JOURNAL OF COMPUTER AND SYSTEM SCIENCES 10, 88–109 (1975)

Two-Way A-Transducers and AFL*

DONALD I. KIEL

California State University at Los Angeles, Los Angeles, California 90032

Received August 24, 1973

For each (abstract family of languages) AFL $\mathscr{L}$, two families of languages, the family $\mathscr{T}(\mathscr{L})$ of nondeterministic and the family $\mathscr{T}_d(\mathscr{L})$ of deterministic two-way a-transducer mappings of languages in $\mathscr{L}$ are defined. For each (abstract family of acceptors) AFA $\mathscr{D}$, two associated AFA, $\mathscr{D}_C$ and $\mathscr{D}_D$, are defined. It is proved that $\mathscr{L}(\mathscr{D}_C) = \mathscr{T}(\mathscr{L}(\mathscr{D}))$, $\mathscr{L}(\mathscr{D}_D) = \mathscr{T}_d(\mathscr{L}(\mathscr{D}))$, and $\mathscr{T}(\mathscr{L}(\mathscr{D}_D)) = \mathscr{L}(\mathscr{D}_C)$. If $\mathscr{L}$ is an AFL, then $\mathscr{T}(\mathscr{L})$ and $\mathscr{T}_d(\mathscr{L})$ are full AFL and are closed under reversal. If $\mathscr{L}$ is a full principal AFL, then so is $\mathscr{T}(\mathscr{L})$. If a full AFL $\mathscr{L}$ is closed under substitution, then so are $\mathscr{T}(\mathscr{L})$ and $\mathscr{T}_d(\mathscr{L})$. If $\mathscr{L}$ is a full AFL, then each one-letter language in $\mathscr{T}_d(\mathscr{L})$ is also in $\mathscr{L}$. For each AFL $\mathscr{L}$, $\mathscr{T}_d\mathscr{T}_d(\mathscr{L}) = \mathscr{T}_d(\mathscr{L})$ and $\mathscr{T}\mathscr{T}_d(\mathscr{L}) = \mathscr{T}(\mathscr{L})$. In contrast, if $\mathscr{L}$ is a subAFL of the family of context-free languages, then $\mathscr{T}_d\mathscr{T}(\mathscr{L}) \neq \mathscr{T}(\mathscr{L})$ and hence $\mathscr{T}\mathscr{T}(\mathscr{L}) \neq \mathscr{T}(\mathscr{L})$.

1. INTRODUCTION

Of the various finite-state transducers in the literature, the one which has recently been receiving the most attention is the one-way a-transducer. One reason for this is its prominent role in (abstract family of languages) AFL theory. For example, it was shown in [6] that full AFL are closed under one-way a-transducers. In view of the importance of one-way a-transducers for families of languages, the question arises as to the significance of two-way a-transducers for families of languages. The purpose of this paper is to investigate, in a general setting, the two-way a-transducer and its connection with families of languages, especially AFL.

Historically, a number of two-way finite-state devices have been studied in connection with families of languages. Two-way generalized sequential machines (2 gsm) were introduced by Aho and Ullman in [1] and used to characterize abstract families of two-way deterministic languages. Two-way sequential transducers (2 st) were defined by Ehrich and Yau in [2] and some properties of two-way mappings of regular

* This work forms part of the author's doctoral dissertation, written under the direction of Dr. Seymour Ginsburg at the University of Southern California. The research was supported in part by the National Science Foundation under Grant No. GJ-28787.

Copyright © 1975 by Academic Press, Inc.
All rights of reproduction in any form reserved.

and context-free languages were examined. In Theorem 4.8 of [2], the 2 st mapping of a regular set was shown to be equivalent to a special stack language related to the checking acceptor language [11]. Both nondeterministic and deterministic two-way finite-state transducers (2 nft and 2 ft) were defined by Rajlich in [13], and the equivalence of the family of languages generated by the 2 nft and the family of checking acceptor languages was shown.

In the initial phase of the present research, it was observed that the above mentioned results of Ehrich and Yau, and Rajlich could be combined to show the equivalence of the two-way a-transducer mappings of the regular sets to the family of the checking acceptor languages. It then seemed feasible to attempt to extend this result by characterizing, in a similar manner, the families of two-way a-transducer mappings of full AFL. This was accomplished by finding (abstract family of acceptors) AFA characterizations for both the families of nondeterministic and deterministic two-way a-transducer mappings of full AFL. (These AFA are generalizations of the family of checking acceptors, and differ only in that words from languages other than regular sets can be written on their checking stacks.) It was then discovered that these characterizations were most appropriate for studying many properties of the families of the nondeterministic and deterministic two-way a-transducer mappings of AFL.

This paper is divided into five sections in addition to the present introductory section. In Section 2, $\mathscr{T}$ and $\mathscr{T}_d$, certain families of nondeterministic and deterministic two-way a-transducers, respectively, are defined. It is shown that $\mathscr{T}(\mathscr{L})$ is closed under reversal and under one-way a-transducers for each family $\mathscr{L}$. If $\mathscr{L}$ is closed under marked $*$ and either marked $\cup$ or marked $\cdot$, then $\mathscr{T}(\mathscr{L})$ is a full AFL.

In Section 3, a structured-storage AFA $\mathscr{D}_C$, called the "AFA of auxiliary $\mathscr{D}$-storage checking acceptors," is associated with each AFA $\mathscr{D}$. It is shown that $\mathscr{L}(\mathscr{D}_C) = \mathscr{T}(\mathscr{L}(\mathscr{D}))$, where $\mathscr{T}(\mathscr{L}(\mathscr{D}))$ is the family of nondeterministic two-way a-transducer mappings of languages in $\mathscr{L}(\mathscr{D})$.

In Section 4, some properties which are preserved by $\mathscr{T}$ are studied. If $\mathscr{L}$ is a full AFL closed under substitution or properly contained in the family of context-sensitive languages, then so is $\mathscr{T}(\mathscr{L})$. If $\mathscr{L}$ is a full principal AFL, then so is $\mathscr{T}(\mathscr{L})$, and a full generator for $\mathscr{T}(\mathscr{L})$ can be constructed from a full generator for $\mathscr{L}$.

In Section 5, a structured-storage AFA $\mathscr{D}_D$, called the "AFA of auxiliary $\mathscr{D}$-storage counting checking acceptors," is associated with each AFA $\mathscr{D}$. It is shown that $\mathscr{L}(\mathscr{D}_D) = \mathscr{T}_d(\mathscr{L}(\mathscr{D}))$, the family of deterministic two-way a-transducer mappings of $\mathscr{L}(\mathscr{D})$. It is also noted that closure under substitution is preserved by $\mathscr{T}_d$. Finally, it is proved that each one-letter language in $\mathscr{T}_d(\mathscr{L})$, for $\mathscr{L}$ a full AFL, is also in $\mathscr{L}$.

In Section 6, the composition of $\mathscr{T}$ and $\mathscr{T}_d$ is studied. It is first shown that for each AFL $\mathscr{L}$, $\mathscr{T}_d\mathscr{T}_d(\mathscr{L}) = \mathscr{T}_d(\mathscr{L})$ and $\mathscr{T}\mathscr{T}_d(\mathscr{L}) = \mathscr{T}(\mathscr{L})$. As a corollary, it follows that $\mathscr{T}(\mathscr{L}(\mathscr{D}_D)) = \mathscr{L}(\mathscr{D}_C)$ for each AFA $\mathscr{D}$. Another consequence is that $\mathscr{T}_d(\mathscr{L})$ is a full AFL closed under reversal for each AFL $\mathscr{L}$. Necessary and sufficient conditions for $\mathscr{T}_a\mathscr{T}(\mathscr{L}) = \mathscr{T}(\mathscr{L})$ and $\mathscr{T}\mathscr{T}(\mathscr{L}) = \mathscr{T}(\mathscr{L})$ are then given. These conditions are used to show that if $\mathscr{L}$ is a subAFL of the family of context-free languages, then $\mathscr{T}_a\mathscr{T}(\mathscr{L}) \neq \mathscr{T}(\mathscr{L})$ and hence $\mathscr{T}\mathscr{T}(\mathscr{L}) \neq \mathscr{T}(\mathscr{L})$.

2. Two-Way A-Transducers

As mentioned in the previous section, the purpose of this paper is to study the connection between two-way a-transducers and families of languages, especially AFL. We assume that the reader is familiar with the concepts of AFL theory, especially as presented in [3, 6, 7, 8, 10, and 12]. In this section we define two-way a-transducers and prove some elementary results concerning two-way a-transducers and families of languages.

We begin with the definition of two-way a-transducers.

Definition. A "two-way a-transducer" is an 8-tuple $T = (K, \Sigma_1, \Sigma_2, H, s_0, ¢, \$, F)$ where

(1) K, Σ_1, and Σ_2 are finite sets of "states," "input symbols," and "output symbols," respectively;

(2) H is a finite subset of $K \times \Sigma_1 \times \Sigma_2{}^* \times K \times \{-, 0, +\}$;

(3) s_0, the "start state," is in K;

(4) ¢ and \$ are special symbols in Σ_1 ("left" and "right endmarkers," respectively); and

(5) F, the set of "final states," is a subset of K.

Remark. The terms "two-way a-transducer" and "nondeterministic two-way a-transducer" will be used synonomously.

We now define the moves of the two-way a-transducer.

Definition. Let $\vdash$ be a relation on $K \times (\Sigma_1 \cup \{\urcorner\})^* \times \Sigma_2{}^*$ defined as follows (for b_1, b_2 in $\Sigma_2{}^*$, $\urcorner$ a new symbol, and x, $\bar{x}$ in $\Sigma_1{}^*$, x_1, x_2, x_3 in Σ_1, y in $(\Sigma_1 - \{¢, \$\})^*$ such that $xx_1x_2\bar{x} = xx_3\bar{x} = ¢y\$$).

(1) $(p, xx_1x_2 \urcorner \bar{x}, b_1) \vdash (q, xx_1 \urcorner x_2\bar{x}, b_1b_2)$ if $(p, x_2, b_2, q, -)$ is in H;

(2) $(p, xx_1 \urcorner x_2\bar{x}, b_1) \vdash (q, x_1x_2 \urcorner \bar{x}, b_1b_2)$ if $(p, x_1, b_2, q, +)$ is in H; and

(3) $(p, xx_3 \urcorner \bar{x}, b_1) \vdash (q, xx_3 \urcorner \bar{x}, b_1b_2)$ if $(p, x_3, b_2, q, 0)$ is in H.

Let $\overset{*}{\vdash}$ be the reflexive, transitive closure of $\vdash$.

Notation. For each two-way a-transducer $T = (K, \Sigma_1, \Sigma_2, H, s_0, ¢, \$, F)$ and each w in $(\Sigma_1 - \{¢, \$\})^*$, let $T(w) = \{z \text{ in } \Sigma_2^* \mid (s_0, ¢ \upharpoonright w\$, \epsilon) \overset{*}{\vdash} (p, ¢w\$ \upharpoonright, z)$ for some p in $F\}$. For every $L \subseteq (\Sigma_1 - \{¢, \$\})^*$, let $T(L) = \bigcup_{w \text{ in } L} T(w)$.

We now define a subset of the set of all two-way a-transducers.

Notation. Let $\mathscr{T}$ be the family of all two-way a-transducers such that H is a finite subset of $K \times \Sigma_1 \times (\Sigma_2 \cup \{\epsilon\}) \times K \times \{-, 0, +\}$. For $\mathscr{L}$, a family of languages, let $\mathscr{T}(\mathscr{L}) = \{T(L) \mid T \text{ in } \mathscr{T}, L \text{ in } \mathscr{L}\}$. Clearly, $\mathscr{T}(\mathscr{L}) = \{T(L) \mid T$ a two-way a-transducer, L in $\mathscr{L}\}$.

Note that the two-way a-transducer defined here is a nondeterministic variation of the 2 gsm defined by Aho and Ullman in [1].

The two-way a-transducer differs from the two-way sequential transducer (2 st) of Ehrich and Yau [2] in that the 2 st has no endmarkers and no accepting states. The 2 st accepts by moving its pointer to the blank at the right of the input word. Ehrich and Yau discuss $\mathscr{S}_2(\mathscr{R})$ and $\mathscr{S}_2(\mathscr{L}_{\text{CF}})$, the 2 st mappings of $\mathscr{R}$, the family of regular sets, and of $\mathscr{L}_{\text{CF}}$, the family of context-free languages, respectively. They prove that $\mathscr{S}_2(\mathscr{R})$ and $\mathscr{S}_2(\mathscr{L}_{\text{CF}})$ are full AFL closed under reversal and substitution. It can be shown that $\mathscr{S}_2(\mathscr{R}) = \mathscr{T}(\mathscr{R})$ and $\mathscr{S}_2(\mathscr{L}_{\text{CF}}) = \mathscr{T}(\mathscr{L}_{\text{CF}})$, the proof depending on the fact that L is in $\mathscr{R}$ (or $\mathscr{L}_{\text{CF}}$) if and only if $¢L\$$ is in $\mathscr{R}$ (or $\mathscr{L}_{\text{CF}}$).

Finally, the two-way a-transducer may be compared with the two-way nondeterministic finite-state transducer (2 nft) introduced by Rajlich [13]. Each 2 nft uses a regular set Σ_1^* as input and generates a language in the family $\mathscr{L}_{2\text{NFT}}$. It can be proved that $\mathscr{L}_{2\text{NFT}} = \mathscr{T}(\mathscr{R})$, the proof depending on the fact that the finite-state control of the 2 nft is used to check whether or not an input word in Σ_1^* is in some particular regular set R.

Definition. A "deterministic two-way a-transducer" $T = (K_1, \Sigma_1, \Sigma_2, H, s_0, ¢, \$, F)$ is a two-way a-transducer such that

(1) for each p in K_1 and a in Σ_1, there is at most one 5-tuple (p, a, b, q, d) in H, and

(2) for p in F there is no 5-tuple in H of the form $(p, \$, b, q, d)$.

Remark. The deterministic two-way a-transducer defined here is equivalent to the 2 gsm of [1].

Notation. Let $\mathscr{T}_d$ be the family of all deterministic two-way a-transducers $(K_1, \Sigma_1, \Sigma_2, H, s_0, ¢, \$, F)$ for which b is in $\Sigma_2 \cup \{\epsilon\}$ if (p, a, b, q, d) is in H. For each family $\mathscr{L}$ of languages, let $\mathscr{T}_d(\mathscr{L}) = \{T(L) \mid T \text{ is in } \mathscr{T}_d, L \text{ is in } \mathscr{L}\}$. It is clear that $\mathscr{T}_d(\mathscr{L}) = \{T(L) \mid T$ is a deterministic two-way a-transducer, L is in $\mathscr{L}\}$.

Remark. For each $T = (K_1, \Sigma_1, \Sigma_2, H, s_0, ¢, \$, F)$ in $\mathscr{T}_d$, each symbol of the input word is read fewer than $\#(K_1) + 1$ times in any accepting computation.[1]

[1] For each set K, $\#(K)$ denotes the number of elements in K.

We now present some elementary results concerning $\mathscr{T}(\mathscr{L})$, for families of languages $\mathscr{L}$.

THEOREM 2.1. *For each family of languages $\mathscr{L}$, $\mathscr{T}(\mathscr{L})$ is closed under reversal and under one-way a-transducers.*[2]

Proof. If L is in $\mathscr{L}$ and T is in $\mathscr{T}$, there exists T' in $\mathscr{T}$ such that $T'(L) = [T(L)]^R$. T' operates as follows. For a word w in L, T' moves to the right end of $¢w\$$, goes to a final state of T, and then simulates, in reverse, the moves of T. (For example, if T reads a in state p, writes b, and moves right to a' in state q, then T' reads a' in state (q, a'), writes b, and moves left in state (p, a). T' continues only if it now reads a.) If T' reaches $¢$ in the start state of T, then T' moves to the right end of $¢w\$$ in an accepting state. Thus $\mathscr{T}(\mathscr{L})$ is closed under reversal.

For L in $\mathscr{L}$ and a one-way a-transducer M, there exists $\bar{T}$ in $\mathscr{T}$ such that $\bar{T}(L) = M(T(L))$. ($\bar{T}$ is constructed so that it uses pairs of states of M and T to simulate each in turn.) Thus $\mathscr{T}(\mathscr{L})$ is closed under one-way a-transducers.

THEOREM 2.2. *If $\mathscr{L}$ is a family of languages closed under marked $*$ and either marked $\cup$ or marked $\cdot$, then $\mathscr{T}(\mathscr{L})$ is a full* AFL.[3]

Proof. If $\mathscr{L}$ is closed under marked $*$, then for L in $\mathscr{L}$, $(cL)^*$ is in $\mathscr{L}$. For T in $\mathscr{T}$, there exists T' in $\mathscr{T}$ such that $T'((cL)^*) = [T(L)]^*$. Hence $\mathscr{T}(\mathscr{L})$ is closed under $*$ if $\mathscr{L}$ is closed under marked $*$.

If $\mathscr{L}$ is closed under marked $\cup$, then $\mathscr{T}(\mathscr{L})$ is closed under $\cup$, since for L_1, L_2 in $\mathscr{L}$ and T_1, T_2 in $\mathscr{T}$, there exists T_3 in $\mathscr{T}$ such that $T_3(c_1L_1 \cup c_2L_2) = T_1(L_1) \cup T_2(L_2)$.

If $\mathscr{L}$ is closed under marked $\cdot$, then $\mathscr{T}(\mathscr{L})$ is closed under $\cdot$, since for L_1, L_2 in $\mathscr{L}$ and T_1, T_2 in $\mathscr{T}$, there exists T_3 in $\mathscr{T}$ such that $T_3(L_1cL_2) = T_1(L_1) \cdot T_2(L_2)$.

Thus, if $\mathscr{L}$ is closed under marked $*$ and marked $\cup$, then $\mathscr{T}(\mathscr{L})$ is a full AFL by Theorem 2 of [12], since it is closed under one-way a-transducers, $*$, and $\cup$. If $\mathscr{L}$ is closed under marked $*$ and marked $\cdot$, then $\mathscr{T}(\mathscr{L})$ is a full AFL by Theorem 1 of [12], since it is closed under one-way a-transducers, $*$, and $\cdot$.

COROLLARY. *If $\mathscr{L}$ is an* AFL, *then $\mathscr{T}(\mathscr{L})$ is a full* AFL.

Remark. It is shown in Corollary 2 of Theorem 6.1 that $\mathscr{T}_d(\mathscr{L})$ is a full AFL closed under reversal if L is an AFL.

We close this section with a result which be useful later.

THEOREM 2.3. *For each* AFL $\mathscr{L}$, $\mathscr{T}(\mathscr{L}) = \mathscr{T}(\hat{\mathscr{F}}(\mathscr{L}))$ and $\mathscr{T}_d(\mathscr{L}) = \mathscr{T}_d(\hat{\mathscr{F}}(\mathscr{L}))$.[4]

[2] The reversal of the work $w = x_1 \cdots x_n$, $n \geqslant 0$, each x_i in Σ, is the word $w^R = x_n \cdots x_1$. The reversal of the language L is the language $L^R = \{w^R \mid w \text{ in } L\}$.

[3] The author is grateful to the referee for suggesting this form of the theorem.

[4] For each set of languages $\mathscr{L}$, $\hat{\mathscr{F}}(\mathscr{L})$ is the smallest full AFL containing $\mathscr{L}$.

Proof. We first prove that $\mathscr{T}(\mathscr{L}) = \mathscr{T}(\hat{\mathscr{H}}(\mathscr{L}))$,[5] For an arbitrary homomorphism h_1, T_1 in $\mathscr{T}$, and L in $\mathscr{L}$, there exists T_2 in $\mathscr{T}$ and an ϵ-free homomorphism h_2 such that $T_2(h_2(L)) = T_1(h_1(L))$. (If $h_1(a) \neq \epsilon$, let $h_2(a) = h_1(a)$ and if $h_1(a) = \epsilon$, let $h_2(a) = e$. T_2 writes no output while reading e, but otherwise simulates T_1, remembering in its state the direction T_1 is moving.) Thus $\mathscr{T}(\mathscr{L}) = \mathscr{T}(\hat{\mathscr{H}}(\mathscr{L}))$, since $\mathscr{L}$ is closed under ϵ-free homomorphism. Hence $\mathscr{T}(\mathscr{L}) = \mathscr{T}(\hat{\mathscr{F}}(\mathscr{L}))$, since if $\mathscr{L}$ is an AFL, then $\hat{\mathscr{H}}(\mathscr{L}) = \hat{\mathscr{F}}(\mathscr{L})$ by Lemma 2.2 of [9]. The proof that $\mathscr{T}_d(\mathscr{L}) = \mathscr{T}_d(\hat{\mathscr{F}}(\mathscr{L}))$ is similar.

3. Auxiliary $\mathscr{D}$-Storage Checking Acceptors

In this section we associate with each AFA $\mathscr{D}$ a structured-storage AFA $\mathscr{D}_C$, called the "AFA of auxiliary $\mathscr{D}$-storage checking acceptors." We then show in Theorem 3.1 that $\mathscr{L}(\mathscr{D}_C)$, the family of languages accepted by $\mathscr{D}_C$, is $\mathscr{T}(\mathscr{L}(\mathscr{D}))$, the family of two-way a-transducer mappings of languages in $\mathscr{L}(\mathscr{D})$. This result will play a key role in the study of the nondeterministic two-way a-transducer mappings of AFL.

Given an AFA $\mathscr{D}$, we now informally define $\mathscr{D}_C$. Acceptors in $\mathscr{D}_C$ have two types of storage, one a stack and the other an auxiliary storage of the same type as $\mathscr{D}$. Each acceptor in $\mathscr{D}_C$ operates in two modes, a writing mode and a checking mode. The auxiliary storage is used only while the acceptor is in the writing mode and it is used in such a way that the words written on the stack form a language in $\mathscr{L}(\mathscr{D})$. Once the word is written on the stack the acceptor operates in the checking mode. In this mode, the acceptor may enter the stack but no further writing is allowed until the storage is erased (in one step). The formal definition follows.

Definition. For each AFA $(\Omega, \mathscr{D})$, where $\Omega = (K, \Sigma, \Gamma_2, I_2, f_2, g_2)$, let $\Gamma_1 = \Sigma$, and $I_1 = \Gamma_1 \cup \Gamma_1 \urcorner \cup \{-, +, 0, E\}$ with $\urcorner$, $-$, $+$, 0 and E new symbols. Let f be the function from $(\Gamma_1^* \times \Gamma_2^*) \times (I_1 \times I_2)$ into $(\Gamma_1^* \times \Gamma_2^*) \cup \{\varnothing\}$ and g be the function from $\Gamma_1^* \times \Gamma_2^*$ into the finite subsets of $\Gamma_1^* \times \Gamma_2^*$ defined as follows (for x, $\bar{x}$ in Γ_1^*, x_1, x_2 in Γ_1, γ_2 in Γ_2^*, and u_2 in I_2).

(1) The "writing" mode:

a. $f((x, \gamma_2), (x_1, u_2)) = (xx_1, f_2(\gamma_2, u_2))$.[6]

b. $f((x, \gamma_2), (0, u_2)) = (x, f_2(\gamma_2, u_2))$.

c. $f((x, \gamma_2), (x_1 \urcorner, u_2)) = (xx_1 \urcorner, f_2(\gamma_2, u_2))$, where $f_2(\gamma_2, u_2) = \epsilon$.

[5] $\hat{\mathscr{H}}(\mathscr{L}) = \{h(L) \mid L \text{ in } \mathscr{L} \text{ and } h \text{ an arbitrary homomorphism}\}$.

[6] If $f_2(\gamma_2, u_2) = \{\varnothing\}$, then $f((x, \gamma_2), (u_1, u_2)) = \{\varnothing\}$ for each u_1 in I_1.

(2) The "checking" mode:

a. $f((xx_1x_2 \upharpoonleft \bar{x}, \epsilon), (-, 1_\epsilon)) = (xx_1 \upharpoonleft x_2\bar{x}, \epsilon)$.

b. $f((xx_1 \upharpoonleft x_2\bar{x}, \epsilon), (+, 1_\epsilon)) = (xx_1x_2 \upharpoonleft \bar{x}, \epsilon)$.

c. $f((x \upharpoonleft \bar{x}, \epsilon), (0, 1_\epsilon)) = (x \upharpoonleft \bar{x}, \epsilon)$.

d. $f((xx_1 \upharpoonleft, \epsilon), (\mathrm{E}, 1_\epsilon)) = (\epsilon, \epsilon)$.

(3) a. $g((\epsilon, \epsilon)) = \{(\epsilon, \epsilon)\}$.

b. $g((xx_1, \gamma_2)) = \{(x_1, \gamma) | \gamma \text{ in } g_2(\gamma_2)\}$.

c. $g((xx_1 \upharpoonleft \bar{x}, \epsilon)) = \{(x_1 \upharpoonleft, \epsilon)\}$.

Then $(\Omega_C, \mathscr{D}_C)$, where $\Omega_C = (K, \Sigma, \Gamma_1 \times \Gamma_2, I_1 \times I_2, f, g)$, is called the AFA of "auxiliary $\mathscr{D}$-storage checking acceptors." (We use $\mathscr{D}_C$ when Ω_C is understood.) Note that what we have called the AFA of auxiliary $\mathscr{D}$-storage checking acceptors is technically not an AFA. However, we shall soon observe that it is closely related to an AFA.

We now define the moves of acceptors in $(\Omega_C, \mathscr{D}_C)$.

Definition. Let $D = (K_1, \Sigma_1, \delta, q_0, F)$ be in $\mathscr{D}_C$. *A* "configuration" C is a triple $C = (q, w, (\gamma_1, \gamma_2))$ where q is in K_1, w is in $\Sigma_1{}^*$, γ_1 is in $\Gamma_1{}^*$, and γ_2 is in $\Gamma_2{}^*$.

Notation. Let $\vdash$ be a relation on configurations defined as follows (for a in $\Sigma_1 \cup \{\epsilon\}$ and w in $\Sigma_1{}^*$): $(q, aw, (\gamma_1, \gamma_2)) \vdash (q', w, (\gamma_1', \gamma_2'))$ if there exist $(\bar{\gamma}_1, \bar{\gamma}_2)$ in $g(\gamma_1, \gamma_2)$ such that $(q', (u_1, u_2))$ is in $\delta(q, a, (\bar{\gamma}_1, \bar{\gamma}_2))$ and $f((\gamma_1, \gamma_2), (u_1, u_2)) = (\gamma_1', \gamma_2')$. Let $\overset{*}{\vdash}$ be the transitive, reflexive closure of $\vdash$.

An auxiliary $\mathscr{D}$-storage checking acceptor defines a language as follows.

Notation. For each $D = (K_1, \Sigma_1, \delta, q_0, F)$ in $\mathscr{D}_C$, let $L(D) = \{w | (q_0, w, (\epsilon, \epsilon)) \overset{*}{\vdash} (q, \epsilon, (\epsilon, \epsilon))$ for some q in $F\}$ and let $\mathscr{L}(\mathscr{D}_C) = \{L(D) | D \text{ in } \mathscr{D}_C\}$.

It is easily seen that the AFA of auxiliary $\mathscr{D}$-storage checking acceptors is a special instance of a structured-storage AFA as defined in [3]. Hence, by Theorem 1 of [3], it follows that $\mathscr{L}(\mathscr{D}_C)$ is a full AFL.

We now consider $\bar{\mathscr{D}}_C$, a special family of auxiliary $\mathscr{D}$-storage checking acceptors which

(1) do no reading of input while in the writing mode, and

(2) use the E instruction only once in accepting a word.

Definition. Let $\bar{\mathscr{D}}_C$ be the family of all $D = (K_1, \Sigma_1, \delta, p_0, F)$ in $\mathscr{D}_C$ such that δ has the following properties.

(1) If $\delta(p, a, (y, z))$ is defined for p in K_1, y in $\Gamma_1 \cup \{\epsilon\}$ and z in $g_2(\Gamma_2{}^*)$, then $a = \epsilon$;

(2) If $(q, (E, 1_\epsilon))$ is in $\delta(p, a, (y \rceil, \epsilon))$, then $\delta(q, a', (\epsilon, \epsilon))$ is undefined for all a' in Σ_1^*.

Notation. For each AFA $\mathscr{D}$, let $\mathscr{L}(\bar{\mathscr{D}}_C) = \{L(\bar{D}) | \bar{D} \text{ in } \bar{\mathscr{D}}_C\}$.

For technical reasons, we next show that the family of languages accepted by $\bar{\mathscr{D}}_C$ is equivalent to the family accepted by $\mathscr{D}_C$.

LEMMA 3.1. *For each* AFA $\mathscr{D}$, $\mathscr{L}(\bar{\mathscr{D}}_C) = \mathscr{L}(\mathscr{D}_C)$.

Proof. Since $\bar{\mathscr{D}}_C \subseteq \mathscr{D}_C$, we have $\mathscr{L}(\bar{\mathscr{D}}_C) \subseteq \mathscr{L}(\mathscr{D}_C)$. To show that $\mathscr{L}(\mathscr{D}_C) \subseteq \mathscr{L}(\bar{\mathscr{D}}_C)$, we prove that for each D in $\mathscr{D}_C$ there exists D' in $\bar{\mathscr{D}}_C$ such that $L(D) = L(D')$. Let $D = (K_1, \Sigma_1, \delta, p_0, F)$ be in $\mathscr{D}_C$ and assume that the moves defined by δ are numbered in some way.

Case 1. Assume that D satisfies property (2) of the definition of $\bar{\mathscr{D}}_C$. Let D' be constructed so that it nondeterministically writes a word, operating as follows.

A. The writing mode: In this mode, D' uses its auxiliary storage in exactly the same way D does, but D' reads no input.

(1) If D writes the symbol y on its checking stack using the ith move of δ, and the move is not a 0-move, then D' writes the symbol (y, i) without reading any input.

(2) If D uses a 0-move, the jth move of δ, then D' writes the symbol $(\bar{y}, j)$ on its checking stack without reading any input.

(3) D' marks its final checking stack symbol in some way.

B. The checking mode:

(1) Without using its auxiliary storage, D' first verifies that the word on its checking stack is a word which D can write on its checking stack. D' also reads the part of the input word which D reads while it is in the writing mode. D' moves as follows.

a. D' moves to the left end of its checking stack word.

b. If D' reads the checking stack symbol (y, i) or $(\bar{y}, i)$, it reads the input symbol which D reads using the ith rule. D' also moves right one symbol on the checking stack.

(2) D', having verified that the checking stack word is correct, now reads the remainder of the input word, simulating the checking moves of D. For these moves, all the second components of the checking stack symbols as well as all the barred symbols are ignored by D'.

It is clear that $L(D') = L(D)$ and that D' reads no input while it is in the writing mode.

Case 2. Assume that D does not satisfy property (2) of the definition of $\bar{\mathscr{D}}_C$. Then D writes the words $w_1, \ldots, w_n$, $n \geqslant 1$, on its checking stack, using and erasing each word in turn. There exists $\bar{D}$ in $\bar{\mathscr{D}}_C$ which writes $x_0 w_1' x_0 \cdots x_0 w_n'$ on its checking stack, using only ϵ-moves. (Assume that x_0 is a new symbol.)

Each w_i' is of the same form as the words in Case 1, that is, for each i, w_i' contains each symbol of w_i as well as the number of the rule which D used to write that symbol. When D erases w_i and writes w_{i+1}, then $\bar{D}$ simulates D by moving right over x_0 and verifying that w_{i+1}' is written on its checking stack. As in Case 1, $\bar{D}$, reading w_{i+1}', then simulates all the checking moves of D on w_{i+1}. When D erases its final stack word w_n and goes to an accepting state, then $\bar{D}$ simulates D by erasing its stack word and going to an accepting state. No further moves are defined for $\bar{D}$. Clearly, $L(\bar{D}) = L(D)$ and the proof of the lemma is complete.

We now establish that the languages which are written on the stacks of acceptors in $\bar{\mathscr{D}}_C$ are related to languages in $\mathscr{L}(\mathscr{D})$.

Notation. For $D = (K_1, \Sigma_1, \delta, q_0, F)$ in $\bar{\mathscr{D}}_C$, let $S(D) = \{y \text{ in } \Gamma_1^* \mid (q_0, \epsilon, (\epsilon, \epsilon)) \overset{*}{\vdash} (p, \epsilon, (y \rceil, \epsilon))\}$ and let $\mathscr{S}(\bar{\mathscr{D}}_C) = \{S(D) \mid D \text{ in } \bar{\mathscr{D}}_C\}$.

LEMMA 3.2. *For each* AFA $\mathscr{D}$, $\mathscr{S}(\bar{\mathscr{D}}_C) = \mathscr{L}(\mathscr{D})$.

Proof. To show $\mathscr{S}(\bar{\mathscr{D}}_C) \subseteq \mathscr{L}(\mathscr{D})$, let D be in $\bar{\mathscr{D}}_C$ and construct D' in $\mathscr{D}$ such that $L(D') = S(D)$. D' operates as follows. Since the storage of D' is the same as the auxiliary storage of D, the reading moves of D' are defined so that D' reads exactly what D writes on its checking stack. Since D can read the last checking stack symbol it has written, D' must remember in its state the last symbol it has read. If D writes its final checking stack symbol, then D' goes to an accepting state.

To show that $\mathscr{L}(\mathscr{D}) \subseteq \mathscr{S}(\bar{\mathscr{D}}_C)$, assume that D is in $\mathscr{D}$. Construct $\bar{D}$ in $\bar{\mathscr{D}}_C$ such that if D reads an input symbol, then $\bar{D}$ writes that symbol on its checking stack, using its auxiliary storage in exactly the same way that D uses its storage. If D reads an input symbol y and goes to an accepting state, then $\bar{D}$ writes $y \rceil$ on its checking stack. Then $S(\bar{D}) = L(D)$ and the proof is complete.

We are now able to establish a characterization result for the family of two-way a-transducer mappings of a full AFL.

THEOREM 3.1. *For each* AFA $\mathscr{D}$, $\mathscr{L}(\mathscr{D}_C) = \mathscr{T}(\mathscr{L}(\mathscr{D}))$.

Proof. Since $\mathscr{L}(\bar{\mathscr{D}}_C) = \mathscr{L}(\mathscr{D}_C)$ and $\mathscr{L}(\mathscr{D}) = \mathscr{S}(\bar{\mathscr{D}}_C)$, we need only prove that $\mathscr{L}(\bar{\mathscr{D}}_C) = \mathscr{T}(\mathscr{S}(\bar{\mathscr{D}}_C))$. To prove that $\mathscr{L}(\bar{\mathscr{D}}_C) \subseteq \mathscr{T}(\mathscr{S}(\bar{\mathscr{D}}_C))$, let $L = L(D)$ where D is in $\bar{\mathscr{D}}_C$. First we construct $\bar{D}$ in $\bar{\mathscr{D}}_C$ which simulates all the moves of D except that when D writes its final stack symbol y and goes to state p, then $\bar{D}$ writes (p, y) as its final stack symbol. Next T in $\mathscr{T}$ is constructed such that for each input word in $S(\bar{D})$, T moves its pointer right to the symbol (p, y), goes to state p, and simulates the checking moves of $\bar{D}$, writing the word which $\bar{D}$ reads. When $\bar{D}$ erases its checking stack word and goes to an accepting state, T moves right to \$ and goes to an accepting state. Then $T(S(\bar{D})) = L(\bar{D}) = L(D)$ and $\mathscr{L}(\bar{\mathscr{D}}_C) \subseteq \mathscr{T}(\mathscr{S}(\bar{\mathscr{D}}_C))$.

To prove that $\mathscr{T}(\mathscr{S}(\bar{\mathscr{D}}_C)) \subseteq \mathscr{L}(\bar{\mathscr{D}}_C)$, let D be in $\bar{\mathscr{D}}_C$ and T be in $\mathscr{T}$. Construct D' in $\bar{\mathscr{D}}_C$ such that if D writes a word w on its checking stack, then D' writes ¢w\$. If D accepts ϵ without writing on its checking stack, then D' writes ¢\$. After writing \$, D' moves its pointer left to ¢ and then simulates the moves of T, reading the word which T writes. When T reaches \$ in an accepting state, D' erases ¢w\$ and goes to an accepting state. Then $L(D') = T(S(D))$ and the proof is complete.

The construction in Theorem 3.1 appears to make $\mathscr{T}(\mathscr{L})$ dependent on the choice of AFA for $\mathscr{L}$. This is not the case, however, as the next result shows.

COROLLARY. *Let $\mathscr{D}$ and $\mathscr{D}'$ be* AFA *such that $\mathscr{L}(\mathscr{D}) = \mathscr{L}(\mathscr{D}')$. Then $\mathscr{L}(\mathscr{D}_C) = \mathscr{L}(\mathscr{D}_C')$*

Proof.
$$\begin{aligned} \mathscr{L}(\mathscr{D}_C) &= \mathscr{T}(\mathscr{L}(\mathscr{D})), \text{ by Theorem 3.1,} \\ &= \mathscr{T}(\mathscr{L}(\mathscr{D}')), \text{ since } \mathscr{L}(\mathscr{D}) = \mathscr{L}(\mathscr{D}'), \\ &= \mathscr{L}(\mathscr{D}_C'), \text{ by Theorem 3.1.} \end{aligned}$$

Remark. Suppose $\mathscr{D}$ is the AFA which does not depend on its storage. Since $\mathscr{D}_C$ does not depend on its auxiliary storage, $\mathscr{L}(\mathscr{D}_C) = \mathscr{L}_{CA}$, where $\mathscr{S}_{CA}$ is the family of checking acceptor languages [11]. By Theorem 3.1, $\mathscr{L}(\mathscr{D}_C) = \mathscr{T}(\mathscr{L}(\mathscr{D}))$ and $\mathscr{T}(\mathscr{L}(\mathscr{D})) = \mathscr{T}(\mathscr{R})$ since $\mathscr{L}(\mathscr{D}) = \mathscr{R}$. So $\mathscr{L}_{CA} = \mathscr{T}(\mathscr{R})$, which is equivalent to the statement proved by Rajlich [13] that $\mathscr{L}_{CA} = \mathscr{L}_{2\text{NFT}}$.

4. PRESERVATION OF PROPERTIES BY $\mathscr{T}$

In this section we study some properties which are preserved by $\mathscr{T}$. We prove in Theorem 4.1 that if $\mathscr{L}$ is a full AFL closed under substitution, then so is $\mathscr{T}(\mathscr{L})$. We show in Theorem 4.2 that if $\mathscr{L}$ is a full principal AFL, then so is $\mathscr{T}(\mathscr{L})$. Finally, we show in Theorem 4.3 that if $\mathscr{L}$ is a full AFL properly contained in the family of extended context-sensitive languages, then so is $\mathscr{T}(\mathscr{L})$. A corollary gives an example of an AFL $\mathscr{L}$, properly contained in the extended context-sensitive languages, such that $\mathscr{T}(\mathscr{L}) = \mathscr{L}$. We now turn to our first result.

THEOREM 4.1. *For each* AFA $\mathscr{D}$, *$\mathscr{L}(\mathscr{D}_C)$ is closed under substitution if $\mathscr{L}(\mathscr{D})$ is.*

Proof. The proof is a variation of the proof in [11] that $\mathscr{L}_{CA}$ is closed under substitution. Assume that D is in $\bar{\mathscr{D}}_C$, $L(D) \subseteq \Sigma_1{}^*$, $S(D) \subseteq \Gamma_1{}^*$, and τ is a substitution on $\Sigma_1{}^*$ defined for each a in Σ_1 by $\tau(a) = aL(D_a)$, where D_a is in $\bar{\mathscr{D}}_C$. (It suffices to consider substitutions of this type since $\mathscr{L}(\mathscr{D}_C)$ is a full AFL.) Then let

$L = \bigcup_{a \text{ in } \Sigma_1} aS(D_a)$ and let σ be the substitution on $\Gamma_1{}^*$ defined for each b in Γ_1 by $\sigma(b) = bL^*$. (This places strings of checking stack words for the D_a between symbols of Γ_1.) Since $\mathscr{L}(\mathscr{D})$ is closed under substitution and the $S(D_a)$ are in $\mathscr{L}(\mathscr{D})$, $\sigma(S(D))$ is in $\mathscr{L}(\mathscr{D})$. Then D' in $\bar{\mathscr{D}}_C$ is constructed such that it writes words in $\sigma(S(D))$ on its checking stack. Then D' simulates D while reading symbols of Σ_1 and, without losing its place on the simulated checking stack of D, simulates D_a while reading words of $L(D_a)$. Hence $L(D') = \tau(L(D))$ and $\mathscr{L}(\mathscr{D}_C)$ is closed under substitution.

COROLLARY. *If $\mathscr{L}$ is a substitution closed full* AFL, *then so is $\mathscr{T}(\mathscr{L})$.*

Proof. If $\mathscr{L}$ is a full AFL, then $\mathscr{L} = \mathscr{L}(\mathscr{D})$ for some AFA $\mathscr{D}$. Then $\mathscr{T}(\mathscr{L}) = \mathscr{T}(\mathscr{L}(\mathscr{D}))$ and by Theorem 3.1, $\mathscr{T}(\mathscr{L}(\mathscr{D})) = \mathscr{L}(\mathscr{D}_C)$. Since $\mathscr{L} = \mathscr{L}(\mathscr{D})$ is a substitution closed full AFL, so is $\mathscr{L}(\mathscr{D}_C)$, by Theorem 4.1. Hence $\mathscr{T}(\mathscr{L}) = \mathscr{L}(\mathscr{D}_C)$ is a substitution closed full AFL.

One question often raised about a full AFL is whether or not it is full principal. We now show that $\mathscr{T}$ preserves full principal AFL.

DEFINITION. An AFL $\mathscr{L}$ is "full principal" if there exists a language L, called a "full generator," such that $\mathscr{L} = \hat{\mathscr{F}}(L)$.[7]

THEOREM 4.2. *Let $\mathscr{D}$ be an* AFA *such that $\mathscr{L}(\mathscr{D})$ is a full principal* AFL *over a countable alphabet. Then $\mathscr{L}(\mathscr{D}_C)$ is full principal.*

Proof. Let $(\Omega, \mathscr{D})$ be the AFA, where $\Omega = (K, \Sigma, \Gamma_2, I_2, f_2, g_2)$. By Lemma 2.1 of [8] and the Corollary to Theorem 3.1, we may assume that $\mathscr{D}$ is finitely encoded. Thus I_2 and $g_2(\Gamma_2{}^*)$ are both finite. Since Σ is countable, $\Sigma = \{a_1, \ldots, a_i, \ldots\}$. For each $n \geqslant 1$, let $\Delta_n = \{a_1, \ldots, a_n\}$. Let $(\Omega_C, \mathscr{D}_C)$ be the AFA of auxiliary $\mathscr{D}$-storage checking acceptors where $\Omega_C = (K, \Sigma, \Gamma_1 \times \Gamma_2, I_1 \times I_2, f, g)$. Let $(\Omega', \mathscr{D}')$ be the structured-storage AFA such that $\Omega' = (K, \Sigma, \Delta_2 \times \Gamma_2, I_1' \times I_2, f', g')$ where $I_1' = \Delta_2 \cup \Delta_2 \rceil \cup \{-, 0, +, E\}$ and f' and g' are defined just as f and g, respectively. (This structured-storage AFA is equivalent to an AFA by Theorem 1 of [3].) For each $D = (K_1, \Sigma_1, \delta, p_0, F)$ in $\mathscr{D}_C$, the checking stack alphabet is in Δ_n for some $n \geqslant 1$, since it is finite. So for each D in $\mathscr{D}_C$ there exists $D' = (K_1', \Sigma_1, \delta', p_0, F')$ in $\mathscr{D}'$ such that the checking stack alphabet is in Δ_2 and $L(D') = L(D)$. (This is done by allowing D' to use $a_1{}^i a_2$ whenever D uses a_i on its checking stack.) Thus $\mathscr{L}(\mathscr{D}_C) \subseteq \mathscr{L}(\mathscr{D}')$. Since $\mathscr{D}' \subseteq \mathscr{D}_C$, $\mathscr{L}(\mathscr{D}') \subseteq \mathscr{L}(\mathscr{D})$ and therefore $\mathscr{L}(\mathscr{D}_C) = \mathscr{L}(\mathscr{D}')$. Since $I_1' \times I_2$ and $g'(\Delta_2{}^* \times \Gamma_2{}^*) = \{\epsilon, a_1, a_2, a_1 \rceil, a_2 \rceil\} \times g_2(\Gamma_2{}^*)$ are both finite, the AFA equivalent to $\mathscr{D}'$, as constructed in Theorem 1 of [3], is finitely encoded. Thus $\mathscr{L}(\mathscr{D}_C)$ is full principal, since $\mathscr{L}(\mathscr{D}_C) = \mathscr{L}(\mathscr{D}')$ and the AFA equivalent to $\mathscr{D}'$ is finitely encoded.

If an AFA $\mathscr{D}$ satisfies certain conditions, then $\bar{L}_{\mathscr{D}}$, the set of those words over the

[7] We write $\hat{\mathscr{F}}(L)$ rather than the more formal $\hat{\mathscr{F}}(\{L\})$.

instruction alphabet of $\mathscr{D}$ which transform the empty storage back to the empty storage, is a full generator of $\mathscr{L}(\mathscr{D})$. By a construction similar to those in [7], an AFA $\mathscr{D}_C'$ can be constructed such that $\mathscr{L}(\mathscr{D}_C') = \mathscr{L}(\mathscr{D}_C)$ and $\bar{L}_{\mathscr{D}_C'}$ is a full generator for $\mathscr{L}(\mathscr{D}_C') = \mathscr{T}(\mathscr{L}(\mathscr{D}))$. We omit the details.

When a new family of languages is introduced, the question arises as to its relation to well-known families. The following theorem relates to that question. It shows that the family of two-way a-transducer mappings of a full AFL contained in the family of extended context-sensitive languages is itself contained in the family of extended context-sensitive languages.

Notation. Let $\mathscr{L}_{\mathrm{CS}}$ be the family of context-sensitive languages and let $\mathscr{L}_{\mathrm{ECS}}$ be the family of extended context-sensitive languages, i.e., $\mathscr{L}_{\mathrm{ECS}} = \{L, L \cup \{\epsilon\} \mid L \text{ in } \mathscr{L}_{\mathrm{CS}}\}$.

THEOREM 4.3. *If $\mathscr{L}$ is a full* AFL *such that $\mathscr{L} \subsetneq \mathscr{L}_{\mathrm{ECS}}$, then $\mathscr{T}(\mathscr{L}) \subsetneq \mathscr{L}_{\mathrm{ECS}}$.*

Proof. By the Corollary to Theorem 2.2, $\mathscr{T}(\mathscr{L})$ is a full AFL if $\mathscr{L}$ is. Since $\mathscr{L}_{\mathrm{ECS}}$ is not a full AFL, it suffices to prove that $\mathscr{T}(\mathscr{L}) \subseteq \mathscr{L}_{\mathrm{ECS}}$.

The proof that $\mathscr{T}(\mathscr{L}) \subseteq \mathscr{L}_{\mathrm{ECS}}$ is a variation of the proof by Ehrich and Yau [2] that a two-way sequential transducer mapping of a context-free language is an extended context-sensitive language. First, given T in $\mathscr{T}$ and a language L in $\mathscr{L}$, an a-transducer M (corresponding to the 1st of [2]) and T' in $\mathscr{T}$ (corresponding to the 2st of [2]) are constructed such that $T'(M(L)) = T(L)$ and for each w in $T(L)$ there exists y' in $M(L)$ such that w is in $T'(y')$ and $|y'| \leqslant |w|$. Then an *lba* is constructed which simulates M and T' and which accepts $T'(M(L)) = T(L)$. Thus$T(L)$ is in $\mathscr{L}_{\mathrm{ECS}}$, so that $\mathscr{T}(\mathscr{L}) \subsetneq \mathscr{L}_{\mathrm{ECS}}$.

In [5] it is shown that each AFL containing $\{\epsilon\}$ has a largest full subAFL. The largest full subAFL of $\mathscr{L}_{\mathrm{ECS}}$ has an interesting property.

COROLLARY. *If $\mathscr{L}$ is the largest full sub*AFL *of $\mathscr{L}_{\mathrm{ECS}}$, then $\mathscr{T}(\mathscr{L}) = \mathscr{L}$.*

Proof. By Theorem 4.3, $\mathscr{T}(\mathscr{L}) \subsetneq \mathscr{L}_{\mathrm{ECS}}$ and by the Corollary to Theorem 2.2, $\mathscr{T}(\mathscr{L})$ is a full AFL. Since $\mathscr{L}$ is the largest full subAFL of $\mathscr{L}_{\mathrm{ECS}}$, $\mathscr{T}(\mathscr{L}) \subseteq \mathscr{L}$. Then $\mathscr{T}(\mathscr{L}) = \mathscr{L}$ since $\mathscr{L} \subseteq \mathscr{T}(\mathscr{L})$.

In conclusion, we note that if $\mathscr{L}$ is closed under complementation or intersection, then $\mathscr{T}(\mathscr{L})$ need not be. In [11] Greibach shows that the family of checking acceptor languages, which we have shown to be equivalent to $\mathscr{T}(\mathscr{R})$, is not closed under complementation or intersection, even though $\mathscr{R}$ is.

5. DETERMINISTIC TWO-WAY A-TRANSDUCERS

In this section we duplicate, for deterministic two-way a-transducers, some of the results proved in Sections 3 and 4 for the nondeterministic two-way a-transducers. We first associate with each AFA $\mathscr{D}$ a structured-storage AFA $\mathscr{D}_D$, called the "AFA of auxiliary $\mathscr{D}$-storage counting checking acceptors." By analogy with Theorem 3.1, we show in Theorem 5.1 that the family $\mathscr{L}(\mathscr{D}_D)$ of languages accepted by $\mathscr{D}_D$ is equivalent to the family $\mathscr{T}_d(\mathscr{L}(\mathscr{D}))$ of deterministic two-way a-transducer mappings of languages in $\mathscr{L}(\mathscr{D})$. We indicate in a remark that substitution is preserved by $\mathscr{T}_d$. One area in which $\mathscr{T}$ and $\mathscr{T}_d$ differ is in their effect on one-letter languages. It is known that there are one-letter languages in $\mathscr{T}(\mathscr{R}) - \mathscr{R}$, where $\mathscr{R}$ is the family of regular sets. But, in Theorem 5.2, we show that one-letter languages in $\mathscr{T}_d(\mathscr{L})$ are in $\mathscr{L}$ for each full AFL $\mathscr{L}$.

Given an AFA $\mathscr{D}$, we now informally define the new family $\mathscr{D}_D$ of acceptors. $\mathscr{D}_D$ will be similar to the AFA $\mathscr{D}_C$ in that there are two types of storage, a stack and an auxiliary storage of the same type as $\mathscr{D}$. The auxiliary storage will be used only while an acceptor in $\mathscr{D}_D$ is in the writing mode. While in the writing mode, the acceptor writes symbols in $\Gamma_1 \times \{1\}$ on its stack. Once a word is written on its stack, the acceptor operates in a checking mode. In this mode, each time the pointer moves left or right from a symbol on the stack, the integer appearing as the second component of the stack symbol is increased by one. Thus we have a count of the number of times a stack symbol has been visited. As in the case of $\mathscr{D}_C$, the symbol in Γ_1 may not be changed. The formal definition follows.

Definition. For each AFA $(\Omega, \mathscr{D})$, where $\Omega = (K, \Sigma, \Gamma_2, I_2, f_2, g_2)$, let $\Gamma_1 = \Sigma$, $I_1 = (\Gamma_1 \times \{1\}) \cup (\Gamma_1 \times \{1\})\rceil \cup (\Gamma_1 \times N \times \{-, +\}) \cup \{0, E\}$, with $\rceil, +, -, 0$, and E new symbols and N the set of positive integers. Let f be the function from $((\Gamma_1 \times N)^* \times \Gamma_2^*) \times (I_1 \times I_2)$ into $((\Gamma_1 \times N)^* \times \Gamma_2^*) \cup \{\varnothing\}$ and let g be the function from $(\Gamma_1 \times N)^* \times \Gamma_2^*$ into the finite subsets of $(\Gamma_1 \times N)^* \times \Gamma_2^*$ defined as follows (for x, $\bar{x}$ in $(\Gamma_1 \times N)^*$, x_1, x_2 in Γ_1, γ_2 in Γ_2^*, u_2 in I_2, and i, j in N).

(1) The "writing" mode:

a. $f((x, \gamma_2), ((x_1, 1), u_2)) = (x(x_1, 1), f_2(\gamma_2, u_2))$.

b. $f((x, \gamma_2), (0, u_2)) = (x, f_2(\gamma_2, u_2))$.

c. $f((x, \gamma_2), ((x_1, 1)\rceil, u_2)) = (x(x_1, 1)\rceil, f_2(\gamma_2, u_2))$, where $f_2(\gamma_2, u_2) = \epsilon$.

(2) The "checking" mode:

a. $f((x(x_1, i)(x_2, j) \rceil \bar{x}, \epsilon), ((x_2, j, -), 1_\epsilon)) = (x(x_1, i) \rceil (x_2, j + 1)\bar{x}, \epsilon)$.

b. $f((x(x_1, i) \rceil (x_2, j)\bar{x}, \epsilon), ((x_1, i, +), 1_\epsilon)) = (x(x_1, i + 1)(x_2, j) \rceil \bar{x}, \epsilon)$.

c. $f((x(x_1, i) \rceil \bar{x}, \epsilon), (0, 1_\epsilon)) = (x(x_1, i) \rceil \bar{x}, \epsilon)$.

d. $f((x(x_1, i) \rceil, \epsilon), (E, 1_\epsilon)) = (\epsilon, \epsilon)$.

(3) a. $g((\epsilon, \epsilon)) = \{(\epsilon, \epsilon)\}$.

b. $g((x(x_1, 1), \gamma_2)) = \{((x_1, 1), \gamma) \mid \gamma \text{ in } g_2(\gamma_2)\}$.

c. $g((x(x_1, i) \rceil \bar{x}, \epsilon)) = \{((x_1, i) \rceil, \epsilon)\}$.

Then $(\Omega_D, \mathscr{D}_D)$, where $\Omega_D = (K, \Sigma, (\Gamma_1 \times N) \times \Gamma_2, I_1 \times I_2, f, g)$ is called the AFA of "auxiliary $\mathscr{D}$-storage counting checking acceptors." (We use $\mathscr{D}_D$ when Ω_D is understood.)

Remark. For each $D = (K_1, \Sigma_1, \delta, q_0, F)$ in $\mathscr{D}_D$, G_D is finite and there are a finite number of moves in δ. This implies that the stack pointer can move away from a stack symbol only a finite number of times in accepting a word.

We note here that $\mathscr{D}_D$ is technically not an AFA, but it is a special instance of a structured-storage AFA. Thus $\mathscr{L}(\mathscr{D}_D)$ is a full AFL by Theorem 1 of [3].

As in Section 3, we define a subset of the previously defined AFA. Then we prove that the two families of acceptors are equivalent, in terms of languages accepted.

DEFINITION. $\bar{\mathscr{D}}_D$ is defined to be the family of a auxiliary $\mathscr{D}$-storage counting checking acceptors which

(1) do no reading of input while in the writing mode, and

(2) use the E instruction only once in accepting a word.

LEMMA 5.1. *For each* AFA $\mathscr{D}$, $\mathscr{L}(\bar{\mathscr{D}}_D) = \mathscr{L}(\mathscr{D}_D)$.

Proof. Clearly $\mathscr{L}(\bar{\mathscr{D}}_D) \subseteq \mathscr{L}(\mathscr{D}_D)$. Consider the reverse inclusion. The proof will be similar to the proof of Lemma 3.2. If D is in $\mathscr{D}_D$, then we construct D' in $\bar{\mathscr{D}}_D$ such that D' nondeterministically writes a word in $S(D)$ on its stack along with the numbers of the rules which D would use in writing the same word. Then D' returns to the left endof the wordandchecksthat the stack word it haswrittencould be writtenby$D$. Since each symbol is scanned once in moving left and once in the process described above, the count of each symbol used by D' is two than the count of the corresponding symbol used by D. Hence $\mathscr{L}(\mathscr{D}_D) \subseteq \mathscr{L}(\bar{\mathscr{D}}_D)$.

We now define some terms related to the languages which are written on the checking stacks of acceptors in $\mathscr{D}_D$. After a preliminary lemma, we are able to characterize the family of deterministic two-way a-transducer mappings of languages in $\mathscr{L}(\mathscr{D})$ in terms of $\mathscr{L}(\mathscr{D}_D)$.

Notation. Let $L(D)$ and $S(D)$ be defined for D in $\bar{\mathscr{D}}_D$ exactly as for acceptors in $\bar{\mathscr{D}}_C$. For D in $\bar{\mathscr{D}}_D$, let $S'(D) = h(S(D))$, where h is the homomorphism from $(\Gamma_1 \times \{1\})^*$ to $\Gamma_1{}^*$ defined by $h((x, 1)) = x$ for each x in Γ_1. Let $\mathscr{S}'(\bar{\mathscr{D}}_D) = \{S'(D) | \ D \text{ in } \bar{\mathscr{D}}_D\}$.

LEMMA 5.2. *For each* AFA $\mathscr{D}$, $\mathscr{S}'(\bar{\mathscr{D}}_D) = \mathscr{L}(\mathscr{D})$.

Proof. The writing moves for $\bar{\mathscr{D}}_D$ differ from the writing moves for $\bar{\mathscr{D}}_C$ only in that all symbols on the checking stack are written with 1 as the second component. Therefore $\mathscr{S}'(\bar{\mathscr{D}}_D) = \mathscr{S}(\bar{\mathscr{D}}_C)$. By Lemma 3.2, $\mathscr{S}(\bar{\mathscr{D}}_C) = \mathscr{L}(\mathscr{D})$, so $\mathscr{S}'(\bar{\mathscr{D}}_D) = \mathscr{L}(\mathscr{D})$.

THEOREM 5.1. *For each* AFA $\mathscr{D}$, $\mathscr{T}_d(\mathscr{L}(\mathscr{D})) = \mathscr{L}(\mathscr{D}_D)$.

Proof. By Lemmas 5.1 and 5.2, $\mathscr{L}(\mathscr{D}_D) = \mathscr{L}(\bar{\mathscr{D}}_D)$ and $\mathscr{L}(\mathscr{D}) = \mathscr{S}'(\bar{\mathscr{D}}_D)$. Thus we need only establish that $\mathscr{T}_d(\mathscr{S}'(\bar{\mathscr{D}}_D)) = \mathscr{L}(\bar{\mathscr{D}}_D)$. To prove that $\mathscr{T}_d(\mathscr{S}'(\bar{\mathscr{D}}_D)) \subseteq \mathscr{L}(\bar{\mathscr{D}}_D)$, assume that D is in $\bar{\mathscr{D}}_D$ and that T is in $\mathscr{T}_d$. We then construct D' in $\bar{\mathscr{D}}_D$ such that $T(S'(D)) = L(D')$. D' first simulates D in writing its checking stack word, except that when D writes $(a_1, 1) \cdots (a_k, 1)$, D' writes $(¢, 1)(a_1, 1) \cdots (a_k, 1)(\$, 1)$. Then D' moves its pointer left to $(¢, 1)$ and simulates the moves of T on $¢a_1 \cdots a_k\$$, using the output of T as input. When T reads \$ in an accepting state, then D' reads $(\$, i)$ for some integer i, erases its checking stack word, and goes to an accepting state. Since T is deterministic, each symbol of $¢a_1 \cdots a_k\$$ will be visited fewer than $\#(K_2) + 1$ times, where $\#(K_2)$ is the number of states of T. Thus D', in simulating T, will visit each symbol on its checking stack fewer than $\#(K_2) + 3$ times. So $L(D') = T(S'(D))$ and D' is in $\bar{\mathscr{D}}_D$. Hence $\mathscr{T}_d(\mathscr{S}'(\bar{\mathscr{D}}_D)) \subseteq \mathscr{L}(\bar{\mathscr{D}}_D)$.

To prove that $\mathscr{L}(\bar{\mathscr{D}}_D) \subseteq \mathscr{T}_d(\mathscr{S}'(\bar{\mathscr{D}}_D))$, assume that $D = (K_1, \Sigma_1, \delta, p_0, F_1)$ is in $\bar{\mathscr{D}}_D$ and that $S(D) \subseteq (\Gamma_1 \times \{1\})^*$ is the language D writes on its checking stack. We define T in $\mathscr{T}_d$ and a regular substitution σ such that $L(D) = T(\sigma(S'(D)))$. Since T is deterministic and D is nondeterministic, each word in $\sigma(S'(D))$ must contain a word of $S(D)$ as well as information about *one* path which D may take through that word. This is done as follows.

For each a in Γ_1, let L_a be the set of all words $(p_1, i_1, j_1, k_1, q_1, a, b_1, d_1) \cdots (p_m, i_m, j_m, k_m, q_m, a, b_m, d_m)$, where for each n, $1 \leqslant n \leqslant m$, and for each d_n in $\{-, 0, +\}$, $(q_n, (a, j_n, d_n), 1_\epsilon)$ is in $\delta(p_n, b_n, (a, j_n) \rceil, \epsilon)$. Also, if $d_n = +$, then $i_{n+1} = i_n$, $j_{n+1} = j_n + 1$, and $k_n < k_{n+1}$. If $d_n = 0$, then $i_{n+1} = i_n$, $j_{n+1} = j_n$, and $k_{n+1} = k_n$. If $d_n = -$, then $i_n < i_{n+1}$, $j_{n+1} = j_n + 1$, and $k_{n+1} = k_n$. Then let σ be the regular substitution defined for each a in Γ_1 by $\sigma(a) = aL_a$. For each (p, i, j, k, q, a, b, d) in $\sigma(a)$, j is used to count the number of moves D has made away from the symbol a. The integer i is used to count the number of times D has moved away from the symbol left of a, and k is the number of times D has moved away from the symbol right of a. If D is in state p, reads b as input and (a, j) on its checking stack, and moves right to a' in state q, then T reads $(p, i, j, k, q, a, b, +)$ in state p, writes b, and moves right to the leftmost symbol in $\sigma(a')$ which has $k + 1$ as its third component. (If the first and second components are not q and j, respectively, then T blocks.) Left moves of D are simulated in a similar manner. That is, if D is in state p, reads (a, j) on its checking stack and b as input, and moves left to a'' in state q, then T reads $(p, i, j, k, q, a, b, -)$, writes b, and moves left to the leftmost symbol in $\sigma(a'')$ which has $i + 1$ as its third component. (If q and j are not the first and fourth components, respectively, of that symbol, then T blocks.) If D reads b using a 0-move, then T writes b and moves right one symbol from $(p, i, j, k, q, a, b, 0)$ to (q, i, j, k, a, b', d).

In this manner, T can deterministically follow the same path through a word in $\sigma(S'(D))$ that D follows through a word in $S(D)$, and $L(D) = T(\sigma(S'(D)))$.

COROLLARY 1. *If $\mathscr{L}$ is a full* AFL, *then $\mathscr{T}_d(\mathscr{L})$ is a full* AFL.

Proof. Since $\mathscr{L}$ is a full AFL, there exists an AFA $\mathscr{D}$ such that $\mathscr{L} = \mathscr{L}(\mathscr{D})$. Then $\mathscr{T}_d(\mathscr{L}) = \mathscr{T}_d(\mathscr{L}(\mathscr{D})) = \mathscr{L}(\mathscr{D}_D)$ is a full AFL, since $\mathscr{D}_D$ is equivalent to an AFA.

COROLLARY 2. *If $\mathscr{D}$ and $\mathscr{D}'$ are* AFA *such that $\mathscr{L}(\mathscr{D}) = \mathscr{L}(\mathscr{D}')$, then $\mathscr{L}(\mathscr{D}_D) = \mathscr{L}(\mathscr{D}_D')$.*

Proof. The proof is analogous to the proof of the Corollary to Theorem 3.1.

Remark. The proof of Theorem 4.1 can be modified to show that for each AFA $\mathscr{D}$, if $\mathscr{L}(\mathscr{D})$ is closed under substitution, then so is $\mathscr{L}(\mathscr{D}_D)$.

It is known that there are one-letter languages in $\mathscr{T}(\mathscr{L}) - \mathscr{L}$ for some AFL $\mathscr{L}$. Greibach [11] proves that $L = \{a^m \mid m \text{ is not prime}\}$ is in $\mathscr{L}_{CA}$, the family of checking acceptor languages. By a remark following the Corollary to Theorem 3.1, $\mathscr{T}(\mathscr{R}) = \mathscr{L}_{CA}$. Since L is obviously not in $\mathscr{R}$, L is in $\mathscr{T}(\mathscr{R}) - \mathscr{R}$. In contrast, the following theorem shows that for each full AFL $\mathscr{L}$, all one-letter languages in $\mathscr{T}_d(\mathscr{L})$ are also in $\mathscr{L}$.

THEOREM 5.2. *If L is a language in a full* AFL *$\mathscr{L}$, T is in $\mathscr{T}_d$, and $T(L) \subseteq b^*$, then $T(L)$ is in $\mathscr{L}$.*

Proof. Let L be a language in a full AFL $\mathscr{L}$, $T = (K_1, \Sigma_1, \Sigma_2, H_1, s_0, ¢, \$, F_1)$ in $\mathscr{T}_d$, and $T(L) \subseteq b^*$. To prove $T(L)$ in $\mathscr{L}$, we define a regular substitution σ on $\Sigma_1{}^*$ and an a-transducer M such that $M(\sigma(¢L\$)) = T(L)$. Then $T(L)$ is in $\mathscr{L}$, since $\sigma(¢L\$)$ is in $\mathscr{L}$ and full AFL are closed under a-transducers.

For a in Σ_1, let $L_a = \{(p_1, d, p_2, b, p_3, d') \mid (p_2, a, b, p_3, d') \text{ is in } H_1, p_1 \text{ is in } K_1, \text{ and } d' \text{ is in } \{-, 0, +\}\}$. The third and fourth components of these 6-tuples store the state T is in and the symbol it writes when it reads a. The fifth and sixth components store the next state and the direction T moves from a. The first and second components store the previous state and the direction T came from in moving to a. For a in Σ_1, let R_a be the set of all words $w_1 \cdots w_n$, where $n \geqslant 1$, w_i is in L_a for each i, $1 \leqslant i \leqslant n$, and no state in K_1 appears more than once as a third component in $w_1, \ldots, w_n$. Note that for each a, R_a is finite, since the number of third components is at most $\#(K_1)$. Let σ be the regular substitution on $\Sigma_1{}^*$ defined for each a in Σ_1 by $\sigma(a) = aR_a$.

We now describe the one-way a-transducer M which, in one pass, simulates the passes of T, and writes the same word in b^* that T does. M has, in its states, three "registers," K_c, K_r, and K_0. While reading $\sigma(a)$, K_r stores the states of T which are involved in the right moves T makes from a, K_0 stores the states involved in the 0-moves T makes on a, and K_c is used to check that the moves made to the symbol a have been accounted for. Informally, M operates as follows.

On reading $(p_1, d, p_2, b', p_3, d')$ in $\sigma(a)$, for p_1, p_2, p_3 states of T, b' in $\{b\} \cup \{\epsilon\}$, and d, d' in $\{-, 0, +\}$, M writes b', moves right one symbol and

(1) If d is $+$, then (p_1, l, p_2) is stored in K_r. If d is $-$, then (p_1, r, p_2) is deleted from K_c. If d is 0, then $(p_1, 0, p_2)$ is deleted from K_0.

(2) If d' is $-$, then (p_2, l, p_3) is deleted from K_c. If d' is $+$, then (p_2, r, p_3) is stored in K_r. If d' is 0, then $(p_2, 0, p_3)$ is stored in K_0.

On reading a in Σ_1, K_c and K_0 are checked to verify that they are empty. If not, then M blocks. If they are empty, then all the triples in K_r are moved to K_c and K_r is emptied. Then M moves right one square to the first symbol in $\sigma(a)$. The 6-tuple representing the final visit to \$ is marked so that when M reads it in an accepting state of T, then M goes to an accepting state. Hence, for w in L, M verifies that $\sigma(w)$ contains information about a path of T through w, and M writes $T(w)$.

In [13], Rajlich proves that the family of languages generated by the 2ft is properly contained in the family of languages generated by the 2nft. Using our notation, this means that $\mathscr{T}_d(\mathscr{R}) \subsetneq \mathscr{T}(\mathscr{R})$. We are now able to prove a more general result.

COROLLARY. *If $\mathscr{L}$ is a full sub*AFL *of $\mathscr{L}_{\mathrm{CF}}$, then $\mathscr{T}_d(\mathscr{L}) \subsetneq \mathscr{T}(\mathscr{L})$.*

Proof. Consider the language $L = \{a^{mn} \mid m, n > 1\}$. Since L is a checking acceptor language [11], L is in $\mathscr{T}(\mathscr{R})$, and therefore is in $\mathscr{T}(\mathscr{L})$ for each full AFL $\mathscr{L}$. But L is not in $\mathscr{T}_d(\mathscr{L}_{\mathrm{CF}})$. For suppose it is. Since L is a one-letter language in $\mathscr{T}_d(\mathscr{L}_{\mathrm{CF}})$, L is in $\mathscr{L}_{\mathrm{CF}}$, by Theorem 5.2. Thus L is in $\mathscr{R}$, since each context-free language over one letter is regular [5]. Hence $\{a^p \mid p \text{ prime}\} \cup \{a\}$, the complement of L, is also in $\mathscr{R}$. But this is a contradiction. Therefore L is not in $\mathscr{T}_d(\mathscr{L}_{\mathrm{CF}})$ and so is not in $\mathscr{T}_d(\mathscr{L})$, for each subAFL $\mathscr{L}$ of $\mathscr{L}_{\mathrm{CF}}$. Since L is in $\mathscr{T}(\mathscr{L}) - \mathscr{T}_d(\mathscr{L})$, $\mathscr{T}_d(\mathscr{L}) \subsetneq \mathscr{T}(\mathscr{L})$ for each subAFL $\mathscr{L}$ of $\mathscr{L}_{\mathrm{CF}}$.

6. COMPOSITION OF $\mathscr{T}$ AND $\mathscr{T}_d$

In this section we study $\mathscr{T}_d\mathscr{T}_d(\mathscr{L})$, $\mathscr{T}\mathscr{T}_d(\mathscr{L})$, $\mathscr{T}_d\mathscr{T}(\mathscr{L})$, and $\mathscr{T}\mathscr{T}(\mathscr{L})$, for $\mathscr{L}$ an AFL. We first prove in Theorem 6.1 that for each full AFL $\mathscr{L}$, $\mathscr{T}_d\mathscr{T}_d(\mathscr{L}) = \mathscr{T}_d(\mathscr{L})$ and $\mathscr{T}\mathscr{T}_d(\mathscr{L}) = \mathscr{T}(\mathscr{L})$. The question then arises as to whether $\mathscr{T}_d\mathscr{T}(\mathscr{L}) = \mathscr{T}(\mathscr{L})$ and $\mathscr{T}\mathscr{T}(\mathscr{L}) = \mathscr{T}(\mathscr{L})$. There are cases when this is true. For example, by the Corollary to Theorem 4.3, if $\mathscr{L}$ is the largest full subAFL of the extended context-sensitive languages, then $\mathscr{T}(\mathscr{L}) = \mathscr{L}$. It follows that $\mathscr{T}\mathscr{T}(\mathscr{L}) = \mathscr{T}(\mathscr{L})$ and hence $\mathscr{T}_d\mathscr{T}(\mathscr{L}) = \mathscr{T}(\mathscr{L})$. On the other hand, there are cases when $\mathscr{T}\mathscr{T}(\mathscr{L}) \neq \mathscr{T}(\mathscr{L})$ and $\mathscr{T}_d\mathscr{T}(\mathscr{L}) \neq \mathscr{T}(\mathscr{L})$. In Theorem 6.2 we give necessary and sufficient conditions for $\mathscr{T}\mathscr{T}(\mathscr{L}) = \mathscr{T}(\mathscr{L})$. In the Corollary to Theorem 6.2 we show that there are AFL for which $\mathscr{T}_d\mathscr{T}(\mathscr{L}) \neq \mathscr{T}(\mathscr{L})$ and hence $\mathscr{T}\mathscr{T}(\mathscr{L}) \neq \mathscr{T}(\mathscr{L})$.

THEOREM 6.1. *For each full* AFL $\mathscr{L}$, $\mathscr{T}_d\mathscr{T}_d(\mathscr{L}) = \mathscr{T}_d(\mathscr{L})$ *and* $\mathscr{T}\mathscr{T}_d(\mathscr{L}) = \mathscr{T}(\mathscr{L})$.

Proof. We first prove that for each full AFL $\mathscr{L}$, $\mathscr{T}_d\mathscr{T}_d(\mathscr{L}) = \mathscr{T}_d(\mathscr{L})$. Since $\mathscr{T}_d(\mathscr{L}) \subseteq \mathscr{T}_d\mathscr{T}_d(\mathscr{L})$ for each full AFL $\mathscr{L}$, it suffices to show that $\mathscr{T}_d\mathscr{T}_d(\mathscr{L}) \subseteq \mathscr{T}_d(\mathscr{L})$. Let $L \subseteq \Sigma_1{}^*$ be in $\mathscr{L}$ and let T_1 and T_2 be in $\mathscr{T}_d$. We use σ, the substitution defined in Theorem 5.2, and construct T_3 in $\mathscr{T}_d$ such that $T_3(\sigma(L)) = T_2T_1(L)$.

We now informally describe the moves of T_3. Let $a_1 \cdots a_k$, for $k \geqslant 0$, be a word in L and let $T_1(a_1 \cdots a_k) = b_1 \cdots b_l$, for $l \geqslant 0$. Let $T_2(b_1 \cdots b_l) = c_1 \cdots c_m$, for $m \geqslant 0$, and let $y = ¢w_¢a_1w_1 \cdots a_kw_k\$w_\$$ be in $\sigma(¢a_1 \cdots a_k\$)$. By the definition of σ, a path of T_1 through $a_1 \cdots a_k$ is incorporated in y. The word $b_1 \cdots b_l$ appears as the fourth component of symbols in $\sigma(y)$, but not necessarily in the same order, from left to right. (For example, if T_1 reads a_i, writes b_j, and then moves left to a_{i-1}, then b_{j+1} will appear as the fourth component of a symbol in $\sigma(a_{i-1})$, which is to the left of $\sigma(a_i)$.) So T_3 must simulate the moves of T_2 as follows. Assume T_3 is reading a symbol $(p_1, d, p_2, b_i, p_3, d')$ in $\sigma(a)$. To simulate a left move by T_2 from b_i, T_3 uses the leftmost two components, p_1 and d, to move to the symbol in $\sigma(y)$ which has p_1 as its third component. For all the cases which follow, T_3 also writes the same symbol that T_2 writes while reading b_i. If d is $-$, then T_3 moves left to the only symbol in $\sigma(a_{i-1})$ which has p_1 as its third component. If d is $+$, then T_3 moves right to the only symbol in $\sigma(a_{i+1})$ which has p_1 as its third component. If d is 0, then T_3 moves left one square and that symbol must have p_1 as its third component. If the desired symbol is not found, then T_3 blocks.

To simulate a right move by T_2 from b_i, T_3 writes the same symbol T_2 writes and in a manner analogous to that defined above uses last the two components of $(p_1, d, p_2, b_i, p_3, d')$ to find b_{i+1}. When T_2 uses a 0-move and writes a symbol, then T_3 uses a 0-move and writes the same symbol.

One problem remains. If T_3 is simulating a left move of T_2 as defined above and finds the proper third component, then the fourth component may be ϵ. In this case T_3 must remember, in its state, that it was simulating a left move. On reading ϵ as the fourth component, T_3 continues to use the first two components of symbols it reads until it finds a symbol whose fourth component is not ϵ. If T_3 is simulating a right move, then it uses the last two components until it finds a symbol whose fourth component is not ϵ.

If T_3 reaches \$ in an accepting state of T_2, then it goes to an accepting state. Thus $T_3(¢w_¢a_1w_1 \cdots a_kw_k\$w_\$) = T_2(b_1 \cdots b_l) = T_2T_1(a_1 \cdots a_k)$ and we have $\mathscr{T}_d\mathscr{T}_d(\mathscr{L}) \subseteq \mathscr{T}_d(\mathscr{L})$.

To prove that $\mathscr{T}\mathscr{T}_d(\mathscr{L}) = \mathscr{T}(\mathscr{L})$, we use the same proof, noting that T_3 is nondeterministic if T_2 is.

For each AFA $\mathscr{D}$, we have defined the structured-storage AFA $\mathscr{D}_C$ and $\mathscr{D}_D$. The question arises as to the connection between the two AFA. The following corollary expresses one such connection.

COROLLARY 1. *For each* AFA $\mathscr{D}$, $\mathscr{T}(\mathscr{L}(\mathscr{D}_D)) = \mathscr{L}(\mathscr{D}_C)$.

Proof. $\mathscr{T}(\mathscr{L}(\mathscr{D}_D)) = \mathscr{T}(\mathscr{T}_d(\mathscr{L}(\mathscr{D})))$, by Theorem 5.1,

$= \mathscr{T}(\mathscr{L}(\mathscr{D}))$, by Theorem 6.1,

$= \mathscr{L}(\mathscr{D}_C)$, by Theorem 3.1.

We also have a corollary which is analogous to Theorem 2.1 and the Corollary to Theorem 2.2.

COROLLARY 2. *If $\mathscr{L}$ is an* AFL, *then $\mathscr{T}_d(\mathscr{L})$ is a full* AFL *closed under reversal.*

Proof. Assume $\mathscr{L}$ is an AFL. By Theorem 2.3, $\mathscr{T}_d(\mathscr{L}) = \mathscr{T}_d(\hat{\mathscr{F}}(\mathscr{L}))$. Therefore $\mathscr{T}_d(\mathscr{L})$ is a full AFL, since $\mathscr{T}_d(\hat{\mathscr{F}}(\mathscr{L}))$ is a full AFL by Corollary 1 of Theorem 5.1.

We now show that $\mathscr{T}_d(\mathscr{L})$ is closed under reversal. There exists T in $\mathscr{T}_d$ such that for all languages L, $T(L) = L^R$. Then for L in $\mathscr{T}_d(\mathscr{L})$, $T(L) = L^R$ is in $\mathscr{T}_d\mathscr{T}_d(\mathscr{L})$. Thus L^R is in $\mathscr{T}_d(\mathscr{L})$, since $\mathscr{T}_d\mathscr{T}_d(\mathscr{L}) = \mathscr{T}_d(\mathscr{L})$ by Theorem 6.1, and the proof of the corollary is complete.

We now consider the case where $\mathscr{L}$ is an arbitrary AFL.

COROLLARY 3. *For each* AFL $\mathscr{L}$, $\mathscr{T}_d\mathscr{T}_d(\mathscr{L}) = \mathscr{T}_d(\mathscr{L})$ *and* $\mathscr{T}\mathscr{T}_d(\mathscr{L}) = \mathscr{T}(\mathscr{L})$.

Proof. $\mathscr{T}_d(\mathscr{L}) = \mathscr{T}_d(\hat{\mathscr{F}}(\mathscr{L}))$, by Theorem 2.3,

$= \mathscr{T}_d\mathscr{T}_d(\hat{\mathscr{F}}(\mathscr{L}))$, by Theorem 6.1,

$= \mathscr{T}_d\mathscr{T}_d(\mathscr{L})$, by Theorem 2.3.

The proof that $\mathscr{T}\mathscr{T}_d(\mathscr{L}) = \mathscr{T}(\mathscr{L})$ is analogous.

We now prove a lemma which will enable us to show that there exist families $\mathscr{L}$ for which $\mathscr{T}_d\mathscr{T}(\mathscr{L})$ is not equal to $\mathscr{T}(\mathscr{L})$.

LEMMA 6.1. *Let $\mathscr{L}$ be a full* AFL, *$L \subseteq \Sigma_2^*$ a language in $\mathscr{L}$, c a symbol not in Σ_2, and $L_k = \{(wc)^k \mid w \text{ in } L\}$ for each $k > 1$. Then the following statements are equivalent.*

(a) *L is in $\mathscr{T}_d(\mathscr{L})$,*

(b) *L_k is in $\mathscr{T}(\mathscr{L})$ for each $k > 1$,*

(c) *L_k is in $\mathscr{T}(\mathscr{L})$ for some $k > 1$, and*

(d) *L_k is in $\mathscr{T}_d(\mathscr{L})$ for some $k > 1$.*

Proof. Obviously (b) implies (c). It thus suffices to prove that (a) implies (b), (c) implies (d), and (d) implies (a). To prove that (a) implies (b), assume that L is in $\mathscr{T}_d(\mathscr{L})$. Then $L = T_d(L')$ for some L' in $\mathscr{L}$ and T_d in $\mathscr{T}_d$. Then for each k, there exists T_k in $\mathscr{T}$ such that T_k simulates T_d exactly k times. Then $T_k(L') = L_k$ and so L_k is in $\mathscr{T}(\mathscr{L})$.

To prove that (c) implies (d), assume that $L_k = T(L'')$ is in $\mathscr{T}(\mathscr{L})$ for some $k > 1$.

Since $\mathscr{T}(\mathscr{L}) = \mathscr{T}(\mathscr{L}(\mathscr{D})) = \mathscr{L}(\mathscr{D}_C)$ for some AFA $\mathscr{D}$, $L_k = L(D)$ for some $D = (K_1, \Sigma_1, \delta, p_0, F)$ in $\mathscr{D}_C$. By the construction in Lemma 3.2, we may assume that only words in L'' are written on the checking stack of D. We may also assume that, in accepting a word, D writes at least one symbol on its checking stack. (If it does not, then there exists D' in $\mathscr{D}_C$ such that $L(D) = L(D')$ and D' writes at least one word on its checking stack.) We first show that

$$(*)$$

for each word w in L, there is a word y in L'' such that if $(wc)^k$ is accepted by D with y on its checking stack, then there is an accepting computation in which D visits each symbol of y at most $\#(K_1)$ times.

Suppose $(*)$ is false. Then there is a word w in L such that for all words $y = y_1 \cdots y_m$ in L'', whenever $(wc)^k$ is accepted by D with y on its checking stack, then for some j, $1 \leqslant j \leqslant m$, y_j is visited more than $\#(K_1)$ times. Then there exists q in K_1 such that the shortest path in accepting $(wc)^k$ is $(p_0, (wc)^k, (\epsilon, \epsilon)) \overset{*}{\vdash} (p, (wc)^k, (y_1 \cdots y_m \uparrow, \epsilon)) \overset{*}{\vdash} (q, w'c(wc)^n, (y_1 \cdots y_j \uparrow \cdots y_m, \epsilon)) \overset{+}{\vdash} (q, w''c(wc)^{n'}, y_1 \cdots y_j \uparrow \cdots y_m, \epsilon)) \overset{*}{\vdash} (q_F, \epsilon, (\epsilon, \epsilon))$ for q_F in F, $1 \leqslant j \leqslant m$, $0 \leqslant n' \leqslant n \leqslant k$, and w' and w'' terminal subwords of w. Clearly, w' must equal w''. (For if $w' \neq w''$, then D also accepts words of the form $(wc)^{k-n-1}\bar{w}'w''(wc)^{n'}$, where $w = \bar{w}'w'$, a contradiction.) Also, n must equal n'. (For if $n \neq n'$, then D also accepts words of the form $(wc)^{k-(n-n')}$, a contradiction.) Then there is a shorter path for accepting $(wc)^k$, namely, $(p_0, (wc)^k, (\epsilon, \epsilon)) \overset{*}{\vdash} (p, (wc)^k, (y_1 \cdots y_m \uparrow, \epsilon)) \overset{*}{\vdash} (q, w'c(wc)^n, (y_1 \cdots y_j \uparrow \cdots y_m, \epsilon)) = (q, w''c(wc)^{n'}, (y_1 \cdots y_j \uparrow \cdots y_m, \epsilon)) \overset{*}{\vdash} (q_F, \epsilon, (\epsilon, \epsilon))$. This is a contradiction. Thus $(*)$ is true.

Now we show that L_k is in $\mathscr{L}(\mathscr{D}_D)$. We construct D' in $\mathscr{D}_D$ so that $(y_1, 1) \cdots (y_m, 1)$ is written on the checking stack of D' whenever $y_1 \cdots y_m$ is written on the checking stack of D. Also, we construct D' so that the second component of a checking stack symbol never exceeds $\#(K_1)$ and D' simulates the moves of D. By $(*)$, $L(D') = L_k$, so that L_k is in $\mathscr{T}_d(\mathscr{L}) = \mathscr{L}(\mathscr{D}_D)$. So (c) implies (d).

Finally, we prove that (d) implies (a). Given L_k in $\mathscr{T}_d(\mathscr{L})$, we construct an a-transducer which copies the first subword w, and erases $c(wc)^{k-1}$. Therefore L is in $\mathscr{T}_d(\mathscr{L})$, since $\mathscr{T}_d(\mathscr{L})$ is a full AFL and is thus closed under a-transducers.

Using the previous lemma, we give necessary and sufficient conditions for $\mathscr{T}_d\mathscr{T}(\mathscr{L}) = \mathscr{T}(\mathscr{L})$ and $\mathscr{T}\mathscr{T}(\mathscr{L}) = \mathscr{T}(\mathscr{L})$.

THEOREM 6.2. *For each* AFL $\mathscr{L}$, *the following statements are equivalent.*

(a) $\mathscr{T}(\mathscr{L}) = \mathscr{T}_d(\mathscr{L})$,

(b) $\mathscr{T}\mathscr{T}(\mathscr{L}) = \mathscr{T}(\mathscr{L})$, and

(c) $\mathscr{T}_d\mathscr{T}(\mathscr{L}) = \mathscr{T}(\mathscr{L})$.

Proof. Since $\mathscr{T}(\mathscr{L}) = \mathscr{T}(\hat{\mathscr{F}}(\mathscr{L}))$ and $\mathscr{T}_d(\mathscr{L}) = \mathscr{T}_d(\hat{\mathscr{F}}(\mathscr{L}))$ by Theorem 2.3, we may assume that $\mathscr{L}$ is a full AFL. We first prove that (a) implies (b). If $\mathscr{T}(\mathscr{L}) = \mathscr{T}_d(\mathscr{L})$, then $\mathscr{T}\mathscr{T}(\mathscr{L}) = \mathscr{T}\mathscr{T}_d(\mathscr{L})$. Therefore $\mathscr{T}\mathscr{T}(\mathscr{L}) = \mathscr{T}(\mathscr{L})$, since $\mathscr{T}\mathscr{T}_d(\mathscr{L}) = \mathscr{T}(\mathscr{L})$, by Theorem 6.1.

We now prove that (b) implies (c). If $\mathscr{T}\mathscr{T}(\mathscr{L}) = \mathscr{T}(\mathscr{L})$, then $\mathscr{T}_d\mathscr{T}(\mathscr{L}) = \mathscr{T}(\mathscr{L})$, since $\mathscr{T}(\mathscr{L}) \subseteq \mathscr{T}_d\mathscr{T}(\mathscr{L}) \subseteq \mathscr{T}\mathscr{T}(\mathscr{L})$.

Finally, we prove that (c) implies (a). We assume that $\mathscr{T}(\mathscr{L}) \neq \mathscr{T}_d(\mathscr{L})$ and prove that $\mathscr{T}_d\mathscr{T}(\mathscr{L}) \neq \mathscr{T}(\mathscr{L})$. Since $\mathscr{T}(\mathscr{L}) \neq \mathscr{T}_d(\mathscr{L})$, there exists a language L in $\mathscr{T}(\mathscr{L}) - \mathscr{T}_d(\mathscr{L})$. Let $L \subseteq \Sigma_2{}^*$ and c be a symbol not in Σ_2. Then $L_2 = \{(wc)^2 \mid w \text{ in } L\}$ is not in $\mathscr{T}(\mathscr{L})$, by Lemma 6.1. But L_2 is in $\mathscr{T}_d\mathscr{T}(\mathscr{L})$, since L is in $\mathscr{T}(\mathscr{L})$ and there exists T in $\mathscr{T}_d$ such that $T(w) = (wc)^2$ for all w in $\Sigma_2{}^*$. Hence L_2 is in $\mathscr{T}_d\mathscr{T}(\mathscr{L}) - \mathscr{T}(\mathscr{L})$, so that $\mathscr{T}_d\mathscr{T}(\mathscr{L}) \neq \mathscr{T}(\mathscr{L})$.

We are now able to give some examples of AFL for which $\mathscr{T}_d\mathscr{T}(\mathscr{L}) \neq \mathscr{T}(\mathscr{L})$.

COROLLARY. *If $\mathscr{L}$ is a sub*AFL *of $\mathscr{L}_{\mathrm{CF}}$, then $\mathscr{T}_d\mathscr{T}(\mathscr{L}) \neq \mathscr{T}(\mathscr{L})$ and $\mathscr{T}\mathscr{T}(\mathscr{L}) \neq \mathscr{T}(\mathscr{L})$.*

Proof. By the Corollary to Theorem 2.1 and Corollary 2 of Theorem 6.1, it suffices to prove the corollary for $\mathscr{L}$ a full subAFL of $\mathscr{L}_{\mathrm{CF}}$. Since $\mathscr{T}_d \subseteq \mathscr{T}$, if $\mathscr{L}$ is a full subAFL of $\mathscr{L}_{\mathrm{CF}}$, then $\mathscr{T}_d(\mathscr{L}) \subsetneq \mathscr{T}(\mathscr{L})$. Then, by Theorem 6.2, we have $\mathscr{T}_d\mathscr{T}(\mathscr{L}) \neq \mathscr{T}(\mathscr{L})$. Finally, we have $\mathscr{T}\mathscr{T}(\mathscr{L}) \neq \mathscr{T}(\mathscr{L})$, since $\mathscr{T}(\mathscr{L}) \subseteq \mathscr{T}_d\mathscr{T}(\mathscr{L}) \subseteq \mathscr{T}\mathscr{T}(\mathscr{L})$.

REFERENCES

1. A. V. AHO AND J. D. ULLMAN, A characterization of two-way deterministic classes of languages, *J. Comput. System Sci.* **4** (1970), 523–538.
2. R. W. EHRICH AND S. S. YAU, Two-way sequential transducers and stack automata, *Information and Control* **18** (1971), 404–446.
3. A. GABRIELIAN AND S. GINSBURG, Structured-storage AFA, *IEEE Trans. Computers* **C-22** (1973), 534–537.
4. S. GINSBURG, "The Mathematical Theory of Context-Free Languages," McGraw–Hill, New York, 1966.
5. S. GINSBURG AND J. GOLDSTINE, On the largest full subAFL of an AFL, *Math. Systems Theory* **6** (1972), 241–242.
6. S. GINSBURG AND S. GREIBACH, Abstract families of languages, *in* "Studies in Abstract Families of Languages," *Mem. Amer. Math. Soc.* **87** (1969), 1–32.
7. S. GINSBURG AND S. GREIBACH, On AFL generators for finitely encoded AFA, *J. Comput. System Sci.* **7** (1973), 1–27.
8. S. GINSBURG AND S. GREIBACH, Principal AFL, *J. Comput. System Sci.* **4** (1970), 308–338.
9. S. GINSBURG AND E. H. SPANIER, Control sets on grammars, *Math. Systems Theory* **2** (1968), 159–177.
10. S. GINSBURG AND E. H. SPANIER, Substitution in families of languages, *Information Sci.* **2** (1960), 83–110.

11. S. GREIBACH, Checking automata and one-way stack languages, *J. Comput. System Sci.* **3** (1969), 196–217.
12. S. GREIBACH AND J. HOPCROFT, Independence of AFL operations, *in* "Studies in Abstract Families of Languages," *Mem. Amer. Math. Soc.* **87** (1969), 33–40.
13. V. RAJLICH, Absolutely parallel grammars and two-way finite-state transducers, *J. Comput. System Sci.* **6** (1972), 324–342.