε -free CFL (context free language) is in $\mathcal{L}(\text{NHTA})$.

Proof. Let L be an ε -free CFL. Then L can be accepted by a nondeterministic PDA (pushdown automaton) M_1 with no ε -moves and adds at most 1 symbol on its pushdown store per move [10]. M_1 can trivially be simulated by an NOTM M_2 which marks the position of the top of the pushdown store (since M_2 cannot erase). \square

Corollary 3.7. Every language accepted by a deterministic PDA without ε -moves can be accepted by a DOTM.

Questions. (1) Can we remove the restriction 'without ε -moves' in Corollary 3.7? (2) Can every ε -free CFL be accepted by a DOTM? This question is hard since a negative (positive) answer would provide a lower (upper) bound of $O(n^2)$ for CFL recognition by DOTM's.

Proposition 3.8. Every NHTA-language has simultaneous time and space complexity $O(n^2)$ and $O(n)$ on a nondeterministic TM.

Proof. Obvious from the definition of the model. \square

It is unlikely that every NHTA-language can be accepted by a deterministic TM in polynomial time because of the following proposition which is easily verified.

Proposition 3.9. $\mathcal{L}(\text{NHTA})$ contains an NP-complete language.

Proof. Let $L = \{d^k \# x_1 \# \cdots \# x_n \mid n \geq 1, \text{ each } x_i \text{ in binary (with least significant digits first and most significant digit 1)}, k \geq |x_1 \# \cdots \# x_n| (= \text{length of } x_1 \# \cdots \# x_n) \text{ and } x_{i_1} + \cdots + x_{i_r} = x_{i_{r+1}} + \cdots + x_{i_n} \text{ for some choice of indices } i_1, \dots, i_n, i_j \neq i_l\}$. Then we can easily construct an NOTM M which accepts L . Obviously, L is in NP. L is NP-hard since if it is in P , then $L' = \{x_1 \# \cdots \# x_n \mid x_{i_1} + \cdots + x_{i_r} = x_{i_{r+1}} + \cdots + x_{i_n}, i_j \neq i_l\}$ is also in P . But L' is the partition problem which is NP complete [13]. \square

Proposition 3.10. $\mathcal{L}(\text{NHTA}) \subseteq \text{DSPACE}(n \log n)$.

Proof. L in $\mathcal{L}(\text{NHTA})$ has nondeterministic time and space complexity $O(n^2)$ and $O(n)$, respectively. Now, by Savitch's result [18], a language accepted by a nondeterministic TM in time $T(n)$ and space $S(n)$ can be accepted by a deterministic TM in space $O(S(n) \log T(n))$. The result follows. \square

As in Proposition 3.8, we have the following.

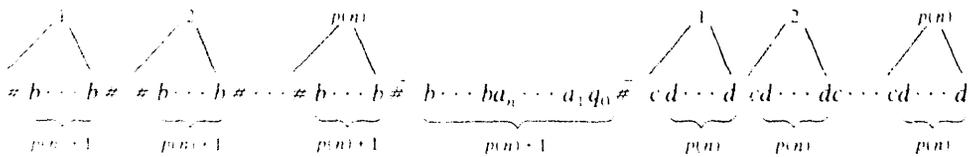
Proposition 3.11. *Every DOTM-language can be accepted by a deterministic TM in time $O(n^2)$ and space $O(n)$.*

Corollary 3.12. *Every DHTA-language can be accepted by a deterministic TM in time $O(n^2)$ and space $O(n)$.*

The next proposition shows that it is very unlikely that the space bounds in Propositions 3.8 and 3.11 and Corollary 3.12 can be reduced substantially. This partially answers a problem posed in [6].

Proposition 3.13. $\mathcal{L}(\text{DHTA})$ contains a language L which is complete for P (= languages accepted by deterministic polynomial time bounded TM's) with respect to log-space reductions. Thus, L is in P and it has the property that every language in P is log-space reducible to it. It follows that if L is in $\text{DSPACE}(\log^k n)$ for some k , then P is contained in $\text{DSPACE}(\log^k n)$ which is very unlikely (see [3, 4]).

Proof. Let L_1 be complete for P with respect to log-space reductions (see [11] for examples of such languages). We describe a language L in $\mathcal{L}(\text{DHTA})$ (hence in P) such that L_1 is log-space reducible to L . Let M_1 be a single-tape deterministic TM with a semi-infinite tape accepting L_1 . Let $p(n)$ be the polynomial time bound of M_1 . We may assume without loss of generality (by cycling in an accepting state if necessary) that each accepted input can be accepted in exactly $p(n)$ steps. We construct a DSTM M^f accepting L such that, for each $x = a_1 \cdots a_n$, $n \geq 1$, M_1 accepts x if and only if M accepts the string $A(x) = (\# b^{p(n)+1})^{p(n)} \bar{\#} b^{p(n)+1} a_n \cdots a_1 q_0 \bar{\#} (cd^{p(n)})^{p(n)}$, where $\#, b, \bar{\#}, c, d$ are new symbols and q_0 is the start state of M_1 . Thus, $A(x)$ looks like



Let $\beta_0 \# \beta_1 \# \cdots \# \beta_{p(n)}$ be an accepting computation of M_1 on input $a_1 \cdots a_n$, where each β_i is an instantaneous description (ID) of M_1 of length $p(n)+1$ (see

[11]). M on input $A(x)$ will simulate the computation of M_1 on x by constructing the accepting sequence of ID's above on its worktape, i.e., at the end of M 's computation, its worktape will have the form

$$\$ \bar{\#} \beta_{p(n)}^R \bar{\#} \beta_{p(n)-1}^R \# \cdots \# \beta_2^R \# \beta_1^R \# \beta_0^R \# (cd^{p(n)})^{p(n)},$$

where R denotes reversal.

We now briefly describe the operation of M on input $A(x)$.

Step 1. M copies the input symbols on its worktape until it sees a "c". Thus, at the end of Step 1, the worktape will have the form

$$\$ \# b \cdots b \# \cdots \# b \cdots b \bar{\#} b \cdots ba_n \cdots a_1 q_0 \bar{\#}.$$

Step 2. When M sees input symbol c , it modifies the worktape so that it looks like

$$\$ \# b \cdots b \# \cdots \# b \cdots \bar{b} \bar{b} \bar{\#} b \cdots ba_n \cdots \bar{a}_1 \bar{q}_0 \bar{\#} c.$$

Note that the block $b \cdots ba_n \cdots \bar{a}_1 \bar{q}_0$ is just β_0^R with the last 2 symbols of it marked, i.e., barred. We need to mark 3 symbols so that we can uniquely determine the symbols in the corresponding positions of the next ID, β_1^R .

Step 3. In this step, M reads the input segment consisting of $p(n)$ d 's to construct β_1^R to the left of β_0^R , i.e., at the end of this step, the worktape will look like

$$\$ \# b \cdots b \# \cdots \# b \cdots b \bar{\#} \underbrace{b \cdots ba_n \cdots a_2 q_1 a_1 \bar{\#}}_{\beta_1^R} \underbrace{b \cdots ba_n \cdots a_2 a_1 q_0 \#}_{\beta_0^R} \underbrace{cd \cdots d}_{p(n)}$$

if M_1 replaced a_1 by a_1' and moved to the right in state q_1 . The two sets of bars are moved 1 position to the left for every d , provided the barred symbols in β_0^R are not of the form $\bar{Z}\bar{q}\bar{W}$. If the barred symbols in β_0^R are of the form $\bar{Z}\bar{q}\bar{W}$, the bars are moved 3 positions to the left. (Thus, the 2 sets of bars are propagating to the left.) The three bars will become a single bar over the symbol $\#$ when it is reached.

Step 4. Repeat Steps 2 and 3 for each input segment $cd^{p(n)}$ until $\beta_{p(n)}^R$ has been constructed. At the end of the process, the worktape will look like

$$\$ \bar{\#} \beta_{p(n)}^R \bar{\#} \beta_{p(n)-1}^R \# \beta_{p(n)-2}^R \# \cdots \# \beta_2^R \# \beta_1^R \# \beta_0^R \# (cd^{p(n)})^{p(n)}$$

and M accepts if and only if $\beta_{p(n)}^R$ has an accepting state.

We conclude the proof by observing that we can construct a deterministic off-line TM transducer [11] which on input $x = a_1 \cdots a_n$ can output $A(x)$ using only $O(\log(p(n))) = O(\log n)$ space. \square

Remark. In [3, 4] strong evidences are given to show that there are languages in P that may need deterministic (respectively, nondeterministic) n^ϵ -space for some $0 < \epsilon < 1$. It follows from this and Proposition 3.13 that there are DHTA-languages (and hence DOTM and NOTM languages) that may require n^ϵ -space on a deterministic (respectively, nondeterministic) TM for some $0 < \epsilon' < 1$.

Proposition 3.13 can be made stronger: DHTA's can simulate arbitrary deterministic TM's in some precise sense.

Definition. A DHTA M with input alphabet $\Sigma \cup \{\#\}$ is *padding* if

- (1) $L(M) \subseteq \{\#^k x \#^k \mid k \geq 0, x \in \Sigma^+\}$ and
- (2) if $\#^i x \#^i$ is accepted by M , then $\#^j x \#^j$ is also accepted for all $j \geq i$.

The construction in Proposition 3.13 together with the programming strategy described in Example 4.3 (Section 4) can be used to prove the following corollary.

Corollary 3.14. *Let M_1 be a single-tape deterministic TM accepting a language $L_1 \subseteq \Sigma^+$. Let $\#$ be a new symbol. Then we can construct a padded DHTA M_2 accepting a language L_2 with the following properties:*

- (1) *If x is in L_1 and M_1 accepts x in space $s \geq n$ and time t , then $\#^{2s} x \#^{2s}$ is in L_2 .*
- (2) *If $\#^k x \#^k$ is in L_2 for some k , then x is in L_1 .*

Thus if M_1 has simultaneous space and time bounds $s(n)$ and $t(n)$, respectively, then to determine if x is in L_1 , we need only input the string $\#^k x \#^k$ to M_2 , where $k = 2s(n)t(n)$.

The above corollary shows that, if we pad M_1 's input with enough $\#$'s (which may depend on x), we can simulate the computation of M_1 on x . The padding on the left and right sides of x are necessary as the following proposition shows.

Proposition 3.15. *Let M_1 be a single-tape deterministic TM accepting a language L_1 which is not accepted by a deterministic Linear Bounded Automaton. Then there is no DHTA M_2 accepting a language $L_2 \subseteq \#^* \Sigma^+ (or $L_2 \subseteq \Sigma^+ \#^*$) such that $L_1 = h(L_2)$, where h is a homomorphism defined by $h(a) = a$ for each a in Σ and $h(\#) = \varepsilon$.$*

Proof. If M_2 exists, then we can construct (from M_2) a deterministic LBA accepting $h(L_2)$. The details are in [6]. \square

If M is a padded DHTA with input alphabet $\Sigma \cup \{\#\}$ and x is in Σ^+ , let $\langle M, x \rangle$ be the binary encoding of M and x . Corollary 3.14 says that we can construct a padded DHTA U with $\Sigma_U = \{0, 1\}$ such that U accepts $\#^k w \#^k$ for some k and binary string w if and only if $w = \langle M, x \rangle$ for some padded DHTA M and input x , and M accepts $\#^m x \#^m$ for some m . Thus, in some sense, U is universal over all padded DHTA's.

One can easily check that Corollary 3.14 and the 'universality' result are still valid for left-padded or right-padded NHTA's. In fact, these results also hold for left-padded DOTM's.

We conclude this section with two open problems:

(1) Is $\mathcal{L}(\text{NOTM}) - \mathcal{L}(\text{DOTM}) \neq \emptyset$? We conjecture that the answer is "yes" since, otherwise, $P = NP$. To see this, let L_1 be accepted by a $p(n)$ -time bounded nondeterministic TM M_1 , where p is some polynomial. Let d be a new symbol, not in the alphabet of L_1 . Then it is easy to construct from M_1 an NOTM M_2 which accepts a string of the form $x d^{p(n)}$ if and only if x is in L_1 , $|x| = n$. If $\mathcal{L}(\text{NOTM}) = \mathcal{L}(\text{DOTM})$, there exists a DOTM M_3 equivalent to M_2 . Clearly, we can construct from M_3 a deterministic TM M_4 accepting L_1 .

(2) Is $\mathcal{L}(\text{NOTM})$ closed under complementation? The answer is probably “no” for, otherwise, we can show (using the padding technique above) that NP would be closed under complementation.

4. Applications

The characterization given in Theorem 3.1 can be used to give simpler proofs of many results concerning DHTA-languages in [5] as well as derive other results. (NHTA-languages were not studied in [5].) We illustrate this with some examples.

Example 4.1. Let D_r be the Dyck language of r letters [5], that is, letting $\Sigma_r = \{a_1, \bar{a}_1, \dots, a_r, \bar{a}_r\}$, $D_r = \{x \mid x \text{ in } \Sigma_r^+ \text{ such that } x \text{ reduces to } \varepsilon \text{ if we apply the cancellation rule } a_i \bar{a}_i = \varepsilon\}$. In [5] it was shown that D_2 is a DHTA-language. Using Theorem 3.1, we can give an easier construction. A DSTM M can be constructed which, when given an input x in Σ_r^+ , writes each a_i it sees and cancels it (by using $\#$) when the matching \bar{a}_i is seen. Formally, $M = \langle Q, \Sigma_r, \Gamma, \delta, q_0, F \rangle$, where $Q = \{q_0, q_1, q_2, q_3\} \cup \{a_1, \dots, a_r\}$, $F = \{q_2\}$, $\Gamma = \{\$, \lambda, \#\} \cup \Sigma_r$ and δ is defined below, where $1 \leq i \leq r$, Z in $\Gamma - \{\$\}$:

- (1) $\delta(q_0, \lambda, a_i) = (q_1, a_i, -1)$,
- (2) $\delta(q_1, Z, \varepsilon) = (q_1, Z, -1)$,
- (3) $\delta(q_1, \$, \varepsilon) = (q_0, \$, +1)$.
- (4) $\delta(q_0, Z, \varepsilon) = (q_0, Z, +1)$.
- (5) $\delta(q_0, \lambda, \bar{a}_i) = \delta(a_i, \#, \varepsilon) = (a_i, \#, -1)$,
- (6) $\delta(a_i, a_i, \varepsilon) = \delta(q_2, \#, \varepsilon) = (q_2, \#, -1)$.
- (7) $\delta(q_2, \$, \varepsilon) = (q_0, \$, +1)$.
- (8) $\delta(q_2, a_i, \varepsilon) = (q_1, a_i, -1)$.

It is easy to check that $L(M) = D_r$.

The next example was also shown in [5] using a nontrivial programming strategy.

Example 4.2. Let $L = \{x \# x \mid x \text{ in } \{0, 1\}^*\}$. We shall construct a DSTM accepting L . By Propositions 2.3 and 2.4 we need only construct a DSTM M which accepts L provided the inputs are of the form $x \# y$, where x, y in Σ^* . M , on input $x \# y$, reads and writes on the worktape the string x . After reading the symbol $\#$, the symbols of y are propagated to the left on the worktape and matched with x . The formal description follows. The example in Fig. 6 shows how the symbols of y are propagated, where each row corresponds to the worktape contents after each right-to-left sweep.

Input: $abc \neq abc$

Worktape: $\$ a$
 $\$ a b$
time $\left\{ \begin{array}{l} \$ a b c \\ \$ a b c \# \\ \$ a b \binom{c}{a} \# a \\ \$ a \binom{b}{a} \binom{c}{b} \# a b \\ \$ \binom{a}{a} \binom{b}{b} \binom{c}{c} \# a b c \end{array} \right.$

Fig. 6. Worktape profile.

Formally, $M = \langle Q, \{0, 1, \#\}, \Gamma, \delta, q_0, F \rangle$, where $Q = \{q_0, q_4, q_5\} \cup \{[q_i, a] \mid i = 1, 2, 3, 4, a \text{ in } \{0, 1\}\}$, $\Gamma = \{0, 1, \#\} \cup \{\binom{a}{b} \mid a, b \text{ in } \{0, 1\}\} \cup \{\lambda, \$\}$, $F = \{q_5\}$, and δ is defined below, where a, b, c are in $\{0, 1\}$ and d is in $\Gamma - \{\lambda, \$\}$:

$$\begin{aligned} \delta(q_0, \lambda, a) &= ([q_1, a], a, -1), \\ \delta([q_1, a], b, \varepsilon) &= ([q_1, a], b, -1), \\ \delta([q_1, a], \$, \varepsilon) &= (q_0, \$, +1), \\ \delta(q_0, d, \varepsilon) &= (q_0, d, +1), \\ \delta(q_0, \lambda, \#) &= (q_5, \#, -1), \\ \delta([q_1, a], \#, \varepsilon) &= ([q_2, a], \#, -1), \\ \delta([q_2, a], b, \varepsilon) &= (q_4, \binom{b}{a}, -1) \quad \text{if } a \neq b, \\ \delta(q_4, c, \varepsilon) &= (q_4, c, -1), \\ \delta(q_4, \$, \varepsilon) &= (q_4, \$, +1), \\ \delta([q_2, a], \binom{b}{c}, \varepsilon) &= ([q_4, c], \binom{b}{a}, -1) \quad \text{if } a \neq b, \\ \delta([q_4, a], \binom{b}{c}, \varepsilon) &= ([q_4, c], \binom{b}{a}, -1), \\ \delta([q_2, a], a, \varepsilon) &= (q_5, \binom{a}{a}, -1), \\ \delta(q_5, c, \varepsilon) &= (q_4, c, -1), \\ \delta([q_2, a], \binom{a}{c}, \varepsilon) &= ([q_3, c], \binom{a}{a}, -1), \\ \delta([q_3, a], \binom{a}{c}, \varepsilon) &= ([q_3, c], \binom{a}{a}, -1), \\ \delta([q_3, a], a, \varepsilon) &= (q_5, \binom{a}{a}, -1), \\ \delta([q_3, a], \binom{b}{c}, \varepsilon) &= ([q_4, c], \binom{b}{a}, -1) \quad \text{if } a \neq b, \\ \delta([q_4, a], b, \varepsilon) &= (q_4, \binom{b}{a}, -1) \quad \text{if } a \neq b. \end{aligned}$$

One can check that $L(M) = L$ provided that the inputs are of the form $x \neq y$, where x, y in Σ^* .

Remark. The technique used in Example 4.2 can be extended to show that the following languages are in $\mathcal{L}(\text{DHTA})$:

- (i) For each $k \geq 1$, the language $\{(x\#)^k \mid x \text{ in } \{0, 1\}^+\}$.
- (ii) $\{x_1\#x_2\#\dots\#x_k \mid k \geq 1, x_i \text{ has most significant digit } 1 \text{ and is the binary representation of integer } i\}$
- (iii) $L_M = \{w \mid w = \alpha_0\#\alpha_1\#\dots\#\alpha_k \text{ is an accepting computation of the deterministic single-tape TM } M \text{ on some input}\}$.

The constructions in the remaining examples will be described informally. The reader should have no difficulty defining the transition δ formally.

Example 4.3. Let $L = \{a^{kn}b^nc^{(k-1)n} \mid k, n \geq 1\}$. A DSTM M accepts L as follows (where by Propositions 2.3 and 2.4, we assume that the input has the form $a^ib^jc^k$, for some $i, j \geq 1, k \geq 0$):

After writing the current input symbol on the worktape, M does the following during the right-to-left sweep:

case the input symbol is

- a: Do nothing;
- b: Replace the first “ a ” met with “ $\#$ ”;
if the leftmost “ a ” is replaced with “ $\#$ ” **then**
accept²
endif;
- c: Replace the symbol “ $\#$ ” or “ b ” whichever is met first with “ c ”, and let the old symbol (“ $\#$ ” or “ b ”) be Z ;
replace the first “ a ” met with Z ;
if the leftmost “ a ” is replaced **and** the symbol Z is from the leftmost $\#$ or b **then**
accept
endif;

endcase;

Fig. 7 shows an example of how M accepts $a^6b^2c^4$.

Example 4.4. Let Σ be an alphabet and $k \geq 2$. For each $1 \leq i \leq k$ let $\Sigma^{(i)} = \{a^{(i)} \mid a \text{ in } \Sigma\}$ be a distinct alphabet. Let $\Delta_k = \Sigma^{(1)} \cup \dots \cup \Sigma^{(k)}$. For $1 \leq i \leq k$, define a homomorphism $h_i: \Delta_k^* \rightarrow \Sigma^*$ by $h_i(a^{(j)}) = a$ if $i = j$, and ϵ otherwise. Let $\#$ be a new symbol. For $k \geq 2$ and $r \geq 0$, define the language

$$L_{k,r} = \{\#^m x \#^m \mid x \text{ in } \Delta_k^+, |x| = kn, \text{ and } h_i(x) = h_j(x) \text{ for all } i \neq j\}.$$

(Note that when $r = 0$, the strings in $L_{k,0}$ have no $\#$'s.)

The language $L_{2,0}$ is called the *twin shuffle* over Σ [6]. It is conjectured in [6] that $L_{2,0}$ is not in $\mathcal{L}(\text{DHTA})$. However, the language $L_{2,2}$ was shown to be in

² More precisely, this means that M , after replacing “ a ” with “ $\#$ ”, moves left in an accepting state. If, after the move, the head is not on “ S ”, M completes the right-to-left scan in a nonaccepting state.

Input: $a a a a a a b b c c c c$

Worktape: $\$ a$
 $\$ a a$
 $\$ a a a$
 $\$ a a a a$
 $\$ a a a a a$
 $\$ a a a a a a$
 $\$ a a a a a \# b$
 $\$ a a a a \# \# b b$
 $\$ a a a b \# \# b c c$
 $\$ a a b b \# \# c c c c$
 $\$ a \# b b \# c c c c c c$
 $\$ \# \# b b c c c c c c c c$

Fig. 7. Worktape profile of M for input $a^6b^2c^4$.

$\mathcal{L}(\text{DHTA})$ by a nontrivial DHTA-programming algorithm. Here we show by a simple construction that, in fact, $L_{k,r}$ and the languages $L'_{k,r} = \{\#^m x \mid \text{same condition as in } L_{k,r}\}$ and $L''_{k,r} = \{x \#^m \mid \text{same condition as in } L_{k,r}\}$ (which have only one sided paddings) are also in $\mathcal{L}(\text{DHTA}) = \mathcal{L}(\text{DSTM})$ for $k \geq 2$ and $r \geq 1$.

We briefly describe a DSTM M' accepting $L'_{k,r}$. The DSTM's for $L_{k,r}$ and $L''_{k,r}$ are constructed similarly. By Propositions 2.3 and 2.4 we may assume that inputs to M' come from $\#^+ \Delta_k^+$. The worktape of M' has k tracks indexed 1 through k . Let y be an input to M' . The i th track in the padded segment is used to construct the string $h_i(y)$ from the alphabet $\Sigma^{(i)}$ with the symbols of $h_i(y)$ spaced r cells apart. Fig. 8 shows an accepting profile for the case of $k = 2, r = 1, \Sigma = \{a, b, c\}, \Sigma^{(1)} = \{a_1, b_1, c_1\}$ and $\Sigma^{(2)} = \{a_2, b_2, c_2\}$. The formal construction of the algorithm is left to the reader.

Let $L \subseteq \Sigma^+$ be a DHTA-language. Let $\#$ be a new symbol. Define $L_\# = \{y \mid y \text{ in } (\Sigma \cup \{\#\})^+ \text{ such that } y \text{ reduces to some } x \text{ in } L \text{ when the } \# \text{'s are deleted}\}$. We give a simpler proof of the following result which was shown in [6].

Proposition 4.5. *If L is in $\mathcal{L}(\text{DHTA})$, then so is $L_\#$.*

Proof. Let M_1 be a DSTM accepting L . A DSTM M_2 accepting $L_\#$ can be constructed as follows. M_2 simulates M_1 using a two-track worktape. The top track contains the 'current' contents of the worktape of M_1 while the bottom track contains the 'previous' contents. When an input symbol different from $\#$ is read, the contents of the top track is shifted to the bottom track and the new contents of the top track are generated as M_1 would. When the input is $\#$, no changes are made on the worktape. However, the contents of the bottom track are used to simulate the moves M_1 made on the last input symbol read that is not $\#$. The formal construction of M_2 is straightforward. \square

Input:	#	#	#	#	#	a_2	a_1	a_2	a_1	b_1	b_2	c_2	c_1	c_2	c_1
Worktape:	\$	#													
	\$	#	#												
	\$	#	#	#											
	\$	#	#	#	#										
	\$	#	#	#	#	$(\#)$	a_2								
	\$	#	#	#	#	(a_2)	a_2	a_1							
	\$	#	#	#	#	$(\#)$	(a_2)	a_2	a_1	a_2					
	\$	#	#	#	#	(a_2)	(a_2)	a_2	a_1	a_2	a_1	a_2	a_1		
	\$	#	#	(a_1)	(a_2)	(b_2)	a_2	a_1	a_2	a_1	a_2	a_1	b_1		
	\$	#	#	(a_2)	(a_2)	(b_2)	a_2	a_1	a_2	a_1	a_2	a_1	b_1	b_2	
	\$	#	$(\#)$	(a_2)	(a_2)	(b_2)	a_2	a_1	a_2	a_1	a_2	a_1	b_1	b_2	c_2
	\$	#	(a_2)	(a_2)	(b_2)	(c_2)	a_2	a_1	a_2	a_1	a_2	a_1	b_1	b_2	c_2
	\$	$(\#)$	(a_2)	(a_2)	(b_2)	(c_2)	a_2	a_1	a_2	a_1	a_2	a_1	b_1	b_2	c_2
	\$	(a_2)	(a_2)	(b_2)	(c_2)	(c_2)	a_2	a_1	a_2	a_1	a_2	a_1	b_1	b_2	c_2
	\$	(a_2)	(a_2)	(b_2)	(c_2)	(c_2)	a_2	a_1	a_2	a_1	a_2	a_1	b_1	b_2	c_2
	\$	(a_2)	(a_2)	(b_2)	(c_2)	(c_2)	a_2	a_1	a_2	a_1	a_2	a_1	b_1	b_2	c_2

Fig. 8. The accepting profile of M' .

Remark. By Proposition 4.5, we can construct a DHTA accepting $L_\#$. Such a DHTA is called a *super-stable DHTA* in [6].

The next proposition which is easily shown using DSTM's was already proved in [5] using a special technique called 'path automaton technique'.

Proposition 4.6. Let $L_1 \subseteq \Sigma_1^+$ and $L_2 \subseteq \Sigma_2^+$ be in $\mathcal{L}(\text{DHTA})$ and $\#$ be a new symbol not in $\Sigma_1 \cup \Sigma_2$.

- (1) If $\Sigma_1 \cap \Sigma_2 = \emptyset$, then $L_1 L_2$ is in $\mathcal{L}(\text{DHTA})$.
- (2) $L_1 \# L_2$ is in $\mathcal{L}(\text{DHTA})$.
- (3) $(L_1 \#)^+$ is in $\mathcal{L}(\text{DHTA})$.

Proof. The constructions for (1)–(3) are similar. We only describe (1). Given the DSTM's M_1 and M_2 accepting languages L_1 and L_2 , we construct a DSTM M accepting $L_1 L_2$ as follows. Let $x_1 x_2$ be the input to M , where x_1 is in Σ_1^+ and x_2 is in Σ_2^+ . During the simulation of M_1 on x_1 , M uses a two track tape which is operated just like that of the machine M_2 in Proposition 4.5. Thus the current contents of M_1 's worktape are stored on the top track while the previous contents are stored on the bottom track. When the first symbol of x_2 is read, M starts simulating M_2 . However, on the right-to-left sweep of the worktape (after reading the first symbol of x_2), M uses the contents of the bottom track to check that x_1 is in L_1 . If x_1 is not in L_1 , M halts in a nonaccepting state. Otherwise, M goes on with the simulation of M_2 . \square

We do not know whether $\mathcal{L}(\text{DSTM})$ is closed under concatenation. However, we can show that it is closed under concatenation with regular sets. For the proof, it is convenient to give an equivalent formulation of a DSTM.

Definition. A *modified DSTM* (MDSTM, for short) is a DSTM which can write on the leftmost cell of the worktape. Formally, an MDSTM is a 6-tuple $M = \langle Q, \Sigma, \Gamma \cup (\{\$ \} \times (\Gamma - \{\$ \})), \delta, q_0, \Gamma_s \rangle$, where we allow transitions of the form $\delta(q, [\$, Z], \epsilon) = (q_0, [\$, Z'], +1)$, $Z, Z' \in (\Gamma - \{\$ \})$. Initially, the leftmost cell of the worktape is $[\$, \lambda]$. (Note that in a DSTM, $\delta(q, \$, \epsilon) = (q_0, \$, +1)$.) All other transitions are defined in the same way as in a DSTM. $\Gamma_s \subseteq \{ [\$, Z] \mid Z \in \Gamma - \{\$ \} \}$ is the accepting tape alphabet. An input $a_1 \cdots a_n$ is accepted if, after reading the string $a_1 \cdots a_n$, M writes on the leftmost cell a symbol in Γ_s .

Proposition 4.7. $\mathcal{L}(\text{DSTM}) = \mathcal{L}(\text{MDSTM})$.

Proof. $\mathcal{L}(\text{DSTM}) \subseteq \mathcal{L}(\text{MDSTM})$ is obvious. Now let M be an MDSTM. We construct a DSTM M' which simulates M . Each cell of M' , except the leftmost one, is divided into two subcells (see Fig. 9). On each cell, during the right-to-left sweep, M' assumes that the left subcell is the leftmost cell of M and the right subcell is the cell currently being scanned by M . On the left subcell, M' writes the tape symbol M would have written if the right subcell were the cell adjacent to the left end of M' 's worktape. If the tape symbol which is written on the left subcell is in Γ_s (the accepting tape alphabet of M), then M' moves to the left in an accepting state.

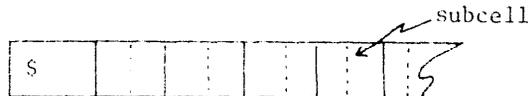


Fig. 9. The worktape of M' .

Formally, let $M = \langle Q, \Sigma, \Gamma \cup (\{\$ \} \times (\Gamma - \{\$ \})), \delta, q_0, \Gamma_s \rangle$ be an MDSTM. We assume without loss of generality that $\delta(q, [\$, Y], \epsilon)$ is defined for all q and Y . Define the DSTM $M' = \langle Q', \Sigma, \Gamma', \delta', q_0, F \rangle$, where

$$Q' = \{q_0\} \cup \{ [q, Y] \mid q \in Q, Y \in (\Gamma - \{\$ \}) \},$$

$$\Gamma' = (\{ \$, \lambda \} \cup (\Gamma - \{\$ \}) \times (\Gamma - \{\$ \})),$$

$$F = \{ [q, Y] \mid q \in Q, [\$, Y] \in \Gamma_s \},$$

and δ' is defined as follows:

$$(1) \quad \delta'(q_0, \lambda, a) = ([p, Y'], [Y', Z], -1)$$

$$\text{if } \delta(q_0, \lambda, a) = (p, Z, -1) \text{ and } \delta(p, [\$, \lambda], \epsilon) = (q_0, [\$, Y'], +1),$$

- (2) $\delta'([q, X], [Y, Z], \epsilon) = ([p, Y'], [Y', Z'], -1)$
 if $\delta(q, Z, \epsilon) = (p, Z', -1)$ and $\delta(p, [\$, Y], \epsilon) = (q_0, [\$, Y'], +1)$,
- (3) $\delta'([q, Y], \$, \epsilon) = (q_0, \$, +1)$,
- (4) $\delta'(q_0, [Y, Z], \epsilon) = (q_0, [Y, Z], +1)$.

It is straightforward to check that $L(M') = L(M)$. Hence $\mathcal{L}(\text{MDSTM}) \subseteq (\text{DSTM})$. It follows that $\mathcal{L}(\text{DSTM}) = \mathcal{L}(\text{MDSTM})$. \square

Theorem 4.8. $\mathcal{L}(\text{DHTA})$ is closed under concatenation with regular sets (to the left or to the right).

Proof. Since $\mathcal{L}(\text{DHTA})$ and regular sets are closed under reversal (see [5] for $\mathcal{L}(\text{DHTA})$), we only need to prove the case when concatenation is to the right.

Let L be accepted by a DSTM $M_1 = \langle Q_1, \Sigma, \Gamma_1, \delta_1, q_{01}, F_1 \rangle$, and R be accepted by a deterministic finite automation $M_2 = \langle Q_2, Z, \delta_2, q_{02}, F_2 \rangle$. We construct an MDSTM M_3 which accepts LR . M_3 's leftmost cell is subdivided into $|Q_2| + 1$ subcells including a subcell for $\$$ (see Fig. 10). Given an input, M_3 starts simulating M_1 .

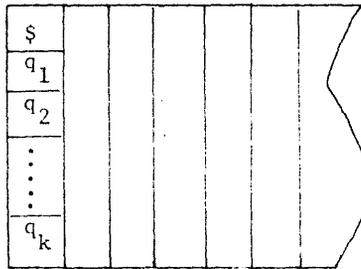


Fig. 10. The worktape of M_3 , $k = |Q_2|$.

Whenever M_1 visits the leftmost cell in an accepting state of M_1 , M_3 initiates the simulation for M_2 recording the start state q_{02} of M_2 on one of the subcells. Once the simulation is initialized for M_2 , it is continued with each succeeding input symbol. Notice that M_1 may visit the leftmost cell in an accepting state of M_1 unbounded number of times, hence creating that many simulation instances for M_2 . But the number of current states to be recorded for the next step of the simulation is at most $|Q_2|$. The input is accepted if any of the 'current states' of the simulation for M_2 is an accepting state after reading the last symbol of the input. The details of the construction of M_3 are straightforward. \square

One can easily show using NSTM's that $\mathcal{L}(\text{NHTA})$ is an Abstract Family of Languages (i.e., closed under intersection with regular sets, inverse homomorphisms, ϵ -free homomorphisms, union, concatenation and Kleene-plus closure). Moreover, $\mathcal{L}(\text{NHTA})$ is closed under intersection. It follows (see [9]) that $\mathcal{L}(\text{NHTA})$ is closed

under many other operations, e.g., substitutions, ϵ -free nondeterministic gsm (generalized sequential machine) mappings, inverse nondeterministic gsm mappings, and limited erasing (i.e., for any language L in $\mathcal{L}(\text{NHTA})$ and any homomorphism h such that, for some fixed k , h does not map more than k consecutive symbols of any string in L to ϵ , $h(L)$ is in $\mathcal{L}(\text{NHTA})$).

Now $\mathcal{L}(\text{DHTA})$ contains the regular sets and is closed under the Boolean operations [5] (see Propositions 2.3 and 2.4). Also $\mathcal{L}(\text{DHTA})$ is closed under marked concatenation and marked Kleene-plus closure [5] (see Proposition 4.6). $\mathcal{L}(\text{DHTA})$ is not closed under letter-to-letter homomorphism [6]. However, it is closed under inverse homomorphism [6]. In fact, it is closed under inverse deterministic gsm mappings as we shall prove later. Thus, $\mathcal{L}(\text{DHTA})$ is an Abstract Family of Deterministic Languages [1], and it is closed under many other operations (see [9]).

The next proposition was already shown in [6]. Here, we give an alternative proof using DSTM's.

Proposition 4.9. *DSTM languages are closed under one-to-one length-multiplying homomorphisms ($h: \Sigma_1^* \rightarrow \Sigma_2^*$ is one-to-one and length-multiplying if $h(a) \neq h(b)$ for all $a \neq b$ in Σ_1 and there exists a $k \geq 1$ such that $|h(a)| = k$ for all a in Σ_1).*

Proof. Let M_1 be a DSTM accepting a language $L \subseteq \Sigma_1^+$. Let $h: \Sigma_1^* \rightarrow \Sigma_2^*$ be a one-to-one length-multiplying homomorphism with $|h(a)| = k$ for all a in Σ_1 . We construct a DSTM M_2 which accepts an input x in Σ_2^+ if and only if M_1 accepts $h^{-1}(x)$. M_2 has a buffer of size k and a worktape organized into two tracks. The top track will keep the input portion that has been read, and the bottom track will contain the inverse homomorphic image of the top track which has been processed by M_1 .

Let the input be $a_1 \cdots a_n$, and assume that M_2 is in the position of reading a_r , $1 \leq r \leq n$. The machine first writes the input a_r , and, moving to the left, copies every symbol (including a_r) it reads into the buffer (see Fig. 11). When the buffer is full, say with w , M_2 writes the 'effect' of M_1 on $h^{-1}(w)$ on the bottom track, i.e., it writes Z , if M_1 has $\delta(q_0, \lambda, h^{-1}(w)) = (q, Z, -1)$. Then M_2 simulates M_1 's action on every k th symbol that M_2 sees when making a right-to-left sweep. M_2 enters an accepting state if and only if M_1 enters an accepting state. Fig. 11 shows an accepting example, where Z_a, Z'_a, Z''_a indicate M_1 's first, second, third effects on a . The same notations are used for the other symbols. \square

$\mathcal{L}(\text{DHTA})$ is closed under inverse homomorphisms [6]. Our next result generalizes this.

Theorem 4.10. *$\mathcal{L}(\text{DHTA})$ is closed under inverse deterministic gsm (with accepting states) mappings.*

Proof. Let $L \subseteq \Delta^+$ be accepted by a DSTM M_1 and $G = \langle Q, \Sigma, \Delta, h, p_0, F \rangle$ be a deterministic gsm with accepting states [11], where h is a partial function from

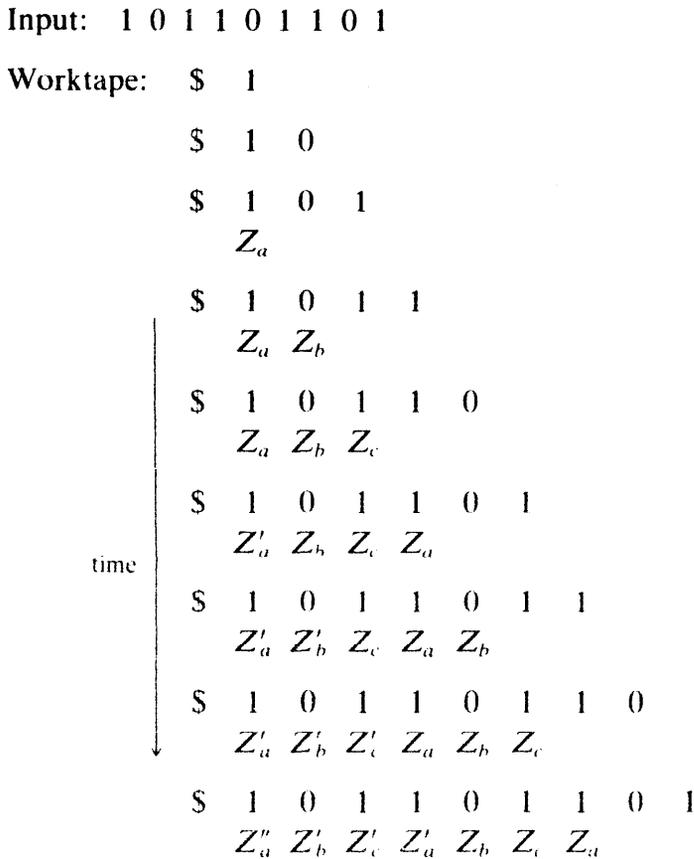


Fig. 11. The worktape profile of M_2 on input 101101101, where $h(a)=101$, $h(b)=011$, $h(c)=110$.

$Q \times \Sigma \rightarrow Q \times \Delta^*$. We describe briefly how a DSTM M_2 accepts the language $G^{-1}(L) = \{x \mid \text{there is a } y \text{ in } L \text{ such that } G, \text{ starting in state } p_0 \text{ on input } x, \text{ outputs } y \text{ and enters an accepting state}\}$.

Let $Q = \{q_1, \dots, q_n\}$ be the state set of G , where $p_0 = q_1$ is the start state. The worktape of M_2 has n tracks, one for each state in Q (see Fig. 12). M_2 simulates the computation of M_1 , but whenever a new input symbol "a" is read (when M_2 is on a λ , see Fig. 12), M_2 computes, for each $1 \leq i \leq n$ such that $h(q_i, a) \neq \emptyset$, the value $h(q_i, a) = (q_j, x_a)$, and writes, in one cell of row q_j , the 'effect' (i.e., the string that would have been written) on the worktape of M_1 due to input segment x_a . Thus, if w_a was to have been written, the composite symbol $[w_a]$ would be written instead (see Fig. 13). Then M_2 moves one cell left, changes $[w_b]$ in row q_i to $[w'_b]$ (as M_1 would), and stores a pointer to indicate that the state changed from q_i to q_j (see Figs. 12 and 13). Note that the pointer could just be the index j . M_2 then moves left towards "\$" simulating the changes that M_1 would have made on the worktape corresponding to the 'backward' path of pointers from row q_i . There are at most n backward paths since q_i can be any one of the n states. Thus, M_2 has to update the worktapes corresponding to these paths in parallel as it moves left to "\$". Note that updating can be done deterministically since the gsm is deterministic. M_2 enters

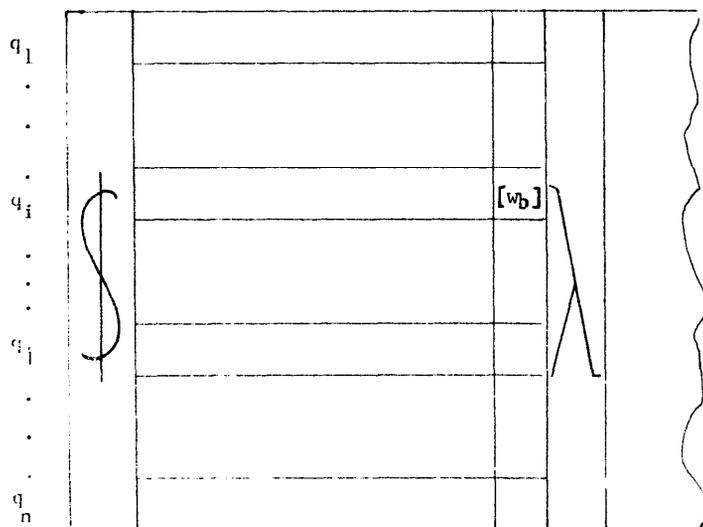


Fig. 12. Worktape of M_2 .

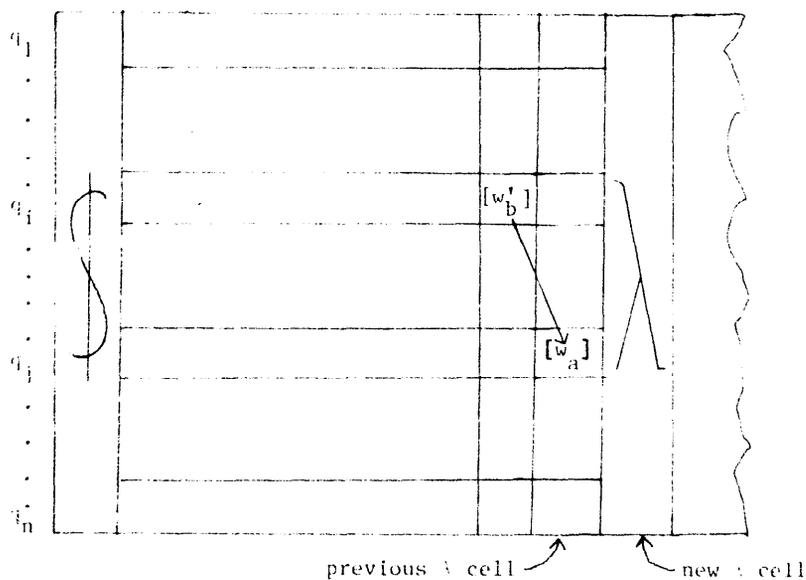


Fig. 13. Updating the worktape.

an accepting state if and only if there is a backward path that leads from a row corresponding to an accepting state of G to q_1 on the square containing $\$$ and M_1 accepts. \square

As one would expect, Theorem 4.10 does not hold when the gsm is nondeterministic. In fact, we have the following.

Proposition 4.11. *$\mathcal{L}(\text{DHTA})$ is not closed under inverse letter-to-letter (hence, finite) substitution.*

Proof. It is easy to show that for every recursively enumerable set $L \subseteq \Sigma^+$, we can find linear context-free languages (LCFL's) $L_1 \subseteq \Sigma^+ \Delta^*$ and $L_2 \subseteq \Sigma^+ \Delta^*$ with $\Delta \cap \Sigma = \emptyset$ such that $L = h(L_1 \cap L_2)$, where h is a homomorphism defined by $h(a) = a$ for each a in Σ and $h(b) = \varepsilon$ for each b in Δ . By Proposition 2.4 and the fact that the LCFL's are in $\mathcal{L}(\text{DHTA})$ [5] (see also Proposition 4.12 below), $L_3 = L_1 \cap L_2$ is in $\mathcal{L}(\text{DHTA})$. Assume that L is not recognizable by a deterministic LBA. Then $h'(L_3)$ is not in $\mathcal{L}(\text{DHTA})$, where h' is a homomorphism defined by $h'(a) = a$ for each a in Σ and $h'(b) = \#$ for each b in Δ (by Proposition 3.15). Now define the substitution s as follows: $s(a) = \{a\}$ for each a in Σ and $s(\#) = \{b \mid b \text{ in } \Delta\}$. Then $h'(L_3) = s^{-1}(L_3)$. It follows that $s^{-1}(L_3)$ is not in $\mathcal{L}(\text{DHTA})$. \square

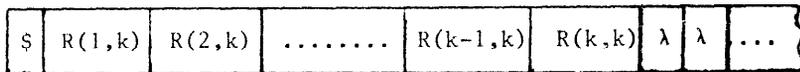
We conclude this section with a simple proof of a result which has already been shown in [5] using a different technique.

Proposition 4.12. *Every ε -free linear context free language (LCFL) is in $\mathcal{L}(\text{DHTA})$.*

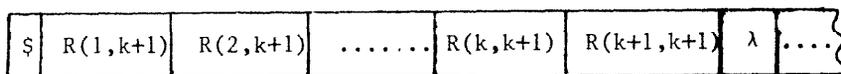
Proof. Let $G = \langle N, \Sigma, P, S \rangle$ be a linear context-free grammar with no ε -rules. Without loss of generality, assume that the rules in P are of the form $A \rightarrow aB$, $A \rightarrow Ba$, $A \rightarrow a$ where A, B are in N and a is in Σ . Let $x = a_1 \cdots a_n$ be in Σ^+ . For $1 \leq i \leq j \leq n$, define $R(i, j) = \{A \mid A \Rightarrow^* a_i \cdots a_j\}$. Then x is in $L(G)$ if and only if S is in $R(1, n)$. Now we observe that, for $1 \leq i < j \leq n$, A is in $R(i, j)$ if and only if one of the following holds:

- (1) $A \rightarrow a_i B$ is in P and B is in $R(i+1, j)$,
 - (2) $A \rightarrow B a_j$ is in P and B is in $R(i, j-1)$.
- (★)

We can construct a DSTM M which on input $x = a_1 \cdots a_n$ operates as follows: On reading input a_1 , M constructs the set $R(1, 1)$ on its worktape. Now assume that, after reading a_k , $1 \leq k < n$, M 's worktape is as shown in Fig. 14(a). Then, M , on its right-to-left sweep after reading a_{k+1} , can modify the worktape (using (★)) to that shown in Fig. 14(b). It follows that M can check, after reading a_n , whether S is in $R(1, n)$. \square



(a)



(b)

Fig. 14. Worktape of M .

5. Characterizations of regular trellis automata

In a DHTA $M = \langle \Sigma, \Gamma, \Delta, f, g \rangle$, the nodes of the trellis represent identical processors. Thus, all nodes (processors) compute the same f 's and g 's. The model can be generalized by allowing different processors which are capable of computing different functions f and g . A simple generalization is a *deterministic regular trellis automaton* (DRTA) introduced in [5]. A DRTA is an 8-tuple $M = \langle \Sigma, \Gamma, \Delta, N, \phi, l, F, G \rangle$, where Σ, Γ and Δ are as in a DHTA, N is a finite set of symbols called labels, ϕ is a distinguished symbol not in N , l is a labelling function from $\{(\phi, \phi)\} \cup (\{\phi\} \times N) \cup (N \times \{\phi\}) \cup N \times N$ into the set N , and $F = \{f_\alpha \mid \alpha \text{ in } N\}$ and $G = \{g_\alpha \mid \alpha \text{ in } N\}$ are sets of functions indexed by N , where f_α and g_α are defined as in a DHTA.

A string $a_1 \cdots a_n$ is accepted if we can construct a directed acyclic graph with weighted edges just like in a DHTA. However, the functions applied to a node with label α are f_α and g_α . The labels of the nodes are assigned in a top-down fashion according to the following rules:

Rule 1. The label of the root node is $l(\phi, \phi)$.

Rule 2. The label of a node whose left father is labeled a and whose right father is labeled b is $l(a, b)$. If the left (right) father does not exist, then $a(b)$ is ϕ .

The nondeterministic version of a DRTA (i.e., l, f_α and g_α need not be functions) is called *NRTA*.

We now define a generalization of a DSTM that will characterize DRTA. We provide a DSTM with another tape which is operated as a '1-turn' pushdown store. We call this new machine a *DRTM*. We shall only describe the operation of such a machine informally. The reader should have no difficulty formalizing the model. For an input $x = a_1 a_2 \cdots a_n$ of length n , the initial configuration of a DRTM M is shown in Fig. 15. The symbols $\$, \#, \lambda$ are distinguished symbols with λ representing blank. As before, "\$" appears only on the left end of the worktape. The operation of M consists of two phases. The first phase, called the *labelling phase*, ignores the input string and works as follows. M , in a distinguished state p_0 with its worktape head in "\$", moves right rewriting the #'s and changing states while at the same time pushing symbols onto the pushdown store. The pushdown store is treated as

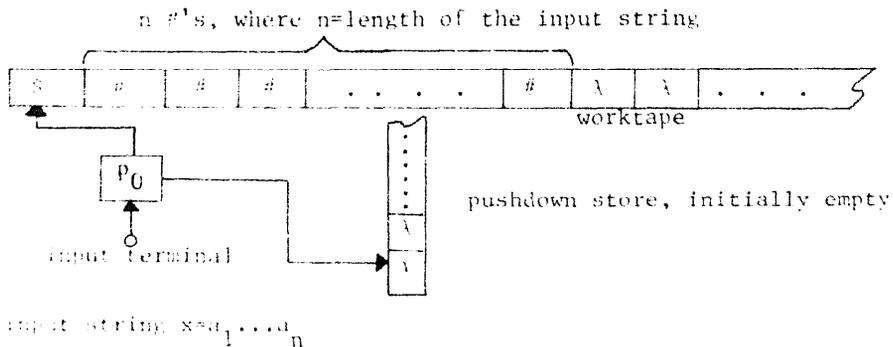


Fig. 15. Initial configuration of a DRTM M

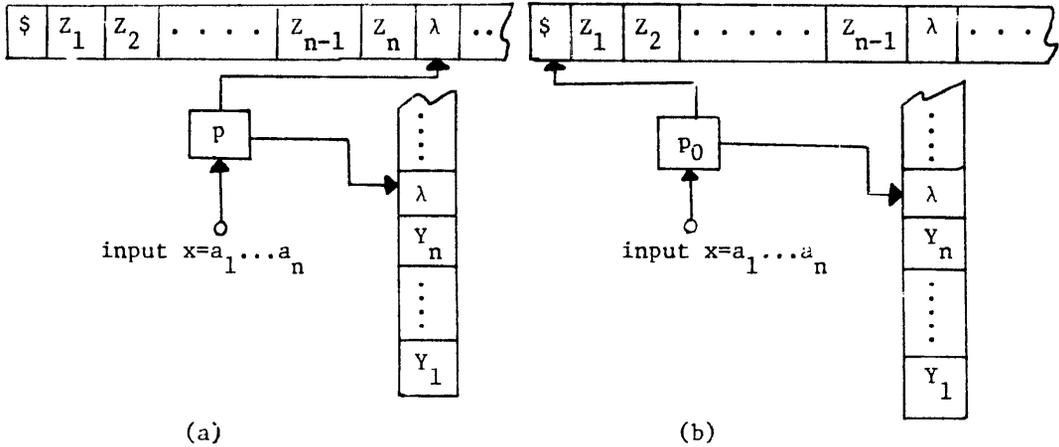


Fig. 16. Label generation by M . (a) After the first sweep to the right. (b) After the sweep back to S .

a write only tape. (Thus the top of the store does not influence the action of the machine during this labelling phase.) When the worktape head reads the leftmost λ , the configuration looks like Fig. 16(a). Then, without altering the pushdown store, the worktape head moves left and replaces Z_n by λ . Then M enters state p_0 and makes a right-to-left sweep to “ $\$$ ” while remaining in state p_0 . The worktape and pushdown store are not altered during the sweep. After this, M 's configuration looks like Figure 16(b). The process described above is repeated until the worktape contains only $\$ \lambda \lambda \dots$, and the next phase, called *computing phase*, is entered in a distinguished state q_0 (see Fig. 17(a)).

In the computing phase, the input string is processed. The processing and the operation on the worktape are done just like in a DSTM. As before, the machine remains in state q_0 without altering the worktape and the pushdown store when it is making a left-to-right sweep. However, the action of the machine when it is making a right-to-left sweep of the worktape now also depends on the topmost symbol of the pushdown store. Thus, whereas in a DSTM δ had the form

$$\delta(q, \lambda, a) = (p, Z, -1) \quad \text{or} \quad \delta(q, Z, \varepsilon) = (p, Z', -1)$$

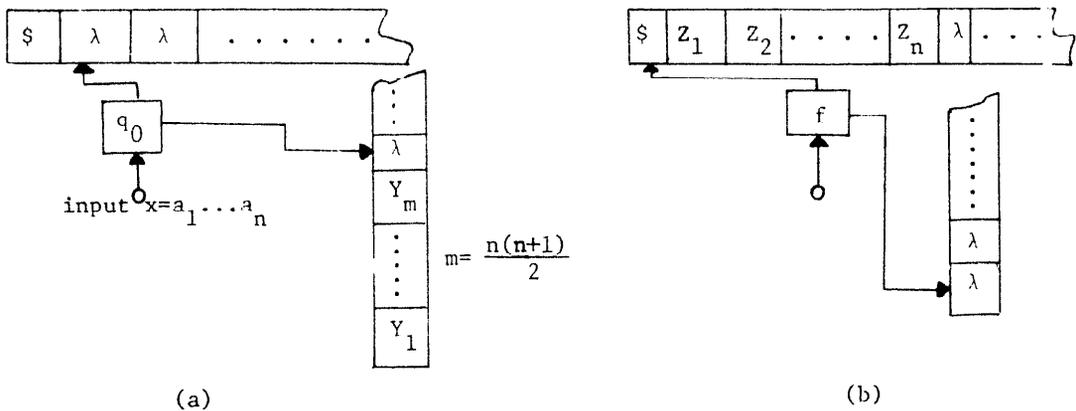


Fig. 17. Computing phase. (a) Initial. (b) Final.

in a DRTM, δ will now have the form

$$\delta(q, \lambda, a, W) = (p, Z, -1, \text{pop}) \quad \text{or} \quad \delta(q, Z, \varepsilon, W) = (p, Z', -1, \text{pop})$$

respectively, where W represents the topmost symbol of the pushdown store. Acceptance is defined as before. Thus, an accepting configuration looks like Fig. 17(b), where f is an accepting state. The nondeterministic version of a DRTM is called *NRTM*.

The proof of Theorem 3.1 can be generalized to show the following result.

Theorem 5.1. $\mathcal{L}(\text{DRTA}) = \mathcal{L}(\text{DRTM})$ and $\mathcal{L}(\text{NRTA}) = \mathcal{L}(\text{NRTM})$.

Proof. We only prove the deterministic case, the nondeterministic case being similar.

Let M_1 be a DRTA. We construct a DRTM M_2 which simulates M_1 . In the proof of Theorem 3.1, we showed how a DSTM can simulate a DHTA sequentially following the left oblique rows starting from the leftmost terminal node. Since the top stack symbol of a DRTM does not affect the direction of the worktape head, we can apply the same idea for this proof.

Now the problem is to generate the labels of the nodes and present them in an efficient way during the sequential simulation of the DRTA. It is easy to see that, during the labelling phase, M_2 can generate the labels in reverse order of the simulation sequence. For example, to generate the labels of the DRTA in Fig. 18,

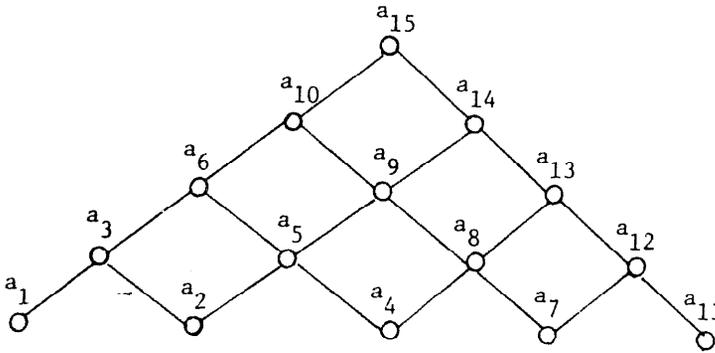


Fig. 18. A labelled DRTA.

M_2 , during its first left-to-right sweep, generates, on its worktape, the labels $a_{15}, a_{14}, \dots, a_{11}$ simulating the topdown labelling $a_{15} = l(\phi, \phi), a_{14} = l(a_{15}, \phi), \dots, a_{11} = l(a_{12}, \phi)$ for the top oblique row. With the labels $a_{15}, a_{14}, \dots, a_{11}$ on its worktape, M_2 generates the labels a_{10}, a_9, \dots, a_7 simulating the labelling $a_{10} = l(\phi, a_{15}), a_9 = l(a_{10}, a_{14}), \dots, a_7 = l(a_8, a_{12})$ for the next oblique row. This process is repeated throughout the labelling phase. Thus, at the end of the labelling phase, the stack will contain the labels $a_{15} \cdots a_2 a_1$ with a_1 the topmost symbol. This is the sequence needed for the simulation (see Fig. 18). Now it is easy to see that M_2 , during the computing phase, can simulate the DRTA M_1 using the labels generated in the stack. It follows that $L(M_2) = L(M_1)$.

Now let M_1 be a DRTM. We shall construct a DRTA M_2 which simulates M_1 . We will use the same technique as in the proof of Theorem 3.1. The labelling function is defined as follows (note that when M_1 moves from $\$$ to the right, in the labelling phase, it enters a unique state p_1):

(1) $l(\$, \phi) = [Z, Y, p]$ if M_1 , in state p_1 , rewrites $\#$ by Z , pushes Y onto the stack, and enters state p .

(2) $l([Z, Y, p], \phi) = [Z', Y', p']$ if M_1 , in state p (we assume that $p \neq p_1$), rewrites $\#$ by Z' , pushes Y' onto the stack, and enters state p' .

(3) $l(\$, [Z, Y, p]) = [Z', Y', p']$ if M_1 , in state p_1 rewrites Z by Z' , pushes Y' onto the stack, and enters state p' .

(4) $l([Z, Y, p], [Z', Y', p']) = [Z'', Y'', p'']$ if M_1 , in state p , rewrites Z' by Z'' , pushes Y'' onto the stack, and enters state p'' .

For a label $\alpha = [A, Y, p]$, the functions f_α and g_α are defined as in Theorem 3.1, i.e.,

$$(1) \quad f_\alpha(a) = \begin{cases} [q_1, Z_1] & \text{if } M_1, \text{ with top stack symbol } Y, \text{ reads } a, \text{ writes} \\ & \text{tape symbol } Z_1, \text{ and enters state } q_1, \\ D & \text{otherwise (the transition is undefined),} \end{cases}$$

$$(2) \quad g_\alpha(D, D) = g_\alpha([q_1, Z_1], D) = g_\alpha(D, [g_1, Z_1]) = D,$$

$$(3) \quad g_\alpha([q_1, Z_1], [q_2, Z_2]) = \begin{cases} [q'_1, Z'_1] & \text{if } M_1, \text{ with top stack symbol } Y \text{ and in} \\ & \text{state } q_2, \text{ rewrites } Z_1 \text{ by tape symbol} \\ & Z'_1 \text{ and enters state } q'_1, \\ D & \text{otherwise (the transition is undefined).} \end{cases}$$

It is straightforward to check that $L(M_2) = L(M_1)$. Hence, $\mathcal{L}(\text{DRTA}) = \mathcal{L}(\text{DRTM})$. \square

$\mathcal{L}(\text{DHNTA}) \subsetneq \mathcal{L}(\text{DRNTA})$ since $L = \{a^{2^n} \mid n \geq 0\}$ is not in $\mathcal{L}(\text{DHNTA})$ but in $\mathcal{L}(\text{DRNTA})$ [5] (see Example 5.4 below). However, we have the following.

Proposition 5.2. $\mathcal{L}(\text{NHNTA}) = \mathcal{L}(\text{NRNTA})$

Proof. $\mathcal{L}(\text{NHNTA}) \subseteq \mathcal{L}(\text{NRNTA})$ is obvious. Now let M_1 be an NRNTA. We construct an NHNTA M_2 which simulates M_1 as follows: Each node of M_2 , except the root node, guesses the label of the corresponding node of M_1 , simulates the mapping with the guessed label and the input, and passes the result along with the guessed label. The root node simulates the mapping of M_1 's root node. The guessed labels are checked as follows:

(1) Each node checks if the guessed label from its descendent diagonally below, if any, can be derived from the guessed labels from the two successor nodes.

(2) Each of the *leg*³ nodes (except the root node) checks if the guessed label from its successor leg node, if any, can be derived from its own guessed label.

³ The left oblique and the right oblique sides of a trellis automaton are called *legs*.

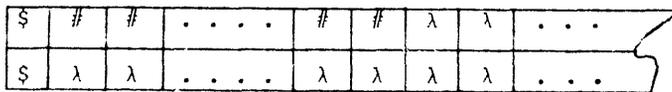
(3) The root node checks if the guessed labels from its two successors can be derived from the label of M_1 's root node.

If the 'label checking' fails, the simulation will be terminated, i.e., the node outputs a rejecting symbol. The reader should have no difficulty constructing M_2 formally. It follows that $\mathcal{L}(\text{NHTA}) = \mathcal{L}(\text{NRTA})$. \square

Now we present two examples of DRTM constructions.

Example 5.3. For each $k \geq 1$, $L_k = \{x^k \mid x \text{ in } \{0, 1\}^+\}$ is accepted by a DRTM. In Section 4 we showed that, for each $k \geq 1$, $\{(x\#)^k \mid x \text{ in } \{0, 1\}^+\}$ can be accepted by a DSTM. The marker “#” was used to locate the boundaries between the segments. The construction will still work even if the marker were deleted provided we can determine the boundaries between segments. We can do this in a DRTM by using the pushdown store (stack) to locate the boundaries. Thus, a DRTM accepting L_k first generates, during the labelling phase, the symbols (labels) in the stack in such a way that during the computation phase, the labels can signal the boundaries between segments. We shall only describe the labelling algorithm, the computation phase being similar to the construction in Section 4. We describe the case $k = 2$ and show how the construction can be generalized for arbitrary k . (Note that the case of $k = 1$ is obvious.)

At the beginning of the labelling phase, the worktape of a DRTM M can be thought of as a two-track tape of the form



where (ζ) , (λ) and (λ) are the same as $\$, \#$ and λ , respectively. On each left-to-right sweep (during the labelling phase), M replaces each leftmost λ below a “#” by a 1 and pushes the resulting tuple onto the stack. Thus, the stack has also two tracks. For example, for $n = 4$, the labelling profile will look like Fig. 19. Then, during the computation phase, M considers the stack tuple $(\#, \lambda)$ as a signal indicating the position of the first symbol of the second half of the input xy , $|x| = |y|$. M can check that $x = y$ using the ‘shifting’ technique described in Section 4 for accepting $\{x\#x \mid x \text{ in } \{0, 1\}^*\}$ by a DSTM. In the algorithm above, each leftmost λ below “#” is replaced by a 1 for every λ that is written on the right end of the worktape. The construction can be generalized for any $k \geq 2$. The worktape will now look like Fig. 20. During the labelling phase, for each track i , $1 \leq i \leq k - 1$, $k - i$ leftmost λ 's below $\#$'s are replaced by 1's for every i λ 's that are written on the right end.

$L = \{a^{2^n} \mid n \geq 0\}$ is in $\mathcal{L}(\text{DRTA})$ [5]. We give a DRTM construction based on the idea in [5].

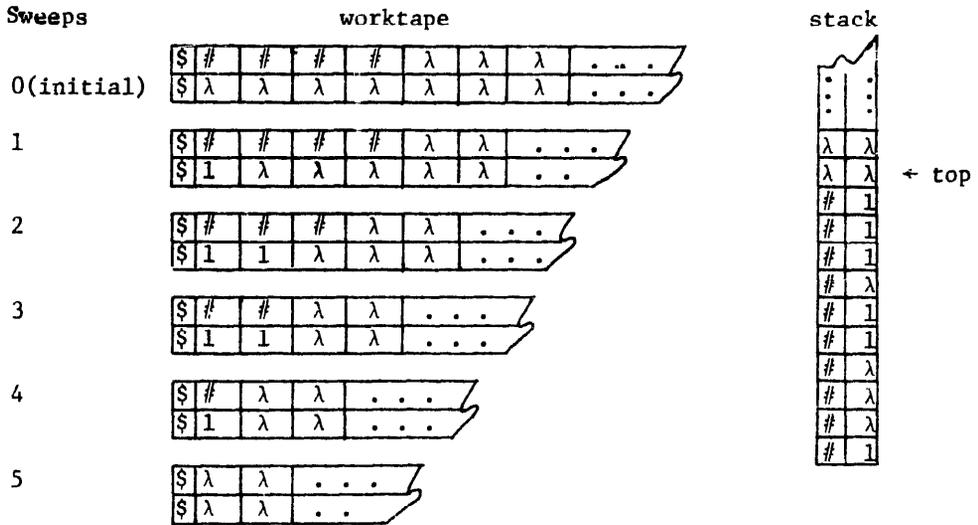


Fig. 19. Labelling profile of M .

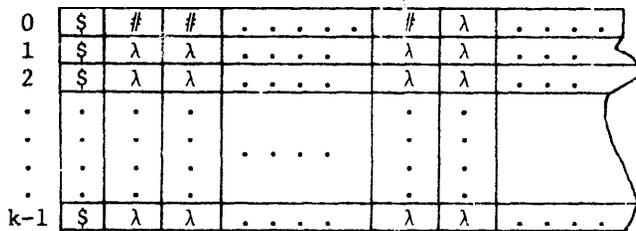


Fig. 20. Tape organization of M for the labelling to accept L_k .

Example 5.4. $L = \{a^{2^n} \mid n \geq 0\}$ is in $\mathcal{L}(\text{DRTM})$.

We construct a DRTM M which accepts L . As in Example 5.3, we program the labelling phase of M such that the labels (i.e., stack symbols) can be used as a control information for the computing phase. We solve this programming problem heuristically. We try with two stack symbols “ a ” and “ b ”. We assume that, during the computation phase, when M reads an input, the machine writes the input “ a ” on to the worktape if the stack symbol is “ a ” and writes “ b ” if the stack symbol is “ b ”. During the right-to-left sweep, M enters an accepting state if and only if the worktape has no b ’s. Except for the reading move, the stack symbol does not affect the computation of M . Now the problem is to program the labelling phase such that all the labels corresponding to the reading moves are a ’s if and only if the input length is 2^n for some $n \geq 0$. Notice that the stack symbols which are generated at the final steps of the left-to-right sweeps correspond (they are popped) to the reading moves of the computation phase. It follows that, as Fig. 21 shows, the last nonblank symbols of the worktape profile from the labelling phase should be all a ’s only for the case when the input length is 2^n , $n \geq 0$. For the other case

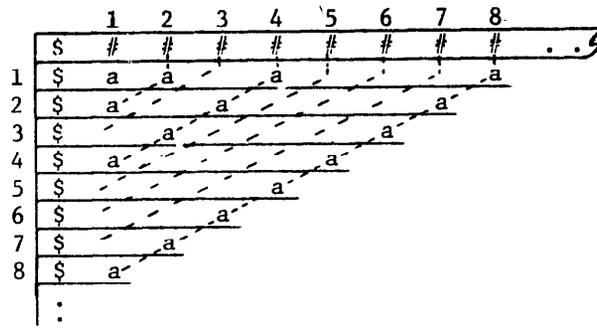


Fig. 21. Necessary entries of the worktape profile of M to accept $L_2 = \{a^{2^n} | n \geq 0\}$.

(empty oblique lines in Fig. 21) there should be at least one “ b ”. Let a move of M which rewrites the current tape symbol Z_c with Z after writing Z_l on the left neighboring cell be depicted as in Fig. 22(b). Then from the first two consecutive a ’s both from column 1 and row 1, we can assign three moves for M as shown in Fig. 22(c), (d), (e). It follows that all the empty entries on both column 1 and row 1 should be a ’s as shown in Fig. 22(a). Now, obviously, the entry for column 2 row 2 (circled one in the figure) should be “ b ”. So M needs the moves as shown in Fig. 22(f), (g), (h). If we fill the empty entries of Fig. 22(a) using the moves we have assigned so far, we can see that M needs the move Fig. 22(i) which gives the required entry “ b ” for column 4, row 3 (and column 3, row 4) which is circled. The moves of Fig. 22(c)–(i) are complete for the labelling phase and functionally the same as the labelling introduced in [5]. We refer the reader to [5] for the proof that the moves generate the stack symbols satisfying the condition for the computation phase of M for all input lengths.

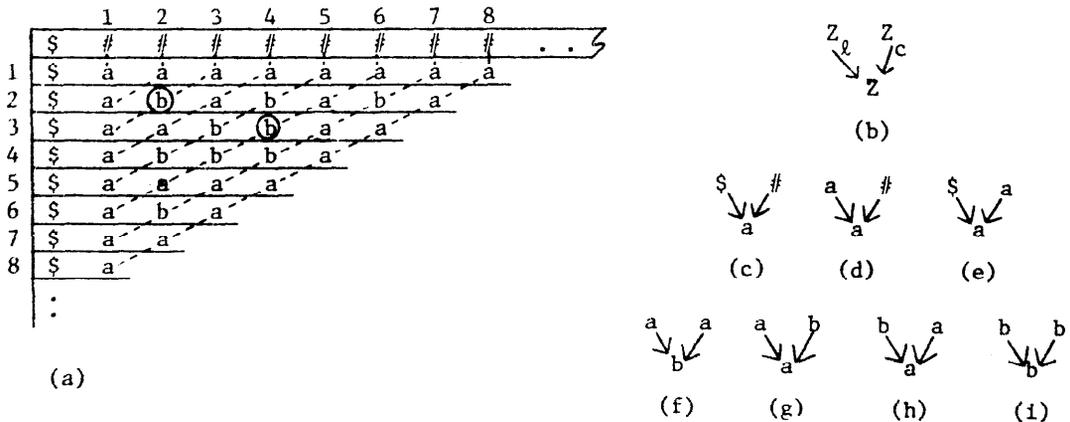


Fig. 22. (a) Worktape profile, (c)–(i) Moves of M .

References

[1] W. Chandler. Abstract families of deterministic languages, *Proc. 1st Ann. ACM Symp. on the Theory of Computing* (1969) pp. 21–30.
 [2] L. Conway and C. Mead, *Introduction to VLSI Systems* (Addison-Wesley, Reading, MA, 1980).

- [3] S. Cook, An observation on time-storage trade-off, *J. Comput. System Sci.* **9** (1974) 308–316.
- [4] S. Cook and R. Sethi, Storage requirements for deterministic polynomial time recognizable languages, *J. Comput. System Sci.* **13** (1976) 25–37.
- [5] K. Culik II, J. Gruska and A. Salomaa, Systolic trellis automata (for VLSI), Res. Rept. CS-81-34, Department of Computer Science, University of Waterloo, 1981; *Internat. J. Comput. Math.*, to appear.
- [6] K. Culik, II, J. Gruska and A. Salomaa, Systolic trellis automata: Stability, decidability and complexity, Res. Rept. CS-82-04, Department of Computer Science, University of Waterloo, 1982.
- [7] K. Culik, II, J. Gruska and A. Salomaa, Systolic automata for VLSI on balanced trees, *Acta Informatica* **18** (1983) 335–344.
- [8] K. Culik, II, A. Salomaa and D. Wood, VLSI systolic trees as acceptors, Res. Rept. CS-81-32, Department of Computer Science, University of Waterloo, 1981.
- [9] S. Ginsburg, *Algebraic and Automata-Theoretic Properties of Formal Languages* (North-Holland, Amsterdam, 1975).
- [10] M.A. Harrison, *Introduction to Formal Language Theory* (Addison-Wesley, Reading, MA, 1978).
- [11] J. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages, and Computations* (Addison-Wesley, Reading, MA, 1979).
- [12] J. Hopcroft and J.D. Ullman, An approach to a unified theory of automata, *Bell System Tech. J.* **46** (1967) 1763–1829.
- [13] R. Karp, Reducibility among combinatorial problems, in: *Complexity of Computer Computations* (Plenum, New York, 1972) pp. 85–104.
- [14] H. Kung, Let's design algorithms for VLSI systems, in: Ch.L. Seitz, ed., *Proc. Caltech Conf. on Very Large Scale Integration*, Pasadena, CA (1979) pp. 65–90.
- [15] H. Kung, The structure of parallel algorithms, Res. Rept., Department of Computer Science, Carnegie-Mellon University, 1979.
- [16] H. Kung and C. Leiserson, Systolic arrays (for VLSI), in: I.S. Duff and G.W. Stewart, eds., *Proc. Sparse Matrix* (Society for Industrial and Applied Mathematics, 1979) pp. 256–282.
- [17] C. Leiserson and J. Saxe, Optimizing synchronous systems, *Proc. 22nd Ann. Symp. on Foundations of Computer Science*, Nashville, TN (1981) pp. 23–26.
- [18] W. Savitch, Relationship between nondeterministic and deterministic tape complexities, *J. Comput. System Sci.* **4** (1970) 177–192.