But

$$\mathrm{sal}\,(p, t) = \psi[P'(1 + X)T] = \psi[P(1 + X)T + X^{q+1} \cdot T]$$
$$= \psi[P(1 + X)T]\ \mathrm{sexp}\,(t_{-q-2}).$$

Thus

$$\mathrm{cal}\,(p, t - \alpha) = \mathrm{sal}\,(p, t).$$

VI. Conclusion

The definition given here, from which the others are easily derived, shows close relationships between the properties of the polynomials (or binary codes) and those of the Walsh function.

Note too that the changing in (3) and (4) of T into T' (T' being defined from t in the same manner as P' from) p only results in a change of the values taken at the discontinuity points.

So we suggest a more symmetrical definition:

$$\mathrm{cal}\,(p, t) = \mathrm{cal}\,(t, p) = \psi[T \cdot P(1 + X)]$$
$$\mathrm{sal}\,(t, p) = \mathrm{sal}\,(p, t) = \psi[T' \cdot P'(1 + X)].$$

References

[1] N. J. Fine, "On the Walsh functions," *Trans. Amer. Math. Soc.*, vol. 65, 1949.

[2] I. Flores, "Reflected number system," *IRE Trans. Electron. Comput.*, vol. EC-5, pp. 79–82, June 1956.

[3] H. F. Harmuth, *Transmission of Information by Orthogonal Functions.* New York: Springer, 1970.

[4] K. W. Henderson, "Some notes on the Walsh functions," *IEEE Trans. Electron. Comput.*, vol. EC-13, pp. 50–52, Feb. 1964.

[5] M. Koenic and J. Zalesio, "Les fonctions de Walsh," presented at *Sém. Théorie des Signaux*, Nice, France, 1970.

[6] R. Lackey and D. Meltzer, "A simplified definition of Walsh functions," *IEEE Trans. Electron. Comput.*, vol. C-20, pp. 211–213, Feb. 1971.

[7] J. Pearl, "Application of Walsh transform to statistical analysis," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-1, pp. 111–119, Apr. 1971.

[8] W. W. Peterson, *Error Correcting Codes.* New York: Wiley, 1961.

[9] R. C. Titsworth, "A Boolean function—multiplexed telemetry system," *IEEE. Trans. Space Electron. Telem.*, vol. SET-9, pp. 42–45, June 1963.

[10] J. L. Walsh, "A closed set of orthogonal functions," *Amer. J. Math.*, vol. 45, 1923.

On the Synthesis of Finite-State Machines from Samples of Their Behavior

A. W. BIERMANN AND J. A. FELDMAN

Abstract—The Nerode realization technique for synthesizing finite-state machines from their associated right-invariant equivalence relations is modified to give a method for synthesizing machines from finite subsets of their input–output behavior. The synthesis procedure includes a parameter that one may adjust to obtain machines that represent the desired behavior with varying degrees of accuracy and that consequently have varying complexities. We discuss some of the uses of the method, including an application to a sequential learning problem.

Index Terms—Finite-state functions, finite-state machines, inference, Nerode realization, sequential learning, synthesis.

Manuscript received June 2, 1970; revised June 21, 1971. The research reported here was done partly in the Department of Computer Science, Stanford University, and was supported by the Advanced Research Projects Agency of the Office of the Secretary of Defense (SD-183). Part of the work was done in the Department of Computer and Information Science, Ohio State University, and was supported by the Office of Science Information Service, National Science Foundation, under Grant GN 534.1.

A. W. Biermann is with the Department of Computer and Information Science, Ohio State University, Columbus, Ohio 43210.

J. A. Feldman is with the Department of Computer Science, Stanford University, Stanford, Calif. 94305.

I. Introduction

Nerode [15] has given a method for synthesizing finite-state machines from their associated right-invariant equivalence relations. In this note, we introduce a modification of the Nerode relation and show how it can be used to synthesize machines from finite subsets of their behavior. The technique described is a method for finding a nondeterministic machine that realizes a given finite set of input–output pairs, and it includes a parameter k that allows one to vary the precision and complexity of the synthesized machine. At low settings of k, the synthesized machine tends to have few states, much nondeterminism, and may yield a number of possible outputs for any given input. At higher values of k, the synthesized machine will have more states, but will be deterministic and will be a precise representation of the desired input–output pairs. If the available input–output pairs sufficiently characterize some finite-state computable function f, and if k is appropriately adjusted, the method will find the machine that realizes f.

There are a number of finite-state machine synthesis algorithms in the literature, but most of them [2], [6], [9], [11]–[14], [16] require that the problem be already formulated in terms of some kind of transition table, state diagram, regular expression, sequential relation, or other representation. This note is concerned with the design of a machine from a finite number of examples of its behavior when no such other representation is available.

Techniques have been given for machine synthesis from input–output behavior by Gill [7], Ginsburg [8], [9], Gray and Harrison [11], Tal [17], and others. Each of these methods requires that enough information be included in the problem statement so that the solution is unique, and in contrast to the method presented here, they do not have a capability to utilize unspecified or DON'T CARE conditions to produce simpler solutions. The method described here yields machines that satisfy the known input–output requirements and that often given "reasonable" behavior outside of the well-specified domain. If the amount of available input–output information is increased to the point that the solution is unique, the algorithm finds the correct solution.

Section II defines a nondeterministic machine $M_S^{(k)}$ that computes the input–output pairs in S. The number of states in $M_S^{(k)}$ and the precision of its representation of S will depend on the value of k. Section III shows how $M_S^{(k)}$ can be used to find the finite-state machine to compute function f if S contains enough information about f. Finally, in Section IV, we discuss some applications of the theory.

II. The Machine $M_S^{(k)}$

Let f be a function that assigns to strings in X^* values in Y, and make the definition $f_w(x) = f(wx)$. Let R_f be the Nerode relation on the set X^*: $(x_1, x_2) \in R_f$ if and only if

$f_{x_1}(z)=f_{x_2}(z)$ for all $z\in X^*$. Then the equivalence class that contains x is

$$[x]_{R_f} = \{x' \in X^* \mid (x, x') \in R_f\}.$$

If there are a finite number of equivalence classes, then f will be called a *finite-state function*, and f can be computed by a finite-state machine $M(f)$ with the equivalence classes $[x]_{R_f}$ as states and with next-state function d_{R_f} defined as follows:

$$d_{R_f}([x]_{R_f}, a) = [xa]_{R_f}.$$

$M(f)$ has initial state $[\Lambda]_{R_f}$[1] and upon receiving input sequence x, advances through the states to $[x]_{R_f}$. After so processing the string x, $M(f)$ returns the output $f(x)$, which is associated with its final state $[x]_{R_f}$. This construction is due to Nerode [15] and is the starting point for the development below.

We will assume that f is not necessarily total. There may exist equivalence classes $[x]_{R_f}$ such that for all $x'\in[x]_{R_f}$, $f(x')$ is not defined, and so the behavior of f is never observed on such values. The reader may wish to think of f as being total and yielding an output $0\notin Y$ for such x', but he should remember that no input–output pairs of the form $(x', 0)$ are ever observed in the set S defined below.

Let S be a finite subset[2] of $X^*\times Y$, such that if $(x, y)\in S$, then $f(x)$ is defined and $f(x)=y$. Define $P(S)=\{x \mid (x, y)\in S$ for some $y\}$. $P(S)$ is the projection of S on its first coordinate and will be written as simply P when its meaning is clear from context. Let f_S be defined to be the restriction of f to the pairs represented in S. $f_S(x)$ is undefined if $x\notin P(S)$.

Next we define a modification of the Nerode relation: $(x_1, x_2)\in E_S^{(k)}$ if and only if for all $w\in X^*$ with length $(x)\leq k$: 1) $x_1w\in P(S)$ if and only if $x_2w\in P(S)$; and 2) $f_S(x_1w)=f_S(x_2w)$ when defined. k may be any nonnegative integer and is a parameter that will be discussed below. $E_S^{(k)}$ is an equivalence relation on X^* that induces equivalence classes $[x]_{S^{(k)}}=\{x' \mid (x, x')\in E_S^{(k)}\}$. These equivalence classes will become the states of a nondeterministic machine $M_S^{(k)}$ with next-state function $d_S^{(k)}$ that we now define.

$d_S^{(k)}([x]_{S^{(k)}}, a)=\{[x'a]_{S^{(k)}} \mid x'\in[x]_{S^{(k)}}\}$, $a\in X$. If $w\in X^*$, then $d_S^{(k)}$ is defined recursively:

$$d_S^{(k)}([x]_{S^{(k)}}, \Lambda) = \{[x]_{S^{(k)}}\}$$
$$d_S^{(k)}([x]_{S^{(k)}}, wa) = \bigcup_{[x']_{S^{(k)}}\in d_S^{(k)}([x]_{S^{(k)}}, w)} d([x']_{S^{(k)}}, a).$$

Finally, the nondeterministic output of $M_S^{(k)}$ is

$$h_S^{(k)}(x)=\{f_S(w) \mid [w]_{S^{(k)}}\in d_S^{(k)}([\Lambda]_{S^{(k)}}, x) \text{ and } w\in P(S)\}.$$

The machines $M_S^{(k)}$ for $k=0, 1, 2, \cdots$ are approximate realizations of the function f_S that improve in precision and increase in number of states as k is increased. For low values of k, $M_S^{(k)}$ computes a set of outputs for each input string, and for higher values of k, $M_S^{(k)}$ becomes deterministic, yielding a unique output. If k is greater than or equal to the length of the longest string in P, then $M_S^{(k)}$ is the minimal deterministic machine that computes f_S exactly. These results are made precise in the following theorems.

Theorem 1: If $f_S(x)$ is defined, then $f_S(x)\in h_S^{(k)}(x)$.

Proof: $[x]_{S^{(k)}}\in d_S^{(k)}([\Lambda]_{S^{(k)}}, x)$ and $x\in P$.

Theorem 2: If k is greater than or equal to the length of the longest string in $P(S)$, then $h_S^{(k)}(x)=\{f_S^{(k)}\}$ if $x\in P(S)$ and $h_S^{(k)}(x)=\phi$ otherwise.

Proof: The length restriction on k means that k is effectively infinite for this particular S, and so $E_S^{(k)}$ is the Nerode relation. Therefore the next-state function $d_S^{(k)}$ is deterministic, so $d_S^{(k)}([w]_{S^{(k)}}, a)$ for $w\in X^*$, $a\in X$ is a singleton set if wa is a prefix to some string in P and is the empty set otherwise.

This determinism implies that if $x\in P$, then $h_S^{(k)}(x)$ can have only one value which, by Theorem 1, must be $f_S(x)$: $h_{S(x)}^{(k)}=\{f_S(x)\}$. If $x\notin P$ but is a prefix of some $x'\in P$, then $d_S^{(k)}([\Lambda]_{S^{(k)}}, x)=\{[x]_{S^{(k)}}\}$. But there is no $w\in[x]_{S^{(k)}}$ such that $f_S(w)$ is defined (by definition of $[x]_{S^{(k)}}$) so $h_S^{(k)}(x)=\phi$. If $x\notin P$ and is not a prefix of any $x'\in P$, then $d_S^{(k)}([\Lambda]_{S^{(k)}}, x)=\phi$ and $h_S^{(k)}(x)=\phi$.

Theorem 3: $h_S^{(k+1)}(x)\subseteq h_S^{(k)}(x)$.

Proof: It is first necessary to show that $[w]_{S^{(k+1)}}\in d_S^{(k+1)}([\Lambda]_{S^{(k+1)}}, z)$ implies $[w]_{S^{(k)}}\in d_S^{(k)}([\Lambda]_{S^{(k)}}, z)$, $w, z\in X^*$. This follows from an induction argument if it can be proved that: 1) $[w]_{S^{(k+1)}}\in d_S([x]_{S^{(k+1)}}, a)$ implies 2) $[w]_{S^{(k)}}\in d_S^{(k)}([x]_{S^{(k)}}, a)$ for $x\in X^*$, $a\in X$. This can be shown by noting that 1) implies that there is a $x'a\in[w]_{S^{(k+1)}}$ such that $x'\in[x]_{S^{(k+1)}}$. However, for all strings $v\in X^*$, $[v]_{S^{(k+1)}}\subseteq[v]_{S^{(k)}}$ by definition of $[v]_{S^{(i)}}$ and therefore $x'a\in[w]_{S^{(k)}}$ and $x'\in[x]_{S^{(k)}}$. But this implies 2), concluding the first part of the proof.

Theorem 3 is thus proven by noting that $y\in h_S^{(k+1)}(x)$ implies that there is a $w\in X^*$ such that $f_S(w)=y$ and $[w]_{S^{(k+1)}}\in d_S^{(k+1)}([\Lambda]_{S^{(k+1)}}, x)$. But by the above argument, $[w]_{S^{(k)}}\in d_S^{(k)}([\Lambda]_{S^{(k)}}, x)$, which yields the result $y\in h_S^{(k)}(x)$.

An example is given in Fig. 1 of the machines $M_S^{(0)}$, $M_S^{(1)}$, and $M_S^{(2)}$ for a particular S. $M_S^{(2)}$ is the minimal deterministic machine for f_S.

If the range Y of the function f has only one element in it, the strings of $P(S)$ may be interpreted as samples from some regular set and the constructed machine $M_S^{(k)}$ will be an acceptor of P. Applying the above theorems, the accepted set of each $M_S^{(k+1)}$ will be a subset of the accepted set of $M_S^{(k)}$, and if k is greater than or equal to the length of the longest string in P, then $M_S^{(k)}$ will be the minimal deterministic acceptor of exactly the set P.

III. The Exact Realization of f

The properties of $M_S^{(k)}$ for the finite set S are described above, but there may be little relationship between $M_S^{(k)}$ and the function f from which S was constructed. In this section, restrictions on S will be described that guarantee that $M_S^{(k)}$ will compute f exactly.

A_f will be defined to be any set of strings from X^* such that: 1) every state of $M(f)$ is reached by some string in A_f (i.e., for each state s in $M(f)$ there is a $w\in A_f$ such that $d_{R_f}([\Lambda]_{R_f}, w)=s$); and 2) every prefix of a string in A_f is also in A_f. B_r will be defined to be the set of strings in X^* of length r or less. $M(f)$ will be said to have *n-distinguishable*

[1] Λ is the string of length zero.

[2] If A and B are sets, $A\times B$ is defined to be $\{(a, b) \mid a\in A$ and $b\in B\}$.

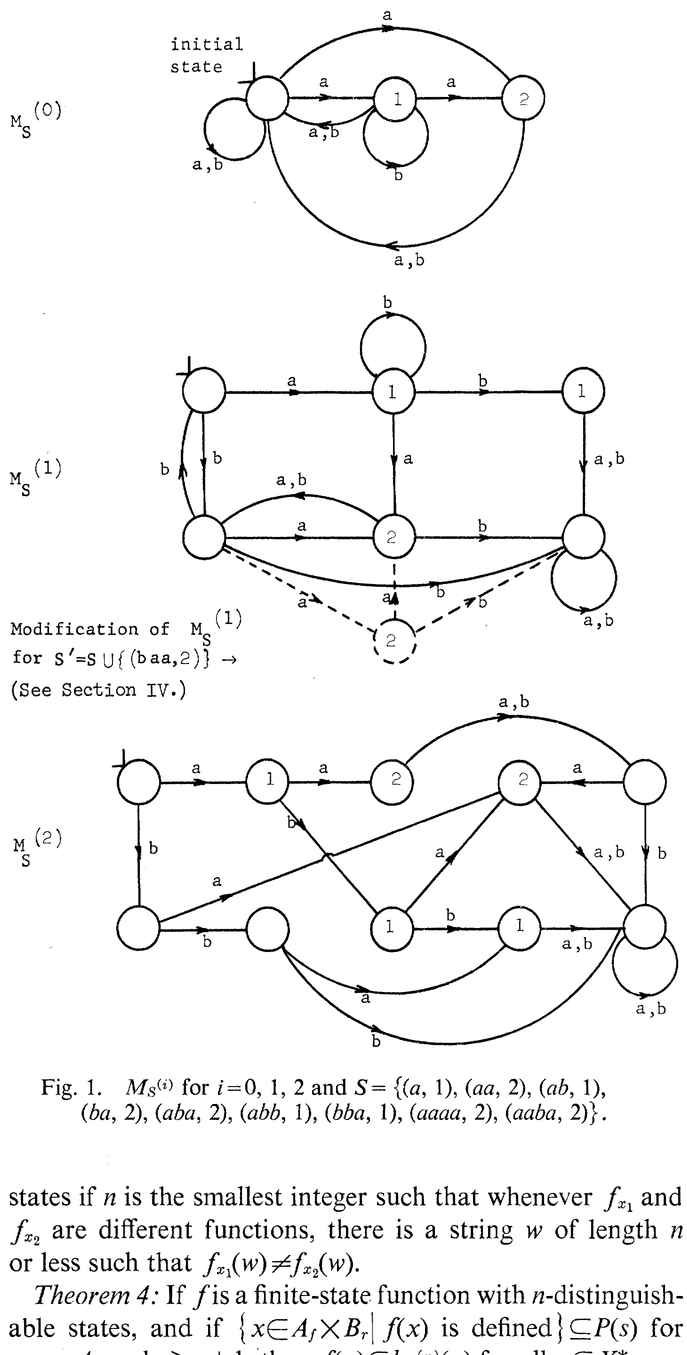

Fig. 1. $M_S^{(i)}$ for $i=0, 1, 2$ and $S=\{(a, 1), (aa, 2), (ab, 1), (ba, 2), (aba, 2), (abb, 1), (bba, 1), (aaaa, 2), (aaba, 2)\}$.

states if n is the smallest integer such that whenever f_{x_1} and f_{x_2} are different functions, there is a string w of length n or less such that $f_{x_1}(w)\neq f_{x_2}(w)$.

Theorem 4: If f is a finite-state function with n-distinguishable states, and if $\{x\in A_f\times B_r \mid f(x) \text{ is defined}\}\subseteq P(s)$ for some A_f and $r\geq n+1$, then $f(x)\in h_{S^{(n)}}(x)$ for all $x\in X^*$.

Proof: Suppose $x=a_1a_2\cdots a_t$, $a_i\in X$. We will show by induction that for each $i=0, 1, 2, \cdots, t$ there is a state $[w_i]_{S^{(n)}}\in d_{S^{(n)}}([\Lambda]_{S^{(n)}}, a_1a_2\cdots a_i)$ such that $w_i\in[a_1a_2\cdots a_i]_{R_f}$.[3] Then there is a state $[w_t]_{S^{(n)}}\in d_{S^{(n)}}([\Lambda]_{S^n}, a_1a_2\cdots a_t)$ such that $w_t\in[x]_{R_f}$ which implies that $f(x)=f(w_t)\in h_{S^{(n)}}(x)$.

For the basis of the induction proof at $i=0$, $[\Lambda]_{S^{(n)}}\in d_{S^{(n)}}([\Lambda]_{S^{(n)}}, \Lambda)$ and $\Lambda\in[\Lambda]_{R_f}$. Assume the induction hypothesis that $[w_i]_{S^{(n)}}\in d_{S^{(n)}}([\Lambda]_{S^{(n)}}, a_1a_2\cdots a_i)$ and $w_i\in[a_1a_2\cdots a_i]_{R_f}$. Then there is a string x' in A_f such that $d_{R_f}([\Lambda]_{R_f}, x')=[a_1a_2\cdots a_i]_{R_f}$ by definition of A_f. So $d_{S^{(n)}}([\Lambda]_{S^{(n)}}, a_1a_2\cdots a_{i+1})$ contains $[x'a_{i+1}]_{S^{(n)}}$. But $x'a_{i+1}\in[a_1a_2\cdots a_{i+1}]_{R_f}$ because $f(x'a_{i+1}u)=f(a_1a_2a_3\cdots a_{i+1}u)$ for all $u\in X^*$ of length n or less (since $\{x\in A_f\times B_r \mid f(x) \text{ is defined}\}\subseteq P(S)$) and because the states of f are n-distinguishable. This completes the induction step and the proof of Theorem 4.

The next result requires a constant m_f that is associated with the function f. Assuming that $M(f)$ has n-distinguishable states, m_f will be defined to be the smallest integer m such that $m\geq n$, and not all of the following conditions occur for any strings w_1 and w_2.

Condition 1: For each function f_{w_i}, $i=1, 2$, there is a string x such that $f_{w_i}(x)$ is defined and the other function f_{w_h}, $h=1, 2$, $h\neq i$ is either undefined or $f_{w_1}(x)\neq f_{w_2}(x)$.

Condition 2: There is an integer $p<n$ such that for each x of length p or less, either both $f_{w_1}(x)$ and $f_{w_2}(x)$ are undefined or $f_{w_1}(x)=f_{w_2}(x)$.

Condition 3: Either f_{w_1} or f_{w_2} is undefined for all strings of length j, $p<j\leq m$.

The proof of Theorem 5 will fail if Conditions 1, 2, and 3 occur for any strings w_1 and w_2. However, there is always a value of m_f that will prevent this, as shown by the next lemma. If f is a total function, then $m_f=n$.

Lemma 1: $m_f\leq 2n+1$.

Proof: We will show that if $m=2n+1$ then Conditions 1, 2, and 3 cannot all be satisfied, so $m_f\leq 2n+1$. Specifically, if Conditions 2 and 3 are true, then Condition 1 cannot be satisfied. Let w_1 and w_2 be two strings that satisfy Condition 2 and such that f_{w_1} is undefined for all strings of length j, $p<j\leq m$. Let u be any string of length n. Then $f_{w_1u}(v)$ is undefined for all strings v of length $n+1$ or less by the above assumption about w_1. Then $f_{w_1u}(v)$ is undefined for all strings v of any length. Then there is no string x such that $f_{w_1}(x)$ is defined and such that $f_{w_2}(x)$ is either undefined or $f_{w_1}(x)\neq f_{w_2}(x)$. But this contradicts Condition 1 and so completes the proof.

Theorem 5: If f is a finite-state function and if $r\geq m_f+1$, then for all S such that $P(S)=\{x\in A_f\times B_r \mid f(x) \text{ is defined}\}$ and for all $x\in X^*$

$$h_{S^{(m_f)}}(x) = \begin{cases}\{f(x)\}, & \text{if } f(x) \text{ is defined}\\ \phi, & \text{otherwise.}\end{cases}$$

Proof: By the previous theorem, $f(x)\in h_{S^{(m_f)}}(x)$ so it is only necessary to show that $h_{S^{(m_f)}}(x)$ contains no elements y that are not equal to $f(x)$. Referring to the proof of Theorem 4, if there is a $y\in h_{S^{(m_f)}}(x)$ where $y\neq f(x)$, then there is a smallest i, $0\leq i\leq t-1$, such that $[x_i]_{S^{(m_f)}}\in d_{S^{(m_f)}}([\Lambda]_{S^{(m_f)}}, a_1a_2\cdots a_i)$ implies $x_i\in[a_1a_2\cdots a_i]_{R_f}$ and an x' such that $[x']_{S^{(m_f)}}\in d_{S^{(m_f)}}([\Lambda]_{S^{(m_f)}}, a_1a_2\cdots a_{i+1})$ where $x'\notin[a_1a_2\cdots a_{i+1}]_{R_f}$. But consider all of the states in $d_{S^{(m_f)}}([x_i]_{S^{(m_f)}}, a_{i+1})=\{[ua_{i+1}]_{S^{(m_f)}} \mid u\in[x_i]_{S^{(m_f)}}\}$.

Case 1: Examine the state $[ua_{i+1}]_{S^{(m_f)}}$ assuming $u\in A_f$. Then $ua_{i+1}\in[a_1a_2\cdots a_{i+1}]_{R_f}$ since $ua_{i+1}v\in P(S)$ for all v of length m_f or less (such that $f(ua_{i+1}v)$ is defined). So this case does not result in an $x'=ua_{i+1}\notin[a_1a_2\cdots a_{i+1}]_{R_f}$.

Case 2: Consider the state $[ua_{i+1}]_{S^{(m_f)}}$ assuming $u=u_1u_2$ where $u_1\in A_f$ and $u_2\in B_r$. Then $ua_{i+1}v\in P(S)$ for all v of length p or less (such that $f(ua_{i+1}v)$ is defined) and $p<m_f$. Furthermore, $f_S(ua_{i+1}v)$ is undefined for all strings v of

[3] The string $a_1a_2\cdots a_i$ with $i=0$ will be defined to be the null string.

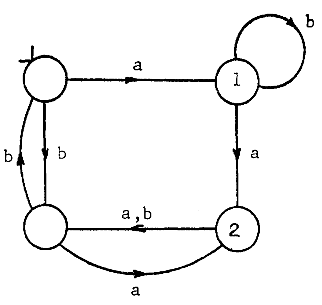

Fig. 2. The machine that represents the function f of the example.

length j, $p<j\leq m_f$. But by definition of m_f, not all of Conditions 1, 2, and 3 can hold for ua_{i+1} and any other string x'.

If Condition 1 occurs, then Conditions 2 and 3 cannot both occur. So there is no string x' such that for all strings of length p or less, $f_{x'}(v)$ and $f_{ua_i}(v)$ are either both undefined or $f_{x'}(v)=f_{ua_i}(v)$ and such that $f_{x'}$ is undefined for all strings of length j, $p<j\leq m_f$. So we conclude that there is no x' such that $[x']_{S^{(m_f)}}\in d_{S^{(m_f)}}([\Lambda]_{S^{(m_f)}}, a_1a_2\cdots a_{i+1})$ where $x'\notin[a_1a_2\cdots a_{i+1}]$.

Condition 1 does not hold for ua_i and some x', then $f_{x'}(v)$ will always either equal $f_{ua_i}(v)$ or be undefined. Again, no nondeterministic output behavior occurs.

Therefore the conditions for having a $y\in h_{S^{(m_f)}}(x)$ where $y\neq f(x)$ cannot occur and the theorem is proved.

Theorem 5 is illustrated by the example in Fig. 2. In this case, $m_f=n=1$ and A_f may be chosen as the set $\{\Lambda, a, aa, b\}$. If r is set equal to 2, then the set S as given in Fig. 1 results and $h_{S^{(1)}}$ is realized by the machine $M_{S^{(1)}}$ as shown.

Corollary 1: Let f be a total finite-state function such that $M(f)$ has n-distinguishable states and every state is reachable from $(\Lambda)_{R_f}$ by a string of length j or less. If $P(S)=\{x\in X^* \mid \text{length } (x)\leq n+j+1\}$, then for all $x\in X^*$, $h_{S^{(n)}}(x)=\{f(x)\}$.

Proof: Since f is a total function, $m_f=n$. Let $A_f=\{x\in X^* \mid \text{length } (x)\leq j\}$ in Theorem 5.

Corollary 2: Let f be a total finite-state function such that $M(f)$ has q states. If $P(S)=\{x\in X^* \mid \text{length } (x)\leq 2q-2\}$ then for all $x\in X^*$, $h_{S^{(q-2)}}(x)=\{f(x)\}$.

Proof: $M(f)$ has q states implies that the states are $(q-2)$-distinguishable and $(q-1)$-reachable.

Corollary 3: Let f be a finite-state function such that $M(f)$ has q states. If $P(S)=\{x\in X^* \mid \text{length } (x)\leq 3q-3 \text{ and } f(x) \text{ is defined}\}$, then for all $x\in X^*$

$$h_{S^{(m_f)}}(x) = \begin{cases}\{f(x)\}, & \text{if } f(x) \text{ is defined}\\ \phi, & \text{otherwise.}\end{cases}$$

Proof: This proof is similar to that of the previous corollary except that Lemma 1 is employed to obtain a bound on m_f since f is not total.

IV. Applications

The above results give a solution to the machine synthesis problem in several situations. If we desire a finite-state automaton that accepts finite set P, then we let $Y=\{1\}$, find $M_{S^{(k)}}$, and adjust k high enough to obtain the desired accuracy. Theorem 5 indicates the number of strings required to construct the acceptor of any regular set.

If we have a finite set of strings that are to be accepted by some machine and a finite set that are to be rejected, we let $Y=\{1, 2\}$ where a 1 or 2 indicates that a string is rejected or accepted, respectively. $M_{S^{(k)}}$ is then constructed with k set high enough so that no nondeterminism (where a string is both rejected and accepted) occurs on strings that are important in the application.

In the more general situation where Y has more than two symbols, the applications are clear. In all cases, the nondeterministic machines created by these methods can be converted to minimal deterministic machines by increasing k to a high enough value and employing standard minimization techniques [6], [8], [9], [12].

The most important use of these methods occurs in the sequential learning situation. Suppose one is given a new input–output pair (x_i, y_i) such that $f(x_i)=y_i$ at each time t_i and is asked to select a machine A_i that describes the sequence up to time t_i. This is the analog of the grammatical inference problem [1], [3]–[5] that was our original motivation and is a model of scientific reasoning and other hypothesis-forming behavior. The interesting questions in sequential learning are the nature of the machines A_i and the limiting behavior of an algorithm as i approaches infinity. We have developed elsewhere [4], [5] a number of general results on this subject. These show[4] that there is an algorithm that will choose the best A_i at each i and will be such that the successive A_i become ever better approximations to the machine $M(f)$.

There are two important advantages of the methods of this note over the general algorithms described in [4], [5]. The latter methods depend on enumerating all finite-state machines in order, while the construction of Section II requires one to consider only a small number of machines. In addition, for fixed k, the machine $M_{S_i^{(k)}}$ is easily constructed from the machine $M_{S_{i-1}^{(k)}}$. This notion of sequential modification of a synthesized machine is extremely important and will be briefly described.

Suppose that $M_{S_{i-1}^{(k)}}$ has been constructed and that $M_{S_i^{(k)}}$ must now be found; $S_i=S_{i-1}\cup\{(x_i,y_i)\}$. Let $x_i=a_1a_2\cdots a_r$, $a_j\in X$. Then only the $k+1$ (or fewer) states $[a_1a_2\cdots a_{r-k}]_{S^{(k)}}$, $[a_1a_2\cdots a_{r-k+1}]_{S^{(k)}}, \cdots [a_1a_2\cdots a_r]_{S^{(k)}}$ in $M_{S_{i-1}^{(k)}}$ are affected by the addition of the pair (x_i, y_i). Each of the equivalence classes $[a_1a_2\cdots a_{r-k+j}]_{S^{(k)}}$ must be altered so that output y_i is associated with the state $d_{S^{(k)}}([a_1a_2\cdots a_{r-k+j}]_{S^{(k)}}, a_{r-k+j+1}a_{r-k+j+2}\cdots a_r)$. These alterations in $M_{S_{i-1}^{(k)}}$ are enough to produce the new machine $M_{S_i^{(k)}}$. In the example of Section II, if the pair $(baa, 2)$ were added to S, the new machine $M_{S'^{(1)}}$ would have one changed state $[ba]_{S'^{(1)}}$, and three new transitions, as indicated in Fig. 1.

We have shown that the construction of Section II will produce machines $M_{S_i^{(k)}}$ that have desirable properties. It remains to describe an algorithm for choosing k and deciding

[4] The actual results concern grammar discovery rather than machine discovery. However, the extension to the sequential machines of this note is straightforward.

which $M_{S_i}{}^{(k)}$ to call the machine A_i. Suppose one is given an upper bound q on the number of states in $M(f)$. Let $S|_j = \{(x, y) \in S | \text{length } (x) \leq j\}$ and consider the following algorithm.

Algorithm 1: Let $k = 2q-3$. At each i, compute

$$B_i = M^{(k)}_{S_i|_{3q-3}}.$$

If

$$y_j \in h^{(k)}_{S_i|_{3q-3}}(x_j)$$

for all $(x_j, y_j) \in S_i$, then $A_i = B_i$; otherwise $A_i = M_{S_i}{}^{(k)}$.

Let f_{A_i} be the nondeterministic function that is computed by A_i.

Theorem 6: Algorithm 1 has the following properties.

Property 1: $y_j \in f_{A_i}(x_j)$, for all $(x_j, y_j) \in S_i$.

Property 2: If $\{w \in A_f \times B_{k+1} | f(w) \text{ is defined}\} \subseteq P(S_i)$, then $f(x) \in f_{A_i}(x)$, for all $x \in X^*$.

Property 3: If $\{w \in X^* | \text{length } (w) \leq 3q-3 \text{ and } f(w) \text{ is defined}\} \subseteq P(S_j)$ for some j then for all $i \geq j$, $f_{A_i}(x) = \{f(x)\}$ for all $x \in X^*$ such that $f(x)$ is defined and $f_{A_i}(x) = \Phi$ otherwise.

Proof: Because $M(f)$ has q or fewer states, those states are $q-2$ or less distinguishable. Using Lemma 1, an upper bound on the value of k required is $k = 2n+1 = 2(q-2)+1 = 2q-3$. Then Property 1 follows directly from Theorem 1, Property 2 from Theorem 4, and Property 3 from Corollary 3.

Algorithm 1 yields a nondeterministic machine with nondeterministic outputs. If determinism is desired in the output behavior, one follows Algorithm 1 as before. But when a nondeterminism occurs in any $M_S{}^{(k)}$, one increases k until the nondeterminism disappears and then continues the algorithm. Theorem 6 will still hold.

If one assumes, as seems reasonable, that every pair (x, y) such that $f(x) = y$ is defined will occur at some time t_i, Algorithm 1 will eventually choose only a machine that represents f exactly. This is known in the literature as the algorithm *identifying* f. There is a problem in that Algorithm 1 depended on an *a priori* estimate of the number of states in $M(f)$. It is shown in [4], [5] that without this estimate, no algorithm will be able to identify every finite-state function from an arbitrary presentation of its behavior. If, however, f is known to be a total function, then no *a priori* information is required, and the following algorithm will identify f after a finite time.

Algorithm 2: Let

$$A_i = M^{(k)}_{S_i|_{2k+2}}$$

where

$$k = \min \{j | (x, y) \in S_i \text{ implies } h^{(j)}_{S_i|_{2j+2}}(x) = \{y\}\}.$$

Theorem 7: Algorithm 2 has the following properties.

Property 1: $f_{A_i}(x_j) = \{y_j\}$, for all $(x_j, y_j) \in S_i$.

Property 2: If $M(f)$ has q states and if $\{x \in X^* | \text{length } (x) \leq 2q-2\} \subseteq P(S_j)$ for some j then for all $i \geq j$, $f_{A_i}(x) = \{f(x)\}$, for all $x \in X^*$.

Proof: Property 1 is true by construction and Theorem 2 guarantees that the construction is possible.

If $M(f)$ has q states and all strings of length $2q-2$ or less are in $P(S_j)$, then the algorithm will choose $k = q-2$ (or less). Then by Corollary 2, Property 2 follows.

V. Discussion and Summary

This note gives a technique for constructing finite-state machines from a finite number of examples of their behavior. The method has been programmed on a computer and extensively tested. Typical constructions of machines with 10 or 20 states require a few seconds or less of CPU time to complete. Many other versions of this method are possible, and some were investigated, although they are not described here.

The construction procedure has a simple operation, and is therefore easy to program and fast in execution. The parameter k enables the user to obtain as exact a fit to the needed behavior as he desires, at the cost of increasing the complexity of the resulting machine. The simplicity of the procedure makes its operation easy to understand and easy to characterize. Finally, the system has the distinct advantage that if a large amount of computational effort is invested in finding a machine for a set of pairs (x, y), changes can be made in its behavior without the necessity of starting the design procedure over again. If $M_S{}^{(k)}$ is created to realize f_S and then S is changed slightly, only the states and transitions in $M_S{}^{(k)}$ that correspond to the changes in S need to be adjusted to obtain a new machine.

Acknowledgment

The authors are indebted to the referees who made many helpful suggestions. In particular, it was recommended that the original model be reframed in terms of the Nerode-like relations $E_S{}^{(k)}$ in order to simplify the proofs and generalize the results.

References

[1] A. W. Biermann and J. A. Feldman, "A survey of results in grammatical inference," presented at the Int. Conf. Frontiers of Pattern Recognition, Univ. Hawaii, Honolulu, Jan 18–20, 1971.

[2] J. A. Brzozowski, "Derivatives of regular expressions," *J. Ass. Comput. Mach.*, vol. 11, no. 4, pp. 481–494, 1964.

[3] J. A. Feldman, "First thoughts on grammatical inference," Dep. Comput. Sci., Stanford Univ., Stanford, Calif., A.I. Memo 55, Aug. 1967

[4] ——, "Some decidability results on grammatical inference and complexity," Dep. Comput. Sci., Stanford Univ., Stanford, Calif., A.I. Memo 93.1, May 1970.

[5] J. A. Feldman, J. Gips, J. J. Horning, and S. Reder, "Grammatical complexity and inference," Dep. Comput. Sci., Stanford Univ., Stanford, Calif., Tech. Rep. CS125, June 1969.

[6] A. Gill, *Introduction to the Theory of Finite-State Machines.* New York: McGraw-Hill, 1962.

[7] ——, "Realization of input-output relations by sequential machines," *J. Ass. Comput. Mach.*, vol. 13, no. 1, pp. 33–42, 1966.

[8] S. Ginsburg, "Synthesis of minimal-state machines," *IRE Trans. Electron. Comput.*, vol. EC-8, pp. 441-449, Dec. 1959.

[9] ——, *An Introduction to Mathematical Machine Theory.* Reading, Mass.: 1962.

[10] ——, *The Mathematical Theory of Context-Free Languages.* New York: McGraw-Hill, 1966.
[11] J. N. Gray and M. A. Harrison, "The theory of sequential relations," *Inform. Contr.*, vol. 9, pp. 435–468, 1966.
[12] M. A. Harrison, *Introduction to Switching and Automata Theory.* New York: McGraw-Hill, 1965.
[13] D. A. Huffman, "The synthesis of sequential switching circuits," *J. Franklin Inst.*, vol. 257, no. 3, pp. 161–190, 1954; no. 4, pp. 275–303, 1954.
[14] G. H. Mealy, "A method for synthesizing sequential circuits," *Bell Syst. Tech. J.*, vol. 34, no. 5, pp. 1045–1079, 1955.
[15] A. Nerode, "Linear automaton transformations," *Proc. Amer. Math. Soc.*, vol. 9, pp. 541–544, 1958.
[16] G. Ott and N. Feinstein, "Design of sequential machines from their regular expressions," *J. Ass. Comput. Mach.*, vol. 8, no. 4, pp. 585–600, 1961.
[17] A. A. Tal, "Questionnaire language and the abstract synthesis of minimal sequential machines," *Avtomat. Telemekh.*, vol. 25, no. 6, pp. 946–962, 1964.

An Improved Bound on the Length of Checking Experiments for Sequential Machines with Counter Cycles

C. E. HOLBOROW, STUDENT MEMBER, IEEE

Abstract—The bound on the length of checking experiments derived by Murakami *et al.* [1] is improved by using a more efficient output specification in the counter cycle.

Index Terms—Checking sequence, fault detection, fault diagnosis, sequential machine with fault-detection capabilities.

In their paper, Murakami *et al.* [1] show that Hennie-type checking experiments are simple to design for a machine with a counter cycle. They define the counter cycle for an n-state sequential machine by

$$\sigma(s_i, I) = s_{i+1} \quad \text{and} \quad \lambda(s_i, I) = 0, \qquad (1 \leq i \leq n-1)$$
$$\sigma(s_n, I) = s_1 \quad \text{and} \quad \lambda(s_n, I) = 1$$

where $\{s_i\}$ are the states of the machine, I is a particular input vector, and σ and λ are, respectively, the next-state and output mappings. For machines without a counter cycle they suggest defining a new input symbol ϵ and specifying the next states and outputs under this input to form a counter cycle.

If M is an n-state m-input machine with a counter cycle, then the length of the checking experiment need never exceed $2n(1+mn)$ [1]. This bound can be almost halved if the output of the counter cycle is specified differently, as will now be shown. Let

$$p = [\log_2 n]$$

where the square brackets denote "smallest integer greater than or equal to the number inside the brackets." Any distinguishing sequence for M must have length at least p since we require at least p output symbols to uniquely identify the state of M prior to the application of the distinguishing sequence. If we can find a binary sequence of period n such that each of the n subsequences of length p is distinct, then we can use this sequence to obtain a length p distinguishing sequence for M. The outputs of M under input ϵ (the input vector which causes M to cycle through all its states) are assigned so that M produces the assumed sequence of period n when it is in the counter cycle. Then, if M starts in state s_i, the p outputs it produces in response to p inputs of ϵ will uniquely identify s_i.

The n-bit sequence required is the sequence of period n generated by a p-stage shift register. A p-stage shift register can generate sequences of period L for all possible $L \leq 2^p$ (see Golomb [2] for proof), so a p-stage shift register is the shortest shift register that can generate a sequence with period n. One of the characteristics of this sequence is that it contains n distinct subsequences of length p.

A new upper bound on the length of the checking sequence can now be derived. The checking sequence consists of: 1) a subsequence of length $n+p$ to verify that the counter cycle is functioning correctly; and 2) at most mn subsequences to check the transitions of the machine.

Each transition is checked by a subsequence consisting of: 1) a transfer sequence of length at most $n-1$ to move the machine to the desired state; 2) a single input to perform the transition under test; and 3) a distinguishing sequence of length p to verify that the transition ended in the correct state.

The total length of the checking sequence is

$$n + p + mn(n + p) = (n + [\log_2 n])(1 + mn).$$

For large n this is little more than half the previous bound.

In most cases it is possible to design a checking sequence much shorter than the upper bound since the number of transfer sequences required will be much less than mn, and their average length will be less than $n-1$. (Because the transitions do not have to be checked in any particular order, a transfer sequence is required only if the machine finishes a distinguishing sequence in a state for which all outward transitions have been checked.) This is particularly true for machines where the number of transitions into each state is approximately the same, since the machine does not end all distinguishing sequences in the same state but is in state s_{i+p} if, prior to the application of the distinguishing sequence, it was in state s_i. Over the length of the checking sequence the machine will end the distinguishing sequences in each state approximately the same number of times, so few transfer sequences will be required.

As long as the machine cycles through all its states under input ϵ, the order in which the states are encountered is unimportant. This fact may allow some savings to be made in the construction of the next-state logic. Another use for this degree of freedom is possible for the output assignment suggested by Murakami *et al.* ($n-1$ zeros and one 1). The states can be ordered according to the number of transitions into them (i.e., the number of times each state appears in the transition table). Then the output of the most frequently occurring state is made 1 and all the other states have output 0. The next-state transitions under input ϵ are specified so

Manuscript received August 1, 1971; revised January 31, 1972.
The author is with the Department of Electrical Engineering, The Johns Hopkins University, Baltimore, Md. 21218.