PUC

Series: Monographs in Computer Science
and Computer Applications

Nº 4/72

A CANONICAL FORM FOR MINIMUM STATE FINITE STATE AUTOMATA

by

A. L. Furtado

Computer Science Department - Rio Datacenter

Pontifícia Universidade Católica do Rio de Janeiro
Rua Marquês de São Vicente, 209 — ZC-20
Rio de Janeiro — Brasil

A CANONICAL FORM FOR MINIMUM STATE FINITE STATE AUTOMATA

A. L. Furtado
Associate Professor
Computer Science Department

PUC/RJ

Series Editor: Prof. A. L. Furtado - August/1972

ACKNOWLEDGMENT

We thank Prof. M. Millan for helpful discussions and for carefully revising the manuscript.

ABSTRACT

A well-known algorithm exists for obtaining from a finite state automaton (fsa) accepting a set L the corresponding minumum state fsa, which is unique up to an isomorphism (see [1], page 29)

The algorithm is indicated in [2], [4], and a thorough and detailed description is given in [3] .

If the algorithm is applied to two fsa M and M' the two resulting minimum state fsa should be isomorphic whenever M and M' are equivalent, in the sense of accepting the same L.

However the time requirement for testing for isomorphism is given by O (n!), n being the number of states in the minimum state machines.

A simple addition to the basic minimization algorithm is presented, whereby the minimum state machines will be equal rather than just isomorphic. Testing for equality is performed in linear time.

The modified algorithm yields a canonical form for all fsa accepting a given L.

The key idea-lexicographic ordering - has been suggested by [6].

1. NOTATION

An fsa is given by:

$M = (K, \Sigma, \delta, q_0, F)$

where:

- K - finite nonempty set of states; an element of K will be represented by a subscripted q;
- Σ - finite input alphabet; an element of Σ will be represented by an a bearing a subscript whenever its corresponding column in some array should be stressed;
- δ - mapping of $K \times \Sigma$ into K, defined by $\delta(q_i , a) = q_j$;
- q_0 - initial state;
- F - set of final states (subset of K).

The mapping δ can be represented by means of a transition matrix, say A, where if $\delta(q_i , a_j) = q_k$ then $a_{ij} = q_k$.

In the minimum fsa the states of the original fsa are grouped into equivalence classes; they will be denoted by a subscripted Q, or by a subscripted q inside square brackets indicating some chosen representative of the equivalence class. The set of final states will be written as F.

The algorithm involves an initial partition of the set of states into two blocks, and successive refinements. At a general stage $k = 0, 1, \ldots$ we use $\Pi^{(k)}$ to indicate the current partition and $B_r^{(k)}$ to denote the r^{th} block at that stage.

2. THE ALGORITHM

The algorithm is described in a PL/I - like notation:

```
minim: PROCEDURE;
     CALL init;
     DO WHILE (refinement);
         DO  FOR EVERY block;
              IF  card (block) > 1
              THEN CALL sort;
         END;
         CALL renum;
     END;
     CALL autmin;
END  minim;
```

The input to the algorithm consists of the transition matrix, and the specification of the initial state and of the final states.

Procedure init initializes work-matrix T, using the following criterion;

a. a state will be assigned a line in block $B_1^{(0)}$ of matrix T if it is a final state; otherwise the state will be assigned a line in block $B_2^{(0)}$;

b. $t_{ij} = 1$ if for the state q_p assigned to line i we have $\delta(q_p , a_j) = q_r$, where q_r is a final state; otherwise $t_{ij} = 2$.

Procedure sort is called for each block of matrix T, if the number of lines in the block is greater than 1. It sorts the lines in ascending lexicographic order.

Procedure renum checks whether all lines in each block are identical. If they are not it splits the block into two or more blocks; doing this for every block, it renumbers the possibly increased set of blocks and updates accordingly the t_{ij}.

The renumbering follows exactly the ascending lexicographic order imposed by the sort procedure, the first block being numbered as $B_1^{(k)}$, the second $B_2^{(k)}$, etc., at a general step k = 0,1, ... If q_i is assigned to line p and $\delta(q_i , a_j) \in B_r^{(k)}$, then $t_{ip} = r$.

It is easy to see that renum performs the refinement process. In accordance with the usual refinement process renum puts two states q_i and q_j in a same block $B_u^{(k)}$ iff:

a. $q_i \in B_s^{(k-1)}$ and $q_j \in B_s^{(k-1)}$, for some r;

b. $\delta(q_i , a) \in B_{t_a}^{(k-1)}$ and $\delta(q_j , a) \in B_{t_a}^{(k-1)}$, $\forall a \in \Sigma$ and some t_a.

Procedure autmin builds the transition matrix of the minimun state automaton from matrix T. It also indicates, with respect to this minimum fsa, which is the initial state and how many the final states are (it turns out that the final states will be the ones with lowest subscript, this being the reason why it suffices to indicate their number).

The output of autmin is the output of minim itself.

3. AN EXAMPLE

We show how minim works by means of an example. Some remarks presented while showing the example will be formalized in section 4.

Let $M = (K, \Sigma, \delta, q_0, F)$ be an fsa with transition matrix:

$A =$

	a_1	a_2	a_3
q_1	q_9	q_9	q_5
q_2	q_7	q_2	q_7
q_3	q_9	q_9	q_5
q_4	q_3	q_9	q_9
q_5	q_6	q_4	q_3
q_6	q_8	q_2	q_6
q_7	q_6	q_9	q_8
q_8	q_4	q_4	q_7
q_9	q_1	q_4	q_4

and $F = \{q_1, q_3, q_5, q_7, q_8\}$

$q_0 = q_5$

Let $M' = (K', \Sigma, \delta', q_0', F')$ be another fsa with transition matrix:

$A' =$

	a_1	a_2	a_3
q_1'	q_2'	q_2'	q_3'
q_2'	q_1'	q_2'	q_2'
q_3'	q_4'	q_2'	q_1'
q_4'	q_1'	q_5'	q_4'
q_5'	q_3'	q_5'	q_3'

and $F' = \{q_1', q_3'\}$

$q_0' = q_3'$

Each step of minim as applied to M and M' is shown below:

M)

A $\xrightarrow{\text{init}}$

		a_1	a_2	a_3
1	q_1	2	2	1
	q_3	2	2	1
	q_5	2	2	1
	q_7	2	2	1
	q_8	2	2	1
2	q_2	1	2	1
	q_4	1	2	2
	q_6	1	2	2
	q_9	1	2	2

$\xrightarrow{\text{sort}}$

		a_1	a_2	a_3
1	q_1	2	2	1
	q_3	2	2	1
	q_5	2	2	1
	q_7	2	2	1
	q_8	2	2	1
2	q_2	1	2	1
	q_4	1	2	2
	q_6	1	2	2
	q_9	1	2	2

$\xrightarrow{\text{renum}}$

		a_1	a_2	a_3
1	q_1	3	3	1
	q_3	3	3	1
	q_5	3	3	1
	q_7	3	3	1
	q_8	3	3	1
2	q_2	1	2	1
3	q_4	1	3	3
	q_6	1	2	3
	q_9	1	3	3

sort →

		a_1	a_2	a_3
1	q_1	3	3	1
	q_3	3	3	1
	q_5	3	3	1
	q_7	3	3	1
	q_8	3	3	1
2	q_2	1	2	1
3	q_6	1	2	3
	q_4	1	3	3
	q_9	1	3	3

renum →

		a_1	a_2	a_3
1	q_1	4	4	1
	q_3	4	4	1
	q_5	3	4	1
	q_7	3	4	1
	q_8	4	4	1
2	q_2	1	2	1
3	q_6	1	2	3
4	q_4	1	4	4
	q_9	1	4	4

sort →

		a_1	a_2	a_3
1	q_5	3	4	1
	q_7	3	4	1
	q_1	4	4	1
	q_3	4	4	1
	q_8	4	4	1
2	q_2	1	2	1
3	q_6	1	2	3
4	q_4	1	4	4
	q_9	1	4	4

renum →

		a_1	a_2	a_3
1	q_5	4	5	2
	q_7	4	5	2
2	q_1	5	5	1
	q_3	5	5	1
	q_8	5	5	1
3	q_2	1	3	1
4	q_6	2	3	4
5	q_4	2	5	5
	q_9	2	5	5

autmin →

	a_1	a_2	a_3
Q_1	Q_4	Q_5	Q_2
Q_2	Q_5	Q_5	Q_1
Q_3	Q_1	Q_3	Q_1
Q_4	Q_2	Q_3	Q_4
Q_5	Q_2	Q_5	Q_5

$\mathcal{F} = \{Q_1 , Q_2\}$

$Q_0 = Q_1$

M')

A' init →

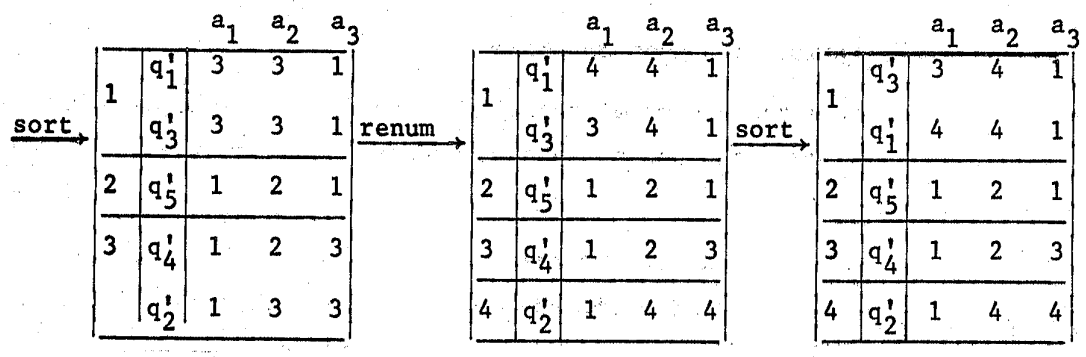
		a_1	a_2	a_3
1	q'_1	2	2	1
	q'_3	2	2	1
2	q'_2	1	2	2
	q'_4	1	2	2
	q'_5	1	2	1

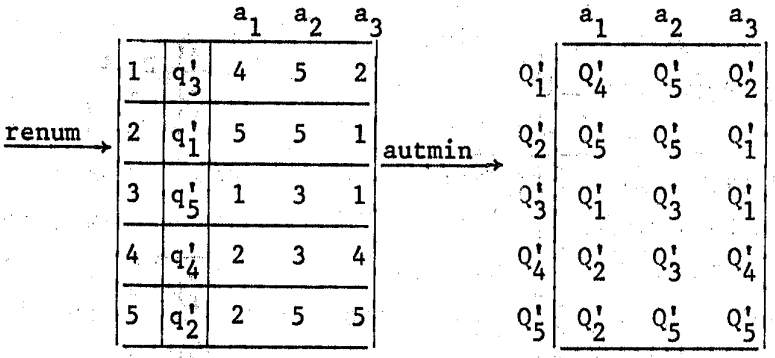
sort →

		a_1	a_2	a_3
1	q'_1	2	2	1
	q'_3	2	2	1
2	q'_5	1	2	1
	q'_2	1	2	2
	q'_4	1	2	2

renum →

		a_1	a_2	a_3
1	q'_1	3	3	1
	q'_3	3	3	1
2	q'_5	1	2	1
3	q'_2	1	3	3
	q'_4	1	2	3





$$\mathfrak{I}' = \{Q'_1 , Q'_2\}$$

$$Q'_0 = Q'_1$$

A comparison of what happens to M and M' shows that:

1. the output of minim is the same in both cases;
2. the blocks of the matrices $T^{(k)}$ and $T'^{(k)}$ have a different number of lines;
3. however, just after init, the classes of distict lines of each of the two blocks are the same, although they do not necessarily appear in the same order;

4. again, at a general stage k:

   a. after renum, matrices $T^{(k)}$ and $T'^{(k)}$ have equal classes of lines in each block, except for a permutation of the lines;

   b. after sort, matrices $T^{(k)}$ and $T'^{(k)}$ still have equal classes of lines in each block, and they come in the same (**lexicographic,** ascending) order.

We shall see in the next section that this happens if and only if M and M' are equivalent.

In the example M is not minimal and minim performs the double task of reducing it to a minimal fsa, this minimal machine being given in a canonical form. Since M' is already minimal, minim merely performs a reordering process which as before gives a **canonical** form as output.

4. THEORETICAL CONSIDERATIONS

Let $\mathfrak{M}(M)$ and $\mathfrak{M}(M')$ be the minimum state fsa corresponding to the fsa M and M'.

There is an isomorphism (see [1], page 29), between K and K' defined by:

$$\Theta: [q_i] \rightarrow [q'_j]$$

Since Θ is an fsa isomorphism, it must take final states into final states and the initial state into the initial state:

1. $\theta(\lceil q_i \rceil) = \lceil q'_j \rceil \Longrightarrow \lceil q_i \rceil \in \mathcal{F}$ iff $\lceil q'_j \rceil \in \mathcal{F}'$

2. $\theta(\lceil q_0 \rceil) = \lceil q'_0 \rceil$

Since θ is also a labelled digraph isomorphism it must preserve the adjacency relation:

3. $\theta(\lceil q_i \rceil) = \lceil q'_j \rceil \Longrightarrow \forall a \in \Sigma, \theta(\lceil \delta(\lceil q_i \rceil, a) \rceil) = \lceil \delta'(\theta(\lceil q_i \rceil), a) \rceil = \lceil \delta'(\lceil q'_j \rceil, a) \rceil$

From (3) and (1) we have:

4. $\theta(\lceil q_i \rceil) = \lceil q'_j \rceil \Longrightarrow \forall a \in \Sigma, \lceil \delta(\lceil q_i \rceil, a) \rceil \in \mathcal{F}$ iff $\lceil \delta'(\lceil q'_j \rceil, a) \rceil \in \mathcal{F}'$

One notational point deserves explanation:

In the minimal state fsa we have:

$\forall a \in \Sigma, \delta(\lceil q_i \rceil, a) = \lceil q_{ja} \rceil$

Now consider any such $\lceil q_{ja} \rceil$ which is a state in the minimal fsa. The fact that additional (trivial) refinements to the minimal fsa can neither split nor fuse nodes so as to change $\lceil q_{ja} \rceil$, can be indicated by writing:

$$\lceil q_{ja} \rceil = \lceil \lceil q_{ja} \rceil \rceil = \ldots = \lceil \ldots \lceil q_{ja} \rceil \ldots \rceil$$

where $\lceil \lceil q_{ja} \rceil \rceil, \ldots, \lceil \ldots \lceil q_{ja} \rceil \ldots \rceil$ are states in the (trivially) successively minimized fsa's, obtained from $\lceil q_{ja} \rceil$.

This means that adding an arbitrary number of matching square brackets to $\lceil q_{ja} \rceil$, and hence to $\delta(\lceil q_i \rceil, a)$ is simply a permissible notational change.

In particular it justifies our writing

$$\lceil \delta (\lceil q_i \rceil, a) \rceil$$

as a state in the minimal fsa in (3) and (4).

We show that if the minim algorithm is applied to $\mathfrak{M}(M)$ and $\mathfrak{M}(M')$ the resulting matrices $T^{(\ell)}$ and $T'^{(\ell)}$ will be the same at each $\Pi^{(\ell)}$

Consider all pairs $(\lceil q_i \rceil, \lceil q'_j \rceil)$ such that $\Theta(\lceil q_j \rceil) = \lceil q'_j \rceil$.

In $\Pi^{(0)}$:

$$\lceil q_i \rceil \in B_1^{(0)} \quad \text{iff} \quad \lceil q'_j \rceil \in B_1'^{(0)} \quad , \text{ by (1)}$$

$$\forall\, a \in \Sigma,\ \lceil \delta (\lceil q_i \rceil, a) \rceil \in B_1^{(0)} \quad \text{iff}$$

$$\lceil \delta' (\lceil q'_j \rceil, a) \rceil \in B_1'^{(0)} \quad , \text{ by (4)}$$

Hence the equality, up to a permutation of lines, of block $B_1^{(0)}$ to $B_1'^{(0)}$, and of $B_2^{(0)}$, to $B_2'^{(0)}$. The sort routine will change this to strict equality, and thus $T^{(0)} = T'^{(0)}$

Assume that in $\Pi^{(k)}$:

$$\lceil q_i \rceil \in B_r^{(k)} \quad \text{iff} \quad \lceil q'_j \rceil \in B_r'^{(k)}$$

$$\forall a \in \Sigma, \lceil \delta (\lceil q_i \rceil, a) \rceil \in B_{s_a}^{(k)} \quad \text{iff}$$

$$\lceil \delta' (\lceil q'_j \rceil, a) \rceil \in B_{s_a}'^{(k)}$$

and consequently, after applying sort, $T^{(k)} = T'^{(k)}$.

In $\Pi^{(k+1)}$; the application of renum will give:

5. $\lceil q_i \rceil \in B_t^{(k+1)}$ iff $\lceil q'_j \rceil \in B_t'^{(k+1)}$, by the induction hypothesis

6. $\forall a \in \Sigma, \lceil \delta (\lceil q_i \rceil, a) \rceil \in B_{n_a}^{(k+1)}$ iff

$\lceil \delta' (\lceil q'_j \rceil, a) \rceil \in B_{n_a}'^{(k+1)}$, by (3) and (5)

To understand (6), note that (5) applies in $\Pi^{(k+1)}$ to all states q_a, q'_b such that $\Theta (\lceil q_a \rceil) = \lceil q'_b \rceil$. But by (3), Θ holds

in particular between all $\lceil\delta(\lceil q_i \rfloor, a)\rceil$ and $\lceil\delta'(\lceil q'_j \rfloor, a)\rceil$, $\forall\, a \in \Sigma$.

So again all blocks are equal up to a permutation of lines, which is performed by sort. We conclude that $T^{(k+1)} = T'^{(k+1)}$, completing the proof.

We see that $\mathfrak{M}(M)$ and $\mathfrak{M}(M')$ being isomorphic implies that, after applying minim:

a. $T = T'$;

b. the initial state will be the same (see (2));

c. the final states will be the same (see (1)) ;

since the foregoing argument shows that states between which θ holds will always be in blocks with the same subscript, and at the end of the process the final subscripts are used to rename the states.

On the other hand if two fsa M_1 and M_2 satisfy a, b, c they certainly are the same fsa.

Until now we considered the application of minim to minimum state fsa. However we note that if $q_{i_1}, \ldots, q_{i_p} \in \lceil q_i \rfloor$ they will always stay in the same block at each partition $\Pi^{(\ell)}$ and they will correspond to identical lines in $T^{(\ell)}$ (otherwise the minimization process will put them in different blocks and they would never be put in the same state in the minimum state fsa); so they can be treated as a single state $\lceil q_i \rfloor$.

This means that minin can be applied to any fsa, minimal or not.

The following summarizes the above discussion:

Theorem: two fsa M and M' are equivalent iff after applying minim to M and M' assertions a, b, and c hold.

Corollary: minim provides a canonical form to all fsa which accept the same regular set.

5. IMPLEMENTATION

A program to implement the algorithm is developed from its description in a straightforward way.

It is easy to check that, if n is the number of states in the given fsa, both init and renum will take O(n) time.

Procedure sort uses the shell sort method [5], which is known to take O(n log n) time.

The DO WHILE loop in the worst case will be executed n-2 times, which corresponds to n-2 refinements. So the loop will total (including sort and renum) $O((n-2)(n + n \log n)) \approx O(n^2 \log n)$, this being the dominating time requirement of the algorithm.

Letting t = ## Σ, storage requirements include the n x t transition matrix, a bit-string of length n having non-zero elements

in the positions corresponding to the subscripts of the final states, a work-matrix T with n lines and t + 1 columns (the additional column indicates in what block is the state whose subscript corresponds to a given line); an n auxiliary array representing the current order of the states, and the $\ell \times t$, $\ell \leq n$ transition matrix of the minimal fsa.

The sort procedure does not actually exchange the lines of the work-matrix. We preferred the usual indirect sorting technique using the auxiliary array.

The total storage is therefore $n.t + n' + n.(t + 1) + n + \ell.t$, where n' is the number of words required to store a bit-string of lenght n.

Clearly the storage requirements are linear on the number of states.

APPENDIX I

PROGRAM LISTING

```
TEST:PROC OPTIONS(MAIN);
     DCL (M,N)BIN FIXED;
     GET LIST(M,N);
     BEGIN;
          DCL (T(M,N),P) BIN FIXED;
          DCL F(M) BIT(1);
          DCL SFS BIN FIXED;
          DCL (IS,SIS) BIN FIXED;
          DCL Z(M,N) BIN FIXED;
          DCL (I,J) BIN FIXED;
          GET LIST(T,F,IS) ;
          CALL MINIM(T,F,IS,M,N,Z,P,SIS,SFS);
          PUT SKIP EDIT('#STATES = ',P,'INITIAL STATE = ',SIS,
                        '#FINAL STATES = ',SFS)
                       (A(10),F(3),X(5),A(16),F(3),X(5),A(16),F(3));
          PUT SKIP(2) EDIT('TRANSITION MATRIX = ') (A(20));
          PUT SKIP EDIT(((Z(I,J) DO J=1 TO N) DO I=1 TO P))
                       (COLUMN(1),(N)F(4));
     END;
MINIM:PROC(T,F,IS,M,N,Z,P,SIS,SFS);
     DCL (T(*,*),P) BIN FIXED;
     DCL F(*) BIT(1);
     DCL SFS BIN FIXED;
     DCL(IS,SIS) BIN FIXED;
     DCL Z(*,*) BIN FIXED;
     DCL A(M,N+1) BIN FIXED;
     DCL B(M) BIN FIXED;
     DCL (L,K) BIN FIXED;
     DCL CHANGE BIT(1) INIT('1'B);
     DCL LESS ENTRY(BIN FIXED,BIN FIXED) RETURNS(BIT(1));
     CALL INIT ;
     DO WHILE(CHANGE);
          CHANGE='0'B;
          L=1;
          DO WHILE(L < M);
               DO K=L BY 1 WHILE(A(B(L),1) = A(B(K),1));
               END;
               K=K-1;
               IF K > L
               THEN CALL SORT;
               L=K+1;
          END;
          CALL RENUM;
     END;
     CALL AUTMIN;
 /*AUXILIARY PROCEDURES */
     INIT:PROC ;
          DCL I BIN FIXED;
          DCL J BIN FIXED INIT(0B);
          DO I=1 TO M;
               IF F(I)
               THEN DO;
                    J=J+1;
                    A(I,1)=1;
                    B(J)=I;
                    END;
```

```
      END;
      DO I=1 BY 1 WHILE(J<M);
           IF ¬ F(I)
           THEN DO;
                J=J+1;
                A(I,1)=2;
                B(J)=I;
                END;
      END;
      DO I=1 TO M;
           DO J=1 TO N;
                A(I,J+1)=A(T(I,J),1);
           END ;
      END;
 END INIT;
 RENUM:PROC;
      DCL (I,J) BIN FIXED;
      DCL K BIN FIXED INIT(1B) ;
      DCL TEMP BIN FIXED INIT(1B) ;
      A(B(1),1)=1;
      DO I=2 TO M;
           IF TEMP=A(B(I),1) & LESS(B(I-1)) | TEMP<A(B(I),1)
           THEN K=K+1;
           TEMP=A(B(I),1);
           A(B(I),1)=K;
      END;
      IF K > TEMP
      THEN CHANGE='1'B;
      DO I=1 TO M;
           DO J=1 TO N;
                A(I,J+1)=A(T(I,J),1);
           END;
      END;
END RENUM;
SORT:PROC;
     DCL (TEMP1,I,J) BIN FIXED;
     DCL D BIN FIXED INIT(1B);
     DO WHILE(D <= K-L+1);
          D=2*D;
     END;
     DO WHILE('1'B);
          D=(D-1)/2;
          IF D=0
          THEN RETURN;
          DO I=L TO K-D;
               J=I;
             R:DO;
               IF LESS(B(J+D),B(J))
               THEN DO;
                    TEMP1=B(J);
                    B(J)=B(J+D);
                    B(J+D)=TEMP1;
                    J=J-D;
                    END;
```

```
                    ELSE GO TO X;
                    IF J >= L
                    THEN GO TO R;
                    END R;
               X: ;
               END;
          END;
     END SORT;
     LESS:PROC(I,J) RETURNS(BIT(1));
          DCL (I,J) BIN FIXED;
          DCL K BIN FIXED;
          DO K=2 TO N+1;
               IF A(I,K) < A(J,K)
               THEN RETURN('1'B);
               IF A(I,K) > A(J,K)
               THEN RETURN('0'B);
          END;
          RETURN('0'B);
     END LESS ;
     AUTMIN:PROC;
          DCL (I,J) BIN FIXED;
          P=1;
          DO J=2 TO N+1;
               Z(1,J-1)=A(B(1),J);
          END;
          DO I=2 TO M;
               IF A(B(I-1),1) < A(B(I),1)
               THEN DO;
                    P=P+1;
                    DO J=2 TO N+1;
                         Z(P,J-1)=A(B(I),J);
                    END;
                    END;
          END;
          SIS=A(IS,1);
          SFS=1;
          DO I=2 TO M WHILE(F(B(I)));
               IF A(B(I),1) > A(B(I-1),1)
               THEN SFS=SFS+1;
          END;
      END AUTMIN;
END MINIM;
END TEST  ;
```

APPENDIX II

SAMPLE PROBLEM

We give the input and output corresponding to the fsa M of section 3.

a. input

```
9     3
2     2     5
1     4     4
2     2     5
3     2     2
6     4     3
8     9     6
6     2     8
4     4     7
7     9     7
'1'B  '0'B  '1'B  '0'B  '1'B '0'B  '1'B  '1'B  '0'B
5
```

b. output

#STATES = 5 INITIAL STATE = 1 #FINAL STATES= 2

TRANSITION MATRIX =

4	5	2
5	5	1
1	3	1
2	3	4
2	5	5

REFERENCES

[1] HOPCROFT, J and ULLMAN, D. - "Formal Languages and Their Relation to Automata" - Addison-Wesley, 1969.

[2] HARRISON, M. - "Introduction to Switching and Automata Theory" McGraw-Hill, 1965.

[3] MILLAN, M. and CARVALHO, R. - "On the Construction and Minimization of Finite State Automata" - technical report - PUC, 6/1971.

[4] BIRKHOFF, G. and BARTEE, T. - "Modern Applied Algebra" McGraw-Hill, 1970.

[5] BERZTISS, A. - "Data Structures - Theory and Practice" Academic Press, 1971.

[6] CORNEIL, D. - "Graph Isomorphism" - Technical report - University of Toronto, 18/1970.