Prof. L. LIVOVSCHI, D. Sc.
University of Bucharest
D. SOMNEA, D. Eng.
Department of Economic Cybernetics
Academy of Economic Studies

Generators of FORTRAN Source-Codes for Determining Some Graph Homomorphisms

This work describes two programmes for computer determination of the oriented graph homomorphisms which preserve the vertexes succession. This problem arises in synthetising finite automaton theory, with the help of some elementary automata with complex structures, as for instance the shift registers.

INTRODUCTION

The FORTRAN source codes generating programmes, which are used to obtain programmes for determining some graph homomorphisms find their applicability in the automation of the process of obtaining homomorphisms of graphs G in graphs GE, irrespective of their structures.

Graphs G and GE are characteristic graphs of automata which are to be synthetised by means of elementary automata with complex structures.

The generators are written in ASSIRIS language. The methods of obtaining homomorphisms of graphs G in GE lead previously the authors to the necessity of working out and codifying some algorithms of detection of graph circuits.

The old approach manner did not integrally carry out the synthesis process of finite automata, represented as G graphs with the help of elementary automata, carried out and described also under the form of characteristic graphs GE. The synthesis must be achieved in several passes through the algorithm of obtaining partial homomorphisms of graphs G in GE, as graphs G were decomposed in circuits and paths of various lengths.

The new approach used by the authors in this paper cancels out this drawback, by definitely solving the dependence problem of the source-code of the graphs G and GE structure.

An oriented graph and two of its vertexes, n_1 and n_2, connected together by an arc with its origin in n_1 and its extremity in n_2, being given, we shall say that those knots are in a succession relation, denoted by $n_1 \rightarrow n_2$.

Two graphs, G and GE, being given, an application φ of GE knots set in the set of GE knots, is called succession homomorphism if:

$$n_1 \rightarrow n_2 \rightarrow \varphi(\eta_1) \rightarrow \varphi(n_2) \qquad (1)$$

If φ exists, we can easily point out [1] that an elementary automation having graph GE can enter the composition of an automaton having graph G.

In [1] and [2] the problem of determining the succession homomorphisms is solved in two stages: in a first stage, the succession homomorphisms of graph G circuits in those of graph GE are determined, and in a second state those homomorphisms are made complete up to the homomorphisms of graph G in graph GE. In [3,4 and 5] various determining programmes in the described mode are presented, among which also two conversational versions.

A programme, principally different from the mentioned ones is described further on.

PROGRAMME 1

So as to state precisely, we consider graphs G and GE from figures 1 and 2. The programme is composed of a main programme P and three subroutines COMP1, COMP2 and COMP3.

In P we construct a cycle composed of a number of 18 cycles, corresponding to the 18 vertexes of G and having the structure from Fig. 3. We can assign to each cycling variable $K1, \ldots, K18$ the values 1 — 8, representing the numbers of GE vertexes. Cycles 2—18 contain each a branching condition

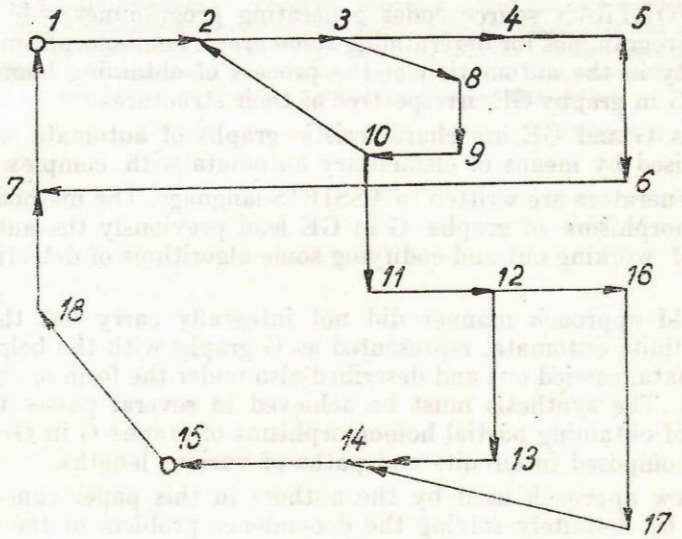

Fig. 1

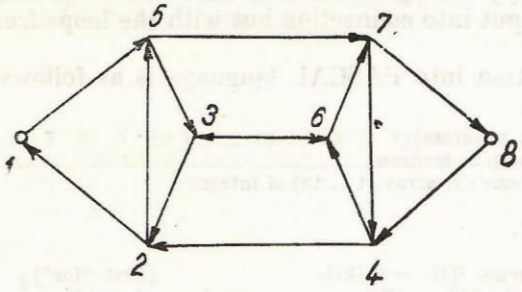

Fig. 2

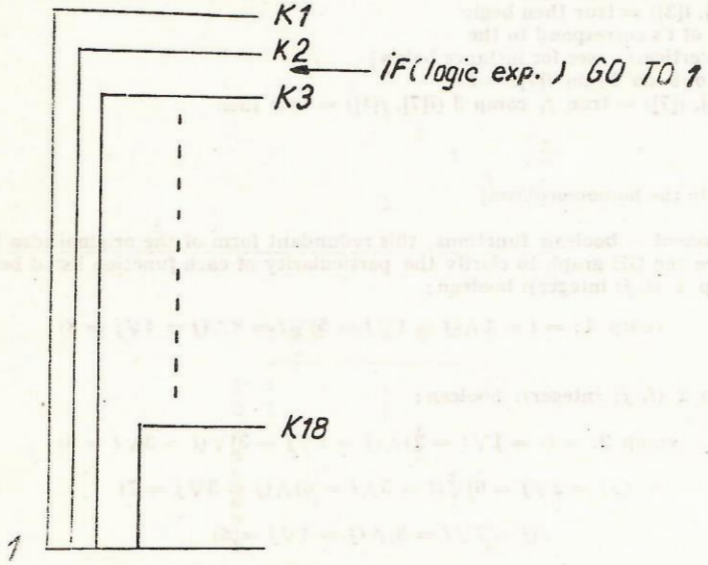

Fig. 3

of the following type: "If the values assigned to variables $K1$, $K2$, ..., $K\check{r}$ fulfill the condition that, to all arcs in G of the form $I \rightarrow \alpha$ or $\alpha \rightarrow I$, where $\alpha \in \{1,2, ..., I-1\}$, arcs in GE are coresponding, then proceed to the next cycle; otherwise, take up again the actual cycle, for the next value of K_i."

The verification of the above condition is effected in three subroutines COMP1, COMP2 and COMP3, which include each an instruction of logical assigning of the form

$$\text{COMP}i = < \textit{logical expression} >$$

where $<$ *logical expression*$>$ is the disjunction of some conditions of the form

I.EQ.1.AND. (J.EQ.1.OR.J.EQ.5)...

and which (see Fig. 2) describe the arcs of GE, divergent from the vertex 1, ..., 8. We further notice that the structure of such a programme can be

slightly simplified by taking into account that the loop from vertexes 1 and 15 of G cannot be put into connection but with the loops from vertexes 1 and 8 of GE.

The transcription into PASCAL language is as follows:

```
{P component — main program}
var comp 1, comp 2, comp 3: boolean;
i1: array [1 ... 2] of integer; i: array [1 ... 18] of integer;
{initial assignment}
i1 [1]: = 1; i1 [2]: = 8;
{nest of 18 for's}
for k1: = 1 to 2 do begin i[1]: = i1 [k1];                    {first "for"}
for k2: = 1 to 8 do begin i[2]: = k2;                  {second "for"}
if comp 1 (i [1], i[2]) = true then begin
for k3: = 1 to 8 do begin i[3]: = k3;
if comp 2 (i[2], i[3]) = true then begin
{index values of i's correspond to the
graph knots (vertices) ... see for instance below}
for k7: = 1 to 8 do begin i[7]: = k7;
if comp 2 (i[6], i[7]) = true ∧ comp 3 (i[7], i[1]) = true then
end
end
end
write (i); {write the homomorphism}
end.
```

{s component — boolean functions; this redundant form of the original idea is easy to recognized; see the GE graph to clarify the particularity of each function listed below}

```
function comp 1 (i, j: integer): boolean;
```

$$\text{comp 1}: = i = 1 \wedge (j = 1 \vee j = 5) \vee i = 8 \wedge (j = 4 \vee j = 8)$$

```
end
function comp 2 (i, j: integer): boolean;
```

$$\text{comp 2}: = (i = 1 \vee i = 2) \wedge (j = 1 \vee j = 5) \vee (i = 3 \vee i = 4)$$

$$\wedge (j = 2 \vee j = 6) \vee (i = 5 \wedge i = 6) \wedge (j = 3 \vee j = 7)$$

$$\vee (i = 7 \vee i = 8) \wedge (j = 4 \vee j = 8)$$

```
end
function comp 3 (i, j: integer): boolean;
```

$$\text{comp 3}: = j = 1 \wedge (i = 1 \vee i = 2) \vee j = 8 \wedge (i = 7 \vee i = 8)$$

```
end.
```

PROGRAMME 2

An essential drawback of the programmes in the proceeding form consists in the fact that their structures integrally depend on the concrete form of graphs G and GE.

By looking at component P in programme 1, we establish easily all recurrence relations referring to the composing of the instructions from the nest of Do' loops. Those recurences lead us to the idea of automatically generating source codes. The programme obtained with such generators is called programme 2.

The source-codes generating programmes were written in ASSIRIS. The input data of those generators are the description tables of the characteristic states of graphs G, GE. In table 1, as an example, we present the automaton with 18 states and in table 2, one with 8 states.

Table 1

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	1	1																
2			1					1										
3				1														
4					1													
5						1												
6					1		1											
7	1																	
8									1									
9										1								
10		1									1							
11												1						
12													1			1		
13														1				
14															1			
15																1		2
16																	1	
17														1				
18							1											

Legend

1 = there exists elementary arc

Table 2

	1	2	3	4	5	6	7	8
1	1				1			
2	1				1			
3		1				1		
4		1				1		
5			1				1	
6			1				1	
7				1				1
8				1				1

The invariant portions of source-code of components P and S of programme 2 are concatenated to the variable portions of the respective source-codes obtained with the help of the generator, catalogued together in a source library, and then linkage edited and carried out. Homomorphisms of graphs G in GE are finally obtained through effecting programme 2.

In annex 1, the source-codes generated by means of generators are reproduced, for the graphs example presented in the tables above. The durations of generating source-codes for components P and S were 2.344 [s] and 1.222 [s], central processing unit-time.

The obtaining durations of the homomorphisms, either by programme 1 or by programme 2 were of about 13 [s] central-processing-unit-time Subprogramme S replaces subroutines COMP1, COMP2, COMP3 in programme 2.

Some details concerning the generator construction must be given.

Both generators work on the principle of positional composition of a source-instruction in a buffer zone. The possibility of prolonging an instruction code on at the most 19 continuation lines was taken into account.

The generator of the variable part of component P can be described as follows:

```
MAIN PROCEDURE
1: Definition of arrays ISGA (50, 50), BUFFER (400) and DOUBLET (100);
   Definition of a common data zone ISGA, BUFFER, DOUBLET and
   other working variables;
   /* ISGA — array with the states of the automaton described by graph G
      BUFFER — here is formed the source instruction
      DOUBLET — working array wherein the succession relations of the
                knots are collected */
2: Data readings NGA, ISGA; /* NGA = number of states of G */
3: IA = 2; /* the generation of the variable part begins from instruction
              B13 — see annex 1 */
4: DO WHILE (IA < = NGA);
      Initialization ID, DOUBLET, BUFFER;
      DO IL = 1,IA;
         IF ISGA(IL, IA) = 0 THEN
            IF IL = IA THEN DO;
                            ID = ID + 1;
                            DOUBLET(ID) = IL;
                            ID = ID + 1;
                            DOUBLET(ID) = IA;
                            END;
                       ELSE;
      END;
      DO JL = 1, IA;
        IF ISGA(IA, JL) = 0 THEN;
                        ELSE DO;
                             ID = ID + 1;
                             DOUBLET(ID) = IA;
                             ID = ID + 1;
                             DOUBLET(ID) = JL;
                             END;
      END;
      CALL PARSE;
      CALL WRITE;
END; /* for DO WHILE */
6. CALL FINISH;
END MAIN;
PARSE PROCEDURE; /* written in ASSIRIS, extracts data from the
                    common zone and edits FORTRAN instruction */
```

```
1: Generating the model of the source-instruction;
2: Addresses the elements of DOUBLET converts them in an unpacked
   form and inserts them in the model body;
END PARSE;
WRITE PROCEDURE; /* written in FORTRAN */
  1: Writes source-lines of the source-instruction;
END WRITE;
FINISH PROCEDURE; /* Writen in FORTRAN */
  1: Generates a source-instruction of the type of that with number
     B82 in annex 1;
END FINISH;
          S component generator is composed as follows:
    MAIN PROCEDURE;
1: Definition of arrays ISGAE (40, 40), BUFFER (340), J (40).
     /* ISGAE — the same role as that of array ISGA in the previous
                 generator, but for graph GE */
2: Definition of the common zone ISGAE, BUFFER, J, NGAE, other
   working variables;
3: Data reading NGAE, ISGAE; /* NGAE — number of states of GE */
4: IL = 1;
5: DO WHILE (IL < NGAE);
       IL = 0;
       initialization BUFFER;
       DO JL = 1, NGAE;
          IF ISGAE(IL, JL) = 0 THEN DO;
                                    IJ = IJ + 1;
                                    J(IJ) = JL;
                                    END;
                               ELSE;
       END;
       CALL PARSE1;
       CALL WRITE;
   END; /* for DO WHILE */
6: CALL PARSE2;
7: CALL WRITE;
   END MAIN;
   PARSE1 PROCEDURE; /* uses data from the common zone, written in
                        ASSIRIS */
1: generates all source-instructions of the type
          L(JL) = I.EQ.JL ... for JL = 1, NGAE
   END PARSE1;
   PARSE2 PROCEDURE; /* idem PARSE1 /*
1  Generates instruction LINK = L(1).OR.L(2).OR .... OR.L (NGAE)
   END PARSE2;
   WRITE PROCEDURE; /* the same role as with the similar procedure in
                       the other case */
1: Edits FORTRAN source-lines;
   END WRITE;
```

With respect to the time performances of programmes 1 or 2 — for obtaining homomorphisms — we emphasize that the problemse lected as example in this paper may be considered as a very complex one, and the central-processing-unit-time durations are reasonable. We must take into account here that the above affirmation does not rely upon a comparison with any similar algorithm in the literature, not met with by the authors. It depends on their experience as regards the obtained duration for the same example of the developed quick algorithm [2] — [5], an algorithm comparable, as speed, with other algorithms in the literature.

G graphs having 50 states make up very complex finite automata, and GE graphs with 40 states, based on shift registers of those lengths, make also up more isolated cases in practice.

Limits of the generators:

— G graph multiplicity order is limited by the number of continuation lines of a FORTRAN enunciation of the type

IF (LINK(L( ), I( )). AND. LINK(I( ), I( )) ... from component P;

— partition dimension, if it is smaller than 36K, taking into consideration the present dimension of the data module;

— GE graph multiplicity order is limited by the line number of an enunciation of type

L( ) = I.EQ. .AND.(J.EQ. ...OR.J.EQ ...)

The shift registers lead to characteristic graphs with a multiplicity order equal to two.

We recommend the use of the algorithm presented in [2] — [5], for graphs whose multiplicity order is greater than two, when the partition dimension problem is executed and instead of it the satisfying of the time performance is required.

The programms presented in [2] — [5] cannot be generated according to a technique similar to that in the present paper.

REFERENCES

1. LIVOVSCHI LEON: *Realizarea unui automat cu ajutorul registrelor de deplasare* (The Achieving of an Automaton by Means of Shift Registers). Studii și cercetări matematice-23, 4, 1971.
2. LIVOVSCHI LEON, SOMNEA DAN: *Programe de calcul pentru realizarea unor automate cu ajutorul registrelor* (Computation Programmes for Achieving Some Automata with the Help of Registers). Paper presented to the Seminar organized by LATEMAS, University of Bucharest, Aug. 30 — Sept. 2, 1973, Brașov.
3. LIVOVSCHI LEON, SOMNEA, DAN: *Programe de calcul pentru realizarea unor automate cu ajutorul registrelor de deplasare.* (Computation Programmes for Achieving Some Automata with the Help of Shift Registers). Preprint. Seminar of systems theory and applications. Oficiul de informare și documentare în științele sociale și politice, București, 1974.
4. LIVOVSCHI LEON, SOMNEA, DAN: *Finite Automata Synthesis with Conversational Terminal.* Paper presented at the 3-rd World Congress of Cybernetics and Systems, București, 1975.
5. LIVOVSCHI LEON, SOMNEA, DAN: *Finite Automata Synthesis with Conversational Terminal. Preprint.* The 3-rd World Congress of Cybernetics and Systems, Springer Verlag, vol III. pp. 59—71, Section V, 1977.

```
      1     * FETCHS DV:MD1,LN:GRAPH,GN:1,VN:1,FN:GENGA
B     2     * DEFINE FILE *1=1,*2=3
B     3           LOGICAL LINK
B     4           DIMENSION ISGA(50,50),ISGAE(40,40),I(50)
B     5           READ(1,101) NGA,NGAE
B     6       101 FORMAT (2I10)
B     7           READ (1,102) ((ISGAE(IA,IB),IB=1,NGAE),IA=1,NGAE)
B     8           READ (1,102) ((ISGA(IA,IB),IB=1,NGA),IA=1,NGA)
B     9       102 FORMAT(80I1)
B    10           DO 1 K1=1,NGA
B    11           I( 1) = K1
B    12           IF(LINK(I(1),I(1))) GOTO 2
B    13           GO TO 1
B    14         2 DO 1 K02 = 1,NGA
B    15           I( 2) = K02
B    16           IF(  LINK(I( 1),I( 2)))GO TO  3
B    17           GO TO 1
B    18         3 DO 1 K03 = 1,NGA
B    19           I( 3) = K03
B    20           IF(  LINK(I( 2),I( 3)))GO TO  4
B    21           GO TO 1
B    22         4 DO 1 K04 = 1,NGA
B    23           I( 4) = K04
B    24           IF(  LINK(I( 3),I( 4)))GO TO  5
B    25           GO TO 1
B    26         5 DO 1 K05 = 1,NGA
B    27           I( 5) = K05
B    28           IF(  LINK(I( 4),I( 5)))GO TO  6
B    29           GO TO 1
B    30         6 DO 1 K06 = 1,NGA
B    31           I( 6) = K06
B    32           IF(  LINK(I( 5),I( 6)).AND.LINK(I( 6),I( 5)))GO TO  7
B    33           GO TO 1
B    34         7 DO 1 K07 = 1,NGA
B    35           I( 7) = K07
B    36           IF(  LINK(I( 6),I( 7)).AND.LINK(I( 7),I( 1)))GO TO  8
B    37           GO TO 1
B    38         8 DO 1 K08 = 1,NGA
B    39           I( 8) = K08
B    40           IF(  LINK(I( 3),I( 8)))GO TO  9
B    41           GO TO 1
B    42         9 DO 1 K09 = 1,NGA
B    43           I( 9) = K09
B    44           IF(  LINK(I( 8),I( 9)))GO TO 10
B    45           GO TO 1
B    46        10 DO 1 K10 = 1,NGA
B    47           I(10) = K10
B    48           IF(  LINK(I( 9),I(10)).AND.LINK(I(10),I( 2)))GO TO 11
B    49            GO TO 1
```

```
B      50          11 DO 1 K11 = 1,NGA
B      51             I(11) = K11
B      52             IF( LINK(I(10),I(11)))GO TO 12
B      53              GO TO 1
B      54          12 DO 1 K12 = 1,NGA
B      55             I(12) = K12
B      56             IF( LINK(I(11),I(12)))GO TO 13
B      57              GO TO 1
B      58          13 DO 1 K13 = 1,NGA
B      59             I(13) = K13
B      60             IF( LINK(I(12),I(13)))GO TO 14
B      61              GO TO 1
B      62          14 DO 1 K14 = 1,NGA
B      63             I(14) = K14
B      64             IF( LINK(I(13),I(14)))GO TO 15
B      65              GO TO 1
B      66          15 DO 1 K15 = 1,NGA
B      67             I(15) = K15
B      68             IF( LINK(I(14),I(15)).AND.LINK(I(15),I(15)))GO TO 16
B      69              GO TO 1
B      70          16 DO 1 K16 = 1,NGA
B      71             I(16) = K16
B      72             IF( LINK(I(12),I(16)))GO TO 17
B      73              GO TO 1
B      74          17 DO 1 K17 = 1,NGA
B      75             I(17) = K17
B      76             IF( LINK(I(16),I(17)).AND.LINK(I(17),I(14)))GO TO 18
B      77              GO TO 1
B      78          18 DO 1 K18 = 1,NGA
B      79             I(18) = K18
B      80             IF( LINK(I(15),I(18)).AND.LINK(I(18),I( 7)))GO TO 19
B      81              GO TO 1
B      82          19  WRITE(3,999)(I(IE),IE=1,NGA)
B      83         999  FORMAT(1H ,5I2)
B      84           1  CONTINUE
B      85              STOP
B      86              END

       1      * FETCHS DV:MD1,LN:GRAPH,GN:1,VN:1,FN:GENGAE
B      2               LOGICAL FUNCTION LINK(I,J)
B      3               LOGICAL L(40)
B      4            L( 1) = I.EQ. 1.AND.(J.EQ. 1.OR.J.EQ. 5)
B      5            L( 2) = I.EQ. 2.AND.(J.EQ. 1.OR.J.EQ. 5)
B      6            L( 3) = I.EQ. 3.AND.(J.EQ. 2.OR.J.EQ. 6)
B      7            L( 4) = I.EQ. 4.AND.(J.EQ. 2.OR.J.EQ. 6)
B      8            L( 5) = I.EQ. 5.AND.(J.EQ. 3.OR.J.EQ. 7)
B      9            L( 6) = I.EQ. 6.AND.(J.EQ. 3.OR.J.EQ. 7)
B      10           L( 7) = I.EQ. 7.AND.(J.EQ. 4.OR.J.EQ. 8)
B      11           L( 8) = I.EQ. 8.AND.(J.EQ. 4.OR.J.EQ. 8)
B      12           LINK = L( 1).OR.L( 2).OR.L( 3).OR.L( 4).OR.L( 5).OR.L( 6).OR
B      13          *       L( 7).OR.L( 8)
B      14              RETURN
B      15              END
```

GENERATEUR DE CODES SOURCES FORTRAN POUR DETERMINER CERTAINS HOMOMORPHISMES DE GRAPHES

On décrit deux programmes pour déterminer, à l'aide de l'ordinateur, les homomorphismes des graphes orientés qui gardent la succession des sommets. Ce problème intervient dans la synthèse des automates finis au moyen des automates élémentaires à structure complexe, tels que les registres à décalage.

ГЕНЕРАТОРЫ КОДОВ ИСТОЧНИКОВ ФОТРАН ДЛЯ ОПРЕДЕЛЕНИЯ ГОМОМОРФИЧЕСКИХ ГРАФОВ

В работе описываются две программы для определения с ЭВМ направленных гомоморфисмов графов сохраняющих последовательность вершин. Этот вопрос возникает при синтезе конечных автоматов с помощью отдельных элементарных автоматов комплексной структуры, как например регистров перемещения.

19.X.1977