Theoretical Computer Science 6 (1978) 175–221.
© North-Holland Publishing Company

ONE WAY FINITE VISIT AUTOMATA*

S.A. GREIBACH
Department of System Science, University of California, Los Angeles, CA 90024, U.S.A.

Communicated by Ronald Book
Received October 1976
Revised July 1977

Abstract. A one-way preset Turing machine with base L is a nondeterministic on-line Turing machine with one working tape preset to a member of L. FINITEREVERSAL($\mathscr{L}$) (FINITEVISIT($\mathscr{L}$)) is the class of languages accepted by one-way preset Turing machines with bases in $\mathscr{L}$ which are limited to a finite number of reversals (visits). For any full semiAFL $\mathscr{L}$, FINITEREVERSAL($\mathscr{L}$) is the closure of $\mathscr{L}$ under homomorphic replication or, equivalently, the closure of $\mathscr{L}$ under iteration of controls on linear context-free grammars while FINITEVISIT($\mathscr{L}$) is the result of applying controls from $\mathscr{L}$ to absolutely parallel grammars or, equivalently, the closure of $\mathscr{L}$ under deterministic two-way finite state transductions. If $\mathscr{L}$ is a full AFL with $\mathscr{L} \neq$ FINITEVISIT($\mathscr{L}$), then FINITEREVERSAL($\mathscr{L}$) $\neq$ FINITEVISIT($\mathscr{L}$). In particular, for one-way checking automata, $k+1$ reversals are more powerful than k reversals, $k+1$ visits are more powerful than k visits, k visits and $k+1$ reversals are incomparable and there are languages definable within 3 visits but no finite number of reversals. Finite visit one-way checking automaton languages can be accepted by nondeterministic multitape Turing machines in space $\log_2 n$. Results on finite visit checking automata provide another proof that not all context-free languages can be accepted by one-way nonerasing stack automata.

1. Introduction

One of the complexity measures studied for both general Turing machines and particular types of machines such as checking automata and pushdown store machines is the so-called reversal complexity [1, 7, 14, 18, 21, 27, 28]. There are different measures which are sometimes lumped together under this term—the number of times the working tape head changes direction, the number of times the boundary between working tape squares is crossed or the number of visits to a square. For nondeterministic machines with more than one working tape (not counting the input tape), the question is irrelevant unless time restrictions are imposed (see [33]) because even restriction to two reversals per tape yields the full power of a Turing machine [1]. It was shown in [20] that for nondeterministic machines with a two-way input tape and one working tape and any bounding function $T(n)$, a bound of $T(n)$ on reversals, crosses or visits yields the same power

* This paper was supported in part by the National Science Foundation under Grant No. DCR 74-15091.

up to a linear factor—the power of a $T(n)\log_2 n$ space bounded multitape Turing machine.

In this paper we concentrate on studying the application of a finite bound on reversals, crosses or visits to machines with a one-way input tape and one working tape. The model we use is a generalization of a checking automaton [16] called a preset Turing machine. A (one-way) preset Turing machine with a base language L is a machine with a one-way input tape and a working tape whose initial contents are "preset" to any member of L. If L is regular the machine is regular-based, while if L belongs to a family of languages $\mathscr{L}$, the machine is $\mathscr{L}$-based. The machine is nonwriting if it is not allowed to write on its working tape. Clearly a nonwriting regular-based machine can be regarded as a checking automaton. We always assume that the base $\mathscr{L}$ is a full semiAFL.[1]

If a preset Turing machine is restricted to a finite number of reversals, crosses or visits, we call it a finite reversal, finite cross or finite visit automaton. Here we show that finite cross and finite visit automata are equivalent in power but finite reversal automata are less powerful. In both cases, the writing and nonwriting variants are equivalent in power.

In Section 2 we give our basic definitions and study the relationship among reversals, crosses and visits and writing and nonwriting machines. We let k-REVERSAL($\mathscr{L}$) be the family of languages accepted by $\mathscr{L}$-based k-reversal bounded machines and

$$\text{FINITEREVERSAL}(\mathscr{L}) = \bigcup_k k\text{-REVERSAL}(\mathscr{L});$$

the other families are defined similarly. We show that k-REVERSAL($\mathscr{L}$) is equal to the family of languages accepted by $\mathscr{L}$-based k-reversal bounded nonwriting machines and k-VISIT($\mathscr{L}$) = k-CROSS($\mathscr{L}$) is equal to the family of languages accepted by $\mathscr{L}$-based k-visit bounded nonwriting machines. Thus in particular k-reversal or k-visit bounded on-line one tape Turing machines are equivalent to one-way checking automata with the same bound.

Section 3 concentrates on FINITEREVERSAL($\mathscr{L}$). We give two characterizations of FINITEREVERSAL($\mathscr{L}$) which show that it is (as one would expect) an old friend, the closure of $\mathscr{L}$ under homomorphic replication (cf. [11, 19]) and, equivalently, the closure of $\mathscr{L}$ under iteration of controls on linear context-free grammars (cf. [17]). These characterizations yield the decomposition theorem

$$k\text{-REVERSAL}(r\text{-REVERSAL}(\mathscr{L})) = kr\text{-REVERSAL}(\mathscr{L})$$

which serves as a "padding" or "translational" lemma for hierarchy results. We conclude the section with the general hierarchy theorem that whenever $\mathscr{L} \neq$ FINITEREVERSAL($\mathscr{L}$) then $k+1$ reversals are strictly more powerful than k reversals. The same arguments show that FINITEREVERSAL($\mathscr{L}$) $\neq$ FINITE-VISIT($\mathscr{L}$) for any full AFL $\mathscr{L}$ such that $\mathscr{L} \neq$ FINITEVISIT($\mathscr{L}$). So in particular,

[1] Formal definitions of full semiAFL's and AFL's appear in Section 2.

finite visits on a one-way checking automaton yield more than finite reversals, unlike the situation in the two-way case.

In Section 4 we concentrate on finite visit automata. We first show that FINITEVISIT($\mathscr{L}$) is an idempotent operator on $\mathscr{L}$ by demonstrating that

$$k\text{-VISIT }(r\text{-VISIT}(\mathscr{L})) \subseteq kr\text{-VISIT}(\mathscr{L}).$$

Next we establish a grammatical characterization of FINITEVISIT($\mathscr{L}$) in terms of control sets on absolutely parallel grammars [25] as well as an operator characterization, and as a consequence show that if $\mathscr{L}$ has the Parikh or semilinear property [23] then so does FINITEVISIT($\mathscr{L}$). Finally we give a strong hierarchy theorem for finite visit checking automata—$k+1$ visits are strictly more powerful than k—and two weak general hierarchy theorems for FINITEVISIT($\mathscr{L}$). Application of these techniques gives an alternative proof that not all context-free languages are one-way nonerasing stack languages.

In Section 5 we give various complexity results for FINITEVISIT(REGL) and FINITEVISIT(CF). Some are drawn directly from the two-way case—e.g., languages in FINITEVISIT(REGL) can be accepted nondeterministically in space $\log_2 n$ and languages in FINITEVISIT(CF) deterministically in polynomial time. Others use the fact that one-way finite visit automata can be made to accept in linear time and space; thus, e.g., languages in FINITEVISIT(CF) can be accepted nondeterministically in realtime. These ideas can be combined with the characterization of the two-way case via multihead automata, to show that languages in FINITEVISIT(CF) can be accepted in polynomial time by nondeterministic multihead pushdown store automata and hence deterministically by Turing machines in space $(\log_2 n)^2$.

Section 6 summarizes the results and gives some open problems.

2. Visits, crosses and reversals

In this section we present our formal definitions and establish some of the basic connections among finite visit, finite reversal, and finite cross automata. Since we are interested here only in the one-way case, we modify slightly the definitions in [20] to make them more convenient for our present purposes.

First we define one-way (on-line) preset Turing machines and their computations.

Definition 2.1. A *one-way preset Turing machine* is a tuple $M = (K, \Sigma, \Gamma, \delta, q_0, F, L)$ where K is a finite set of *states*, Σ is a finite *input* alphabet, Γ is a finite *working* alphabet, q_0 in K is the *initial* state, $F \subseteq K$ is the set of *final* or *accepting* states, L is

[2] For a set of strings S, S^* is the monoid generated by S under concatenation with identity λ, the empty string, and $S^+ = SS^*$.

a language contained in Γ^+ and called the *base* of M and the *transition* function δ maps $K\times(\Sigma\cup\{e\})\times\Gamma$ into the subsets of $K\times\Gamma\times\{0,1,-1\}$.[2] A transition (q', B, j) in $\delta(q, e, A)$ is called an *e-rule* or *e-transition* and if M has no such transitions it is *e-free*. A transition (q', B, j) in $\delta(q, a, A)$ with $A\neq B$ is a *write* and if M has no such transitions it is *nonwriting*. A transition $(q', B, 0)$ in $\delta(q, a, A)$ is a *standstill* transition and if M has no such transitions it is *nonresting*. If for each q in K and A in Γ either $\#\delta(q, e, A)\leqslant 1$ and $\delta(q, a, A)=\emptyset$ for every a in Σ or $\delta(q, e, A)=\emptyset$ and $\#\delta(q, a, A)\leqslant 1$ for every a in Σ, then M is *deterministic*.[3]

Informally, (q', B, r) in $\delta(q, a, A)$ means that for state q, working tape symbol A and input a, M can change state to q', overprint A with B, advance the input for $a\neq e$, and move the working tape head in direction r, where as usual 0 means no move, 1 means right and -1 means left. During an e-rule, M neither consults nor advances the input tape. Computations start in the initial state with the input head on the leftmost symbol and working tape set to some string in the base with the head on the leftmost square.

Acceptance means entry into a final state with the input head falling off the right of its tape and the working head falling off either the right or the left of its tape. This is formalized in the next definition.

Definition 2.2. An *instantaneous description* (ID) of one-way preset Turing machine $M=(K, \Sigma, \Gamma, \delta, q_0, F, L)$ is a tuple (q, w, y, i) with $q\in K$, $w\in\Sigma^*$, $y\in\Gamma^+$, $0\leqslant i\leqslant|y|+1$.[4]. It is *initial* if $q=q_0$, $y\in L$ and $i=1$ and *accepting* if $q\in F$, $w=e$ and either $i=0$ (*left exiting*) or $i=|y|+1$ (*right exiting*). If $(q', B, j)\in\delta(q, a, A)$ and $w=aw'$, $a\in\Sigma\cup\{e\}$, $y=y'Ay''$, $i=|y'A|$, and $0\leqslant i+j\leqslant|y|+1$, then we write

$$(q, w, y, i)\vdash(q', w', y'By'', i+j).$$

We let $\overset{*}{\vdash}$ be the transitive reflexive closure of the relation $\vdash$. We call a sequence of relations among ID's $I_0\vdash I_1\vdash\cdots\vdash I_n$ a *computation for input w with working tape y* if $I_0=(q_0, w, y, 1)$; if I_n is accepting, it is an *accepting* computation, *left exiting* or *right exiting* as I_n is. The *language accepted by M* is

$$\mathrm{L}(M)=\{w\in\Sigma^*\mid\exists\text{ accepting computation for } w \text{ with working tape } y \text{ in } \mathrm{L}\}$$

Now we must define the number of visits, reversals and crosses of a computation as well as some useful additional technical terms.

[3] For a finite set S, $\#S$ is the number of members of S.

[4] For a word w, $|w|$ is the length of w.

Definition 2.3. Let $I_0 \vdash I_1 \vdash \cdots \vdash I_n$ be a computation in machine M with each $I_i = (q_i, w_i, y_i, s_i)$ and $m = |y_0| = |y_1| = \cdots = |y_n|$. For $1 \leq j \leq m$, the number of *visits* to square j is

$$\#\{i \mid s_i = j\},$$

while the number of *crosses* of boundary j is

$$\#\{i \mid s_i = j, s_{i+1} = j+1\} + \#\{i \mid s_i = j+1, s_{i+1} = j\}.$$

The computation is *k-visit bounded* if no square is visited more than k times and *strictly k-visit* if every square is visited exactly k times; it is *k-crossing bounded* if no boundary is crossed more than k times and *strictly k-crossing* if every boundary j for $1 \leq j \leq m-1$ is crossed exactly k times. A *reversal* occurs at I_i if $s_i \neq s_{i+1}$ and there is a $j \leq i-1$ such that $s_u = s_i$ for $j+1 \leq u \leq i$ and $s_{i+1} = s_j$. The number of reversals during the computation is

$$1 + \#\{i \mid \text{a reversal occurs at } I_i\}.$$

If this number is less than or equal to k, the computation is *k-reversal bounded.* A *bounce* occurs at I_i if $s_i = s_{i+2}$ but $s_{i+1} \neq s_i$; if there are no bounces then the computation is *bounce-free.* The computation is *full sweep* if whenever a reversal occurs at I_i, $s_i = 1$ or $s_i = m$. The computation is *right touching* if for some i, $s_i = m$.

The number of reversals is set in Definition 2.3 at 1 more than the number of times the working head changes its direction: this is done so that we count sweeps through the working tape, a count more compatible with visits and crosses. A full sweep computation reverses the working head only at the ends of the working tape. A right touching computation must visit the rightmost working tape square and hence must visit each square at least once. A bounce-free computation does not shuttle back and forth between adjacent squares which could make the number of crosses larger than the number of visits.

Definition 2.4. A preset Turing machine M is *k-visit bounded* (respectively, *k-crossing bounded, k-reversal bounded*) if for each w in L(M) there is a *k-visit bounded* (respectively, *k-crossing bounded, k-reversal bounded*) accepting computation for w. It is *strictly k-visit* (respectively, *strictly k-crossing, strictly k-reversal bounded*) if every computation is k-visit bounded (respectively, k-crossing bounded, k-reversal bounded) and every accepting computation is strictly k-visit (respectively, strictly k-crossing, has reversal number k). If for any $k \geq 1$, M is k-visit bounded (respectively, k-crossing bounded, k-reversal bounded) then M is a *finite visit automaton,* (respectively, *finite crossing automaton, finite reversal automaton*) abbreviated *fva* (respectively, *fca, fra*). If every accepting computation of M is right touching (respectively, bounce-free, full sweep), then M is *right touching* (respectively, *bounce-free, full sweep*).

Now we can define the classes of languages accepted by these machines. The classification specifies the bound involved and the family of languages to which the base language belongs. That is. if L is the base of preset Turing machine M, then we call M "L-based" and if L belongs to a family of languages $\mathscr{L}$, we extend the notation to call M "$\mathscr{L}$-based". All our machines below are assumed to be one-way.

Definition 2.5. For any family of languages $\mathscr{L}$ and integer $k \geq 1$, let
k-VISIT($\mathscr{L}$) = {L(M) | M is $\mathscr{L}$-based and k-visit bounded},
k-CROSS($\mathscr{L}$) = {L(M) | M is $\mathscr{L}$-based and k-crossing bounded}, and
k-REVERSAL($\mathscr{L}$) = {L(M) | M is $\mathscr{L}$-based and k-reversal bounded}.

When we take the union over all finite bounds k, we use the designation FINITE.

Definition 2.6. For any family of languages $\mathscr{L}$,
FINITEVISIT($\mathscr{L}$) = {L(M) | M is an $\mathscr{L}$-based fva},
FINITECROSS($\mathscr{L}$) = {L(M) | M is an $\mathscr{L}$-based fca}, and
FINITEREVERSAL($\mathscr{L}$) = {L(M) | M is an $\mathscr{L}$-based fra}.

Sometimes one also wants to restrict attention to nonwriting or deterministic machines. This is done by adding NW or DET to the family name. Thus k-DETVISIT($\mathscr{L}$) is the class of languages accepted by k-visit bounded $\mathscr{L}$-based deterministic machines; k-NWREVERSAL($\mathscr{L}$) is the family of languages accepted by k-reversal bounded $\mathscr{L}$-based nonwriting machines while DETNWFINITE-CROSS($\mathscr{L}$) is the class of languages accepted by $\mathscr{L}$-based deterministic nonwriting finite crossing automata.

We shall also have occasion to regard a preset Turing machine as an operator on its base and thus families like k-VISIT($\mathscr{L}$) as operators on $\mathscr{L}$. We shall indicate iteration of this operator by a subscript. So

$$k\text{-VISIT}_0(\mathscr{L}) = \mathscr{L},$$

$$k\text{-VISIT}_1(\mathscr{L}) = k\text{-VISIT}(\mathscr{L})$$

and for $n \geq 1$,

$$k\text{-VISIT}_{n+1}(\mathscr{L}) = k\text{-VISIT}(k\text{-VISIT}_n(\mathscr{L})).$$

We define k-REVERSAL$_n$($\mathscr{L}$) and k-CROSS$_n$($\mathscr{L}$) similarly.

We will be particularly interested in the regular-based machines. A regular-based machine is equivalent in power to an on-line one working tape Turing machine; the simulations involved preserve the number of visits, crosses or reversals [20]. Similarly, a one-way checking automaton (see [16] for definition of a checking automaton) can be simulated by a one-way nonerasing regular-based machine without affecting the number of visits, crosses or reversals. The obvious simulation of a nonwriting regular-based machine by a checking automaton appears to increase the number of visits or crosses by one, but can be modified to preserve the number of reversals. Further discussion appears in [20]. Thus results for k-visit or k-reversal regular-based machines translate into results for k-visit or k-reversal on-line (one working tape) Turing machines while those for corresponding nonwriting machines translate into facts about checking automata. Notice that k-reversal bounded checking automata in this sense are strictly more powerful than those of Siromoney [28] which only sweep from left-to-right on the working tape.

We shall usually restrict our families of base languages to those having certain closure properties.

Definition 2.7. A *full semiAFL* is a family of languages containing at least one nonempty e-free language and closed under union, homomorphism and intersection with regular sets[5]; a full semiAFL closed under Kleene + is a *full AFL.*[6] For a family of languages $\mathcal{L}$, the least full semiAFL (full AFL) containing $\mathcal{L}$ is designated $\hat{\mathcal{M}}(\mathcal{L})$ ($\hat{\mathcal{F}}(\mathcal{L})$); if $\mathcal{L} = \{L\}$, we write $\hat{\mathcal{M}}(L)$ ($\hat{\mathcal{F}}(L)$) and call it a *full principal semiAFL* (*full principal AFL*).

Two additional classes of operations we shall have occasion to use are the a-transductions and the substitutions.

Definition 2.8. An *a-transducer* is a tuple $M = (K, \Sigma, \Delta, H, q_0, F)$ where K is a finite set of states, $q_0 \in K, F \subseteq K, \Sigma$ is a finite input alphabet, Δ is a finite output alphabet and H is a finite subset of $K \times \Sigma^* \times \Delta^* \times K$. An ID of M is any member of $K \times \Sigma^* \times \Delta^*$. If (q, uw, y) is an ID and $(q, u, v, q') \in H$, then we write $(q, uw, y) \vdash (q', w, yv)$. The relation $\overset{*}{\vdash}$ among ID's is the transitive reflexive extension of $\vdash$. For $w \in \Sigma^*$,

$$M(w) = \{v \mid \exists q \in F, (q_0, w, e) \overset{*}{\vdash} (q, e, v)\},$$

and for a language L,

$$M(L) = \{v \mid \exists w \in L, v \in M(w)\}.$$

If for each (q, u, v, q') in H, $|u| \geq |v|$, then M is *nonincreasing*. If $H \subseteq K \times \Sigma \times \Delta^* \times K, F = K$, and for each $q \in K, a \in \Sigma$, there is eactly one (q, a, v, q') in H, then M is a *general sequential machine* (*gsm*).

Intuitively, an a-transducer is a nondeterministic one-way finite state transducer with accepting states; output is "legal" only when the machine is in an accepting state.

[5] A language is *e-free* if it does not contain the empty string e.
[6] The operation taking L into L^+ is called *Kleene* +.

We shall use the fact that every full semiAFL is closed under a-transducer mapping and, more strongly, is characterized by union and a-transducer mapping [8, 10]. That is, for any family of languages $\mathscr{L}$, containing at least one nonempty language,

$$\hat{\mathscr{M}}(\mathscr{L})=\{M_1(L_1)\cup\cdots\cup M_n(L_n)\mid L_1,\ldots,L_n\in\mathscr{L},\ M_1,\ldots,M_n \text{ are } a\text{-transducers}\}$$

and for a language L,

$$\hat{\mathscr{M}}(L)=\{M(L)\mid M \text{ is an } a\text{-transducer}\}.$$

Properties of semiAFL's and AFL's can be found in [8, 10]; a-transducers are also described in [6].

Definition 2.9. A *substitution* τ on a finite alphabet Σ takes each a in Σ into a language $\tau(a)$. We extend τ to words by

$$\tau(e)=\{e\}$$

and

$$\tau(xy)=\tau(x)\tau(y)$$

and to languages L by

$$\tau(L)=\{u\mid \exists w\in L,\ u\in\tau(w)\}.$$

If $\tau(a)$ is in $\mathscr{L}$ for each a in Σ, then τ is an *$\mathscr{L}$ substitution*. For families of languages $\mathscr{L}_1$ and $\mathscr{L}_2$ the family of languages obtained by substituting members of $\mathscr{L}_2$ into $\mathscr{L}_1$ is

$$\mathscr{L}_1\hat{\sigma}\mathscr{L}_2=\{\tau(L)\mid L\in\mathscr{L}_1,\ \tau \text{ is an } \mathscr{L}_2\text{-substitution}\}.$$

If $\mathscr{L}\hat{\sigma}\mathscr{L}_1\subseteq\mathscr{L}$, then $\mathscr{L}$ is *closed under substitution by* $\mathscr{L}_1$, and if $\mathscr{L}_1\hat{\sigma}\mathscr{L}\subseteq\mathscr{L}$, then $\mathscr{L}$ is closed under *substitution into* $\mathscr{L}_1$. If $\mathscr{L}\hat{\sigma}\mathscr{L}\subseteq\mathscr{L}$, then $\mathscr{L}$ is *closed under substitution.*

Full semiAFL's are closed under regular substitution (substitution by regular languages) while full AFL's are also closed under substitution into regular sets [8, 10].

Standard arguments show that our families reflect properties of the base family and also possess a few more. The proofs follow lines in [10, 16, 18] and will be omitted.

Theorem 2.10. *For any $k\geq 1$ and any full semi*AFL $\mathscr{L}$,

(1) k-REVERSAL($\mathscr{L}$) *and* k-VISIT($\mathscr{L}$) *are full semi*AFL'*s*,

(2) FINITEREVERSAL($\mathscr{L}$) *and* FINITEVISIT($\mathscr{L}$) *are full semi*AFL'*s and are closed under concatenation if $\mathscr{L}$ is so closed, and*

(3) *if $\mathscr{L}$ is a full* AFL *so is* FINITEVISIT($\mathscr{L}$) *and if $\mathscr{L}$ is closed under substitution so is* FINITEVISIT($\mathscr{L}$).

Now we want to see that for these finite bounds there is no difference between the writing and nonwriting variants, so that all our results are really results on checking automata.

First notice that if M is k-reversal bounded we can add endmarkers to the working tape and imagine the tape as divided into k tracks. Track i contains the working tape alphabet symbol and the action to be taken on a square during sweep i. Also track i contains a little mark indicating the rightmost (for i odd) or leftmost (for i even) symbol seen during sweep i. This nondeterministic alteration of the working tape can obviously be done by an a-transducer and so we can get a new base language within the same full semiAFL. Then we can simulate M by a machine which first reads track 1 from left-to-right, attempting to perform the operations inscribed therein; if any are illegal it blocks. If it reads the right mark it runs down to the right endmarker, reverses and returns to the special mark. Now it reads track 2 from right-to-left again performing the indicated operations and blocking if this cannot be done. This time the special mark in track 2 tells it to run to the left endmarker and reverse and return to the mark and start on track 3. This continues until either a block occurs or a whole computation has been simulated and the machine can accept or reject as appropriate. Details are left to the reader; we state the relevant result.

Lemma 2.11. Let $\mathscr{L}$ be a full semiAFL. *Let M be a k-reversal bounded, $\mathscr{L}$-based fra. There is a strictly k-reversal full sweep nonwriting deterministic $\mathscr{L}$-based fra $\bar{M}$ with* $L(M) = L(\bar{M})$.

Observe that if M is a nonresting k-crossing bounded fca it is automatically k-visit bounded since every new visit to a square means crossing some boundary to the square. By padding out the working tape with dummy symbols we can force the machine to be nonresting while by adding endmarkers and a special mark it can be made right touching. This is discussed in [20] for two-way machines but, since input head motion does not affect the construction, the proof carries over for on-line machines.

Lemma 2.12. *Let $\mathscr{L}$ be a full semi* AFL. *Let M be a k-crossing bounded, $\mathscr{L}$-based fca. There is a k-visit bounded, k-crossing bounded, $\mathscr{L}$-based, nonresting and right touching fva $\bar{M}$ with* $L(M) = L(\bar{M})$.

If a machine has bounces, each bounce from square i to square $i+1$ and back will increase the number of visits to i by 1 but the number of crosses of boundary i by 2. This is the only way the number of crosses can exceed the number of visits overall.

Hence a bounce-free k-visit bounded machine is also k-crossing bounded. Again the conversion to a bounce-free machine mentioned in [20] for the two-way case applies to the on-line case also.

Lemma 2.13. *Let $\mathscr{L}$ be a full semi*AFL. *Let M be a k-visit bounded, $\mathscr{L}$-based fva. There is a k-crossing bounded, $\mathscr{L}$-based, bounce-free fca $\bar{M}$ with* $\mathrm{L}(M) = \mathrm{L}(\bar{M})$.

The conversion from k-visit bounded to nonwriting $(k+1)$-reversal bounded machines in [20] definitely does not go over for on-line machines. Indeed, we shall see in the next section that

$$\mathrm{FINITEREVERSAL}(\mathscr{L}) \neq \mathrm{FINITEVISIT}(\mathscr{L})$$

for any full AFL $\mathscr{L}$ not closed under the finite visit operator ($\mathscr{L} \neq$ FINITE-VISIT($\mathscr{L}$)). However the construction in the proof of Theorem 2.1 of [20] of a nonwriting deterministic strictly k-visit machine from a nonresting right-touching k-visit machine does not involve input head motion and so still works. The idea is again to divide the working tape into k tracks; the trick is to show that the k-visit machine can find its right track—i.e., know which visit to a square it is simulating. A similar construction appears in Section 4 where we show that

$$\mathrm{FINITEVISIT}(\mathrm{FINITEVISIT}(\mathscr{L})) = \mathrm{FINITEVISIT}(\mathscr{L}).$$

Lemma 2.14. *Let $\mathscr{L}$ be a full semi*AFL. *Let M be a k-visit bounded, $\mathscr{L}$-based, nonresting right touching fva. There is a strictly k-visit, $\mathscr{L}$-based, nonwriting deterministic fva $\bar{M}$ with* $\mathrm{L}(M) = \mathrm{L}(\bar{M})$.

We can summarize our relationships as follows.

Theorem 2.15. *Let $\mathscr{L}$ be a full semi*AFL. *For any $k \geq 1$,*
(1) k-REVERSAL($\mathscr{L}$) = k-DETNWREVERSAL($\mathscr{L}$), *and*
(2) k-VISIT($\mathscr{L}$) = k-CROSS($\mathscr{L}$)
= k-DETNWVISIT($\mathscr{L}$) = k-DETNWCROSS($\mathscr{L}$).

Using this theorem as our justification we shall speak of just k-REVERSAL($\mathscr{L}$) or k-VISIT($\mathscr{L}$) and let our machines be writing or nonwriting as convenience dictates. We shall not bother to restate results for k-NWREVERSAL($\mathscr{L}$), etc., or for the crossing bounded families like k-CROSS($\mathscr{L}$). The conversion to a deterministic machine is of less significance than it might seem: since a preset Turing machine always has a choice of working tapes, the restriction to deterministic action on one of these tapes is valueless.

We first turn to FINITEREVERSAL($\mathscr{L}$) as most facts on this family can be obtained by playing with known results on homomorphic replication [11, 19], and control sets on linear context-free grammars [17].

3. Finite reversal automata

In this section we characterize FINITEREVERSAL($\mathscr{L}$) as the closure of $\mathscr{L}$ under homomorphic replication and also as the result of iterating controls on linear context-free grammars starting with control sets in $\mathscr{L}$. We establish a decomposition theorem

$$kr\text{-REVERSAL}(\mathscr{L}) = k\text{-REVERSAL}(r\text{-REVERSAL}(\mathscr{L})),$$

which is useful in providing padding or translation techniques for our hierarchy results. Finally we give some syntactic lemmas and conclude the section with the general hierarchy theorem that whenever $\mathscr{L} \neq$ FINITEREVERSAL($\mathscr{L}$), then $k+1$ reversals are strictly more powerful than k reversals and that if $\mathscr{L}$ is a full AFL not closed under FINITEVISIT, then

$$\text{FINITEVISIT}(\mathscr{L}) \neq \text{FINITEREVERSAL}(\mathscr{L}).$$

First we give an "algebraic", i.e., operation-theoretic, characterization of k-REVERSAL($\mathscr{L}$) in terms of replications.

Definition 3.1. For $k \geq 1$, a language L and a symbol c not appearing in any word of L, let

$$f_c(\mathrm{L}, 1) = \mathrm{L}c,$$

$$f_c(\mathrm{L}, 2k) = \{(wcw^Rc)^k \mid w \in \mathrm{L}\},$$

and

$$f_c(\mathrm{L}, 2k+1) = \{(wcw^Rc)^k wc \mid w \in \mathrm{L}\}.[7]$$

For most purposes the identity of the symbol c is irrelevant so we simply drop it and write $f(\mathrm{L}, k)$ rather than $f_c(\mathrm{L}, k)$. This operation is similar to the operations of homomorphic replication and of duplication.

Definition 3.2. Let ρ be a function from $\{1, \ldots, n\}$ to $\{1, R\}$, L be a language and $h_1, \ldots, h_n$ homomorphisms. The operation

$$g(\rho, \mathrm{L}, h_1, \ldots, h_n) = \{h_1(w^{\rho(1)}) \cdots h_n(w^{\rho(n)}) \mid w \in \mathrm{L}\}$$

is a *homomorphic replication of degree n*; if each $\rho(i) = 1$, it is a *duplication*. If $\rho(i) = 1$ for i odd and $\rho(i) = R$ for i even, it is an *alternating homomorphic replication of degree n*; in this case, we also write $\rho = \rho_n$ and call ρ_n the *alternating replication of degree n*. For a family of languages $\mathscr{L}$, let

$$\mathscr{L}_\rho = \{g(\rho, \mathscr{L}, h_1, \ldots, h_n) \mid \mathrm{L} \in \mathscr{L}, h_1, \ldots, h_n \text{ homomorphisms}\}.$$

[7] We write $e^R = e$ and for a symbol a, $a^R = a$; this notation is extended to words by $(xy)^R = y^R x^R$ and to languages L by $\mathrm{L}^R = \{w^R \mid w \in \mathrm{L}\}$. This operation is called *reversal*.

Clearly, if ρ is of degree n, $\mathscr{L}_\rho \subseteq \mathscr{L}_{\rho_{2n}}$, so $\bigcup_\rho \mathscr{L}_\rho = \bigcup_n \mathscr{L}_{\rho_n}$.

Obviously, k-REVERSAL($\mathscr{L}$) contains every alternating homomorphic replication of degree k of members of $\mathscr{L}$. Now we observe that this characterizes k-REVERSAL($\mathscr{L}$).

Theorem 3.3. *Let* $\mathscr{L}$ *be a full semi*AFL *and* $k \geq 1$. *Then*

$$k\text{-REVERSAL}(\mathscr{L}) = \mathscr{L}_{\rho_k} = \hat{\mathscr{M}}(\{f(\mathrm{L}, k) \mid \mathrm{L} \in \mathscr{L}\}).$$

Proof. Clearly $f(\mathrm{L}, k) \in k$-REVERSAL($\mathscr{L}$) whenever $\mathrm{L} \in \mathscr{L}$ and $g(\rho_k, \mathrm{L}, h_1, \ldots, h_k)$ can be obtained as an a-transducer mapping of $f(\mathrm{L}, k)$ for any alternating replication. Thus it remains to express members of k-REVERSAL($\mathscr{L}$) as alternating replications of members of $\mathscr{L}$. The proof follows lines of one in [17].

Let $M = (K, \Sigma, \Gamma, \delta, q_0, F, \mathrm{L}_1)$ be k-reversal bounded for L_1 in $\mathscr{L}$. We can assume that M is nonwriting, deterministic and full sweep. By padding the base language, we can also assume that M is nonresting. Further, we can assume that each word in L_1 is of size at least 3.

For our new alphabet we use special symbols called sigma symbols which describe all the information needed about M's actions on its k visits to a square. For this proof, a sigma symbol is a new symbol of the form $\sigma = \langle A, \alpha_1, \ldots, \alpha_k \rangle$ where $A \in \Gamma$, each $\alpha_i = (q_i, a_i, q_i', r_i)$ and $\delta(q_i, a_i, A) = (q_i', A, r_i)$. Thus σ encodes k visits to a square with contents A. It is a *middle* sigma symbol if $r_i = 1$ for odd i and $r_i = -1$ for even i. It is an *initial* sigma symbol if $q_1 = q_0$, $\alpha_i = \alpha_{i+1}$ for each even i, $r_i = 1$ for each odd i and if k is even, $r_k = -1$ and $q_k' \in F$. This represents the leftmost working tape square. A *last* sigma symbol has $\alpha_i = \alpha_{i+1}$ for each odd i, and for k even, $r_i = -1$ and $q_k' \in F$; this represents the rightmost working tape square.

Let $Q_i(s) = q_i$, $Q_i'(s) = q_i'$ and $W(s) = A$. For a middle sigma symbol σ, let $h_i(\sigma) = a$. For σ initial, let $h_i(\sigma) = a_i$ for $i = 1$ or i even, and $h_i(\sigma) = e$ elsewhere. For σ last, let $h_i(\sigma) = a_i$ for i odd and $h_i(\sigma) = e$ for i even. If we assume that M can be in q_0 only initially, and can never leave a state of F, then the sets of middle, initial and last symbols are mutually disjoint and hence the h_i are well defined.

We must also make sure that two sigma symbols can encode adjacent squares. Call (σ, σ') consistent if for each odd i, $Q_i'(\sigma) = Q_i(\sigma')$ and for each even i, $Q_i'(\sigma') = Q_i(\sigma)$. Then the language

$$R = \{\sigma_1 \cdots \sigma_m \mid m \geq 3, \sigma_1 \text{ initial}, \sigma_m \text{ last}, \sigma_i \text{ middle sigma}$$
$$\text{symbol}, 2 \leq i \leq m-1, \text{ and } (\sigma_i, \sigma_{i+1}) \text{ consistent}, 1 \leq i \leq m-1\}$$

is regular. We can consider V as a homomorphism on the vocabulary of sigma symbols. Thus $\mathrm{L}_2 = W^{-1}(\mathrm{L}_1) \cap \mathrm{R}$ is in $\mathscr{L}$. Likewise, each h_i defines a homomorphism acting on L_2 and moreover one which is nonincreasing so $h_i(w^R) = (h_i(w))^R$. A string in L_2 completely encodes a computation of M so if we "unfold" the string to represent all k sweeps and decode the inputs read at each move, we get a word in

L(M) and all words of L(M) can be obtained in this fashion Thus L(M) = $g(\rho_k, \mathrm{L}_2, h_1, \ldots, h_k)$. □

Corollary 3.3.1. *Let $\mathscr{L}$ be a full semi*AFL *and $k \geq 1$. Then every member of k-*REVERSAL($\mathscr{L}$) *can be expressed either as $g(\rho_k, \mathrm{L}, h_1, \ldots, h_k)$ where $\mathrm{L} \in \mathscr{L}$, and each h_i is a nonincreasing homomorphism or as $M(f(\mathrm{L}, k))$ for $\mathrm{L} \in \mathscr{L}$ and M a nonincreasing gsm.*

Proof. In the construction above, the h_i were obtained as nonincreasing. Then $g(\rho_k, \mathrm{L}, h_1, \ldots, h_k)$ can be obtained from $f(\mathrm{L}, k)$ by a nonincreasing gsm. □

Corollary 3.3.2. *For any full semi*AFL $\mathscr{L}$, FINITEREVERSAL($\mathscr{L}$) *is the closure of $\mathscr{L}$ under homomorphic replication.*

Corollary 3.3.3. *The family of finite reversal checking automaton languages is the closure of the regular sets under homomorphic replication.*

Remark. Corollary 3.3.3 was noted without proof in [19] where it was observed that the equal matrix languages or finite turn checking automaton languages of Siromoney [28, 29] are actually the closure of the regular sets under duplication, not replication. It was also shown by Klingenstein [40].

Theorem 3.3 gives us at once a "translational" or "decomposition" theorem.

Theorem 3.4. *Let $\mathscr{L}$ be a full semi*AFL *and let $k, r \geq 1$. Then*

$$kr\text{-REVERSAL}(\mathscr{L}) = k\text{-REVERSAL}(r\text{-REVERSAL}(\mathscr{L})).$$

Proof. First notice that $f_c(\mathrm{L}, kr)$ can be obtained from $\mathrm{L}' = f_d(f_c(\mathrm{L}, r), k)$ by simply turning each block cdc, cd or each d alone (not adjacent to c's) into a c. Hence Theorem 3.3 yields

$$kr\text{-REVERSAL}(\mathscr{L}) \subseteq k\text{-REVERSAL}(r\text{-REVERSAL}(\mathscr{L})).$$

On the other hand, any member of k-REVERSAL(r-REVERSAL($\mathscr{L}$)) can be represented as

$$\mathrm{L} = g(\rho_k, g(\rho_r, \mathrm{L}_1, u_1, \ldots, u_r), v_0, \ldots, v_{k-1})$$

for $u_1, \ldots, u_r, v_0, \ldots, v_{k-1}$ nonincreasing homomorphisms. and L_1 in $\mathscr{L}$. For $0 \leq i \leq k-1$ and $1 \leq j \leq r$, define the homomorphism h_{ir+j} by $h_{ir+j} = v_i u_j$ for i even and $h_{ir+j} = v_i u_{1+(r-j)}$ for i odd. This defines kr nonincreasing homomorphisms. Notice that since u_i and v_j are nonincreasing homomorphisms, $v_i((u_j(w))^R) = v_i(u_j(w^R))$. Thus $\mathrm{L} = g(\rho_{kr}, \mathrm{L}_1, h_1, \ldots, h_{kr})$. So L is in kr-REVERSAL($\mathscr{L}$). □

Corollary 3.4.1. *For any full semi*AFL $\mathscr{L}$ *and any* $k \geq 2$,

$$\text{FINITEREVERSAL}(\mathscr{L}) = \bigcup_r k\text{-REVERSAL}_r(\mathscr{L}).$$

Theorem 3.3 also gives us a grammatical characterization of FINITEREVERSAL($\mathscr{L}$) in terms of control sets on linear context-free grammars. Let us give the formal definitions for future use.

Definition 3.5. A *context-free* grammar is a quadruple $G = (V, \Sigma, P, S)$ where V is a finite vocabulary, $\Sigma \subset V$, $S \in V - \Sigma$ and P is a finite set of productions of the form $Z \to y$, $Z \in V - \Sigma$, $y \in V^*$. For a production $p: Z \to y$ in P, $u, v \in V^*$, we write $uZv \Rightarrow uyv$ and if $u \in \Sigma^*$, $uZv \overset{\mathrm{L}}{\Rightarrow} uyv$ and $uZv \overset{p}{\Rightarrow} uyv$. We let $\overset{*}{\Rightarrow}$ ($\overset{\mathrm{L}^*}{\Longrightarrow}$) be the transitive reflexive closure of $\Rightarrow$ ($\overset{\mathrm{L}}{\Rightarrow}$) and extend $\overset{p}{\Rightarrow}$ by letting $u_1 \overset{xy}{\Longrightarrow} u_3$ whenever $u_1 \overset{x}{\Rightarrow} u_2$ and $u_2 \overset{y}{\Rightarrow} u_3$. The language *generated by G* is

$$\mathrm{L}(G) = \{w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w\},$$

and *the language generated by G with control C* is

$$\mathrm{L}(G, C) = \{w \in \Sigma^* \mid \exists y \in C, S \overset{y}{\Rightarrow} w\}.$$

For a set S of symbols we shall let $\#_S(w)$ be the number of occurrences of members of S in w; if $S = \{a\}$, we write $\#_a(w)$.

Definition 3.6. A context-free grammar $G = (V, \Sigma, P, S)$ is *regular* if every production of P is of the form $Z \to y$ for y in $\Sigma^*(V \cup \{e\})$. It is *linear* if every production of P is of the form $Z \to y$ for $\#_{V-\Sigma}(y) \leq 1$. It is *nonterminal bounded* if there is a k such that whenever $Z \overset{*}{\Rightarrow} y$, then $\#_{V-\Sigma}(y) \leq k$. It is *left derivation bounded* if there is a k such that whenever $Z \overset{\mathrm{L}^*}{\Longrightarrow} y$ then $\#_{V-\Sigma}(y) \leq k$ and *derivation bounded* if there is a k such that for each w in L(G) there is a derivation $S \Rightarrow y_1 \Rightarrow \cdots \Rightarrow y_r \Rightarrow w$ with $\#_{V-\Sigma}(y_i) \leq k$ for $1 \leq i \leq r$. Each subtype of context-free grammar generates a language of the same type. We denote the families of context-free (respectively, regular, linear context-free, nonterminal bounded, left derivation bounded, derivation bounded) languages by CF(REGL, LINCF, NTB, LDB, DB) and denote the corresponding families of grammars by adding G to the end of the name.

A derivation $u \overset{\mathrm{L}^*}{\Longrightarrow} v$ is a *left-to-right derivation*; we place controls only on left-to-right derivations. The various restrictions in Definition 3.6 are restrictions on the number and placement of nonterminals. Linear context-free grammars have at most one nonterminal in any word derived from S while regular grammars are linear ones in which the nonterminal must appear rightmost. Nonterminal bounded grammars put a uniform limit on the number of nonterminals in any derived word while left derivation bounded grammars limit only left-to-right derivations and derivation bounded means that each word in the generated language has some derivation obeying the nonterminal limit. Further properties of these families can be found in [2, 13, 14, 18, 25, and 31].

We now give a general notation for the result of applying controls from a family $\mathscr{L}$ of languages to grammars from a family $\mathscr{G}$ and for iterating the process.

Definition 3.7. For a family of grammars $\mathscr{G}$ and a family of languages $\mathscr{L}$, let

$$\mathrm{CONTROL}(\mathscr{G},\mathscr{L})=\{L(G,C)\mid G\in\mathscr{G}, C\in\mathscr{L}\}.$$

Let

$$\mathrm{CONTROL}_0(\mathscr{G},\mathscr{L})=\mathscr{L}$$

and for $n\geqslant 0$, let

$$\mathrm{CONTROL}_{n+1}(\mathscr{G},\mathscr{L})=\mathrm{CONTROL}(\mathscr{G},\mathrm{CONTROL}_n(\mathscr{G},\mathscr{L}))$$

and

$$\mathrm{CONTROL}_\infty(\mathscr{G},\mathscr{L})=\bigcup_n \mathrm{CONTROL}_n(\mathscr{G},\mathscr{L}).$$

Thus CONTROL(LINCFG, CF) is the family of languages obtained by using context-free languages to control left-to-right (i.e., $\xrightarrow{L*}$) derivations in linear context-free grammars and $\mathrm{CONTROL}_\infty$(LINCFG, CF) is the result of iterating this process.

Theorem 3.8. *Let* $\mathscr{L}$ *be a full semi*AFL. *Then*

$$2\text{-REVERSAL}(\mathscr{L})=\mathrm{CONTROL}(\mathrm{LINCFG},\mathscr{L})$$

and

$$\mathrm{FINITEREVERSAL}(\mathscr{L})=\mathrm{CONTROL}_\infty(\mathrm{LINCFG},\mathscr{L})$$
$$=\mathrm{CONTROL}_\infty(\mathrm{NTBG},\mathscr{L}).$$

Proof. In [17] it is shown that

$$\mathrm{CONTROL}(\mathrm{LINCFG},\mathscr{L})=\hat{\mathscr{M}}(\{f(\mathrm{L},2)\mid \mathrm{L}\in\mathscr{L}\})=\mathscr{L}_{\rho_2}$$

and

$$\mathrm{CONTROL}_\infty(\mathrm{LINCFG},\mathscr{L})=\mathrm{CONTROL}_\infty(\mathrm{NTBG},\mathscr{L})$$

so iterating controls on linear context-free grammars is as powerful as iterating controls on nonterminal bounded grammars. The result is immediate from Theorems 3.3 and 3.4. □

Corollary 3.8.1. *The family of finite reversal checking automaton languages is*

$$\mathrm{CONTROL}_\infty(\mathrm{LINCFG}, \mathrm{REGL}) = \mathrm{CONTROL}_\infty(\mathrm{NTBG}, \mathrm{REGL}),$$

the family of languages obtained by iterating controls on linear context-free or on nonterminal bounded grammars starting with the regular sets.

We now wish to establish our general hierarchy theorem for finite reversal automata. We shall use established results on $\mathrm{CONTROL}_n(\mathrm{LINCFG}, \mathscr{L})$ and two translational results, the following lemma and Theorem 3.4.

Lemma 3.9. *Let $\mathscr{L}$ be a full semi*AFL. *For any $s \geq 1, r \geq 1$, if $f_c(\mathrm{L}, k+s)$ is in k-*REVERSAL($\mathscr{L}$), *then*

$$f_c(\mathrm{L}, k+s+r) \in (k+r+1)\text{-REVERSAL}(\mathscr{L}).$$

Proof. Let $f_c(\mathrm{L}, k+s) = \mathrm{L}(M)$ for a strictly k-reversal nonwriting full sweep fra M. Since $s \geq 1$, in every accepting computation on $wcw^Rc\ldots$, at least one w or w^R must be processed in one sweep without any reversal of the working tape head.

We construct a $(k+r+1)$-reversal bounded fra $\bar{M}$ from M to accept $f_c(\mathrm{L}, k+s+r)$ as follows. The key is to get some copy of w or w^R printed during the simulation of M; this copy can then be used to check off as many copies of w or w^R as desired.

At the start of a computation, $\bar{M}$ guesses that sweep i of M will entirely process $w^t, t \in \{1, R\}$ and records (i, t) in its finite state control. It starts simulating M, keeping track in its finite state control of which sweep M is on, the number of c's read as input to date and thus the number of w's or w^R's, and whether w or w^R is being read at each step. When sweep i arrives, $\bar{M}$ waits until w^t is started and then writes w^t on a second track in special "colors" surrounded by special barrier symbols as it is processing w^t. If w^t is not completely processed in sweep i, $\bar{M}$ blocks. Otherwise it succeeds in writing down w^t in correct order, though perhaps with differently colored symbols in between (because there may be transitions on e input); later, symbols not in w^t can be ignored. Now $\bar{M}$ continues the simulation until it finishes sweep k of M in an accepting state of M, having read $k+s$ c's and no further symbols. If it gets this far without a block, $\bar{M}$ does not accept. Instead it goes and finds the special colored copy of w^t and locates the working tape head in such a position that it now can read w if $k+s$ is even and w^R if $k+s$ is odd. This may at worst take an extra sweep. Then $\bar{M}$ simply sweeps back and forth r times over w or w^R, checking off the input. Thus $\bar{M}$ takes at most $k+r+1$ sweeps and accepts $f(\mathrm{L}, k+s+r)$. □

In view of Theorem 3.3, we can draw the following conclusion.

Lemma 3.10. *Let $\mathscr{L}$ be a full semi* AFL. *If*

$$k\text{-REVERSAL}(\mathscr{L}) = (k+s)\text{-REVERSAL}(\mathscr{L})$$

for any $k, s \geq 1$, then for all $r \geq 1$,

$$(k+r+1)\text{-REVERSAL}(\mathscr{L}) = (k+s+r)\text{-REVERSAL}(\mathscr{L}).$$

Proof. If

$$k\text{-REVERSAL}(\mathscr{L}) = (k+s)\text{-REVERSAL}(\mathscr{L}),$$

then by Theorem 3.3, $f_c(\mathrm{L}, k+s)$ is in k-REVERSAL($\mathscr{L}$) for all L in $\mathscr{L}$. Hence by Lemma 3.9, $f_c(\mathrm{L}, k+s+r)$ is in $(k+r+1)$-REVERSAL($\mathscr{L}$) for all L in $\mathscr{L}$. Since $k+r+1 \geq k+s+r$, by Theorem 3.3,

$$(k+s+r)\text{-REVERSAL}(\mathscr{L}) = (k+r+1)\text{-REVERSAL}(\mathscr{L}) \quad \square$$

Now we have the general hierarchy theorem.

Theorem 3.11. *Let $\mathscr{L}$ be a full semi* AFL *with* $\mathscr{L} \neq$ FINITEREVERSAL($\mathscr{L}$).

(1) *For every $k \geq 1$,* k-REVERSAL($\mathscr{L}$) $\subsetneq$ $(k+1)$-REVERSAL($\mathscr{L}$).

(2) *For every $k \geq 2$, $n \geq 1$,* k-REVERSAL$_n$($\mathscr{L}$) $\subsetneq$ k-REVERSAL$_{n+1}$($\mathscr{L}$).

(3) *For every $k \geq 2$, $n \geq 1$,* k-REVERSAL$_n$($\mathscr{L}$) *is not closed under concatenation and does not contain the closure of* 2-REVERSAL($\mathscr{L}$) *under concatenation.*

(4) FINITEREVERSAL($\mathscr{L}$) *is not a full* AFL *and does not contain the closure of* 2-REVERSAL($\mathscr{L}$) *under Kleene* +.

Proof. Statements (2), (3) and (4) are shown in [17] for $k = 2$, since

$$2\text{-REVERSAL}(\mathscr{L}) = \text{CONTROL}(\text{LINCFG}, \mathscr{L}).$$

By Theorem 3.4, if $t = \lfloor \operatorname{Log}_2 k \rfloor$ then

$$2\text{-REVERSAL}_{nt}(\mathscr{L}) \subseteq k\text{-REVERSAL}_n(\mathscr{L}) \subseteq 2\text{-REVERSAL}_{n(t+1)}(\mathscr{L}).^{8}$$

Hence k-REVERSAL$_n$($\mathscr{L}$) cannot contain the closure of 2-REVERSAL($\mathscr{L}$) under concatenation and so is not concatenation closed. If

$$k\text{-REVERSAL}_n(\mathscr{L}) = k\text{-REVERSAL}_{n+1}(\mathscr{L}),$$

then by Corollary 3.4.1,

$$\text{FINITEREVERSAL}(\mathscr{L}) = k\text{-REVERSAL}_n(\mathscr{L}) = 2\text{-REVERSAL}_{n(t+1)}(\mathscr{L}),$$

a contradiction. This establishes (2) and (3).

[8] For a real number r, $\lfloor r \rfloor = \operatorname{Max}(\{n \leq r \mid n \text{ is an integer}\})$.

Suppose that

$$k\text{-REVERSAL}(\mathscr{L}) = (k+2)\text{-REVERSAL}(\mathscr{L}).$$

We claim that for all $p \geqslant 2$,

$$k\text{-REVERSAL}(\mathscr{L}) = (k+p)\text{-REVERSAL}(\mathscr{L}).$$

It is true by hypothesis for $p = 2$. Suppose we have shown it for p. In Lemma 3.10, let $k+(p-2)$ play the role of k, $s = 2$ and $r = 1$. Then

$$\begin{aligned}(k+p+1)\text{-REVERSAL}(\mathscr{L}) = (k+p)\text{-REVERSAL}(\mathscr{L})\\ = k\text{-REVERSAL}(\mathscr{L}).\end{aligned}$$

Hence, by Corollary 3.4.1,

$$\begin{aligned}\text{FINITEREVERSAL}(\mathscr{L}) &= \bigcup_p p\text{-REVERSAL}(\mathscr{L})\\ &= \bigcup_p (k+p)\text{-REVERSAL}(\mathscr{L}) = k\text{-REVERSAL}(\mathscr{L}),\end{aligned}$$

contradicting (2).

Thus kREVERSAL$(\mathscr{L}) \neq (k+2)$-REVERSAL$(\mathscr{L})$ for all $k \geqslant 1$. Now suppose that k-REVERSAL$(\mathscr{L}) = (k+1)$-REVERSAL$(\mathscr{L})$. By Theorem 3.4,

$$\begin{aligned}2k\text{-REVERSAL}(\mathscr{L}) &= 2\text{-REVERSAL}(k\text{-REVERSAL}(\mathscr{L}))\\ &= 2\text{-REVERSAL}((k+1)\text{-REVERSAL}(\mathscr{L}))\\ &= (2k+2)\text{-REVERSAL}(\mathscr{L}),\end{aligned}$$

a contradiction. This establishes (1). □

Corollary 3.11.1. *For one-way checking automata, $k+1$ reversals are more powerful than k reversals for each $k \geqslant 1$.*

If $\mathscr{L}$ is a full AFL, then FINITEVISIT$(\mathscr{L})$ is also a full AFL, but if $\mathscr{L} \neq$ FINITEREVERSAL$(\mathscr{L})$, FINITEREVERSAL$(\mathscr{L})$ cannot be a full AFL. Hence finite reversals are strictly less powerful than finite visits for on-line machines.

Theorem 3.12. *If $\mathscr{L}$ is a full* AFL *and $\mathscr{L} \neq$* FINITEVISIT$(\mathscr{L})$, *then*

$$\text{FINITEREVERSAL}\mathscr{L}) \subsetneq \text{FINITEVISIT}(\mathscr{L}).$$

Corollary 3.12.1. *The class of finite reversal checking automaton languages is properly contained in the class of finite visit checking automaton languages.*

Remark. Various special cases of the main results of this section have been noticed repeatedly under different formulations: homomorphic replications, control sets, reversal bounded machines. For example, Klingenstein [40] established a version of Theorem 3.3 for $\mathscr{L}$ = REGL, and a much stronger version of Theorem 3.11(1) involving the precise form of a replication function $\rho:\{1,\ldots,n\}\to\{1,R\}$ for the cases $\mathscr{L}$ = REGL and $\mathscr{L}$ = CF. The former result will generalize to arbitrary full semiAFL's and it seems plausible that the latter might do so also. Parts of Theorem 3.11 were shown by Ginsburg and Spanier [11] and Rodriguez [27] and, for $\mathscr{L}$ = CF, by Ibarra [37].

Both Erni [35] and Sudborough [41] have noticed the connection between CONTROL$_k$(LINCFG, $\mathscr{L}$) and homomorphic replication (proven for general full semiAFL $\mathscr{L}$ in [17]) for the special case of $\mathscr{L}$ = CF. Khabbaz [38, 39] established

$$\mathrm{CONTROL}_k(\mathrm{LINCFG}, \mathscr{L}) \subsetneq \mathrm{CONTROL}_{k+1}(\mathrm{LINCFG}, \mathscr{L})$$

for the special cases $\mathscr{L}$ = LINCF and $\mathscr{L}$ = CF.

It was shown in [19] that if $\mathscr{L}$ is a concatentation closed full semiAFL not closed under homomorphic replication, then the closure of $\mathscr{L}$ under homomorphic replication—call it $\bigcup_\rho \mathscr{L}_\rho$— is not a full principal semiAFL nor an AFL, $\mathscr{F}(\bigcup_\rho \mathscr{L}_\rho)$ is not closed under homomorphic replication and is not full principal and the smallest full AFL containing $\mathscr{L}$ and closed under homomorphic replication is not principal; the assumption of concatenation closure is unnecessary. Similar observations were made by Ginsburg and Spanier [11] for $\mathscr{L}$ = REGL and for $\mathscr{L}$ = CF by Ibarra [37].

4. Finite Visit Automata

Now we concentrate on finite visit automata. We first establish a weaker (but harder to prove) variant of Theorem 3.4, namely k-VISIT(r-VISIT($\mathscr{L}$)) is contained in kr-VISIT($\mathscr{L}$). This shows that FINITEVISIT($\mathscr{L}$) is an idempotent operator on families of languages. Next we establish a grammatical characterization of FINITEVISIT($\mathscr{L}$) in terms of control sets of absolutely parallel grammars and as a consequence show that if $\mathscr{L}$ has the Parikh property so does FINITEVISIT($\mathscr{L}$). Then, two syntactic lemmas allow us to establish a strong hierarchy theorem for k-VISIT(REGL) and a weak general hierarchy theorem. Finally, we use these ideas to show that not all context-free languages are one-way nonerasing stack languages.

First we show that

$$k\text{-VISIT}(r\text{-VISIT}(\mathscr{L})) \subseteq kr\text{-VISIT}(\mathscr{L}).$$

The key is to take the working tape of the r-visit machine, fold in the input accepted by that machine, and then use this as the base for a kr-visit machine.

Theorem 4.1. *Let $\mathscr{L}$ be a full semi*AFL, k, $r \geq 1$. *Then*

$$k\text{-VISIT}(r\text{-VISIT}(\mathscr{L})) \subseteq kr\text{-VISIT}(\mathscr{L}).$$

Proof. We wish to show that $L(M) \in kr$-VISIT($\mathscr{L}$) if $M = (K, \Sigma, \Gamma, \delta, q_0, F, L_1)$ is a strictly k-visit nonwriting fva with a base L_1 in r-VISIT($\mathscr{L}$). We can assume that $L_1 = L(M_1)$ for $M_1 = (K_1, \Sigma_1, \Gamma_1, \delta_1, p_0, F_1, L_2)$ a strictly r-visit nonwriting deterministic fva with base L_2 in $\mathscr{L}$. We can also assume that M_1 is e-free for otherwise we can substitute for an e-rule $\delta(q, e, A) = \{(q', A, i)\}$ a rule reading a dummy input d, $\delta(q, d, A) = \{(q', A, i)\}$ and then have M simply skip over any d's read in its working tape. We construct a new machine M_2 which will be $\mathscr{L}$-based and kr-visit bounded and will accept $L(M)$.

The general idea is that working tapes of M_2 will consist of working tapes of M_1 with the input of M_1 wrapped around with suitable pointers for unwrapping.

As in the proof of Theorem 3.3 we use special "sigma" symbols to encode the behavior of M_1 during its r visits to a square. However our coding is more complicated. Here a sigma symbol is a symbol of the form

$$\sigma = \langle A, d_1, \alpha_1, \ldots, d_r, \alpha_r \rangle$$

for $A \in \Gamma_1$, each $d_i \in \{0, 1, -1\}$, each $\alpha_i = (q_i, a_i, q_i', j_i)$ for $\delta_1(q_i, a_i, A) = \{(q_i', A, j_i)\}$, $a_i \in \Sigma_1$, with the additional restrictions that $d_1 = -1$, $j_r \neq 0$, each $d_{i+1} = j_i$ and if $j_i = 0$, then $q_{i+1} = q_i'$. If $j_r = 1$ it is a *right* sigma symbol (to encode a right exiting computation) and if $j_r = -1$ it is a *left* sigma symbol. Let $h(\sigma) = A$ and for $1 \leq i \leq r$, $D_i(\sigma) = d_i$, $Q_i(\sigma) = q_i$, $Q_i'(\sigma) = q_i'$, $g_i(\sigma) = a_i$, $J_i(\sigma) = j_i$. Let Γ_L be the set of left sigma symbols and Γ_R the set of right sigma symbols.

Call a sigma symbol σ *initial* if $Q_1(\sigma) = p_0$ and each $J_i(\sigma) \in \{0, 1\}$, *final* if $Q_r'(\sigma) \in F_1$ and *last* if $J_i(\sigma) \in \{0, -1\}$ for each $i \neq r$.

In a sigma symbol σ, $h(\sigma)$ indicates the contents of a square (which does not change since M_1 is assumed to be nonwriting), $D_i(\sigma)$ indicates the direction from which the square was entered at visit i (1 for entry from the right, 0 for standstill, -1 for entry from the left), $g_i(\sigma)$ indicates the input symbol read and $Q_i(\sigma)$ the state at visit i, and $Q_i'(\sigma)$ and $J_i(\sigma)$ give the state change and working head change during visit i. An initial symbol encodes a leftmost square, a last symbol encodes a rightmost square, and a final symbol encodes the last square seen during an accepting computation

Now we must define consistency of pairs of sigma symbols. Roughly speaking, (σ, σ') is *consistent* if the listed behaviors are consistent with σ' encoding the square to the right of the square coded by σ. Thus each right exit listed in σ must agree with each left entry in σ' and left exists from σ' with right entries to σ. First, the two symbols must be either both right or both left sigma symbols. Let $1 \leq j_1 < \cdots < j_u \leq r$ list exactly those i with $J_i(\sigma) = 1$ and $1 \leq i_1 < \cdots < i_v \leq r$ list exactly those i for which $J_i(\sigma') = -1$. We must also have $u = \#\{i \mid D_i(\sigma') = -1\}$ and $v = \#\{i \mid D_i(\sigma) = 1\}$ and for right sigma symbols $u = v + 1$ and for left sigma symbols $u = v$. Then we need $Q_1(\sigma') = Q_{j_1}'(\sigma)$ and $Q_{i_p+1}(\sigma') = Q_{j_{p+1}}'(\sigma)$ for $1 \leq p \leq u - 1$ to line up right exits from σ, and $Q_{j_p+1}(\sigma) = Q_{i_p}'(\sigma')$ for $1 \leq p \leq v$ to handle left exits from σ'.

Let ¢ and \$ be new symbols. The sets

$$R_1 = \{¢\sigma_1 \cdots \sigma_m\$ \mid \text{each } \sigma_i \in \Gamma_R, \sigma_1 \text{ initial}, \sigma_m \text{ last and final}, (\sigma_i, \sigma_{i+1}) \text{ consistent}, 1 \leq i \leq m-1\}$$

and

$$R_2 = \{¢\sigma_1 \cdots \sigma_m\$ \mid \text{each } \sigma_i \in \Gamma_L, \sigma_1 \text{ initial and final}, \sigma_m \text{ last } (\sigma_i, \sigma_{i+1}) \text{ consistent for } 1 \leq i \leq m-1\}$$

are regular. Hence $L_3 = ¢h^{-1}(L_2)\$ \cap (R_1 \cup R_2)$ is in $\mathscr{L}$.

The language L_3 is the base for M_2. It contains the complete encoding of all accepting computations of M_1 and no other computations. All M_2 need do is follow these computations, simulating the actions of M on the input of M_2 and working tape symbols which are the input symbols of M_1 encoded into the strings of L_3. Right moves on the working tape of M will be simulated by advancing M_1's computation one step and left moves by going back one unit of time in M_1's computation.

Let us sketch how this is accomplished. Suppose M_2 has a working tape $y = ¢\sigma_1 \cdots \sigma_m\$$, each σ_i a sigma symbol. In its finite state control M_2 has registers containing the sigma symbol of y examined and the visit to that square of M_1's computation as well as the current state of the computation of M. Initially M_2 starts on σ_1 with visit number 1 and simulates state q_0 of M.

At some point in time, let M_2 have its working tape head on σ_i with visit number j simulating state q of M. It selects (if possible) an action $(q', g_j(\sigma_i), d)$ in $\delta(q, a, g_j(\sigma))$ for input symbol a (if $a \in \Sigma$ or without consulting its input tape if $a = e$), as if M were scanning working symbol $g_j(\sigma_i)$. Next M_2 will simulate M in state q'. But first M_2 must find the next working tape symbol of M. If $d = 0$, M does not alter its working tape head so M_2 remains in σ_i with visit number $j = j + d$.

If $d = 1$, M moves its working tape head right and so sees the symbol input to M_1 after $g_j(\sigma_i)$. Let $u = J_j(\sigma_i)$; this is the action of M_1's working tape head. If $u = 0$, M_2 stays in σ_i with visit number $j + 1 = j + d$. If $u \neq 0$, let

$$s = \#\{t \leq j \mid J_t(\sigma_i) = u\}.$$

This means that visit j was the sth right exit from σ_i for $u = 1$ or sth left exit for $u = -1$. Now M_2 moves its working tape head to σ_{i+u} with visit number that unique v with

$$s = \#\{t \leq v \mid D_t(\sigma_{i+u}) = -u\},$$

i.e., the vth visit to σ_{i+u} is the sth left entry for $u = 1$ or right entry for $u = -1$.

If $d = -1$, M moves its working tape head left and so sees the symbol input to M_1 just before $g_j(\sigma_i)$. Thus one essentially exchanges the roles of D and J (entry and exit) in the above formula. Here let $u = D_j(\sigma_i)$. If $u = 0$, M_2 stays in σ_i with visit number $j+1 = j+d$. If $u \neq 0$, let

$$s = \#\{t \leq j \mid D_t(\sigma_i) = u\}.$$

Now M_2 moves to σ_{i+u} with visit number that unique v with

$$s = \#\{t \leq v \mid J_t(\sigma_{i+u}) = -u\}.$$

There is one other set of possibilities to mention. It may not be possible for M_2 to complete the action above because $i+u = 0$ or $i+u = m+1$ in which case M_2 either moves left to ¢ or right to \$. If the new state of q' is not in F, M_2 halts and rejects because M falls off its working tape in a nonaccepting state. If $q' \in F$ there are three possibilities, for all of which M_2 will accept. To have $i+u = m+1$, M_2 must first be in σ_m with a visit number j such that either $d = 1$ and $J_j(\sigma_m) = 1$ or $d = -1$ and $D_j(\sigma_m) = 1$. The second case is impossible because σ_m is a last (rightmost) sigma symbol and could never be entered from the right. In the first case we must for this reason have $j = r$ and σ_m both right sigma, and final. Thus M_1 has an accepting right exiting computation as does M. Now to have $i+u = 0$, M_2 must be in σ_1 with a visit number j such that either $d = 1$ and $J_j(\sigma_1) = -1$ or $d = -1$ and $D_j(\sigma_1) = -1$. Since σ_1 is initial, in the second case $j = 1$ so M has a left exiting accepting computation and left exits off the first symbol input to M_1. In the first case σ_1 must be a left sigma symbol and $j = r$, so M and M_1 have left exiting accepting computations. In all these cases, M_2 accepts.

Since M is strictly k-visit. the simulation of M takes M_2 to each $g_j(\sigma_i)$ exactly k times and there are r such symbols encoded per square of working tape. Thus M_2 is kr-visit bounded. Clearly M_2 accepts $L(M)$. □

Corollary 4.1.1. *For any full semi*AFL $\mathscr{L}$,

$$\text{FINITEVISIT}(\text{FINITEVISIT}(\mathscr{L})) = \text{FINITEVISIT}(\mathscr{L}).$$

The proof of Theorem 4.1 depends heavily on the fact that the input tape of M_1 was one-way but was independent of the input head motion of M. Thus if we let TWOFINITEVISIT($\mathscr{L}$) be the family of languages accepted by $\mathscr{L}$-based finite visit automata with a two-way input tape with endmarkers, we get the following corollary.

Corollary 4.1.2. *For any full semi*AFL $\mathscr{L}$,

$$\text{TWOFINITEVISIT}(\text{FINITEVISIT}(\mathscr{L})) = \text{TWOFINITEVISIT}(\mathscr{L}).$$

Corollary 4.1.3. *For any full semi*AFL $\mathscr{L}$, FINITEVISIT($\mathscr{L}$) *is a full semi*AFL *closed under deterministic two-way finite state transducer mappings, and is the closure of $\mathscr{L}$ under deterministic two-way finite state transducer mappings.*

Proof. Clearly a deterministic L-based fva can be regarded as a deterministic two-way finite state transduction on L while every deterministic two-way finite state transducer can be regarded as finite visit (otherwise it cycles; see [5, 25, 26]). □

We turn to a characterization of finite visit languages in terms of the absolutely parallel grammars introduced by Rajlich [25].

Definition 4.2. An *absolutely parallel grammar* (*apg*) is a quadruple $G = (V, \Sigma, P, S)$, where V is a finite alphabet, $\Sigma \subset V$ is the subalphabet of *terminals*, $S \in V - \Sigma$ is the *initial symbol* and P is a finite set of rules of the form

$$(Y_1, \ldots, y_r) \to (y_1, \ldots, y_r), \quad Y_1, \ldots, Y_r \in V - \Sigma, y_1, \ldots, y_r \in V^*;$$

such a rule is of *degree r*. If all rules of P are of degree less than or equal to k, then G is of *degree k* and is called a *k-apg*. If $p:(Y_1, \ldots, Y_r) \to (y_1, \ldots, y_r)$ is in P, $u_1, \ldots, u_{r+1} \in \Sigma^*$, $w_1 = u_1 Y_1 \cdots u_r Y_r u_{r+1}$ and $w_2 = u_1 y_1 \cdots u_r y_r u_{r+1}$, then we write $w_1 \overset{p}{\Rightarrow} w_2$ and $w_1 \Rightarrow w_2$. The relation $\overset{*}{\Rightarrow}$ is the transitive reflexive extension of $\Rightarrow$. We extend $\overset{p}{\Rightarrow}$ transitively by letting $w_1 \overset{x}{\Rightarrow} w_2$ and $w_2 \overset{y}{\Rightarrow} w_3$ for $x, y \in P^+$ imply $w_1 \overset{xy}{\Longrightarrow} w_3$ The *language generated by G* is

$$L(G) = \{w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w\},$$

while for $C \subseteq P^+$, the language *generated by G with control C* is

$$L(G, C) = \{w \in \Sigma^* \mid \exists x \in C, S \overset{x}{\Rightarrow} w\}.$$

Although our definition explicitly had productions name themselves and so control words are in P^+, we shall not be rigid on this point and shall let productions have any arbitrary names when that is more convenient. Since all families of control sets used are closed under renaming, this is harmless.

We can now define the families of languages involved

Definition 4.3. The language generated by an apg is an *absolutely parallel language* (*apl*). The family of absolutely parallel grammars of degree k is denoted *k-APG* and the family of languages generated is *k-APL* while $APG = \bigcup_k k$-APG and $APL = \bigcup_k k$-APL.

The usual arguments show that APL = CONTROL(APG, REGL) and k-APL = CONTROL(k-APG, REGL)—that is applying regular control sets to absolutely parallel grammars of degree k does not increase their power. We shall show something stronger. A consequence of our characterization result for finite visit automata, FINITEVISIT($\mathscr{L}$) = CONTROL(APG, $\mathscr{L}$) for any full semiAFL $\mathscr{L}$, is that CONTROL(APG, APL) = APL, so that controlling absolutely parallel grammars by the languages they generate does not increase their power.

To show that FINITEVISIT($\mathscr{L}$) = CONTROL(APG, $\mathscr{L}$) one can merely observe that Rajlich's proof of the equivalence of APL with the results of two-way deterministic finite state transductions of regular sets goes over to transductions of other families of languages [25]. However, since we want to establish the precise result that CONTROL(k-APG, $\mathscr{L}$) = $2k$-VISIT($\mathscr{L}$) we must modify the constructions, particularly the simulation of machines by grammars.

Definition 4.4. A k-apg $G=(V,\Sigma,P,S)$ is in *normal form* if there is a symbol X in $(V-\Sigma)-\{S\}$ such that all rules of P are of the forms

(1) $S\to X^n$,

(2) $(Y_1,\ldots,Y_k)\to(y_1,\ldots,y_k)$,

where $Y_1,\ldots,Y_k\in V-\Sigma-\{S\}$ and either

(a) each $y_i=e$, $1\le i\le k$, or

(b) for some i, $y_i\in(\Sigma\cup\{e\})(V-\Sigma-\{S\})(\Sigma\cup\{e\})$, and $y_r\in V-\Sigma-\{S\}$ for $r\ne i$, or

(c) for some i, j, $y_i\in(V-\Sigma-\{S\})(V-\Sigma-\{S\})$, $y_j=e$ and $y_r\in V-\Sigma-\{S\}$ for $r\ne i,j$.

The rule of type (1) is *initial*; rules of type (2a) are *terminating* and of type (2c) are *i-splitting* and *j-terminating*.

We observe without proof that Definition 4.4 actually gives a normal form for k-apg's. The constructions needed follow the usual lines for obtaining grammars in Chomsky Normal Form, with suitable modifications in the control sets (cf. [4, 12, 17]).

Lemma 4.5. *Given a k-apg G we can construct a k-apg G' in normal form such that* L(G) = L(G') *and for any control set C there is a control set* $\mathscr{C}'$ *in* $\hat{\mathscr{M}}(C)$ *such that* L(G, C) = L(G', C').

We use this normal form to go from controlled grammars to machines.

Lemma 4.6. *Let G be a k-apg in normal form and let C be a control set. Then* L(G, C) $\in 2k$-VISIT($\hat{\mathscr{M}}(C)$).

Proof. Let $G=(V,\Sigma,P,S)$ be a k-apg in normal form with P labeled by symbols of C. We shall construct a $2k$-visit bounded machine M to accept L(G, C). Essentially we organize a computation of M on a word of C to trace the derivation tree of the derivation of G controlled by that word from left-to-right.

To simplify the construction of M, its base will be not all of C but C cut down to those words which actually control complete derivations of G. For a rule

$$p:(Y_1,\ldots,Y_k)\to(y_1,\ldots,y_k),$$

let $\lambda(p)=Y_1\cdots Y_k$ while $r(p)$ is either e if p is terminating or else lists the nonterminals of $y_1\cdots y_k$ in order of appearance. A pair of rules (p_1,p_2) is *consistent* if $\lambda(p_2)=r(p_1)$.

The language

$$R=\{p_0p_1\cdots p_t\mid p_i \text{ rule of } P,\ p_0 \text{ initial},\ p_t \text{ terminating},\ (p_i,p_{i+1}) \text{ consistent},\ 1\leqslant i\leqslant t\}$$

is regular so $C\cap R\in\hat{\mathcal{M}}(C)$. We take $C\cap R$ as the base of M.

The state set of M will be $K=\{f\}\cup\{1,\ldots,k\}\times\{\mathrm{L},R\}$ for a new symbol f, with initial state $(1,R)$ and final state set $F=\{f\}$. State f will be used only to accept when M walks off the left end of the working tape and so no square will be visited in state f. Each working tape square will be visited at most once in any of the other $2k$ states so the total number of visits per square will be at most $2k$. State (i,R) means the working tape head is moving left to right and is reading the derivation tree downward, currently tracing the history of coördinate i. State (i,L) means the working tape head is moving right to left and so is reading the tree upwards. Since each level of the tree has exactly k nonterminal nodes, a finite state machine can simulate tree motion on a linear tape.

We define the transition function of M as follows. For p_0 initial. M has transitions (we write $\delta(\alpha)=\{\beta\}$ as $\delta(\alpha)=\beta$):

$$\delta((i,R),e,p_0)=((i,R),p_0,1),\qquad 1\leqslant i\leqslant k,$$

$$\delta((i,L),e,p_0)=((i+1,R),p_0,1),\qquad 1\leqslant i\leqslant k-1,$$

$$\delta((k,L),e,p_0)=(f,p_0,-1).$$

Now we give transitions for production $p:(Y_1,\ldots,Y_k)\to(y_1,\ldots,y_k)$, by examining the possibilities. First, for p terminating we just have

$$\delta((i,R),e,p)=((i,L),p,-1),\qquad 1\leqslant i\leqslant k.$$

If p is neither terminating nor splitting, then for each i, $y_i=a_iZ_ib_i$, $a_i,b_i\in\Sigma\cup\{e\}$, $Z_i\in V-\Sigma-\{S\}$. Then down the tree we read a_i and up the tree b_i, so we have

$$\delta((i,R),a_i,p)=((i,R),p,1),\qquad 1\leqslant i\leqslant k.$$

and

$$\delta((i,L),b_i,p)=((i,L),p,-1),\qquad 1\leqslant i\leqslant k.$$

Suppose now p is i-splitting and j-terminating. If we are reading downwards, we continue to do so, possibly changing coordinate. First we have the transitions, for (s, R) with $s \neq j$.

$$\delta((s,R),e,p)=\begin{cases}((s,R),p,1) & \text{for } 1\leq s\leq i<j,\ 1\leq s<j<i \text{ or } s>i,j,\\ ((s-1,R),p,1) & \text{for } j<s\leq i,\\ ((s+1,R),p,1) & \text{for } i<s<j.\end{cases}$$

Since p is j-terminating, if M is reading the j^{th} coordinate downwards it must now read upwards. Thus:

$$\delta((j,R),e,p)=((j,L),p,-1).$$

There are more cases to distinguish for states (s, L). First we have the cases in which we are reading the right branch and continue reading the right branch upwards (leftwards), possibly switching coordinates.

$$\delta((s,L),e,p)=\begin{cases}((s,L),p,-1) & \text{for } s<i<j,\ s<j<i,\ s\geq i>j \\ & \qquad \text{or } s>j>i,\\ ((s+1,L),p,-1) & \text{for } j\leq s<i-1,\\ ((s-1,L),p,-1) & \text{for } i\leq s<j.\end{cases}$$

Now we have the two cases in which we are reading the left branch and must switch to reading the right branch downwards (rightwards).

$$\delta((i-1,L),e,p)=((i,R),p,1) \qquad \text{for } j<i,$$

$$\delta((i,L),e,p) \quad =((i+1,R),p,1) \qquad \text{for } i>j.$$

This completes the construction of M. Clearly $\mathrm{L}(M)=\mathrm{L}(G, C\cap R)=\mathrm{L}(G,C)$. $\square$

The construction for going from machines to grammars is more complicated. We shall define, as in the proof of Theorem 4.1, special sigma symbols to describe the transitions of M and shall associate to each sigma symbol a rule which will give the input symbols read during up to $2k$ visits to a working square. The nonterminals will be located so that in the final word the input symbols appear in proper order. Strings of sigma symbols representing accepting computations will form the control set for the grammar. We impose the bounce-free condition on the machine in order to go from a $2k$-visit machine to a k-apg.

Lemma 4.7. *Let $\mathscr{L}$ be a full semi*AFL. *Let M be a $2k$-visit bounded nonwriting* L*-based fva such that every accepting computation is $2k$-visit bounded, right touching and bounce-free. Then* L(M) *is in* CONTROL(k-APG, $\hat{\mathscr{M}}$(L)).

Proof. Let $M=(K,\Sigma,\Gamma,\delta,q_0,F,\mathrm{L})$ for $\mathrm{L}\in\mathscr{L}$. We construct a k-apg $G=(V,\Sigma,P,S)$ and a control set C in $\hat{\mathscr{M}}(\mathrm{L})$ such that $\mathrm{L}(G,C)=\mathrm{L}(M)$.

Define the sigma symbols for M as in the proof of Theorem 4.1. In this case a sigma symbol is a symbol of the form

$$\sigma=\langle A,d_1,\alpha_1,\ldots,d_r,\alpha_r\rangle \quad \text{for } 1\leqslant r\leqslant 2k,$$

with A, d_i and α_i as before. We define h, g_i, Q_i, Q_i', D_i and J_i as before for $1\leqslant i\leqslant r$. We let $r=[\sigma]$. We let $[\sigma]$ vary from 1 to $2k$ because imposing the bounce free condition means that we can't be sure that each square is visted exactly $2k$ times and r represents a guess as to the number of visits. The definitions of initial, last, final, right and left sigma symbols are the same except that $[\sigma]$ plays the role of $2k$. We let Γ_L be the set of left sigma symbols and Γ_R the set of right sigma symbols, and let ¢ and \$ be new. Consistency is defined as before.

Again, the set

$$\begin{aligned} R=\{\sigma_1\cdots\sigma_t \mid {} & \text{each } \sigma_i \text{ in } \Gamma_L,\ \sigma_1 \text{ initial and final},\ \sigma_t \text{ last},\\ & (\sigma_i,\sigma_{i+1}) \text{ consistent},\ 1\leqslant i\leqslant t-1\}\cup\\ \{\sigma_1\cdots\sigma_t \mid {} & \text{each } \sigma_i \text{ in } \Gamma_R,\ \sigma_1 \text{ initial},\ \sigma_t \text{ last and final},\\ & (\sigma_i,\sigma_{i+1}) \text{ consistent},\ 1\leqslant i\leqslant t-1\}\end{aligned}$$

is regular. The language $\mathrm{L}_1=h^{-1}(\mathrm{L})\cap R$ in $\hat{\mathscr{M}}(L)$ encodes accepting computations of M. We take as control set

$$C=¢(\mathrm{L}_1\cap\Gamma_L^*)\cup¢(\mathrm{L}_1\cap\Gamma_R^*)\$,$$

which is in $\hat{\mathscr{M}}(\mathrm{L})$.

Let $S,X_1,\ldots,X_k$ be new and $V=\Sigma\cup\{S,X_1,\ldots,X_k\}$. We define P, associating to ¢, \$ and each sigma symbol a rule labeled by that symbol.

First we associate $¢:S\to X_1$ and $\$:X_1\to e$.

To each σ we associate a rule as follows. First, let $u=\#\{i\mid D_i(\sigma)=-1\}$ and $v=\#\{i\mid J_i(\sigma)=1\}$. The rule σ will have left side $(X_1,\ldots,X_u)$ and the right side will contain nonterminals $X_1,\ldots,X_v$ in that order, so that the left hand side of the rule always has coordinates corresponding to all moves into the square from the left. Since M is bounce-free we can assume that $D_i(\sigma)=-1$ implies $D_{i+1}(\sigma)\neq-1$ (or else we have a right bounce into σ) and $J_i(\sigma)=1$ implies $J_{i+1}(\sigma)\neq1$ (or else we have a left bounce into σ). Thus $u,v\leqslant k$ so the rule will have degree at most k.

Let $1\leqslant j_1<\cdots<j_u\leqslant[\sigma]$ be those i for which $D_i(\sigma)=-1$ and $1\leqslant i_1<\cdots<i_v\leqslant[\sigma]$ those i with $J_i(\sigma)=1$. Let

$$t(s)=\#\{i_p\mid j_s\leqslant i_p\leqslant j_{s+1}\}\quad\text{for } s\neq u$$

and

$$t(u)=\#\{i_p\mid i_p>j_u\}.$$

This gives the number of right exits between the s^{th} and $(s+1)^{\text{st}}$ left entry; there is of course just one left exit. Let $r(s)=1+\sum_{i<s} t(i)$; this gives the number of the corresponding coordinate.

Our rule σ will have the form $(X_1,\ldots,X_u)\to(y_1,\ldots,y_u)$. Let us define y_s by cases. For convenience we use the convention that $j_{u+1}-1=[\sigma]$.

(1) $t(s)=0$. This means there are some (at least one) standstill moves but no right exits so no need for a nonterminal placeholder. Then we let

$$y_s=g_{j_s}(\sigma)g_{j_s+1}(\sigma)\cdots g_{j_{s+1}-1}(\sigma),$$

which simply gives the inputs between left entry and left exit.

(2) $t(s)\geqslant 1$. This means at least one right exit and maybe some standstill moves. Now y_s will list all the inputs from visit j_s through j_{s+1} and will insert a nonterminal after the input for each of the $t(s)$ right exits. So y_s looks like:

$$g_{j_s}(\sigma)\cdots g_{i_{r(s)}}(\sigma)X_{r(s)}\cdots g_{i_{r(s)+1}}(\sigma)X_{r(s)+1}\cdots$$
$$g_{i_{r(s)+t(s)-1}}(\sigma)X_{r(s)+t(s)-1}\cdots g_{j_{s+1}-1}(\sigma).$$

The correspondence between derivations of G and computations of M can be established by induction on the length of derivations and on the length of working tape read up to a point, as in Rajlich [26]. Then one can conclude that $S\overset{¢x}{\Longrightarrow}w$ if and only if x describes an accepting left exiting computation of M on w with working tape $h(x)$ and $S\overset{¢x\$}{\Longrightarrow}w$ if and only if the same thing holds for an accepting right exiting computation. Hence $\mathrm{L}(M)=\mathrm{L}(G,C)$. □

We can put these three lemmas together for our characterization theorem.

Theorem 4.8. *For any full semi*AFL $\mathscr{L}$ *and any* $k\geqslant 1$,

$$2k\text{-VISIT}(\mathscr{L})=\text{CONTROL}(k\text{-APG},\mathscr{L})$$

and

$$\text{FINITEVISIT}(\mathscr{L})=\text{CONTROL}(\text{APG},\mathscr{L}).$$

Corollary 4.8.1. *The family of finite visit checking automata languages is the family of absolutely parallel languages with 2k-visit automata corresponding precisely to degree k grammars.*

Corollary 4.8.2. *The family of absolutely parallel languages is closed under the operation of controlling absolutely parallel grammars. Formally,* CONTROL(APG, APL) = APL.

Proof. Since APL is a full semiAFL,

$$\begin{aligned}\text{CONTROL(APG, APL)} &= \text{FINITEVISIT(APL)}\\ &= \text{FINITEVISIT(FINITEVISIT(REGL))}\\ &= \text{FINITEVISIT(REGL)} = \text{APL}. \quad \square\end{aligned}$$

Another immediate corollary is that if $\mathcal{L}$ has the Parikh property so does FINITEVISIT($\mathcal{L}$).

Definition 4.9. A set S of n-tuples of nonnegative integers is *linear* if there are n-tuples of nonnegative integers $c, p_1, \ldots, p_r$ such that

$$S = \{c + t_1 p_1 + \cdots + t_r p_r \mid t_1, \ldots, t_r \text{ nonnegative integers}\}.$$

Such a set is *semilinear* if it is the finite union of linear sets.

Definition 4.10. Let $L \subseteq \Sigma^*$ and let $\Sigma = \{a_1, \ldots, a_n\}$, and $\bar{a} = (a_1, \ldots, a_n)$. A *Parikh mapping* of L is a function $f_{\bar{a}}$ from L into n-tuples of non-negative integers defined by

$$f_{\bar{a}}(L) = \{(\#_{a_1}(w), \ldots, \#_{a_n}(w)) \mid w \in L\}.$$

A language L has the *Parikh property* or the *semilinear property* if $f(L)$ is semilinear for some Parikh mapping f; a family $\mathcal{L}$ of languages has the *Parikh property* or the *semilinear property* if every member of $\mathcal{L}$ does.

Theorem 4.11. *If $\mathcal{L}$ is a full semi*AFL *with the Parikh property, so is* FINITEVISIT($\mathcal{L}$).

Proof. It suffices to consider $L(G, C)$ for C in $\mathcal{L}$ and G an apg. Let $G = (V, \Sigma, P, S)$: we can assume without loss of generality that $C \subseteq P^+$. For a rule p in P we can define $\lambda(p)$ and $r(p)$ as in the proof of Lemma 4.6 and let $t(p)$ be the terminals in the right hand side of p in order. Let $[\lambda(p)]$ and $[r(p)]$ be new symbols and $I = \{[\lambda(p)], [r(p)] \mid p \in P\}$. We define a new rule with label $h(p)$ as $[\lambda(p)] \to t(p)[r(p)]$ if $r(p) \neq e$ and $[\lambda(p)] \to t(p)$ otherwise. We regard h as a homomorphism on P^+. Thus $G' = (I \cup \Sigma, \Sigma, h(P), [S])$ is a regular grammar and $h(C)$ is in $\mathcal{L}$. Further, for any Parikh maping f, $f(L(G, C)) = f(L(G', h(C)))$. Since G' is a regular context-free grammar, $L(G', h(C))$ is in $\mathcal{L}$ [12] and so $f(L(G', h(C)))$ is semilinear. $\square$

Corollary 4.11.1. APL *has the Parikh property.*

We can give a characterization of FINITEVISIT($\mathscr{L}$) akin to the characterization of FINITEREVERSAL($\mathscr{L}$) in Theorem 3.3; however the languages involved are much less attractive.

For an alphabet $\Sigma = \{a_1, \ldots, a_n\}$, let $\bar{a}_1, \ldots, \bar{a}_n$ be n new matching symbols and $\bar{\Sigma} = \{\bar{a}_1, \ldots, \bar{a}_n\}$. For each w in $(\Sigma \cup \bar{\Sigma})^*$, let $\mu(w)$ be the minimal member of the equivalence class of w in the congruence relation generated by $a_i\bar{a}_i \sim e$. For a word w, let Init(w) be the set of all initial substrings of w. Let c be a new symbol. For $\mathrm{L} \subseteq \Sigma^*$ we can define a language

$$c(\mathrm{L}) = \{wcy \mid w \in \mathrm{L},\ y \in (\Sigma \cup \bar{\Sigma})^*,\ \mu(\mathrm{Init}(y)) \subseteq \mathrm{Init}(w)\},$$

and for each k

$$c_k(\mathrm{L}) = \{wcy \in c(\mathrm{L}) \mid \text{for } 1 \leq i \leq |w|,\ \#\{y' \in \mathrm{Init}(y) \mid |\mu(y')| = i\} \leq k\}.$$

It is shown in [9] that the family of one-way checking automaton languages is $\hat{\mathscr{M}}(c(\{a_1, a_2\}^*))$. Rodriguez provides a characterization of $2k$-reversal bounded one-way checking automaton languages in terms of a restriction of $c(\{a_1, a_2\}^*)$ [27]. By methods similar to those in [9] and [18] and in the proof of Theorem 3.3, one can establish the following result; the proof is left to the reader.

Theorem 4.12. *For any full semiAFL $\mathscr{L}$, and $k \geq 1$*

$$k\text{-VISIT}(\mathscr{L}) = \hat{\mathscr{M}}(\{c_k(\mathrm{L}) \mid \mathrm{L} \in \mathscr{L}\}).$$

For the particular case $\mathscr{L}$ = REGL one can show the following corollary.

Corollary 4.12.1. *The family of k-visit one-way checking automaton languages is* $\hat{\mathscr{M}}(c_k(\{a_1, a_2\}^*))$.

We proceed to establish two hierarchy theorems. First we give a lemma used in proving the strong hierarchy theorem for finite visit checking automata.

Lemma 4.13. *For $k \geq 1$, the language*

$$\mathrm{L}_{k+1} = \{c(a^m c)^{k+1} \mid m \geq 1\}$$

*is not in k-*VISIT(REGL).

Proof. Assume to the contrary that $\mathrm{L}_{k+1} = \mathrm{L}(M)$ for a strictly k-visit nonwriting machine $M = (K, \Sigma, \Gamma, \delta, q_0, F, R)$ with regular base R. We use the following simple combinatorial fact which we state without proof.

Sublemma. *There is no way to place $k+1$ connected line segments on a straight line in such a way that every two segments overlap in at least one point but no point is touched by all $k+1$ lines.*

Consider any accepting computation C of M on $c(a^m c)^{k+1}$ with working tape y. We can regard as line segment i the portion of y visited by M during the scan of the ith block of a's. Since the computation is k-visit bounded, there can be no point of y on which all $k+1$ line segments coincide. Hence by the sublemma there must be at least one pair, say r and s, $1 \leq r < s \leq k+1$, such that no square visited during the r^{th} block of a's is also visited during block s. We can factor y as either $y = y_r y_s$ or $y = y_s y_r$ such that during block r only y_r is visited and only y_s during block s.

Since M is strictly k-visit there is some t such that every accepting computation of m is t-crossing bounded. So there is a list S of up to $2t$ states of M giving the states in order for the up to t crossings of the border between y_r and y_s. Since R is regular, we can factor Γ^+ into some finite number of congruence classes $R_1, \ldots, R_n$ such that R is the union of some of these classes [22].

Hence we can associate with C a tuple (r, s, b, S, i_r, i_s) with $b =$ left if $y = y_r y_s$ and $b =$ right if $y = y_s y_r$ and $y_r \in R_{i_r}$, $y_s \in R_{i_s}$. There are only finitely many such tuples but infinitely many words $c(a^m c)^{k+1}$ accepted by M. Hence there are $m_1 \neq m_2$ and accepting computations C_i for $c(a^{m_i} c)^{k+1}$, $i = 1, 2$ such that C_1 and C_2 are associated with the same tuple. The cases are similar, so suppose they are associated with tuple $(r, s, \text{left}, S, i_r, i_s)$. This means that C_i has working tape $y_{ir} y_{is} \in R$, with $y_{ir} \in R_{i_r}$, $y_{is} \in R_{i_s}$ and visits only y_{ir} during block r and only y_{is} during block s, $i = 1, 2$. Thus $y = y_{1r} y_{2s}$ is also in R and M has an accepting computation C with working tape y for some word of the form $uca^{m_1} cvca^{m_2} cz$; this computation follows C_1 while visiting y_{1r} and C_2 while visiting y_{1s}. But this word cannot be in L_{k+1}, a contradiction. □

We can now summarize our hierarchy results for reversals and visits in checking automata.

Theorem 4.14. *For each* $k \geq 1$,

(1) k-REVERSAL(REGL) $\subsetneq$ $(k+1)$-REVERSAL(REGL),

(2) k-VISIT(REGL) $\subsetneq$ $(k+1)$-VISIT(REGL), *and*

(3) $(k+1)$-REVERSAL(REGL) $-$ k-VISIT(REGL) $\neq \emptyset$.

For $k \geq 3$

(4) k-REVERSAL(REGL) $\subsetneq$ k-VISIT(REGL),

(5) 3-VISIT(REGL) *and* $(k+1)$-REVERSAL(REGL) *are incomparable, and* 3-VISIT(REGL) *and* FINITEREVERSAL(REGL) *are incomparable.*

Proof. Theorem 3.11 gives us (1) since REGL $\neq$ 2-REVERSAL(REGL) $=$ LINCF. The language L_{k+1} of Lemma 4.13 is not in k-VISIT(REGL) but is obviously in $(k+1)$-REVERSAL(REGL). This establishes (2), (3) and part of (5). For (4) and the rest of (5), notice that the closure of 2-REVERSAL(REGL) $=$ LINCF under Kleene + is not in FINITEREVERSAL(REGL) by Theorem 3.11 but clearly is contained in 3-VISIT(REGL). □

Notice that 2-REVERSAL($\mathscr{L}$) = 2-VISIT($\mathscr{L}$) always since a 2-visit bounded computation is necessarily 2-reversal bounded. Thus we cannot improve Theorem 4.14(4). It also shows that while for any semiAFL $\mathscr{L}$, $\mathscr{L} \neq$ FINITEREVERSAL($\mathscr{L}$) if and only if $\mathscr{L} \neq$ 2-REVERSAL($\mathscr{L}$), we can have

$$\mathscr{L} = \text{2-VISIT}(\mathscr{L}) \neq \text{FINITEVISIT}(\mathscr{L}).$$

For example, take $\mathscr{L}$ = FINITEREVERSAL(REGL); then

$$\mathscr{L} = \text{2-REVERSAL}(\mathscr{L}) = \text{2-VISIT}(\mathscr{L})$$

but

$$\text{FINITEVISIT}(\mathscr{L}) = \text{FINITEVISIT(REGL)} \neq \mathscr{L}.$$

Corollary 4.14.1. *For each $k \geq 1$, k-APL $\subsetneq$ $(k+1)$-APL; that is, degree $(k+1)$ absolutely parallel grammars are strictly more powerful than degree k.*

We now turn to a general hierarchy theorem for k-VISIT($\mathscr{L}$). It is not a strong hierarchy theorem since it requires that $\mathscr{L}$ be closed under substitution. We need a syntactic lemma regarding special types of substitution.

Definition 4.15. *For languages* L_1 *and* L_2 *contained in* Σ^*, *let*

$$\tau(L_1, L_2) = \{a_1w_1 \cdots a_nw_n \mid a_1, \cdots, a_n \in \Sigma, a_1 \cdots a_n \in L_1, w_1, w_2, \ldots, w_n \in L_2\}.$$

A proof that FINITEVISIT($\mathscr{L}$) is closed under substitution whenever $\mathscr{L}$ is, can readily be obtained from the proof that the family of one-way checking automaton languages is closed under substitution [16]. Since we want bounds on how many visits are needed, we indicate briefly how the construction works.

Lemma 4.16. *Let $\mathscr{L}$ be a full semi*AFL *closed under substitution. Let* L_i *be in* k_i-VISIT($\mathscr{L}$), $L_i \subseteq \Sigma_i^*$, $i = 1, 2$, $\Sigma_1 \cap \Sigma_2 = \emptyset$. *Then* $L = \tau(L_1, L_2)$ *is in* $(k_1 + k_2 + 2)$-VISIT($\mathscr{L}$).

Proof. Let $L_i = L(M_i)$, M_i a strictly k_i-visit nonwriting fva with base R_i and working alphabet Γ_i, $i = 1, 2$. We can assume without loss of generality that $\Gamma_1 \cap \Gamma_2 = \emptyset$. We may as well also assume that each M_i is e-free since, by inserting dummy symbols (d_i in M_i), we can get e-free strictly k_i-visit nonwriting machines M_i' accepting L_i' such that $\tau(L_1, L_2)$ is obtainable from $\tau(L_1', L_2')$ by an a-transducer mapping, and $(k_1 + k_2 + 2)$-VISIT($\mathscr{L}$) is a full semiAFL. Let c, d, ¢, and \$ be new symbols.

We sketch the construction of a $(k_1 + k_2 + 2)$-visit bounded $\mathscr{L}$-based fva M_3 to accept $\tau(L_1, L_2)$. The new machine will not be nonwriting. The base for M_3 is

$$R_3 = ¢(cR_2c)^+\tau(R_1, (cR_2c)^+)\$.$$

Since $\mathcal{L}$ is substitution closed, R_3 is in $\mathcal{L}$. Now M_3's working tapes are tapes of M_1 with tapes of M_2 surrounded by c's interspersed everywhere.

Machine M_3 switches between an M_1 simulation and an M_2 simulation, starting with an M_1 simulation and ending with an M_2 simulation. We have assumed M_1 to be e-free so during the M_1 simulation M_3 reads one input from Σ_1 and executes one corresponding step ot M_1. To do so, it reads a working tape symbol of Γ_1.

Next M_3 gets ready for the M_2 simulation. If the M_1 step results in a right move, M_3 moves right to find the first block cyc, $y \in R_2$, turns the left c to d and starts simulating M_2 using y as its working tape and reading inputs only from Γ_2. If it sees c or d in a nonfinal state it blocks. If it sees c in an accepting state it changes c to d and moves right to the first symbol of Γ_1 to resume the M_1 simulation while if it sees d in the accepting state it does the same thing except it changes the first c to the right to d in the process (this is the right c in cyc now dyc and finally dyd). If the M_1 step on the other hand resulted in a left move, M_3 would move left to find the first cyc, $y \in R_2$, this time positioning itself on the leftmost symbol of y and turning both c's to d initially. Now it simulates M_2 on y with input from Σ_2 as in the previous case. When it finds itself in an accepting state of M_2 it moves left to the first Γ_1 symbol and resumes the M_1 simulation

If M_3 tries to resume the M_1 simulation in an accepting state of M_1 and finds no symbol of Γ_1 in the direction in which it is moving it accepts.

Each square containing a symbol from Γ_1 is visited only when simulating M_1 for a total of k_1 visits. A string cyc, $y \in R_2$ can be visited as M_3 sweeps over it from an M_2 simulation to an M_1 simulation. Since this corresponds to a visit of M_1 to the Γ_1 symbols to the left and right of cyc, this situation occurs at most k_1 times. String cyc can also be used for one and only one simulation of M_2 since after this simulation it becomes dyd and is ignored; this means at most k_2 more visits. However, if M_3 starts this M_2 simulation moving right and the computation is left exiting one more sweep right over y will be needed. If M_3 starts this M_2 simulation moving left, it first passes over cyc, turning it to dyd, in order to be on the leftmost symbol of y at the start, and if the simulated computation is right exiting it will have to go over dyd once more. This possible complication adds at most 2 extra visits per square. Hence M_3 is (k_1+k_2+2)-visit bounded. □

Remark. Lemma 4.16 implies that k_1-VISIT($\mathcal{L}$) $\hat{\sigma}$ k_2-VISIT($\mathcal{L}$) $\subseteq (k_1+k_2+2)$-VISIT($\mathcal{L}$) whenever $\mathcal{L}$ is closed under substitution. A more complicated construction shows that (k_1+k_2+1)-VISIT($\mathcal{L}$) suffices and if $\mathcal{L}$ is closed under reversal then (k_1+k_2)-VISIT($\mathcal{L}$) works. However k_1+k_2+2 is a good enough bound for our present purposes

Now we need a syntactic lemma which says that if $\tau(L_1, L_2)$ can be recognized within k visits, either L_1 is already in the base family or L_2 could be recognized within k-1 visits. This is shown by the sort of dichotomy argument found in [15, 16 and 19] and the proof will only be outlined.

Lemma 4.17. *Let* $L_1 \subseteq \Sigma_1^*$, $L_2 \subseteq \Sigma_2^*$ *with* $\Sigma_1 \cap \Sigma_2 = \emptyset$. *Let* \$ *be a new symbol. Let* $\mathscr{L}$ *be a full semi*AFL *and let* $k \geq 2$. *If* $\tau(L_1, L_2)\$$ *is in* k-VISIT($\mathscr{L}$) *then either* L_1 *is in* $\mathscr{L}$ *or* L_2 *is in* (k-1)-VISIT($\mathscr{L}$).

Proof. We use the following two sublemmas which we state without proof. They can be proven by standard arguments for checking automata and stack languages (cf. [16]); recall that $\mathscr{L}$ is a full semiAFL.

Sublemma 1. *Let M be an $\mathscr{L}$-based nonwriting preset Turing machine with input alphabet Σ. Suppose that for each $w = a_1 \cdots a_n \in$ L(M), $a_1, \ldots, a_n \in \Sigma$, there is an accepting computation C such that during C for each i, $1 \leq i \leq n-2$, no working tape square read during the scanning of a_i and subsequent e moves is read again after a_{i+2} is input. Then* L(M) *is in* $\mathscr{L}$.

Sublemma 2. *Let M be an $\mathscr{L}$-based nonwriting preset Turing machine with input alphabet Σ. Let $s \geq 1$. Suppose that for each $w = a_1 \cdots a_n \in$ L(M), $a_1, \ldots, a_n \in \Sigma$, there is an accepting computation C such that during the period from the input of a_1 through the input of a_n—but not necessarily including any e-moves before a_1 or after a_n—no working tape square is visited more than s times. Then* L(M) *is in s-*VISIT($\mathscr{L}$).

Now suppose that $L = \tau(L_1, L_2)\$$ is accepted by a strictly k-visit nonwriting $\mathscr{L}$-based *fva* $M = (K, \Sigma, \Gamma, \delta, q_0, F, R)$, $R \in \mathscr{L}$. We can define from M machines M_1 and M_2 for L_1 and L_2 as follows.

Let $M_1 = (K, \Sigma_1, \Gamma, \delta_1, q_0, F, R)$ for $\delta_1(q, a, A) = \delta(q, a, A)$, $a \in \Sigma_1$ and

$$\delta_1(q, e, A) = \bigcup_{b \in \Sigma_2 \cup \{\$, e\}} \delta(q, b, A).$$

Thus M_1 simply simulates M on inputs from Σ_1 and uses e-rules to simulate M on other inputs.

Let

$$M_2 = (K \times \{0, 1, 2, 3\}, \Sigma_2, \Gamma, \delta_2, (q_0, 0), F \times \{3\}, R),$$

where δ_2 is defined as follows. For $b \in \Sigma_2$ we have

$$\delta_2((q, 1), b, A) = \{((q', 1), A, s) \mid (q', A, s) \in \delta(q, b, A)\}.$$

Otherwise we have e-rules. For $(q, 0)$ we have

$$\delta_2((q,0), e, A) = \{((q',0), A, s) \mid (q', A, s) \in \delta(q, a, A), a \in \Sigma_1 \cup \Sigma_2 \cup \{e\}\}$$
$$\cup \{((q',1), A, s) \mid (q', A, s) \in \delta(q, a, A), a \in \Sigma_1\}.$$

For $(q, 1)$ we have

$$\delta_2((q,1), e, A) = \{((q',1), A, s) \mid (q', A, s) \in \delta(q, e, A)\}$$
$$\cup \{((q',2), A, s) \mid (q', A, s) \in \delta(q, a, A), a \in \Sigma_1\}$$
$$\cup \{((q',3), A, s) \mid (q', A, s) \in \delta(q, \$, A)\},$$

for $(q, 2)$

$$\delta_2((q,2), e, A) = \{((q',2), A, s) \mid (q', A, s) \in \delta(q, a, A), a \in \Sigma_1 \cup \Sigma_2 \cup \{e\}\}$$
$$\cup \{((q',3), A, s) \mid (q', A, s) \in \delta(q, \$, A)\},$$

and for $(q, 3)$

$$\delta_2((q,3), e, A) = \{((q',3), A, s) \mid (q', A, s) \in \delta(q, e, A)\}.$$

Essentially M_2 first uses e-rules to simulate M on any input, then, when M is about to start on a word in L_2 switches to real input (states $(q, 1)$) and when M returns to input from Σ_1 (or \$) continues the simulation with e input, making sure that \$ is read by M (states $(q, 2)$ and $(q, 3)$).

Clearly $L_i = L(M_i)$, $i = 1, 2$. We want to show that either M_1 satisfies the hypotheses of Sublemma 1 so $L_1 \in \mathscr{L}$ or M_2 satisfies the hypotheses of Sublemma 2 with $s = k-1$, so L_2 is in $(k-1)$-VISIT$(\mathscr{L})$.

Consider an accepting computation C for $a_1w_1 \cdots a_nw_n\$$, $a_1, \ldots, a_n \in \Sigma_1$, $w_1, \ldots, w_n \in L_2$. If no working tape square scanned during a_iw_i is scanned after w_{i+1} is through, for each i, $1 \leq i \leq n-2$, call C *locally 1-visit on* $a_1 \cdots a_n$. If no square is scanned more than $k-1$ times during the period from the input of the first symbol of w_i to the last, call *C locally* $(k-1)$*-visit on* (i, w_i); *if* $w_1 = w_2 = \cdots = w$, we just call *C locally k-visit on some occurrence of* w. Suppose C is not locally $(k-1)$-visit on (i, w_i) for each i, $1 \leq i \leq n$. During w_1 some square is visited k times. This square can never be seen again since M is strictly k-visit, setting up a "barrier". Similarly some square is seen k times during w_2, setting up a new barrier. Thus no square seen during a_1w_1 can be seen after w_2 is through. Arguing in this fashion, we see that for each i, $1 \leq i \leq n-1$, no square seen during a_iw_i can be seen after the scan of w_{i+1}. Hence C is locally 1-visit on $a_1 \cdots a_n$.

In particular, if C is an accepting computation on

$$a_1w \cdots a_nw\$ = \mu(a_1 \cdots a_n, w),$$

then either C is locally 1-visit on $a_1 \cdots a_n$ or C is locally $(k-1)$-visit on some occurrence of w. If for each $a_1 \cdots a_n$ in L_1 there is a w in L_2 and an accepting computation for $\mu(a_1 \cdots a_n, w)$, which is locally 1-visit on $a_1 \cdots a_n$, then clearly M_1 satisfies the hypotheses of Sublemma 1 and $L_1 \in \mathscr{L}$. Otherwise there is $a_1 \cdots a_n$ in L_1 such that for each w in L_2, no accepting computation of M on $\mu(a_1 \cdots a_n, w)$ is locally 1-visit on $a_1 \cdots a_n$ and so every accepting computation of M on $\mu(a_1 \cdots a_n, w)$ is locally $(k-1)$-visit on some occurrence of w (and there is at least one such accepting computation). Hence M_2 satisfies the hypotheses of Sublemma 2 and L_2 is in $(k-1)$-VISIT$(\mathscr{L})$. □

Now we can state our general hierarchy theorem for finite visit automata.

Theorem 4.18. *Let $\mathscr{L}$ be a substitution closed full* AFL. *If $\mathscr{L} \neq$* FINITEVISIT$\mathscr{L}$), *then there is an integer $k_0 \geq 4$ such that for all $k \geq 1$,*

$$k\text{-VISIT}(\mathscr{L}) \subsetneq (k+k_0)\text{-VISIT}(\mathscr{L}).$$

Proof. If $\mathscr{L} \neq$ FINITEVISIT$(\mathscr{L})$ then for some $k_1 \geq 2$,

$$\mathscr{L} = 1\text{-VISIT}(\mathscr{L}) \subsetneq k_1\text{-VISIT}(\mathscr{L}).$$

Let $k_0 = k_1 + 2$. Clearly

$$k\text{-VISIT}(\mathscr{L}) \subsetneq (k+k_0)\text{-VISIT}(\mathscr{L}) \quad \text{for } k = 1.$$

Suppose $k \geq 2$ is the smallest integer such that

$$k\text{-VISIT}(\mathscr{L}) = (k+k_0)\text{-VISIT}(\mathscr{L}).$$

Let L_1 be in k_1-VISIT$(\mathscr{L}) - \mathscr{L}$. Consider any member L_2 of k-VISIT$(\mathscr{L})$. Since all our families are closed under renaming we can assume that the vocabularies of L_1 and L_2 are disjoint and that \$ is a new symbol. By lemma 4.16, $\tau(L_1, L_2)$ and so $\tau(L_1, L_2)\$$ is in

$$(k+k_1+2)\text{-VISIT}(\mathscr{L}) = (k+k_0)\text{-VISIT}(\mathscr{L}) = k\text{-VISIT}(\mathscr{L}).$$

Since L_1 is not in $\mathscr{L}$, by Lemma 4.17, L_2 is in $(k-1)$-VISIT$(\mathscr{L})$. As L_2 was an arbitrary member of k-VISIT$(\mathscr{L})$ we see that

$$k\text{-VISIT}(\mathscr{L}) \subseteq (k-1)\text{-VISIT}(\mathscr{L})$$

and thus

$$\begin{aligned}(k-1)\text{-VISIT}(\mathscr{L}) = k\text{-VISIT}(\mathscr{L}) &= (k+k_0-1)\text{-VISIT}(\mathscr{L}) \\ &= (k+k_0)\text{-VISIT}(\mathscr{L}).\end{aligned}$$

This contradicts the minimality of k. Hence we conclude that

$$k\text{-VISIT}(\mathscr{L}) \subsetneq (k+k_0)\text{-VISIT}(\mathscr{L}) \quad \text{for all } k \geq 1. \quad \square$$

Corollary 4.18.1. *For any substitution closed full* AFL $\mathscr{L}$, *if* $\mathscr{L} \neq$ CONTROL(APG, $\mathscr{L}$), *then there is a* $k_0 \geq 2$ *such that*

$$\text{CONTROL}(k\text{-APG}, \mathscr{L}) \subsetneq \text{CONTROL}((k+k_0)\text{-APG}, \mathscr{L})$$

for each $k \geq 1$.

We can provide a somewhat different extension of Theorem 4.14 by considering iterative properties of languages.

Definition 4.19. A language L is *k-iterative* if there is a $k_1 \geq 1$ such that whenever w is in L and $|w| > k_1$, then $w = u_1 v_1 \cdots u_k v_k u_{k+1}$ for $v_1 \cdots v_k \neq e$ and $u_1 v_1^n \cdots u_k v_k^n u_{k+1}$ is in L for all $n \geq 0$. A language L is *weakly k-iterative* if it is either finite or contains a k-iterative subset. A family of languages $\mathscr{L}$ is *k-iterative* (*weakly k-iterative*) if every member of $\mathscr{L}$ is k-iterative (weakly k-iterative).

We shall sketch the proof that if $\mathscr{L}$ is a weakly k-iterative full semiAFL, then r-VISIT($\mathscr{L}$) is weakly kr-iterative. First we need an auxiliary definition and lemma which says in essense that finite visit automata can be made to operate in linear time and space in a strong way.

Definition 4.20. Let C be a computation of fva M with working tape y and let $1 \leq i \leq |y|$. We say that C *skips* i if every visit to square i results in an e-rule transition. If for every i, $1 \leq i \leq |y|$, C does not skip i, then C is *nonskipping*. If every accepting computation of M for nonempty input is nonskipping, then M is *nonskipping*.

In a nonskipping computation, every working tape square must be visited at least once while input is read and the input tape advanced. An $\mathscr{L}$-based k-visit bounded automaton can be made nonskipping, employing standard arguments of the type used for checking automata and stack automata. Essentially one can use an a-transducer to replace a portion of working tape "skipped" during a computation by a table summarizing the behavior of the machine on that portion for e-input. The proof is sketched in Section 5.

Lemma 4.21. *Let* $\mathscr{L}$ *be a full semi*AFL *and let* L *be in* k-VISIT($\mathscr{L}$). *There is a nonwriting, right touching, nonskipping* $\mathscr{L}$*-based k-visit bounded automaton M such that* L $=$ L(M), *and every accepting computation of M is k-visit bounded.*

Now we sketch the proof of the weak iteration lemma.

Lemma 4.22. *Let* $\mathscr{L}$ *be a weakly k-iterative full semi*AFL. *Then* r-VISIT($\mathscr{L}$) *is weakly kr-iterative.*

Proof. We need only consider a nonwriting, right touching, nonskipping, $\mathscr{L}$-based r-visit bounded automaton $M=(K,\Sigma,\Gamma,\delta,q_0,F,\mathrm{L})$, with $\mathrm{L}\in\mathscr{L}$, and every accepting computation of M r-visit bounded.

Define the sigma symbols and associated functions and sets R_1 and R_2 for M as in the proof of Theorem 4.1 and let

$$R=\{x \mid ¢x\$\in R_1\cup R_2\}.$$

Let $\mathrm{L}_1=h^{-1}(\mathrm{L})\cap R$. As before, L_1 consists of encodings of working tapes of L along with complete accepting computations using these tapes. Every accepting computation of M is encoded in L_1 and the corresponding input can be obtained by appropriate decoding. Further, R is regular and so L_1 is in $\mathscr{L}$.

If L_1 is finite, $\mathrm{L}(M)$ certainly is finite. Otherwise L_1 contains a subset $I=\{u_1v_1^n\cdots v_s^nu_{s+1}\mid n\geqslant 0\}$ with $1\leqslant s\leqslant k$ and every v_i nonempty. Let

$$I_r=\{u_1v_1^{r+n}\cdots u_sv_s^{r+n}u_{s+1}\mid n\geqslant 1\}.$$

We wish to argue that I_r encodes accepting computations for an rs-iterative (and hence rk-iterative) subset of $\mathrm{L}(M)$.

Notice that since v_i encodes actions on $h(v_i)$, identical actions can be taken by M on each repetition of $h(v_i)$ corresponding to repetitions of v_i in words of I_r. Consider the computation C on $h(u_1v_1^{r+1}\cdots u_sv_s^{r+1}u_{s+1})$ encoded by $u_1v_1^{r+1}\cdots u_sv_s^{r+1}u_{s+1}$. Suppose $h(v_i)$ is first entered from the left in a state p. Thus the first entry into each repetition of $h(v_i)$ is also in state p and the same inputs and actions are involved. Since M is right touching, M will eventually cross $(h(v_i))^{r+1}$. There are two possibilities. If M crosses $h(v_i)$ directly, without going left of this subtape using input x_1, then repetition of input x_1 causes M to cross all of $(h(v_i))^{r+n}$ for any n. Otherwise, M may have some wiggles left before crossing, and may repeatedly go left of $h(v_i)$. But M is r-visit bounded. Hence there must be some repetition of $h(v_i)$ (before the $r+1^{st}$) such that C enters it in p and enters the next segment $h(v_i)$ in p under some input x_1 without leaving $(h(v_i))^{r+1}$. Thus further iterations of $h(v_i)$ can also be crosssed using x_1 and so x_1 is one of the iterative factors sought. Similar arguments apply to each left entry to the first $h(v_i)$ and each right entry to the last one in C. Since there are at most r such entries and crossings of $((h(v_i))^{r+1}$, we identify at most r iterative factors. Since M is nonskipping, at least one is nonempty. This holds for each i. Thus I_r describes an rs-iterative subset of $\mathrm{L}(M)$. □

The point is that the language L_{k+1} of Lemma 4.13 is $(k+1)$-iterative but not k-iterative.

Theorem 4.23. *Let $\mathscr{L}$ be a weakly k-iterative full semi*AFL. *Let* $r\geqslant 1$. *Then*:

(1) r-VISIT$(\mathscr{L})\subsetneqq(kr+1)$-VISIT$(\mathscr{L})$,

(2) $(kr+1)$-REVERSAL$(\mathscr{L})-r$-VISIT$(\mathscr{L})\neq\emptyset$, *and*

(3) *if* L_k *is in* $\mathscr{L}$, *then* $(r+1)$-REVERSAL$(\mathscr{L})-r$-VISIT$(\mathscr{L})\neq\emptyset$ *for each* $r\geqslant 1$.

Proof. On one hand, L_{kr+1} is in

$$(kr+1)\text{-REVERSAL(REGL)} \subseteq (kr+1)\text{-REVERSAL}(\mathscr{L}) \subseteq (kr+1)\text{-VISIT}(\mathscr{L}).$$

On the other hand, since L_{kr+1} is not weakly kr-iterative, it cannot be in r-VISIT($\mathscr{L}$). This establishes (1) and (2). If L_k is in $\mathscr{L}$, then using it as a base language we can certainly get L_{kr+1} using $r+1$ reversals; this yields (3). □

Corollary 4.23.1. *If $\mathscr{L}$ is any full semi*AFL *contained in* CF,

$$r\text{-VISIT}(\mathscr{L}) \subsetneq (2r+1)\text{-VISIT}(\mathscr{L}) \quad \textit{for each } r \geq 1$$

and if $\{a^nb^n \mid n \geq 1\}$ is in $\mathscr{L}$, then

$$r\text{-VISIT}(\mathscr{L}) \subsetneq (r+1)\text{-VISIT}(\mathscr{L}) \quad \textit{for each } r \geq 1.$$

Proof. Every context-free language is obviously 2-iterative in a very strong way. Clearly L_{2r+1} can be obtained from $\{a^nb^n \mid n \geq 1\}$ using $r+1$ reversals. □

We conclude this section by observing that Lemma 4.17 can also be used to provide an alternative proof[9] that not all context-free languages are checking automaton languages. Let CAL be the family of one-way checking automaton languages and NESA the family of one-way nonerasing stack languages.

Lemma 4.17 tells us that any substitution closed full principal semiAFL contained in FINITEVISIT($\mathscr{L}$) must be in $\mathscr{L}$, so CF cannot be contained in FINITEVISIT(REGL). But every language in CAL—FINITEVISIT(REGL) must be weakly 1-iterative, while there are generators of CF which do not contain an infinite regular set.

More formally, we derive the next lemma from Lemma 4.17.

Lemma 4.24. *Let $\mathscr{L}$ be a full semi*AFL *and let $\mathscr{L}_1$ be a substitution closed full principal semi*AFL. *If $\mathscr{L}_1 \subseteq$ FINITEVISIT($\mathscr{L}$), then $\mathscr{L}_1 \subseteq \mathscr{L}$.*

Proof. Let $\mathscr{L}_1 = \hat{\mathscr{M}}(\mathrm{L})$. If $\mathrm{L} \in 1$-VISIT($\mathscr{L}$) $= \mathscr{L}$, we are done. Otherwise, for some $k \geq 2$, $\mathrm{L} \in k$-VISIT($\mathscr{L}$)$-(k-1)$-VISIT($\mathscr{L}$). Let L_1 be a renaming of L in a new vocabulary (i.e., $\mathrm{L}_1 = h(\mathrm{L})$ for a one-one length-preserving homomorphism h) and let \$ be new. Since $\mathscr{L}_1$ is substitution closed, $\tau(\mathrm{L}, \mathrm{L}_1)\$ \in \hat{\mathscr{M}}(\mathrm{L}) \subseteq k$-VISIT($\mathscr{L}$). This contradicts Lemma 4.17, since $\mathrm{L} \notin \mathscr{L}$ and $\mathrm{L}_1 \notin (k-1)$-VISIT($\mathscr{L}$). Hence $\mathrm{L} \in \mathscr{L}$. □

Lemma 4.25. *Let $\mathscr{L}$ be a full semi*AFL. *If* L *is not in* FINITEVISIT($\mathscr{L}$) *but can be accepted by a nonwriting one-way preset $\mathscr{L}$-based Turing machine, then* L *is weakly 1-iterative.*

[9] See [43].

Proof. Let $L = L(M)$, where $M = (K, \Sigma, \Gamma, \delta, q_0, F, R)$ is a preset one-way $\mathscr{L}$-based Turing machine. Let $k = 1 + \#K$. Since L is not in k-VISIT($\mathscr{L}$), there is a word w in L such that no accepting computation of M on input w is k-visit bounded. Let C' be the shortest accepting computation of M on input w, and let C have working tape y. During C, M must visit some square in y at least k times and thus twice in the same state q. Hence $w = uavbz$, $a, b \in \Sigma \cup \{e\}$, $y = y_1Ay_2$, $A \in \Gamma$ and C includes visits to A in state q at a and b. Since C is the shortest accepting computation, $av \neq e$. Hence $u(av)^*bz$ is an infinite regular subset of L. □

Theorem 4.26. *There are context-free languages which are not one-way nonerasing stack languages.*

Proof. Since CF is substitution closed, Lemma 4.1 of [16] allows us to conclude that CF $\subseteq$ NESA if and only if CF $\subseteq$ CAL.

Now CF $= \hat{\mathscr{M}}$(L) for a parenthesis language L which is not weakly 1-iterative [36]. By Lemma 4.25, L is not in CAL − FINITEVISIT(REGL). By Lemma 4.24, L is not in FINITEVISIT(REGL). Hence L is not in CAL, so L $\in$ CF − CAL and L $\in$ CF − NESA. □

Remark. The same argument shows that some index languages cannot be expressed as nondeterministic two-way finite state transductions of stack languages[10].

5. Complexity questions

Upper bounds on the space or time complexity of FINITEVISIT($\mathscr{L}$) can be obtained by applying the results for the two-way case and using special considerations pertinent to the one-way case.

The space and time complexity of two-way finite visit automata was studied in [20]. It wa shown in particular that

$$\text{TWOFINITEVISIT(REGL)} = \text{NSPACE}(\log_2 n),$$

the class of languages accepted in space $\log_2 n$ by off-line nondeterministic multitape Turing machines and TWOFINITEVISIT(CF) $= \mathscr{P}$, the class of languages accepted in polynomial time by deterministic off-line multitape Turing machines. Corresponding characterizations do not exist in the one-way case. For example, FINITEVISIT(REGL) is incomparable with the family of languages accepted by on-line nondeterministic multitape Turing machines in space $\log_2 n$ since the former is closed under homomorphism while the homomorphic image of the latter is the family of recursively enumerable sets and the former contains $\{wcw^R \mid w \in \{a, b\}^*\}$ which the latter does not. We state the obvious restrictions of the two-way results.

[10] Kiel [42] proved variants of Corollary 4.1.1, Theorem 4.11 and Lemma 4.25 in a different model.

Theorem 5.1. FINITEVISIT(REGL) = APL $\subsetneq$ NSPACE($\log_2 n$),

$$\text{FINITEVISIT (CF)} \subsetneq \mathcal{P}.$$

Corollary 5.1.1. *The family of derivation bounded languages is contained in* NSPACE($\log_2 n$).

Proof. Clearly DB $\subseteq$ APL [25]. □

Corollary 5.1.1 strengthens Sudborough's observation that all linear context-free languages are in NSPACE($\log_2 n$) [30].

Corollary 5.1.2.

$$\text{CONTROL}_\infty(\text{LDBG, REGL}) \subsetneq \text{NSPACE}(\log_2 n),$$

$$\text{CONTROL}_\infty(\text{LDBG, DB}) \subsetneq \text{NSPACE}(\log_2 n),$$

$$\text{CONTROL}_\infty(\text{LDBG, CF}) \subsetneq \mathcal{P}.$$

Proof. For a left derivation bounded grammar G and control set C one can clearly find an npg G and control set C' in $\hat{\mathcal{M}}(C)$ with $L(G, C) = L(G', C')$. □

We can obtain further complexity results in the one-way case from the fact that one-way finite visit automata can be made to operate in linear time and space. If a k-visit bounded computation on input $w \neq e$ and working tape y is nonskipping, then $|y| \leq |w|$ and the computation takes at most $k|y| \leq k|w|$ steps. Hence, we rephrase Lemma 4.21 and sketch the proof.

Lemma 5.2. *Let $\mathcal{L}$ be a full semi*AFL *and let* L *be in k*-VISIT($\mathcal{L}$). *There is a k-visit bounded nonwriting $\mathcal{L}$-based automaton $\hat{M}$ such that* L = L($\hat{M}$), *$\hat{M}$ is nonskipping and, for every w in* L *and every accepting computation C of $\hat{M}$ on w with working tape y, C is k-visit bounded,* $1 \leq |y| \leq \text{Max}(|w|, 1)$ *and C takes at most k* $\cdot$ Max($|w|$, 1) *steps*

Proof. Let L = L(M), where $M = (K, \Sigma, \Gamma, \delta, q_0, F, L_1)$ is a nonwriting strictly k-visit $\mathcal{L}$-based automaton. Further, M enters an accepting state only on a left or right exit.

For y in Γ^+, the table for y is the $0-1$ function T_y on $K \times (\Sigma \cup \{e\}) \times \{L, R\} \times \{L, R\} \times K$ such that $T_y(p, a, i, j, q) = 1$ if and only if

$$(p, a, d(i)) \overset{*}{\vdash} (q, e, y, d(j)),$$

where $d(L)=1$ and $d(R)=|y|+1$. The relation on Γ^+, defined by $x \equiv y$ if and only if $T_x = T_y$, is clearly a congruence relation of finite index[9] on Γ^+ so, by Nerode's Theorem [22], for each table T the set $\{y \in \Gamma^+ \mid T_y = T\}$ is regular. Let

$$\mathcal{T} = \{(T, i, j) \mid T \text{ is a table, } 1 \leq i \leq k+1, 1 \leq j \leq k\},$$

$$\mathcal{T}_1 = \{(T, i, j) \in \mathcal{T} \mid i = 1\},$$

and

$$\mathcal{T}_e = \{(T, 0) \mid T \text{ is a table}\}.$$

There is an a-transducer M_1 taking Γ^+ into subsets of $\mathcal{T}_1^+ \cup \mathcal{T}_e$, such that $S_1 \cdots S_r \in M_1(y) \cap \mathcal{T}_1^+$ if and only if $y = y_1 \cdots y_r$ and $T_{y_i} = S_i$, $1 \leq i \leq r$ and $(S, 0) \in M_1(y) \cap \mathcal{T}_e$ if and only if $T_y = S$. The new machine $\bar{M}$ has base $L_2 = M_1(L_1)$ and tape vocabulary $\Gamma_1 = \mathcal{T} \cup \mathcal{T}_e$.

A tape $(T, 0)$ in L_2 is used only for input e, which is accepted if and only if, for some q in F and d in $\{L, R\}$, $T(q_0, e, d, L, q) = 1$. Otherwise, $\bar{M}$ has working tape symbols of the form (T, i, j) and states of the form (q, L) or (q, R). A symbol $(T, k+1, j)$ causes a block. For $i \leq k$, $\bar{M}$ has a transition from state (q, d) to state (q', d') changing (T, i, j) to $(T, i+1, j)$ and moving in direction d' on input $b \in \Sigma \cup \{e\}$ if and only if $T(q, b, d, d', q') = 1$ and if $i = j$, then $b \neq e$. The start state is (q_0, L) and the final state set is $\{(q, L), (q, R) \mid q \in F\}$.

Clearly $\bar{M}$ is nonskipping since the j^{th} visit to (T, i, j) must be on nonempty input. The behavior of $\bar{M}$ on input w and working tape $(S_1, 1, j_1) \cdots (S_r, 1, j_r)$ simulates a computation of M on input w and working tape $u_1 A_1 v_1 \cdots u_r A_r v_r$, $A_i \in \Gamma$, such that the u_i and v_i (which may be empty) are "skipped" but nonempty input is read at least on the j_i^{th} visit to A_i. So $L = L(\bar{M}) = L(M)$.

Machine $\bar{M}$ is not nonwriting. However, the conversion to a nonwriting machine can be accomplished without adding new e-rules. □

Now we observe that the one-way FINITEVISIT operator preserves nondeterministic time and space and deterministic space complexities for full semiAFL's in a certain sense. Let us assume that all bounding functions are monotonic functions from the nonnegative integers into the nonnegative integers (so, e.g., $\log_2 n$ means $\lfloor \log_2 n \rfloor$).

Definition 5.3. Let NTIME($T(n)$)(NSPACE($S(n)$)) be the family of languages accepted by nondeterministic multitape Turing machines in time $T(n)$(space $S(n)$)); let DTIME($T(n)$)(DSPACE($S(n)$)) be the family of languages accepted by deterministic multitape Turing machines in time $T(n)$(space $S(n)$).

[9] A relation $R \subseteq S \times S$ is a *congruence relation* if it is an equivalence relation and for all x, y, z, w in S, (x, y) and (z, w) in R imply (xz, yw) in R; it is of *finite index* on S if it partitions S into finitely many equivalence classes.

Theorem 5.4. *Let $\mathscr{L}$ be a full semi*AFL.

(1) *If $\mathscr{L} \subseteq \mathrm{NTIME}(T(n))$, and $T(n) \geq n$ for all n, then*

$$\mathrm{FINITEVISIT}(\mathscr{L}) \subseteq \mathrm{NTIME}(T(n)).$$

(2) *If $\mathscr{L} \subseteq \mathrm{NSPACE}(S(n))$ for $S(n) \geq n$ a.e., then*

$$\mathrm{FINITEVISIT}(\mathscr{L}) \subseteq \mathrm{NSPACE}(S(n)).$$

(3) *If $\mathscr{L} \subseteq \mathrm{DSPACE}(S(n))$ for $S(n) \geq n$ a.e. then*

$$\mathrm{FINITEVISIT}(\mathscr{L}) \subseteq \mathrm{DSPACE}(S(n)).$$

Proof. For L in k-VISIT($\mathscr{L}$), we let $\mathrm{L} = \mathrm{L}(M_1)$ for nonskipping fva M_1 with base C in $\mathscr{L}$ satisfying the conclusions of Lemma 5.2. If $C \in \mathrm{NTIME}(T(n))$, let multitape Turing machine M_2 accept C in time $T(n)$. We construct a multitape Turing machine M for L. On input w, M writes down on one of its tapes a guess at a working tape y with $|y| \leq \mathrm{Max}(|w|, 1)$. Next M simulates M_2 to determine whether y is in C. If y is in C, M concludes by simulating M_1 for input w and working tape y. Hence if w is in L, some accepting computation of M takes at most $(k+2)|y| + T(|y|) \leq (k+3)T(|w|)$ steps. Thus $\mathscr{L}$ is in $\mathrm{NTIME}((k+3)T(n))$. Since $\mathrm{NTIME}(T(n))$ has linear speedup [3], L is in $\mathrm{NTIME}(T(n))$. This establishes (1). The argument for (2) is similar.

To obtain (3), we note that a deterministic Turing machine M can cycle through all those computations of M_1 on input w which use working tape bounded in length by $\mathrm{Max}(|w|, 1)$ and take at most $k|w|$ steps; furthermore, M needs at most $|w|$ tape squares for this process (this is essentially the proof that $\mathrm{DSPACE}(S(n))$ is closed under nonerasing homomorphism for $S(n) \geq n$ a.e.). □

A few corollaries of Theorem 5.4 follow, letting STACK be the family of one-way stack languages.

Corollary 5.4.1. $\mathrm{FINITEVISIT(CF)} \subsetneq \mathrm{NTIME}(n)$.

Corollary 5.4.2. $\mathrm{FINITEVISIT(STACK)} \subsetneq \mathrm{NTIME}(n^2)$.

Corollary 5.4.3. $\mathrm{FINITEVISIT(STACK)} \subsetneq \mathrm{DSPACE}(n)$.

It was shown in [20] that, if $\mathscr{L}$ is a full semiAFL whose members are accepted by one-way nondeterministic machines of the type $\mathscr{D}$, then every member of k-VISIT($\mathscr{L}$)(k-REVERSAL($\mathscr{L}$)) can be accepted by a two-way $(k+1)$-head (k-head) nondeterministic machine of type $\mathscr{D}$. Moreover, the simulation of a one-way k-visit (k-reversal) bounded machine acting on input w and working tape y took at most $r|w| \cdot |y|$ steps ($r(|y| + |w|)$ steps) for some constant r. Hence using Lemma 5.2 we have the following metatheorem.

Theorem 5.5. *If $\mathscr{L}$ is a full semi*AFL *whose members are accepted by one-way one-head nondeterministic machines of type $\mathscr{D}$ in time $T(n)$ and space $S(n)$, then every member of k-*VISIT($\mathscr{L}$) (k-REVERSAL($\mathscr{L}$)) *can be accepted by a two-way $(k+1)$-head (k-head) nondeterministic machine of type $\mathscr{D}$ in time $nT(n)$ (time $T(n)$) and space $S(n)$.*

Corollary 5.5.1. *Every member of k-*VISIT(REGL) (k-REVERSAL($\mathscr{L}$)) *can be accepted by a nondeterministic $(k+1)$-head (k-head) finite state acceptor in time n^2 (time n).*

Corollary 5.5.2. *Every member of k-*VISIT(CF) (k-REVERSAL(CF)) *can be accepted by a nondeterministic k-head pushdown store acceptor in time n^2 (time n).*

Cook [34] showed that all languages accepted by multihead pda in polynomial time are in DSPACE($(\log_2 n)^2$).

Corollary 5.5.3. FINITEVISIT(CF) $\subsetneq$ DSPACE($(\log_2 n)^2$).

Remark. Arora and Sudborough [32] showed that CONTROL$_\infty$(LINCFG, CF) $\subseteq$ DSPACE($(\log_2 n)^2$), while Erni [35] extended this to $\mathscr{F}$(FINITEREVERSAL(CF)).

6. Summary and open questions

We have seen that under both the finite reversal and the finite visit restriction writing and nonwriting preset Turing machines have the same power. Characterizations were provided for k-REVERSAL($\mathscr{L}$) and k-VISIT($\mathscr{L}$) in terms of operations on languages (Theorems 3.3 and 4.12) and control sets on grammars (Theorems 3.8 and 4.8). Both FINITEREVERSAL and FINITEVISIT are idempotent operators on families of languages (Theorems 3.4 and 4.1); the proof for reversal bounds gives us the decomposition theorem

$$k\text{-REVERSAL}(r\text{-REVERSAL}(\mathscr{L}) = kr\text{-REVERSAL}(\mathscr{L}).$$

Finally we established hierarchy theorems which state that under fairly general conditions, if $\mathscr{L}$ is not closed under FINITEREVERSAL or FINITEVISIT then increasing the number of reversals or visits increases the class of languages defined; for reversals this followed from established results on homomorphic replication while different arguments were needed for visit bounds. The hierarchy theorem for reversals also showed that in general visit bounds are more powerful than reversal bounds.

Some of these results have conditions which are needed for the particular proof method employed but perhaps might be removed with other techniques. In Theorem 3.12 we showed that FINITEREVERSAL($\mathscr{L}$) is properly contained in FINITEVISIT($\mathscr{L}$) for any full AFL $\mathscr{L}$ with $\mathscr{L} \neq$ FINITEVISIT($\mathscr{L}$). We conjecture that $\mathscr{L}$ does not have to be an AFL and the result should hold for full semiAFL's. For visits we showed that if $\mathscr{L}$ is a substitution closed full AFL with $\mathscr{L} \neq$ FINITE-VISIT($\mathscr{L}$), then there is an integer k_0 such that $k+k_0$ visits are more powerful than k for all k. We conjecture that the substitution closed condition is unnecessary and that one should always be able to take $k_0 = 3$.

No decomposition or padding theorem was given for k-VISIT($\mathscr{L}$) akin to those for reversals in Theorem 3.3 and Lemma 3.9. This is one reason why translational techniques could not be used to further strengthen the hierarchy theorem. Although exact analogs of Theorem 3.3 and Lemma 3.9 are probably not true, some results of that nature would be useful.

One curious open question is the relationship between CONTROL$_\infty$(LDBG, $\mathscr{L}$) and CONTROL(APG, $\mathscr{L}$) = FINITEVISIT($\mathscr{L}$). It is clear that CONTROL$_\infty$ (LDBG, $\mathscr{L}$) is contained in CONTROL(APG, $\mathscr{L}$): are they equal? We saw that

$$\text{CONTROL(APG, CONTROL(APG}, \mathscr{L})) = \text{CONTROL(APG}, \mathscr{L}).$$

A similar result cannot hold for LDBG since obviously LDB = CONTROL (LDBG, REGL) is properly contained in CONTROL$_2$(LDBG, REGL). It can be shown that for any full semiAFL $\mathscr{L}$ with $\mathscr{L} \subseteq$ CF, and any $k > 1$,

$$\text{CONTROL}_k(\text{LDBG}, \mathscr{L}) \subsetneq \text{CONTROL}_{k-1}(\text{LDBG}, \mathscr{L}).$$

However, is there a full semiAFL and a $k \geq 2$ such that $\mathscr{L} \neq$ CONTROL(LDBG, $\mathscr{L}$) but

$$\text{CONTROL}_k(\text{LDBG}, \mathscr{L}) = \text{CONTROL}_\infty(\text{LDBG}, \mathscr{L})?$$

In Section 5, we listed several complexity results for FINITEVISIT($\mathscr{L}$). In particular, we could establish some of the same upper bounds for FINITE-VISIT(CF) as for CF, namely, membership in DSPACE($(\log_2 n)^2$) and NTIME(n). We have FINITEVISIT(CF) $\subsetneq \mathscr{P}$. It seems plausible to conjecture that languages in FINITEVISIT(CF) can be accepted deterministically in time n^3. One might ask whether FINITEVISIT(REGL) is contained in DSPACE($\log_2 n$), but that would imply DSPACE($\log_2 n$) = NSPACE($\log_2 n$) [30].

References

[1] B.S. Baker and R.V. Book, Reversal-bounded multipushdown machines, *J. Comput. System Sci.* **8** (1974) 315–332.

[2] R.B. Banerji, Phrase structure languages, finite machines and channel capacity, *Information and Control* **6** (1963) 153–162.

[3] R.V. Book and S.A.Greibach, Quasi-realtime languages, *Math. Systems Theory* **4** (1970) 97–111.

[4] N. Chomsky, On certain formal properties of grammars, *Information and Control* **2** (1959) 137–167.

[5] R.W. Ehrich and S.S. Yau, Two-way sequential transductions and stack automata, *Information and Control* **18** (1971) 404–446.

[6] C.C. Elgot and J.E. Mezei, On relations defined by generalized finite automata, *IBM J. Res. Develop.* **9** (1975) 47–68.

[7] P.C. Fischer, The reduction of tape reversal for off-line one-tape turing machines, *J. Comput. System Sci.* **2** (1968) 136–147.

[8] S. Ginsburg and S.A. Greibach, Abstract families of languages, in: Ginsburg, Greibach, and Hopcroft, *Studies in Abstract Families of Languages, Mem. Amer. Math. Soc.* **87** (1969) 1–32.

[9] S. Ginsburg and S. Greibach, On AFL generators for finitely encoded AFA, *J. Comput. System Sci.* **7** (1973) 1–27.

[10] S. Ginsburg and S.A. Greibach, Principal AFL, *J. Comput. System Sci.* **4** (1970) 308–338.

[11] S. Ginsburg and E.H.Spanier, AFL with the semilinear property, *J. Comput. System Sci.* **5** (1971) 365–396.

[12] S. Ginsburg and E.H. Spanier, Control sets on grammars, *Math. Systems Theory* **2** (1968) 159–178.

[13] S. Ginsburg and E.H. Spanier, Derivation-bounded languages, *J. Comput. System Sci.* **2** (1968) 228–250.

[14] S. Ginsburg and E.H. Spanier, Finite-turn pushdown automata, *SIAM J. Control* **4** (1966) 429–453.

[15] S.A. Greibach, Chains of full AFL's *Math. System Theory* **4** (1970) 231–242.

[16] S.A. Greibach, Checking automata and one-way stack languages, *J. Comput. System Sci.* **3** (1969) 196–217.

[17] S.A. Greibach, Control sets on context-free grammar forms, *J. Comput. System Sci.* (to appear).

[18] S.A. Greibach, An infinite hierarchy of context-free languages, *J. Assoc. Comput. Mach.* **16** (1969) 91–106.

[19] S.A. Greibach, Syntactic operators on full SemiAFLs, *J. Comput. System Sci.* 6 (1972) 30–76.

[20] S.A. Greibach, Visits, crosses and reversals for nondeterministic offline machines, *Information and Control* (to appear).

[21] J. Hartmanis, Tape-reversal bounded Turing machine computations, *J. Comput. System. Sci.* **2** (1968) 117–35.

[22] A. Nerode, Linear Automata Transformations, *Proc. Amer. Math. Soc.* **9** (1958) 541–544.

[23] R.J. Parikh, On context-free languages, *J. Assoc. Comput. Mach.* **13** (1966) 570–581.

[24] M.O. Rabin and D. Scott, Finite automata and their decision problems, *IBM J. Res. Develop.* **3** (1959) 114–125.

[25] V. Rajlich, Absolutely parallel grammars and two-way finite-state transducers, *J. Comput. System Sci.* **6** (1972) 324–342.

[26] V. Rajlich, Bounded-crossing transducers, *Information and Control* **27** (1975) 329–335.

[27] F. Rodriguez, Une double hiérarchie infinie de langages vérifiables, *Rev. Française Automat. Informat. Recherche Opérationnelle* Sér. R-1 **9** (1975) 5–20.

[28] R. Siromoney, Finite-turn checking automata, *J. Comput. System Sci.* **5** (1971) 549–559.

[29] R. Siromoney, On equal matrix languages, *Information and Control* **14** (1969) 135–151.

[30] I. H. Sudborough, A note on tape-bounded complexity classes and linear context-free languages, *J. Assoc. Comput. Mach.* **22** (1975) 499–500.

[31] S. J. Walljasper, Left-derivation bounded languages, *J. Comput. System Sci.* **8** (1974) 1–7.

[32] A. Arora and I.H. Sudborough, On languages log-tape reducible to context-free languages, *Proc. 1976 Conf. Information Sci. Systems*, Baltimore, MD (April 1976) 27–32.

[33] R. Book, Simple representations of certain classes of languages, *J. Assoc. Comput. Mach.* (to appear).

[34] S. Cook, Path systems and language recognition, *Proc. 2nd Ann. ACM Symp. Theory Comput.*, Northampton, MA (May 1970) 70–72.

[35] W. Erni, Some further languages log-tape reducible to context-free languages, Report #45, Institut für Angewandte Informatik und Formale Beschreibungsverfahren, Universität Karlsruhe (November 1976); also Ph.D. dissertation, Universität Karlsruhe.

[36] S. Ginsburg, J. Goldstine, and S. Greibach, Uniformly erasable AFL, *J. Comput. System Sci.* **10** (1975) 165–182.

[37] O.H. Ibarra, Controlled pushdown automata, *Information Sci.* **6** (1973) 327–342.

[38] N.A. Khabbaz, Control sets on linear grammars, *Information and Control* **25** (1974) 206–221.

[39] N.A. Khabbaz, A geometrical hierarchy of languages, *J. Comput. System Sci.* **8** (1974) 142–157.

[40] F. Klingenstein, Structures of bounded languages in certain families of languages, Ph D. thesis, University of California at Berkeley, CA (1975).

[41] I.H. Sudborough, On the complexity of the membership problem for some extensions of context-free languages, *Int. J. Comput. Math.* (to appear).

[42] D.I. K'el, Two-way *a*-transducers and AFL, *J. Comput. System Sci.* **10** (1975) 88–109.

[43] J. Engelfriet, E.M. Schmidt and J. van Leewen, Stack machines and classes of nonnested maero-languages, Technical report.