INFORMATION AND CONTROL 16, 7–35 (1970)

Full AFLs and Nested Iterated Substitution*

SHEILA A. GREIBACH

University of California, Los Angeles, California 90024†

A superAFL is a family of languages closed under union with unitary sets, intersection with regular sets, and nested iterated substitution and containing at least one nonunitary set. Every superAFL is a full AFL containing all context-free languages. If $\mathscr{L}$ is a full principal AFL, then $\hat{\mathscr{S}}^{\infty}(\mathscr{L})$, the least superAFL containing $\mathscr{L}$, is full principal. If $\mathscr{L}$ is not substitution closed, the substitution closure of $\mathscr{L}$ is properly contained in $\hat{\mathscr{S}}^{\infty}(\mathscr{L})$. The indexed languages form a superAFL which is not the least superAFL containing the one-way stack languages. If $\mathscr{L}$ has a decidable emptiness problem, so does $\hat{\mathscr{S}}^{\infty}(\mathscr{L})$. If $\mathscr{D}^s$ is an AFA, $\mathscr{L} = \mathscr{L}(\mathscr{D}^s)$ and $\mathscr{D}^w$ is the family of machines whose data structure is a pushdown store of tapes of $\mathscr{D}^s$, then $\mathscr{L}(\mathscr{D}^w) = \hat{\mathscr{S}}^{\infty}(\mathscr{L})$ if, and only if, $\mathscr{D}^s$ is nontrivial. If $\mathscr{D}^s$ is uniformly erasable and $\mathscr{L}(\mathscr{D}^s)$ has a decidable emptiness problem, then it is decidable if a member of $\mathscr{D}^w$ is finitely nested.

1. INTRODUCTION

Recently there have been several investigations of the closure under substitution of various families of languages [12], [13], [19], [21], [22], and of the relation of substitution to AFLs in general [15], [10]. A machine realization of the substitution closure of a full AFL $\mathscr{L}$ was obtained by finitely nesting tapes of any machine representation for $\mathscr{L}$ [15]. It was shown that if any full AFL $\mathscr{L}$ is not closed under substitution, then $\mathscr{L}\hat{\sigma}\mathscr{L}$, the result of substituting members of $\mathscr{L}$ into $\mathscr{L}$, is not substitution closed and hence $\mathscr{L}$ generates an infinite hierarchy of full AFLs and the substitution closure of $\mathscr{L}$ is not full principal [14]. In this paper we define a generalization of the

* This research has been supported in part by Air Force Cambridge Research Laboratory, Office of Aerospace Research, USAF, under Contract F19628-68-C-0029, and the Division of Engineering and Applied Physics of Harvard University.

† This research was completed while the author was at Harvard University, Cambridge, Massachusetts 02138.

substitution operator, called nested iterated substitution, examine the properties of families of languages closed under nested iterated substitution, establish the appropriate machine characterization and discuss related decision problems.

If τ is a substitution such that $a \in \tau(a)$ for all a, then we let $\tau^1(L) = \tau(L)$, and $\tau^{k+1}(L) = \tau(\tau^k(L))$ for $k \geqslant 1$, and $\tau^\infty(L) = \bigcup_n \tau^n(L)$. We call τ^∞ a nested iterated substitution. Král showed that the context-free languages are closed under nested iterated substitution [18]. We define a superAFL as a family of languages closed under union with unitary sets (sets whose members have length at most one), intersection with regular sets and nested iterated substitution and containing at least one nonunitary set, and show in Section 2 that every superAFL is a full AFL containing all context-free languages. In Section 4 we show that if $\mathscr{L}$ is a full principal AFL, so is $\hat{\mathscr{S}}^\infty(\mathscr{L})$, the least superAFL containing $\mathscr{L}$; if a full AFL $\mathscr{L}$ is not substitution closed, then the substitution closure of $\mathscr{L}$ is properly contained in $\hat{\mathscr{S}}^\infty(\mathscr{L})$.

If $\mathscr{D}^s$ is a family of one-way nondeterministic acceptors (an AFA [6]), we define $\mathscr{D}^w$, a well-nested AFA, as the family of all one-way nondeterministic acceptors having as data structure a pushdown store of tapes which are themselves data structures for members of $\mathscr{D}^s$. That is, members of $\mathscr{D}^w$ have nests, each of which can be viewed as the single tape of a member of $\mathscr{D}^s$, and can in each step either compute in the top (i.e., last opened) nest, push open a new nest which opens in the initial configuration (taken as the null string) or pop closed a nest which contains only the null string and so is back in the initial configuration. We define $\mathscr{D}$, the corresponding nested AFA, as possessing in addition the power to duplicate the top nest. For any family $\mathscr{M}$ of acceptors we let $\mathscr{L}(\mathscr{M})$ be the corresponding family of languages they accept. An AFA is nontrivial if its tape can ever contain a nonempty string and later erase that string. Aho and Ullman have shown that the indexed languages are superAFL and that they are defined by nested pdas with duplicate instructions [3]. We generalize this by showing that every nontrivial nested AFA and every nontrivial well-nested AFA defines a superAFL. In Sections 3 and 4 we show that $\mathscr{L}(\mathscr{D}^w) = \hat{\mathscr{S}}^\infty(\mathscr{L}(\mathscr{D}^s))$, if and only if $\mathscr{D}^s$ is nontrivial. Thus, well-nested AFAs are the machine realization for superAFLs precisely as finitely nested AFAs are the machine realization for substitution closed full AFLs, and AFAs characterize full AFLs [6], [15].

In Section 5 we show that if the emptiness problem is decidable for a full AFL $\mathscr{L}$, it is decidable for $\hat{\mathscr{S}}^\infty(\mathscr{L})$. If the emptiness problem is decidable for $\mathscr{L}(\mathscr{D}^s)$ and $\mathscr{D}^s$ satisfies a condition we call uniform erasability, then it is decidable whether a member of $\mathscr{D}^w$ is finitely nested, namely, whether the number of nests opened is bounded for all inputs. If $\mathscr{D}^s$ does not meet this condition, then it may be undecidable whether a member of $\mathscr{D}^w$ is finitely nested, even if $\mathscr{L}(\mathscr{D}^s)$ contains only regular sets.

This raises the whole question of representations and AFA-invariant operations, which is briefly examined in Section 6. The operation of nested iterated substitution appears to be "invariant" since if $\mathscr{L} = \mathscr{L}(\mathscr{D}^s)$, that is, if $\mathscr{D}^s$ is any nontrivial machine representation of $\mathscr{L}$, then $\mathscr{L}(\mathscr{D}^w)$ is the least full AFL containing $\mathscr{L}$ and closed under nested iterated substitution. On the other hand, we can find machine representations $\mathscr{D}^s$ and $\bar{\mathscr{D}}^s$ for the context-free languages ($\mathscr{L}(\mathscr{D}^s)$ = context-free languages) such that the corresponding nested AFA $\mathscr{D}$ yields the indexed languages but the nested AFA $\bar{\mathscr{D}}$ yields all recursively enumerable sets. Thus, whatever operations yield the indexed languages from the context-free languages do not appear to be AFA-invariant. It seems to be an interesting open problem to formulate a precise definition of AFA and AFL invariant operations or properties; the problem appears closely tied to effective closure of families of languages under various operations.

2. SuperAFLs

In this section we shall formally define iterated substitutions and nested iterated substitutions. We shall define superAFLs as families closed under nested iterated substitution and see that they are indeed full AFLs closed under substitution. In the next sections we shall see that the closure of a full principal AFL under nested iterated substitution is indeed a full principal AFL, so that in particular if a full principal AFL is not substitution closed, its substitution closure is properly contained in its closure under nested iterated substitution.

Definition 2.1. Let $L \subseteq \Sigma_1$ and for each a in Σ_1, let Σ_a be finite and $L_a \subseteq \Sigma_a{}^*$. Let $\tau(a) = L_a$ for each a in Σ_1. Let $\tau(xy) = \tau(x)\,\tau(y)$ and $\tau(L) = \bigcup_{w \in L} \tau(w)$. Then τ is a *substitution.* If $\mathscr{L}$ contains $\tau(L)$ whenever $L \in \mathscr{L}$ and $\tau(a) \in \mathscr{L}$ for each a, then $\mathscr{L}$ is *substitution closed.*

Definition 2.2. Let $L \subseteq \Sigma_1$, and let τ be a substitution, with $\tau(a) \subseteq \Sigma_a{}^*$ for each $a \in \Sigma_1$. Extend τ to $\Sigma_1 \cup (\bigcup_a \Sigma_a)$ by defining $\tau(b) = \{b\}$ for $b \in (\bigcup_a \Sigma_a) - \Sigma_1$. Let $\tau^1(L) = \tau(L)$ and $\tau^{n+1}(L) = \tau(\tau^n(L))$ for $n \geqslant 1$. Let $\tau^\infty(L) = \bigcup_n \tau^n(L)$. Then τ^∞ is an iterated *substitution.* It is a *nested iterated substitution* if $a \in \tau(a)$ for all a in $\Sigma_1 \cup (\bigcup_a \Sigma_a)$. *$\mathscr{L}$ is closed under nested iterated substitution* if $\tau^\infty(L)$ is in $\mathscr{L}$ whenever (a) $L \in \mathscr{L}$, (b) τ^∞ is a nested iterated substitution, and (c) $\tau(\mathrm{a}) \in \mathscr{L}$ for all $a \in \Sigma_1$.

DEFINITION 2.3. A *full* AFL is a family of languages containing at least one nonempty set and closed under homomorphism, inverse homomorphism, intersection with regular sets, union, concatenation and Kleene closure. For any family of languages $\mathscr{L}$, let $\hat{\mathscr{F}}(\mathscr{L})$ be the least full AFL containing $\mathscr{L}$. If $\mathscr{L} = \{L\}$, we write $\hat{\mathscr{F}}(L)$ instead of $\hat{\mathscr{F}}(\{L\})$; $\hat{\mathscr{F}}(L)$ is *full principal*, and L is a *generator* of $\hat{\mathscr{F}}(L)$.

DEFINITION 2.4. If $\mathscr{L}$ is a family of languages, $\hat{\mathscr{S}}(\mathscr{L})$ is the least substitution closed full AFL containing $\mathscr{L}$, and $\hat{\mathscr{S}}^{\infty}(\mathscr{L})$ is the least full AFL containing $\mathscr{L}$ and closed under nested iterated substitution.

We need one more definition in order to define superAFLs.

DEFINITION 2.5. A set L is *unitary* if $|w| \leqslant 1$ for all w in L.[1]

DEFINITION 2.6. A family of languages $\mathscr{L}$ is a *super*AFL if it contains some nonunitary set and is closed under intersection with regular sets, union with unitary sets and nested iterated substitution.

Our goal in this section is to show that every superAFL is a substitution closed full AFL containing all context-free languages. Now since a superAFL is nonempty and closed under intersection with regular sets it must contain the empty set. Since it is closed under union with unitary sets, it must contain all unitary sets.

LEMMA 2.1. *If $\mathscr{L}$ is a super*AFL, *$\mathscr{L}$ contains all unitary sets.*

For the next three lemmas we assume that $\mathscr{L}$ is a superAFL. We show in succession that $\mathscr{L}$ is closed under length-preserving homomorphism, that $\mathscr{L}$ is substitution closed and that $\mathscr{L}$ contains every set of the form a^*.

LEMMA 2.2. *$\mathscr{L}$ is closed under length-preserving homomorphism.*

Proof. Let $L \subseteq \Sigma_1^*$, and let $h : \Sigma_1^* \to \Sigma_2^*$ be a length-preserving homomorphism; that is, $h(a) \in \Sigma_2$ for all $a \in \Sigma_1$. For each $a \in \Sigma_1$, let $\hat{a}$ be new. Let $\Sigma_3 = \{\hat{a} \mid a \in \Sigma_1\}$. Let $\tau_1(a) = \{a, \hat{a}\}$ for all $a \in \Sigma_1$. Since $\mathscr{L}$ contains all unitary sets, $\tau_1(a) \in \mathscr{L}$. Let $L_1 = \tau_1^{\infty}(L) \cap \Sigma_3^*$. Then $\mathscr{L}$ contains L_1. Let $\tau_2(\hat{a}) = \{\hat{a}, h(a)\}$. Again, $\tau_2^{\infty}(L_1) \in \mathscr{L}$, and thus

$$h(L) = \tau_2^{\infty}(L_1) \cap \Sigma_2^* \in \mathscr{L}.$$

LEMMA 2.3. *$\mathscr{L}$ is substitution closed.*

[1] For a string w, $|w|$ is the length of w.

Proof. Let $L_1 \subseteq \Sigma_1{}^*$, $L_1 \in \mathscr{L}$ and let τ be a substitution, with $\tau(a) \in \mathscr{L}$, for $a \in \Sigma_1$. Let $\bigcup_a \tau(a) \subseteq \Sigma_2{}^*$. For each $a \in \Sigma_1$, let $\hat{a}$ be new and let $h_1(a) = \hat{a}$. Let $\tau_1(\hat{a}) = \{\hat{a}\} \cup \tau(a)$ for $a \in \Sigma_1$. As in the definition of iterated substitution we assume that $\tau_1(b) = \{b\}$ where not otherwise explicitly defined. Then clearly $\tau(L_1) = \tau_1{}^\infty(h_1(L_1)) \cap \Sigma_2{}^*$.

LEMMA 2.4. *$\mathscr{L}$ contains all a^* for single symbols a.*

Proof. Let $L \in \mathscr{L}$ be a nonunitary set; by definition $\mathscr{L}$ must contain at least one such set. Then L contains some string w with $|w| \geqslant 2$. Hence, $\{w\} \in \mathscr{L}$. Let $w = a_1 \cdots a_n$. Let b be a new symbol. Let $\tau_1(a_i) = \{b, e\}$.[2] Let $R = \{b, bb, e\}$. Then $R = \tau_1(\{w\}) \cap R \in \mathscr{L}$. Let $\tau_2(b) = R$. Then $\tau_2{}^\infty(R) = b^*$. Since $\mathscr{L}$ is closed under length-preserving homomorphism, $\mathscr{L}$ contains all a^*.

We have shown that every superAFL is closed under substitution, contains each a^*, and by definition is closed under intersection with regular sets. Hence by the corollary to Theorem 4 of [16], it must be a full AFL.

THEOREM 2.1. *Every super*AFL *is a substitution closed full* AFL.

COROLLARY 1. *Every super*AFL *contains all regular sets.*

COROLLARY 2. *$\mathscr{S}^\infty(\mathscr{L})$ is the least super*AFL *containing $\mathscr{L}$.*

Every superAFL contains more than all regular sets; it contains all context-free languages. Thus, in a sense to be made more precise in the next section, superAFLs correspond to full AFLs as context-free languages correspond to regular sets.

THEOREM 2.2. *Every super*AFL *contains all context-free languages.*

Proof. Let $\Sigma_1 = \{X, \ a_1, a_2, a_{-1}, a_{-2}\}$. Let $L = \{X\}$ and $\tau(X) = \{X, XX, a_1Xa_{-1}, a_2Xa_{-2}, e\}$. Obviously $\tau(X)$ is regular and so contained in any superAFL. The expression for $\tau(X)$ clearly corresponds to a standard grammar for the Dyck set on two letters, K_2. Hence

$$K_2 = \tau^\infty(L) \cap (\Sigma_1 - \{X\})^*.$$

By the Chomsky-Schützenberger Theorem, K_2 is a generator of the context-free languages, so $\mathscr{L}$ must contain all context-free languages.

As noted, the context-free languages are closed under nested iterated

[2] For a set A we let A^* be the monoid generated by A with identity e.

substitution and thus by Theorem 2.2 are the smallest superAFL. We note that the context-free languages are not closed under iterated substitution if the condition $a \in \tau(a)$ is violated. For example, if $L = \{a\}$, and $\tau(a) = \{aa\}$, then $\tau^{\infty}(a) = \{a^{2^n} \mid n \geqslant 1\}$.

We could show by a direct syntactic proof that if $\mathscr{L}$ is a full AFL, then every member of $\hat{\mathscr{S}}^{\infty}(\mathscr{L})$ can be obtained from $\mathscr{L}$ by one application of nested iterated substitution and intersection with regular sets of the form $\Sigma_1{}^*$, and that if $\mathscr{L}$ is full principal, then so is $\hat{\mathscr{S}}^{\infty}(\mathscr{L})$. It will be more convenient to postpone this to Section 4 as these results are direct corollaries of the theorems for the corresponding machine realizations.

3. Nested AFA

In [15] we saw that the operation on acceptors corresponding to the substitution closure of AFLs was a finite nesting of data structures of a given type. That is, an acceptor M is entitled to some fixed number k of tapes, which can be either active or inactive. Nesting means that only the last activated tape can be acted on; an inner tape cannot be reached again until the outer tapes are deactivated. Thus the machine has a last-in-first-out or pushdown store of tapes, limited in length to k tapes. If we take the union over all k of languages accepted by nested k-tape machines we get the substitution closure of the corresponding single tape family. We extend this by removing the finiteness condition, so that the machine may open as many nests as it chooses, always limited to treating only the last opened active nest. In other words, our machines have as data structure a pushdown store of data structures of the corresponding single-tape kind. As long as the base structure is nontrivial (i.e., not the type of tape which is always empty), these well-nested AFAs characterize superAFLs in precisely the way that AFAs characterize full AFLs [6].

Aho and Ullman [3] have shown that the indexed languages can be characterized by AFAs whose data structure is a pushdown store of pushdown stores, with an added duplicate order which replicates the topmost store. They call these degree 2 pushdown stores and show that this idea can be extended to degree n, for any n, and that all these families have decidable emptiness problems and are contained in the context-sensitive languages. We see in Sections 5 and 6 that in general the addition of a duplicate order to well-nested AFAs raises problems of representation. For example, we can find machine representations of the context-free languages such that adding the duplicate order yields all recursively enumerable sets.

Our notation for multitape or multinest AFAs will differ from that in [15] which does not easily generalize to the infinite case.

DEFINITION 3.1. An AFA *schema* is a quadruple $\mathcal{A} = (\Gamma, I, f, g)$ such that

(1) Γ and I are abstract sets of symbols, and

(2) f is a partial function, $f : \Gamma^* \times I \to \Gamma^*$ and g is a mapping from Γ^* into the finite subsets of Γ^* such that

  (a) $g(e) = \{e\}$ and $e \in g(y)$ if and only if $y = e$,

  (b) for each $z \in g(\mathcal{Y})$, there is a $u_z \in I$ with $f(x, u_z) = x$ whenever $z \in g(x)$, and

  (c) for each $u \in I$, there is a finite $\Gamma_u \subset \Gamma$ such that if $\Gamma_1 \subset \Gamma$, $y \in \Gamma_1{}^*$, then $f(y, u) \in (\Gamma_1 \cup \Gamma_u)^*$.

We extend the transition function f by defining $F^1(x, u) = f(x, u)$, and $F^{n+1}(x, u_1, ..., u_{n+1}) = f(F^n(x, u_1, ..., u_n), u_{n+1})$.

DEFINITION 3.2. An AFA schema $\mathcal{A} = (\Gamma, I, f, g)$ is *nontrivial* if there are $u_0, u_1, ..., u_n \in I$, $n \geqslant 1$, such that

$$f(e, u_0) \neq e \qquad \text{and} \qquad F^{n+1}(e, u_0, u_1, ..., u_n) = e.$$

DEFINITION 3.3. An AFA schema (Γ, I, f, g) is *finitely encoded* if $I \cup g(\Gamma^*)$ is finite.

Most of our results will hold only for nontrivial AFA schema. The concept of finitely encoded corresponds to the concept of full principal [7].

DEFINITION 3.4. A *nested* AFA is a couple $(\mathcal{A}, \mathcal{D})$, or $\mathcal{D}$ where $\mathcal{A}$ is understood, such that

(1) $\mathcal{A} = (\Gamma, I, f, g)$ is an AFA schema,

(2) there are three strings DUPL, PUSH, and POP not in I, and

(3) there is a countable set K and an infinite set Σ such that $\mathcal{D}$ is the family of all $M = (K_1, \Sigma_1, \delta, q_0, F)$ with

  (a) K_1 and Σ_1 finite, $K_1 \subset K$, $\Sigma_1 \subset \Sigma$, $q_0 \in K_1$, $F \subseteq K_1$, and

  (b) δ a mapping from $K_1 \times (\Sigma_1 \cup \{e\}) \times g(\Gamma^*)$ into the finite subsets of $K_1 \times (I \cup \{\text{DUPL, PUSH, POP}\})$ such that

$$G_M = \{z \mid \delta(q, a, z) \neq \varnothing \qquad \text{for some } q, a\}$$

  is finite.

We introduce notation to describe the languages defined by a machine M. An instantaneous description of M is a $n+2$ tuple $(q, w, y_1, \ldots, y_n)$ where $q \in K_1$, $w \in \Sigma_1^*$, and $y_i \in \Gamma^*$, $1 \leqslant i \leqslant n$, and n is some positive integer. We define a relation $\underset{M}{\vdash}$ (or $\vdash$ where M is understood) among instantaneous descriptions as follows.

Let $(q', u) \in \delta(q, a, z)$ and $z \in g(y)$. Then for all $w \in \Sigma_1^*, y, y_1, \ldots, y_n \in \Gamma^*$:

(1) if $u \in I$ and $x = f(y, u)$

$$(q, aw, y_1, \ldots, y_n, y) \vdash (q', w, y_1, \ldots, y_n, x),$$
$$(q, aw, y) \vdash (q', w, x),$$

(2) if $u = \text{PUSH}$

$$(q, aw, y_1, \ldots, y_n, y) \vdash (q', w, y_1, \ldots, y_n, y, e),$$
$$(q, aw, y) \vdash (q', w, y, e),$$

(3) if $u = \text{POP}$ and $y = e$

$$(q, aw, y_1, \ldots, y_n, y) \vdash (q', w, y_1, \ldots, y_n), \quad \text{and}$$

(4) if $u = \text{DUPL}$

$$(q, aw, y_1, \ldots, y_n, y) \vdash (q', w, y_1, \ldots, y_n, y, y),$$
$$(q, aw, y) \vdash (q', w, y, y).$$

As usual, we let $\overset{*}{\vdash}$ be the reflexive transitive closure of $\vdash$. Then $L(M) = \{w \mid \exists q \in F, (q_0, w, e) \overset{*}{\vdash} (q, e, e)\}$ and $\mathscr{L}(\mathscr{D}) = \{L(M) \mid M \in \mathscr{D}\}$.

We shall be interested in certain subfamilies of $\mathscr{D}$. If $\delta(q, a, z) \subseteq K_1 \times I$ for all q, a, z, then we call M *single tape* and let $\mathscr{D}^s$ be the family of all single tape acceptors and $\mathscr{L}^s(\mathscr{D}) = \{L(M) \mid M \in \mathscr{D}^s\}$. We call M *well-nested* if $\delta(q, a, z) \subseteq K_1 \times (I \cup \{\text{PUSH, POP}\})$ for all q, a, z, and let $\mathscr{D}^w$ be the family of all well-nested M, and $\mathscr{L}^w(\mathscr{D}) = \{L(M) \mid M \in \mathscr{D}^w\}$. We call M *finitely nested* if it is well-nested and there is an integer k such that if

$$(q_0, w, e) \overset{*}{\vdash} (q, e, y_1, \ldots, y_n),$$

then $n \leqslant k$; we say that M is k-nested. We let $\mathscr{D}^f$ be the family of all finitely nested M in $\mathscr{D}$, and let $\mathscr{L}^f(\mathscr{D}) = \{L(M) \mid M \in \mathscr{D}^f\}$.

It is obvious that we can tell if a machine is single tape or well-nested, although in most cases it is undecidable if $L(M)$ is in $\mathscr{L}^s(\mathscr{D})$ or $\mathscr{L}^w(\mathscr{D})$. We shall see in Section 5 that if $\mathscr{L}^s(\mathscr{D})$ has a decidable emptiness problem, then for some representations it is decidable if M is finitely nested, but not for others. We shall apply the terms "nontrivial" and "finitely encoded" to nested AFAs if they are true of the corresponding AFA schemata.

The single tape machines never employ the instructions PUSH, POP, or DUPL. It is evident that $(\mathcal{Cl}, \mathcal{D}^s)$ forms an AFA in the sense of [6] or a single tape AFA in the sense of [15]. Thus, we can borrow the results of [6], [7], and [15], and summarize them below.

THEOREM 3.1. *$\mathcal{L}$ is a full* AFL *if, and only if, there is a nested* AFA $(\mathcal{Cl}, \mathcal{D})$ *such that $\mathcal{L} = \mathcal{L}^s(\mathcal{D})$. $\mathcal{L}$ is a full principal* AFL *if, and only if, there is a finitely encoded nested* AFA $(\mathcal{Cl}, \mathcal{D})$ *such that $\mathcal{L} = \mathcal{L}^s(\mathcal{D})$. Furthermore, in each case we can assume that the corresponding* AFA *schema is nontrivial.*

We can think of the AFA schema as representing a particular kind of data structure. The machines in $\mathcal{D}$ have as storage structure a pushdown store whose members are tapes or nests of the kind specified by the AFA schema. The machine can act on the top nest in the manner of a single tape machine, it can open a new nest on top with the instruction PUSH, it can open a new nest on the top identical to the former top nest with the instruction DUPL, or it can use the instruction POP to remove the top nest. The top nest cannot be POPPED unless it is already reinitialized, that is, unless it contains the null string, e. In Section 5 we shall see that this condition makes a difference; there are AFA schemata whose single tape machines define only regular sets but whose two-tape machines would define every recursively enumerable set if allowed to POP nonnull nests.

In order to avoid reproving the results of [6], [7], and [15] we should like to show that each of $\mathcal{D}^f$, $\mathcal{D}^w$, and $\mathcal{D}$ are equivalent to a family of single tape automata with respect to the family of languages accepted. It is clear that instead of representing a string of nests as an ordered n tuple, $(y_1, \ldots, y_n)$ we could represent it as one string with appropriate separators: $y_1cy_2c \cdots cy_n$, and change the definition of f and g accordingly. The natural f' and g'would be defined by

$$f'(y_1cy_2c \cdots cy_n, u) = y_1cy_2c \cdots cf(y_n, u) \text{ and } g'(y_1cy_2c \cdots cy_n) = g(y_n).$$

The main difficulty is that g' would violate clause (2a) of the definition of an AFA schema, with $g'(e) = g'(\text{cec})$ and so on. That is, there would be no way for the machine to distinguish whether it was seeing the null string in the tenth nest or the first and so it could never know when the entire storage had been reinitialized to e. More colloquially, the machines cannot tell their "bottoms." It was noted in [6] that this condition, clause (2a), is needed for closure under concatenation and Kleene closure, though not for closure under the other AFL operations (homomorphism, inverse homomorphism, intersection with regular and union).

This difficulty would obviously be removed if we further modified g' to g'' with $g''(e) = \{e\}$ but $g''(c^n) = \{c\}$ for $n \geqslant 1$. Now it is no longer obvious that the new single tape machines can be simulated by the old multitape machines. Indeed, the simulation can be carried out if, and only if, we are dealing with a nontrivial AFA schema; otherwise neither $\mathscr{L}(\mathscr{D}^w)$ nor $\mathscr{L}(\mathscr{D})$ is a full AFL. We postpone the proof for Lemma 4.1 and Theorem 4.1 in the next section.

The proof that $\mathscr{L}(\mathscr{D}^f)$ is a full AFL is quite straightforward. We can clearly obviate the difficulty mentioned above if for each machine in $\mathscr{D}^f$ we could find another machine in $\mathscr{D}$, accepting the same language, and having a distinguished set of states which it enters when and only when it is in the bottom nest; such a new machine can tell whether it is in its bottom nest simply by the state it is in, without examining its storage.

The construction is obvious. Indeed, for any $M \in \mathscr{D}^w$ and k we can effectively find a k-tape machine M_k with the desired property and such that $L(M) = L(M_k)$ if M is k-tape. For the instruction PUSH means open a new nest and POP if executed means the number of nests has been decreased. Hence, M_k can easily count in its finite state control the number of active nests and block if the number would go above k. The states of M_k are divided into k equivalence classes such that M_k is in a state of the r-th class if, and only if, there are currently r active nests [i.e., M has an instantaneous description $(q, w, y_1, \ldots, y_r)$]. Thus, we can show that $\mathscr{L}^f(\mathscr{D})$ is a full AFL. Indeed, we can echo the results of [15] to get the following theorem.

THEOREM 3.2. *$\mathscr{L}^f(\mathscr{D})$ is a substitution closed full* AFL.

Proof. We have noted that $\mathscr{L}^f(\mathscr{D})$ is a full AFL. To show $\mathscr{L}^f(\mathscr{D})$ closed under substitution it is clearly sufficient to show it closed under bracketed substitutions of the form $\tau(a) = [_a L_a]_a$, where each bracket $[_a,]_a$ is distinct and different from all symbols appearing in any L_b. We assume that each L_a has a k_a-tape machine M_a and we are substituting into $L(M)$ where M is finite tape. The machine $\overline{M}$ for $\tau[L(M)]$ is constructed as follows. $\overline{M}$ starts by turning M on, computing on e input (i.e., without consulting the input tape). Whenever $\overline{M}$ sees a symbol $[_a$, it remembers the current state of M, and turns M_a on. During the computation of M_a on L_a, $\overline{M}$ counts the number of nests M_a opens, so that it can recognize an accepting configuration of M_a. If it sees $]_a$ in a nonaccepting configuration it blocks (and likewise if $[_b$ appears before $]_a$; if it sees $]_a$ in an accepting configuration of M_a, it turns off M_a and returns to M computing on a. Any accepting configuration of $M[(f, e, e)$ for final state $f]$ is of course an accepting configuration for $\overline{M}$, and these are the only accepting configurations; of course, after recognizing a member of L_a and turning M on again, $\overline{M}$ makes sure that it does not accept until it has indeed imitated M on a. Then $L(\overline{M}) = \tau(L(M))$, and $\overline{M}$ is finite tape if M is finite tape. Note that this construction as given is effective only if we are told that each M_a is k_a-tape or else there is an effective procedure to find k_a given M_a.

The nested finite tape machines defined in [15] possess a little more information than ours do. In our definition, if M is in a configuration $(q, w, y_1, \ldots, y_n)$ it only knows $g(y_n)$; in the other, it knows $g(y_1), \ldots, g(y_n)$. This difficulty can be eliminated by a tedious construction. Instead we show that $\mathscr{L}^f(\mathscr{D}) = \hat{\mathscr{S}}(\mathscr{L}^s(\mathscr{D}))$, which shows the models equivalent since in both cases we get the substitution closure of the single tape family. Since single tape machines are indeed finite tape, Theorem 3.2 yields that $\hat{\mathscr{S}}(\mathscr{L}^s(\mathscr{D})) \subseteq \mathscr{L}^f(\mathscr{D})$. The other half of our theorem is given by the following lemma which is also used in the next section to characterize $\mathscr{L}^w(\mathscr{D})$. Both the construction sketched in the proof of Theorem 3.2 and the one given in detail below are much simpler than the corresponding ones in [15] (although the ideas are the same) because the definitions in that paper, meant to cover the general multi-tape case as well as the case of finite nested tapes, are much more awkward when dealing with the nested case and cannot be easily extended to cover the general well-nested case.

LEMMA 3.1. *If M is in $\mathscr{D}^w$, we can effectively find a single tape machine M_1, a finite set Σ_1, a finite set T disjoint from Σ_1, and a single tape machine M_a for each a in T, such that if $\tau(a) = \{a\}$ for a in Σ_1 and $\tau(a) = \{a\} \cup L(M_a)$ for a in T, then*

(1) $L(M) = \tau^\infty(L(M_1)) \cap \Sigma_1^*$,
(2) $L(M) = L(M_1)$ if M is single tape, and
(3) $L(M) = \tau^{k-1}(L(M_1)) \cap \Sigma_1^*$ if M is k-tape for some $k > 1$.

Proof. Let M be well-nested. By an effective construction we mean effective with respect to a given description of M and a given choice function for picking new symbols. Let $M = (K_1, \Sigma_1, \delta, q_0, F)$. For each p, q in K_1 let (p, q) be a new symbol and let $T = \{(p, q) \mid p, q \in K_1\}$ and $\Sigma_2 = \Sigma_1 \cup T$. Let $K_2 = K_1 \cup (K_1 \times G_M)$, where $G_M = \{z \mid \delta(q, a, z) \neq \emptyset$ for some $q, a, z\}$. Let $M_1 = (K_2, \Sigma_2, \delta_1, q_0, F)$, where δ_1 is defined as follows. M_1 has all the instructions of M that do not involve PUSH or POP. That is, if $(q', u) \in \delta(q, a, z)$ and $u \in I$, then $(q', u) \in \delta_1(q, a, z)$. If $(p, \text{PUSH}) \in \delta(q', a, z)$, then $((p, z), u_z) \in \delta_1(q', a, z)$ and $(q, u_z) \in \delta_1((p, z), (p, q), z)$ for all $q \in K_1$, where u_z is the instruction defined in clause (2b) of the definition of AFA schema [i.e., $f(x, u_z) = x$ whenever $z \in g(x)$]. If M is single tape, there are no PUSH instructions, so that $L(M) = L(M_1)$. On the other hand, if w is in $L(M_1)$, and for each (p, q) in w we substitute some word $w_1 a$ such that $(p, w_1, e) \overset{*}{\vdash} (q', e, e)$ and (q, POP) is in $\delta(q', a, e)$, then the result is in $L(M)$. So we must define $\tau((p, q))$ accordingly. Let $\hat{f}$ be new and let $K_3 = K_2 \cup \{\hat{f}\}$, and let $M_{pq} = (K_3, \Sigma_2, \delta_{pq}, p, \{\hat{f}\})$, where δ_{pq} is defined below.

M_{pq} imitates M on Σ_1 as M_1 does. That is, $\delta(s, a, z) \cap (K_1 \times I) \subseteq \delta_{pq}(s, a, z)$ for all $s \in K_1$ and $a \in \Sigma_1 \cup \{e\}$. If $(s_2, \text{PUSH}) \in \delta(s, a, z)$, then $((s_2, z), u_z) \in \delta_{pq}(s_1, a, z)$ and $(s_3, u_z) \in \delta_{pq}((s_2, z), (s_2, s_3), z)$ for all $s_3 \in K_1$. Finally, for any rule $(q, \text{POP}) \in \delta(q', a, e)$, M_{pq} has $(\hat{f}, u_e) \in \delta_{pq}(q', a, e)$. We let $\tau(a) = \{a\}$ for $a \in \Sigma_1$, and $\tau((p, q)) = \{(p, q)\} \cup L(M_{pq})$ for $(p, q) \in T$. Clearly each $\tau(a)$ and $\tau((p, q))$ is in $\mathscr{L}^s(\mathscr{D})$.

Now wa is in $L(M_{pq})$ for $w \in \Sigma_1{}^*$, $a \in \Sigma_1 \cup \{e\}$, if, and only if, there is a q' such that $(p, w, e) \overset{*}{\underset{M}{\vdash}} (q', e, e)$ and $(q, \text{POP}) \in \delta(q', a, e)$ and during some computation of $(p, w, e) \overset{*}{\underset{M}{\vdash}} (q', e, e)$ no new nest is opened. If $wa \in \Sigma_1{}^*$ and $L(M_{pq})$ contains some word $wa(\hat{p}, \hat{q})\ldots$, then M has a computation $(p, w, e) \overset{*}{\underset{M}{\vdash}} (p', e, y)$ in which no new nest is opened and $(\hat{p}, \text{PUSH}) \in \delta(p', a, z)$ for some $z \in g(y)$. Thus $\tau(L(M_1)) \cup \Sigma_1{}^*$ contains all and only words w for which M has some computation in which at most two nests are open at the same time. Continuing, we can show by induction that for each n, w is in $\tau^n(L(M_1)) \cap \Sigma_1{}^*$ if, and only if, M has an accepting computation for w in which at most $n + 1$ nests are open at any time. If M is k-tape, then no computation and in particular no accepting computation ever has more than k open nests. Hence, if M is k-tape for $k > 1$, then $L(M) = \tau^{k-1}(L(M_1))$ and if M is single tape, $L(M) = L(M_1)$. Now be definition w is in $L(M)$ if and only if there is some accepting computation for w. This computation must have some maximum number m of simultaneously open nests. If $m = 1$, then w is in $L(M_1)$ and hence in $\tau^r(L(M_1))$ for any $r \geqslant 1$. If $m > 1$, then w is in $\tau^{m-1}(L(M_1))$. Hence in any case, $L(M) = \tau^\infty(L(M_1)) \cap \Sigma_1{}^*$.

Lemma 3.1 together with Theorem 3.2 yields the desired characterization.

Theorem 3.3. $\mathscr{L}^f(\mathscr{D}) = \hat{\mathscr{S}}(\mathscr{L}^s(\mathscr{D}))$.

Since $\mathscr{L}^f(\mathscr{D})$ is its own substitution closure and $\mathscr{L}^f(\mathscr{D})$ is a full AFL we can rephrase Theorem 3.2 as a characterization theorem for substitution closed full AFLs.

Theorem 3.3. *The following are equivalent*:

(1) *$\mathscr{L}$ is a substitution closed full* AFL.

(2) *There is a full* AFL $\mathscr{L}'$ *such that* $\mathscr{L} = \hat{\mathscr{S}}(\mathscr{L}')$.

(3) *There is a nested* AFA $(\mathfrak{A}, \mathscr{D})$ *such that* $\mathscr{L} = \mathscr{L}^f(\mathscr{D})$.

(4) *There is a nested* AFA $(\mathfrak{A}, \mathscr{D})$ *such that* $\mathscr{L} = \mathscr{L}^s(\mathscr{D}) = \mathscr{L}^f(\mathscr{D})$.

COROLLARY. *If* $\mathscr{D}$ *is finitely encoded, then* $\mathscr{L}^f(\mathscr{D})$ *is full principal if, and only if,* $\mathscr{L}^f(\mathscr{D}) = \mathscr{L}^s(\mathscr{D})$.

Proof. If $\mathscr{L}^s(\mathscr{D}) \neq \mathscr{L}^f(\mathscr{D})$, then $\mathscr{L}^s(\mathscr{D})$ is a full AFL that is not substitution closed; hence its substitution closure cannot be full principal [14].

Lemma 3.1 also gives the first half of the characterization theorem for $\mathscr{L}^w(\mathscr{D})$ we shall prove in the next section.

THEOREM 3.4. $\mathscr{L}^w(\mathscr{D}) \subseteq \hat{\mathscr{S}}^\infty(\mathscr{L}^s(\mathscr{D}))$.

4. CHARACTERIZATION THEOREMS FOR WELL-NESTED AFA

In this section our main goal shall be to show that $\mathscr{L}(\mathscr{D})$ is a superAFL and $\mathscr{L}^w(\mathscr{D}) = \hat{\mathscr{S}}^\infty(\mathscr{L}^s(\mathscr{D}))$ if, and only if $\mathscr{D}$ is nontrivial. First we must show that $\mathscr{L}^w(\mathscr{D})$ and $\mathscr{L}(\mathscr{D})$ are full AFLs if, and only if, $\mathscr{D}$ is nontrivial. We could subsume this under the proof that they are superAFLs, except that we also wish to show that if $\mathscr{D}$ is finitely encoded they are full principal. This involves showing that for $\mathscr{D}^w$ and $\mathscr{D}$ we can find equivalent single-tape AFAs which are finitely encoded if $\mathscr{D}$ is finitely encoded.

As we mentioned before, the construction of the desired single tape AFA is trivial. The difficulty comes in showing that for each machine M in the original AFA, $\mathscr{D}$, there is an equivalent machine in $\mathscr{D}$ which "knows its bottom." To show this we introduce the useful notion of "punctuation" which is crucial to the constructions of this section. If $\mathscr{D}$ is nontrivial, then there are instructions $u_0, u_1, ..., u_n$ such that $f(e, u_0) = \alpha \neq e$, and

$$F^n(\alpha, u_1, ..., u_n) = F^{n+1}(e, u_0, u_1, ..., u_n) = e.$$

So we shall use alternating nests with sole content e or α to encode desired punctuation or information such as: we are in the bottom nest. After picking up the encoded information we can eliminate the punctuation nests by using POP on e and $u_1, ..., u_n$ followed by POP on α. We shall abbreviate the nest e by E and the nest α by A and use an expression like $EAAE$ to represent the succession of nests: e, α, α, e. Clause (2a) of the definition of an AFA schema insures that although a machine cannot distinguish which nest is on the bottom, it can distinguish E from A; that is, $g(e) \cap g(\alpha) = \emptyset$.

The details of manipulating the finite state controls of AFAs or one-way nondeterministic balloon automata are thoroughly documented in [6], [15], and [17]. In the constructions in this section we shall generally sketch the actions of the desired machines in words and omit the detailed formalism which usually would only obscure the idea of the proof.

LEMMA 4.1. *Let $(\mathcal{A}, \mathcal{D})$ be a nontrivial nested* AFA. *There are nested* AFA *$(\mathcal{A}_1, \mathcal{D}_1)$ and $(\mathcal{A}_2, \mathcal{D}_2)$ such that $\mathcal{L}(\mathcal{D}) = \mathcal{L}^s(\mathcal{D}_1)$ and $\mathcal{L}^w(\mathcal{D}) = \mathcal{L}^s(\mathcal{D}_2)$ and $\mathcal{A}_1$ and $\mathcal{A}_2$ are finitely encoded if $\mathcal{A}$ is.*

Proof. Let $\mathcal{A} = (\Gamma, I, f, g)$. Let c be a new symbol. Let $\bar{\Gamma} = \Gamma \cup \{c\}$ and $\bar{I} = I \cup \{\text{DUPL}, \text{PUSH}, \text{POP}\}$. Let $\bar{g}(xcy) = \bar{g}(y) = g(y)$ for $y \in \Gamma^*$ and $y \neq e$, let $\bar{g}(xc) = \{c\}$, and let $\bar{g}(e) = \{e\}$. Let $\bar{f}(xcy, u) = xcf(y, u)$ and $\bar{f}(y, u) = f(y, u)$ for y in Γ^* and $u \in I$. Let $\bar{f}(x, \text{PUSH}) = xc$ and $\bar{f}(xc, \text{POP}) = x$ for all $x \in (\Gamma \cup \{c\})^*$; let $\bar{f}(xcy, \text{DUPL}) = xcycy$ and $\bar{f}(y, \text{DUPL}) = ycy$ for $y \in \Gamma^*$ and $y \neq e$. Let $\bar{f}'$ be the restriction of $\bar{f}$ to $(\Gamma \cup \{c\})^* \times (I \cup \{\text{PUSH}, \text{POP}\})$. Let $\mathcal{A}_1 = (\bar{\Gamma}, \bar{I}, \bar{f}, \bar{g})$ and $\mathcal{A}_2 = (\bar{\Gamma}, I \cup \{\text{PUSH}, \text{POP}\}, \bar{f}', \bar{g})$. Clearly $\mathcal{A}_1$ and $\mathcal{A}_2$ are finitely encoded if $\mathcal{A}$ is finitely encoded. We claim that they have the desired properties.

We shall treat the two cases simultaneously since we note that $\mathcal{A}_2$ is distinguished from $\mathcal{A}_1$ precisely as $\mathcal{D}^w$ differs from $\mathcal{D}$: the absence of the DUPL instruction. First we note that $\mathcal{L}(\mathcal{D}) \subseteq \mathcal{L}^s(\mathcal{D}_1)$ and $\mathcal{L}^w(\mathcal{D}) \subseteq \mathcal{L}^s(\mathcal{D}_2)$. All the instructions of $\mathcal{A}$ correspond exactly to the instructions of $\mathcal{A}_1$ and $\mathcal{A}_2$ with the same name. One slight difference is that $\mathcal{A}_1$ does not allow the instruction DUPL to be applied to a null nest, that is, to e. But clearly any instruction $(q', \text{DUPL}) \in \delta(q, a, e)$ has the precise effect of $(q', \text{PUSH}) \in \delta(q, a, e)$, so this restriction has no effect on the families of languages defined. The only other difference is that any instruction (q', u) in $\delta(q, a, e)$ must be represented as two instructions, (q', u) in $\delta'(q, a, e)$ and (q', u) in $\delta'(q, a, c)$. Otherwise any M in $\mathcal{D}$ translates directly into M' in $\mathcal{D}_1$, with the same transition function and accepting the same set of languages, and M' is in $\mathcal{D}_2$ if M has no DUPL instructions.

The other half of the proof does depend on the fact that $\mathcal{A}$ is nontrivial so that we have available the punctuation nests, A and E. For any M in $\mathcal{D}_1$ we want to find M_1 in $\mathcal{D}$ accepting the same language and such that M_1 is in $\mathcal{D}^w$ if M has no DUPL instruction. We note that the only time M knows whether or not it is in the bottom nest is when it is in a nest with null storage; then $e \in \bar{g}(y)$ indicates that it is at the bottom and $c \in \bar{g}(y)$ indicates that it is not. So that is the only time M_1 needs to find out where it is. M_1 has five parts.

(1) *Initialization.* M_1 starts by putting down punctuation AE and then using PUSH to start a new nest and enter the initial state of M. That is, if q_0 is the initial state of M and $\bar{q}_0$ the initial state of M_1, we have $(\bar{q}_0, e, e) \overset{*}{\vdash}_{M_1} (q_0, e, \alpha, e, e)$.

(2) *Bottom Search.* Any time M_1 notices its current nest is e, it examines the nests below. If the nest below is not E or if the two nests below are EE [see part (3) below] it returns to place, restoring any nests POPPED, and tries to execute some instruction $(q', u) \in \delta(q, a, c)$ of M. If the two nests are AE, it restores them, returns to place and tries to execute some instruction $(q', u) \in \delta(q, a, e)$ of M.

(3) *Simulation of* $I \cup \{\text{DUPL}, \text{PUSH}\}$. M_1 imitates without change those instructions of M in I with nonnull storage. Before executing instructions with null storage M_1 examines the nests below as indicated in (2). Whenever M has a DUPL instruction, so does M_1; this never takes place with null storage. Whenever M has a PUSH instruction, M_1 first lays down punctuation EE and then executes the PUSH instruction of M.

(4) *Simulation of* POP. An instruction $(q', \text{POP}) \in \delta(q, a, e)$ of M could not be executed, since M would be in its bottom nest; hence, we can assume M has no such instructions. An instruction $(q', \text{POP}) \in \delta(q, a, c)$ in M corresponds to a computation $(q, a, y_1c \cdots y_nc) \vdash_{\overline{M}} (q', e, y_1c \cdots y_{n-1}cy_n)$. M_1 tests whether it is in its bottom nest as described in (2). If it is in the bottom, it hangs up Otherwise it POPs any punctuation nests below and returns control to M; it has either $(q, a, \ldots, y_n, e, e, e) \overset{*}{\vdash}_{M_1} (q', e, \ldots, y_n)$ or else $(q, a, \ldots, y_n, e) \vdash_{\overline{M_1}} (q', e, \ldots, y_n)$. In the former case the nest just POPPED was initiated by a PUSH instruction and in the latter case by a DUPL instruction.

(5) *Acceptance.* If M_1 is in a final state of M with null storage it nondeterministically decides to examine the two nests below. If they are AE it erases them and accepts. Otherwise it blocks.

Hence, M_1 can imitate all the computations of M and accept precisely the same set of words; if M has no DUPL instructions, neither does M_1.

The preceding lemma combined with Theorem 3.1 gives us at once companion results for $\mathscr{L}(\mathscr{D})$ and $\mathscr{L}^w(\mathscr{D})$.

THEOREM 4.1. *$\mathscr{L}(\mathscr{D})$ and $\mathscr{L}^w(\mathscr{D})$ are full* AFLs *if, and only if, $\mathscr{D}$ is nontrivial. If $\mathscr{D}$ is nontrivial and finitely encoded, then $\mathscr{L}(\mathscr{D})$ and $\mathscr{L}^w(\mathscr{D})$ are full principal* AFLs.

Proof. If $\mathscr{D}$ is nontrivial, Lemma 4.1 and Theorem 3.1 yield our result. Now if $\mathscr{D}$ is not nontrivial, either $f(e, u) = e$ for all instructions u, so that any machine M always has null storage or else if M ever has nonnull storage it can never return to null storage and accept. In either case, $\mathscr{L}^s(\mathscr{D})$ must be the regular sets. We can ignore any instructions u with $f(e, u) \neq e$. Hence, DUPL and PUSH are always identical in effect, and so $\mathscr{L}(\mathscr{D}) = \mathscr{L}^w(\mathscr{D})$. Piling up E nests causes an automaton to act like a counter that cannot distinguish between 0 and 1. As noted in [6] the family of languages defined by such machines is not a full AFL (although it is closed under homomorphism, inverse homomorphism, intersection with regular sets and union).

The next theorem utilizes a construction similar to the one in Lemma 4.1 but more complicated. In order to show closure under nested iterated substitution we must encode more than merely the fact that a nest is the bottom nest; however as long as the amount of information is finite it can be encoded in much the same way it is encoded in a pushdown store.

THEOREM 4.2. *If $\mathscr{D}$ is nontrivial, $\mathscr{L}(\mathscr{D})$ and $\mathscr{L}^w(\mathscr{D})$ are super*AFLs.

Proof. We shall show this for $\mathscr{L}(\mathscr{D})$; the construction will introduce no new DUPL instructions and hence will also hold for $\mathscr{L}^w(\mathscr{D})$.

We have seen that $\mathscr{L}(\mathscr{D})$ and $\mathscr{L}^w(\mathscr{D})$ are full AFLs. Hence, it only remains to show them closed under nested iterated substitution. As before it is sufficient to consider bracketed substitutions. Namely, let $M \in \mathscr{D}$ and $L = L(M)$, $L \subseteq \Sigma_1^*$. For each a in Σ_1, let M_a be in $\mathscr{D}$ and let $[_a$ and $]_a$ be new symbols not appearing in Σ_1 or any M_b. Let $\tau(a) = \{a\} \cup [_a L(M_a)\,]_a$ for a in Σ_1, and $\tau(b) = \{b\}$ elsewhere. Since $\mathscr{L}(\mathscr{D})$ is a full AFL it suffices to show $\tau^\infty(L)$ in $\mathscr{L}(\mathscr{D})$.

Again we assume that M and all the M_a use DUPL only on nonnull nests. Let m be the maximum number of states of M or the M_a and let the states be ordered in some way. Let $\Sigma_1 = \{a_1, ..., a_n\}$. In this case we shall use $m + n + 4$ nests. The nest E^{m+n+4} will mean no change in status. AE^{m+n+3} will mean that the nests for L lie below. $EEA^iE^{(n-i)+1}A^jE^{(m-j)+1}$ will encode state q_j of machine M_{a_i}; $EEA^iE^{(n-i)+m+2}$ will encode a_i. Let Σ_2 contain Σ_1 and all symbols appearing in the M_a; the other symbols, the $[_a$ and $]_a$ we shall call the bracket symbols. We construct M_1 to recognize $\tau^\infty(L)$. M_1 will act in five phases, each controlled by the sort of input it sees; Phases (III) and (IV) have one subphase for each a_i; Phase (V) has a subphase for each pair (a_i, a_j). M_1 starts out in Phase (I) imitating M from the initial state of M.

Phase (I)

Input in $\Sigma_2 \cup \{e\}$: M_1 imitates the behavior of M.

Input $[_{a_i}$: M_1 remembers the current state of M in its finite state control. It puts down $AE^{m+n+3}EEA^iE^{(n-i)+m+2}$. In the future, seeing AE^{m+n+3} will return control to (I). M_1 remembers a_i in its finite state control and proceeds to Phase (II), opening a new nest.

Input $]_b$: M_1 blocks.

Phase (II)

Input in $\Sigma_2 \cup \{e\}$: M_1 has some a_i stored in its finite state control; it turns on M_{a_i} in its initial state and goes to Phase $(\text{III})_i$.

Input $[_{a_j}$: M_1 forgets previous a_i in its finite state control and remembers a_j. It puts down $EEA^jE^{(n-j)+m+2}$ and remains in this phase, opening a new nest.

Input $]_b$: M_1 blocks.

Phase $(\text{III})_i$

Input $\Sigma_2 \cup \{e\}$: In this subphase, $(\text{III})_i$, M_1 has turned on machine M_{a_i}. It continues to imitate M_{a_i}. Instructions in $I \cup \{\text{DUPL}\}$ are imitated without change. When pushing down, M_1 first lays down punctuation E^{m+n+4}. When popping, M_1 first checks the nest below. If it is not E, M_1 stays there and continues imitating M_{a_i}. If it is, M_1 checks for E^{m+n+4}. If it finds these nests, it pops them up in succession and continues in the nest below them. Otherwise, it blocks.

Input $[_{a_j}$: If M_1 is imitating state q_k of M_{a_i}, it puts down $EEA^iE^{(n-i)+1}A^kE^{(m-k)+1}EEA^jE^{(n-j)+m+2}$. Then it forgets a_i, remembers a_j and goes to Phase (II).

Input $]_{a_i}$: If M_1 is not in a final state of M_{a_i} and with null storage (e) it blocks. If it is, it examines the $m+n+4$ nests below. If they are not $EEA^iE^{(n-i)+m+2}$ it blocks. If they are, it POPs them all and then checks the $m+n+4$ nests below. If they are AE^{m+n+3}, then M_1 POPs them, returns control to M in the remembered state, and proceeds to subphase $(\text{IV})_i$. If they are $EEA^rE^{(n-r)+1}A^jE^{(m-j)+1}$, where q_j is a state of M_{a_r}, then M_1 POPs them and goes to subphase $(\text{V})_{r,i}$ in state q_j of M_{a_r}. Otherwise, M_1 blocks.

Input $]_{a_j}$: For $j \neq i$, M_1 blocks. The case $j = i$ was treated above.

Phase $(\mathrm{IV})_i$

Input e: M_1 imitates M acting on input from $\{a_i, e\}$. If it imitates M on input $e((q', u) \in \delta(q, e, z))$ it stays in Phase $(\mathrm{IV})_i$. As soon as it imitates M on input $a_i((q', u) \in \delta(q, a_i, z))$ it goes to Phase (I).

$\left.\begin{array}{l}\text{Input in } \Sigma_2 \\ \text{Input } [_a \text{ or }]_b\end{array}\right\} M_1$ blocks.

Phase $(\mathrm{V})_{r,i}$

Input e: M_1 imitates M_{a_r} acting on input from $\{a_i, e\}$. As in Phase $(\mathrm{IV})_i$, if it imitates M_{a_r} on input e it stays in Phase $(\mathrm{V})_{r,i}$. As soon as it imitates M_{a_r} on input a_i it goes to Phase $(\mathrm{III})_r$. Just as in Phase $(\mathrm{III})_r$, instructions in $I \cup \{\mathrm{DUPL}\}$ are imitated without change. When pushing down, M_1 first lays down punctuation E^{m+n+4}. When popping, M_1 first checks the nest below. If it is not E, the popped nest was opened by a DUPL instruction, so M_1 stays there and continues imitating M_{a_r}. If the nest is E, M_1 checks for E^{m+n+4}. If it finds these nests, it pops them up in succession and continues in the nest below them. Otherwise, it blocks.

$\left.\begin{array}{l}\text{Input in } \Sigma_2 \\ \text{Input } [_a \text{ or }]_b\end{array}\right\} M_1$ blocks.

M_1 will accept only when it is in Phase (I) and in an accepting configuration of M. M_1 is using the pushdown mode to check that the brackets are nested, in the same way a pushdown store can recognize a Dyck set, and to remember the state it was in while checking a word of $L(M_a)$ when it finds a word of $L(M_b)$ nested inside. The punctuation insures that when M_1 reads $]_a$ it will block unless it has just imitated an accepting configuration of M_a, that is, recognized a member of $L(M_a)$. When inside innermost $[_a$ and $]_a$ brackets, M_1 imitates M_a. When outside all such nested brackets, it imitates M. Thus $L(M_1) = \tau^\infty(L)$.

Theorems 4.1 and 4.2 and Theorem 3.4 yield the desired characterization of $\mathscr{L}^w(\mathscr{D})$.

THEOREM 4.3. *$\mathscr{L}^w(\mathscr{D}) = \hat{\mathscr{S}}^\infty(\mathscr{L}^s(\mathscr{D}))$ if, and only if, $\mathscr{D}$ is nontrivial.*

We can rephrase Theorems 4.2 and 4.3 to give characterization theorems for superAFLs.

THEOREM 4.4. *The following are equivalent.*

(1) $\mathscr{L}$ *is a super*AFL.

(2) *There is a nested* AFA $(\mathfrak{A}, \mathscr{D})$ *with* $\mathscr{L} = \mathscr{L}^w(\mathscr{D})$.

THEOREM 4.5. *Let* $\mathscr{L}$ *be a full* AFL. *Then the following are equivalent*:

(1) $\mathscr{L}' = \hat{\mathscr{S}}^\infty(\mathscr{L})$.

(2) $\mathscr{L}' = \{\tau^\infty(L) \cap R \mid L \in \mathscr{L}, a \in \tau(a) \in \mathscr{L}, R \textit{ regular}\}$.

(3) *There is a nontrivial nested* AFA $\mathscr{D}$ *with* $\mathscr{L} = \mathscr{L}^s(\mathscr{D})$ *and* $\mathscr{L}' = \mathscr{L}^w(\mathscr{D})$.

Theorem 4.5 allows us to prove for superAFLs in general a weak version of the Bar-Hillel, Perles and Shamir lemma for context-free languages [4].

THEOREM 4.6. *Let* $\mathscr{L}$ *be a full* AFL. *If* $L \in \hat{\mathscr{S}}^\infty(\mathscr{L}) - \hat{\mathscr{S}}(\mathscr{L})$, *then there are strings* u, v, w, x, y *with* $vx \neq e$ *such that for all* $k \geqslant 0$, $uv^kwx^ky \in L$.

Proof. By Theorem 4.5(3) there is a nontrivial nested AFA $\mathscr{D}$ with $\mathscr{L} = \mathscr{L}^s(\mathscr{D})$ and $\hat{\mathscr{S}}^\infty(\mathscr{L}) = \mathscr{L}^w(\mathscr{D})$. Now if L is not in $\hat{\mathscr{S}}(\mathscr{L})$, then it cannot be recognized by any automaton in $\mathscr{D}^f$. Let $L = L(M)$, where M is well-nested and has r states. If there is a k such that for each $w \in L$, M has an accepting computation for w in which at most k nests are open at the same time, then we can construct from M a machine M_k in $\mathscr{D}^f$ such that $L(M) = L(M_k)$. Hence, for every n there must be a word $z \in L(M)$ such that every accepting computation for z has at least n open nests at some time. In particular, take $n = r^2 + 2$. Let z be as described and consider some accepting computation for z of minimal length (i.e., no accepting computation for z is shorter.) By definition of z, this accepting computation has at least n open at some time. In an accepting computation every nest must have been PUSHed open in some state and later POPPed out in some state. If at least $r^2 + 2$ nests are open, two of them must have been opened in the same state and will be closed in the same state. That is we can factor $z = uvwxy$ such that for appropriate states and storage strings (f is final):

$$
\begin{aligned}
(q_0, uvwxy, e) &\overset{*}{\vdash} (q, vwxy, y_1, \dots, y_s, e) \overset{*}{\vdash} (q, wxy, y_1, \dots, y_s, \bar{y}_1, \dots, \bar{y}_t, e)\\
&\overset{*}{\vdash} (p, xy, y_1, \dots, y_s, \bar{y}_1, \dots, \bar{y}_t)\\
&\overset{*}{\vdash} (p, y, y_1, \dots, y_s)\\
&\overset{*}{\vdash} (f, e, e)
\end{aligned}
$$

Furthermore, this computation is such that no nest y_i is reached during the computation on vwx and no nest y_j is reached during the computation on w. That is, we have:

$$(q, v, e) \overset{*}{\vdash} (q, e, \bar{y}_1, ..., \bar{y}_t, e); \qquad (q, w, \bar{y}_t, e) \overset{*}{\vdash} (p, e, \bar{y}_t);$$

$(p, x, y_s, \bar{y}_1, ..., \bar{y}_t) \overset{*}{\vdash} (p, e, y_s)$; and $(p, y, y_1, ..., y_s) \overset{*}{\vdash} (f, e, e)$, and $t \geqslant 1$.

But then

$$(q, vv, e) \overset{*}{\vdash} (q, e, \bar{y}_1, ..., \bar{y}_t, \bar{y}_1, ..., \bar{y}_t)$$

and

$$(p, xx, y_s, \bar{y}_1, ..., \bar{y}_t, \bar{y}_1, ..., \bar{y}_t) \overset{*}{\vdash} (p, e, y_s).$$

Thus, we can find an accepting computation for uv^kwx^ky for all k. If $vx = e$, then $(q_0, uwy, e) \overset{*}{\vdash} (q, wy, y_1, ..., y_s, e) \overset{*}{\vdash} (p, y, y_1, ..., y_s) \overset{*}{\vdash} (f, e, e)$ would be an accepting computation for z of shorter length. Hence, $vx \neq e$ as desired.

We notice that if a full AFL is full principal, so is its closure under nested iterated substitution.

THEOREM 4.7. *If $\mathscr{L}$ is a full principal* AFL, *then $\mathscr{S}^\infty(\mathscr{L})$ is full principal.*

Proof. If $\mathscr{L}$ is full principal, then by Theorem 3.1 we can find a nontrivial finitely encoded nested AFA $(\mathfrak{A}, \mathscr{D})$ with $\mathscr{L} = \mathscr{L}^s(\mathscr{D})$. By Theorem 4.1, $\mathscr{L}^w(\mathscr{D})$ is a full principal AFL; by Theorem 4.3, $\mathscr{L}^w(\mathscr{D}) = \mathscr{S}^\infty(\mathscr{L})$.

Finally, we use Theorem 4.7 to show that if a full AFL is not substitution closed, then its substitution closure is properly contained in its closure under nested iterated substitution.

DEFINITION 4.1. If $\mathscr{L}$ is a full AFL, let $\hat{\mathscr{S}}_1(\mathscr{L}) = \mathscr{L}$, and for $n \geqslant 1$, let $\hat{\mathscr{S}}_{n+1}(\mathscr{L}) = \{\tau_1(L) \mid L \in \hat{\mathscr{S}}_n(\mathscr{L}), \text{ each } \tau_1(a) \in \mathscr{L}\}$.

THEOREM 4.8. *If $\mathscr{L}$ is a full* AFL *that is not substitution closed, then $\mathscr{L} \subsetneq \hat{\mathscr{S}}(\mathscr{L}) \subsetneq \mathscr{S}^\infty(\mathscr{L})$.*

Proof. If $\mathscr{L}$ is not substitution closed, then there is a language $L \in \mathscr{L}$, $L \subseteq \Sigma_1{}^*$ and a substitution τ with $\tau(a) \in \mathscr{L}$ for a in Σ_1 such that $\tau(L)$ is not in $\mathscr{L}$. Let $Q = \{L\} \cup \{\tau(a) \mid a \in \Sigma_1\}$. Let $\mathscr{L}_1 = \hat{\mathscr{F}}(Q)$. Then $\mathscr{L}_1$ is a full principal AFL [7]. Then $\tau(L) \in \hat{\mathscr{S}}_2(\mathscr{L}_1) - \mathscr{L}$. Hence by Lemma 2.2 of [14], $\hat{\mathscr{S}}_{2n}(\mathscr{L}_1) - \hat{\mathscr{S}}_n(\mathscr{L}) \neq \emptyset$ for all n. Now $\hat{\mathscr{S}}(\mathscr{L}_1) = \bigcup_n \hat{\mathscr{F}}_n(\mathscr{L})$, and $\hat{\mathscr{S}}(\mathscr{L}) = \bigcup_n \hat{\mathscr{S}}_n(\mathscr{L})$ [15]. Hence, no $\hat{\mathscr{S}}_n(\mathscr{L})$ contains all of $\hat{\mathscr{S}}(\mathscr{L}_1)$. But $\mathscr{S}^\infty(\mathscr{L}_1)$ contains all of $\hat{\mathscr{S}}(\mathscr{L}_1)$ and is a full principal AFL. Let L_1 be a generator of $\mathscr{S}^\infty(\mathscr{L}_1)$. If $L_1 \in \hat{\mathscr{S}}(\mathscr{L})$, then $L_1 \in \hat{\mathscr{S}}_n(\mathscr{L})$ for some n, so that $\hat{\mathscr{S}}(\mathscr{L}_1) \subseteq$

$\mathscr{S}^\infty(\mathscr{L}_1) = \hat{\mathscr{F}}(L_1) \subseteq \mathscr{S}_n(\mathscr{L})$. But that is impossible. Consequently, $L_1 \in \mathscr{S}^\infty(\mathscr{L}) - \hat{\mathscr{S}}(\mathscr{L})$.

Example

The one-way stack languages [8] form a full AFL $\mathscr{L}$ that is not substitution closed [13], [21]. Hence by Theorem 4.8, the substitution closure of $\mathscr{L}$ is properly contained in $\mathscr{S}^\infty(\mathscr{L})$; by Theorem 4.3, $\mathscr{S}^\infty(\mathscr{L})$ is the family of languages accepted by one-way well-nested stack automata. Aho and Ullman [3] have shown that the indexed languages, $\mathscr{L}_I$ [1], [2] are a superAFL; this superAFL is full principal since it can be obtained from a finitely encoded nested AFA. We shall show that $\mathscr{L}_I$ is not in fact the least superAFL containing $\mathscr{L}$; that is, $\mathscr{S}^\infty(\mathscr{L})$ is properly contained in the indexed languages. The language $L = \{a^{2^n} b^n a^{2^n} \mid n \geqslant 1\}$ is an indexed language [it can be generated by the linear basic macro grammar $S \to F(aa, b, aa)$, $F(x, y, z) \to F(xx, yb, zz)$, $F(x, y, z) \to xyz$; see [5] for details]. It is not a one-way stack language [20]. By Theorem 4.6, L cannot be in $\mathscr{S}^\infty(\mathscr{L}) - \hat{\mathscr{S}}(\mathscr{L})$. It remains to show that L is not in $\hat{\mathscr{S}}(\mathscr{L})$.

It remains to show that L is not in $\hat{\mathscr{S}}(\mathscr{L})$. Observe that L has the property that, if R and S are any two finite sets with $RS \subseteq L$, then either $\#R \leqslant 1$ or else $\#S \leqslant 1$.[3] We use the notation of the proof of Theorem 4.8 and show that if $\mathscr{L}_1$ is any full AFL such that $L \in \hat{\mathscr{S}}_2(\mathscr{L}_1)$, then $L \in \mathscr{L}_1$ (this holds for $\{a^{f(n)} b^n a^{f(n)} \mid n \geqslant 1\}$ whenever f is any one-one function from the positive integers into the positive integers). This will complete the demonstration, since if $L \in \hat{\mathscr{S}}(\mathscr{L})$, then $L \in \hat{\mathscr{S}}_k(\mathscr{L})$ for some k and if $2^n \geqslant k$, then $L \in \hat{\mathscr{S}}_{2^n}(\mathscr{L})$, so by induction we would have $L \in \mathscr{L}$. Assume $L = \tau(L_1)$, where $L_1 \subseteq \Sigma_1{}^*$ and τ is a substitution, with $L_1 \in \mathscr{L}_1$ and $\tau(c) \in \mathscr{L}_1$ for c in Σ_1. We can assume that $\tau(c) \neq \phi$ and $L_1 \cap \Sigma_1{}^* c \Sigma_1{}^* \neq \phi$ for each c in Σ_1. Suppose $\#\tau(c) \geqslant 2$ and $xcy \in L_1$. If $a^{2^n} b^k \in \tau(x)$ for $k \geqslant 1$, then $\#\tau(cy) = 1$. So we must have $\tau(x) \subseteq a^*$, $\tau(y) \subseteq a^*$ and $\tau(c) \leqslant a^* b^* a^*$. Suppose $a^r b^n a^s \in \tau(c)$. Then $\tau(x) = \{a^{2^n - r}\}$ and $\tau(y) = \{b^{2^n - s}\}$. Hence if $x'cy' \in L_1$, then $\tau(x') = \tau(x)$ and $\tau(y') = \tau(y)$. Thus for every $c \in \Sigma_1$ with $\#\tau(c) \geqslant 2$, select some $x_c c y_c$ in L_1 (of course, x_c or y_c or both may be the empty string). Let $\Sigma_2 = \{d \mid \#\tau(d) = 1\}$. Let $L_2 = \tau(L_1 \cap \Sigma_2{}^*)$ and $L_3 = \tau(\bigcup_{c \in \Sigma_1 - \Sigma_2} x_c c y_c)$. Then $L = L_2 \cup L_3$. Now L_2 involves a regular substitution since $\tau(d)$ is regular for $d \in \Sigma_2$, and L_3 involves substitution into a regular set. A full AFL is closed under both operations. Hence $L \in \mathscr{L}_1$. Consequently, we cannot have $L \in \hat{\mathscr{S}}(\mathscr{L})$ and thus $L \in \mathscr{L}_I - \mathscr{S}^\infty(\mathscr{L})$.

[3] For a finite set R, let $\#R$ be the number of elements in R.

5. Decision Problems

In this section we shall show that if $\mathscr{L}^s(\mathscr{D})$ has a decidable emptiness problem then $\mathscr{L}^w(\mathscr{D})$ has a decidable emptiness problem, and that if in addition $\mathscr{D}$ meets a condition we shall call "uniformly erasable" then it is decidable if M is finitely nested, and if it is, the maximum degree of nesting of M can be effectively calculated. We shall see that if $\mathscr{D}$ is not uniformly erasable, then, even if $\mathscr{L}^s(\mathscr{D})$ is the family of regular sets, it may not be decidable if M in $\mathscr{D}^w$ is finitely nested.

All questions of decidability for a nested AFA must be with respect to a particular enumeration of machines. We shall proceed in the spirit of [17] and assume that we are dealing with an enumeration reflecting only the transition structure of $M(\delta$ and $F)$ and such that any manipulation of the finite state control of M can be effectively carried out by manipulations of names of automata. We assume that we are dealing with effective AFA schemata according to the following definition and all sets of states and symbols are countable and effectively enumerated.

Definition 5.1. (Γ, I, f, g) is an *effective* AFA *schema* if it is an AFA schema and Γ and I are countable and there are orderings, $\Gamma^* = \{x_0, x_1, x_2, ...\}$ and $I = \{u_0, u_1, u_2, ...\}$ such that:

(1) The function $\bar{f}$ defined as follows is partial recursive: $\bar{f}(i, j) = k$ if $f(x_i, u_j) = x_k$, and $\bar{f}(i, j)$ is undefined if $f(x_i, u_j)$ is undefined.

(2) The functions $\bar{g}_1$ and $\bar{g}_2$ defined as follows are total recursive:

$$\bar{g}_1(i) = \#g(x_i),$$

$$\bar{g}_2(i, j) = 0 \qquad \text{if} \quad x_j \in g(x_i),$$

and

$$\bar{g}_2(i, j) = 1 \qquad \text{if} \quad x_j \notin g(x_i).$$

(3) $x_1 = e$.

(4) There is a total recursive function $0(i)$ such that if $x_i \in g(x_j)$, then $\bar{f}(j, 0(i)) = j$ (i.e., $0(i)$ has the properties of u_{x_j} in the definition of an AFA schema).

(5) There are total recursive functions $\bar{h}_1(i)$ and $\bar{h}_2(i, j)$ such that if $S_i = \{x_j \mid \bar{h}_2(i, j) = 0\}$, then

(a) $S_i \subset \Gamma$,

(b) $\bar{h}_1(i) = \#S_i$,

(c) if $x \in \Gamma_1^*$, then if $f(x, u_i)$ is defined, $f(x, u_i) \in (\Gamma_1 \cup S_i)^*$ (i.e., $S_i = \Gamma_{u_i}$ in the definition of an AFA schema).

Informally, an effective AFA schema is an AFA schema in which f is partial computable (in both variables), g is computable (in the sense that we can decide the size of $g(x)$ and list its members if any) and all the sets and instructions guaranteed to exist by the definition of an AFA can be effectively found. When we say that the emptiness problem is decidable for a certain class of machines, we mean that there is a uniform procedure for deciding for each M if $L(M) = \phi$. We also assume that the input vocabulary Σ has an effective choice function for finding new symbols.

THEOREM 5.1. *If the emptiness problem is decidable for $\mathscr{D}^s$, it is decidable for $\mathscr{D}^w$.*

Proof. Let $M = (K_2, \Sigma_1, \delta, q_0, F)$. The proof of Lemma 3.1 shows that we can effectively find $M_1 = (K_2, \Sigma_1 \cup K_1 x K_1, \delta_1, q_0, F)$ and, for each $p, q \in K_1$ machines M_{pq}, all in $\mathscr{D}^s$ such that if $\tau((p, q)) = \{(p, q)\} \cup L(M_{pq})$, then $L(M) = \tau^\infty(L(M_1)) \cap \Sigma_1^*$. Then $L(M) \neq \phi$ if, and only if, there is a m such that $\tau^m(L(M_1)) \cap \Sigma_1^* \neq \phi$. We use a procedure similar to the one in Bar-Hillel, Perles and Shamir [4] for showing that the emptiness problem is decidable for context-free languages. Since $\mathscr{D}^s$ has a decidable emptiness problem, clearly we can tell for any finite Σ_2 if $\tau((p, q)) \cap \Sigma_2^* = \phi$. First, if $L(M_1) \cap \Sigma_1^* \neq \phi$, then $L(M) \neq \phi$, since $a \in \tau(a)$ for $a \in \Sigma_1$. Now if $L(M_1) \cap \Sigma_1^* = \phi$, any accepting path must open at least one new path. Then let $S_1 = \{(p, q) \mid \tau((p, q)) \cap \Sigma_1^* \neq \phi\}$. If $S_1 = \phi$ and $L(M_1) \cap \Sigma_1^* = \phi$, no strings will be accepted since all accepting paths require new nests to be opened and no new nest can ever be POPPed shut. So we assume that $S_1 \neq \phi$. For each $n \geqslant 1$, let $S_{n+1} = \{(p, q) \mid \tau((p, q)) \cap (\Sigma_1 \cup S_n)^* \neq \phi\}$. We always have $S_n \subseteq S_{n+1}$, and if $S_n = S_{n+1}$, then $S_n = S_{n+k}$ for all k. But $\#S_n \leqslant (\#K_1)^2$ for all n. Hence, there is a $k \leqslant (\#K_1)^2$ such that $S_k = S_{k+1}$. Thus, S_k contains all (p, q) such that for some w, a, q', $(p, w, e) \overset{*}{\underset{M}{\vdash}} (q', e, e)$ and $(q, \text{POP}) \in \delta(q', a, e)$. Consequently, $L(M) \neq \phi$ if and only if

$$L(M_1) \cap (\Sigma_1 \cup S_k)^* \neq \phi.$$

DEFINITION 5.2. Let $\mathscr{A} = (\Gamma, I, f, g)$ be an AFA schema. A string $x \in \Gamma^*$ is *accessible* if there are $u_1, \ldots, u_n \in I$ such that $F^n(e, u_1, \ldots, u_n) = x$. $\mathscr{A}$ is *uniformly erasable* if for each finite Γ_1 we can effectively find finite I_1 and G_1 such that if $x \in \Gamma_1^*$ is accessible, then there are $v_1, \ldots, v_m \in I_1$ and $z_1, \ldots, z_m \in G_1$ such that $F^n(x, v_1, \ldots, v_m) = e$, and $z_1 \in g(x)$, and

$$z_i \in g(F^{i-1}(x, v_1, \ldots, v_{i-1})) \qquad \text{for} \quad 2 \leqslant i \leqslant m.$$

If an AFA is uniformly erasable we can erase any string to e without changing the computing power of the AFA.

THEOREM 5.2. *If $\mathscr{D}$ is uniformly erasable and the emptiness problem is decidable for $\mathscr{D}^s$, it is decidable for any $M \in \mathscr{D}^w$ if $M \in \mathscr{D}^f$, and if $M \in \mathscr{D}^f$, what is the maximum number of nests M opens in any computation.*

Proof. Let $M = (K_1, \Sigma_1, \delta, q_0, F)$, and let $m = \# K_1$. First we note that M is finitely nested if, and only if, M is at most m-nested. For it M opens $m + 1$ nests or more, at least two successive nests must be opened in the same state. That is, for appropriate states and strings: $(q_0, w_1, e) \overset{*}{\vdash} (q, e, y_1, ..., y_r, e)$ (or else $\overset{*}{\vdash} (q, e, e)$), and $(q, w_2, e) \overset{*}{\vdash} (q, e, x_1, ..., x_s, e)$, where $s \geqslant 1$. Then for all $t \geqslant 0$, M has a computation on $w_1 w_2{}^t$ which has $r + st$ nests open at the same time, so M is not finitely nested. Thus, if we can test whether M opens at least k nests for $k = m + 1, m, ..., 2$, we can tell if M is finitely nested and if it is finitely nested, we can find the maximum number of nests.

Let $I_M = \{u \mid (q', u) \in \delta(q, a, z)$, some $q, a, z, q'\}$. For $u \in I_M$ find Γ_u as guaranteed in (5) of the definition of an effective AFA schema. Let $\Gamma_1 = \bigcup_{u \in I_M} \Gamma_u$. Note that if $(q_0, w, e) \overset{*}{\underset{M}{\vdash}} (q, e, x)$, then $x \in \Gamma_1{}^*$. Find I_1 and G_1 as specified in Definition 5.2. Now let $\bar{K}_k = (K_1 x \{1, ..., k\}) \cup \{\hat{f}\}$, where $\hat{f}$ is new; and let $M_k = (\bar{K}_k, \Sigma_1, \delta_k, (q_0, 1), \{\hat{f}\})$, where δ_k is defined as follows. Let $(q', u) \in \delta(q, a, z)$. If

(1) $u \in I$, then $((q', i), u) \in \delta_k((q, i), a, z)$, $1 \leqslant i \leqslant k$; if

(2) $u =$ POP, then $((q', i), \text{POP}) \in \delta_k((q, i + 1), a, z)$, $1 \leqslant i \leqslant k$, and if

(3) $u =$ PUSH, then $((q', i + 1), \text{PUSH}) \in \delta_k((q, i), a, z)$, $1 \leqslant i \leqslant k$, and $(\hat{f}, u_z) \in \delta_k((q, k), a, z)$.

Finally, for all z in G_1 and u in I_1, $(\hat{f}, u) \in \delta_k(\hat{f}, e, z)$ and $(\hat{f}, \text{POP}) \in \delta(\hat{f}, e, e)$. These rules insure that all nests can be erased and then POPPed. Then M opens at least $k + 1$ nests if and only if $L(M_k) \neq \phi$. Since M_k can be effectively constructed from M, we are done.

If $\mathscr{D}$ is not uniformly erasable, then even if $\mathscr{D}^s$ has a decidable emptiness problem and $\mathscr{D}$ is nontrivial, it may not be decidable whether well-nested machines are finitely nested. For let $L \subseteq \Sigma_1{}^*$ be some generator of the recursively enumerable sets. Let A, D, E, u_L be new symbols. Let $\Gamma = \Sigma_1 \cup \{A, D\}$, $I = \Sigma_1 \cup \{A, E, e, u_L\}$, $g(xa) = \{a\}$ for $a \in \Sigma_1$, $g(e) = \{e\}$, $g(A) = \{A\}$, $g(D) = \{D\}$ and let g be undefined elsewhere. Let $f(x, a) = xa$ for $a \in \Sigma_1$, $f(x, e) = x$ for all x, $f(e, A) = A$, $f(A, E) = e$, and $f(x, u_L) = D$ if $x \in L$; let f be undefined elsewhere. The instructions in Σ_1 allow a word in $\Sigma_1{}^*$ to be written on the storage. The instruction u_L is an "oracle" for L. It causes the machine to hang up if the storage is not a word in L. Otherwise the storage is erased and the single symbol D is written. But storage D can never be changed and in particular can never be erased. The instructions A and e are the minimal instructions necessary to give us a nontrivial AFA schema.

Only the strings A and e can be erased; any other storage contents will never lead to acceptance. Hence, to decide if $L(M) = \phi$, we can eliminate all instructions other than e, A or E; $L(M)$ is regular and we can use standard methods to decide the emptiness problem.

Note that for every recursively enumerable set S there is a $M \in \mathscr{D}^s$ such that $S = L'(M) = \{w \mid \exists \text{ final state } f, (q_0, w, e) \overset{*}{\vdash} (f, e, D)\}$. Thus, if we add instructions to make $\mathscr{D}$ uniformly erasable, we would get all recursively enumerable sets; similarly if we allowed nonempty nests to be POPPed. We can tack on to any $M \in \mathscr{D}^s$ instructions which from an instantaneous description (f, e, D) start PUSHing open arbitrarily many new nests. Then the new machine, M', will be finitely nested if, and only if, $L'(M) = \phi$, and the latter is undecidable. Similarly, for any k we can find M_k such that M_k has at most k nests if, and only if, $L'(M) = \phi$.

Thus we see that, not surprisingly, different AFA can generate the same family of languages and yet have different decision properties.

Definition 5.3. An AFA schema $\mathfrak{A}$ is a *representation* for $\mathscr{L}$ if $\mathscr{L} = \mathscr{L}^s(\mathscr{D})$, where $(\mathfrak{A}, \mathscr{D})$ is the corresponding AFA. It is an *effective representation* if $\mathfrak{A}$ is an effective AFA schema.

Definition 5.4. A property P defined on $\mathscr{L}$ is *decidable* for $\mathscr{L}$ if there is an effective representation $\mathfrak{A}$ for $\mathscr{L}$ with corresponding AFA $(\mathfrak{A}, \mathscr{D})$ such that it is decidable for $M \in \mathscr{D}^s$ whether $L(M)$ has property P.

Using Definition 5.4 we can rephrase Theorem 5.1:

Theorem 5.3. *If the emptiness problem is decidable for a full* AFL $\mathscr{L}$, *it is decidable for* $\mathscr{S}^\infty(\mathscr{L})$.

Proof. Let $\mathfrak{A}$ be an effective representation of $\mathscr{L}$ with a decidable emptiness problem. Then the emptiness problem is decidable for $\mathscr{D}^w$. We can use the construction in Lemma 3.1 to find an effective AFA $(\mathfrak{A}_1, \mathscr{D}_1)$ with $\mathscr{L}^s(\mathscr{D}_1) = \mathscr{L}^w(\mathscr{D})$; from the construction it is evident that $\mathscr{D}_1{}^s$ has a decidable emptiness problem if $\mathscr{D}^w$ does. But $\mathscr{S}^\infty(\mathscr{L}) = \mathscr{S}^\infty(\mathscr{L}^s(\mathscr{D})) = \mathscr{L}^w(\mathscr{D}) = \mathscr{L}^s(\mathscr{D}_1)$.

We note that since every superAFL contains all context-free languages, we have the usual theorems on undecidable properties.

THEOREM 5.4. *For any full* AFL $\mathscr{L}$:

(1) *If* $\mathscr{L}$ *is not substitution closed, then for* $L \in \hat{\mathscr{S}}^{\infty}(\mathscr{L})$ *it is undecidable if* $L \in \hat{\mathscr{S}}(\mathscr{L})$.

(2) *If* $\mathscr{L}$ *is not a super*AFL, *then for* $L \in \hat{\mathscr{S}}^{\infty}(\mathscr{L})$ *it is undecidable if* $L \in \mathscr{L}$.

(3) *It is undecidable if a member of* $\hat{\mathscr{S}}^{\infty}(\mathscr{L})$ *is regular.*

6. REPRESENTATIONS AND INVARIANT OPERATIONS

We have noticed that some of the results of the previous sections hold for any nontrivial representation of a given family of languages whereas others depend heavily on properties of a particular representation. This brings up the question of which properties are independent of the representation used, and in particular raises the question of invariance of operations.

We can define effective closure under operations in the spirt of [11].

DEFINITION 6.1. Let $(\mathscr{O}, \mathscr{D})$ be an effective AFA. Let $0(L_1, ..., L_n)$ be an operation on languages. We say that $\mathscr{D}^s$ ($\mathscr{D}^f$, $\mathscr{D}^w$, $\mathscr{D}$, respectively) is *effectively closed under operation* 0 if there is a uniform, recursive procedure to find for each $M_1, ..., M_n \in \mathscr{D}^s$ ($\mathscr{D}^f, \mathscr{D}^w, \mathscr{D}$, respectively) a $\bar{M}$ in $\mathscr{D}^s$ ($\mathscr{D}^f, \mathscr{D}^w, \mathscr{D}$, respectively) such that $L(\bar{M}) = 0(L(M_1)), ..., L(M_n))$.

The import of our previous theorems is that for *any* nontrivial effective AFA $(\mathscr{O}, \mathscr{D})$, $\mathscr{D}^s$, $\mathscr{D}^f$, $\mathscr{D}^w$ and $\mathscr{D}$ are effectively closed under the six defining operations for full AFLs; that $\mathscr{D}^f$, $\mathscr{D}^w$ and $\mathscr{D}$ are effectively closed under substitution; and that $\mathscr{D}^w$ and $\mathscr{D}$ are effectively closed under nested iterated substitution (the proofs for $\mathscr{D}^f$ were not constructive as given, but alternative constructions could be supplied). This suggests that substitution and nested iterated substitution are in some sense AFA invariant operations, just as the six defining operations are; that is, there is a manipulation on AFA (such as obtaining $\mathscr{D}^w$ from $\mathscr{D}^s$) which applied to any effective representation not only yields the desired closure property but is such that the resulting family of machines is effectively closed under the operation or class of operations under consideration (strictly speaking, substitution is a class of operations, not a single operation).

It seems an interesting and important open problem to formulate a precise definition of "AFA invariant" and study which operations or classes of operations are AFA invariant in the way that substitution and nested iterated substitution (and intersection; see [15]) are AFA invariant. An example of an operation which does not seem to be AFA invariant is reversal.[4] We can extend the definition of erasable as follows.

DEFINITION 6.2. An AFA schema (Γ, I, f, g) is *uniformly reversible* if there is a uniform effective procedure that yields for all $u_0 \in I$, and $z \in g(\Gamma^*)$, instructions $u_1, \ldots, u_n \in I$ and strings $z_1, \ldots, z_n \in g(\Gamma^*)$, such that if x is accessible, $z \in g(x)$ and $f(x, u_0)$ is defined, then $F^{n+1}(x, u_0, u_1, \ldots, u_n) = x$ and for $0 \leqslant i \leqslant n$, $z_{i+1} \in g(F^{i+1}(x, u_0, u_1, \ldots, u_n))$.

Thus an AFA is uniformly reversible if not only can each accessible string be erased but it can be erased in a step by step manner. Clearly a uniformly reversible AFA is effectively closed under reversal since we can in effect run the machines backward; that is, let the initial state be final, the final states reached in one step from a new initial state and each rule (q', u) in $\delta(q, a, z)$ replaced by a series of rules going from q' to q reversing the effect of u on any x with $z \in g(x)$; we can effectively find such a finite series for any u, z.[5]

But we have seen that even the regular sets have effective AFA representations that are not erasable, let alone reversible, and cannot be made reversible by adding suitable instructions that do not affect computational power. Further some full AFLs with decidable emptiness problems closed under reversal cannot have *any* uniformly reversible representation. For example, let $\mathscr{L}_1 = \hat{\mathscr{F}}(\{a^n b^m \mid n > m > 1\})$ and $\mathscr{L}_2 = \hat{\mathscr{F}}(\{a^n b^m \mid m > n > 1\})$. Neither is closed under reversal [6] but the reversals of members of the one lie in the other. Thus, $\mathscr{L} = \hat{\mathscr{F}}(\mathscr{L}_1 \cup \mathscr{L}_2)$ is closed under reversal; $\mathscr{L}$ is properly contained in $\hat{\mathscr{F}}(\{a^n b^n \mid n \geqslant 1\})$. No representation for $\mathscr{L}$ can be uniformly reversible since any representation must have instructions that allow it to "count"; if these can be uniformly reversed, step by step, a machine could not only tell if $n > m$ or $n < m$, but also if $n = m$ and so define $\{a^n b^n \mid n \geqslant 1\}$.

There are effective representations for the regular sets that are so pathological as not to be effectively closed under reversal. Given a Turing machine M with an undecidable halting problem and input vocabulary 1^*, we can find a series of recursive sets $L_1, \ldots, L_n, \ldots$ (i.e., there is a Turing machine which decides for any w and integer i if w is in L_i) such that each L_i is empty

[4] The reversal of a word is the word read backwards; the reversal of a set of words is the set of reversals of its words.

[5] Ginsburg and Harrison have a similar result for a different condition [9].

or contains an unsymmetric encoding of a halted computation for 1^{i+1}, and $L_i \neq \phi$ if and only if M halts on 1^{i+1}. The L_i are regular sets, so we can add to any representation of the regular sets oracles for L_i, i.e., instructions u_i with $f(x, u_i) = e$ if $x \in L_i$ and $f(x, u_i)$ undefined otherwise; the new AFA will still define only regular sets. But now we have an effective AFA with an undecidable emptiness problem and not effectively closed under reversal (essentially because there is no uniform way of knowing which regular set is equal to L_i). Thus, we conjecture that reversal is not an AFA invariant operation.

Since Aho and Ullman have shown that the indexed languages can be obtained from the context-free languages by nesting the standard *pda* representation for the context-free languages, and since the work of Aho [1] and [2] and Fischer [5] seem to indicate that the indexed languages rather than the stack languages are the natural next extension of the context-free languages (they have a convenient grammar, the OI Macro grammar [5] and like the context-free languages are a superAFL though not the least superAFL properly containing the context-free languages), it would be desirable to have an operation or class of operations which together with the AFL operations yield $\mathscr{L}(\mathscr{D})$ from $\mathscr{L}^s(\mathscr{D})$ for any representation. This is not the case.

By nesting a suitably selected effective representation of the context-free languages, with a decidable emptiness problem, we can obtain the recursively enumerable sets. We can clearly find an effective enumeration of the context-free languages, $L_1, L_2, \ldots, L_n, \ldots$ such that not only the membership but also the emptiness problem is decidable for that enumeration. Now if we add oracles for the L_i (again $f(x, u_i) = e$ if $x \in L_i$, undefined otherwise) to any standard pda representation of the context-free languages we obtain a new representation for the context-free languages whose emptiness problem is still decidable. Call this representation $(\mathscr{O}\!\ell, \mathscr{D})$. Now $\mathscr{L}(\mathscr{D})$ does not have a decidable emptiness problem. It is easy to see that $\mathscr{L}(\mathscr{D})$ will contain the intersection of any two context-free languages (to see if $x \in L_i X L_j$, write down x, duplicate and use first an oracle for L_i and then one for L_j) and so all recursively enumerable sets [8].

Thus, we see that the decidability of the emptiness problem for $\mathscr{D}^s$ does not imply the corresponding result for $\mathscr{D}$, even if $\mathscr{D}$ is uniformly erasable. Furthermore the study of nested AFA cannot solve the open question of finding AFA-invariant operations leading from the context-free to the indexed languages in the way nested iterated substitution leads from the regular sets to the context-free languages. Of course, if $\mathscr{L}_1$ and $\mathscr{L}_2$ are any full AFLs with $\mathscr{L}_1 \subseteq \mathscr{L}_2$ and $\mathscr{L}_2$ full principal and L_2 is any generator of $\mathscr{L}_2$, if we define $0(L) = L \cup L_2$, then $\mathscr{L}_2$ is the closure of $\mathscr{L}_1$ under 0 and the AFL operations and a representation for $\mathscr{L}_2$ can always be obtained from one for $\mathscr{L}_1$ by adding an oracle for L_2. We should wish our definition of AFA-invariant and operation closure to exclude such trivial cases.

RECEIVED: May 28, 1969

REFERENCES

1. A. V. AHO, Indexed grammars—an extension of context-free grammars, *JACM* **15** No. 4 (1968), 647–671.
2. A. V. AHO, Nested stack automata, *JACM* **16** No. 3 (1969), 383–406.
3. A. V. AHO AND J. ULLMAN, private communication.
4. Y. BAR-HILLEL, M. PERLES, AND E. SHAMIR, On formal properties of simple phrase structure grammars, *Z. Phonetik Sprachwiss Kummunikat* **14** (1961), 143–172.
5. M. J. FISCHER, "Grammars With Macro-Like Productions," Ph.D. Dissertation, Harvard University, May 1968.
6. S. GINSBURG AND S. GREIBACH, Abstract families of languages, (to appear in a *Memoir* of the American Mathematical Society).
7. S. GINSBURG AND S. GREIBACH, "Principal AFL," SDC document TM-738/051/00, April 1969.
8. S. GINSBURG, S. GREIBACH, AND M. HARRISON, One-way stack automata, *JACM* **14** (1967), 389–418.
9. S. GINSBURG AND M. HARRISON, private communication.
10. S. GINSBURG AND E. H. SPANIER, "Substitutions in Families of Languages," SDC document TM-738/049/00, September 1968.
11. S. GREIBACH, A note on undecidable properties of formal languages, *Mathematical Systems Theory* **2**, No. 1 (1968), 1–6.
12. S. GREIBACH, An infinite hierarchy of context-free languages, *JACM* **16**, No. 1 (1969), 91–106.
13. S. GREIBACH, Checking automata and one-way stack languages, *JCSS* **3**, No. 2 (1969), 196–217.
14. S. GREIBACH, "Chains of Full AFLs," SDC document TM-738/053/00.
15. S. GREIBACH AND S. GINSBURG, "Multitape AFA," SDC document TM-738/050/00, January 1969.
16. S. GREIBACH AND J. HOPCROFT, Independence of AFL operations, to appear in a *Memoir* of the American Mathematical Society.
17. J. HOPCROFT AND J. ULLMAN, Decidable and undecidable questions about automata, *JACM* **15**, No. 2 (1968), 317–324.
18. J. KRÂL, "A Modification of Substitution Theorem and Some Necessary and Sufficient Conditions for Sets to be Context-free," unpublished manuscript.
19. M. NIVAT, "Transductions des languages de Chomsky," unpublished thesis, Grenoble University, 1967.
20. W. OGDEN, "Intercalation Theorems for Stack Languages," Conference Record, ACM Symposium on Theory of Computing, May 1969, Marina del Rey, California, pp. 31–42.
21. M. SCHKOLNICK, "Two-Type Bracketed Grammars," IEEE 9th Annual Symposium on Switching and Automata Theory, October 1968, pp. 315–326.
22. M. K. YNTEMA, Inclusion relations among families of context-free languages, *Information and Control* **10** (1967), 572–597.