Signature Table Systems and Learning

ALAN W. BIERMANN, MEMBER, IEEE, JOHN R. C. FAIRFIELD, MEMBER, IEEE, AND THOMAS R. BERES

Abstract—**A characterization theorem is given for the classes of functions which are representable by signature table systems. The usefulness of the theorem is demonstrated in the analysis and synthesis of such systems. The limitations on the power of these systems come from the restrictions on the table alphabet sizes, and a technique is given for evaluating these limitations. A practical learning system is proposed and analyzed in terms of the theoretical model of this paper. Then an improved method is described and results are presented from a series of experiments.**

I. Introduction

Any system which is capable of learning must include an internal decisionmaking mechanism that can be automatically modified to improve behavior. A very popular such mechanism employed in many early learning systems was the linear evaluation polynomial (Nilsson [1], Samuel [2]) which was used to compute an evaluation of alternative decisions so that the most desirable one could be chosen. Learning was achieved by modifying the values of the polynomial coefficients but the linearity assumption proved to be unduly restrictive and convergence to optimum behavior was rare (Minsky and Papert [3]). In an attempt to break away from the linearity assumption, Griffith [4], [5], and Samuel [6] developed signature table systems as the central evaluation mechanism for checker playing programs and were consequently able to obtain significant improvements in performance. Since that time, signature tables have been used effectively in other systems (Truscott [7], Smith [8], Page [9], Mamrak and Amer [14]) and promise to be an important tool in the future. Despite the successes, however, the basic properties of signature tables are still little understood and few conceptual or mathematical tools exist for dealing with them.

This paper will attempt to clarify at least some of the properties of this mechanism by giving a characterization theorem for the classes of functions representable by signature tables and showing its usefulness in the analysis and synthesis of such systems. The limitations on the power of these systems come from the restrictions on the table alphabet sizes and a technique is given for evaluating these limitations. Finally, a practical learning system will be proposed and analyzed and a method for improving it will be given in the light of current results.

Manuscript received December 13, 1979; revised March 4, 1981 and March 6, 1982. This work was supported in part by the National Science Foundation under Grants DCR74-14445 and MCS74-14445. Part of the work was also sponsored by the Air Force Office of Scientific Research, Air Force Systems Command, USAF, under Grant 81-0221.

A. Biermann is with the Department of Computer Science, Duke University, Durham, NC 27706.

J. R. C. Fairfield is with the Department of Mathematics and Computer Science, James Madison University, Harrisonburg, VA 22804

T. R. Beres is with RLG Associates, Reston, VA 22090.

II. Signature Table Systems and Their Representable Functions

A *signature table* has a finite nonzero number of inputs each with a finite nonzero-sized alphabet and is a list of the outputs associated with all possible input combinations. A *signature table system* is a tree of signature tables where the outputs from bottom-level tables are inputs to higher level tables until at a highest level there is a single table that gives the output for the total system. Fig. 1 shows an example of a signature table system with six binary inputs and five binary valued tables. Evaluation of a six digit input given the values of $x_1, x_2, \cdots, x_6$, respectively, follows through the tables to the top level where the output is given. The set of all possible inputs and their associated outputs as computed by this system appears in Fig. 2.

The top table in a signature table system will be designated t_1 and the tables immediately below the t_w table will be labeled from left to right t_{w1}, t_{w2}, and so forth. The size of the output alphabet for table t_w will be designated n_w. If a table t_w in a table system has system inputs x_i, then its *base set of inputs* S_w will include each such x_i. If t_w has inputs from other tables $t_{w1}, t_{w2}, \cdots$, its base set of inputs will also include $S_{w1}, S_{w2}, \cdots$. S_w will include no other elements. Using Fig. 1 to illustrate notations, $n_w = 2$ for all w, $S_{111} = \{x_1, x_2\}$, $S_{112} = \{x_3, x_4\}$, $S_{12} = \{x_5, x_6\}$, $S_{11} = \{x_1, x_2, x_3, x_4\}$, and $S_1 = \{x_1, x_2, x_3, x_4, x_5, x_6\}$.

A *signature table schema* is a signature table system with the outputs of all the tables uninstantiated (left blank). In a typical learning system, a signature table schema is built for system decisionmaking, and behavior modification and improvement comes from proper instantiation of the table outputs. The sizes of the input alphabets for all of the tables are usually strictly limited because each table size is equal to the product of its input alphabet sizes and table size can easily exceed practical limitations.

Suppose a particular signature table system computes the function f and has table alphabet sizes n_w for $w = 1, 11, 12, \cdots$. The system will be called *reducible* if there is a modification of its table outputs with alphabet sizes n'_w, $w = 1, 11, 12, \cdots$, such that $n'_w \leqslant n_w$, for all w, $n'_w < n_w$ for some w, and the new system still computes f. An example of a reducible system appears in Fig. 3(a) where one can see that the portion of the output table associated with a first input of 0 is identical to the portion associated

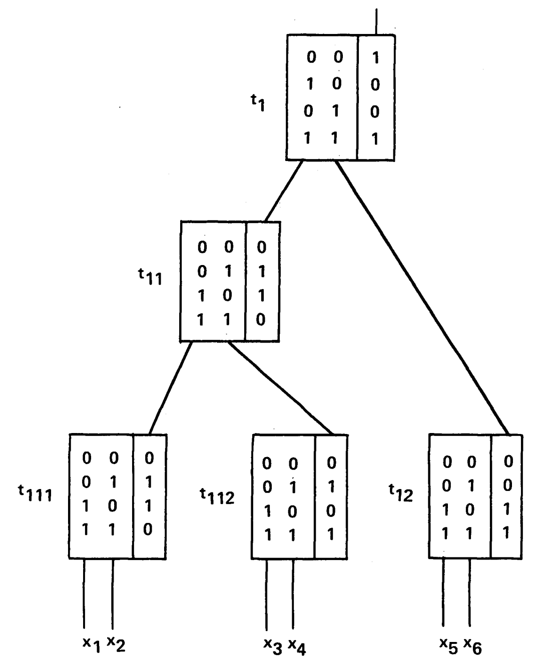


Fig. 1. Example signature table system.

Input	Output	Input	Output
0 0 0 0 0 0	1	1 0 0 0 0 0	0
0 0 0 0 0 1	1	1 0 0 0 0 1	0
0 0 0 0 1 0	0	1 0 0 0 1 0	1
0 0 0 0 1 1	0	1 0 0 0 1 1	1
0 0 0 1 0 0	0	1 0 0 1 0 0	1
0 0 0 1 0 1	0	1 0 0 1 0 1	1
0 0 0 1 1 0	1	1 0 0 1 1 0	0
0 0 0 1 1 1	1	1 0 0 1 1 1	0
0 0 1 0 0 0	1	1 0 1 0 0 0	0
0 0 1 0 0 1	1	1 0 1 0 0 1	0
0 0 1 0 1 0	0	1 0 1 0 1 0	1
0 0 1 0 1 1	0	1 0 1 0 1 1	1
0 0 1 1 0 0	0	1 0 1 1 0 0	1
0 0 1 1 0 1	0	1 0 1 1 0 1	1
0 0 1 1 1 0	1	1 0 1 1 1 0	0
0 0 1 1 1 1	1	1 0 1 1 1 1	0
0 1 0 0 0 0	0	1 1 0 0 0 0	1
0 1 0 0 0 1	0	1 1 0 0 0 1	1
0 1 0 0 1 0	1	1 1 0 0 1 0	0
0 1 0 0 1 1	1	1 1 0 0 1 1	0
0 1 0 1 0 0	1	1 1 0 1 0 0	0
0 1 0 1 0 1	1	1 1 0 1 0 1	0
0 1 0 1 1 0	0	1 1 0 1 1 0	1
0 1 0 1 1 1	0	1 1 0 1 1 1	1
0 1 1 0 0 0	0	1 1 1 0 0 0	1
0 1 1 0 0 1	0	1 1 1 0 0 1	1
0 1 1 0 1 0	1	1 1 1 0 1 0	0
0 1 1 0 1 1	1	1 1 1 0 1 1	0
0 1 1 1 0 0	1	1 1 1 1 0 0	0
0 1 1 1 0 1	1	1 1 1 1 0 1	0
0 1 1 1 1 0	0	1 1 1 1 1 0	1
0 1 1 1 1 1	0	1 1 1 1 1 1	1

Fig. 2. Function computed by signature table system of Fig. 1.

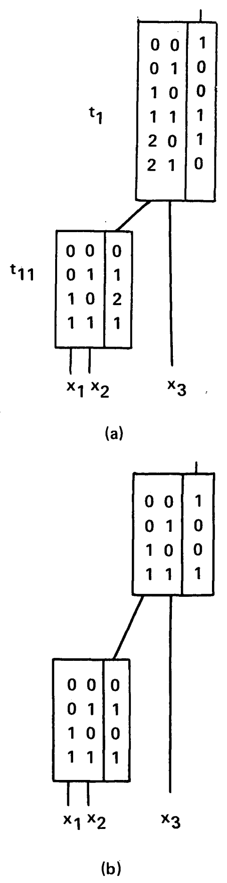


Fig. 3. (a) Reducible signature table system. (b) Its reduced form.

with a first input of 2. So the 0 and 2 in the output of the bottom table are merged yielding the reduced form in Fig. 3(b).

Suppose a particular function f has finite valued arguments $A = \{x_1, x_2, \cdots, x_n\}$ and that $S = \{x_{i_1}, x_{i_2}, \cdots, x_{i_j}\}$ is a subset of these arguments. The *characterization matrix* $M(f, S)$ for f in terms of S is an array with a column corresponding to each instantiation of the variables in $A - S$ and a row for each possible instantiation of the variables in S. The entry in the array corresponding to the $x_{i_1}, x_{i_2}, \cdots, x_{i_j}$th row and the $x_{k_1}, x_{k_2}, \cdots, x_{k_{n-j}}$th column is $f(x_1, x_2, \cdots, x_n)$ where each x_h is instantiated by the row if x_h is in S or the column if x_h is in $(A - S)$.

As an illustration, if f is the function of Fig. 2 and $S = \{x_1, x_2\}$, then $M(f, S)$ is as shown in Fig. 4. One can find the value of $f(x_1, x_2, \cdots, x_6)$ for any input by looking at the intersection of the appropriate row and column. For example, $f(0,0,0,0,0,1)$ is found to have value 1 in the second entry of the first row. The construction of Fig. 4 in this case is easy to see because the first quarter in the table of Fig. 2 is the first row in Fig. 4, the second quarter is the second row, and so forth. If S had been chosen differently, the construction would be just as simple although not so easy to visualize unless one reordered the table of Fig. 2 such that the elements of S were the high order (leftmost) bits in the enumeration of the input space of the function. We will be interested in the number of different patterns in the rows which appear in the matrix $M(f, S)$ and will

x_3	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
x_4	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
x_5	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
x_6	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
x_1x_2																
0 0	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1
0 1	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0
1 0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0
1 1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1

Fig. 4. $M(f, S)$ where $S = \{x_1, x_2\}$.

denote this *row multiplicity* by Rowmult $(M(f, S))$. For example, the row multiplicity of $M(f, S)$ in Fig. 4 is two. This leads to a characterization theorem for the set of functions computable by a signature table schema.

Theorem 1: If T is a signature table schema, then there is an irreducible instantiation of T with alphabet sizes $n_1, n_{11}, n_{12}, \cdots$, which computes f if and only if Rowmult $(M(f, S_w)) = n_w$ for each $w = 1, 11, 12 \cdots$.

A proof of Theorem 1 appears in the Appendix. This theorem is similar to some results found in the literature on switching circuit decomposition (Curtis [10], Karp [11], Ashenhurst [12]). We will be concerned here with its applicability in the analysis and synthesis of signature table systems.

Theorem 1 can be better understood if its meaning is examined in relation to an example function f and its associated signature table system. Assume for the moment that f is given by Fig. 2 and the form of the desired table system T is given by Fig. 1 but none of its table outputs are known. However, it is known that all of these tables must have an output alphabet of size two. Then the matrix $M(f, S_w)$, is constructed for each of the tables t_w in T and the row multiplicity is checked. For example, in Fig. 4, $M(f, S_{111})$, is constructed corresponding to table t_{111} and the row multiplicity is found to be two. Similarly, $M(f, S_w)$ is constructed for $t_w = t_1, t_{11}, t_{112}$, and t_{12} and in each case row multiplicity is examined. Theorem 1 says that there is an irreducible instantiation of T which computes f if and only if all of those row multiplicities are two.

The first thing to be noted about Theorem 1 is that if there are no restrictions on the alphabet sizes then T can compute any finite function and its only computational limitations come from these size restrictions. However, if the output alphabet of any particular table t_w is limited to size n_w or less, then a certain kind of repetitiveness will appear in the behavior of the represented function. Specifically, suppose that $S_w = \{x_1, x_1, \cdots, x_j\}$ and $(A - S_w) = \{x_{j+1}, x_{j+2}, \cdots, x_n\}$. Then $M(f, S_w)$ will be as shown in Fig. 5(a) and will have a row multiplicity of N_w or less. Then the complete table for f will appear as shown in Fig. 5(b) where the small number of columns, N_w or fewer, appear again and again (in arbitrary order) down the table. If the proposed function does not have this kind of repetitiveness, the signature table system with this partitioning of variables and this size constraint will not be able to represent it. Thus the designer of a signature table learning

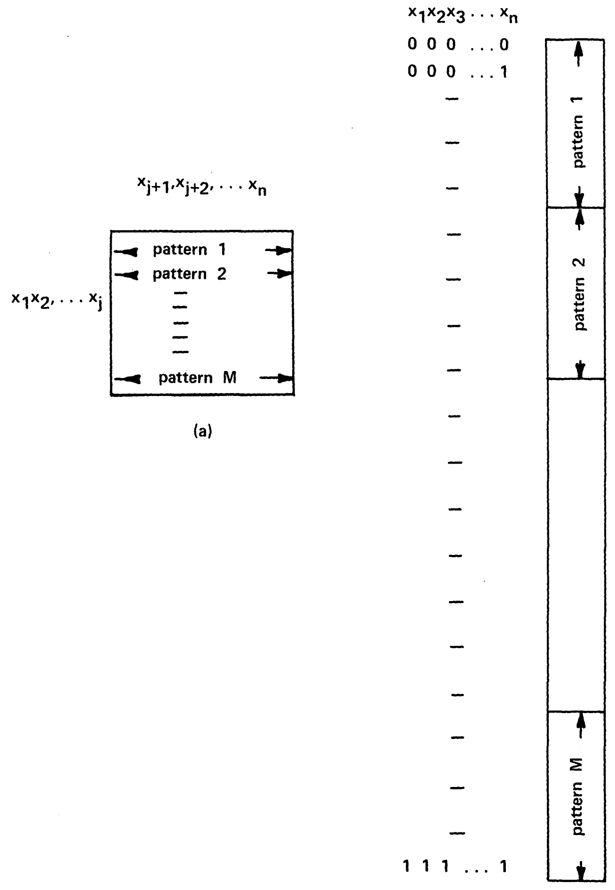


Fig. 5. (a) Example $M(f, S)$. (b) Associated function table.

system makes sharp limitations on the class of learnable functions as the design decisions are made and those limitations are of the nature described here.

The matrix $M(f, S_w)$ provides an easy method for filling in a given table schema to realize a given function. Assume X_w is a particular instantiation of variables in S_w and that z is the row in $M(f, S_w)$ which corresponds to X_w. Let a_z be an abstract symbol. Then the output entry in table t_w which corresponds to X_w is a_z if t_w is not the top table and is z if t_w is the top table.

Fig. 6 gives an example of this procedure where (a) defines the function to be realized by a schema of form (d). However, we assume the outputs for the two tables t_1 and t_{11} in (d) are not known and must be derived. These outputs are found for lowest level tables first and higher level tables in succeeding stages. In this case, the outputs for t_{11} are found first as explained in the above paragraph. $M(f, S_{11})$ is constructed as shown in (b) and it is noted that there are two distinct rows, 0 1 and 1 0. The row multiplicity is two and abstract symbols a_{01} and a_{10} are created and filled into t_{11} as shown in (d). Next the synthesis proceeds to table t_1 and $M(f, S_1)$ as shown in (c) is created. Row multiplicity is again two for this table but, as explained in the previous paragraph, outputs 0 and 1 are used instead of a_0 and a_1 since this is a top table. Table t_1 is thus filled in as shown.

Suppose the function of Fig. 7(a) is to be represented by a signature table system but it is not known what the form

$x_1x_2x_3$	f
0 0 0	0
0 0 1	1
0 1 0	1
0 1 1	0
1 0 0	0
1 0 1	1
1 1 0	0
1 1 1	1

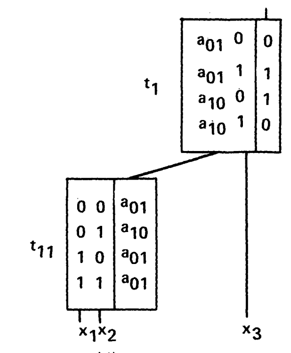
(a)

$$\begin{array}{c|cc} x_1x_2 \backslash x_3 & 0 & 1 \\ \hline 0\,0 & 0 & 1 \\ 0\,1 & 1 & 0 \\ 1\,0 & 0 & 1 \\ 1\,1 & 0 & 1 \end{array}$$

(b)

$x_1x_2x_3$	
0 0 0	0
0 0 1	1
0 1 0	1
0 1 1	0
1 0 0	0
1 0 1	1
1 1 0	0
1 1 1	1

(c)

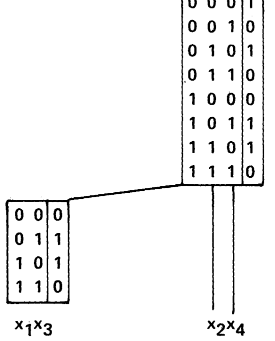


(d)

Fig. 6. (a) Function to be realized. (b) $M(f, S_{11})$. (c) $M(f, S_1)$. (d) Instantiated schema.

$x_1x_2x_3x_4$	f
0 0 0 0	1
0 0 0 1	0
0 0 1 0	0
0 0 1 1	1
0 1 0 0	1
0 1 0 1	0
0 1 1 0	1
0 1 1 1	0
1 0 0 0	0
1 0 0 1	1
1 0 1 0	1
1 0 1 1	0
1 1 0 0	1
1 1 0 1	0
1 1 1 0	1
1 1 1 1	0

(a)



(b)

Fig. 7. (a) Function to be represented by signature table system. (b) Solution found.

of the table system should be. Then one can think of Fig. 7(a) as an output table t_1 and attempt to find a subset S of the variables which can be *drawn out* to be inputs to a lower level table t_{11}. If each $M(f, S)$ is examined for each possible nontrivial subset $\{x_1, x_2\}, \{x_1, x_3\}, \cdots, \{x_2, x_3, x_4\}$ in the hope of finding an $M(f, S)$ whose row multiplicity is small compared to the number of rows, it is found that $S = \{x_1, x_3\}$ provides a good decomposition as shown in Fig. 7(b). In general, this process can be repeated drawing out a new set of variables each time either from the top level table t_1 or from some lower level table. This is an extremely expensive process computationally but no important shortcuts to a perfect solution are known.

In summary, this section has given two results. First, Theorem 1 gives necessary and sufficient conditions for a particular signature table schema to realize a given function. Second, a synthesis technique is given for finding a table schema which realizes a given function. If the form of the table schema is known, then the associated $M(f, S)$ matrices are constructed which give the needed outputs for the individual tables. If the form of the table schema is not known and must be derived, then every possible $M(f, S)$ must be constructed and an optimum decomposition must be chosen according to some criterion. Section IV will examine this procedure again for the case where the data may include some noise.

III. Evaluating the Power of a Signature Table Schema

An important point mentioned in the last section is that the size limitations on the table output alphabets restrict the power of the table schema so that it may compute only a fraction of the set of all possible functions. For example, we might wonder how many different functions can be represented by the schema of Fig. 8(a) if all output alphabets are limited to size two or less. We show in this section that there are a total of 18 664 such functions. This may be compared to the total number of binary functions of six binary variables $2^{2^6} = 18\ 446\ 744\ 073\ 709\ 551\ 616$ showing that the signature table system computes only a tiny fraction of the set of all possible functions. This section shows how to compute the number of representable functions for any table schema so that these limitations can be evaluated.

Call a signature table schema with its table output alphabet sizes $n_1, n_{11}, n_{12}, \cdots$, specified a *sized schema.* If T is a sized schema, call P_T the set of all sized schemas identical to T in structure and inputs, and with the same output alphabet size n_1 specified in the top table as T, but with output alphabet sizes in the nontop tables less than or equal to the sizes specified in the corresponding tables in T. Note that T is in P_T. The example sized schema T of Fig. 8(a) has 16 members in its P_T as shown in Fig. 8(b)–8(q).

For each sized schema in P_T we will count a particular set of functions called the functions *specified* by the schema. A function specified by a sized schema is any function

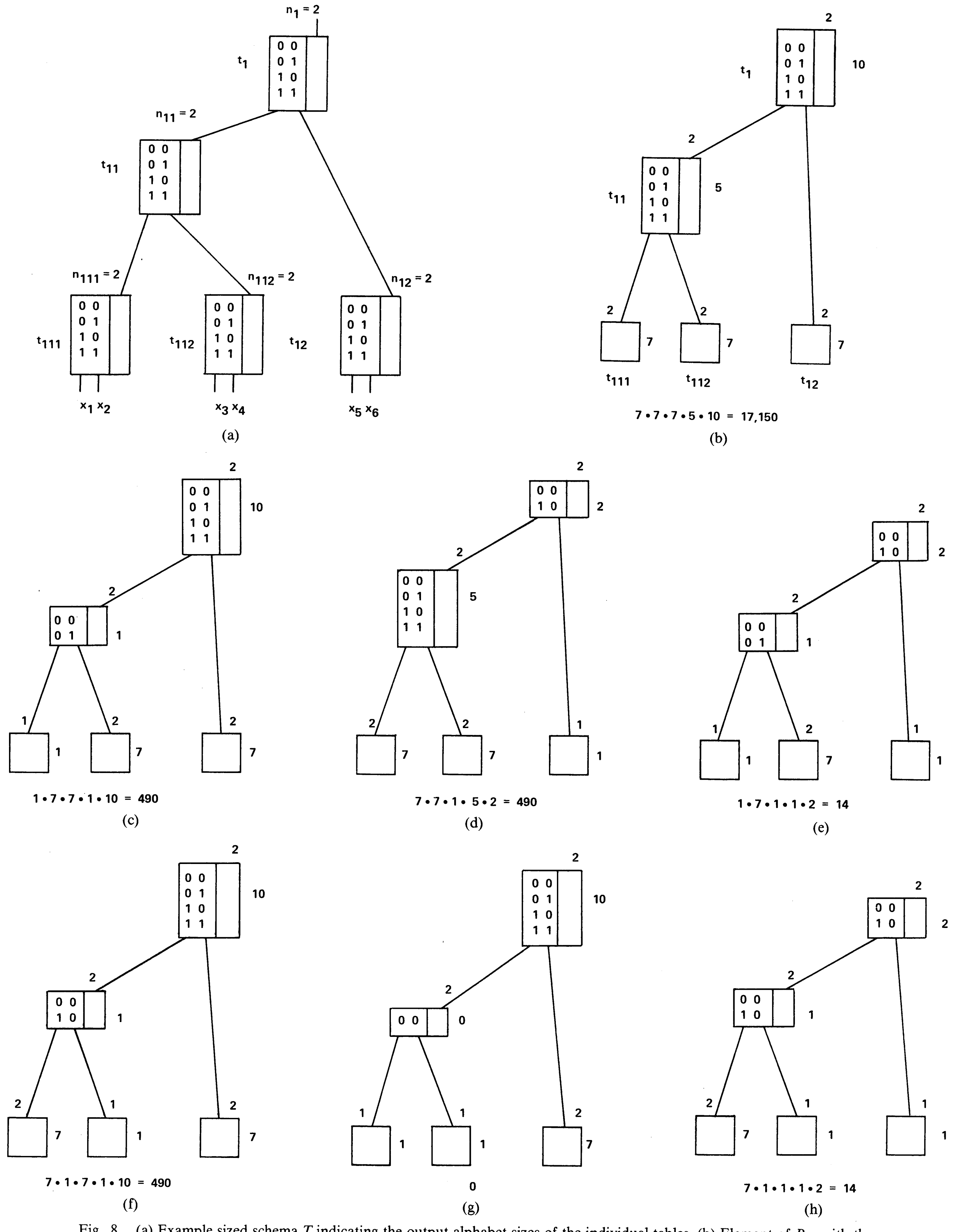


Fig. 8. (a) Example sized schema T indicating the output alphabet sizes of the individual tables. (b) Element of P_T, with the output alphabet sizes and number of permitted output columns for each individual table. 17 150 functions. (c) Element of P_T. 490 functions. (d) Element of P_T. 490 functions, (e) Element of P_T. 14 functions, (f) Element of P_T. 490 functions. (g) Element of P_T. No realizable functions. (h) Element of P_T. 14 functions. (i) Element of P_T. No realizable functions. (j) Element of P_T. No realizable functions. (k) Element of P_T. No realizable functions. (l) Element of P_T. No realizable functions. (m) Element of P_T. No realizable functions. (n) Element of P_T. No realizable functions. (o) Element of P_T. 14 functions. (p) Element of P_T. No realizable functions. (q) Element of P_T. 2 functions.

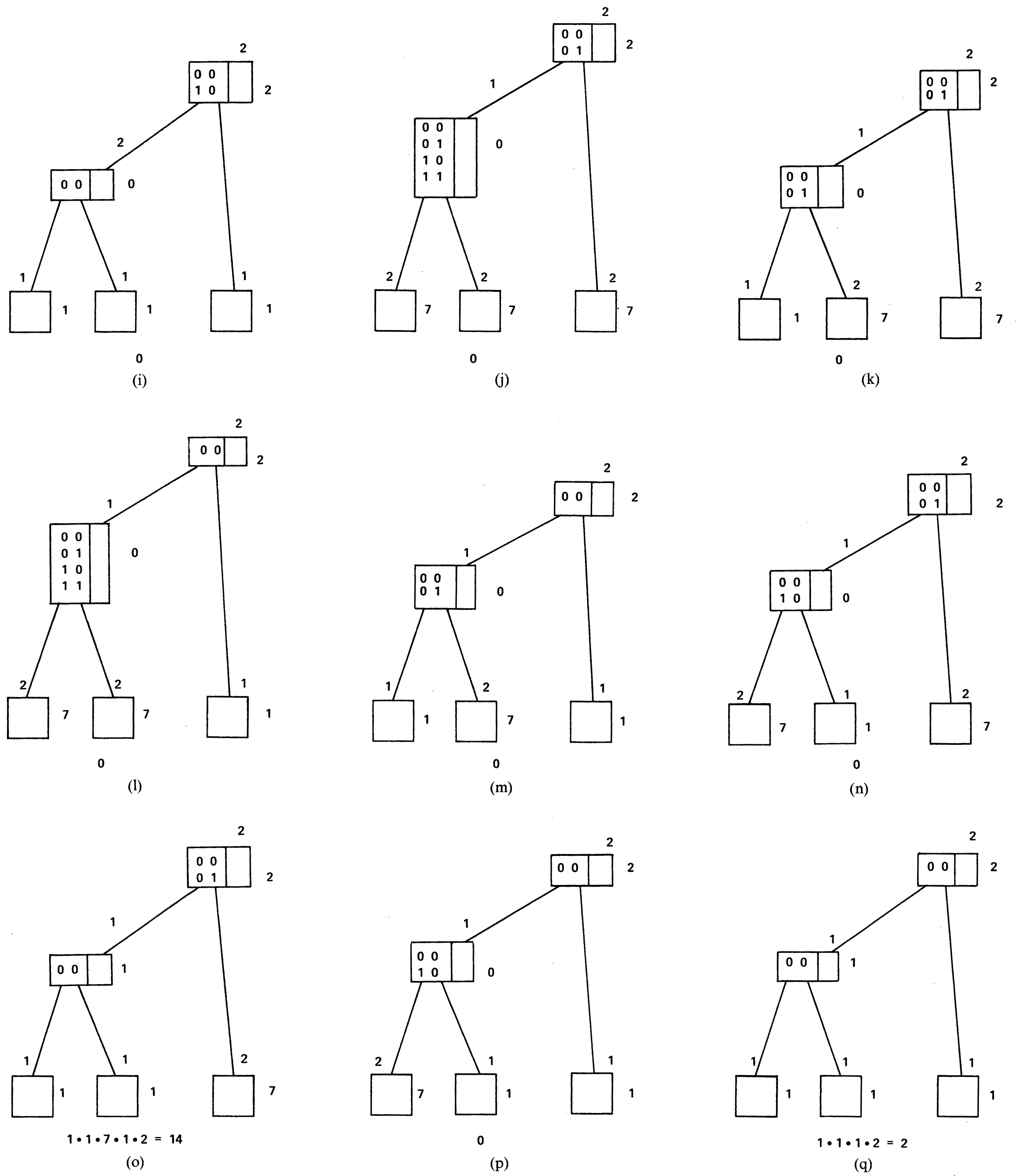


Fig. 8. *Continued.*

representable by the sized schema whose row multiplicity in each *nontop* table is *equal* to the output alphabet size specified for that table.

We need to show that if we count the number of functions specified by each sized schema in P_T, and add them up, that we will have counted the set of functions representable by T. The sets of functions specified by two different elements of P_T are disjoint, since a function specified by one is guaranteed to have a different row multiplicity in some nontop table than any function speci-

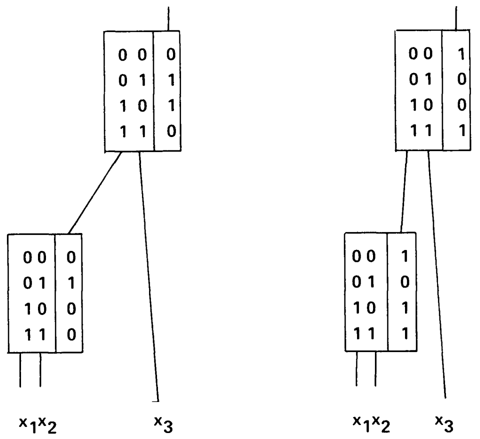


Fig. 9. Two different instantiations of same function.

fied by the other. Further, it follows from Theorem 1 that any function representable by T has row multiplicity in each table in T less than or equal to the output alphabet size for that table, and is thus specified by some element of P_T.

We now turn to counting the functions specified by a given sized schema, by restricting the kinds of output columns we will permit in our instantiations of the schema such that each specified function is represented precisely once. We introduce three important notions.

The first is that all nontop tables must be *exact* in the sense that a proposed output column for a given table must have exactly the specified number of symbols in it. It is worth noting that all of the entries in all of the tables of a system will be utilized in some computation if and only if all of its individual nontop tables are exact. If Y is a proposed output column for individual table t, we will write $E_t(Y) = 1$ if Y does have the specified alphabet size for t, and $E_t(Y) = 0$ otherwise.

Secondly, consider an individual table in a system of tables and assume it has k input variables of alphabet sizes $m_1, m_2, \cdots, m_k$. This table will have $m_1 m_2 \cdots m_k$ output entries but not every different set of outputs will yield a different total system function. Fig. 9, for example, shows two different instantiations for the same function. The point is that the outputs for any table except the top table are abstract symbols for internal bookkeeping only which may be interchanged or renamed in any way without affecting the system performance. So only one naming of the output symbols for a nontop table will be considered, the *canonical* naming which is assigned as follows. The first symbol is zero and every subsequent symbol is either identical to a previous symbol or one larger than the highest digit seen thus far. Thus the canonical representations for 4142150 and 7772721 are 0102134 and 0001012, respectively. Let Y be a proposed column of output entries for a table. Define $C(Y) = 1$ if Y is canonical and $C(Y) = 0$ otherwise.

Finally, let $g(u_1, u_2, \cdots, u_k)$ represent the function computed by the individual table under consideration and suppose that u_i where $1 \leqslant i \leqslant k$ is an output from a lower level table. Remember that u_i has an alphabet of m_i symbols and that no functions are to be counted which can use less than m_i symbols in that alphabet. Then it must be true that there do not exist two different symbols a and b in the alphabet of u_i such that $g(u_1, u_2, \cdots, u_{i-1}, a, u_{i+1}, \cdots, u_k) = g(u_1, u_2, \cdots, u_{i-1}, b, u_{i+1}, \cdots, u_k)$ for all $u_1, u_2, \cdots, u_{i-1}, u_{i+1}, \cdots, u_k$ because this would mean that a and b could be merged allowing the total function to be computed with fewer than m_i symbols in the u_i alphabet. If a proposed column Y of output entries for a table satisfies the condition that no two distinct input symbols from a lower level table can be so merged, then Y will be called *irreducible*. Fig. 3(a) shows a table t_1 whose output column is not irreducible because the symbols 0 and 2 from the previous table t_{11} have identical behaviors and can be merged. If Y is a proposed column of output entries for table t, define $I_t(Y) = 1$ if Y is irreducible and $I_t(Y) = 0$ otherwise. These considerations are summarized in the following theorem which is proved in the Appendix.

Theorem 2: If a signature table system is irreducible, it has a unique instantiation representing the same function such that every individual table is irreducible and every nontop table is canonical and exact.

Theorem 2 can serve as the basis for computing the number of functions specified by a sized schema. One simply computes the number of permitted output columns for each individual table and multiplies the resulting values together. Let Q represent the set of all possible output columns for an individual table t. Then the number of permitted output columns is $\Sigma_{Y \in Q} I_t(Y)$ if t is a top table, and $\Sigma_{Y \in Q} C(Y) E_t(Y) I_t(Y)$ if t is not a top table.

We will now consider in order bottom tables with only direct inputs, top tables, and nontop tables computing the number of functionally distinct output columns in each case. Consider the case of bottom tables with no inputs from other tables. These are trivially irreducible. For such a nontop table it is only necessary to compute the number of canonical exact output columns, which for a table with output alphabet size n and of output column length m is zero if $m < n$, else

$$\frac{\sum_{i=0}^{n} (-1)^i \binom{n}{i} (n-i)^m}{n!}$$

(formula A, bottom table).

This is a well-known formula (Riordan [13], pp. 32–34, 99) which for further reference we will call $A\ (n, m)$. It is the number of ways of placing m different objects into n nonempty sets. A development of $A(n, m)$ is given in the Appendix. The notation $\binom{n}{i}$ refers to the binomial coefficient $n!/(i!(n-i)!)$.

Consider next top tables. We will consider an easy special case first and then develop a general formula. Suppose a top table has direct inputs as well as inputs from lower tables. Let g be the product of the alphabet sizes of the direct inputs. If there are none, then g equals one.

An input from a lower table with alphabet size equal to one can essentially be disregarded. If all inputs from lower tables are size one, the table is trivially irreducible and the number of permissible output columns is n^g, where n is the table's output alphabet size. If there is just one input from a lower table with alphabet size v greater than one, the number of irreducible output columns is 0 if $n^g < v$, else

$$\frac{n^g!}{(n^g - v)!}$$

(top table with one nontrivial input from a lower table and some direct inputs) where $0! = 1$.

We can now consider top tables in general. The number of different output columns of a top table with alphabet size n is n^m, where m is the length of the column. It follows from Theorem 1 that any function represented by a reducible output column of a top table t can be represented irreducibly by some smaller table t' similar to t save that some of the inputs from lower tables have smaller alphabet sizes than in t. So if one computes the number of reducible output columns of t that "reduce" to irreducible output columns in a given smaller t', for all t' smaller than t, and adds them up, one has the number of reducible output columns of t, which subtracted from n^m gives the number of irreducible output columns.

The nature of a reducible output column is that it ignores any distinction between some alphabet symbols of its inputs from lower tables. Fig. 10 shows three different output columns in an example table t that reduce to the same irreducible output column in the t' shown.

If t has input alphabet sizes from lower tables $u_1, u_2, \cdots, u_j$ and t' is similar to t except for its input alphabet sizes from lower tables $v_1, v_2, \cdots, v_j$, where the v are less than or equal to corresponding u's, then the number of output columns of t that reduce to a given irreducible output column in t' is $A(v_1, u_1)A(v_2, u_2) \cdots A(v_j, u_j)$ where A is the function referred to in the discussion on canonical exact output columns.

To summarize, if $t' < t$ indicates a table t' similar to but smaller than t in the sense given above, then the formula for $IR(t, n)$ the number of irreducible output columns of table t with output alphabet size n is

$$IR(t, n) = n^m - \sum_{t'<t} IR(t', n) \prod_{i=1,j} A(v_i, u_i)$$

(general top tables) where m is the length of the output column in t. This is a recursive formula, since to calculate $IR(t, n)$ one must calculate $IR(t', n)$ for all t' smaller than t. Fortunately the smallest tables, those with only one input alphabet size v from a lower table greater than one, can be dealt with by the closed form $n^g!/(n^g - v)!$ mentioned previously.

Turning now to nontop tables, we remember from Theorem 2 that they must be canonical, exact, and irreducible. The general formula for the number of canonical exact irreducible output columns of table t with output alphabet size n is

$$\frac{\sum_{i=0}^{n} (-1)^i \binom{n}{i} IR(t, n-i)}{n!}$$

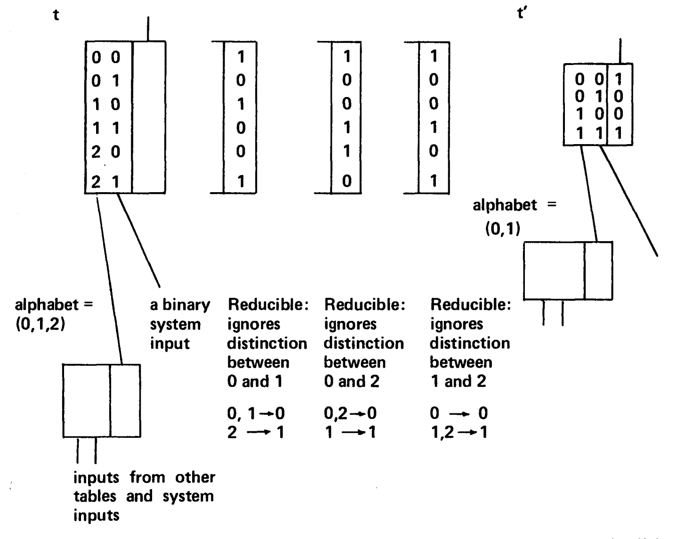

t
t'
0 0
0 1
1 0
1 1
2 0
2 1
1 0 1 0 0 1
1 0 0 1 1 0
1 0 0 1 0 1
0 0 1
0 1 0
1 0 0
1 1 1
alphabet = (0,1)
alphabet = (0,1,2)
a binary system input
Reducible: ignores distinction between 0 and 1
0, 1→0
2 →1
Reducible: ignores distinction between 0 and 2
0,2→0
1 →1
Reducible: ignores distinction between 1 and 2
0 → 0
1,2→1
inputs from other tables and system inputs

Fig. 10. Three output columns in t that reduce to same irreducible output column in t'.

(general nontop tables). This is again similar to function A mentioned above, and reduces to it when none of the table's inputs come from lower tables. See the Appendix for a development of this formula and the function A.

Simpler formulas can be given for a table system with binary inputs and binary table outputs in all nontop tables, and where tables have either system inputs exclusively or inputs from other tables exclusively. If t is a top table with r different binary inputs from previous tables (and no direct inputs), the number of irreducible columns is shown in the Appendix to be

$$\sum_{i=0}^{r} (-1)^i \binom{r}{i} n^{2^{(r-i)}}$$

(top table with binary inputs) where n is the output alphabet size.

If t is not a top table and has r binary inputs from previous tables (and no direct inputs) and a binary output, then the number of irreducible canonical exact output columns is exactly one-half the number of irreducible output columns:

$$\frac{1}{2} \sum_{i=0}^{r} (-1)^i \binom{r}{i} 2^{2^{(r-i)}}$$

(nontop table, no direct inputs, binary input and output).

For an example, these formulas can be used to compute the number of functions specified by the sized schema T of Fig. 8. Considering the first member of P_T in Fig. 8(b), formula A applies to tables t_{111}, t_{112}, t_{12} and tells us there are seven canonical exact output columns. They are

```
0  0  0  0  0  0  0
0  0  1  0  1  1  1
0  1  0  1  0  1  1
1  0  0  1  1  0  1
```

Table t_1 is a top table with binary inputs and the associated formula indicates there are ten irreducible columns. They

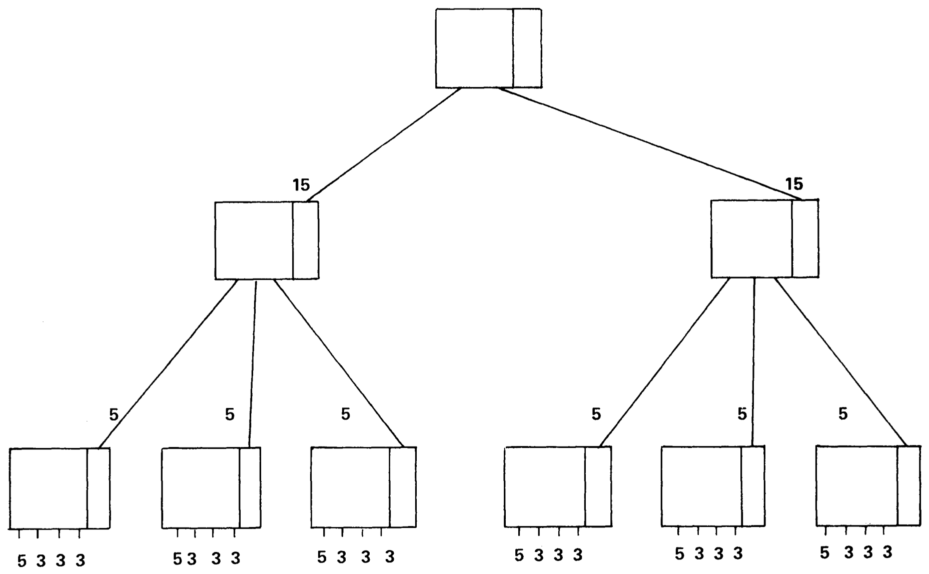


Fig. 11. Schema used by Samuel with alphabet sizes indicated.

are

$$\begin{matrix} 0&0&0&0&0&1&1&1&1&1\\ 0&0&1&1&1&0&0&0&1&1\\ 0&1&0&1&1&0&0&1&0&1\\ 1&0&0&0&1&0&1&1&1&0 \end{matrix}.$$

The formula for a nontop binary table with no direct inputs applies to table t_{11} and gives the number of irreducible canonical exact columns, five:

$$\begin{matrix} 0&0&0&0&0\\ 0&0&1&1&1\\ 0&1&0&1&1\\ 1&0&0&0&1 \end{matrix}.$$

So the number of functions specified by this member of P_T is $7 \times 7 \times 7 \times 10 \times 5 = 17\,150$.

The number of functions specified by each of the other sized schemas in P_T is similarly calculated and given in Fig. 8(c)–(q). The sum then of all functions representable by the sized signature table schema T is 18 664.

IV. Toward the Construction of a Practical Learning System

The signature table systems that have been used as examples in this paper are much smaller than those which might appear in practical situations. This section will show how the insights of the model can be used to help in the understanding and design of much larger systems.

In the typical application, there may be several dozen measurements (features) from a particular domain and possibly thousands of examples of input–output data which the table system is to match. Furthermore, the data may be noisy in the sense that a particular input may give different outputs on different occasions while other inputs may have no specified output at all. Thus the signature table system is expected to give only an approximation to perfect performance.

An example of such an application is described by Samuel [6] where signature tables were used to give a quantitative evaluation of "goodness" or "badness" of checkerboard positions. The inputs to the system were the values of a number of board features such as the number of kings, the number of unrestricted moves, or the number of occupied first row squares, and the output was a number which would (hopefully) be larger for good positions than bad ones. Various signature table schemas were tried but one example form used three levels of tables as shown in Fig. 11. Other forms were also tried but the details are not important for the discussion here.

The Samuel application is one where the desired output is not necessarily well defined. That is, it is possible that there are some combinations of feature values which reflect no possible board positions. It is also possible that a very good position and a poor one might have the same feature values even though tremendous efforts were taken to design features so that this would not happen. So the system output must give a compromise value which will (hopefully) give reasonable performance most of the time and the system will necessarily make occasional errors.

If we review the discussion of Section II and study the form of the Samuel schema, it is possible to build an image of the form of the function represented. Thus, since the leftmost bottom table has an output alphabet size of five, a table for the represented function would have five repeated patterns down the output column as shown in Fig. 5(b). Or in terms of $M(f, S)$, there would be five distinct rows in the matrix if S is the set of inputs associated with that table. Each other nontop table forces a similar restriction of its own on the form of the total function. To the extent that good checker evaluations can be made by a function with these characteristics using Samuel's features, this table configuration will be useful. To the extent that the game of checkers violates these restrictions, this signature table system will make errors.

The central issue to be faced in the usage of signature tables in real applications is how to compute the table outputs from the possibly voluminous and noisy input–

output data. For example, Samuel attempted to find coefficients that would successfully mimic 183 877 book moves for checkers. The strategies of Section II for constructing $M(f, S)$ matrices and looking for identical rows would seem to be inappropriate both because of the memory and time requirements. The rows of the $M(f, S)$ matrices would have lengths in the thousands or millions and the cost of comparing two of them would be very high.

In order to get started on this problem, we will examine Samuel's approach:

> Two totals (called D and A) are accumulated for each of the possible signature types. Additions of 1 each are made to the D totals for each signature for the moves that were not identified as the preferred book move and an addition of n, where n is the number of nonbook moves, is made to the A totals for the signatures identified with the recommended book move.

In other words, instead of characterizing a signature type by its row in the $M(f, S)$ matrix as described in Section II only two counters are used. The savings in memory space is clearly large if the rows are long but the amount of information lost is proportionately large.

Samuel then computes a coefficient C for each table entry:

$$C = \frac{(A - D)}{(A + D)}$$

where C is interpreted as a correlation coefficient having a value of $+1$ if the table entry always predicts the book move, a value of -1 if it never predicts the book move, and a value near zero if there is no correlation between the table entry and the book.

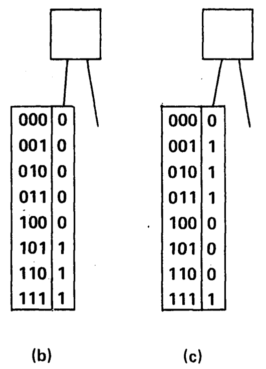
Finally, if the table output alphabet is to have size n, the various values of C appearing in the table are quantized into n groups and the actual entry in the table associated with each C will be the symbol for the group within which C falls. Thus two entries in the table will have the same output value if they have similar C which means they have similar ratios of A to D.

Now let us follow Samuel's lead as others [5], [9], [14] have done and construct a learning scheme based upon counters A and D. We wish to construct a binary function with outputs 1 and 0 and we will let A and D count the number of ones and zeros, respectively, in the $M(f, S)$ matrix row for each signature type. (This is similar to the Samuel method applied to a game with only two moves at each point, the book move and another move, and with one example of every input–output behavior.) Then we will compute and quantize C coefficients using the Samuel method to obtain table output values. We will call this the *summing method* for finding table outputs.

Let us now examine what the summing method does. Consider, for example, the matrix of Fig. 12(a) where A records the number of ones in a row and D records the number of zeros. Then the summing method would classify rows with similar proportions of zeros and ones together,

													A	D	C
000	0	1	1	1	1	0	0	1	1	0	0	0	6	6	0
001	0	0	0	0	1	1	1	0	1	1	1	0	6	6	0
010	0	0	0	1	1	0	1	0	1	0	1	1	6	6	0
011	0	0	0	0	1	1	1	0	1	0	1	1	6	6	0
100	0	1	1	1	0	1	0	1	1	0	0	0	6	6	0
101	0	1	1	1	1	0	0	1	1	1	0	0	7	5	1/6
110	1	1	0	1	1	0	0	1	1	1	0	0	7	5	1/6
111	0	0	0	1	1	1	1	0	1	0	1	1	7	5	1/6

(a)



000	0
001	0
010	0
011	0
100	0
101	1
110	1
111	1

(b)

000	0
001	1
010	1
011	1
100	0
101	0
110	0
111	1

(c)

Fig. 12. (a) Computing A, D, and C for matrix $M(f, S)$. (b) Applying summing method. (c) Classification on basis of row similarity.

yielding the output columns shown in Fig. 12(b). This is a poor classification, since two rows with similar numbers of zeros and ones could still be everywhere different. Assuming all input configurations are equally probable, the probability of error using this classification is 0.28.

A much wiser policy as was indicated in Section II is to classify table entries together if they have identical or nearly identical rows in $M(f, S)$ as shown in Fig. 12 (c). In this case, with proper instantiation of the top output table, the probability of error drops to 0.11. In other words, the summing method classifies two rows in $M(f, S)$ the same if they have the same "weight" and this is in general a very gross approximation. One should classify two rows the same only if they differ on a small number of entries.

As a second example, one could apply the method to find the output values for the tables of Fig. 1. Here it will be found that all coefficients evaluate to zero so the resulting table system will compute the function which always yields zero. This gives an error probability of one-half when an instantiation exists with no error at all.

These examples, of course, are contrived to show this method in its worst light. It does no better than a random decisionmaker in the worst case. But it is hoped that they help to illuminate the nature of the summing method and a direction to look for improvements. The next section will show that improvements can be achieved.

It is clear that Samuel's technique applied to checkers does a great deal better than random behavior. For example, let $p\ (i)$ be the observed probability that in a checker decision situation the system will rank i moves higher than or equal to the book recommended move. Then he gives the following table of performance after his system analyzed 173 989 moves.

i	$p(i)$
0	0.38
1	0.26
2	0.16
3	0.10
4	0.06
5	0.03
6	0.01

It would appear that this very respectable level of performance was achieved because a) there must have been a significant correlation between the weights of the rows in $M(f, S)$ and their profiles and b) the different signatures were not equiprobable as we have assumed for the sake of rough approximation.

V. An Improved Learning Technique

It would be desirable to make a complete comparison of rows in $M(f, S)$ and classify rows together if they are identical or nearly identical. However, this is often impractical because of the immense amount of data, so the following compromise was tried: the rows of $M(f, S)$ were divided into r (equal) segments and C coefficients were computed for each segment as in the previous section. This step reduced the rows to vectors of r elements, and then a standard algorithm for clustering vectors was used to partition the rows into the desired number of groups.

The parameter r can, of course, be varied to obtain the proper trade-off between accuracy and computational cost. If r is set to 1, the method is identical to the summing method except that the clustering algorithm may partition the rows differently. With r at 1, the computation is inexpensive because it requires little more than computing A and D for each row. At the other extreme, r may be set to the length of the rows. This allows for a complete comparison of the rows maximizing accuracy but incurring all of the associated cost in memory space and computation. If r is set to an intermediate value, then a significant improvement can be achieved over the summing technique at a moderate cost. As an example, if the rows in $M(f, S)$ of Fig. 12(a) are segmented into $r = 4$ parts, the A, D, and C can be computed as shown in Fig. 13.

The clustering algorithm which was used in this study was the "k-means algorithm" as described by Hartigan [15].

1) The set of vectors was partitioned into k clusters using some arbitrary method. In this study, the summing method was used to find the initial partition.

2) Let $d(v, K)$ represent the distance from vector v to the center of cluster K (found by averaging all the vectors in K).

a) The center of each cluster was computed.

b) For each vector v, the cluster K was found such that $d(v, K)$ was minimum. If K did not contain v, then v was moved from its current cluster into K and control returned to Step 2a). The computation proceeded until one complete pass over the vectors produced no changes to the partition.

	A	D	C	A	D	C	A	D	C	A	D	C
000	2	1	.33	2	1	.33	2	1	.33	0	3	-1.0
001	0	3	-1.0	2	1	.33	2	1	.33	2	1	.33
010	0	3	-1.0	2	1	.33	2	1	.33	2	1	.33
011	0	3	-1.0	2	1	.33	2	1	.33	2	1	.33
100	2	1	.33	2	1	.33	2	1	.33	0	3	-1.0
101	2	1	.33	2	1	.33	2	1	.33	1	2	-.33
110	2	1	.33	2	1	.33	2	1	.33	1	2	-.33
111	0	3	-1.0	3	0	1.0	2	1	.33	2	1	.33

Fig. 13. A, D, and C for the segmented rows of Fig. 12(a).

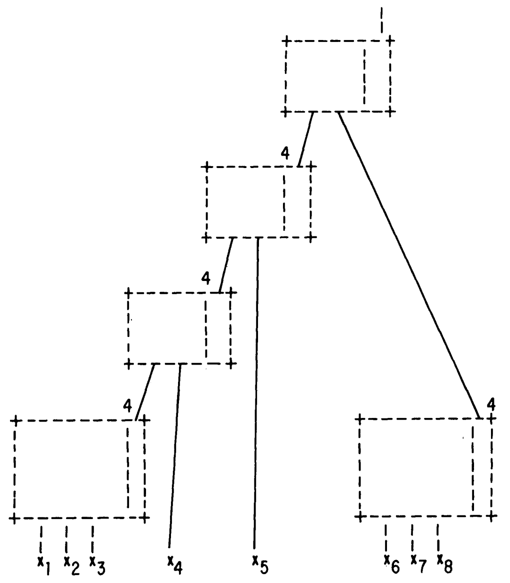


Fig. 14. Sized schema for tests: Schema 1.

This algorithm finds a local but not a global optimum for the partition. The Euclidean distance was used for $d(v, K)$ although there may be distance functions that would perform better.

Applying this algorithm to the example of Fig. 13, the initial partition would be as indicated in Fig. 12(b): rows 1, 2, 3, 4, and 5 would be classified together, and rows 6, 7, and 8 would constitute the other group. This partition will be denoted $(1, 2, 3, 4, 5)$, $(6, 7, 8)$. The centers of these clusters are at $(-7/15, 1/3, 1/3, -3/15)$ and $(-1/9, 5/9, 1/3, -1/9)$, respectively. The Euclidean distances of $v_1 = (1/3, 1/3, 1/3, -1)$ from the two cluster centers are about 1.28 and 1.04 indicating that v_1 should be moved to the second cluster. This yields the partition $(2, 3, 4, 5)$, $(1, 6, 7, 8)$. Continuing the calculation results in row 5 moving to the second group and row 8 moving to the first. This yields the much better performing partition $(2, 3, 4, 8)$, $(1, 5, 6, 7)$ which is indicated in Fig. 12(c).

Once this learning procedure was devised, a series of experiments were run to test its usefulness. Two schemas were chosen as shown in Figs. 14 and 15 and specific instantiations were made. Thus a particular realizable function f was known in each case, and the target functions for learning were generated by making a fixed number c of random changes to f. This procedure has the advantage

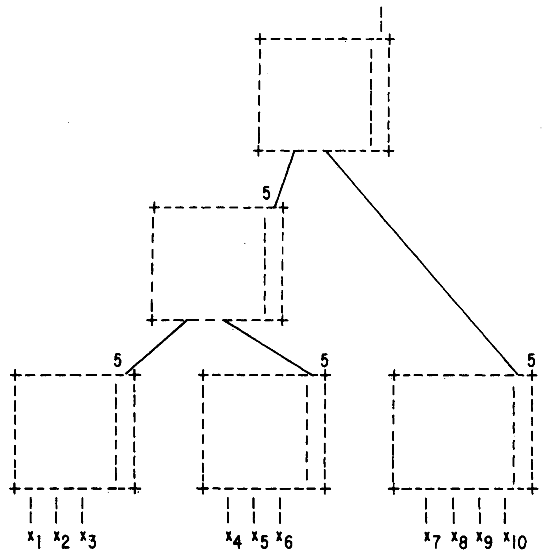


Fig. 15. Schema 2.

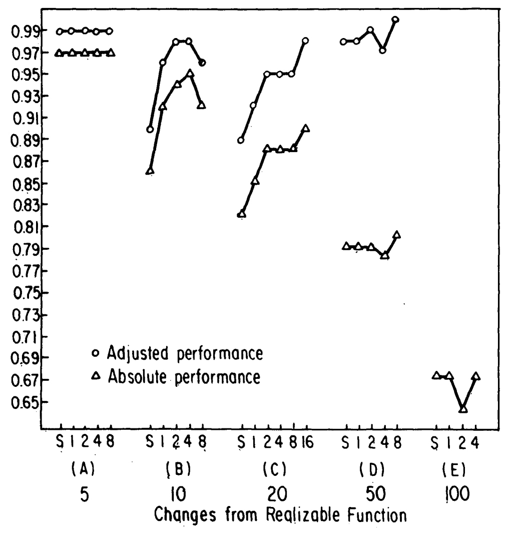


Fig. 16. Performance increases through clustering and partitioning for Schema 1. S = Summing method. 1 = Clustering performed on 1-column tables; no partitioning. 2 = Partitioning on 2 columns with clustering. 4 = Partitioning on 4 columns with clustering. 8 = Partitioning on 8 columns with clustering.

that it gives the experimenter a measure of the quality of the learning algorithm. That is, suppose the target function f_t is f with c changes and that the table system after learning computes f_t with e errors. If e is much greater than c, the learning algorithm performed poorly because it could have found realization f with only c errors. If e nearly equals c, the algorithm has probably done as well as could be hoped for. It is possible that e be less than c but this is not common.

The graphs in Figs. 16 and 17 give the results of the experiments. The "absolute performance" is given indicating the fraction of the target function f_t that was correctly computed by the table system after learning. Also, the "adjusted performance" is given which assumes that a minimum of c errors will be made by a perfect learning system. Thus an adjusted performance rating of 1.0 means

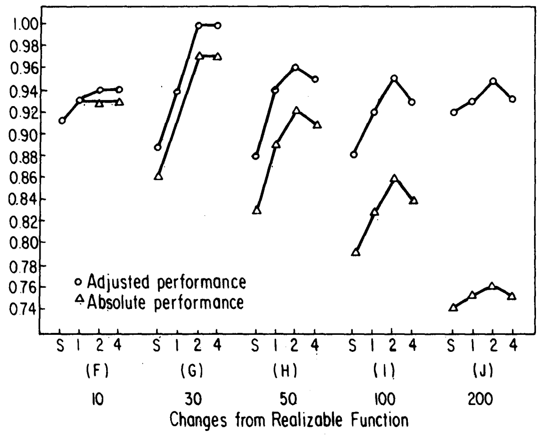


Fig. 17. Performance increases through clustering and partitioning for Schema 2.

the table system makes exactly c errors. The graphs show the performance given by the summing method (indicated with an S) and the performance achieved for $r = 1, 2, 4$, and so forth. In the case labeled E, all methods found a system better than the original realizable function, so no adjusted performance is given.

The results can be summarized as follows. In most of the tests, the revised learning procedure gave substantial improvements over the summing method. However, if the summing method happened to find a very good solution as occurred in A, D, and E, then the new procedure could not improve on it. As expected, the accuracy of the learned function tended to increase as r was increased. However, peculiarities within f_t occasionally caused a deviation from the trend.

VI. Conclusion

Signature table systems have been useful for decision-making in various applications especially where learning capabilities are desired. However, studies of this mechanism have tended to be experimental in nature and a theoretical framework for understanding them has been slow to develop. This paper introduces the concept of the $M(f, S)$ matrix and shows its usefulness in understanding the form of realizable functions and in synthesizing signature table systems. It is further shown that the limitations on the computational power for such systems comes from the restrictions on the table alphabets and a technique is given for evaluating these limitations. Finally it is shown how the insights of this paper can be used in understanding and designing signature table learning systems in situations where the construction of the $M(f, S)$ matrix is not practical.

Appendix

Proof of Theorem 1: Assume irreducible signature table system T with alphabet sizes n_1, n_{11}, n_{12}, in the system with its alphabet size n_w. Then there are exactly n_w different types of instantiation of the variables in S_w. That is, all of the possible different instantiations of variables in S_w result in only n_w different inputs to the rest of the system. Let X_w

be an instantiation of variables in S_w and f_{X_w} be f with the instantiations X_w made. Then there are at most n_w different functions f_{X_w} if all possible X_w are examined. There are no fewer than n_w because T is irreducible. But the rows in $M(f, S_w)$ are just the behaviors of the different f_{X_w} as X_w is varied so Rowmult $[M(f, S_w)] = n_w$.

Next assume that Rowmult $[M(f, S_w)] = n_w$ for each S_w associated with signature table system T. Then using the construction at the end of Section II, one can show that there is an irreducible instantiation of T with alphabet sizes $n_1, n_{11}, n_{12}, \cdots$ which computes f. Specifically, let z be a row in $M(f, S_w)$ associated with instantiation X_w and let a_z be an abstract symbol. Further let a_z be the output associated with instantiation X_w in table t_w if t_w is not the top table. Let z be the output associated with instantiation X_w in t_w if t_w is the top table. This construction, if it is well defined, will most certainly result in t_w having the correct alphabet size n_w and will most certainly compute f since the top table is defined to properly output the value of f for each instantiation of all input variables. The remaining questions are whether this construction yields a unique and well-defined T and whether T is irreducible.

To show that T is uniquely defined, consider any one of its individual tables t_w with associated input variables S_w. Assume two tables t_{w1} and t_{w2} input to t_w with associated variables S_{w1} and S_{w2} so that $S_w = S_{w1} \cup S_{w2}$. (We assume for the sake of simplicity that table T_w has no direct system inputs. The more general proof is trivially different from what follows here.) Suppose X_{w1} and X'_{w1} are two instantiations of the variables in S_{w1} which yield the same output in t_{w1}, namely $a_{z_{w1}}$. Suppose X_{w2} and X'_{w2} are two instantiations of the variables in S_{w2} which yield the same output in t_{w2}, namely $a_{z_{w2}}$. In order to show uniqueness, it is necessary to show that the entry a_{z_w} in t_w associated with an input to that table of $a_{z_{w1}}, a_{z_{w2}}$ is unique. In other words, is the row of entries z_w in $M(f, S_w)$ associated with input X_{w1}, X_{w2} identical with the row associated with X'_{w1}, X'_{w2}? Rephrasing this, is it true that $f(X_{w1}, X_{w2}, Q) = f(X'_{w1}, X'_{w2}, Q)$ for all instantiations of Q where Q is the set of all input variables to f which are not in S_{w1} or S_{w2}? (Here f is being written with three arguments, the set of inputs to t_{w1}, the set of inputs to t_{w2}, and the set of all other inputs). But from the fact that X_{w1} and X'_{w1} yield the same output in t_{w1}, it is clear that 1) $f(X_{w1}, P, Q) = f(X'_{w1}, P, Q)$ for all instantiations of P and Q. Similarly, since X_{w2} and X'_{w2} yield the same output in t_{w2}, we have 2) $f(P, W_{w2}, Q) = f(P, X'_{w2}, Q)$ for all instantiations of P and Q. Applying 1) and 2) sequentially we obtain $f(X_{w1}, X_{w2}, Q) = f(X'_{w1}, X'_{w2}, Q)$ for all Q which is the desired result. One can show T is irreducible by a straightforward application of the definition of irreducibility and this completes the proof.

Proof of Theorem 2: It will be assumed that the discussion of Section III is adequate to show that if a signature table system is irreducible with alphabet sizes $n_1, n_{11}, n_{12}, \cdots$, it has a representation such that every individual table is irreducible and every nontop table is canonical and exact. It remains to be shown that this representation is unique. The proof of uniqueness proceeds by assuming there are two different instantiations T and T' of the same schema that meet the requirements of the theorem and that compute the same function f. If the two systems are identical everywhere except in the top table, they clearly represent different functions. This is true because the exactness of the nontop tables implies every entry in the top table is exercised by some system input so that any modification in the top table alone will yield a modification in the system input–output performance. So T and T' must differ in some nontop table. Choose lowest level tables t_w and t'_w where T and T' are different. Since the output columns of t_w and t'_w are canonical, there must be two instantiations of the variables in S_w, say X_w and X'_w, such that one table, say t_w, yields the same output on both X_w and X'_w and the other table, say $t_{w'}$, yields different outputs on X_w and $X_{w'}$. But since these systems are irreducible, the different outputs from $t_{w'}$ imply functions f_{X_w} and $f_{X'_w}$ are different. This contradicts the fact that f_{X_w} and $f_{X'_w}$ must be the same since t_w yields the same output on X_w and X'_w and completes the proof.

A development of the formula A, and the formula for irreducible canonical exact output columns: Let $E(n, m)$ be the number of exact output columns of alphabet size n and length m. Now $E(n, m)$ equals n^m minus the number of columns that leave out at least one alphabet symbol. The number of columns that leave out at least one alphabet symbol is $\binom{n}{1}(n - l)^m$ (where $\binom{n}{1}$ is the number of ways of leaving out one symbol) minus the number of columns that leave out at least two symbols, which is $\binom{n}{2}(n - 1)^m$ minus the number that leave out at least three symbols, etc. Then

$$E(n, m) = n^m - \left[\binom{n}{1}(n-1)^m - \left[\binom{n}{2}(n-2)^m - \cdots \left[\binom{n}{n}(n-n)^m\right]\cdots\right]\right]$$
$$= \sum_{i=0}^{n} (-1)^i \binom{n}{i}(n-i)^m.$$

From each canonical output column of alphabet size n and length m that uses all elements of its alphabet, $n!$ exact output columns can be generated by permuting the alphabet symbols. So

$$A(n, m) = \frac{E(n, m)}{n!} = \frac{\sum_{i=0}^{n} (-1)\binom{n}{i}(n-i)^m}{n!}.$$

The development of the formula for the number of irreducible canonical exact output columns is similar to the above. $E'(n, m)$ is defined to be the number of irreducible exact output columns of alphabet size n and length m. Now $E'(n, m)$ is equal to $IR(n, m)$ minus the number of irreducible columns that leave out at least one alphabet symbol, which is $\binom{n}{1}IR(n - 1, m)$ minus the number of irreducible columns that leave out at least two symbols, and so on. The number then of irreducible canonical exact output columns is

$$\frac{E'(n,m)}{n!} = \frac{\sum_{i=0}^{n}(-1)^i\binom{n}{i}IR(n-i,m)}{n!}.$$

Proof that the number of irreducible output columns for a table with r binary inputs exclusively from lower tables, and n output symbols is

$$\sum_{i=0}^{r}(-1)^i\binom{r}{i}n^{2^{(r-i)}}.$$

The number of irreducible output columns is the total number of columns n^{2^r} minus the number of reducible columns. The number of columns reducible on the first of the r inputs is the number of columns which have identical first and last halves, $n^{2^{r-1}}$. The number of columns reducible on the second of the r inputs is the number of columns whose first and third quarters are identical to their second and fourth quarters, $n^{2^{r-1}}$. There are r such cases or $\binom{r}{1}n^{2^{(r-2)}}$ reducible columns except that certain reducible columns have been subtracted out more than once because they appeared in more than one of the r groups. So some number N must be added back in to account for the over subtraction. At this point the total number of output columns is $n^{2r} - \binom{r}{1}n^{2^{r-1}} + N$ and N is yet undetermined.

Consider the first two inputs. If a column is identical in all of its first, second, third, and fourth quarters, it would be counted as reducible over the first input and the second input and would thus have been subtracted out in both of the above computations. So $n^{2^{r-2}}$ columns should be added back in because of the error related to the first two inputs. A similar term will appear for each other pair of inputs yielding a total of $\binom{r}{2}n^{2^{r-2}}$ columns which should be added back in to the total. Unfortunately the number added back in is too large because there are cases where a column added back in because of one pair of columns may be identical to a column added back in because of some other pair of columns. This leads to an additional term of $\binom{r}{3}n^{2^{r-3}}$ and an eventual general term of the form $\binom{r}{i}n^{2^{(r-i)}}$ where the sign is positive if i is even and negative otherwise.

Acknowledgment

Mr. James T. Ellis helped us in checking the accuracy of the formulas in Section III by brute force generating all possible tables in some test cases and checking our counting formulas against the actual number generated.

References

[1] N. J. Nilsson, *Learning Machines*. New York: McGraw-Hill, 1965.
[2] A. Samuel, "Some studies in machine learning using the game of checkers," *IBM J. Res. Develop.*, vol. 3, pp. 221–229, 1959, reprinted in *Computers and Thought*, E. Feigenbaum and J. Feldman, Eds. New York: McGraw-Hill, 1963, pp. 71–105.
[3] M. Minsky and S. Papert, *Perceptrons*. Cambridge, MA: MIT, 1969.
[4] A. K. Griffith, "A new machine learning technique applied to the game of checkers," Mass. Instit. Technol. Project MAC A.I. Memo 94, 1966.
[5] ____, "A comparison and evaluation of three machine learning procedures as applied to the game of checkers," *Artificial Intell.*, vol. 5, pp. 137–148, 1974.
[6] A. Samuel, "Some studies in machine learning using the game of checkers II. Recent progress," *IBM J. Res. Develop.*, vol. 11, pp. 601–617, 1967.
[7] T. R. Truscott, "The Duke checker program," *J. Recreational Math.*, vol. 12, 1979.
[8] M. Smith, "A learning program which plays partnership dominoes," *Comm. ACM*, vol. 16, pp. 462–467, 1973.
[9] C. V. Page, "Heuristics for signature table analysis as a pattern recognition technique," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-7, pp. 77–86, 1977.
[10] H. A. Curtis, *A New Approach to the Design of Switching Circuits*. Princeton, NJ: D. Van Nostrand, 1962.
[11] R. M. Karp, "Functional decomposition and switching circuit design," *J. Soc. Industrial Applied Math.*, vol. 11, pp. 291–335, 1963.
[12] R. L. Ashenhurst, "The decomposition of switching functions," *Ann. Comput. Laboratory Harvard Univ.*, vol. 29, pp. 74–116, 1959.
[13] J. Riordan, *An Introduction to Combinatorial Analysis*. New York: John Wiley and Sons, 1958.
[14] S. A. Mamrak and P. D. Amer, "Estimation of run times using signature table analysis," *NBS Special Publication 500-14* Fourteenth Comput. Perf. Evaluation User's Group, Boston, MA., Oct. 1978.
[15] J. A. Hartigan, *Clustering Algorithms*. New York: Wiley & Sons, 1975.