Inference of Reversible Languages

DANA ANGLUIN

Yale University, New Haven, Connecticut

Abstract. A family of efficient algorithms for inferring certain subclasses of the regular languages from finite positive samples is presented. These subclasses are the k-reversible languages, for $k = 0, 1, 2, \ldots$. For each k there is an algorithm for finding the smallest k-reversible language containing any finite positive sample. It is shown how to use this algorithm to do correct identification in the limit of the k-reversible languages from positive data. A reversible language is one that is k-reversible for some $k \geq 0$. An efficient algorithm is presented for inferring reversible languages from positive and negative examples, and it is shown that it leads to correct identification in the limit of the class of reversible languages. Numerous examples are given to illustrate the algorithms and their behavior.

Categories and Subject Descriptors: F.1.1 [**Computation by Abstract Devices**]: Models of Computation—*automata*; F.4.3 [**Mathematical Logic and Formal Languages**]: Formal Languages—*classes defined by grammars or automata*; I.2.6 [**Artificial Intelligence**]: Learning—*induction*; I.5.1 [**Pattern Recognition**]: Models—*structural*

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Inductive inference, grammatical inference, reversible languages

1. *Introduction*

This paper concerns the problem of inductively inferring general rules from examples. People seem to agree that this is an important problem area, but there is much less agreement about how to study it. One of the primary goals of this paper is to indicate that a relatively theoretical approach, using tools from complexity theory and formal language theory, appears to be both feasible and fruitful.

When people communicate complex procedures to other people, they often seem to rely on a somewhat sketchy description of the general ideas, together with examples to elucidate particular details. If we could find sound, uniform, and convenient methods that would allow computer programs to generalize appropriately from examples, these would probably increase the usability of computers by experts and nonexperts both. This paper is part of a general study of what makes a class of rules efficiently and reliably inferable from examples, with the goal of eventually finding such methods.

In this paper we study inferability in a particular abstract domain, that of inferring formal languages from finite samples. Imagine that we are given a finite set of strings from some formal language, and possibly another finite set of strings from the complement of the language, and we are required to make a guess of what the

This research was supported by the National Science Foundation under Grant MCS 80-02447

Author's address: Department of Computer Science, Yale University, P.O. Box 2158, Yale Station, New Haven, CT 06520

unknown language is. For example, given the set of strings {00111, 01, 000011, 001111}, we might guess that the underlying language is the set of all strings consisting of any number of 0's followed by any number of 1's. It may be objected that there are infinitely many regular languages that contain this sample, and we have no way of knowing whether such a guess is "right." One of the fundamental problems for this area of study is to find appropriate, natural, and theoretically sound criteria for the "goodness" of generalizations such as this one.

The concept of identification in the limit, formulated by Gold [18], has been of basic importance in theoretical studies of inductive inference. This concept relies on looking at the limiting behavior of the inferring process as it is given more and more examples of a particular rule. If, for any sequence that eventually contains each example of a given rule, the inferring process produces a sequence of guesses that eventually converges to one that is a correct description of the underlying rule, then the process is said to identify this rule in the limit. Much has been learned about the classes of formal languages and partial recursive functions that can be correctly identified in the limit by various kinds of effective inference processes [6, 7, 12, 18]. However, the inference procedures described in these abstract studies are generally enumerative in character and appear to be too inefficient to be of practical use.

A number of methods have been proposed to perform inductive inference in concrete domains, for example, finite automata, context-free grammars, logical theories, and programs in LISP and other programming languages. An overview of inference techniques for formal grammars and applications in the domain of syntactic pattern recognition may be found in the survey article of Fu and Booth [16], and the books of Fu [14, 15] and Gonzalez and Thomason [20]. Further references concerning both abstract and concrete results in inductive inference may be found in the bibliography of Smith [31]. Few studies of inference in concrete domains have given analyses of the efficiency of the methods used. In the absence of such analysis, behavior on test cases is often used to give some indication of the computational feasibility of an inference method.

However, there has been some theoretical work concerning the possibility of efficient inference procedures for specific concrete domains, which we now sketch. On the negative side, there are results that show that finding a smallest finite automaton or regular expression compatible with a given finite sample consisting of a finite set of strings marked as "accepted" and another finite set of strings marked as "rejected" is an NP-hard problem, even under rather strong restrictions on the samples and possible answers [2, 19]. The idea of searching for the "smallest" description compatible with the sample is natural and attractive at first sight. However, these results suggest that such an approach will not lead to a general understanding of the concept of efficient inferability. On the positive side, polynomial-time inference algorithms have been found for certain classes of parenthesis grammars [9, 10, 27] and for the one-variable pattern languages [3].

The primary contribution of the present paper is a family of new, efficient algorithms to infer certain subclasses of the regular languages from positive samples. These algorithms were discovered in the course of trying to understand variants of an inference heuristic originally proposed by Feldman [13] and compare favorably with a heuristic method of tail-clustering recently proposed by Miclet [29]. The classes inferred by these algorithms are the *k-reversible languages*, for $k = 0, 1, 2, \ldots$. The class of zero-reversible languages was studied, though not named, by McNaughton [26], who proved that the loop-complexity (or star-height) of a zero-reversible language is exactly equal to the cycle rank of its reduced state-graph. This

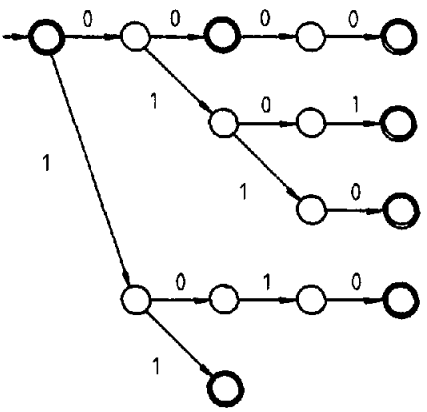

FIG. 1 The prefix tree machine for {λ, 00, 11, 0000, 0101, 0110, 1010}

suggests both the naturalness of the k-reversible languages and the possibility of unexplored, fruitful links between inferability and algebraic structure.

In evaluating results of this kind, important criteria are: how efficient and incremental the algorithm is, how precisely and naturally its guess is defined for any given sample, whether it does correct identification in the limit, and how natural and useful the class of rules inferred is. The algorithms we present are both efficient and incremental. The guess for a given sample is shown to be the smallest k-reversible language containing the sample, and it is shown that this leads to correct identification in the limit of the k-reversible languages. The property of producing the best k-reversible "summary" of the input sample suggests that these algorithms may ultimately be useful as components of more complex inference procedures employing "summaries" of various different kinds.

In Section 2 we give an informal example of the zero-reversible inference algorithm. Formal preliminaries are in Section 3. The definitions and basic results about the reversible languages are in Section 4. The zero-reversible inference algorithm is formally described, justified, and analyzed in Section 5, and the generalization to k-reversible languages is presented in Section 6. The use of negative examples and an algorithm to infer reversible languages from positive and negative data are presented in Section 7. Section 8 contains comparisons with other methods of inferring regular languages from positive data, and Section 9 contains concluding remarks.

2. *An Informal Example*

A zero-reversible acceptor is a deterministic finite-state acceptor with at most one final state such that no two arrows entering any state are labeled with the same input symbol. Suppose we are given the sample

$$S = \{\lambda, 00, 11, 0000, 0101, 0110, 1010\}$$

and are told that it is part of the language of some zero-reversible acceptor. We might then proceed as follows.

First we construct the prefix tree acceptor for S, shown in Figure 1. This is not zero-reversible, because it has more than one final state. So we merge all the final states together to produce the acceptor shown in Figure 2a. However, this acceptor is not deterministic, so we merge the two states labeled B to produce the acceptor shown in Figure 2b. This acceptor is not zero-reversible, because several states (labeled B) are 0-predecessors of the state labeled A, and several states (labeled D) are 1-predecessors of state A. We therefore merge the states labeled B and the states

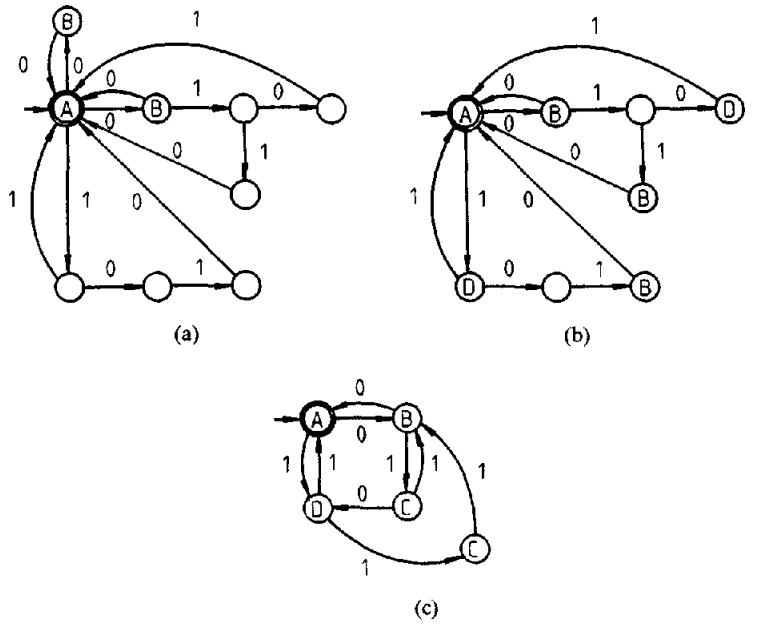

FIG. 2 Stages in the zero-reversible algorithm

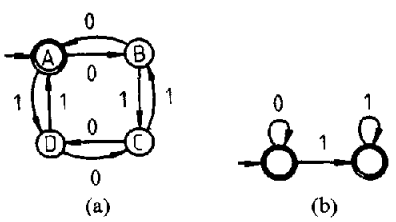

FIG 3. A zero-reversible and a 1-reversible acceptor

labeled D to produce the acceptor shown in Figure 2c. This is still not zero-reversible, since the two states labeled C are both 1-predecessors of state B. Merging these two states gives the zero-reversible acceptor shown in Figure 3a. This machine accepts the set of all strings over {0, 1} that contain an even number of 0's and an even number of 1's, which is a plausible inference from the original sample.

We leave it to the reader to verify that the method we have just sketched infers the universal language $(0 + 1)^*$ from the sample

$$S = \{01, 00011, 00111, 001111\},$$

which does not seem to be a very plausible inference. In subsequent sections of this paper we precisely describe and analyze the behavior of the method sketched and present a generalization that handles the second kind of example more satisfactorily.

3. *Basic Definitions and Notation*

In this section we define certain notions we shall need from the theory of formal languages and automata. We generally follow Hartmanis and Stearns [23] in spirit, though we require some technical modifications of their definitions.

The *alphabet* is a fixed finite nonempty set U of symbols. U^* denotes the set of all finite strings over U. λ denotes the empty string. The length of the string w is denoted $|w|$. The reverse of the string w is denoted w^r. The concatenation of the strings u and v is denoted uv. The string u is a *prefix* of the string v if and only if there exists a string w such that $uw = v$. The string u is a *suffix* of the string v if and only if there exists a string w such that $wu = v$.

A *language* is any subset of U^*. The reverse of a language L is defined by $L^r = \{w^r : w \in L\}$. If L is any language, then we define the set $\Pr(L)$ of prefixes of elements of L by

$$\Pr(L) = \{u : \text{for some } v,\ uv \in L\}.$$

Also, for any $w \in U^*$, we denote the left-quotient of L and w by

$$T_L(w) = \{v : wv \in L\}.$$

Thus, $T_L(w) \neq \emptyset$ if and only if $w \in \Pr(L)$. When L is clear from the context, we write $T(w)$ instead of $T_L(w)$.

A *positive sample* is a finite set of strings. S is a *positive sample of the language* L if and only if S is a finite subset of L. (We consider samples containing negative as well as positive information in Section 7.)

If S is any set, $|S|$ denotes the cardinality of S. A *partition* of S is a set of pairwise disjoint nonempty subsets of S whose union is S. If π is a partition of S, then for any element $s \in S$ there is a unique element of π containing s, which we denote $B(s, \pi)$ and call the *block* of π containing s. A partition π is said to *refine* another partition π' if and only if every block of π' is a union of blocks of π. We denote this by $\pi \leq \pi'$. In this case we also say that π' is *coarser* than π, or that π is *finer* than π'. Note that both of these relations are reflexive. If π is a partition of a set S and S' is a subset of S, then the *restriction* of π to S' is the partition π' consisting of all those sets B' that are nonempty and are the intersection of S' and some block of π. The *trivial partition* of a set S is the class of all sets $\{s\}$ such that $s \in S$.

A *right congruence* is a partition π of U^* with the property that $B(w_1, \pi) = B(w_2, \pi)$ implies $B(w_1u, \pi) = B(w_2u, \pi)$ for all $w_1, w_2, u \in U^*$. If L is any language, then $T_L(w_1) = T_L(w_2)$ implies $T_L(w_1u) = T_L(w_2u)$ for all u, so L determines an associated right congruence π_L by $B(w_1, \pi_L) = B(w_2, \pi_L)$ if and only if $T_L(w_1) = T_L(w_2)$. Other natural right congruences are associated with automata. Note that a language L is regular just in case π_L contains finitely many blocks. We assume familiarity with the basic facts about regular sets [21–24].

An *acceptor* is a quadruple $A = (Q, I, F, \delta)$ such that Q is a finite set, I and F are subsets of Q, and δ is a map from $Q \times U$ to subsets of Q. Q is the set of *states*, I is the set of *initial states*, F is the set of *final* or *accepting* states of A, and δ is the *transition function* of A. The acceptor is *deterministic* if and only if there is at most one initial state, and for each state $q \in Q$ and symbol $a \in U$ there is at most one element in $\delta(q, a)$. Note that we allow undefined transitions in deterministic automata. The *empty acceptor* is the unique acceptor with $Q = \emptyset$. (The empty acceptor is deterministic.) We shall sometimes write $\delta(q, b) = q'$ for $\delta(q, b) = \{q'\}$.

Let $A = (Q, I, F, \delta)$ and $A' = (Q', I', F', \delta')$ be acceptors. A is *isomorphic* to A' if and only if there exists a bijection h of Q onto Q' such that $h(I) = I'$, $h(F) = F'$, and for every $q \in Q$ and $b \in U$, $h(\delta(q, b)) = \delta'(h(q), b)$. Isomorphic acceptors are the same up to renaming of the states. A' is a *subacceptor* of A if and only if Q', I', and F' are subsets of Q, I, and F, respectively, and for every $q' \in Q'$ and $b \in U$, $\delta'(q', b)$

is a subset of $\delta(q', b)$. Pictorially, a subacceptor is obtained by removing some states, initial states, final states, and transition arrows from the diagram of an acceptor.

If Q_0 is a subset of Q, then the *subacceptor of A induced by Q_0* is the acceptor $(Q_0, I_0, F_0, \delta_0)$, where I_0 is the intersection of Q_0 and I, F_0 is the intersection of Q_0 and F, and $q' \in \delta_0(q, b)$ if and only if $q, q' \in Q_0$ and $q' \in \delta(q, b)$. A state of A is called *useful* if and only if there exist strings u and v such that $q \in \delta(I, u)$ and $\delta(q, v)$ contains some element of F. States that are not useful are called *useless.* An acceptor that contains no useless states is called *stripped.* If A is any acceptor, then the *stripped subacceptor* of A is the subacceptor of A induced by the useful states of A.

We extend the transition function δ to map a set of states and a string to a set of states in the usual way. If $q \in \delta(q', a)$, then q' is called an *a-predecessor* of q, and q is called an *a-successor* of q'.

A string u is *accepted by* an acceptor $A = (Q, I, F, \delta)$ if and only if $\delta(I, u)$ contains some element of F. The set of strings accepted by A is called the *language* of A and is denoted $L(A)$. If A and A' are isomorphic, then $L(A) = L(A')$. If A' is a subacceptor of A, then $L(A')$ is a subset of $L(A)$. If A' is the stripped subacceptor of A, then $L(A') = L(A)$.

Let A be a deterministic acceptor with initial state set I. Define the partition π_A by $B(w_1, \pi_A) = B(w_2, \pi_A)$ if and only if $\delta(I, w_1) = \delta(I, w_2)$. Since $T_L(w_1) = T_L(w_2)$ if $\delta(I, w_1) = \delta(I, w_2)$, π_A is a right congruence that refines π_L, where $L = L(A)$.

Let $A = (Q, I, F, \delta)$ be any acceptor. If π is any partition of Q, we define another acceptor $A/\pi = (Q', I', F', \delta')$ as follows. Q' is the set of blocks of π. I' is the set of all blocks of π that contain an element of I. F' is the set of all blocks of π that contain an element of F. The block B_2 is in $\delta'(B_1, a)$ whenever there exist $q_1 \in B_1$ and $q_2 \in B_2$ such that $q_2 \in \delta(q_1, a)$. A/π is called the *quotient* of A and π.

Let L be any regular language. We define the *canonical acceptor* for L, $A(L) = (Q, I, F, \delta)$, as follows:

$$\begin{aligned} Q &= \{T_L(u) : u \in \Pr(L)\}, \\ I &= \{T_L(\lambda)\} && \text{if } L \neq \emptyset, \text{ otherwise } I = \emptyset, \\ F &= \{T_L(w) : w \in L\}, \\ \delta(T_L(u), a) &= T_L(ua) && \text{if } u, ua \in \Pr(L). \end{aligned}$$

(Note that if $L = \emptyset$, then $\Pr(L) = \emptyset$ and $A(L)$ is the empty acceptor.) Recall that $T_L(u_1) = T_L(u_2)$ implies $T_L(u_1 v) = T_L(u_2 v)$ for all strings v, so that this transition function is well defined and $A(L)$ is deterministic. The acceptor $A(L)$ accepts the language L and has the minimum possible number of states among all acceptors of L. $A(L)$ is stripped, that is, contains no useless states. Note that the right congruence $\pi_{A(L)}$ induced by $A(L)$ coincides with the right congruence π_L induced by L. An acceptor A is called *canonical* if and only if A is isomorphic to the canonical acceptor for the language of A. Given a deterministic acceptor A, there is an efficient procedure to find a canonical acceptor for $L(A)$, as described in [1].

Let S be a positive sample, that is, a finite set of strings. Define the *prefix tree acceptor* for S, $\mathrm{PT}(S) = (Q, I, F, \delta)$, as follows:

$$\begin{aligned} Q &= \Pr(S), \\ I &= \{\lambda\} && \text{if } S \neq \emptyset, \text{ otherwise } I = \emptyset, \\ F &= S, \\ \delta(u, a) &= ua && \text{whenever } u, ua \in Q. \end{aligned}$$

Then $\mathrm{PT}(S)$ is a deterministic acceptor that accepts precisely the set S. (Note that if $S = \emptyset$, then $\mathrm{PT}(S)$ is the empty acceptor.) The inference algorithms that we shall

consider begin with the prefix tree acceptor for the input sample and generalize it by merging states. The following lemma concerns what kinds of acceptors may be obtained by merging states of the prefix tree acceptor for a sample.

LEMMA 1. *Let S be a positive sample of the regular language L, and let A_0 be the prefix tree acceptor for S. Let π be the partition π_L restricted to the set $Pr(S)$ of prefixes of elements of S. Then A_0/π is isomorphic to a subacceptor of $A(L)$.*

PROOF. The result holds trivially if $S = \emptyset$, so assume that $S \neq \emptyset$. We shall denote $T_L(w)$ by $T(w)$. The partition π is defined by $B(w_1, \pi) = B(w_2, \pi)$ if and only if $T(w_1) = T(w_2)$, for all $w_1, w_2 \in \Pr(S)$. Hence $h(B(w, \pi)) = T(w)$ is a well-defined and injective map from the states of A_0/π to the states of $A(L)$. The initial state $B(\lambda, \pi)$ of A_0/π is mapped to the initial state $T(\lambda)$ of $A(L)$. If B_1 is a final state of A_0/π, then $B_1 = B(w, \pi)$ for some w in S, and since L contains S, $T(w)$ is a final state of $A(L)$. Hence h maps final states of A_0/π to final states of $A(L)$.

If B_2 is a b-successor of B_1 in A_0/π, then for some $w \in \Pr(S)$ we have $B_2 = B(wb, \pi)$, $B_1 = B(w, \pi)$, and $wb \in \Pr(S)$. Thus $h(B_2) = T(wb)$ is a b-successor of $h(B_1) = T(w)$ in $A(L)$.

Thus h is an isomorphism between A_0/π and a subacceptor of $A(L)$. □

COROLLARY 2. *$L(A_0/\pi)$ is contained in L.*

It may also be shown that if S is a *complete* sample for L, that is, exercises every transition in $A(L)$, then the acceptor A_0/π defined above is isomorphic to $A(L)$. Of course, in the inference procedures we consider, π is not given but rather must be guessed. These results show only that a correct guess exists.

4. *Reversible Languages*

In this section we define reversible regular languages and establish some of their basic properties. Inference algorithms for these languages are described in later sections.

4.1. ZERO-REVERSIBLE ACCEPTORS AND LANGUAGES. Let $A = (Q, I, F, \delta)$ be an acceptor, and let $L = L(A)$. The *reverse* of δ, denoted δ^r, is defined by

$$\delta^r(q, a) = \{q' : q \in \delta(q', a)\} \qquad \text{for all} \quad a \in U, \quad q \in Q.$$

The *reverse* of the acceptor A is $A^r = (Q, F, I, \delta^r)$. Pictorially, we obtain A^r from A by interchanging the initial and final states and reversing each of the transition arrows. It is not difficult to verify by induction that $L(A^r) = (L(A))^r$.

The acceptor A is said to be *zero-reversible* if and only if both A and A^r are deterministic. Using the terminology of Cohen and Brzozowski [8], an acceptor is *reset-free* if and only if for no two distinct states q_1 and q_2 do there exist $b \in U$ and $q_3 \in Q$ such that $\delta(q_1, b) = q_3 = \delta(q_2, b)$. Then an acceptor is zero-reversible if and only if it is deterministic, has at most one final state, and is reset-free. Alternatively, a zero-reversible acceptor is any subacceptor of a permutation acceptor with at most one initial and one final state.

Remark 3. Note that if A is zero-reversible and accepts uv, then $\delta(q_0, u) = \delta^r(q_f, v^r)$ if $I = \{q_0\}$ and $F = \{q_f\}$. Consequently, if A accepts both u_1v and u_2v, then $\delta(q_0, u_1) = \delta(q_0, u_2)$.

Remark 4. If A is a zero-reversible acceptor and A' is any subacceptor of A, then A' is a zero-reversible acceptor.

LEMMA 5. *Suppose A is a zero-reversible acceptor. Then the stripped subacceptor A' of A is canonical.*

PROOF. A' is a zero-reversible acceptor and accepts $L = L(A)$. If L is the empty language, then A' is the empty acceptor and therefore canonical. So suppose that L is not the empty language. Let $A' = (Q, \{q_0\}, \{q_f\}, \delta)$. Let q_1 and q_2 be states of A', and suppose that $\{v: \delta(q_1, v) = q_f\} = \{v: \delta(q_2, v) = q_f\}$. Since A' is stripped, this implies that there exist strings u_1, u_2, v such that $q_1 = \delta(q_0, u_1)$, $q_2 = \delta(q_0, u_2)$, and u_1v, $u_2v \in L$. Thus, by Remark 3, $q_1 = q_2$. Hence A' is canonical. □

A language L is defined to be *zero-reversible* if and only if there exists a zero-reversible acceptor A such that $L = L(A)$. The following lemma shows that we need only test the canonical acceptor to determine whether a language is zero-reversible.

LEMMA 6. *A regular language L is zero-reversible if and only if the canonical acceptor $A(L)$ is zero-reversible.*

PROOF. The "if" direction is immediate from the definitions. Suppose that L is zero-reversible and A is a zero-reversible acceptor for L. The stripped subacceptor A' of A is canonical, zero-reversible, and accepts L. Since $A(L)$ is isomorphic to A', $A(L)$ is zero-reversible. □

The next result gives a purely language-theoretic characterization of the zero-reversible sets.

THEOREM 7. *Let L be a regular language. Then L is zero-reversible if and only if whenever u_1v and u_2v are in L, $T_L(u_1) = T_L(u_2)$.*

PROOF. We denote $T_L(w)$ by $T(w)$. Suppose L is zero-reversible. By the above lemma, the canonical acceptor $A(L)$ is zero-reversible. Thus, if u_1v and u_2v are in L, then by Remark 3, u_1 and u_2 lead to the same state of $A(L)$, that is, $T(u_1) = T(u_2)$.

Conversely, suppose that whenever u_1v and u_2v are in L, $T(u_1) = T(u_2)$. Thus, if u_1 and u_2 are in L, $T(u_1) = T(u_2)$, so $A(L)$ has at most one accepting state. If $T(u_1)$ and $T(u_2)$ are states of $A(L)$ such that $T(u_1b) = T(u_2b)$ is a state of $A(L)$ for some $b \in U$, then there exists a string v such that u_1bv and u_2bv are in L, so by the hypothesis, $T(u_1) = T(u_2)$. Hence, $A(L)$ is zero-reversible, and therefore L is zero-reversible. □

Example 8. Let $U = \{0, 1\}$. The language of all strings over U containing an even number of 1's and an even number of 0's is accepted by the acceptor shown in Figure 3a and is therefore zero-reversible. The canonical acceptor for the language 0^*1^* is shown in Figure 3b. Since the canonical acceptor is not zero-reversible, the language 0^*1^* is not zero-reversible. □

In order to do correct identification in the limit from positive data, we must avoid the problem of "overgeneralizing," that is, of accidentally guessing a language that is a strict superset of the unknown language. This problem is analyzed in a general setting in [4]. For this particular setting we define a *characteristic sample* of a zero-reversible language L to be a sample S_0 of L with the property that L is the smallest zero-reversible language that contains S_0. If we detect a characteristic sample for L among the input strings, then we are assured that a guess of L will not be an overgeneralization. The following result is used in the proof of correct identification in the limit of the zero-reversible languages.

THEOREM 9. *For any zero-reversible language L there exists a characteristic sample.*

PROOF. Clearly, if $L = \emptyset$, then $S_0 = \emptyset$ is a characteristic sample for L. Suppose $L \neq \emptyset$, and let $A = (Q, \{q_0\}, \{q_f\}, \delta)$ be the canonical acceptor for L. For each state $q \in Q$, let $u(q)$ and $v(q)$ be strings of the minimum possible lengths such that $\delta(q_0, u(q)) = q$ and $\delta(q, v(q)) = q_f$. Let S_0 consist of all strings of the form $u(q)v(q)$ such that $q \in Q$ and all strings of the form $u(q)bv(q')$ such that $q \in Q$, $b \in U$, and $q' = \delta(q, b)$. We show that S_0 is a characteristic sample for L.

Let L' be any zero-reversible language containing S_0. We show that $T_{L'}(w) = T_{L'}(u(q))$ for all strings $w \in \Pr(L)$, where $q = \delta(q_0, w)$. Since $u(q_0) = \lambda$, this holds for λ. Inductively suppose that this holds for all elements of $\Pr(L)$ of length at most n, for some $n \geq 0$. Let w be a string from $\Pr(L)$ of length n, and suppose that $b \in U$ is such that $wb \in \Pr(L)$. By the inductive hypothesis, $T_{L'}(w) = T_{L'}(u(q))$, where $q = \delta(q_0, w)$. Thus, $T_{L'}(wb) = T_{L'}(u(q)b)$. If $q' = \delta(q, b) = \delta(q_0, wb)$, then $u(q')v(q')$ and $u(q)bv(q')$ are both elements of S_0. Thus $T_{L'}(u(q')) = T_{L'}(u(q)b)$ because L' is zero-reversible, by Remark 3. Hence $T_{L'}(wb) = T_{L'}(u(q'))$, which completes the induction.

Thus for every $w \in L$, $T_{L'}(w) = T_{L'}(u(q_f))$, and since $u(q_f) \in S_0$, this implies that $w \in L'$. Therefore, L is contained in L', and L is the smallest zero-reversible language that contains S_0. Hence S_0 is a characteristic sample for L. □

Example 10. Consider the language over $\{0, 1\}$ consisting of all strings containing an even number of 0's and an even number of 1's, whose canonical acceptor is pictured in Figure 3a. Applying the construction process described in the above proof to obtain a characteristic sample for this language, we may define $u(A) = \lambda$, $v(A) = \lambda$, $u(B) = 0$, $v(B) = 0$, $u(C) = 01$, $v(C) = 10$, $u(D) = 1$, and $v(D) = 1$, which gives the sample

$$S_0 = \{\lambda, 00, 11, 0101, 0110, 1010\}. \qquad \square$$

4.2. *k*-REVERSIBLE ACCEPTORS AND LANGUAGES. The notion of k-reversibility is a generalization of zero-reversibility. Let k be a fixed nonnegative integer. Let $A = (Q, I, F, \delta)$ be an acceptor. The string u is said to be a *k-follower* (resp. *k-leader*) of the state q in A if and only if $|u| = k$ and $\delta(q, u) \neq \emptyset$ (resp. $\delta^r(q, u^r) \neq \emptyset$). Note that every state has exactly one 0-follower and one 0-leader, namely, λ. The acceptor A is defined to be *deterministic with lookahead k* if and only if for any pair of distinct states q_1 and q_2, if $q_1, q_2 \in I$ or $q_1, q_2 \in \delta(q_3, a)$ for some $q_3 \in Q$ and $a \in U$, then there is no string that is a k-follower of both q_1 and q_2. This guarantees that any nondeterministic choice in the operation of A can be resolved by looking ahead k symbols past the current one.

An acceptor A is defined to be *k-reversible* if and only if A is deterministic and A^r is deterministic with lookahead k. A language L is defined to be *k-reversible* if and only if there exists a *k-reversible* acceptor A such that $L = L(A)$. Note that these definitions coincide with the definitions for zero-reversible acceptors and languages when $k = 0$.

Remark 11. If $A = (Q, \{q_0\}, F, \delta)$ is k-reversible and u_1vw and u_2vw are accepted by A, where $|v| = k$, then there is a unique state q such that $\delta(q_0, u_1v) = q = \delta(q_0, u_2v)$.

Remark 12. Any subacceptor of a k-reversible acceptor is k-reversible.

The analog of Lemma 5 for k-reversible languages is not in general true, but it still suffices to test the canonical acceptor for a language to decide whether it is k-reversible, as we now show.

LEMMA 13. *A regular set L is k-reversible if and only if $A(L)$ is k-reversible.*

PROOF. We denote $T_L(w)$ by $T(w)$. The "if" direction is immediate. Suppose L is k-reversible and is accepted by the k-reversible acceptor $A = (Q, \{q_0\}, F, \delta)$. $A(L)$ is a deterministic acceptor of L, and π_A refines $\pi_{A(L)}$. We must check that $(A(L))^r$ is deterministic with lookahead k. Suppose $T(w_1)$ and $T(w_2)$ are states of $(A(L))^r$ and v is a k-follower of $T(w_1)$ and of $T(w_2)$ in $(A(L))^r$. Then there exist strings u_1 and u_2 such that $T(u_1v^r) = T(w_1)$ and $T(u_2v^r) = T(w_2)$. If $T(w_1)$ and $T(w_2)$ are both initial in $(A(L))^r$, or if for some $a \in U$ there is a state $T(w_3)$ of $A(L)$ such that $T(w_1a) = T(w_3) = T(w_2a)$, then there exists a string w such that $A(L)$ accepts both u_1v^rw and u_2v^rw. Thus A accepts both of these strings, and by the above remark, $\delta(q_0, u_1v^r) = q = \delta(q_0, u_2v^r)$ for some q in A. Thus u_1v^r and u_2v^r are in the same block of π_A, and therefore in the same block of $\pi_{A(L)}$, so $T(u_1v^r) = T(u_2v^r)$ and $T(w_1) = T(w_2)$. Thus $A(L)$ is k-reversible. □

We now give a characterization of the k-reversible languages purely in terms of the languages.

THEOREM 14. *Let L be a regular language. Then L is k-reversible if and only if whenever u_1vw and u_2vw are in L and $|v| = k$, $T_L(u_1v) = T_L(u_2v)$.*

PROOF. We denote $T_L(w)$ by $T(w)$. Suppose L is k-reversible. By the above lemma, the canonical acceptor $A(L)$ is k-reversible. Suppose u_1vw and u_2vw are in L, where $|v| = k$. Then by Remark 11, u_1v and u_2v lead to the same state of $A(L)$, so $T(u_1v) = T(u_2v)$.

Conversely, suppose that L is such that whenever u_1vw and u_2vw are in L, where $|v| = k$, then $T(u_1v) = T(u_2v)$. Suppose that u_1, u_2, and v are such that $T(u_1v)$ and $T(u_2v)$ are accepting states of $A(L)$, where $|v| = k$. Then u_1v and u_2v are in L, so by the hypothesis, $T(u_1v) = T(u_2v)$. Similarly, if u_1, u_2, and v are such that $T(u_1vb) = T(u_2vb)$ is some state of $A(L)$, where $|v| = k$, then there exists some string w such that u_1vbw and u_2vbw are in L. Then $T(u_1v) = T(u_2v)$ by the hypothesis on L. Hence $A(L)$ is k-reversible, and therefore L is k-reversible. □

Example 15. The acceptor shown in Figure 3b is 1-reversible, so the language 0^*1^* is 1-reversible. Consider the regular set L denoted by the regular expression $ba^*c + d(aa)^*c$. The canonical acceptor for L is shown in Figure 4. This acceptor is not k-reversible for any $k \geq 0$, since for any $k \geq 0$, a^k is a k-leader of two distinct states that have a common c-successor. Hence L is not k-reversible for any $k \geq 0$. □

We now consider characteristic samples for k-reversible languages. A positive sample S is a *characteristic sample* for a k-reversible language L if and only if L is the smallest k-reversible language containing S. (Note that whether a sample is a characteristic sample for a given language depends on the value of k under consideration. Thus a sample that is characteristic for a zero-reversible language L may not be characteristic for the same language considered as a 1-reversible language.)

THEOREM 16. *Let L be any k-reversible language. Then there exists a characteristic samples S_0 for L.*

PROOF. Since Theorem 9 establishes this result in the case $k = 0$, we assume $k \geq 1$. If $L = \emptyset$, then $S_0 = \emptyset$ is a characteristic sample for L, so we suppose that $L \neq \emptyset$. Let $A = (Q, \{q_0\}, F, \delta)$ be the canonical acceptor for L. For each $q \in Q$ let L_q denote the set of k-leaders of q in A. For each pair $q \in Q$ and $x \in L_q$ let $u(q, x)$

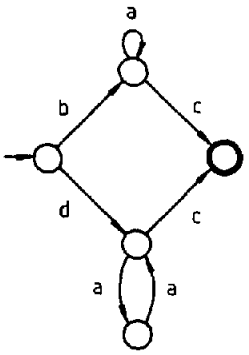

FIG. 4. A nonreversible acceptor

denote a string u of the minimum possible length such that $\delta(q_0, ux) = q$. For each $q \in Q$ let $v(q)$ denote a string v of the minimum possible length such that $\delta(q, v) \in F$. The sample S_0 is defined to consist of all strings $w \in L$ of length less than k, all strings $u(q, x)xv(q)$ such that $q \in Q$ and $x \in L_q$, and all strings $u(q, x)xbv(q')$ such that $q \in Q$, $x \in L_q$, and $q' = \delta(q, b)$. No other strings are in S_0.

Let L' be any k-reversible language containing S_0. We must show that L is contained in L'. Clearly any element w of L of length less than k is in S_0 and therefore in L'. We show by induction that for every $w \in \Pr(L)$ of length at least k, $T_{L'}(w) = T_{L'}(u(q, x)x)$, where x is the suffix of w of length k and $q = \delta(q_0, w)$. If w has length exactly k, then $w = x$ and $u(q, x) = \lambda$, so this condition is satisfied. Suppose that for some $n \geq k$ this condition is satisfied for all strings $w \in \Pr(L)$ of length at most n. Suppose w is any element of $\Pr(L)$ of length $n + 1$. Write $w = w'axb$, where $|x| = k - 1$ and $a, b \in U$. By the induction hypothesis, $T_{L'}(w'ax) = T_{L'}(u(q, ax)ax)$, where $q = \delta(q_0, w'ax)$. Thus $T_{L'}(w) = T_{L'}(u(q, ax)axb)$. Let $q' = \delta(q, b) = \delta(q_0, w)$. Then S_0 contains the strings $u(q, ax)axbv(q')$ and $u(q', xb)xbv(q')$, so L' contains both of these strings. By Remark 11, this implies that $T_{L'}(u(q, ax)axb) = T_{L'}(u(q', xb), xb)$, so $T_{L'}(w) = T_{L'}(u(q', xb)xb)$, completing the induction step.

Now let w be any element of L of length at least k, and let x be the suffix of w of length k. Then $T_{L'}(w) = T_{L'}(u(q_f, x)x)$, where $q_f \in F$. Since q_f is an accepting state, $v(q_f) = \lambda$, so $u(q_f, x)x$ is in S_0 and therefore in L'. Hence w is in L', which completes the proof that L is contained in L'. Thus L is the smallest k-reversible language containing S_0, and S_0 is a characteristic sample for L. □

Example 17. Consider the language 0^*1^* whose canonical acceptor is shown in Figure 3b. Applying the method of the above proof to construct a characteristic sample S_0 for this 1-reversible language, we obtain $L_A = \{0\}$, $L_B = \{1\}$, $u(A, 0) = \lambda$, $v(A) = \lambda$, $u(B, 1) = \lambda$, $v(B) = \lambda$, and $S_0 = \{\lambda, 0, 1, 00, 01, 11\}$. □

4.3 GENERAL PROPERTIES OF REVERSIBLE LANGUAGES. Let R_k denote the class of k-reversible languages over the alphabet U, and let R_* denote the union of all the R_k for $k \geq 0$. The languages in R_* are called simply the *reversible languages.*

THEOREM 18. *If k is any nonnegative integer,*

(1) *R_k is properly contained in R_{k+1},*
(2) *R_k is closed under pairwise intersection,*
(3) *R_* is not closed under pairwise union or complementation,*
(4) *R_* is not closed under concatenation,*
(5) *R_* is not closed under Kleene closure,*
(6) *R_k is closed under reversal.*

PROOF. It is immediate from the definitions that R_k is contained in R_{k+1}. The language denoted by $1^{k+1}1^*$ is $(k+1)$-reversible but not k-reversible, showing that the containment is proper.

It is straightforward to verify that the usual direct product construction applied to two k-reversible acceptors produces a k-reversible acceptor that recognizes the intersection of the languages of the two acceptors. A description of the product construction may be found in [24].

Since the languages denoted by the expressions ba^*c and $d(aa)^*c$ are easily checked to be zero-reversible, Example 15 shows that the union of two zero-reversible languages may not be reversible. Thus R_* is not closed under pairwise union. De Morgan's laws then imply that R_* is not closed under complementation with respect to U^*.

The languages $L_1 = (a+b)^*$ and $L_2 = (a+c)^*$ are easily seen to be zero-reversible, but their concatenation $L_3 = L_1L_2$ is not reversible. To see this, note that for every k, ba^k, ca^k, and ba^kb are in L_3, while ca^kb is not in L_3, so by Theorem 14, L_3 is not k-reversible for any k. Thus R_* is not closed under concatenation.

Let L_4 denote the set of all strings over the alphabet $\{0, 1, 2\}$ whose digits add up to 1 modulo 3. L_4 is easily seen to be zero-reversible, but $(L_4)^*$ is not reversible. To see this, we note that for every k, 10^k1, 0^k1, and 10^k21 are in $(L_4)^*$, while 0^k21 is not. Thus, by Theorem 14, $(L_4)^*$ is not k-reversible for any k, and R_* is not closed under Kleene closure.

Closure of R_k under reversal is immediate when $k = 0$ but requires proof in the case that $k \geq 1$. Fix $k \geq 1$. Let L be a k-reversible language, and let $A = (Q, \{q_0\}, F, \delta)$ be the canonical acceptor for L. Then A is k-reversible, so A^r is deterministic with lookahead k. We construct another acceptor, $A_1 = (Q_1, I_1, F_1, \delta_1)$, to accept L^r. This acceptor is deterministic and operates by "looking ahead" k symbols in the input to make A^r deterministic. We show that A_1 is k-reversible, which implies that L^r is k-reversible.

Let z be distinct from all the elements of Q. States of A_1 are of the form (z, u), where u is a string of length less than k, or of the form (q, u), where $q \in Q$ and u is a string of length k. The element z is a "place holder" before k symbols of the input have been read. The state (q, u) signifies being in state q of A^r looking ahead at the string u in the input.

Let $f(q)$ denote the set of k-followers of state q in A^r, that is,

$$f(q) = \{u : |u| = k \text{ and } \delta^r(q, u) \neq \emptyset\}.$$

We define A_1 as follows:

$$\begin{aligned}
Q_1 &= \{(z, u) : |u| < k\} \cup \{(q, u) : u \in f(q)\},\\
I_1 &= \{(z, \lambda)\},\\
F_1 &= \{(z, u) : u \in L^r\} \cup \{(q, u) : \delta(q_0, u^r) = q\},\\
\delta_1((z, u), a) &= \{(z, ua)\} \qquad \text{if } |u| < k-1,\\
\delta_1((z, u), a) &= \{(q, ua) : q \in F, ua \in f(q)\} \qquad \text{if } |u| = k-1,\\
\delta_1((q, a_1u), a_2) &= \{(q', ua_2) : q' \in \delta^r(q, a_1), ua_2 \in f(q')\}.
\end{aligned}$$

Clearly A_1 has one initial state. We now show that there is at most one a-successor for each state of A_1, for each symbol $a \in U$. If $|u| < k-1$, then the state (z, u) has a unique a-successor for each $a \in U$. If $|u| = k-1$, then for each $a \in U$, because A^r is deterministic with lookahead k, there is at most one state q in F with $ua \in f(q)$, so

there is at most one a-successor of (z, u) in A_1. If $|u| = k - 1$ and $a_1 \in U$ is such that $(q, a_1u) \in Q_1$, then for any $a_2 \in U$ there is at most one state $q' \in \delta^r(q, a_1)$ such that $ua_2 \in f(q')$, so (q, a_1u) has at most one a-successor in A_1. Thus A_1 is deterministic.

To see that A_1 accepts L^r, we argue as follows. If $|u| < k$, then $\delta_1((z, \lambda), u) = (z, u)$, and (z, u) is an accepting state of A_1 if and only if $u \in L^r$. If $|u| \geq k$, then let $u = u_1v$, where $|v| = k$. Then $\delta_1((z, \lambda), u_1v)$ is either empty or is a state (q, v) such that $v \in f(q)$ and $q \in \delta^r(q_f, u_1)$, for some $q_f \in F$. Thus, if A_1 accepts u, $\delta(q_0, v^r) = q$ and $q \in \delta^r(q_f, u_1)$, so $q_0 \in \delta^r(q_f, u_1v)$ and $u = u_1v$ is in L^r. Conversely, if $u = u_1v$ is in L^r, then $v^ru_1^r$ is in L, so if $q = \delta(q_0, v^r)$, then $(q, v) \in F_1$ and $\delta_1((z, \lambda), u_1v) = (q, v)$, so A_1 accepts u. Thus $L^r = L(A_1)$.

It remains to show that A_1 is k-reversible. We have already seen that it is deterministic. Suppose two states of A_1 have a common k-follower v in the reverse of A_1. Then they must be of the form (q_1, v^r) and (q_2, v^r), since they have a common k-leader v^r in A_1. If these states are both in F_1, then $q_1 = \delta(q_0, v) = q_2$, so $(q_1, v^r) = (q_2, v^r)$. If for some $a \in U$, $\delta_1((q_1, v^r), a) = \delta_1((q_2, v^r), a) = (q_3, v')$, then $q_3 \in \delta^r(q_1, b)$ and $q_3 \in \delta^r(q_2, b)$, where b is the initial symbol of v^r. Thus $q_1 = \delta(q_3, b) = q_2$, by the determinism of A, and $(q_1, v^r) = (q_2, v^r)$. Thus A_1 is k-reversible, which shows that L^r is k-reversible. Hence R_k is closed under reversal. □

R_* consists of a hierarchy of classes, contains all the nonempty finite languages over U, and does not contain all the regular languages over U. We now compare R_* with another class of regular sets with generally similar properties, the definite languages [30]. A regular language L is *k-definite* if and only if whenever u_1 and u_2 have a common suffix of length k, $T_L(u_1) = T_L(u_2)$. Note that this implies that there are finitely many distinct k-definite regular languages over a fixed alphabet U. (This is true because there must be fewer than $|U|^{k+1}$ states in the canonical acceptor for a k-definite language.) Let D_k denote the class of k-definite languages and D_* the union of all D_k for $k \geq 0$.

THEOREM 19. *R_* properly contains D_*. For every $k \geq 0$, R_k properly contains D_k.*

PROOF. Let L be a k-definite language; we show that L is k-reversible. We denote $T_L(w)$ by $T(w)$. If $L = \emptyset$, then L is k-reversible; so suppose $L \neq \emptyset$, and let $A(L) = (Q, \{q_0\}, F, \delta)$ be the canonical acceptor for L. Suppose q_1 and q_2 are states of $(A(L))^r$ with a common k-follower v. Then there exist strings u_1 and u_2 such that

$$\delta(q_0, u_1v^r) = q_1 \qquad \text{and} \qquad \delta(q_0, u_2v^r) = q_2.$$

Since L is k-definite, $T(u_1v^r) = T(u_2v^r)$, that is, $q_1 = q_2$. Hence no two distinct states of $(A(L))^r$ have a common k-follower, so $A(L)$ is k-reversible, and therefore L is k-reversible. Thus D_k is contained in R_k.

In Example 8 it was shown that the language over $\{0, 1\}$ consisting of those strings that contain an even number of 1's and an even number of 0's is zero-reversible and therefore contained in R_k for all $k \geq 0$. Since this language is not k-definite for any $k \geq 0$, we conclude that R_k properly contains D_k for all $k \geq 0$ and that R_* properly contains D_*.

(We also note for each $k \geq 0$, R_k contains infinitely many languages, while D_k contains only finitely many languages.) □

The *reverse definite* languages [17] are simply the reversals of the definite languages. Thus, as a corollary of this theorem and the closure of each R_k under reversal, we

have that each R_k properly contains the k-reverse definite languages, and R_* properly contains the reverse definite languages.

Ginzburg has also defined a generalization of the definite and reverse definite languages called the generalized definite languages [17]. A language is *generalized definite* if and only if it may be expressed as the union of a finite set of strings and a finite union of languages of the form FU^*G, where F and G are finite sets of strings, and U is the alphabet. An algebraic characterization of these languages has been given by Zalcstein [33]. In order to compare R_* with the generalized definite languages, we need some additional facts about R_*.

LEMMA 20. *Let F be a finite set of strings and L a reversible language. Then $F \cup L$ is a reversible language.*

PROOF. If F is the empty set, then the result clearly holds, so assume F is nonempty. Let k_1 denote the length of the longest string in F, and let k_2 denote the smallest nonnegative integer such that L is k_2-reversible. Let $k = k_1 + k_2 + 1$. We shall show that $F \cup L$ is k-reversible using Theorem 14.

Suppose u_1vz and u_2vz are in $F \cup L$, where $|v| = k$. Then these strings must be in L, since their lengths exceed k_1. Since L is k_2-reversible and k is greater than k_2, for every string w, u_1vw is in L if and only if u_2vw is in L. But u_ivw is in L if and only if u_ivw is in $F \cup L$ for $i = 1, 2$, because of length considerations. Hence $F \cup L$ is k-reversible. □

LEMMA 21. *Let F be a prefix-free finite set of strings, G a suffix-free finite set of strings, and L a reversible language. Then FL and LG are reversible languages.*

PROOF. Since $LG = (G^rL^r)^r$ and G^r is a prefix-free finite set of strings, it suffices to prove that FL is reversible, since the reversible languages are closed under reversal. If F is the empty set, then FL is empty and therefore reversible, so assume that F is nonempty. Let k_1 denote the length of the longest string in F, and let k_2 denote the least nonnegative integer such that L is k_2-reversible. Let $k = k_1 + k_2$. We shall use Theorem 14 to show that FL is k-reversible.

Suppose u_1vz and u_2vz are in FL, where $|v| = k$. Write $v = v_1v_2$, where $|v_1| = k_1$ and $|v_2| = k_2$. There exist strings f_i and g_i such that $u_ivz = u_iv_1v_2z = f_ig_iv_2z$ for $i = 1, 2$, and, moreover, f_i is in F and g_iv_2z is in L for $i = 1, 2$. Since L is k_2-reversible, this implies that for every string w, g_1v_2w is in L if and only if g_2v_2w is in L.

Let an arbitrary string w be given. We shall show that if u_1vw is in FL then u_2vw is in FL, the converse being proved similarly. So, suppose that u_1vw is in FL. Since $u_1vw = f_1g_1v_2w$ and F is prefix-free, this implies that g_1v_2w is in L. Thus g_2v_2w is in L, and $f_2g_2v_2w = u_2vw$ is in FL. This shows that FL is k-reversible. □

The following example shows that the restriction of F to be prefix-free is essential in the above lemma.

Example 22. Let $L = (111)^*$ and $F = \{1, 11\}$. Then FL is not reversible, since for every k, $u_k = 1^{3k+1}$ and $v_k = 1^{3k+2}$ are in FL, but u_k1 is in FL while v_k1 is not. Thus, by Theorem 14, FL is not k-reversible for any k. □

THEOREM 23. *Let E, F, and G be finite sets of strings. Then $E \cup FU^*G$ is a reversible language, where U is the alphabet.*

PROOF. Let F' denote $\{u \in F$: for no v in F is v a proper prefix of $u\}$ and G' denote $\{u \in G$: for no v in G is v a proper suffix of $v\}$. Then F' is prefix-free, G' is

suffix-free, and $FU^*G = F'U^*G'$, where U is the alphabet. Thus, since U^* is zero-reversible, by the preceding two lemmas we have that $E \cup FU^*G$ is a reversible language. (Note that this gives an alternative proof of the fact that R_* contains the definite and reverse definite languages.) □

Thus R_* contains some of the generalized definite languages, but the following example shows that it does not contain all of them.

Example 24. Let U denote $\{a, b, c, d\}$ and $L = aU^*a + dU^*bca$. Then L is clearly generalized definite, but it is not reversible. To see this, note that for every $k > 0$, $a(bc)^ka$, $d(bc)^ka$, and $a(bc)^kba$ are in L, but $d(bc)^kba$ is not in L. Thus, by Theorem 14, L is not k-reversible for any k.

Since the set of strings of even parity is not generalized definite, we see that the classes of reversible and generalized definite languages are incomparable. The same is true of the reversible and locally testable languages, defined in [28].

5. *The Zero-Reversible Inference Algorithm*

In this section we describe, justify, and analyze the algorithm ZR to infer zero-reversible regular sets from positive samples. The generalization to k-reversible regular sets, which is somewhat more complicated, is treated in the next section.

The input to ZR is a finite nonempty set of strings S. The output is a particular deterministic acceptor $A = \mathrm{ZR}(S)$. Theorem 26 shows that $L(A)$ is the smallest zero-reversible language that contains S. Theorem 27 shows that using ZR at the finite stages of an infinite inference procedure leads to correct identification in the limit of the zero-reversible languages. Theorem 28 shows that ZR runs in nearly linear time. A simple incremental updating scheme for ZR is described.

5.1 The Algorithm ZR. On input S, ZR first constructs $A_0 = \mathrm{PT}(S)$, the prefix tree acceptor for S. It then constructs the finest partition π_f of the set Q_0 of states of A_0 with the property that A_0/π_f is zero-reversible, and outputs A_0/π_f.

To construct π_f, ZR beings with the trivial partition of Q_0 and repeatedly merges any two distinct blocks B_1 and B_2 such that either B_1 and B_2 both contain accepting states of A_0 or there exists a block B_3 and a symbol b such that there are b-successors (resp. b-predecessors) of states of B_3 in both B_1 and B_2. When there no longer remains any such pair of blocks, the resulting partition is π_f.

To implement this merging process efficiently, ZR keeps track of the further merges immediately implied by each merge performed. The variable LIST contains a pointer to a list of pairs of states whose corresponding blocks are to be merged. ZR initially selects some accepting state q' of A_0 and places on LIST all pairs (q', q) such that q is an accepting state of A_0 other than q'. This ensures that all blocks containing an accepting state of A_0 will eventually be merged.

For each block B of the current partition and each symbol $b \in U$, ZR maintains two quantities, $s(B, b)$ and $p(B, b)$, indicating the b-successors and b-predecessors of B. If there exists some state $q \in B$ such that $\delta_0(q, b)$ is defined, then $s(B, b)$ is some such $\delta_0(q, b)$; otherwise, $s(B, b)$ is the empty set. Similarly, if for some $q \in B$, $\delta_0^r(q, b)$ is defined, then $p(B, b)$ is defined to be some such $\delta_0^r(q, b)$; otherwise, $p(B, b)$ is the empty set. These quantities are initialized as $s(\{q\}, b) = \delta_0(q, b)$ and $p(\{q\}, b) = \delta_0^r(q, b)$ for all $q \in Q_0$ and $b \in U$.

After these initializations, ZR proceeds as follows. While the list LIST is nonempty, ZR removes the first pair of states (q_1, q_2). If q_1 and q_2 are already in the

same block of the current partition, ZR goes on to the next pair of states from LIST. Otherwise, the blocks containing q_1 and q_2, call them B_1 and B_2, are merged to form a new block B_3.

This action entails that LIST and the p- and s-values be updated as follows. For each $b \in U$, if $s(B_1, b)$ and $s(B_2, b)$ are both nonempty, then the pair $(s(B_1, b), s(B_2, b))$ is added to LIST. Similarly, if $p(B_1, b)$ and $p(B_2, b)$ are both nonempty, then the pair $(p(B_1, b), p(B_2, b))$ is added to LIST. Also, if either $s(B_1, b)$ or $s(B_2, b)$ is nonempty, then an element q is chosen from one of them and $s(B_3, b)$ is set to q; otherwise, $s(B_3, b)$ is set to the empty set. Similarly, $p(B_3, b)$ is defined according to $p(B_1, b)$ and $p(B_2, b)$. After this updating, ZR goes on to the next pair of states from LIST.

When LIST becomes empty, the current partition is π_f. ZR outputs A_0/π_f and halts.

A somewhat more formal description of ZR may now be given.

Algorithm ZR

Input. a nonempty positive sample S.

Output a zero-reversible acceptor A.

```
* Initialization
Let A_0 = (Q_0, I_0, F_0, δ_0) be PT(S)
Let π_0 be the trivial partition of Q_0
For each b ∈ U and q ∈ Q_0 let s({q}, b) = δ_0(q, b) and p({q}, b) = δ_0^{-1}(q, b)
Choose some q' ∈ F_0.
Let LIST contain all pairs (q', q) such that q ∈ F_0 − {q'}.
Let i = 0.
* Merging
While LIST ≠ ∅ do
begin
    Remove some element (q_1, q_2) from LIST
    Let B_1 = B(q_1, π_i), B_2 = B(q_2, π_i)
    If B_1 ≠ B_2 then
    begin
        Let π_{i+1} be π_i with B_1 and B_2 merged
        For each b ∈ U, s-UPDATE(B_1, B_2, b) and p-UPDATE(B_1, B_2, b)
        Increase i by 1
    end
end
* Termination
Let f = i and output the acceptor A_0/π_f
```

The procedure s-UPDATE(B_1, B_2, b) places $(s(B_1, b), s(B_2, b))$ on LIST if both $s(B_1, b)$ and $s(B_2, b)$ are nonempty and defines $s(B_3, b)$ to be $s(B_1, b)$ if this is nonempty and $s(B_2, b)$ otherwise (where B_3 is the union of B_1 and B_2). The procedure p-UPDATE is defined similarly, with p in place of s. The description of the algorithm ZR is now complete, and we turn to analyzing its correctness and running time.

5.2. THE CORRECTNESS OF ZR. In this section we show that ZR correctly finds the smallest zero-reversible language that contains the input sample. The following lemma may be interpreted as saying that the algorithm ZR performs the minimal generalization of the sample that produces a zero-reversible inference.

LEMMA 25. *Let S be any nonempty positive sample, A_0 the prefix tree acceptor $PT(S)$ for S, and π_f the final partition found by ZR on input S. Then π_f is the finest partition π such that A_0/π is zero-reversible.*

PROOF. Let $A_0 = (Q_0, I_0, F_0, \delta_0)$. If the pair (q_1, q_2) is ever placed on LIST, then q_1 and q_2 must be in the same block of the final partition, that is, $B(q_1, \pi_f) = B(q_2, \pi_f)$. Also, it is not difficult to verify by induction on i that for $i = 0, 1, \ldots, f$, if q_1 and q_2 are distinct elements of $\delta_0(B, b)$ (resp. $\delta_0^r(B, b)$) for some block B of π_i and symbol $b \in U$, then there exists a chain $q_1', q_2', \ldots, q_n'$ of elements of $\delta_0(B, b)$ (resp. $\delta_0^r(B, b)$) such that $q_1 = q_1'$, $q_2 = q_n'$, and for each j, $0 \leq j < n$, either (q_j', q_{j+1}') or (q_{j+1}', q_j') is placed on LIST prior to or during the construction of π_i.

Therefore the initialization guarantees that all the accepting states of A_0 are in the same block of π_f, so A_0/π_f has exactly one accepting state. Also, for each block B of π_f and symbol $b \in U$, all the elements of $\delta_0(B, b)$ (resp. $\delta_0^r(B, b)$) are contained in one block of π_f. Thus A_0/π_f is zero-reversible.

It remains to show that if π is any partition of Q_0 such that A_0/π is zero-reversible, then π_f refines π. We prove by induction that π_i refines π for $i = 0, 1, \ldots, f$. Clearly π_0, the trivial partition of Q_0, refines π. Suppose $\pi_0, \pi_1, \ldots, \pi_i$ all refine π and π_{i+1} is obtained from π_i in the course of processing entry (q_1, q_2) from LIST. Thus π_{i+1} is obtained from π_i by merging the blocks $B(q_1, \pi_i)$ and $B(q_2, \pi_i)$. Since π_i refines π, $B(q_1, \pi_i)$ is a subset of $B(q_1, \pi)$ and $B(q_2, \pi_i)$ is a subset of $B(q_2, \pi)$, so to show that π_{i+1} refines π, it suffices to show that $B(q_1, \pi) = B(q_2, \pi)$.

Either (q_1, q_2) was first placed on LIST during the initialization stage or not. If so, then q_1 and q_2 are both accepting states, and since A_0/π is zero-reversible, it has only one accepting state, so $B(q_1, \pi) = B(q_2, \pi)$. Otherwise, (q_1, q_2) was first placed on LIST in consequence of some previous merge, let us say the merge to produce π_j from π_{j-1}, where $0 < j \leq i$. Then $(q_1, q_2) = (s(B_1, b), s(B_2, b))$ (resp. $(p(B_1, b), p(B_2, b))$), where B_1 and B_2 are the blocks of π_{j-1} merged in forming π_j and b is some symbol. Then q_1 and q_2 are b-successors (resp. b-predecessors) of two states in some block B of π_j. Since π_j refines π by the induction hypothesis, q_1 and q_2 are b-successors (resp. b-predecessors) of some block B' in π, and since A_0/π is zero-reversible, $B(q_1, \pi) = B(q_2, \pi)$. Thus in either case π_{i+1} refines π, and by induction we conclude that π_f refines π. □

THEOREM 26. *Let S be a nonempty positive sample, and let A_f be the acceptor output by algorithm ZR on input S. Then $L(A_f)$ is the smallest zero-reversible language containing S.*

PROOF. The preceding lemma shows that $L(A_f)$ is a zero-reversible language containing S. Let L be any zero-reversible language containing S, and let π be the restriction of the partition π_L to the elements of $\mathrm{Pr}(S)$. If A_0 denotes the prefix tree acceptor for S, then Lemma 1 shows that A_0/π is isomorphic to a subacceptor of $A(L)$, and Corollary 2 shows that $L(A_0/\pi)$ is contained in L. Lemma 6 shows that $A(L)$ is zero-reversible, and therefore A_0/π is zero-reversible, by Remark 4. By the above lemma, π_f therefore refines π, so $L(A_0/\pi_f) = L(A_f)$ is contained in $L(A_0/\pi)$. Consequently, $L(A_f)$ is contained in L, and $L(A_f)$ is the smallest zero-reversible language containing S. □

5.3 IDENTIFICATION IN THE LIMIT OF THE ZERO-REVERSIBLE LANGUAGES. In this section we show that the algorithm ZR may be used at the finite stages of an infinite inference process to identify the zero-reversible languages in the limit. The idea is simply to run ZR on the sample at the nth stage and output the result as the nth guess. Define an operator ZR_∞ from infinite sequences of strings $w_1, w_2, w_3, \ldots$ to infinite sequences of acceptors $A_1, A_2, A_3, \ldots$ by

$$A_n = \mathrm{ZR}(\{w_1, w_2, \ldots, w_n\}) \qquad \text{for all} \quad n \geq 1.$$

It is clear that the operator ZR_∞ is effective; the nth output is the result of running the algorithm ZR on the first n input strings. (Later we show how A_{n+1} may be obtained from A_n and w_{n+1} by a simple updating scheme based on ZR.) We need to show that this converges to a correct guess after a finite number of stages.

An infinite sequence of strings $w_1, w_2, w_3, \ldots$ is defined to be a *positive presentation* of a language L if and only if the range of the sequence is precisely L, that is, every element of the sequence is an element of L and vice versa. An infinite sequence of acceptors $A_1, A_2, A_3 \ldots$ is said to *converge to* an acceptor A if and only if there exists an integer N such that for all $n \geq N$, A_n is isomorphic to A. The result that ZR_∞ correctly identifies the zero-reversible languages in the limit from positive data is formulated as follows.

THEOREM 27. *Let L be a nonempty zero-reversible language, $w_1, w_2, w_3, \ldots$ a positive presentation of L, and $A_1, A_2, A_3, \ldots$ the output of ZR_∞ on this input. Then $A_1, A_2, A_3, \ldots$ converges to the canonical acceptor $A(L)$ for L.*

PROOF. By Theorem 9, L contains a characteristic sample. Let N be sufficiently large that $\{w_1, w_2, \ldots, w_N\}$ contains a characteristic sample for L. For $n \geq N$, $L(A_n)$ is the smallest zero-reversible language containing $\{w_1, w_2, \ldots, w_n\}$, by definition of ZR_∞ and Theorem 26. Thus $L(A_n) = L$, by the definition of a characteristic sample. Moreover, it is easily checked that the acceptor output by ZR is stripped, and therefore canonical, by Lemma 5. Hence A_n is isomorphic to $A(L)$ for all $n \geq N$, so $A_1, A_2, A_3, \ldots$ converges to $A(L)$. □

5.4 THE RUNNING TIME OF ZR

THEOREM 28. *The algorithm ZR may be implemented to run in time $O(n\alpha(n))$, where n is one more than the sum of the lengths of the input strings and α is a very slowly growing function. (Tarjan [32] defines α.)*

PROOF. Let S be the set of input strings, and let n be one more than the sum of the lengths of the strings in S. The prefix tree acceptor $A_0 = \mathrm{PT}(S)$ may be constructed in time $O(n)$ and contains at most n states. Similarly, the time to output the final acceptor is $O(n)$. The partitions π_i of the states of A_0 may be queried and updated using the collapsing FIND and weighted UNION operations described and analyzed by Tarjan [32]. Processing each pair of states from LIST entails two FIND operations to determine the blocks containing the two states. If the blocks are distinct, which can happen at most $n - 1$ times, they are merged with a UNION operation, and $O(|U|)$ further processing may place at most $2|U|$ new pairs on LIST. Thus a total of at most $(2|U| + 1)(n - 1)$ pairs must be processed. Thus at most $(4|U| + 2)(n - 1)$ FIND operations and $n - 1$ UNION operations are required, which requires a total time of $O(n\alpha(n))$. □

5.5 UPDATING A GUESS. It is useful in the context of the process ZR_∞ to show that ZR may be modified to have good incremental behavior. That is, given the output A_f computed by ZR on input S, and given a new string w, we may easily update A_f to be the output computed by ZR on input $S' = S \cup \{w\}$. The method for doing this is to start at the initial state of A_f and follow the transitions A_f makes on the input string w. If no undefined transitions are encountered and the last state reached is the accepting state, then A_f already accepts w and nothing need be done. Otherwise, add new states and transitions for each symbol of w starting with the first undefined transition (if any). Mark the last state reached by w as accepting, and place the pair consisting of this state and the accepting state of A_f on LIST. Continue

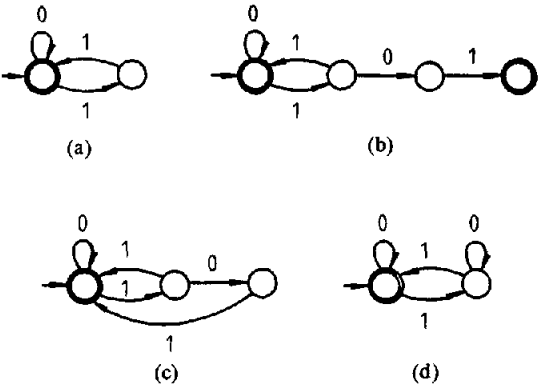

FIG. 5 Updating a guess.

the merging portion of the algorithm ZR until LIST is empty, and output A_f/π', where π' is the final partition of the states of A_f. The correctness of this procedure is verified in the same way as that of the original algorithm ZR, since the order of detecting and performing required merges is immaterial.

Example 29. If we run ZR on the input {0, 00, 11, 1100}, we obtain the acceptor shown in Figure 5a. If we then add the string 101 to the sample and perform the updating procedure just described, we first obtain the acceptor shown in Figure 5b. This is then "folded up" as shown in Figure 5c and d to obtain as a final result an acceptor for strings with an even number of 1's. □

6. *The k-Reversible Inference Algorithms*

Let k be a fixed positive integer. We describe an algorithm, k-RI, to infer k-reversible sets from positive samples. Let S be a nonempty positive sample. On input S, k-RI constructs the prefix tree acceptor for S, $A_0 = \mathrm{PT}(S)$, and finds the finest partition π_f of the states of A_0 such that A_0/π_f is k-reversible. It then outputs a canonical acceptor for $L(A_0/\pi_f)$ and halts.

To find π_f, k-RI begins with the trivial partition and repeatedly merges any two blocks that violate the conditions for k-reversibility in the quotient acceptor A_0/π. When no pair of blocks is found to violate these conditions, the resulting partition is π_f.

More precisely, let S be the set of input strings. If S is empty, then k-RI outputs the empty acceptor and halts. Otherwise, it constructs the prefix tree acceptor for S, say $A_0 = (Q_0, I_0, F_0, \delta_0)$. It takes π_0 to be the trivial partition of Q_0 and i to be equal to 0. While there exist two distinct blocks B_1 and B_2 of π_i such that either

(1) for some $b \in U$ and $B_3 \in \pi_i$, B_1 and B_2 are both b-successors of B_3 in A_0/π_i, or
(2) B_1 and B_2 have a common k-leader in A_0/π_i, and either B_1 and B_2 are both final states of A_0/π_i or there exists a block B_3 of π_i and a symbol $b \in U$ such that B_3 is a b-successor of both B_1 and B_2 in A_0/π_i,

k-RI constructs π_{i+1} by merging B_1 and B_2 in π_i, and increments i by 1. When no such pair of blocks exists, k-RI sets f to the final value of i. It then constructs and outputs a canonical acceptor for $L(A_0/\pi_f)$.

6.1 CORRECTNESS, IDENTIFICATION IN THE LIMIT, ANALYSIS. Since this treatment is largely analogous to the zero-reversible case, an abbreviated presentation is given.

LEMMA 30. *Let S be a nonempty positive sample, A_0 the prefix tree acceptor for S, and π_f the final partition found by k-RI on input S. Then π_f is the finest partition of the states of A_0 such that A_0/π_f is k-reversible.*

PROOF. It is clear that A_0/π_f is k-reversible, since k-RI will continue to merge blocks as long as the conditions for k-reversibility are violated. Suppose π is any partition of the states of A_0 such that A_0/π is k-reversible. As in the proof of Lemma 25, it is not difficult to show by induction that π_i refines π for $i = 0, 1, \ldots, f$. □

THEOREM 31. *Let S be a nonempty positive sample, and let A_f be the acceptor output by k-RI on input S. Then $L(A_f)$ is the smallest k-reversible language containing S.*

PROOF. This proof is completely analogous to the proof of Theorem 26 and is omitted. □

We define an operator k-RI$_\infty$ from infinite sequences $w_1, w_2, w_3, \ldots$ of strings to infinite sequences $A_1, A_2, A_3, \ldots$ of acceptors by

$$A_n = k\text{-RI}(\{w_1, w_2, \ldots, w_n\}) \qquad \text{for all} \quad n \geq 1.$$

It is clear that k-RI$_\infty$ is effective.

THEOREM 32. *Let L be a nonempty k-reversible language and $w_1, w_2, w_3, \ldots$ any positive presentation of L. On this input, the output $A_1, A_2, A_3, \ldots$ of k-RI$_\infty$ converges to $A(L)$.*

PROOF. Let N be sufficiently large that $\{w_1, w_2, \ldots, w_N\}$ contains a characteristic sample for the k-reversible language L. For each $n \geq N$, $L(A_n)$ is the smallest k-reversible language containing $\{w_1, w_2, \ldots, w_n\}$, so $L(A_n) = L$. By definition of k-RI, A_n is canonical, so A_n is isomorphic to $A(L)$ for all $n \geq N$. Thus $A_1, A_2, A_3, \ldots$ converges to $A(L)$. □

THEOREM 33. *The algorithm k-RI may be implemented to run in time $O(kn^3)$, where n is one more than the sum of the lengths of the input strings.*

PROOF. We represent each quotient acceptor A_0/π_i by a directed graph G_i with one node for each block of π_i and an edge labeled b from B_1 to B_2 if and only if B_2 is a b-successor of B_1 in A_0/π_i. Each accepting node is so marked. The nodes are numbered by integers in the range 1 to n. It suffices to keep a list of the current nodes, a (possibly redundant) list of the current edges (where (b, x, y) represents edge labeled b from node x to node y), and a Boolean vector indexed by node numbers indicating whether each node is accepting or not.

For each G_i we compute some auxiliary quantities: a list H_i, and $k + 1$ matrices $C_i^0, C_i^1, \ldots, C_i^k$. For each ordered pair (b, x, y) and (b', x', y') of edges of G_i (not necessarily distinct), if $b = b'$, then put the pair $((x, x'), (y, y'))$ on the list H_i. $C_i^0[x, y]$ is defined to be 1 for all pairs (x, y) of nodes of G_i. For each r, $0 \leq r < k$, C_i^{r+1} is defined from C_i^r as follows. Initialize $C_i^{r+1}[x, y]$ to be 0 for all pairs (x, y) of nodes of G_i. Then for each pair $((x, x'), (y, y'))$ from H_i, if $C_i^r[x, x'] = 1$, then set $C_i^{r+1}[y, y'] = 1$.

It is not difficult to verify that $((x, x'), (y, y'))$ is on the list H_i if and only if there exists a symbol $b \in U$ such that x is a b-predecessor of y and x' is a b-predecessor

of y' in G_i. Also, for each pair (x, y) of nodes of G_i and for each r, $0 \leq r \leq k$, $C_i^r[x, y] = 1$ if and only if there exists a common r-leader of x and y in G_i.

Thus, once these quantities have been computed, we may test for the existence of a pair of nodes of G_i violating the conditions on k-reversibility as follows. For each pair (b, x, y) and (b', x', y') of edges of G_i, if $b = b'$, $x = x'$, and $y \neq y'$, then y and y' give a violation of determinism and should be merged. For each pair $x \neq y$ of nodes of G_i that are both accepting, if $C_i^k[x, y] = 1$, then x and y have a common k-leader in G_i and should be merged. Similarly, for each pair (b, x, y) and (b', x', y') of edges of G_i such that $b = b'$, $y = y'$, $x \neq x'$, and $C_i^k[x, x'] = 1$, x and x' have a common b-successor and a common k-leader in G_i and should be merged.

In order to merge x and y in G_i, where $x < y$, we scan all the edges on the edge list of G_i and replace each occurrence of y by x. If y is accepting, then x is also marked accepting. Finally, y is removed from the node list for G_i. The resulting graph is G_{i+1}. The auxiliary quantities $H_{i+1}, C_{i+1}^0, \ldots, C_{i+1}^k$ are computed, and the search begins for another pair of nodes to merge. When no pair remains to be merged, the graph is converted to an acceptor and minimized by a standard algorithm [1].

The initial graph G_0 has at most n nodes and n edges, and its node and edge lists and acceptance vector may be computed in time $O(n)$. Thus the node and edge lists for all of the graphs G_i contain at most n elements. Computing the auxiliary quantities H_i and $C_i^0, \ldots, C_i^k$ from G_i may be done in time $O(kn^2)$. Searching for a pair to merge may be done in time $O(n^2)$, and updating the node and edge lists and the acceptance vector when two nodes are merged takes time $O(n)$. Since at most $n - 1$ merges may be performed, the total time to find the final graph is $O(kn^3)$, which dominates the time required to convert it to an acceptor and minimize it. □

We note that the approach to updating a guess used for zero-reversible inference works also in this more general case. In practice it will probably prove useful to remove duplicates from the edge lists of the quotient acceptors, since in general they may be expected to shrink dramatically as merges are performed.

7. *Using Negative Data*

Theorem 32 shows that for each k the k-reversible languages can be correctly identified in the limit from positive data. It is however *not* the case that the whole class of reversible languages, R_*, can be correctly identified in the limit from positive data, as the general results in [4, 18] on identification in the limit from positive data show.

If we consider the problem of inferring the reversible languages from positive and negative data, one approach immediately presents itself. That is to construct the k-reversible inference from the positive examples for $k = 0, 1, 2, \ldots$ until a language is found that does not contain any of the negative examples.

We define a *positive and negative sample* to be an ordered pair (S_0, S_1) such that S_0 and S_1 are disjoint finite sets of strings. We describe an algorithm RI whose input is a positive and negative sample (S_0, S_1), and whose output is a reversible acceptor A that accepts all the strings in S_1 and none of the strings in S_0. If $S_1 = \varnothing$, then RI outputs the empty acceptor and halts. Otherwise, for each $k = 0, 1, 2, \ldots$, RI calls k-RI on the set S_1 to produce an acceptor A_k, until an acceptor A_k is found that does not accept any of the strings in S_0. This final acceptor is the output of RI. (Note that 0-RI is the algorithm ZR.) The correctness of this algorithm is formulated as follows.

THEOREM 34. *With the positive and negative sample (S_0, S_1) as input, the algorithm RI finds the smallest k such that there exists a k-reversible language containing S_1 and*

disjoint from S_0, and outputs a canonical acceptor for the smallest k-reversible language with this property.

PROOF. This is immediate from Theorems 26 and 31 and the order in which RI searches for an acceptor with the required properties. □

The running time of RI is also easily established.

THEOREM 35. *The algorithm RI runs in time $O(m^2n^3)$, where m is one more than the final value of k found and n is one more than the sum of the lengths of the input strings. In particular, $m \leq n$.*

PROOF. Let (S_0, S_1) be an input to RI. The total time spent in calls to k-RI for $k = 0, 1, \ldots, m-1$ is $O(m^2n^3)$, by Theorems 28 and 33. The total time spent in testing whether the resulting acceptors contain an element of S_0 is $O(mn)$, since following the transitions of a string through a deterministic acceptor may be done in time linear in the length of the string. Since S_0 and S_1 are disjoint, if $k \geq n-1$, then the prefix tree acceptor for S_1 is a k-reversible acceptor whose language contains S_1 and is disjoint from S_0. Thus $m \leq n$. □

Now we turn to the question of identification in the limit. We need a notion of a presentation of a language containing positive and negative examples. A *complete presentation* of a language L is an infinite sequence $(w_1, e_1), (w_2, e_2), (w_3, e_3), \ldots$ such that w_i is a string, e_i is either 0 or 1, $e_i = 1$ if and only if $w_i \in L$, and for every string $u \in U^*$ there exists an index i such that $u = w_i$. We define two related functions, giving the positive and negative examples in initial segments of the complete presentation:

$$S_0(n) = \{w_i : 1 \leq i \leq n,\ e_i = 0\},$$
$$S_1(n) = \{w_i : 1 \leq i \leq n,\ e_i = 1\}.$$

We may define an effective operator RI_∞ that maps complete presentations to infinite sequences of acceptors $A_1, A_2, A_3, \ldots$ such that

$$A_n = \mathrm{RI}(S_0(n), S_1(n)) \qquad \text{for all} \quad n \geq 1,$$

where $S_0(n)$ and $S_1(n)$ are the functions associated with the given complete presentation. (The behavior of RI_∞ on sequences of ordered pairs that are not complete presentations of some language is not specified.)

The following lemma shows that RI_∞ searches at each stage through a descending chain of languages.

LEMMA 36. *Let S be a positive sample, and let $A = j\text{-}RI(S)$ and $A' = k\text{-}RI(S)$, where $j < k$. Then $L(A)$ contains $L(A')$.*

PROOF. $L(A')$ is the smallest k-reversible language containing S. $L(A)$ is a j-reversible, and therefore k-reversible, language containing S. Thus $L(A')$ is contained in $L(A)$. □

THEOREM 37. *Let L be a reversible language, and let $(w_1, e_1), (w_2, e_2), \ldots$ be a complete presentation of L. Let $A_1, A_2, A_3, \ldots$ be the output of RI_∞ on this input. Then $A_1, A_2, A_3, \ldots$ converges to $A(L)$.*

PROOF. The result holds trivially if $L = \emptyset$, so assume $L \neq \emptyset$. Let k be the least nonnegative integer for which L is k-reversible. If $k = 0$, then the guess of RI_∞ will always be the zero-reversible guess, which converges to $A(L)$ by Theorem 27.

Suppose $k \geq 1$. Let N be sufficiently large that $S_1(N)$ contains a characteristic sample for L as a k-reversible language. Consider L', the language accepted by the acceptor $(k-1)$-RI$(S_1(N))$. This language contains L by the preceding lemma and the fact that $L = k$-RI$(S_1(N))$. However, since L' is $(k-1)$-reversible, it is not equal to L. Choose some element w' that is in L' but not in L. Let $N' \geq N$ be sufficiently large that $S_0(N')$ contains w'.

For each $n \geq N'$, RI_∞ computes j-RI$(S_1(n))$ for $j = 0, 1, \ldots, k-1$ and finds that they each contain the element w' of $S_0(n)$. Hence RI_∞ computes and outputs k-RI$(S_1(n))$, which by Theorem 32 is a canonical acceptor for L. Thus in this case also $A_1, A_2, A_3, \ldots$ converges to $A(L)$. □

Thus RI_∞ does correct identification in the limit of the reversible languages from positive and negative data. Its behavior in each finite stage is reasonable, both in terms of the characterization of the guess and the time required to compute it. In practice, its running time could probably be improved by incorporating the incremental updating scheme of the algorithms ZR and k-RI for new positive examples and by keeping track of the largest value of k required so far to use as the starting place for the search for new acceptors in the case when a negative example invalidates the current guess.

Other approaches to the use of negative data should also be considered. For example, an approach that overcame the asymmetry of RI in the use of positive and negative data would be interesting.

8. *Some Comparisons*

The zero-reversible inference algorithm is based on a heuristic originally proposed by Feldman [11, 13], so it is not surprising that it duplicates the performance of the heuristic on the primary examples given in those papers, which we reproduce below:

$$L_1 = \{\text{strings of } a\text{'s and } b\text{'s with an even number of } a\text{'s}\},$$
$$S_1 = \{b, bb, aa, baa, aba, baba, abba, bbaba, bbaa, aabb\},$$
$$L_2 = (c + bb)a^*b,$$
$$S_2 = \{cb, bbb, cab, bbab, caab, bbaab, caaab\}.$$

Itoga [25] describes some experiments with a variant of Feldman's heuristic that is rather sensitive to the order of presentation of the data. The primary example studied is a seven-state transducer over $\{a, b, c\}$ that outputs 1 if it has seen the three symbols a, b, c (in any order) since the beginning of the string or the last time it output 1, whichever is later, and otherwise outputs 0. Itoga's method correctly identifies this transducer from a particular sample of 104 strings. If we convert this to an acceptor in the standard way, the resulting language is not reversible, so our methods do not apply.

Biermann and Feldman [5] describe a method of inferring regular sets from positive data by merging states with equal sets of k-tails, starting with an acceptor derived from the sample strings. The set of k-tails of a state is simply the set of strings of length k or less that are accepted from that state. The final acceptor is generally nondeterministic. The user specifies the value of k that is used by the method; for each regular set, k may be set sufficiently large to guarantee correct identification in the limit of that regular set.

Miclet [29] describes an approach to inferring regular sets from positive data by merging states according to a similarity measure on their sets of k-tails (for k chosen by the algorithm and decremented at each stage). The general approach is quite

flexible and interesting; however, the particular heuristic implementation illustrated in the paper seems to be dominated in its performance by the zero-reversible algorithm. The three examples considered in the paper are all zero-reversible, and are correctly identified from the given data by the zero-reversible algorithm:

$L_3 = a(bc)^*d$,
$S_3 = \{abcd, abcbcd\}$,
$L_3 = (aa + ba(aa)^*c)^*$,
$S_4 = \{aabac, baaac, bacbac, aabaaac, bacaabac, aaaaaabac,$
$aaaabaaac, baaaaaaac, bacbacbac, bacaabaaaaac\}$,
$L_5 = \{$strings with an even number of a's and an odd number of b's$\}$,
$S_5 = \{aba, bbb, abbab, bbaba, bbbbb, abaabab, babbbaaab,$
$bbaaabbab, bbbaaabab, baabbbbbaba\}$,

(The tail-clustering method described by Miclet requires another string added to S_5 in order to identify L_5 correctly.)

These comparisons suggest that the class of zero-reversible languages captures an interesting class of phenomena in the inference of regular sets from positive data, though it certainly does not exhaust the domain. It is hoped that a clear understanding of this class will help to stimulate investigation of other types of phenomena in this domain.

9. *Conclusions*

We have presented efficient algorithms to infer k-reversible languages from positive data and reversible languages from positive and negative data. The zero-reversible inference algorithm is of particular interest because of the relation between zero-reversible languages and automata whose syntactic monoids are permutation groups. This suggests that it may be possible to use the algebraic properties of automata to help characterize their "inferability." It is interesting to contrast these results with those of Crespi-Reghizzi et al. [10], which specifically concern "noncounting" (i.e., permutation-free) languages. Perhaps there will be a useful synthesis of these two approaches.

The algorithms and implementations we have given are fairly straightforward and may not be the most efficient possible. However, this is appropriate in a domain such as this one, where much of the current work must be viewed as exploratory feasibility studies. Time and further experience will tell whether these algorithms are indeed solving the "right" problems. For this, experience with computer implementations of relatively straightforward algorithms may be more relevant at present than a search for faster algorithms. However, faster algorithms for these problems would certainly be welcome.

It should be evident that inductive inference swarms with unsolved, mostly ill-defined, problems. This paper will have achieved one of its primary goals if it has helped to indicate how one line of approach may lead to well-defined problems and solid theoretical progress.

ACKNOWLEDGMENTS. Talks with Manuel Blum and Ehud Shapiro have been valuable in the development of these ideas and results. The comments of the referees have been helpful in improving the correctness and completeness of the paper.

REFERENCES

1 AHO, A V, HOPCROFT, J E, AND ULLMAN, J D. *The Design and Analysis of Computer Algorithms* Addison-Wesley, Reading, Mass., 1974

2. ANGLUIN, D On the complexity of minimum inference of regular sets. *Inf. Control 39* (1978), 337–350
3 ANGLUIN, D Finding patterns common to a set of strings *J Comput. Syst Sci. 21* (1980), 46–62.
4 ANGLUIN, D Inductive inference of formal languages from positive data *Inf Control 45* (1980), 117–135
5. BIERMANN, A W , AND FELDMAN, J.A On the synthesis of finite-state machines from samples of their behavior *IEEE Trans. Comput C-21* (1972), 592–597
6 BLUM, L , AND BLUM, M. Toward a mathematic theory of inductive inference. *Inf. Control 28* (1978), 125–155.
7 CASE, J., AND SMITH, C Anomaly hierarchies of mechanized inductive inference. Proc. 10th ACM Symp on Theory of Computing, San Diego, Calif., 1978, pp 314–319.
8 COHEN, R.S , AND BRZOZOWSKI, J A. General properties of star height of regular events. *J Comput. Syst. Sci 4* (1970), 260–280.
9 CRESPI-REGHIZZI, S An effective model for grammar inference. In *Information Processing*, North Holland, 1972, pp 524–529
10. CRESPI-REGHIZZI, S , GUIDA, G , AND MANDRIOLI, D Noncounting context-free languages. *J. ACM 25*, 4 (Oct. 1978), 571–580.
11 FELDMAN, J.A First thoughts on grammatical inference Tech. Rep., Artificial Intelligence Project, Stanford Univ., Stanford, Calif., 1967
12. FELDMAN, J A Some decidability results in grammatical inference *Inf. Control 20* (1972), 244–262.
13 FELDMAN, J.A., GIPS, J., HORNING, J.J., AND REDER, S Grammatical complexity and inference Tech Rep , Computer Science Dep., Stanford Univ , Stanford, Calif., 1969.
14 FU, K S *Syntactic Methods in Pattern Recognition* Academic Press, New York, 1975.
15 FU, K S *Syntactic Pattern Recognition, Applications* Springer-Verlag, New York, 1977.
16 FU, K S , AND BOOTH, T.L. Grammatical inference· Introduction and survey, Parts 1 and 2 *IEEE Trans. Syst., Man Cybern SMC-5* (1975), 95–111, 409–423.
17 GINZBURG, A. About some properties of definite, reverse-definite and related automata. *IEEE Trans Electron. Comput. EC-15* (1966), 806–810
18. GOLD, E M Language identification in the limit *Inf Control 10* (1967), 447–474
19 GOLD, E.M. Complexity of automaton identification from given data *Inf. Control 37* (1978), 302–320
20 GONZALEZ, R.C , AND THOMASON, M.G. *Syntactic Pattern Recognition, An Introduction.* Addison-Wesley, Reading, Mass , 1978.
21 HARRISON, M.A *Introduction to Switching and Automata Theory.* McGraw-Hill, New York, 1965
22 HARRISON, M.A. *Introduction to Formal Language Theory* Addison-Wesley, Reading, Mass , 1978.
23. HARTMANIS, J., AND STEARNS, R.E *Algebraic Theory of Sequential Machines* Prentice-Hall, Englewood Cliffs, N J , 1966
24. HOPCROFT, J E , AND ULLMAN, J.D *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, Mass., 1979.
25 ITOGA, S Y. A new heuristic for inferring regular grammars. *IEEE Trans Pattern Anal. Mach. Intell. PAMI-3* (1981), 191–197.
26 MCNAUGHTON, R. The loop complexity of pure-group events. *Inf. Control 11* (1967), 167–176
27 MCNAUGHTON, R Parenthesis grammars *J ACM 14*, 3 (July 1967), 490–500
28 MCNAUGHTON, R , AND PAPERT, S *Counter-Free Automata* M.I T Press, Cambridge, Mass., 1971.
29. MICLET, L. Regular inferences with a tail-clustering method *IEEE Trans Syst. Man Cybern. SMC-10* (1980), 737–743
30 PERLES, M , RABIN, M.O , AND SHAMIR, E. The theory of definite automata. *IEEE Trans. Electron Comput EC-12* (1963), 233–243
31 SMITH, C.H An inductive inference bibliography Tech Rep CSD TR 323, Computer Science Dep., Purdue Univ , Lafayette, Ind , 1979
32. TARJAN, R E Efficiency of a good but not linear set union algorithm *J ACM 22*, 2 (Apr 1975), 215–225
33 ZALCSTEIN, Y Locally testable languages *J. Comput. Syst. Sci 6* (1972), 151–167

RECEIVED SEPTEMBER 1980; REVISED APRIL 1981; ACCEPTED SEPTEMBER 1981