Tomus 3. Fasciculus 1.

ACTA CYBERNETICA

FORUM CENTRALE PUBLICATIONUM CYBERNETICARUM HUNGARICUM

Szeged, 1976

Curat: Universitas Szegediensis de Attila József nominata

3. kötet 1. füzet

ACTA CYBERNETICA

A HAZAI KIBERNETIKAI KUTATÁSOK KÖZPONTI PUBLIKÁCIÓS FÓRUMA

Szeged, 1976. december
A Szegedi József Attila Tudományegyetem gondozásában

LÁSZLÓ KALMÁR
1905—1976

The Editors of the Acta Cybernetica announce with deep sorrow that the Acta's Editor-in-Chief, Professor László Kalmár, ordinary member of the Hungarian Academy of Sciences, has unexpectedly deceased on 2nd August 1976.

The sudden death of Professor Kalmár, overcoming him in his full creative energy, is an irreplaceable loss of the mathematical and cybernetical investigations in Hungary. He was an outstanding researcher of mathematical logic and computer science; the main areas he dealt with were the decision problem, the topics of completeness and incompleteness theorems in logic, the theory of programming languages and the planning of formula-controlled computers. In addition to his achievements as an eminent scientist, he exerted a fatigueless effort through personal connections, both as a teacher of young colleagues and as the main organizer of the labour concerning the theory and employment of computers in Hungary.

In our country, every specialist of computer science and a large circle of mathematicians lost in him an inspiring educator and a master of admirably comprehensive knowledge.

On graphs satisfying some conditions for cycles, I.

By A. ÁDÁM

To the memory of my friend Professor Andor Kertész

Introduction

The aim of the present paper is to give a structural description of the finite directed graphs satisfying the conditions that

to any edge e the number of cycles containing e is 1 or 2, and

there exists a vertex contained in every cycle of the graph.

It is obvious that a graph fulfilling these requirements can have at most one cut vertex.

We rely upon some results of the earlier paper [1]. In §§ 2—3 we give some constructions and prove that they produce the graphs that possess the properties mentioned above and having no cut vertex. The description is extended in § 4 to graphs in which a cut vertex occurs.

§ 1.

By a graph, we mean always a finite directed graph with at least two vertices. We suppose that it is connected and contains neither loops nor parallel edges with the same orientation.

It is assumed that §§ 2—3 of the preceding paper [1] are known to the reader. The terminology introduced in § 2 of [1] is mostly further applied (but the notations $\mathfrak{A}_k(G)$ and $\mathfrak{A}(C)$ do not occur in this paper). We say that e.g. $Z(A) \geqq 1$ is *universally*[1] satisfied in G if it is true for every vertex A of the graph G. In accordance with [1], we denote by C_1 the class of connected directed finite graphs in which $Z(A) \leqq 2$ and $Z(e) \geqq 1$ are universally valid. Construction I, Theorems 1 and 2 of [1] will be referred to as Construction I*, Theorems 1* and 2*, respectively.

The sum of the indegree and outdegree of a vertex A is called the *total degree* of A.

A vertex A of a graph G is called *pancyclic* if A is contained in each cycle of G.

[1] In [1] the word "identically" was applied for expressing the universal quantification.

Let us consider three conditions (imposed upon a graph G):

(α) $1 \leqq Z(e) \leqq 2$ is universally satisfied in G,

(β) G has a pancyclic vertex,

(γ) G has no cut vertex.

We define the class C_5 as the collection of finite directed graphs fulfilling (α) & (β) & (γ) and we denote by C_6 the set of finite directed graphs in which (α) & (β) is satisfied.[2] It is clear that $C_5 \subseteqq C_6$. The condition (α) implies the universal validity of $Z(A)>0$ in G.

The vertices of degree (1, 1) are called *simple* vertices. Let c be a path of positive length in the graph G, denote the vertices of c by $A_0, A_1, \ldots, A_n$ (as they follow in c) ($n \geqq 1$); c is called an *arc* (or more precisely, an (A_0, A_n)-arc) if its inner vertices $A_1, A_2, \ldots, A_{n-1}$ are simple vertices (in G).

§ 2.

We describe four constructions. In any construction, the arcs are supposed to have no edge and no inner vertex in common. The lengths of the arcs are arbitrary positive integers.

CONSTRUCTION I. Let $k(\geqq 4)$ be an even number. Take $k+1$ vertices $A, B_1, B_2, \ldots, B_k$ and the following $2k$ arcs:

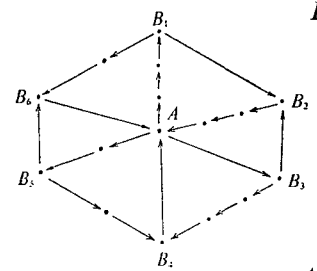


Fig. 1. A I-constructible graph ($k=6$)

an (A, B_i)-arc for each odd number i $(1 \leqq i \leqq k-1)$,
a (B_i, A)-arc for each even number i $(2 \leqq i \leqq k)$,
a (B_i, B_{i-1})-arc for each odd number i $(3 \leqq i \leqq k-1)$,
a (B_i, B_{i+1})-arc for each odd number i $(1 \leqq i \leqq k-1)$,
a (B_1, B_n)-arc.

(It is clear that, in a graph G resulted by Construction I, $A, B_1, B_2, \ldots, B_k$ and the inner vertices of the arcs are the vertices of G, and the edges of the arcs are the edges of G.)

CONSTRUCTION II/a. Let $k(\geqq 2)$ be an integer. Take the $k+1$ vertices $A, B_1, B_2, \ldots, B_k$ and the following $2k+1$ arcs:

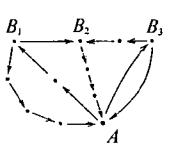


Fig. 2. A II/*a*-constructible graph ($k=3$)

an (A, B_1)-arc,
a (B_1, A)-arc,
an (A, B_i)-arc for each odd number i $(3 \leqq i \leqq k-1)$,
a (B_i, A)-arc for each even number i $(2 \leqq i \leqq k-1)$,
a (B_i, B_{i-1})-arc for each odd number i $(3 \leqq i \leqq k)$,
a (B_i, B_{i+1})-arc for each odd number i $(1 \leqq i \leqq k-1)$,
an (A, B_k)-arc,
a (B_k, A)-arc.

[2] We do not use the notations C_2, C_3, C_4 which occur in [1] but they are not referred to in this paper.

CONSTRUCTION II/b. Let $k(\geqq 2)$ be an integer. Take the $k+1$ vertices $A, B_1, B_2, \ldots, B_k$ and the following $2k+1$ arcs:

an (A, B_1)-arc,
a (B_1, A)-arc,
a (B_i, A)-arc for each odd number i $(3 \leqq i \leqq k-1)$,
an (A, B_i)-arc for each even number i $(2 \leqq i \leqq k-1)$,
a (B_i, B_{i-1})-arc for each even number i $(2 \leqq i \leqq k)$,
a (B_i, B_{i+1})-arc for each even number i $(2 \leqq i \leqq k-1)$,
an (A, B_k)-arc,
a (B_k, A)-arc.

CONSTRUCTION III. Take the vertices A, B, two (A, B)-arcs c_1, c_2 and two (B, A)-arcs c_3, c_4 such that $l_1+l_2 \geqq 3$ and $l_3+l_4 \geqq 3$ where l_j is the length of c_j (j can be 1, 2, 3, 4).

If a graph G can be built up by Construction I, then it is said that G is I-constructible. The II/a-constructible, II/b-constructible, III-constructible and I*-constructible graphs are meant analogously. G is said to be II-constructible if it is either II/a-constructible of II/b-constructible. A II/a-constructible graph is said to be II/a/o-constructible or II/a/e-constructible if it results with an odd or an even k, respectively (by Construction II/a). The II/b/o-constructible and II/b/e-constructible graphs are understood in a similar manner.

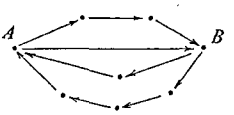


Fig. 3.
A III-constructible graph

Proposition 1. *A graph is II/a/e-constructible if and only if it is II/b/e-constructible.*

Proof. Let k be even. If the notation of the vertices $B_1, B_2, \ldots, B_k$ is replaced by $B_k, B_{k-1}, \ldots, B_1$ (respectively), then the definitions of II/a/e-constructibility and II/b/e-constructibility are interchanged.

Proposition 2. *The sets of*
I-constructible graphs,*
I-constructible graphs,
II/a/o-constructible graphs,
II/a/e-constructible graphs,
II/b/o-constructible graphs and
III-constructible graphs
are pairwise disjoint.

Proof. It is clear that the total degree of a vertex of a I*-constructible graph is $\leqq 4$ and equality holds precisely in case of cut vertices. On the other hand, the total degree of the vertex A is $\geqq 4$ in case of any of the constructions described above, although A is not a cut vertex. (Indeed, the total degree of A is k for Construction I, $k+2$ for Constructions II/a and II/b, it is 4 for Construction III.) Therefore a I*-constructible graphs cannot belong to any other type mentioned in the proposition.

A III-constructible graph has two vertices (namely A and B) whose total degree is 4. If a graph is I-constructible or II-constructible, then all vertices $C(\neq A)$ of it have a total degree $\leqq 3$. Hence a III-constructible graph is neither I-constructible nor II-constructible.

Let G be a II-constructible graph. The (A, B_1)-arc and the (B_1, A)-arc connect the same vertices A and B (with opposite orientations). The lack of a pair of arcs of this nature in any I-constructible graph implies that G cannot be I-constructible.

To any graph G denote by $\tau(G)$ the pair (v, w) where v is the number of vertices of degree (2, 1) and w is the number of vertices having degree (1, 2). We have

$$\tau(G) = \left(\frac{k-1}{2}, \frac{k+1}{2}\right), \quad \tau(G) = \left(\frac{k}{2}, \frac{k}{2}\right) \quad \text{and} \quad \tau(G) = \left(\frac{k+1}{2}, \frac{k-1}{2}\right)$$

if G is II/a/o-constructible, II/a/e-constructible or II/b/o-constructible, respectively. Consequently, any graph is contained in at most one of these three types.

Proposition 3. *If a graph G is I-constructible or II-constructible or III-constructible, then $1 \leqq Z(e) \leqq 2$ holds for any edge e of G.*

Proof. Let G be I-constructible. Each cycle c of G can be characterized by the sequence of that vertices of G whose degree differs from (1, 1). In this manner, the sequences

(A, B_i, B_{i-1}) where $3 \leqq i \leqq k-1$ and i is odd,

(A, B_i, B_{i+1}) where $1 \leqq i \leqq k-1$ and i is odd,

(A, B_1, B_k)

characterize cycles in G, and it is obvious that all the cycles of G have thus been exhausted. This survey of cycles guarantees $1 \leqq Z(e) \leqq 2$.

If G is II/a-constructible, then the inference is similar, namely the cycles are determined by the sequences

(A, B_1)

(A, B_i, B_{i-1}) where $3 \leqq i \leqq k$ and i is odd,

(A, B_i, B_{i+1}) where $1 \leqq i \leqq k-1$ and i is odd,

(A, B_k).

When G is II/b/o-constructible, then the sequences determining the cycles of G are the following ones:

$$\left.\begin{array}{l}(A, B_i, B_{i-1}) \\ (A, B_i, B_{i+1}) \\ (A, B_k)\end{array}\right\} \text{ where } 2 \leqq i \leqq k-1 \text{ and } i \text{ is even.}$$

The II/b/e-constructible graphs do not require a further treatment (by Proposition 1).

It is evident that in any III-constructible graph there are precisely four cycles and $Z(e)=2$ is universally satisfied.

Proposition 4. *If a graph G is I-constructible or II-constructible or III-constructible, then $G \in C_5$.*

Proof. The universal validity of $1 \leqq Z(e) \leqq 2$ was stated in Proposition 3. It is clear from the constructions that G has no cut vertex and the vertex A (in any construction) is pancyclic.

§ 3.

Proposition 5. *Assume that one of the next five conditions* (a)—(e) *is true for the graph G:*

(a) *G is a cycle,*
(b) *G is I*-constructible, it has exactly two cycles and it has no cut vertex,*[3]
(c) *G is I-constructible,*
(d) *G is II-constructible,*
(e) *G is III-constructible.*

Choose two different vertices C, D in G. Take a new (C, D)-arc to G, denote the resulting graph by G. Suppose that either there is no edge from C to D (in G) or the new arc has at least two edges. Then G* satisfies one of the following three statements:*

(1) *G* fulfils one of* (b), (c), (d), (e),
(2) *G* has an edge e such that* $Z(e)>2$,
(3) *G* has no pancyclic vertex.*[4]

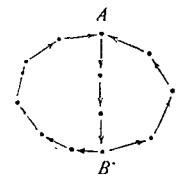


Fig. 4. A graph satisfying the condition (b) (occurring in Proposition 5 and Theorem 1)

Proposition 6. *Let* G_1, G_2 *be two graphs such that each of them fulfils one the requirements* (a)—(e) *exposed in Proposition* 5. *Let* A_i *be a pancyclic vertex in* G_i (*i is* 1 *or* 2). *Form the union G of* G_1 *and* G_2 *such that the vertices* A_1 *and* A_2 *are identified with each other (and this vertex is denoted by A). Choose a vertex* $C(\neq A_1)$ *in* G_1 *and a vertex* $D(\neq A_2)$ *in* G_2. *Take a new (C, D)-arc to G, denote the resulting graph by G*. Then G* satisfies one of the statements* (1), (2) *occurring in Proposition 5.*

Since the proofs of Propositions 5 and 6 are lengthy and of technical character, they will be given at the end of the paper as Appendix I and Appendix II, respectively.

Lemma. Let G′ be a subgraph of the graph G such that G′ has a cycle. If G′ has no pancyclic vertex, then the same holds for G.

Proof. Let A be an arbitrary vertex of G. If A belongs to G', then G' has a cycle a which does not contain A (since A is not pancyclic in G'). If A is not a vertex of G', then no cycle of G' can contain A. We have got that A is not pancyclic in G.

Proposition 7. *If* $G\in C_5$, *then one of the requirements* (a)—(e), *occurring in Proposition 5, is true for G.*

Proof. Denote by $\varkappa$ the number of cycles of G. We use induction on $\varkappa$.

If $\varkappa=1$, then (a) is true; if $\varkappa=2$, then (b) is valid (because of Theorem 2* and (γ)).

Consider the case when $\varkappa\geqq 3$. Let us select an edge e_0 such that $Z(e_0)$ is minimal in G. Delete e_0 and those vertices C and edges e which satisfy $Z(C)=0$ and $Z(e)=0$ (resp.) in the graph obtained by removing e_0. Denote the remaining graph by G'. G' exists since $Z(e_0)<3$. It is clear that $1\leqq Z(e)\leqq 2$ holds universally in G'. If a vertex A has been pancyclic in G, then A is (contained and) pancyclic in G', too.

---

[3] In other words: G has been formed by Construction I* from the tree with only one edge, such that $V'\neq 0$ (i.e. Step 3 has really been applied).

[4] The assertions (2) and (3) do not exclude each other.

Our next aim is to show that whenever a vertex C of G does not occur in G', then C is simple. Indeed, any cycle containing C contains also e_0, therefore (by $1 \leqq Z(e_0) \leqq 2$ in G) the indegree and outdegree of C may be 1 or 2. If e.g. the indegree of C is 2, then $Z(e_0)=2$ and $Z(e')=Z(e'')=1$ (where e' and e'' are the edges of G terminating at C), contradicting the minimality of $Z(e_0)$. Thus the indegree of C is 1, the outdegree of C is also 1 (by similar reason).

Consequently, G can be represented as an edge-disjoint union of G' and certain arcs $a_1, a_2, \ldots, a_t$ ($t \geqq 1$) such that the inner vertices of any arc a_i ($1 \leqq i \leqq t$) occur neither in G' nor in $a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_t$, furthermore, the beginning vertex and end vertex of any a_i belong to G'.

Define the graphs

$$G_0, G_1, G_2, \ldots, G_t \qquad (t \geqq 1)$$

successively such that $G_0=G'$ and G_i proceeds from G_{i-1} (where $1 \leqq i \leqq t$) by adding the edges and inner vertices of a_i. We have $G_t=G$. The further proof splits to two cases.

Case 1. G' has no cut vertex. Then, by the induction hypothesis, one of (a)—(e) is valid for $G'=G_0$. We are going to prove that the same holds also for $G_1, G_2, \ldots, G_t$. Suppose that i is the smallest subscript such that each of (a)—(e) is false for G_i ($1 \leqq i \leqq t$). By applying Proposition 5 for G_{i-1} and the arc a_i, we get then that either $Z(e) \geqq 3$ is satisfiable in G_i (thus in G, too) or G_i (hence, by the Lemma, also G) has no pancyclic vertex. Consequently, $G \notin C_5$, this contradicts the assumption.

Case 2. G' has a cut vertex. It is then obvious that the pancyclic vertex A (in G) is cut vertex of G', and G' does not possess any other pancyclic or cut vertex. Furthermore, there exists a number w ($0 \leqq w < t$) such that the (single) cut vertex of $G_0, G_1, G_2, \ldots, G_w$ is A but none of $G_{w+1}, G_{w+2}, \ldots, G_t$ has a cut vertex. Moreover, the number of blocks (separated by A) of G_i ($1 \leqq i \leqq t$) is either the same as the number of blocks of G_{i-1} or less by one, dependingly on the situation of a_i.

Since $G_0=G'$ satisfies (α), the induction hypothesis guarantees the validity of one of (a)—(e) for any block of G_0. Similarly to Case 1, we can show that the same holds for the blocks of each G_i (by applying Proposition 5 or Proposition 6 according as the addition of a_i does not or does diminish the number of blocks of G_{i-1}).

Theorem 1. *Let G be a finite directed connected graph. G belongs to the class C_5 if and only if one of the following five conditions is satisfied:*

(a) *G is a cycle,*
(b) *G is I^*-constructible, it has exactly two cycles and it has no cut vertex,*
(c) *G is I-constructible,*
(d) *G is II-constructible,*
(e) *G is III-constructible.*

Moreover, (a), (b), (c), (d) *and* (e) *pairwise exclude each other.*

Proof. It follows from Proposition 2 that G can satisfy at most one of (b)—(e). It is obvious that a graph, obtained by any of the constructions, cannot be a single cycle.

The sufficiency of (a) is trivial, that of (c), (d), (e) has been stated in Proposition 4. It is easy to see that (b) is also sufficient.

The necessity part of the theorem coincides with Proposition 7.

§ 4.

CONSTRUCTION IV. Let

$$G_1, G_2, \ldots, G_t \qquad (t \geqq 2)$$

be (pairwise disjoint) graphs contained in the class C_5. Let us choose a pancyclic vertex[5] A_i in any G_i. Let us form a graph G such that the vertices $A_1, A_2, \ldots, A_t$ are identified with each other, denote this new vertex by A.

Construction IV is completed. The graphs originating by it will be called IV-constructible graphs.

Let us recall the well-known fact that, in any graph, the relation "the edges e_1 and e_2 are completable to a circuit" is an equivalence relation and the subgraphs determined by the equivalence classes are precisely the blocks separated from each other by the cut vertices of the graph (see e.g. Section 5.4 in [3] or Chapter 3 in [2]).

We have the following immediate consequence of Construction IV:

Proposition 8. *Let the graph G result by Construction IV. Then A is a cut vertex of G and G has no other cut vertex. The blocks of G, separated by A, are the graphs $G_1, G_2, \ldots, G_t$. Whenever c is a circuit (or, particularly, a cycle) of G, then all the edges of c belong to the same G_i $(1 \leqq i \leqq t)$.*

Proposition 9. *If a graph G is IV-constructible, then $G \in C_6$.*

Proof. Let G be produced by Construction IV. It is obvious that G is connected. $1 \leqq Z(e) \leqq 2$ holds in G because of the last sentence of Proposition 8 and the validity of these inequalities in every G_i. It follows from the construction (more precisely, from the choice of the A_i's) that A is pancyclic.

Proposition 10. *If a graph G belongs to the difference set $C_6 - C_5$, then G is IV-constructible.*

Proof. Since $G(\in C_6 - C_5)$ satisfies (β), we can choose a pancyclic vertex A in it. Our next aim is to show that no vertex $C(\neq A)$ of G can be a cut vertex. In the contrary case, some part G' of G (separated by C) does not contain A, consequently, A does not occur in the cycles consisting of edges of G' what is impossible by (β).

Since G belongs to C_6 but does not belong to C_5, it must have a cut vertex. Therefore A is the single cut vertex of G. The blocks

$$G_1, G_2, \ldots, G_t \qquad (t \geqq 2)$$

of G, separated by A, are contained in the class C_5. It is evident that G arises from these subgraphs by Construction IV.

By Propositions 9, 10 and Theorem 1, we have reached to a complete description of the graphs belonging to C_6. Our results can be summarized in the following assertion:

---

[5] This requirement means (by Theorem 1 and the constructions mentioned in it) that
A_i is an arbitrary vertex if G_i satisfies (a),
A_i is a vertex fulfilling $Z(A_i)=2$ if (b) is valid for G_i,
A_i is the vertex denoted as A in the corresponding construction if (c) or (d) holds for G_i, and
A_i is either A or B (with the notation used in Construction III) if G_i fulfils (e).

Theorem 2. *A finite directed graph G is contained in the class C_6 if and only if either one of the five conditions* (a), (b), (c), (d), (e) *(occurring in Theorem 1) is true for G or*

(f) *G is IV-constructible.*

Furthermore, these six conditions pairwise exclude each other.

Appendix I.

In this section we verify Proposition 5.

The assumption on the length of the (C, D)-arc guarantees the non-existence of parallel edges with coinciding orientation in G^*.

We write $Z(e)$ or $Z^*(e)$ according as the number of cycles (containing e) is considered in G or in G^*.

Instead of (3) we shall sometimes show the assertion

(3′) there are two cycles in G^* having no vertex in common.

It is obvious that (3′) implies (3).

We use the short expression "$(F, H; G)$-path" instead of "a path from F to H in G". Let a be an $(F, H; G)$-path and let b be an $(F', H'; G)$-path such that b is a subpath of a. If at most one of the equalities $F' = F$ and $H' = H$ holds, then we say that b is a *proper subpath* of a. If $F' \neq F$ and $H' \neq H$, then b is called a *strongly proper subpath* of a.

If a graph G is I-constructible or II-constructible, then we denote by $\pi(G)$ the value of the numerical parameter k (occurring in Constructions I, II) yielding G.

Case 1. G satisfies (a). Then (b) is obviously fulfilled by G^*.

Case 2. (b) holds for G. Denote by A and B the (uniquely determined) vertices whose degree is (2, 1) and (1, 2) (resp.) in G; it is clear that all other vertices of G are simple. Evidently, either the $(C, D; G)$-path or the $(D, C; G)$-path (or both) is uniquely determined by C and D.

Case 2/a. There exists only one $(C, D; G)$-path and this is a proper subpath of a $(B, A; G)$-path. Then $Z^*(e) = 3$ for each edge e of the (single) $(A, B; G)$-path.

Case 2/b. There exists only one $(D, C; G)$-path and this is a strongly proper subpath of a $(B, A; G)$-path. Then G^* satisfies the statement (3′).

Case 2/c. There exists only one $(D, C; G)$-path, this is a subpath of a $(B, A; G)$-path and exactly one of the equalities $A = C$ and $B = D$ holds. It is then evident that G^* is II-constructible (with $\pi(G^*) = 2$).

Case 2/d. There exists only one $(C, D; G)$-path and this is a proper subpath of the (single) $(A, B; G)$-path. Then $Z^*(e) = 4$ for each edge e of the $(A, B; G)$-path which is not contained in the $(C, D; G)$-path.

Case 2/e. There exists only one $(D, C; G)$-path and this is a subpath of the $(A, B; G)$-path. Then $Z^*(e) = 3$ for the edges of the $(D, C; G)$-path.

Case 2/f. $A = C$ and $B = D$. Then G^* is III-constructible.

Case 2/g. C is an inner vertex of the $(A, B; G)$-path and D is an inner vertex of the $(B, A; G)$-path. Then the edges of the $(A, C; G)$-path fulfil $Z^*(e)=3$.

Case 2/h. C is an inner vertex of a $(B, A; G)$-path and D is an inner vertex of the $(A, B; G)$-path. Then $Z^*(e)=3$ for the edges of the $(D, B; G)$-path.

Case 2/i. C and D are inner vertices of the two $(B, A; G)$-paths (resp.). Then $Z^*(e)=3$ for the edges of the $(A, B; G)$-path.

It can be checked that every possible subcase of Case 2 has been exhausted.

Case 3. (c) or (d) holds for G. It follows from Constructions I, II that the number of the $(A, C; G)$-paths and the number of the $(D, A; G)$-paths is 1 or 2. Denote by c an $(A, C; G)$-path, by d a $(D, A; G)$-path and by c^* the new (C, D)-arc (in G^*).

Case 3/a. c and d have no vertex in common[6] but A. Let e_1, e_2 be the edges of c, d (resp.) incident to A. One of e_1, e_2 exists.

Case 3/a/α. One of $Z(e_1)$, $Z(e_2)$ equals 2. Then the paths c^*, c and d form together a cycle in G^*, therefore $Z^*(e_1)$ or $Z^*(e_2)$ is $\geqq 3$.

Case 3/a/β. $Z(e_1)=Z(e_2)=1$. This is possible only if G is II-constructible with an even k, e_1 is the first edge of the (A, B_k)-arc and e_2 is the last edge of the (B_1, A)-arc (we have here used the notation of Construction II/a, cf. Proposition 1). It is easy to see that either $Z^*(e)>2$ is satisfiable or G^* is I-constructible (with $\pi(G^*)=\pi(G)+2$).

Case 3/a/γ. $Z(e_i)=1$ and e_{3-i} does not exist (where i is 1 or 2). Then we can ascertain that either $Z^*(e)>2$ for some edge or G^* is II-constructible (with $\pi(G^*)=\pi(G)+1$).

Case 3/b. c and d have at least two vertices in common. Then $A\neq C$, $A\neq D$ and either C or D is a common vertex of c and d. Let a be a cycle (of G^*) got by taking the union of c^* and the part a' of c or d from D to C. a does not contain A. Let Γ be the set of cycles b of G such that a and b have a vertex in common. It is clear that $1\leqq|\Gamma|\leqq 3$. Let us recall the survey of cycles of G given in the proof of Proposition 3.

Case 3/b/α. G is I-constructible. G has $k(\geqq 4)$ cycles, hence some cycle b' of G is disjoint to a, thus (3′) is true.

Case 3/b/β. G is II-constructible with $\pi(G)\geqq 3$. The number of cycles of G is $k+1(\geqq 4)$, this implies again (3′).

Case 3/b/γ. G is II/a-constructible with $\pi(G)=2$ and $C=B_2$, $D=B_1$. (3) is obviously fulfilled.

Case 3/b/δ. G is II/a-constructible with $\pi(G)=2$ and a' is a proper subpath of either the (B_1, A)-arc or the (B_1, B_2)-arc or the (A, B_2)-arc. Then (3′) holds.

[6] It may happen that either C or D equals A (but not both).

Case 3/b/ε. G is II/a-constructible with $\pi(G)=2$ and either C is an inner vertex of the (B_2, A)-arc or D is an inner vertex of the (A, B_1)-arc. Then $Z^*(e)>2$ holds clearly for the first or last edge of a'.

Case 4. G satisfies (e). Since $Z(e)=2$ is universally valid in a III-constructible graph G and G has a path from D to C (however C and D may be chosen) it is evident that $Z^*(e)>2$ is satisfiable in G^*.

Appendix II.

Now we are going to prove Proposition 6.

Similarly to Appendix I (Case 3), let c denote an $(A_1, C; G_1)$-path and let d denote a $(D, A_2; G_2)$-path. Let e_1 be the first edge of c and e_2 be the last edge of d. We use the notations Z_1, Z_2, Z^* according to the function Z is understood in G_1, G_2, G^* (resp.). $\pi(G)$ has the same meaning as in Appendix I.

Case 1. Either $Z_1(e_1)=2$ or $Z_2(e_2)=2$. Then[7] the conclusion (2) is evidently satisfied.

In the subsequent cases *we shall always assume that* $Z_1(e_1)=Z_2(e_2)=1$. (Therefore G_1 may satisfy (b) only if the degree of A_1 is (1, 2) in G_1; G_2 may fulfil (b) only if the degree of A_2 is (2, 1) in G_2.)

Case 2. G_1 and G_2 fulfil (a). It is obvious that G^* is II-constructible (and $\pi(G^*)=2$).

Case 3. G_1 is a cycle and G_2 satisfies (b). Then either G^* is II-constructible (with $\pi(G^*)=3$) or $Z_1(e_1)=3$ (accordingly to that $Z_2(D)$ is 1 or 2).

Case 4. G_2 satisfies (b) and G_1 is a cycle. The inference is analogous to Case 3 (a distinction is made dependingly on the value of $Z_1(C)$).

Case 5. G_1 is a cycle and G_2 satisfies (d). This case can be treated by the method of Case 3 (with some improvements); G^* may be II-constructible with $\pi(G^*)=\pi(G_2)+2$.

Case 6. G_1 satisfies (d) and G_2 is a cycle. The treatment of this case is an improved version of Case 4 (likely to the interrelation of Cases 5 and 3).

Case 7. (b) holds for G_1 and (d) holds for G_2. Either G^* is II-constructible (with $\pi(G^*)=\pi(G_2)+3$); or one of $Z^*(e_1)$, $Z^*(e_2)$ equals 3.

Case 8. (d) is true for G_1 and (b) is true for G_2. The treatment is symmetrical to Case 7.

Case 9. G_1 and G_2 satisfy (d). If $Z_1(C)=Z_2(D)=1$, then G^* is II-constructible (with $\pi(G^*)=\pi(G_1)+\pi(G_2)+2$); otherwise either $Z^*(e_1)$ or $Z^*(e_2)$ equals 3.

[7] We can perceive that Case 1 comprises a large collection of possible situations; among others, the possibilities when (c) or (e) is valid for G_1 or G_2 are entirely included.

О графах удовлетворяющих некоторым условиям для циклов, I.

Цель настоящей работы — дать структурное описание конечных ориентированных графов удовлетворяющих условиям:

для всякого ребра e, число циклов содержащих e равняется 1 или 2,

существует вершина содержаемая в каждом цикле графа.

Ясно, что граф выполняющий эти требования может иметь не больше чем одну точку сочленения.

Опираемся на результаты предыдушей стати [1]. В §§ 2—3 даём некоторые конструкции и доказываем, что они представляют все графы обладающие вышеупомянутыми свойствами и не имеющими точку сочленения. В § 4 описание распространяется на графы в которых бывает точка сочленения.

MATHEMATICAL INSTITUTE OF THE
HUNGARIAN ACADEMY OF SCIENCES
H-1053 BUDAPEST, HUNGARY
REÁLTANODA U. 13—15.
(PERMANENTLY)

MATHEMATICS DEPARTMENT OF THE
ARTS AND SCIENCE UNIVERSITY
RANGOON, BURMA
(IN THE TIME OF PREPARING THIS PAPER)

References

[1] ÁDÁM, A., On some generalizations of cyclic networks, *Acta Cybernet.* v. 1, 1971, pp. 105—119.
[2] HARARY, F., *Graph theory,* Addison—Wesley, Reading, 1969.
[2a] Харари, Ф., *Теория графов,* Мир, Москва, 1973.
[3] ORE, O., *Theory of graphs,* Amer. Math. Soc. Coll. Publ. v. 38, Providence, 1962.

(Received Feb. 17, 1975)

Zur Theorie kommutativer Automaten

Von J. DUSKE

1. Ein Automat ist ein Tripel $\mathscr{A}=(A, F, \delta)$. Hierbei ist A eine endliche Menge, die Zustandsmenge von $\mathscr{A}$, F ein Monoid mit Einselement e und $\delta: A\times F\to A$ eine Abbildung, für die gilt:

$$\forall a\in A,\ \forall f_1, f_2\in F\colon \delta(a,\ f_1 f_2)=\delta\big(\delta(a, f_1), f_2\big)$$

und

$$\forall a\in A \qquad\qquad : \delta(a, e) = a$$

Für $\delta(a, f)$ schreibt man kurz af. $\mathscr{A}=(A, F, \delta)$ heißt kommutativ in $a\in A$, wenn $af_1 f_2=af_2 f_1$ für alle $f_1, f_2\in F$ gilt. $\mathscr{A}$ heißt kommutativ, wenn $\mathscr{A}$ in allen $a\in A$ kommutativ ist.

$\mathscr{A}$ heißt zyklisch, wenn es ein $a\in A$ mit $A=\{af\,|\,f\in F\}$ gibt. Ein solches $a\in A$ heißt ein erzeugendes Element von $\mathscr{A}$.

Eine Äquivalenzrelation ϱ auf A heißt Relation mit S.E. (Substitutionseigenschaft), falls gilt:

$$\forall a_1, a_2\in A\colon (a_1, a_2)\in\varrho\Rightarrow\forall f\in F\colon (a_1 f, a_2 f)\in\varrho$$

Die zugehörende Partition $A_\varrho=\{A_\varrho(a)|a\in A\}$ von A heißt dann Partition mit S.E. bzgl. $\mathscr{A}$. Hierbei ist $A_\varrho(a)=\{a'|(a, a')\in\varrho\}$. Die Menge aller Relationen mit S.E. bzgl. $\mathscr{A}$ bildet einen Verband $R(\mathscr{A})$ mit den in üblicher Weise für Relationen erklärten Verbandsoperationen $\wedge$, $\vee$.

Eine Relation ϱ mit S.E. bzgl. $\mathscr{A}$ heißt kommutativ, wenn $\forall a\in A$, $\forall f_1, f_2\in F$ gilt: $(af_1 f_2, af_2 f_1)\in\varrho$. Die Menge aller Relationen auf A mit S.E. bzgl. $\mathscr{A}$, die kommutativ sind, bildet einen Teilverband $R_k(\mathscr{A})$ von $R(\mathscr{A})$. ϱ_k sei das kleinste Element von $R_k(\mathscr{A})$. Mit $A_k=\{A_{\varrho_k}(a)|a\in A\}$ werde die zugehörende Partition bezeichnet. Der Quotientenautomat $\mathscr{A}_{\varrho_k}=\mathscr{A}_k=(A_k, F, \delta_k)$ mit $\delta_k(A_{\varrho_k}(a), f)=A_{\varrho_k}\big(\delta(a, f)\big)$ ist kommutativ und heißt maximaler kommutativer Quotient von $\mathscr{A}$.

Ist speziell $F=X^*$ das freie, von der endlichen Menge X erzeugte Monoid, dann wird $\mathscr{A}$ angegeben durch $\mathscr{A}=(A, X, \delta)$, wobei δ eine Funktion von $A\times X$ nach A ist. Durch $\delta(a, e)=a$ und $\delta(a, wx)=\delta\big(\delta(a, w), x\big)$ für alle $a\in A$, $x\in X$ und $w\in X^*$ wird δ zu einer Funktion von $A\times X^*$ nach A erweitert. $\mathscr{A}=(A, X, \delta)$ ist genau dann kommutativ, wenn $ax_1x_2=ax_2x_1$ für alle $a\in A$ und alle $x_1, x_2\in X$ gilt. $\mathscr{A}_k$ läßt sich in diesem Fall folgendermaßen berechnen:

Man bilde für alle Paare $(x_1, x_2) \in X \times X$, $x_1 \neq x_2$, und für alle $a \in A$ die Paare (ax_1x_2, ax_2x_1), und bestimme die kleinste Relation ϱ mit S.E. bzgl. $\mathscr{A}$, die alle diese Paare enthält. Es ist dann $\varrho = \varrho_k$.

1.1. *Beispiel.* $\mathscr{A} = (A, X, \delta)$ sei gegeben durch (vgl. [8]):

$\mathscr{A}$	1	2	3	4	5	6
x	4	5	6	1	2	3
y	3	1	2	5	6	4

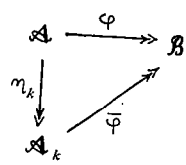
$A = \{1, 2, 3, 4, 5, 6\}$
$X = \{x, y\}$

$\mathscr{A}$ ist nicht kommutativ. Es gilt z.B. $1xy = 5$ und $1yx = 6$. Mit $\alpha = \{1, 2, 3\}$ und $\beta = \{4, 5, 6\}$ wird $\mathscr{A}_k$ durch folgende Tafel gegeben:

$\mathscr{A}_k$	α	β
x	β	α
y	α	β

Mit η_k werde die Projektion von $\mathscr{A}$ auf $\mathscr{A}_k$ bezeichnet. Für die im folgenden verwendeten Begriffe Homomorphismus, Isomorphismus, Automorphismus und Automorphismengruppe vgl. man [4]. $\mathscr{A}_k$ besitzt folgende universelle Eigenschaft:

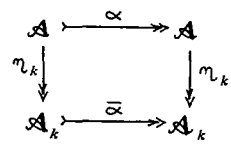
1.2. **Satz.** $\mathscr{B} = (B, F, \delta_1)$ sei ein kommutativer Automat und $\varphi: \mathscr{A} \to \mathscr{B}$ ein Homomorphismus von $\mathscr{A}$ nach $\mathscr{B}$. Dann gibt es genau einen Homomorphismus $\bar{\varphi}: \mathscr{A}_k \to \mathscr{B}$, so daß folgendes Diagramm kommutativ wird:



1.3. *Lemma.* $\mathscr{A} = (A, F, \delta)$ sei ein Automat, $G(\mathscr{A})$ die Automorphismengruppe von $\mathscr{A}$ und $\alpha \in G(\mathscr{A})$. Dann gilt:

$$\forall a_1, a_2 \in A: (a_1, a_2) \in \varrho_k \Rightarrow (\alpha(a_1), \alpha(a_2)) \in \varrho_k$$

Beweis. Es gibt genau ein $\bar{\alpha} \in G(\mathscr{A}_k)$, so daß folgendes Diagramm kommutativ wird:



(I)

Aus $(a_1, a_2) \in \varrho_k$ folgt dann $\eta_k(\alpha(a_1)) = \eta_k(\alpha(a_2))$, also $(\alpha(a_1), \alpha(a_2)) \in \varrho_k$.

1.4. *Lemma.* $\mathscr{A}$ sei ein Automat. Die Abbildung, die jedem $\alpha \in G(\mathscr{A})$ das eindeutig bestimmte $\bar{\alpha} \in G(\mathscr{A}_k)$ zuordnet, für das (I) kommutativ wird, ist ein Homomorphismus von $G(\mathscr{A})$ nach $G(\mathscr{A}_k)$.

Dieser Homomorphismus wird im folgenden mit π bezeichnet. $\mathscr{A}$ heißt streng zusammenhängend, wenn jedes $a \in A$ ein erzeugendes Element von $\mathscr{A}$ ist. Für den im folgenden verwendeten Begriff der Halbgruppe eines Automaten vgl. man [4], S. 239.

1.5. **Satz.** $\mathscr{A}=(A, F, \delta)$ sei streng zusammenhängend. H sei der Kern von π, und K die Kommutatorgruppe von $G(\mathscr{A})$. Dann gilt $K \trianglelefteq H$, d.h., K ist Normalteiler von H.

Beweis. $\mathscr{A}_k$ ist streng zusammenhängend und kommutativ. Die Halbgruppe $S(\mathscr{A}_k)$ von $\mathscr{A}_k$ (vgl. [4]) ist kommutativ und isomorph zu $G(\mathscr{A}_k)$ (vgl. [3]). Also ist $G(\mathscr{A})/H$ kommutativ, und daher $K \trianglelefteq H$.

Ist $\alpha \in H$ und $a \in A$, dann gilt:

$$\eta_k(\alpha(a))=\pi(\alpha)\eta_k(a)=\eta_k(a), \quad \text{also} \quad (\alpha(a), a) \in \varrho_k.$$

Auf Grund dieser Bemerkung ergibt sich:

1.6. *Korollar.* $\mathscr{A}=(A, F, \delta)$ sei streng zusammenhängend. Für alle $\alpha, \beta \in G(\mathscr{A})$ und alle $a \in A$ gilt:

$$(\alpha\beta(a), \beta\alpha(a)) \in \varrho_k$$

Beweis. Der Kommutator $\alpha^{-1}\beta^{-1}\alpha\beta$ liegt in H. Daher gilt $(\alpha^{-1}\beta^{-1}\alpha\beta(a), a) \in \varrho_k$ für alle $a \in A$.

2. In den folgenden Abschnitten wird $F=X^*$ angenommen. $\mathscr{A}=(A, X, \delta)$ sei zyklisch und $a \in A$ ein erzeugendes Element. Man definiere folgende Rechtskongruenz auf F:

$$\forall f_1, f_2 \in F: (f_1, f_2) \in \varrho_a \Leftrightarrow af_1 = af_2$$

B_a sei die zu ϱ_a gehörende Menge von Blöcken von F. $\mathscr{F}_a=(B_a, X, \delta_a)$ mit $\delta_a(B_a(f), f_1)=B_a(ff_1)$ ist isomorph zu $\mathscr{A}$. Ein Isomorphismus wird gegeben durch $\chi(a_1)= =B_a(f)$ mit $a_1=af$. $(\mathscr{F}_a)_k$ ist isomorph zu $\mathscr{A}_k$.

Zur Beschreibung von $(\mathscr{F}_a)_k$ definiere man zunächst folgende Kongruenz $\varkappa$ auf F:

$$\forall f, h \in F: (f, h) \in \varkappa \Leftrightarrow f = x_1 \ldots x_n \quad \text{und}$$

$$h = x_{i_1} \ldots x_{i_n}, \quad \text{wobei} \quad i_1, \ldots i_n$$

eine Permutation von $1, \ldots, n$ ist.

Ein solches h soll Kommutant von f genannt werden, und mit $k(f)$ wird die Menge aller Kommutanten von f bezeichnet.

Sei nun τ eine beliebige Rechtskongruenz mit endlichem Index auf F und B_τ die zugehörende Partition von F. $\mathscr{F}_\tau=(B_\tau, F, \delta_\tau)$ wird in natürlicher Weise definiert. Die obere Grenze der Rechtskongruenzen $\varkappa$ und τ, $\varkappa \vee \tau$, bestimmt eine Relation η auf B_τ mit S.E. bzgl. $\mathscr{F}_\tau$ durch

$$(B_\tau(f_1), B_\tau(f_2)) \in \eta \Leftrightarrow (f_1, f_2) \in \varkappa \vee \tau \quad \text{für alle} \quad f_1, f_2 \in F.$$

Der Quotientenautomat von $\mathscr{F}_\tau$ nach η ist isomorph zu $\mathscr{F}_{\tau \vee \varkappa}$.

η ist kommutativ bzgl. $\mathscr{F}_\tau$. Seien dazu $h_1, h_2 \in F$ und $B_\eta(B_\tau(f))$ beliebig ausgewählt. Es gilt dann $B_\eta(B_\tau(f))h_1h_2=B_\eta(B_\tau(f)h_1h_2)=B_\eta(B_\tau(fh_1h_2))$. Weiter ist $(fh_1h_2, fh_2h_1) \in \tau \vee \varkappa$ also gilt $(B_\tau(fh_1h_2), B_\tau(fh_2h_1)) \in \eta$.

Bezeichnet man wieder mit ϱ_k die kleinste kommutative Relation auf B_τ mit S.E. bzgl. $\mathscr{F}_\tau$, dann ist also $\varrho_k \leqq \eta$. Es gilt auch die umgekehrte Inklusion:

$$(B_\tau(f_1), B_\tau(f_2)) \in \eta \Leftrightarrow (f_1, f_2) \in \tau \vee \varkappa$$
$$\Leftrightarrow \exists h_0, h_1, \ldots, h_n \in F \quad \text{mit} \quad f_1 = h_0, h_n = f_2 \quad \text{und}$$
$$(h_{i-1}, h_i) \in \varkappa \quad \text{oder} \quad (h_{i-1}, h_i) \in \tau \quad \text{für} \quad i \in [1:n]$$

Gilt $(h_{i-1}, h_i) \in \varkappa$, dann ist $(B_\tau(h_{i-1}), B_\tau(h_i)) = (B_\tau(e)h_{i-1}, B_\tau(e)h_i) \in \varrho_k$. Gil $(h_{i-1}, h_i) \in \tau$, dann ist $B_\tau(h_{i-1}) = B_\tau(h_i)$. Daraus folgt $(B_\tau(f_1), B_\tau(f_2)) \in \varrho_k$, und damit $\varrho_k = \eta$, also ist $(\mathscr{F}_\tau)_k$ isomorph zu $\mathscr{F}_{\tau \vee \varkappa}$.

In $\mathscr{F}_\tau = (B_\tau, F, \delta_\tau)$ werde $B_\tau(e)$ als Anfangszustand und weiter eine Menge $S \subseteqq B_\tau$ als Menge von Endzuständen ausgezeichnet. Der so bestimmte erkennende Automat erkennt die Wortmenge

$$E = \{f | B_\tau(f) \in S\}.$$

In $\mathscr{F}_{\tau \vee \varkappa} = (B_{\tau \vee \varkappa}, F, \delta_{\tau \vee \varkappa})$ zeichne man $B_{\tau \vee \varkappa}(e)$ als Anfangszustand und $S_k = \{B_{\tau \vee \varkappa}(f) | f \in E\}$ als Menge von Endzuständen aus. Dieser Automat erkennt dann die Wortmenge

$$E' = \{h | B_{\tau \vee \varkappa}(h) \in S_k\} = \{h | \exists f \in E \quad \text{mit} \quad (h, f) \in \tau \vee \varkappa\}.$$

$E^k = \{h | \exists f \in E \text{ und } h \in k(f)\}$ sei die kommutative Hülle von E. Es ist $E \subseteqq E^k \subseteqq E'$, und es gilt:

$$E' = E^k \Leftrightarrow \bigcup_{f \in E} B_{\tau \vee \varkappa}(f) = \bigcup_{f \in E} B_\varkappa(f)$$
$$\Leftrightarrow \bar{\exists} (f_1, f_2) \in \tau \quad \text{mit} \quad f_1 \notin E^k \quad \text{und} \quad f_2 \in E_k$$
$$\Leftrightarrow \forall h \in F \text{ gilt: aus } (h, f_1) \in \tau \text{ mit } f_1 \in k(f) \text{ für ein}$$
$$f \in E \text{ folgt: } \exists \bar{f} \in E \text{ mit } h \in k(\bar{f})$$

Man gehe nun von einer beliebigen Wortmenge $E \subseteqq X^*$ aus. E ist genau dann regulär, wenn E Vereinigung von Äquivalenzklassen einer Rechtskongruenz τ mit endlichem Index auf $F = X^*$ ist. Für τ kann man dann z.B. dei Myhill-Kongruenz oder die Nerode-Rechtskongruenz wählen (vgl. [7]). E' ist dann sicher regulär, und man erhält z.B.:

2.1. **Satz.** $E \subseteqq X^* = F$ sei regulär. Falls für alle h aus F und für alle $f \in E^k$ gilt:

$$\{\forall w \in F: hw \in E \Leftrightarrow fw \in E\} \Rightarrow h \in E^k,$$

dann ist E^k regulär.

$E \subseteqq X^* = F$ sei eine reguläre Menge, und $\mathscr{A}(E)$ die Menge aller endlichen erkennenden Automaten, die E erkennen, und deren Anfangszustand ein erzeugendes Element ist. $k(\mathscr{A}(E))$ sei die Menge aller maximalen kommutativen Quotienten von Automaten aus $\mathscr{A}(E)$. Es gilt:

2.2. **Satz.** E sei regulär. E^k ist genau dann regulär, wenn E^k von einem Automaten aus $k(\mathscr{A}(E))$ erkannt wird.

Beweis. E^k sei regulär. σ sei die Myhill-Kongruenz von E und γ die Myhill-Kongruenz von E^k. $\tau=\sigma\wedge\gamma$ besitzt einen endlichen Index. E ist Vereinigung von Äquivalenzklassen bzgl. τ. Es ist also $\mathscr{F}_\tau=(B_\tau, F, \delta_\tau)\in\mathscr{A}(E)$. (Als Anfangszustand ist wieder $B_\tau(e)$ und als Menge von Endzuständen ist $\{B_\tau(f)|\, f\in E\}$ gewählt). $(\mathscr{F}_\tau)_k=$ $=(B_{\tau\vee\varkappa}, F, \delta_{\tau\vee\varkappa})$ ist aus $k(\mathscr{A}(E))$ und erkennt

$$E'=\bigcup_{f\in E} B_{\tau\vee\varkappa}(f)\supseteq E^k=\bigcup_{f\in E} B_\varkappa(f).$$

Es ist $\gamma=\gamma\vee\varkappa$, also $\varkappa\leqq\gamma$. Da auch $\tau\leqq\gamma$ gilt, folgt $\tau\vee\varkappa\leqq\gamma$, und damit

$$E'=\bigcup_{f\in E} B_{\tau\vee\varkappa}(f)\subseteq E^k, \quad \text{also} \quad E'=E^k.$$

Es gilt weiter:

2.3. **Satz.** $E\subseteqq X^*=F$ sei regulär. E^k ist genau dann regulär, wenn es eine Rechtskongruenz τ mit endlichen Index auf F gibt, für die gilt:

(a) E ist Vereinigung von Äquivalenzklassen bzgl. τ

(b) $\forall h\in F,\ \forall f\in E^k$ gilt: $(h,f)\in\tau\Rightarrow h\in E^k$

Beweis. E^k sei regulär. Wähle dann τ wie im vorangehenden Beweis. Es möge umgekehrt ein τ existieren, das die Bedingungen des Satzes erfüllt. Dann wird E^k von (B_τ, F, δ_τ) erkannt.

3. $G(\mathscr{A})$ sei die Automorphismengruppe von $\mathscr{A}=(A, F, \delta)$. Für Untergruppen J von $G(\mathscr{A})$ ist dann ein Quotientenautomat $\mathscr{A}/J$ von $\mathscr{A}$ definiert. Die Zustände von $\mathscr{A}/J$ sind gerade die Transitivitätsklassen von A bzgl. J (vgl. [4]). $\mathscr{A}/G(\mathscr{A})$ ist genau dann kommutativ, wenn jeder Block bzgl. ϱ_k in einer Transitivitätsklasse bzgl. $G(\mathscr{A})$ enthalten ist. Hierbei ist ϱ_k wieder die kleinste kommutative Relation auf A mit S.E. bzgl. $\mathscr{A}$.

$\mathscr{A}$ sei zyklisch und $a\in A$ ein erzeugendes Element von $\mathscr{A}$. Ist $(a_1, a_2)\in\varrho_k$, dann gibt es Zustände $z_1=a_1, z_2, \ldots, z_l=a_2$ und $g_1, g_2, \ldots, g_l\in F$ mit $z_i=ag_i$ für $i\in[1:l]$ und $g_{i+1}\in k(g_i)$ für $i\in[1:l-1]$. Es ist also $\mathscr{A}/G(\mathscr{A})$ genau dann kommutativ, wenn es für alle $f\in F$ und $g\in k(f)$ ein $\alpha\in G(\mathscr{A})$ mit $\alpha(af)=\alpha(a)f=ag$ gibt. Ist $\mathscr{A}$ transitiv (vgl. [4]), dann ist diese Bedingung sicher erfüllt. Ist $B(a)$ die Transitivitätsklasse bzgl. $G(\mathscr{A})$, in der a liegt, dann kann man diese Bedingung auch folgendermaßen formulieren:

Für alle $f\in F$ und $g\in k(f)$ gibt es ein $a_1\in B(a)$ mit $af=a_1g$.

H sei wie in 1. der Kern des Homomorphismus $\pi: G(\mathscr{A})\to G(\mathscr{A}_k)$. Die Transitivitätsklassen von A bzgl. H sind ganz in den Blöcken bzgl. ϱ_k enthalten. Es gilt:

3.1. **Satz.** $\mathscr{A}$ sei streng zusammenhängend und $\mathscr{A}/G(\mathscr{A})$ kommutativ. Dann ist $\mathscr{A}_k=\mathscr{A}/H$.

Beweis. Die Blöcke bzgl. ϱ_k sind in diesem Fall gerade die Transitivitätsklassen von A bzgl. H. Seien dazu $(a_1, a_2)\in\varrho_k$. Es gibt ein $\alpha\in G(\mathscr{A})$ mit $\alpha(a_1)=a_2$. Es genügt jetzt zu zeigen, daß α aus H ist. Da $\mathscr{A}$ streng zusammenhängend ist, ist a_1 ein erzeugendes Element von $\mathscr{A}$. $B=\eta_k(a_1)$ ist dann ein erzeugendes Element von $\mathscr{A}_k$. Für $\bar{\alpha}=$ $=\pi(\alpha)$ gilt:

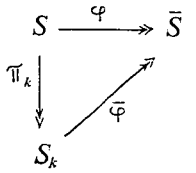
$\bar{\alpha}(B)=\bar{\alpha}(\eta_k(a_1))=\eta_k(\alpha(a_1))=\eta_k(a_2)=B$, und daher ist $\bar{\alpha}$ die identische Abbildung von $\mathscr{A}_k$.

H besitzt folgende ausgezeichnete Stellung unter allen Untergruppen J von $G(\mathscr{A})$, für die $\mathscr{A}/J$ kommutativ ist. Es gilt (vgl. [1]):

3.2. **Satz.** $\mathscr{A}$ sei streng zusammenhängend und transitiv. J sei eine Untergruppe von $G(\mathscr{A})$, und $\mathscr{A}/J$ sei kommutativ. Dann ist H eine Untergruppe von J.

4. Ist S ein Monoid und σ eine Kongruenz auf S, dann heißt σ genau dann kommutativ, wenn S/σ ein kommutatives Monoid ist, d.h., genau dann, wenn $(s_1 s_2, s_2 s_1) \in \sigma$ für alle $s_1, s_2 \in S$ gilt. Der Durchschnitt ϱ aller kommutativen Kongruenzen auf S ist kommutativ. $S_k = S/\varrho$ heißt maximales kommutatives Bild von S. Mit π_k werde die Projektion von S auf S_k bezeichnet. S_k besitzt folgende universelle Eigenschaft (vgl. [2]):

Ist $\varphi\colon S \to \bar{S}$ ein Homomorphismus auf ein kommutatives Monoid $\bar{S}$, dann gibt es genau einen Homomorphismus $\bar{\varphi}\colon S_k \to \bar{S}$, so daß folgendes Diagramm kommutativ wird:



Mit $S(\mathscr{A})$ und $S(\mathscr{A}_k)$ bezeichne man die Halbgruppen von $\mathscr{A}$ und $\mathscr{A}_k$. $S(\mathscr{A}_k)$ ist homomorphes Bild von $S(\mathscr{A})$. Mit $S_k(\mathscr{A})$ bezeichne man das maximale homomorphe kommutative Bild von $S(\mathscr{A})$. $S(\mathscr{A}_k)$ ist homomorphes Bild von $S_k(\mathscr{A})$, im allgemeinen sind diese Halbgruppen jedoch nicht isomorph, wie das folgende Beispiel zeigt.

4.1. *Beispiel.* $\mathscr{A}$ sei gegeben durch:

$\mathscr{A}$	a	b
0	a	a
1	b	b

$\mathscr{A}$ ist nicht kommutativ, und $\mathscr{A}_k$ besitzt genau einen Zustand. Die Halbgruppe $S(\mathscr{A})$ von $\mathscr{A}$ ergibt sich zu:

$S(\mathscr{A})$	$\bar{e}$	$\bar{0}$	$\bar{1}$
$\bar{e}$	$\bar{e}$	$\bar{0}$	$\bar{1}$
$\bar{0}$	$\bar{0}$	$\bar{0}$	$\bar{1}$
$\bar{1}$	$\bar{1}$	$\bar{0}$	$\bar{1}$

Die Teilmengen $D = \{\bar{0}, \bar{1}\}$ und $E = \{\bar{e}\}$ bestimmen eine Kongruenz ϱ auf $S(\mathscr{A})$. $S(\mathscr{A})/\varrho$ ist kommutativ und gleich $S_k(\mathscr{A})$.

$S_k(\mathscr{A})$	E	D
E	E	D
D	D	D

In diesem Fall ist also $S_k(\mathscr{A})$ nicht isomorph zu $S(\mathscr{A}_k)$.

$\mathscr{A}$ heißt transitiv, wenn es für je zwei Zustände $a_1, a_2 \in A$ ein $\alpha \in G(\mathscr{A})$ mit $\alpha(a_1) = a_2$ gibt. Ist $\mathscr{A}$ transitiv, dann ist auch $\mathscr{A}_k$ transitiv. Seien dazu $\eta_k(a_1)$ und $\eta_k(a_2)$ zwei Zustände von $\mathscr{A}_k$ und $\alpha(a_1) = a_2$ für ein $\alpha \in G(\mathscr{A})$. Dann gilt:

$$\bar{\alpha}(\eta_k(a_1)) = \eta_k(\alpha(a_1)) = \eta_k(a_2) \quad \text{mit} \quad \bar{\alpha} = \pi(\alpha) \in G(\mathscr{A}_k)$$

4.2. **Satz.** Ist $\mathscr{A}$ zyklisch und transitiv, dann ist $S_k(\mathscr{A})$ isomorph zu $S(\mathscr{A}_k)$.

Beweis. $a \in A$ sei ein erzeugendes Element von $\mathscr{A}$ und $\tau = \varrho_a$. $\mathscr{A}$ ist isomorph zu $\mathscr{F}_\tau$, es genügt daher zu zeigen, daß $S_k(\mathscr{F}_\tau)$ isomorph zu $S(\mathscr{F}_{\tau \vee \varkappa})$ ist. $\mathscr{F}_\tau$ und $\mathscr{F}_{\tau \vee \varkappa}$ sind transitiv und damit zustandsunabhängig (vgl. [4]). Also ist $S(\mathscr{F}_\tau) = F/\tau$ und $S(\mathscr{F}_{\tau \vee \varkappa}) = F/\tau \vee \varkappa$. Die kleinste kommutative Kongruenz auf F/τ werde mit ϱ bezeichnet. Dann ist $S_k(\mathscr{F}_\tau) = (F/\tau)/\varrho$. Mann zeigt leicht (vgl. den Beginn von 2.), daß $(F/\tau)/\varrho$ isomorph zu $F/\tau \vee \varkappa$ ist, womit der Satz bewiesen ist.

4.3. *Beispiel.* $\mathscr{A}$ aus 1.1. erfüllt die Voraussetzungen des vorangehenden Satzes. Für $S(\mathscr{A})$ ergibt sich die symmetrische Gruppe $\mathfrak{S}_3$ (vgl. [8]).

$S(\mathscr{A})$	$\bar{e}$	$\bar{y}$	$\bar{x}$	$\overline{yy}$	$\overline{yx}$	$\overline{y^2x}$
$\bar{e}$	$\bar{e}$	$\bar{y}$	$\bar{x}$	$\overline{yy}$	$\overline{yx}$	$\overline{y^2x}$
$\bar{y}$	$\bar{y}$	$\overline{yy}$	$\overline{yx}$	$\bar{e}$	$\overline{y^2x}$	$\bar{x}$
$\bar{x}$	$\bar{x}$	$\overline{y^2x}$	$\bar{e}$	$\overline{yx}$	$\overline{yy}$	$\bar{y}$
$\overline{yy}$	$\overline{yy}$	$\bar{e}$	$\overline{y^2x}$	$\bar{y}$	$\bar{x}$	$\overline{yx}$
$\overline{yx}$	$\overline{yx}$	$\bar{x}$	$\bar{y}$	$\overline{y^2x}$	$\bar{e}$	$\overline{yy}$
$\overline{y^2x}$	$\overline{y^2x}$	$\overline{yx}$	$\overline{yy}$	$\bar{x}$	$\bar{y}$	$\bar{e}$

Die Kommutatorgruppe der $\mathfrak{S}_3$ ist $\{\bar{e}, \bar{y}, \overline{yy}\}$. Die Faktorgruppe nach der Kommutatorgruppe wird dann gegeben durch:

$S_k(\mathscr{A})$	E	G
E	E	G
G	G	E

mit $E = \{\bar{e}, \bar{y}, \overline{yy}\}$ und $G = \{\bar{x}, \overline{yx}, \overline{y^2x}\}$. Für die Halbgruppe $S(\mathscr{A}_k)$ ergibt sich

$S(\mathscr{A}_k)$	$\bar{e}$	$\bar{x}$
$\bar{e}$	$\bar{e}$	$\bar{x}$
$\bar{x}$	$\bar{x}$	$\bar{e}$

$S(\mathscr{A}_k)$ ist also isomorph zu $S_k(\mathscr{A})$.

5. Für die im folgenden verwendeten Begriffe (reduzierter) Mealyautomat und Automatenabbildung vgl. man [4]. $\mathscr{M} = (A, X, Y, \delta, \lambda)$ sei ein Mealyautomat und $\mathscr{A} = (A, X, \delta)$ der zugehörende (Zustands-) Automat. (A braucht nicht notwendig endlich zu sein. X und Y sollen jedoch immer endliche Alphabete sein.) λ wird auf $A \times X_e^*$ durch $\lambda(a, wx) = \lambda(\delta(a, w), x)$ für alle $a \in A$, $x \in X$ und $w \in X^*$ erweitert. Hierbei ist $X_e^* = X^* - \{e\}$.

5.1. *Definition.* $\mathcal{M}=(A, X, Y, \delta, \lambda)$ heißt fast-kommutativ, wenn $\lambda(a, x_1x_2f)=\lambda(a, x_2x_1f)$ für alle $x_1, x_2\in X, f\in X_e^*$ und $a\in A$ gilt.

Wenn $\mathcal{M}$ fast kommutativ ist, dann gilt natürlich auch $\lambda(a, px_1x_2f)=\lambda(a, px_2x_1f)$ für alle $p\in X^*$ und für alle a, x_1, x_2 und f wie oben. Der durch folgende Tafel gegebene Mealyautomat ist z.B. fast-kommutativ:

$\mathcal{M}$	a	b
0	(a, x)	(a, x)
1	(b, y)	(b, y)

5.2. **Satz.** $\mathcal{M}=(A, X, Y, \delta, \lambda)$ sei fast-kommutativ, und $\mathcal{M}_r=(A_r, X, Y, \delta_r, \lambda_r)$ sei der zu $\mathcal{M}$ gehörende reduzierte Mealyautomat. Dann ist $\mathcal{A}_r=(A_r, X, \delta_r)$ kommutativ.

Beweis. h sei die Projektion von $\mathcal{M}$ auf $\mathcal{M}_r$. Falls $\mathcal{A}_r$ nicht kommutativ ist, dann gilt:

$$\exists\, \bar{z}\in A_r \text{ und } \exists\, x_1, x_2\in X \text{ mit } \quad \delta_r(\bar{z}, x_1x_2)\neq\delta_r(\bar{z}, x_2x_1).$$

Man wähle ein $z\in A$ mit $h(z)=\bar{z}$ aus. Es sei

$$z_1=\delta(z, x_1x_2) \quad \text{und} \quad h(z_1)=\bar{z}_1=\delta_r(\bar{z}, x_1x_2)$$

und

$$z_2=\delta(z, x_2x_1) \quad \text{und} \quad h(z_2)=\bar{z}_2=\delta_r(\bar{z}, x_2x_1)$$

Da $\mathcal{M}_r$ reduziert und $\bar{z}_1\neq\bar{z}_2$ ist, gibt es ein $f\in X_e^*$ mit $\lambda_r(\bar{z}_1, f)\neq\lambda_r(\bar{z}_2, f)$, woraus sich $\lambda(z_1, f)\neq\lambda(z_2, f)$ oder $\lambda(z, x_1x_2f)\neq\lambda(z, x_2x_1f)$ ergibt, im Widerspruch zur Voraussetzung über $\mathcal{M}$.

Mit Hilfe dieses Satzes ergibt sich sofort:

5.3. *Korollar.* $\mathcal{M}=(A, X, Y, \delta, \lambda)$ sei fast-kommutativ. Dann gilt:

$$\forall p, q\in X^*, \forall r\in X_e^*, \forall a\in A: \lambda(a, pqr)=\lambda(a, qpr)$$

Ist ein Mealyautomat $\mathcal{M}=(A, X, Y, \delta, \lambda)$ gegeben, dann definiere man für jeden Zustand $a\in A$ eine Abbildung $\lambda_a: X^*\to Y^*$ durch

$$\lambda_a(e)=e$$

und

$$\forall u\in X^*, \forall x\in X: \lambda_a(ux)=\lambda_a(u)\,\lambda(\delta(a, u), x).$$

λ_a ist eine Automatenabbildung (vgl. [4]), und jede Automatenabbildung $\alpha: X^*\to Y^*$ kann als λ_a für einen geeigneten Mealyautomaten $\mathcal{M}$ und ein geeignetes $a\in A$ dargestellt werden. Man sagt dann, daß $a\in A$ die Abbildung α induziert. Für $w\in X^*$ sei im folgenden $\bar{w}$ der letzte Buchstabe von w ($\bar{e}=e$).

Ist $\alpha: X^*\to Y^*$ eine Automatenabbildung, dann kann man für jedes $p\in X^*$ eine Automatenabbildung $\alpha_p: X^*\to Y^*$ durch $\alpha(pu)=\alpha(p)\alpha_p(u)$ für alle $u\in X^*$ definieren. Setzt man $A_\alpha=\{\alpha_p | p\in X^*\}$, $\delta(\alpha_p, x)=\alpha_{px}$ und $\lambda(\alpha_p, x)=\overline{\alpha(px)}$ für alle $\alpha_p\in A_\alpha$ und $x\in X$, dann ist $\mathcal{M}_\alpha=(A_\alpha, X, Y, \delta, \lambda)$ ein reduzierter Mealyautomat, und $\alpha_e=\alpha\in A_\alpha$ induziert α.

5.4. *Definition.* Eine Automatenabbildung $\alpha: X^*\to Y^*$ heißt fast-kommutativ, wenn $\overline{\alpha(px_1x_2f)}=\overline{\alpha(px_2x_1f)}$ für alle $x_1, x_2\in X, f\in X_e^*$ und $p\in X^*$ gilt.

Ist α fast-kommutativ, dann ist der oben definierte Mealyautomat $\mathcal{M}_\alpha$ fast-kommutativ, denn es gilt:

$$\lambda(\alpha_p, x_1x_2f) = \lambda\big(\delta(\alpha_p, x_1x_2), f\big) = \lambda(\alpha_{px_1x_2}, f) =$$
$$= \overline{\alpha(px_1x_2f)} = \overline{\alpha(px_2x_1f)} = \lambda(\alpha_p, x_2x_1f)$$

$\mathcal{A}_\alpha = (A_\alpha, X, \delta)$ ist also kommutativ.

Zustände von fast-kommutativen Mealyautomaten induzieren fast-kommutative Automatenabbildungen. Es gilt also:

5.5. **Satz.** Die Automatenabbildung $\alpha: X^* \to Y^*$ ist genau dann fast-kommutativ, wenn sie durch einen Zustand eines fast-kommutativen Mealyautomaten $\mathcal{M} = (A, X, Y, \delta, \lambda)$ induziert wird, für den $\mathcal{A} = (A, X, \delta)$ kommutativ ist.

Summary

To every finite automaton $\mathcal{A} = (A, F, \delta)$ we shall define a minimal commutative congruence ϱ_K and a maximal commutative quotient-automaton $\mathcal{A}_K = (A_K, F, \delta_K)$. After investigating some properties of ϱ_K and A_K we shall give conditions for the regularity of the commutative closure of a regular event. Connections between $S(\mathcal{A})$ and $S(\mathcal{A}_K)$, the semigroups of $\mathcal{A}$ and $\mathcal{A}_K$, will be studied. Finally some remarks are made concerning quasi-commutativity of Mealy-automata and automaton-mappings.

INSTITUT FÜR INFORMATIK
DER TU HANNOVER
D-3 HANNOVER
WELFENGARTEN 1

Literatur

[1] BAYER, R., Automorphism groups and quotients of strongly connected automata and monadic algebras, *IEEE Conf. Rec. on Switching and Automata Theory*, 1966, pp. 282—297.
[2] CLIFFORD, A. H. & G. B. PRESTON, *The algebraic theory of semigroups*, Amer. Math. Soc., Providence, R. I., v. 1, 1961.
[3] FLECK, A. C., Isomorphism groups of automata, *J. Assoc. Comput. Mach.*, v. 9, 1962, pp. 469—476.
[4] GÉCSEG, F. & I. PEÁK, *Algebraic theory of automata*, Akadémiai Kiadó, Budapest, 1972.
[5] HARTMANIS, J. & R. E. STEARNS, *Algebraic structure theory of sequential machines*, Englewood Cliffs, N. J., Prentice Hall, 1966.
[6] OEHMKE, R. H., On the structure of an automaton and its input semigroup, *J. Assoc. Comput. Mach.*, v. 10, 1963, pp. 521—525.
[7] RABIN, M. O. & D. SCOTT, Finite automata and their decision problems, *IBM J. Res. Develop.*, v. 3, 1959, pp. 114—125.
[8] TRAUTH, CH. A., Group-type automata, *J. Assoc. Comput. Mach.*, v. 13, 1966, pp. 170—175.

(Eingegangen am 20. Jan. 1976)

A note on optimal performance of page storage

By M. ARATÓ

Introduction

In my earlier papers (see Arató [1], [2]) I have shown that on the basis of the Bayesian approach it is possible to prove that a simple algorithm exists which yields optimal page fault probability rate when the sequence of page references, the so called reference string, forms an independent random sequence with unknown probabilities. With the help of the Bellman equations it was proved that the replacement of the pages depends only on their posterior probabilities. Using this algorithm the expected number of page faults would be minimal.

Benczúr, Krámli and Pergel (see their paper [4] in this volume) pointed out to me that the posterior probabilities depend only on the frequencies of the page references. By the help of their ingenious remark we can prove that under very weak conditions the least frequently used (LFU) algorithm is the optimal one when the reference string is an independent sequence of random variables.

In this paper in a special case I give an elementary proof of the statement of Benczúr, Krámli and Pergel, on the basis of my previous work. The proof will show that the LFU algorithm, which is used in most cases, is the best one. The algorithm means that page should be put on the second level which has the minimal frequency.

The mathematical model I shall use to describe and evaluate the replacement problem of pages is a statistical one on the basis of Bayesian approach. This description seems adequate as it may be used in computer praxis, but we need a proof on the optimality of LFU algorithm without the Bayesian assumption. That this is possible I recall the well known example in the statistical literature, the Wald theorem in the sequential analysis, where the optimality of the likelihood ratio test can be easily proved under the Bayesian conditions (see e.g. Shiryaev [11], De Groot [6]). We have to remark that the "two-armed bandit problem" is meaningless without the Bayesian assumption (see Feldman [8]), but in our case the reference string is independent of the decisions and so the non-Bayesian approach is also allowed and has meaning.

The replacement problems arise in computer system management whenewer the executable memory space available is insufficient to contain all date and programs.

that may be accessed during the execution. An example of this kind of problem is page replacement in virtual memory computers.

In my earlier papers I discussed the problem using the terminology of storage allocation problems. It should be stressed that I consider the problem as being of broader interest (see Easton [7], Casey and Osman [5]). We remember that in virtual memory computer systems a program's address space is divided into equal size blocks called pages. In this paper I consider two-level hierarchies. The first level denotes the faster device, and the backing store represents the larger but slower memory. The first level memory space is divided into page frames each of which may contain a page of some programs. At a given instant of time, not all of a program's pages need reside in the first level memory so that when the program references a page not in the first level, a *page fault* occurs. Supposing that a program's set of pages is $(A_1, A_2, \ldots, A_n)$ and that exactly k of them can be kept in the first level then, if $k<n$, each time a page fault occurs that page is brought into main memory which was demanded and one must be removed from the main memory (demand paging). The purpose of the replacement algorithm is to minimize the average number of page faults. We take as a cost criterion the average number of page faults generated during execution.

In this paper we shall discuss a replacement algorithm in which the main memory is considered full of it contains $k-1$ pages and a new page from the second level is delivered to the k-th place and after delivering the contents of it a page must be removed to the second level.

Any theoretical evaluation of a page replacement algorithm requires a mathematical model of the reference string. Here I shall use for the sequence of requests $\eta_1, \eta_2, \ldots$ the independent identically distributed model

$$P(\eta_t = i) = p_i,$$

where the probability distribution $\{p_i\}$ is unknown. As a first step solving the problem I assume that $p_1 \geqq p_2 \geqq \ldots \geqq p_n$ are known values but the relation is not known between these probabilities and the pages $A_1, \ldots, A_n$. After solving this problem we shall see that the solution, in fact, does not depend on the exact values of the p_i-s.

In all the earlier papers (see Bélády [3], Gelenbe [9], Ingargiola and Korsh [10]) the authors assumed that the probability distribution of the reference string was known and given. The proposed algorithms are depending on the distribution and they are not the exact solutions of the practical problems as the distribution has to be estimated.

Here I do not give propositions for the case when the reference string is a Markov chain with unknown probabilities.

1. A theorem for the case of two pages

In the most elementary case in a program given are two pages A_1, A_2 with request probabilities $p_1 > p_2$ $(=1-p_1)$, but it is not known which probability is related with the first page. The computer is a multiprogrammed one and in the first level memory one page may be kept constantly. In case of a page fault the page of the second level is taken to the main memory and the replacement of one page to the

second level occurs after delivering the contents of the demanded page. A sequence of N references is to be made and at each stage either A_1 or A_2 is on the second level, the loss being 1 if a page fault occurs, 0 otherwise. Let ξ denote the a priori probability that A_1 has the less request probability, p_2.

Let η_t ($t=1, 2, \ldots$) denote the reference string, $\eta_t=i$ ($i=1, 2$) if the i-th page (A_i) was referenced. Let X_t ($t=1, 2, \ldots$) denote the random variable which gives that at time moment t which level was referenced

$$X_t = \begin{cases} 1 & \text{page on the first level was referenced,} \\ 0 & \text{page on the second level was referenced.} \end{cases}$$

Let d_t ($t=0, 1, 2, \ldots$) denote the decision which page has to be removed to level 2

$$d = \begin{cases} 1 & \text{if page } A_1 \text{ goes to level 2,} \\ 2 & \text{if page } A_2 \text{ goes to level 2.} \end{cases}$$

It is obvious that

$$X_t^{(d_{t-1})} = \begin{cases} 1 & \text{if } \eta_t \neq d_{t-1}, \\ 0 & \text{if } \eta_t = d_t. \end{cases}$$

We introduce the non observable random variable w, which gives the relation between the request probabilities (p_1, p_2) and pages A_1, A_2

$$w = \begin{cases} 1 & \text{pages } (A_1, A_2) \text{ having reference probabilities } (p_2, p_1), \\ 2 & \text{pages } (A_1, A_2) \text{ having reference probabilities } (p_1, p_2). \end{cases}$$

The distribution of w is

$$P(w=1)=\xi, \quad P(w=2)=1-\xi.$$

We seek among all Markov decision rules $\delta=(d_0, d_1, \ldots, d_{N-1})$ (see Shiryaev [11]), where d_t depends only on $\eta_t, \eta_{t-1}, \ldots, \eta_1$, such a $\delta^*=(\delta_0^*, \ldots, d_{N-1}^*)$ for which

$$\max_{\delta} E(X_1^{(d_0)}+\ldots+X_N^{(d_{N-1})}) = E(X_1^{(d_0^*)}+\ldots+X_N^{(d_{N-1}^*)}). \tag{1}$$

Simple calculations give that

$$E(X_1^{(1)}) = p_1\xi + p_2(1-\xi),$$

$$E(X_1^{(2)}) = p_2\xi + p_1(1-\xi),$$

and so

$$E(X_1^{(1)}) - E(X_1^{(2)}) = (p_2-p_1)(1-2\xi). \tag{2}$$

From (2) we get that the difference is greater than 0 if $\xi>1/2$ (it does not depend on p_1), this means $d_0=1$ if $\xi>1/2$ and $d_0=2$ if $\xi<1/2$. Now we prove the following lemma.

Lemma 1. Let $\xi=1/2$ and $\xi(t)$ denote the a posteriori probabilities, then

$$\xi(t) = P(w=1\,|\eta_1, \ldots, \eta_t) = \frac{p_1^k p_2^{t-k}}{p_1^k p_2^{t-k} + p_2^k p_1^{t-k}}, \tag{3}$$

where k denotes the number of occurrences of page A_2 (i.e. $\eta_s=2$, $1 \leqq s \leqq t$).

Proof. On the basis of Bayes' theorem we get

$$\xi(1) = \frac{P(\eta_1|w=1)\,P(w=1)}{P(\eta_1|w=1)\,P(w=1)+P(\eta_1|w=2)\,P(w=2)} =$$

$$= \begin{cases} \dfrac{p_1\xi}{p_1\xi+p_2(1-\xi)} & \text{if} \quad \eta_1 = 2, \\ \dfrac{p_2\xi}{p_2\xi+p_1(1-\xi)} & \text{if} \quad \eta_1 = 1, \end{cases}$$

and from here in case $\xi=1/2$

$$\xi(1) = \begin{cases} p_1 & \text{if} \quad \eta_1 = 2, \\ p_2 & \text{if} \quad \eta_1 = 1. \end{cases}$$

In the same way we get ($\xi=1/2$)

$$\xi(2) = \begin{cases} \dfrac{p_1^2}{p_1^2+p_2^2} & \text{if} \quad \eta_1 = 2, \quad \eta_2 = 2, \\ 1/2 & \text{if} \quad \eta_1 = 2, \quad \eta_2 = 1 \quad \text{or} \quad \eta_1 = 1, \quad \eta_2 = 2, \\ \dfrac{p_2^2}{p_1^2+p_2^2} & \text{if} \quad \eta_1 = 1, \quad \eta_2 = 1. \end{cases}$$

By induction the lemma can be easily proved. By the same method as in (2) we can prove that after the first observation η_1 the best decision

$$d_1 = \begin{cases} 1 & \text{if} \quad \xi(1) = p_1 (\text{i.e. } \eta_1 = 2), \\ 2 & \text{if} \quad \xi(1) = p_2 (\text{i.e. } \eta_1 = 1). \end{cases}$$

We prove the following

Theorem. Let $p_1>1/2$, $\xi=1/2$ and N fixed. Let the reference string η_t be an independent, identically distributed sequence of random variables with two states. Then the optimal sequential procedure δ^*, which minimizes the expected number of page faults (see (1)), puts at each stage that page to the second level which has the less request frequency.

Proof. In my paper (Arató [2], Theorem 1) it was proved that the optimal procedure δ^* has the following form (a similar result is known in the "two-armed bandit" problem)

$$d_{t-1}^* = \begin{cases} 1 & \text{if} \quad \xi(t) > 1/2, \\ 2 & \text{if} \quad \xi(t) < 1/2. \end{cases} \tag{4}$$

Comparing (3) and (4) we get (using again the fact $p_1>1/2$)

$$d_{t-1}^* = \begin{cases} 1 & \text{if} \quad k > \dfrac{t}{2}, \\ 2 & \text{if} \quad k < \dfrac{t}{2}, \end{cases}$$

and the theorem is proved.

2. Some generalizations

In the general case when the number of pages $n>2$, their number on the first level $1 \leqq k<n$ and the a priori distribution $\xi=(\xi_1, \ldots, \xi_{n!})$ of the known probabilities $p_1 \geqq p_2 \geqq \ldots \geqq p_n$ are given, the optimal sequential procedure has the same construction as in §1 (see Benczúr, Krámli, Pergel [1]). The proof of their theorem, which is a natural generalization of the "many-armed bandit" problem, is not elementary. Here I recall that there may be two types of the replacement algorithms (see Arató [1]). The first type means that the main memory contains $k-1$ pages of the program and there is one place for the content of a page demanded from the second level. After delivering the content of the new page a page must be removed to the second level. The second type algorithm means that the main memory is full when it contains k pages of the program and if a page is requested from the second level then one page from first level has to be sent to the second level (two pages are changed). If we assume that in the Bayesian approach the a priori distribution ξ is uniform

$$\xi_i = P(w = i) = \frac{1}{n!}, \quad (i = 1, 2, \ldots, n!),$$

which is a natural assumption, we get again that the optimal decision rule is the least frequently used algorithm. The proof is based on the following lemma.

Lemma 2. Let $f_1, f_2, \ldots, f_n$ denote the frequencies of the pages $A_1, \ldots, A_n$ in the independent reference string $\eta_1, \ldots, \eta_t$, then

$$P(w = i|\eta_1, \ldots, \eta_t) = \frac{\prod_{j=1}^{n} p_{(w=i)}^{f_j}}{\sum_{l=1}^{n!} \prod_{j=1}^{n} p_{(w=l)}^{f_j}} \tag{5}$$

The proof of (5) is only a slight extension to that of (3) by the same induction method and so will not be repeated here.

Abstract

An example of the replacement problem in computer system management is the page replacement in virtual memory computers. In this note an elementary proof is given that the "least frequently used" algorithm is the optimal one, using the assumption that the references to the pages are indentically distributed independent random variables with unknown distribution. The general case of this problem is discussed in the paper of Benczúr—Krámli—Pergel.

RESEARCH INSTITUT FOR APPLIED
COMPUTER SCIENCE
H-1536 BUDAPEST, HUNGARY
CSALOGÁNY U. 30—32.

References

[1] Arató, M., Statistical sequential methods in performance evaluation of computer system, 2nd Internat. workshop on modelling and performance evaluation of computer systems, Stresa—Italy, 1976, pp. 1—10.
[2] Arató, M., Számítógépek hierarchikus laptárolási eljárásainak optimalizálásáról, *MTA SZTAKI Közlemények* v. 16, 1976, pp. 7—23.
[3] Bélády, L. A., A study of replacement algorithms for a virtual storage computer. *IBM Systems J.* v. 5, 1966, pp. 78—101.
[4] Benczúr, A., A. Krámli, J. Pergel, On the Bayesian approach to optimal performance of page storage hierarchies, *Acta Cybernet.*, v. 3, 1976.
[5] Casey, R. G., I. Osman, Generalized page replacement algorithms in a relational data base, ACM SIGMOD, Workshop on data description, 1974, pp. 101—124.
[6] De Groot, M. H., *Optimal statistical decisions*, Mc Graw-Hill, N. Y., 1970.
[7] Easton, M. C., Model for interactive data base reference string, *IBM J. Res. Develop*, v. 19, 1975, pp. 550—556.
[8] Feldman, D., Contributions to the "two-armed bandit" problem, *Ann. Math. Statist.*, v. 33, 1962, pp. 847—856.
[9] Gelenbe, E., A unified approach to the evaluation of a class of replacement algorithms, *IEEE Trans. Computers*, v. C-22, 1973, pp. 611—617.
[10] Ingargiola, G., J. F. Korsh, Finding optimal demand paging algorithms, *J. Assoc. Comput. Mach.*, v. 21, 1974, pp. 40—53.
[11] Shiryaev, A. N., *Statistical sequential analysis*, (in Russian), Nauka, Moscow, 1969.

(Received March 11, 1976)

A language for Markov's algorithms composition

By G. GERMANO and A. MAGGIOLO—SCHETTINI

The present paper gives an improvement of [2]. There, after having noted that Markov's normal algorithms cannot be composed immediately like flow-charts, the authors presented four operations on Markov's normal algorithms without concluding formulas which helped to overcome the difficulty; among these operations there were the analogs of **if** and **goto**. Here other operations are given: the analogs of **if** and **goto** are substituted by the analog of **while**. This allows to use only one alphabet for output strings, whereas in [2] infinitely many alphabets are used (in the sense that two different algorithms might have as output alphabets A^i and A^j respectively with $i \neq j$), and to use a simpler definition of computability.

Furthermore an Algol-like language L is given which is interpreted into Markov's normal algorithms without concluding formulas via the new operations defined for composing them. So the statements of L come out to be names of Markov's normal algorithms and it is immediate to pass from traditional programming to algorithms. As a practical motivation, we have already mentioned the fact that this work gives the possibility of writing Markov's algorithms along the familiar patterns of computer programming. As a theoretical motivation, after having recalled the known relationship between programming and combinatory logic, we offer the following quotation from H.B. Curry [1]:

"Altough it is well known that any partial recursive numerical function can be represented in combinatory logic... and thus, by Church's thesis, any effective process can be so represented via the detour of Gödel representation, yet there is some interest in a direct representation, not involving this detour, of certain processes, like... Markov algorithms".

§ 1. Operations on algorithms

We will use alphabets $\{\star_i, |_i\}$ with $1 \leqq i \leqq \omega$ and will consider algorithms which transform words in the alphabet $\{\star_1, |_1\}$ into words in some alphabet $\bigcup_{i \in I} \{\star_i, |_i\}$ (where I is finite for each algorithm) and eventually naturally transform such words into words in the alphabet $\{\star_\omega, |_\omega\}$ (see [3] for the notions of "transforms" and "naturally transforms"). We will use "x_i" and "y_i" to denote letters in the alphabet

$\{\curlywedge_i, |_i\}$. Analogously to [2], we will use the following translations for words in the alphabets above:

$$\curlywedge_i^{T_j} := \curlywedge_{i+j} \qquad |_i^{T_j} := |_{i+j}$$
$$\curlywedge_\omega^{T_j} := \curlywedge_\omega \qquad |_\omega^{T_j} := |_\omega$$
$$\curlywedge_i^{\tau_j} := \curlywedge_i \qquad |_i^{\tau_j} := |_i$$
$$\curlywedge_\omega^{\tau_j} := \curlywedge_j \qquad |_\omega^{\tau_j} := |_j$$

These translations are extended to words in the usual way and we will write P_j to mean that each letter in P is indexed by j. Algoritmhs are translated word by word.

We introduce three operations for composing algorithms, namely *juxtaposition, connection* and *controlled repetition* and prove the relative theorems.

1.1 **Juxtaposition.** The juxtaposition of the algorithms $\mathfrak{A}$ and $\mathfrak{B}$ is the algorithm

$$\begin{cases} x_1 \to x_2 x_{m+3} \\ y_{m+3} x_2 \to x_2 y_{m+3} \\ (\mathfrak{A}^{T_1})^{\tau_{m+2}} \\ (\mathfrak{B}^{T_{m+2}})^{\tau_{m+n+3}} \\ x_{m+2} \to x_\omega \\ x_{m+n+3} \to x_\omega \end{cases}$$

where $m=$maxind $\mathfrak{A}$ and $n=$maxind $\mathfrak{B}$ (maxind $\mathfrak{A}$ is defined as the highest index $i<\omega$ occurring in $\mathfrak{A}$, see [2]).

Theorem. If $\mathfrak{C}$ is the juxtaposition of $\mathfrak{A}$ and $\mathfrak{B}$ then for every $P\in\{\curlywedge_1, |_1\}^*$ it holds that

$$\mathfrak{C}(P) \cong \mathfrak{A}(P)\,\mathfrak{B}(P).$$

The proof is immediate (see [2] p. 305 for analogy).

1.2 **Controlled repetition.** The *repetition* of the algorithm $\mathfrak{B}$ *controlled* by the algorithm $\mathfrak{A}$ is the algorithm

$$\begin{cases} x_1 y_{m+n+3} \to x_1 y_1 & (1) \\ x_1 \to x_2 x_{m+3} & (2) \\ y_{m+3} x_2 \to x_2 y_{m+3} & (3) \\ (\mathfrak{A}^{T_1})^{\tau_{m+2}} & (4) \\ \curlywedge_{m+2} x_{m+2} y_{m+2} \to \curlywedge_{m+2} x_{m+2} & (5) \\ \curlywedge_{m+2} x_{m+2} y_{m+3} \to y_{m+3} & (6) \\ \curlywedge_{m+2} x_{m+3} \to x_\omega \curlywedge_{m+2} & (7) \\ \curlywedge_{m+2} \to & (8) \\ (\mathfrak{B}^{T_{m+2}})^{\tau_{m+n+3}} & (9) \\ \curlywedge_{m+n+3} \to \curlywedge_1 & (10) \end{cases}$$

where $m=$maxind $\mathfrak{A}$ and $n=$maxind $\mathfrak{B}$.

Theorem. If $\mathfrak{C}$ is the repetition of $\mathfrak{B}$ controlled by $\mathfrak{A}$ then for every $P \in \{\curlywedge_1, |_1\}^*$ it holds that

$$\mathfrak{C}(P) = \mathfrak{B}^j(P)$$

if j is the least (non negative) integer such that $\mathfrak{A}(\mathfrak{B}^j(P)) = \curlywedge_\omega$ whereas

$$\mathfrak{C}(P) \text{ is undefined}$$

if no such j exists.

Proof. **I.** If for i with $0 \leqq i < j$ $\mathfrak{A}(\mathfrak{B}^i(P)) \neq \curlywedge_\omega$ then it holds that

$$\begin{aligned}
&\mathfrak{C} : P \\
&\models\ P_2 P_{m+3} && \text{by (2), (3)} \\
&\models\ (\mathfrak{A}(P))_{m+2} P_{m+3} && \text{by (4)} \\
&\models\ P_{m+3} && \text{by (5), (6)} \\
&\models\ (\mathfrak{B}(P))_{m+n+3} && \text{by (9)} \\
&\models\ (\mathfrak{B}(P))_1 && \text{by (10), (1)} \\
&\quad \cdots \\
&\models\ (\mathfrak{B}^j(P))_1 .
\end{aligned}$$

II. If $\mathfrak{A}(P) = \curlywedge_\omega$ then it holds that

$$\begin{aligned}
&\mathfrak{C} : P \\
&\models\ P_2 P_{m+3} && \text{by (2), (3)} \\
&\models\ \curlywedge_{m+2} P_{m+3} && \text{by (4)} \\
&\models\ P_\omega \curlywedge_{m+2} && \text{by (7)} \\
&\models\ P_\omega \quad \neg && \text{by (8).}
\end{aligned}$$

From I and II the thesis of the theorem follows immediately.

1.3. **Connection.** The *connection* of the algorithms $\mathfrak{A}$ and $\mathfrak{B}$ is the algorithm

$$\begin{cases} \mathfrak{A}^{\tau_{m+1}} \\ \mathfrak{B}^{T_m} \end{cases}$$

where $m = \text{maxind } \mathfrak{A}$.

Theorem. If $\mathfrak{C}$ is the connection of $\mathfrak{A}$ and $\mathfrak{B}$ then for every $P \in \{\curlywedge_1, |_1\}^*$ it holds that

$$\mathfrak{C}(P) \cong \mathfrak{B}(\mathfrak{A}(P)).$$

The proof is immediate (see [2] p. 304 for analogy).

§ 2. The language

The syntactic definition of the language L is

⟨initial statement⟩ := **zero**|**succ**|**pred**

⟨projection statement⟩ := **proj i**|⟨projection statement⟩**i**

⟨statement⟩ := ⟨initial statement⟩|⟨projection statement⟩|
(⟨statement⟩,⟨statement⟩)|
while⟨statement⟩**do**(⟨statement⟩|
(⟨statement⟩; ⟨statement⟩)

For short we will write $\mathbf{proj}^{(j)}$ instead of $\mathbf{proj}\ \overbrace{\mathbf{i} \ldots \mathbf{i}}^{j}$. As variables for statements we will use $\mathfrak{S}$ and $\mathfrak{T}$.

We define now an interpretation $\mathscr{I}$ of L into Markov's normal algorithms without concluding formulas by induction on the syntactic definition of L:

$$\mathscr{I}(\mathbf{zero}) := \{ \ast_1 \to \ast_\omega$$

$$\mathscr{I}(\mathbf{succ}) := \begin{cases} \ast_1 \to \ast_\omega|_\omega \\ |_1 \to |_\omega \end{cases}$$

$$\mathscr{I}(\mathbf{pred}) := \begin{cases} \ast_1|_1 \to \ast_\omega \\ \ast_1 \to \ast_\omega \\ |_1 \to |_\omega \end{cases}$$

$$\mathscr{I}(\mathbf{proj}^{(j)}) := \begin{cases} \ast_\omega \ast_1|_1 \to \ast_\omega \ast_1 \\ |_\omega \ast_1|_1 \to |_\omega \ast_1 \\ \ast_\omega \ast_1 \to \ast_\omega \\ |_\omega \ast_1 \to |_\omega \\ \ast_\omega|_1 \to \ast_\omega|_\omega \\ |_\omega|_1 \to |_\omega|_\omega \\ \ast_2^j \to \ast_\omega \\ \ast_2 \ast_1 \to \ast_2 \ast_2 \\ \ast_2|_1 \to \ast_2 \\ \ast_1 \to \ast_2 \end{cases}$$

$\mathscr{I}((\mathfrak{S}, \mathfrak{T}))$ is defined as the juxtaposition of $\mathscr{I}(\mathfrak{S})$ and $\mathscr{I}(\mathfrak{T})$. $\mathscr{I}(\mathbf{while}\ \mathfrak{S}\ \mathbf{do}\ \mathfrak{T})$ is defined as the repetition of $\mathscr{I}(\mathfrak{T})$ controlled by $\mathscr{I}(\mathfrak{S})$. $\mathscr{I}((\mathfrak{S}; \mathfrak{T}))$ is defined as the connection of $\mathscr{I}(\mathfrak{S})$ and $\mathscr{I}(\mathfrak{T})$.

§ 3. Application

We say that a function f is *computable* by $\mathfrak{S}$ (relatively to the input alphabet $\{\curlywedge_1, |_1\}$ and to the output alphabet $\{\curlywedge_\omega, |_\omega\}$) if and only if

$$\mathfrak{S}: \curlywedge_1 |_1^{n_1} \dots \curlywedge_1 |_1^{n_k} \vDash \curlywedge_\omega |_\omega^{f(n_1, \dots, n_k)} \neg.$$

As concerns computability of partial recursive functions (characterized as in [4]) it is immediate to see how to write programs for initial functions and concerning substitution. We give the programs concerning recursion scheme and μ-operator.

Let the function g be computable by $\mathfrak{S}$, the function h be computable by $\mathfrak{T}$ and

$$\begin{cases} f(0, y) \cong g(y) \\ f(S(x), y) \cong h(y, f(x, y)) \end{cases}$$

Then the function f is computable by

$$((((\mathbf{proj}^{(1)}, \mathbf{proj}^{(2)}), (\mathbf{proj}^{(2)}; \mathfrak{S}));$$
$$\mathbf{while}\ \mathbf{proj}^{(1)}\ \mathbf{do}\ (((\mathbf{proj}^{(1)}; \mathbf{pred}), \mathbf{proj}^{(2)}), ((\mathbf{proj}^{(2)}, \mathbf{proj}^{(3)}); \mathfrak{T})); \mathbf{proj}^{(3)}).$$

Let the function g be computable by $\mathfrak{S}$ and

$$f(x_1, \dots, x_k) \cong \mu x(g(x, x_1, \dots, x_k) = 0)$$

Then the function f is computable by

$$(((\dots((\mathbf{zero}, \mathbf{proj}^{(1)}), \dots, \mathbf{proj}^{(k)});$$
$$\mathbf{while}\ \mathfrak{S}\ \mathbf{do}\ (\dots((\mathbf{proj}^{(1)}; \mathbf{succ}), \mathbf{proj}^{(2)}), \dots, \mathbf{proj}^{(k+1)})); \mathbf{proj}^{(1)}).$$

So we may conclude that every partial recursive function is computable by Markov's normal algorithms without concluding formulas relatively to the input alphabet $\{\curlywedge_1, |_1\}$ and to the output alphabet $\{\curlywedge_\omega, |_\omega\}$.

Abstract

A programming language is introduced to whose statements Markov's algorithms univocally correspond via the operations of juxtaposition, connection and controlled repetition. Avoiding goto statements allows to use only one output alphabet.

LABORATORIO DI CIBERNETICA
DEL C. N. R.
80072 ARCO FELICE, ITALY

INSTITUTO DI SCIENZE DELL'INFORMAZIONE
DELL'UNIVERSITA, VIA VERNIERI 42
84100 SALERNO, ITALY

References

[1] CURRY, H. B., Representation of Markov algorithms by combinators, *Notices Amer. Math. Soc.*, v. 20, A-590, 1973.

[2] GERMANO, G. & A. MAGGIOLO-SCHETTINI, A flow diagram composition of Markov's normal algorithms without concluding formulas, *BIT*, v. 13, 1973, pp. 301—312.

[3] MARKOV, A. A., Teoria algoritmov (Russian), *Trudy Math. Inst. Steklov.*, v. 42, 1954.

[4] ROBINSON, R. M., Primitive recursive functions, *Bull. Amer. Math. Soc.*, v. 53, 1947, pp. 925—942.

(Received Oct. 7, 1974)

On minimal R-complete systems of finite automata

By P. DÖMÖSI

To the memory of Professor L. Kalmár

From papers by F. GÉCSEG (see [1], [2]) it is known, that there exist neither finite homomorphically, nor minimal isomorphically R-complete systems of finite automata. In the book by F. GÉCSEG and I. PEÁK [3] it is mentioned as an unsolved problem whether or not there exists a minimal homomorphically R-complete system of finite automata.

In this paper we prove that the answer to this problem is in the affirmative. Namely, it is shown that there exists a minimal homomorphically R-complete system of finite automata. Moreover, we prove that there exists a homomorphically R-complete system of finite automata which does not contain any minimal subsystem.

Before proving our statements, we introduce some notions and notations. Take an arbitrary, finite partially ordered set $R=\langle 1, 2, \ldots, n\rangle$ of indices, and for every i $(=1, 2, \ldots, n)$ let an automaton $\mathbf{A}_i=\mathbf{A}_i(X_i, A_i, Y_i, \delta_i, \lambda_i)$ be given. Suppose that for an automaton $\mathbf{A}=\mathbf{A}(X, A, Y, \delta, \lambda)$ with state set $A=A_1\times A_2\times\ldots\times A_n$ the functions $\varphi: A_1\times A_2\times\ldots\times A_n\times X\to X_1\times X_2\times\ldots\times X_n$, $\psi: A_1\times A_2\times\ldots\times A_n\times X\to Y$ are given. Then $\mathbf{A}=\prod_{i=1}^{n}\mathbf{A}_i[X, Y, \varphi, \psi]$ is called a loop-free or *R-product* of the automata $\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_n$, if the conditions $\delta((a_1, a_2, \ldots, a_n), x)=(\delta_1(a_1, x_1), \delta_2(a_2, x_2), \ldots, \delta_n(a_n, x_n))$, $\lambda((a_1, a_2, \ldots, a_n), x)=\psi(a_1, a_2, \ldots, a_n, x)$ hold for arbitrary $(a_1, a_2, \ldots, a_n)\in A$ and $x\in X$, where $(x_1, x_2, \ldots, x_n)=\varphi(a_1, a_2, \ldots, a_n, x)$; moreover $\varphi(a_1, a_2, \ldots, a_n, x)=$ $=(\varphi_1(a_1, a_2, \ldots, a_n, x), \varphi_2(a_1, a_2, \ldots, a_n, x), \ldots, \varphi_n(a_1, a_2, \ldots, a_n, x))$ holds as well, where φ_i $(i=1, 2, \ldots, n)$ is independent of states having indices not less (in the original definition not greater) than i under the partial ordering R. The functions φ and ψ of the R- product are called *feedback function* and *output function,* respectively.

If in the considered R-product $\mathbf{A}$ the set R is completely ordered, then $\mathbf{A}$ is called a *quasi-superposition* of $\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_n$.

Let $\mathbf{A}_1=\mathbf{A}_1(X_1, A_1, Y_1, \delta_1, \lambda_1)$ and $\mathbf{A}_2=\mathbf{A}_2(X_2, A_2, Y_2, \delta_2, \lambda_2)$ be arbitrary automata, where $Y_1\subseteq X_2$. Then a quasi-superposition $\mathbf{A}=\prod_{i=1}^{2}\mathbf{A}_i[X_1, Y_2, \varphi, \psi]$ of $\mathbf{A}_1$ and $\mathbf{A}_2$, where $\varphi(a_1, a_2, x)=(x, \lambda_1(a_1, x))$, $\psi(a_1, a_2, x)=\lambda_2(a_2, \lambda_1(a_1, x))$ are for any

$a_1 \in A_1$, $a_2 \in A_2$ and $x \in X_1$, is said to be the *superposition of* $\mathbf{A}_1$ by $\mathbf{A}_2$. The superposition can naturally be generalized for an arbitrary finite system of automata $\mathbf{A}_i = \mathbf{A}_i(X_i, A_i, Y_i, \delta_i, \lambda_i)$ $(i=1, 2, \ldots, n)$ with $Y_j = X_{j+1}$ $(j=1, 2, \ldots, n-1)$.

A system $\mathfrak{A}$ of finite automata is called homomorphically (isomorphically) *R-complete,* if for every given finite automaton $\mathbf{A}$ there exists a finite *R*- product $\mathbf{B}$ of automata from $\mathfrak{A}$, such that an *A*-subautomaton of $\mathbf{B}$ can be mapped *A*-homomorphically (*A*-isomorphically) onto $\mathbf{A}$. $\mathfrak{A}$ is a *minimal* (homomorphically or isomorphically) *R*-complete system if for arbitrary $\mathbf{C} \in \mathfrak{A}$ the system $\mathfrak{A}/\langle \mathbf{C} \rangle$ is not (homomorphically or isomorphically) *R*-complete.

Then the following theorem holds(*).

Theorem 1. *There exists a minimal homomorphically R-complete system of finite automata.*

Proof. Denote by Γ a system of finite automata, where the elements of Γ are pair-wise not isomorphic, and simultaneously for every finite automaton $\mathbf{A}$ there exists an element $\mathbf{B}$ of Γ, such that $\mathbf{A}$ is isomorphic to $\mathbf{B}$. It can easily be seen, that Γ is enumerable. Take an arrangement $\Gamma = \langle \mathbf{A}_i(X_i, A_i, Y_i, \delta_i', \lambda_i') | i=1, 2, \ldots \rangle$ of the (enumerable) set Γ.

Let $p_0, p_1, \ldots, p_n, \ldots$ be an infinite sequence of prim numbers, where $p_0 \geqq 2$, $p_1 > p_0$, and for every further p_j $(j=2, 3, \ldots)$, $p_j > p_{j-1} + p_0 \cdot p_1 \cdot \ldots \cdot p_{j-2} \cdot \overline{A}_{j-1}$ holds.

Give the elements of automaton-system $\mathcal{A} = \langle \mathbf{B}_0, \mathbf{B}_1, \ldots, \mathbf{B}_n, \ldots \rangle$ as follows: $\mathbf{B}_0 = \mathbf{B}_0(X_0, D_0, Y_0, \delta_0, \lambda_0)$ is an arbitrary automaton, such that $D_0 = \langle 1, 2, \ldots, p_0 \rangle$, furthermore for any pair $u \in D_0$, $x \in X_0$

$$\delta_0(u, x) = \begin{cases} u+1, & \text{if} \quad 1 \leqq u < p, \\ 1, & \text{if} \quad u = p_0. \end{cases}$$

For every further $\mathbf{B}_i$ $(i=1, 2, \ldots)$ let $\mathbf{B}_i = \mathbf{B}_i(C_i \times X_i, D_i \cup C_i \times A_i, Y_i', \delta_i, \lambda_i)$ be, where Y_i' is an arbitrary nonempty and finite set,

$$C_i = \langle 1, 2, \ldots, p_0 \cdot p_1 \cdot \ldots \cdot p_{i-1} \rangle, \tag{1}$$

$$D_i = \langle 1, 2, \ldots, p_i \rangle, \tag{2}$$

and $\lambda_i : (D_i \cup C_i \times A_i) \times C_i \times X_i \to Y_i'$ is arbitrary function, moreover for every triple $s \in D_i$, $(u, a) \in C_i \times A_i$, $(r, x) \in C_i \times X_i$

$$\delta_i(s, (r, x)) = \begin{cases} s+1, & \text{if} \quad 1 \leqq s < p_i, \\ 1, & \text{if} \quad s = p_i, \end{cases} \tag{3}$$

$$\delta_i((u, a), (r, x)) = \begin{cases} (u+1, \delta_i'(a, x)), & \text{if} \quad r = u \quad \text{and} \quad 1 \leqq u < p_0 \cdot p_1 \cdot \ldots \cdot p_{i-1}, \\ (1, \delta_i'(a, x)), & \text{if} \quad r = u \quad \text{and} \quad u = p_0 \cdot p_1 \cdot \ldots \cdot p_{i-1}, \\ 1 (\in D_i), & \text{if} \quad r \neq u. \end{cases} \tag{4}$$

(*) The proof of Theorem 1 is based on an idea of F. Gécseg.

First we prove that Δ is homomorphically R-complete system of finite automata.

Take an arbitrary finite automaton $\mathbf{A}=\mathbf{A}(X, A, Y, \delta, \lambda)$, and let $\langle \Psi_1, \Psi_2, \Psi_3\rangle$ denote an isomorphism of $\mathbf{A}$ onto a suitable element $\mathbf{A}_i$ in Γ. Let the automata $\mathbf{C}_i=\mathbf{C}_i(X, C_i, C_i\times X_i, \delta_i'', \lambda_i'')$, $\mathbf{B}_i'=\mathbf{B}_i'(C_i\times X_i, D_i\cup C_i\times A_i, Y, \delta_i, \lambda_i^*)$ be constructed in the following way:

For any $r\in C_i$, $x\in X$, $s\in D_i$, $(u, a)\in C_i\times A_i$,

$$\delta_i''(r, x)=\begin{cases} r+1, & \text{if} \quad 1\leqq r\prec p_0\cdot p_1\cdot\ldots\cdot p_{i-1},\\ 1, & \text{if} \quad r=p_0\cdot p_1\cdot\ldots\cdot p_{i-1},\end{cases} \tag{5}$$

$$\lambda_i''(r, x)=\big(r, \Psi_1(x)\big); \tag{6}$$

let $\lambda_i^*\big(s, (r, \Psi_1(x))\big)$ be an arbitrary element in Y given unambigously,

$$\lambda_i^*\big((u, a), (r, \Psi_1(x))\big)=\begin{cases} \Psi_3^{-1}\big(\lambda_i'(a, \Psi_1(x))\big), & \text{if} \quad r=u\\ \text{arbitrary element in } Y \text{ given}\\ \text{unambigously, otherwise.}\end{cases} \tag{7}$$

From the above constructions it is evident that the superposition $\mathbf{C}_i*\mathbf{B}_i'$ of $\mathbf{C}_i$ by $\mathbf{B}_i'$ exists. On the other hand, using (4), (5) and (6), it can easily be proved that there is an A-subautomaton of $\mathbf{C}_i*\mathbf{B}_i'$ with set of states $B=\langle (u, u, a)|u\in C_i, a\in A_i\rangle$.

Consider the mapping Ψ_2': $B\to A$ given as follows:

For every state $(u, u, a)\in B$ let $\Psi_2'\big((u, u, a)\big)=\Psi_2^{-1}(a)$. From constructions (4)—(7) it can be seen that Ψ_2' is an A-homomorphism of the A-subautomaton of $\mathbf{C}_i*\mathbf{B}_i'$ with set of states B onto $\mathbf{A}$. On the other hand, using (2) and (3), it is not difficult to prove that $\mathbf{C}_i$ can be represented as an A-subautomaton of a quasi-superposition of automata $\mathbf{B}_0, \mathbf{B}_1, \ldots, \mathbf{B}_{i-1}$. So in consequence of construction $\mathbf{B}_i'$, the superposition $\mathbf{C}_i*\mathbf{B}_i'$ is an A-subautomaton of a quasi-superposition of $\mathbf{B}_0, \mathbf{B}_1, \ldots, \mathbf{B}_i$. Since $\mathbf{A}$ is arbitrary chosen, Δ is a homomorphically R-complete system of finite automata.

Let us prove that Δ is minimal, i.e. in case of any $\mathbf{B}_i\in\Delta$ the system $\Delta\setminus\langle \mathbf{B}_i\rangle$ is not homomorphically R-complete. To this we shall show, that no R-product of elements in $\Delta\setminus\langle \mathbf{B}_i\rangle$ has any A-subautomaton which can be mapped A-homomorphically onto $\mathbf{B}_i$.

Suppose that contrary to our assumption such R-product there exists. Denote by $\langle \Psi_1, \Psi_2, \Psi_3\rangle$ a homomorphism of an A-subautomaton of this R-product onto $\mathbf{B}_i$, moreover, let $(e_1, e_2, \ldots, e_m)$ be a state of this A-subautomaton such that $\Psi_2\big((e_1, e_2, \ldots, e_m)\big)=s(\in D_i)$.

From (3) it is evident that

$$s\cdot q=s\Leftrightarrow p_i\Big|q| \quad (q\in F(C_i\times X_i)). \tag{8}$$

Also from (3) and $\Psi_2\big((e_1, e_2, \ldots, e_m)\big)\in D_i$ it can be supposed that for a suitable

element x of $C_i \times X_i$ the

$$(e_1, e_2, \ldots, e_m) \cdot x^l = (e_1, e_2, \ldots, e_m) \tag{9}$$

holds, where l is an appropriate natural number. Thus, due to (8), $p_i | l$ also holds. Suppose that the l is minimal among all numbers satisfying (9). For every $i (= 1, 2, \ldots, m)$ let l_i be a minimal natural number for which $(e_1, e_2, \ldots, e_i) \cdot x^{l_i} = (e_1, e_2, \ldots, e_i)$ holds, moreover, let φ_i be the i th function-component of the feedback function of the R-product in question. Finally, let $\mathbf{M}_i$ be the i th component-automaton in our R-product.

Suppose that $\mathbf{M}_1 = \mathbf{B}_j (\in \Delta)$. In this case, refering to the equalities $\varphi_1(e_1 \cdot \varphi_1(e_1, e_2, \ldots, e_m, x), x) = \varphi_1((e_1, e_2, \ldots, e_m) \cdot x, x)$, and (4), either $\mathbf{M}_1 = \mathbf{B}_0$, or $e_1 \cdot \varphi_1(e_1, e_2, \ldots, e_m, x) \varphi_1((e_1, e_2, \ldots, e_m) \cdot x, x) \in D_j$ holds. Then, because of (3) and (4), equality (9) holds only in case $e_1 \in D_j$. Hence $l_1 \in \langle p_0, p_1, p_{i-1}, p_{i+1}, p_{i+2}, \ldots \rangle$ that is $p_i \nmid l_1$. If $\mathbf{M}_2$ in the R-product is independent of $\mathbf{M}_1$, $p_i \nmid l_2$ similarly holds. Othervise there are two possible cases.

(a) The number of states in $\mathbf{M}_2$ is less than that in $\mathbf{B}_i$. Hence for arbitrary input word q of $\mathbf{M}_2$ the number of pairly different states from the series $e_2, e_2 \cdot q, e_2 \cdot q^2, \ldots, e_2 \cdot q^s, \ldots$ is less than p_i (see the construction of $\langle p_0, p_1, \ldots \rangle$). Namely, if by the effect of e_1 and x^{l_1} the input word q is given to $\mathbf{M}_2$, then $p_i \nmid l_2$ since $l_2 = l_1 t$, where t is a natural number with $t < p_i$.

(b) The number of states in $\mathbf{M}_2$ is greater than that in $\mathbf{B}_i$. Suppose that by the effect of e_1 and x^{l_1} the input word q is given to $\mathbf{M}_2$. In this case for every natural number k by the effect of e_1 and $x^{k \cdot l_1}$ the automaton $\mathbf{M}_2$ in state e_2 has the input word q^k and $p_i \nmid |q|$. Suppose that $\mathbf{M}_2 = \mathbf{B}_h (\in \Delta, h > i)$ and $e_2 = (s, a) (\in C_h \times A_h)$. Because of (1) and (4), $e_2 \cdot q \notin \langle s \rangle \times A_h$. Therefore, by (4), for any $k (\geqq 1)$ we have $e_2 \cdot q \notin C_h \times A_h$. Thus $e_2 \cdot q^k \in D_h$, which, by (9) and (3), means that $e_2 \in D_h$. Consequently, taking into considerations the minimality of l_2, by (8) we get $l_2 = [l_1, p_j]$, where $[m, n]$ denotes the least common multiple of m and n. Therefore, $p_i \nmid l_2$ holds as well.

Repeating our procedure for the components $e_3, e_4, \ldots, e_m$, finally we get that $p_i \nmid l_m$. Since $l = l_m$ holds *per definitionem*, thus $p_i \nmid l$. Therefore, by (8), $\Psi_2((e_1, e_2, \ldots, e_m)) \notin D_i$. Thus none of the A-subautomaton of the considered R-product can be mapped A-homomorphically onto the A-subautomaton of $\mathbf{B}_i$ with the set of states D_i. Consequently, it also cannot be mapped A-homomorphically onto $\mathbf{B}_i$. Hence the system Δ is minimal, which ends the proof of Theorem 1.

Finally we prove

Theorem 2. *There exists a homomorphically R-complete system of finite automata which does not contain any minimal homomorphically R-complete subsystem.*

Proof. Again let $\Gamma = \langle \mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_n, \ldots \rangle$ denote a system of finite automata such that the elments of Γ are pairly not isomorphic and for every finite automaton $\mathbf{A}$ there exists an element $\mathbf{B}$ of Γ which is isomorphic to $\mathbf{A}$. Now let us take the system $\Delta = \langle \mathbf{B}_1, \mathbf{B}_2, \ldots, \mathbf{B}_n, \ldots \rangle$ where for arbitrary $i (= 1, 2, \ldots)$ every automaton $\mathbf{A}_j (j = 1, 2, \ldots, i)$ is a subautomaton of $\mathbf{B}_i$.

It can easily be seen that Δ is homomorphically R-complete system of finite automata. By a result of F. GÉCSEG [1], no finite subset of Δ is homomorphically R-complete.

Denote by Ω an infinite subset of Λ. It is evident that for every natural number i there is a j with $j \geqq i$ such that $\mathbf{B}_j \in \Lambda \cap \Omega$. Since every $\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_j \in \Gamma$ is a subautomaton of $\mathbf{B}_j$, thus Ω is also homomorphically R-complete. It is obvious that Ω is not minimal, which completes the proof of Theorem 2.

THE BOULDING SOCIETY OF
SOUTHERN HUNGARY /DÉLÉP/
H-6721 SZEGED, HUNGARY
BOCSKAI U. 10—12.

References

[1] GÉCSEG, F., О композиции автоматов без петель, *Acta Sci. Math.* (Szeged), v. 26, 1965, pp. 269—272.
[2] GÉCSEG, F., On complete systems of automata, *Acta Sci. Math.* (Szeged), v. 30, 1969, pp. 295—300.
[3] GÉCSEG, F. & I. PEÁK, *Algebraic Theory of automata*, Disquisitiones Matematicae, Akadémiai Kiadó, Budapest, 1972.

(Received May 14, 1975)

Homogeneous event indexes

By F. FEIND, E. KNUTH, P. RADÓ, J. VARSÁNYI

1. Discrete event simulation

General purpose discrete simulation languages are based on the so-called "event notice" concept. It means special data patterns assigned to each simulation event and handled by the run-time timing routines of the systems.

These routines have the following main functions:

1) scheduling future events (generated in the course of program execution);
2) registering the events in a properly linked order so as to be able to produce the "next event" in any case.

We assume in this paper that whenever an event is scheduled its event time is always known. The examination of more general, e.g. conditional scheduling possibilities would lead to much more complicated structures. Therefore we can assume that definite time values are assigned to each event notice and they are necessarily ordered according to their time values.

We are not dealing with the problems of multiple schedulations into the same time point, which may be a question of disciplines or priorites but has no importance as to the performance of the algorithms we do.

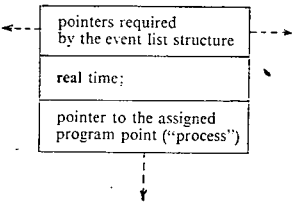


Fig. 1. General structure of an event notice

Now discrete event simulation works as follows: at the initiation and during the whole execution event notices are generated and inserted into the event list for all arisen simulation activity demanding a definit timing. The program execution is controlled by the event list i.e. having finished an activity assigned to an event notice the activity corresponding to the next one is going to be carried out according to the instantaneous state of the list structure. (More detailed descriptions can be found in references [1], [2] and [3].)

2. Event list algorithms

The functional activities of event list algorithms may be comprised by the following four procedures:

1. scan (t)

The procedure finds the event after which a new event will have to be inserted if its event time value is t. Thus using the SIMULA formalism [4] the procedure specification is

ref (event) **procedure** scan (t); **real** t;

or according to the PASCAL formalism [7]

function scan (t:real): event;

where we denoted the data structure "event notice" simply by "event".

2. insert (E, P, t)

E is the event after which the insertion must be done. P is the simulation activity having to be timed. The procedure must generate a new event notice of time value t and insert it after E referring to the simulation activity P.

The formal spedification is

procedure insert (E, P, t); **ref** (event) E; **ref** (process) P; **real** t;

or

procedure insert (E:event, P:process, t:real);

and the most typical call of the procedure is insert (scan (t), P, t);.

3. delete (E)

This procedure must delete the event E from the event list preserving the correctness of the remaining list. Formally:

procedure delete (E); **ref** (event) E;

or

procedure delete (E: event);

4. delete current

This procedure is equivalent to the call *delete* (current); where "current" is always the first event of the list. For a deletion implied by the termination of any simultation activity is always related to the current event, it is worth doing to develop this procedure in a more special way than the previous, general one.

3. Linear list structure

The simplest structure, the linear list is widely used in simulation languages including SIMULA [4], SIMSCRIPT [5], GPSS [6].

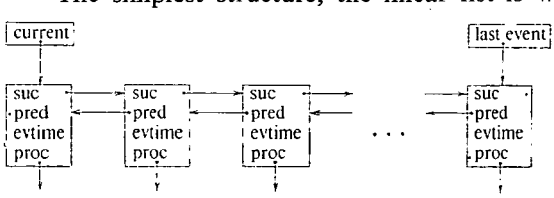


Fig. 2. Structure of the linear list

Suc and *pred* are the linking pointers. *Evtime* is the time value assigned to the notices. If E is an event notice then the relation

$$E.suc.evtime \geqq E.evtime^*$$

must always be satisfied. The pointers *current* and *last event* always point to the first and last events respectively.

The event list algorithms for linear lists are very simple:

1. scan (t)

Begin to compare t with the values *evtime* from the *last event* and follow it sequentially while *evtime*$>t$ holds. (This simple algorithm is described in the Appendix in detail.) The reason of scanning from the end of the list is its better performance.

2. insert (E, P, t)

The procedure is to perform a usual insertion into a two-way list as it is shown in Fig. 3 and in formal way in the Appendix.

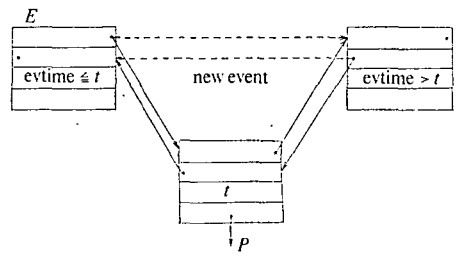


Fig. 3. Insertion into a linear list

3—4. delete (E)

Now a deletion simply means resetting the pointers having been set by the insertion Fig. 3 (see Appendix). The special procedure "delete current" need not be done in different way.

4. Further structures proposed

With the linear list structure, the overhead time taken by a call of the procedure scan is proportianal to N, the number of events in the list. (It leads to the amount of scheduling overhead proportional to N^2.)

Myhrhaug [8] gave the first results to improve it replacing the linear list by binary tree structures. Knuth [9,p. 150] also gave a brief account under the title of "priority queues" and suggested the use of the so-called "post-order trees".

A complete investigation on this topic can be found in the work [10] with tests using a set of typical stochastic scheduling distributions. The paper explicitely produces the algorithms of three different structures and compares them with the linear one.

These structures are the following:

- post — order tree,
- end — order tree,
- indexed list.

The behaviour of all the structures highly depends on the probabilistic nature of the event stream, but the indexed list structure provides the best overall performance. This structure, however, needs an adaptive mechanism to set an interval parameter according to the operating conditions.

---

* In connection with dot notation we refer to [4] and [6].

In this paper we introduce a dynamic version of the indexed list structure which has only a bit worse performance than a well chosen static one, but it needs no adaptive mechanism and it is completely independent of the probabilistic properties of the arrival stream because of its homogeneous nature.

5. General characteristic of homogeneous structures

Let $k>1$ be an integer. Suppose that we have a linear list and every k-th of the elements is pointed to by indexes constituting a new linear list. All the k-th elements of this list are also pointed to by second level indexes and the structure of levels is continued terminating at a highest level consisting of only one element.

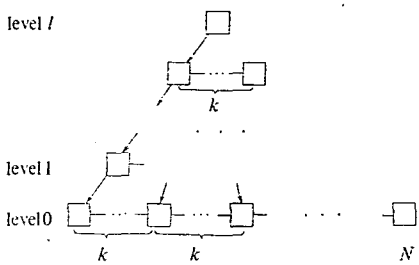


Fig. 4. Fixed homogeneous indexes

The first task is to find the optimal value of the indexing step k:

Theorem 1. The average number of comparisons needed by a call of the procedure scan is minimal if $k=3$.

Proof. The procedure scan works in a natural way: beginning at the highest level the procedure executes a linear list scan in each level from the entry point designated by the pointer having been found at the previous level. Thus, supposing that the probabilities of terminating the scan in a given level are all the same for any of the elements, the average number of conparisons in any level is equal to $k/2$. Hence the average number $m(k)$ of the comparisons in all levels will be

$$m(k) = \frac{k \log N}{2 \log k}.$$

Considering $m(x)$ as a continuous function of $x>1$ simple derivation shows that its only local minimum is achieved at the point $x=e$, and referring to the monotonity when $x>e$ the simple comparison $m(2)>m(3)$ proves the statement.

6. The „2/3-structure"

The following structure (proposed by one of the authors of this paper E. Knuth) is theoretically based on the result of Theorem 1.

Let us allow to use steps of size both $k=2$ and $k=3$ at random in the following way:

- If an arrival occurs into a "molecule" of 2 elements it will simply become a "molecule" of 3 elements.
- If an arrival occurs into a "molecule" of 3 elements it will form two "molecules" of 2 elements and a new index will have to be inserted into the next level in the very same way just described. This process continues

up to a level in which the insertion can be done into a "molecule" of only 2 elements.

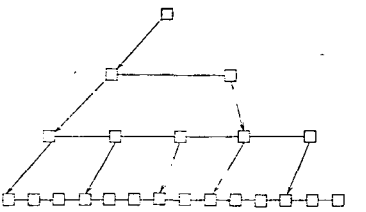


Fig. 5. Example of a "2/3-structure"

The main properties of the structure we defined are the following:

1. The average number of new elements having to be inserted when calling the procedure insert is less than two. This follows from the simple fact that in the worst case, when all the molecules have two elements the whole structure has $2N-1$ elements. (The exact value of the mean number will be determined in paragraph 7.)

2. The average number of comparisons needed by a call of the procedure scan is less than $m(3)$ i.e. it is nearer to the ideal value $m(e)$. (Also proved in paragraph 7.)

3. The "2/3-structure" has a homogeneous nature i.e. it is independent of the distribution of the arrival stream. (This follows from its logical symmetry and has been empirically tested too.)

4. The static structure described in paragraph 5. is naturally unsuitable to practical use for preserving the fixed structure would need complicated insertion procedures. This is also solved by allowing variable size molecules.

7. Stochastic behaviour

Let α_i and β_i be the numbers of the molecules of 2 elements and 3 elements respectively after the i-th insertion.

Theorem 2. $\frac{\alpha_i}{\beta_i}$ stochastically tends to 2.

Proof. Let ξ_k be the number of elements contained in all the molecules of 2 elements after the insertion of the $k+7$-th element. Then ξ_k is an inhomogeneous Markov process with $\xi_0=4$ (seven elements can be arranged in a unique way).

Let

$$\varphi_k = \begin{cases} 0 \text{ if the } k+7\text{-th arrival changes a molecule of} \\ 2 \text{ elements to a molecule of 3 elements;} \\ 1 \text{ if the } k+7\text{-th arrival cuts a molecule of} \\ 3 \text{ elements to two molecules of 2 elements.} \end{cases}$$

Considering the effect of an arrival we have

$$\xi_k = \xi_{k-1}+4\varphi_k+2(\varphi_k-1) = \xi_{k-1}+6\varphi_k-2$$

hence

$$\xi_k = \xi_0+6\sum_{i=1}^{k}\varphi_i-2k.$$

From the construction

$$P\{\varphi_k = 1|\varphi_{k-1}, \ldots, \varphi_1\} = 1-\frac{\xi_{k-1}}{k+6} = 1-\frac{4+6\sum_{i=1}^{k-1}\varphi_i-2(k-1)}{k+6}$$

holds with probability 1. For

$$P\{\varphi_k = 1\} = E\varphi_k$$

it follows

$$E\{\varphi_k|\varphi_{k-1}, \ldots, \varphi_1\} = \frac{3k-6\sum_{i=1}^{k}\varphi_i}{k+6},$$

therefore

$$E\varphi_k = \frac{3k-6\sum_{i=1}^{k-1}E\varphi_i}{k+6}.$$

Since $\xi_0=4$ we get $E\varphi_1=\frac{3}{7}$ and by induction $E\varphi_k=\frac{3}{7}$ for $k\geqq 1$.

Now we have to find the variance

$$D^2(E(\varphi_k|\varphi_{k-1}, \ldots, \varphi_1)).$$

From the relation

$$P\{\varphi_i = 1|\varphi_j = 0, \varphi_{j+1} = y_{j+1}, \ldots, \varphi_{i-1} = y_{i-1}, \xi_{j-1} = x) -$$

$$-P\{\varphi_i = 1|\varphi_j = 0, \varphi_{j+1} = y_{j+1}, \ldots, \varphi_{i-1} = y_{i-1}, \xi_{i-1} = x) = \frac{6}{i}$$

for any $y_{j+1}, \ldots, y_{i-1}=0, 1$ and $0\leqq x\leqq j-1$; we get

$$P\{\varphi_i = 1|\varphi_j = 0\} - P\{\varphi_i = 1|\varphi_j = 1\} = \frac{6}{i} \quad (j < i).$$

And from

$$\frac{4}{7}P\{\varphi_i = 1|\varphi_j = 0\} + \frac{3}{4}P\{\varphi_i = 1|\varphi_j = 1\} = \frac{3}{7}$$

it follows

$$P\{\varphi_i = 1|\varphi_j = 1\} = \frac{3}{7} - \frac{24}{7i}.$$

Hence

$$E(\varphi_i\varphi_j) - E(\varphi_i)E(\varphi_j) = P\{\varphi_j = 1\}(P\{\varphi_i = 1|\varphi_j = 1\} - P\{\varphi_i = 1\}) = \frac{3}{7}\cdot\frac{24}{7i}.$$

On this basis

$$\lim_{k\to\infty} D^2(E(\varphi_k|\varphi_{k-1}, \ldots, \varphi_1)) = \lim_{k\to\infty}\left[\frac{\sum_{i=1}^{k} D^2(\varphi_i)}{(k+6)^2} + \frac{2\sum_{1\leqq i<j\leqq k}\frac{72}{49i}}{(k+6)^2}\right] = 0$$

and referring to the Tchebycheff inequality, it implies the stochastic convergence

$$\frac{\xi_k}{k+6} \to \frac{4}{7}$$

which is equivalent to the statement of the theorem.

Corollary 1. From theorem 2. we get by simple conputation that the average number of events having to be inserted when calling the procedure insert is equal to 1.75.

Corollary 2. The average number of comparisons needed by a call of the procedure scan is $m\left(\frac{7}{3}\right)$. Comparing it to the best fixed structure $m(3)$ we find $m\left(\frac{7}{3}\right) < m(3)$.

8. Algorithms for the 2/3-structure

To build the algorithms in detail first we have to define the following pointers:

- *suc* (successor) and *pred* (predecessor) are the usual linkage pointers for linear lists.
- *for* (forward link) is the indexing pointer between the levels, see Fig. 4. (When being in the lowest level we use *for* to point at the process assigned.)
- *back* (backward link) is a redundant pointer not shown in Figures 4 and 5. The reason of introducing it is to make the building of procedure delete easier. The pointer value is defined by

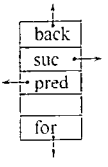


Fig. 6. Event notice pointers at homogeneous structures

for $F:-E.for$, $F.suc$ **while** $F=/=E.suc.for$ **do** $F.back:-E$;

where E is any event not at the lowest level.

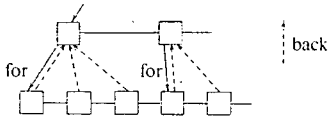


Fig. 7. Definition of the pointer *back*

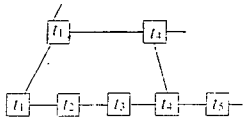


Fig 8. evtime values

Now let us define the *evtime* values at all the levels in the natural way:

$$E.evtime = E.for.evtime.$$

On these bases the logical structures of the procedures we tested are the following. (The exact versions are contained in the Appendix.)

The general procedure delete (E) is contained only by the Appendix for its structure is quite similar one barring that technically more complicated.

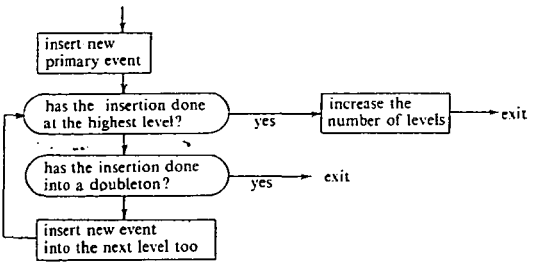


Fig. 10. Logical structure of procedure insert (E, P, t)

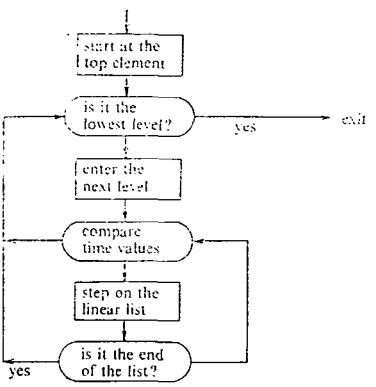


Fig. 9. Logical structure of procedure scan (t)

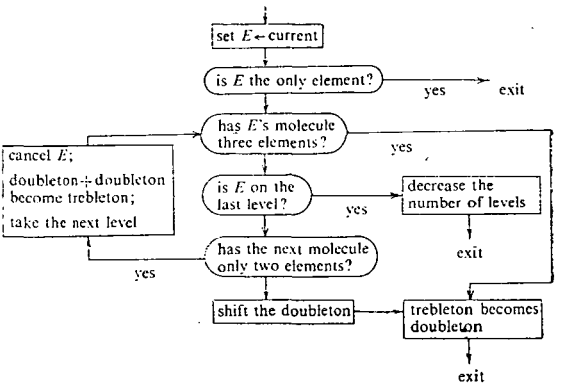


Fig. 11. Logical structure of procedure delete current

9. Experimental results

We chose the execution times of the typical call insert (scan (t), P, t) to compare. The algorithms used are exactly those contained in the Appendix. The test were performed on a Control Data 3300 computer.

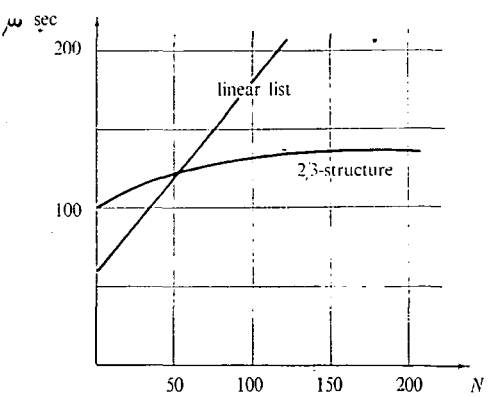


Fig. 12. Comparison of scan + insertion times

In the test the parameter t in the call insert (scan (t), P, t) was exponentially distributed. As it is known the increase of the line describing the performance of the linear list depends on the choice of the distribution of t, but our experimental results showed that using homogeneous indexes the performance was quite independent of it.

The deletion procedure proved to be about 1.8.-times slower than the simple linear one with a small variance. (This is 1.85 for the general version.)

These results altogether designate a limit of about 100 events below which the simple linear method may well be used and the relative performance of the homogeneous structure fastly increases beyond.

10. Further problems

There are several further questions which seem very useful to study. Constructing more and more effective structures is important not only for discrete event simulation but for any other linked structures sorted by continuous keys too. We propose the following problems:

1. We used duplicated events at the higher levels indexing the original ones of the lowest level. We have seen that it leads to an average number $3N/4$ of duplications. However, it seems possible that using certain further pointers even a linear list can be supplied by a crafty additional structure ensuring effective search without duplicated events.

2. The performance can be improved by not cancelling the events when their activity is terminated but leaving them for some kind of "garbage collector". It will not decrease the effectivity of the procedure scan if it is always starts from the end of the list.

3. Finally we notice that during a discrete event simulation the region of time values interested is permanently being shifted to higher values. If we could describe stochastically the distributions of events resulted by such a consistent shift, then it would be possible to develop special structures based on this probabilistic behaviour.

Appendix

In this part we use the SIMULA 67 formalism [4] but the reader needs not to know it precisely because the denotations used are self-explanatory.

A. Algorithms for linear lists

```
ref (event) procedure scan (t); real t;
    begin  ref (event) F;
           F: — last event;
           for F: — F while F. evtime>t do F: — F.pred;
           scan: — F;

    end;

procedure insert (E, P, t);
    ref (event) E; ref (process) P; real t;
```

```
begin ref (event) F;
      F:— new event (t, P);
      F.pred:— E;
      F.suc:— E.suc;
      F.pred.suc:— F.suc.pred:— F;
end;
```

(The last procedure body illustrates how to set the new pointer references. If the subclass relation *link* **class** *event* is assumed using the standard SIMULA list processing facilities the whole procedure body may be simply replaced by **new** event (*t*, *P*). follow (*E*).)

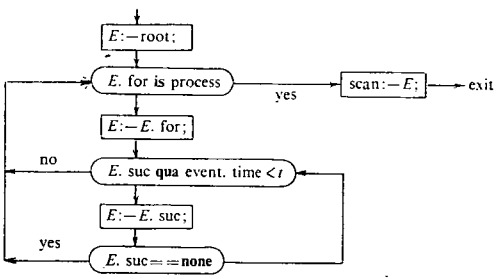


Fig. 13. **ref** (event) **procedure** scan (*t*);

```
procedure delete (E); ref (event) E;
  begin
      E. suc. pred:— E. pred;
      E. pred. suc:— E. suc;
      E. suc:— E. pred:— none;
  end;
```

(The procedure body is equivalent to the standard SIMULA procedure *E*.out.)

B. Algorithms for the 2/3-structure

The structure of the event notice could be defined by

```
class event (back,for, time);
  ref (event) back, for; real time;
  begin
      ref (event) suc,pred;
  end;
```

but for practical reasons in the algorithms it is replaced by the following version using the SIMULA linkage possibilities:

```
link class event (back,for, time);
  ref (event) back,for; real time;;
```

We use the global pointers:

ref (event) root, current;

and the initiation is

root: — current: — **new** event (**none**, **new** process, 0);
root.into (**new** head);

The algorithms are contained by figures 13—16.

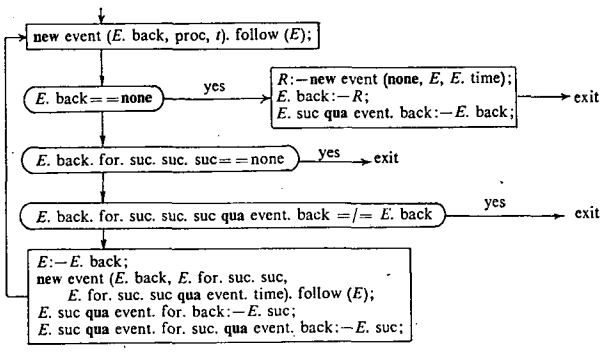


Fig. 14. **procedure** insert (*E*, proc, *t*);

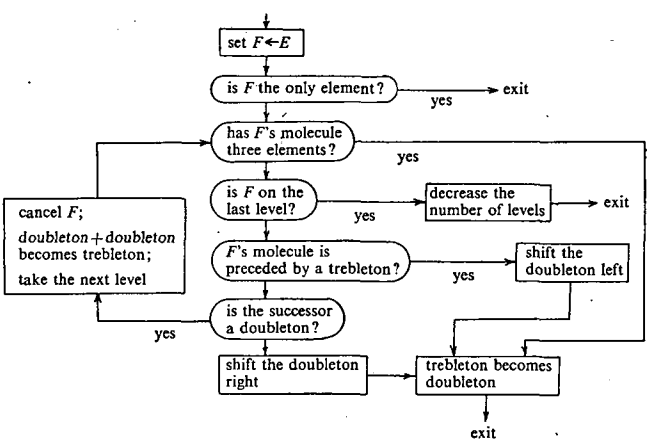


Fig. 15. **procedure** delete current;

Finally we give the logical structure of the general procedure delete. The diagram may readily be programmed in a similar way as the procedure delete current.

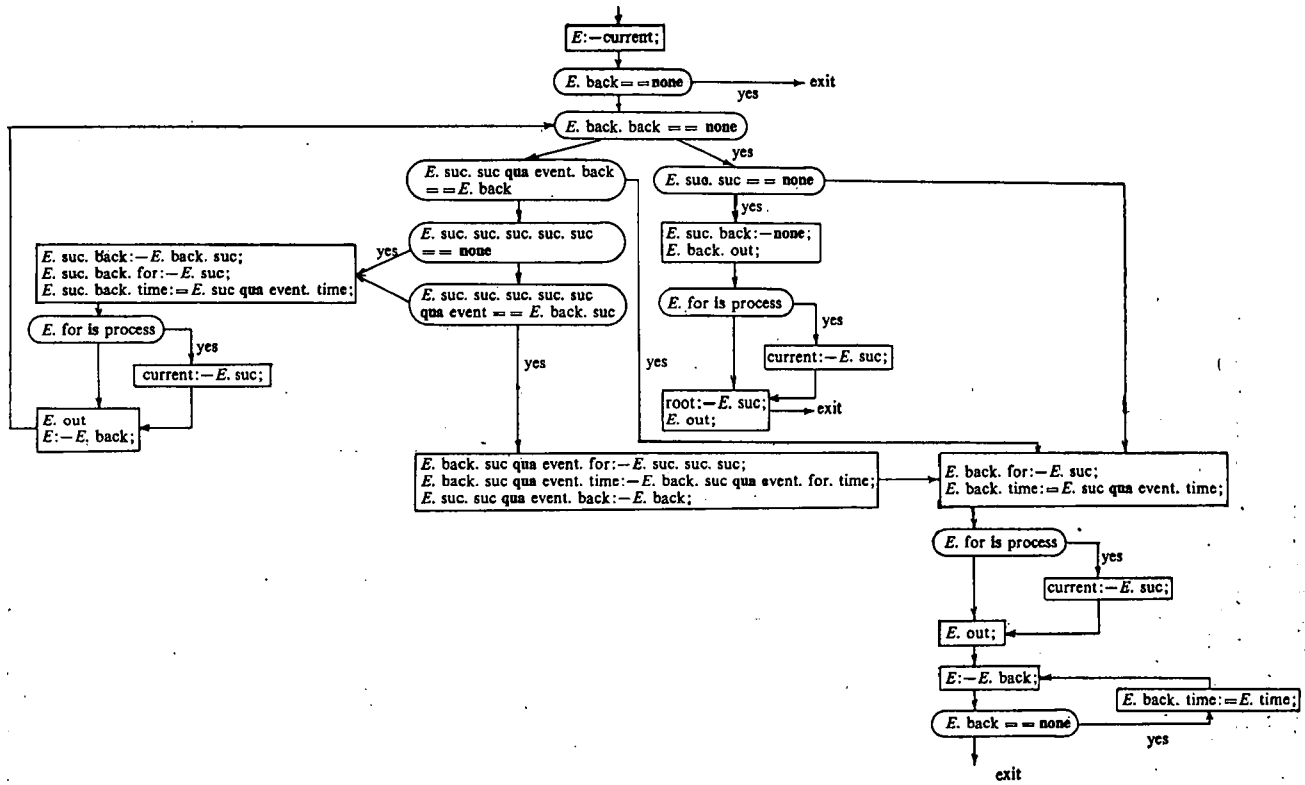


Fig. 16. Logical structure of procedure delete (*E*);

Abstract

Simulation event scheduling algorithms based on linear lists are generally used in simulation languages. Under heavy traffic conditions these algorithms have poor performance. The best of the more complicated algorithms having proposed to improve it is the indexed list method.

In this paper we introduce a multiple-indexed list structure of fully homogeneous nature to eliminate certain disadvantages of the use of static indexes and to gain further improvements. We give all the necessary program routines in detail. The sense of probabilistic behaviour is also given and simulation results are presented to make a comparison with linear list algorithm.

COMPUTER AND AUTIMATION INSTITUTE
HUNGARIAN ACADEMY OF SCIENCES
H-1502 BUDAPEST, HUNGARY

References

[1] BUXTON, J. N. (ed.), *Simulation programming languages*, North—Holland, Amsterdam, 1968.
[2] GENUYS, F. (ed.), *Programming languages*, Academic Press, N. Y., 1968.
[3] GORDON, G., *System simulation*, Prentice — Hall, Englewood Cliffs, N. J., 1968.
[4] DAHL, O. J., B. MYHRHAUG, K. NYGAARD, SIMULA *67 common base language*, S22, Norwegian Computing, Centre, 1967.
[5] KIVIAT, P. J., R. VILLANVEVA, H. H. MARKOWITZ, *The* SIMSCRIPT II *programming language*, Prentice—Hall, Englewood Cliffs, N. J., 1968.
[6] *General purpose simulation system* 360 — User's manual, H20—0326, IBM Corp., White Plain, N. Y., 1968.
[7] WIRTH, N., The programming language PASCAL, *Acta Informat.*, v. 1, 1971, pp. 35—63.
[8] MYHRHAUG, B., *Sequencing set efficiency*, A9, Norwegian Computing Centre.
[9] KNUTH, D. E., *The art of computer programming*, v. 3, Addison Wesley, Reading, Mass., 1973.
[10] VAUCHER, J. G. & P. DUVAL, A comparison of simulation event list algorithms, *Comm. ACM*, v. 18, 1975, pp. 223—230.

(Received March 11, 1976)

Method for simulation of digital automata

By T. VELITCHKOV and K. BOYANOV

Simulation of digital automata is more and more widely applied in designing of digital devices, as it permits full functional testing before their building. The methods for simulation known so far may be classified into two groups. The first one includes the methods for simulation based on the algorithm describing the behaviour of the tested devices [1], [2]. The second group includes the methods based on the logical equations, describing the components the digital device consists of [3], [4]. Some of these methods give an idea of the time sequence of the signals of the real device, others do not. The methods of the first group can be easily applied when the simulation of the automaton as a whole is required, while those of the second group are convenient when the simulation of some components of the automaton is necessary. The existing methods for simulation have some inconveniences. In the detailed examination of the behaviour of a digital automaton the volume of the data processed by the simulating programme and the machine time required, grow much faster than the increase of complexity of the automaton. Due to that it is accepted that the whole automaton consists of few, but complex components. This approach does not allow the detailed study, which is often required.

This paper proposes a method permitting the detailed examination of the behaviour of each of the structurally described components when the simulation of an algorithmically described complex digital automaton is carried out. The method makes possible the determination of the output signals of the automaton for every time interval for which the input signals are determined. Moreover, the simulating programmes remain unchangeable for different structures of the simulated digital automaton and for the various input signals.

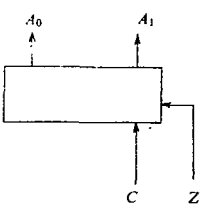


Fig. 1

Basic definitions

We define a finite automaton according to [5]. Logical network for us is a multitude of finite automata with connected inputs and outputs. We accept that time is divided into equal elementary time intervals during which the input and the output symbols of the automaton remain unchangeable and

$$T = \frac{t}{\tau}$$

where t is a real time interval, τ is the elementary time interval and T is a number without dimensions. We call T *delay factor*. The multitude of all input and output symbols assigned for an elementary time interval to the automata the logical network consists of is called *momentary state*. *Time diagram* is the multitude of momentary states for the time interval $t=n\tau$ where $n>1$. We classify the internal states of the

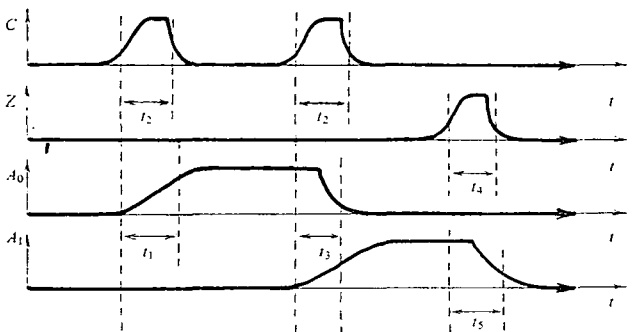


Fig. 2

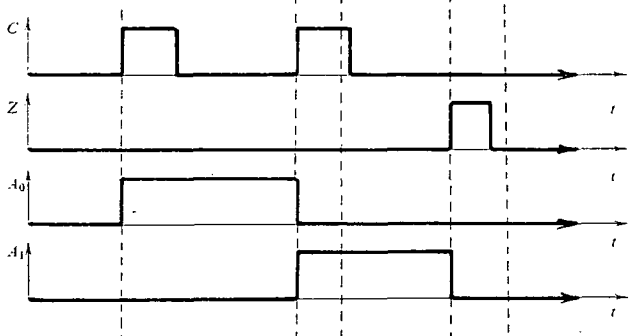


Fig. 3

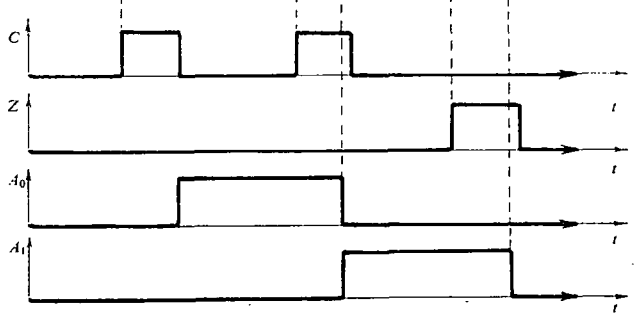


Fig. 4

automaton into two groups: *stable states*, in which the automaton can stay during more than one successive time intervals and *quasistable states*, in which the automaton cannot stay during more than one elementary time interval. We call *signals* the physical phenomena representing the input or the output symbols of the finite automata.

Essential ideas

We will try to represent the real automata so as to obtain the time-diagram of a logical network and its components.

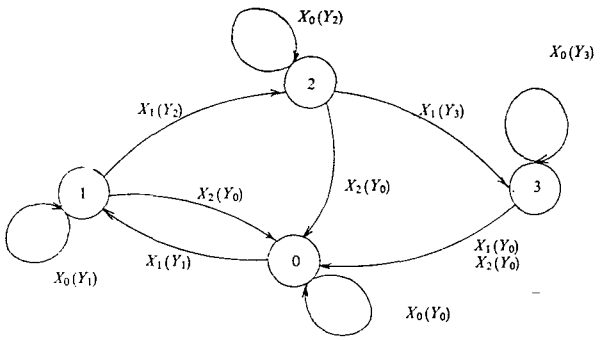


Fig. 5

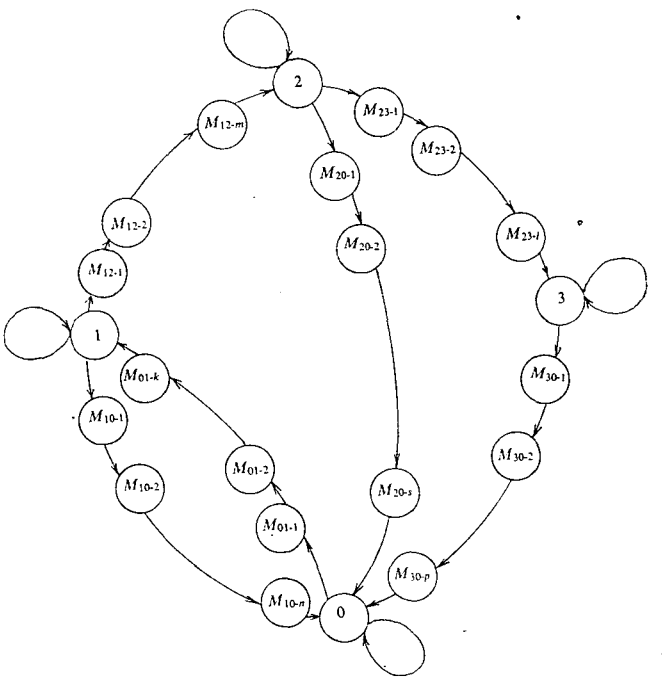


Fig. 6

Let us consider a real automaton having two inputs C and Z and two outputs A_0 and A_1 (fig. 1), whose input and output symbols are represented with real signals. The time-diagram of the automaton is shown on fig. 2. It can be seen that during the time intervals t_1, t_2, t_4 and t_5 the output state of the automaton is undetermined.

We will approximate the behaviour of the real automaton to this of a finite automaton, whose graph is shown on fig. 5 and whose time-diagram is given on fig.3.

Both the input and the output alphabets of the automaton are shown on fig. 14. (The symbol Δ will be discussed later.)

On fig. 3 it can be seen that there is no undetermined in the time-diagram. However, it does not show the time delay between the input affectations and the output reactions of real automaton (time intervals $t_1 \div t_5$.)

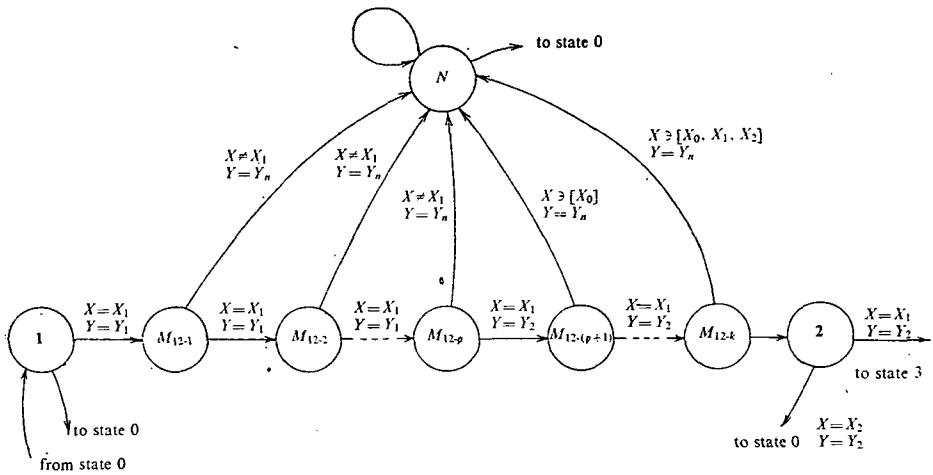


Fig. 7

The time-diagram on fig.4 approximates permissibly precisely the time-diagram given on fig.2. Fig.6, fig.7 and fig.8 show the graphs of automata meeting the requirements of the time-diagram on fig.4. The automaton defined by the graph on fig.6 analyses its own input symbol for every elementary time interval and according to its internal state provides the corresponding output symbol. The stable states of the automaton (0, 1, 2 and 3) are separated by sequences of quasistable states in which the automaton can be during the time intervals $t_1 \div t_5$. The number of the quasistable states is obviously dependent on the duration of the corresponding time interval. The

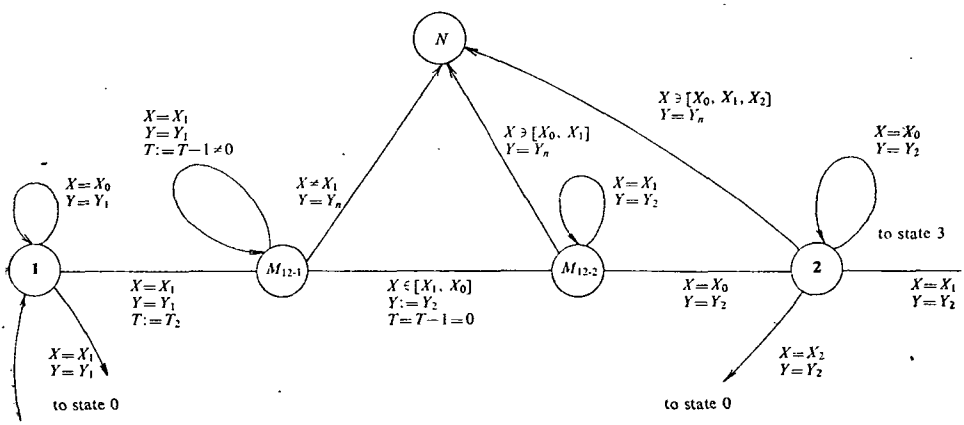


Fig. 8

output symbol of the automaton being in a quasistable state is identical with its output symbol of the latest stable internal state it has been into. The described automaton carries out the time-diagram from fig.4, but in this representation the number of the internal states is too great and this makes difficult the representation and the simulation of complex automata. Besides that, two functionally equivalent real automata with different delay factors will have a different number of internal states and this is quite inconvenient. Moreover, it don't makes possible the checking of the correctness of the input symbols. This inconvenience can be avoided by the introduction of an additional stable state N (undeterminated state, fig. 7). Before getting into any successive state (regardless of its being stable or quasistable) the automaton analyses its input symbol and in case it proves to be unacceptable for the given internal state, the automaton gets into the undeterminated state N and supplies the undetermined output symbol Y_n. The automaton remains in the state N till some preliminarily defined input affectation does not make it change its state. For the automaton on fig.1 this is the input affectation X_0. Nevertheless, the number of the internal states of the automaton is still great in this representation. In order to avoid this inconvenience the automaton is represented on fig.8 by a graph where the successive quasistable states are joined into new stable states M_{12-1}, M_{12-2}... The automaton will remain in these new states for as many elementary time intervals as is the duration of the intervals $t_1 \div t_5$. We consider this representation of the real automata to be enough convenient and will use it in simulation of digital devices.

Simulation

To simulate the work of logical network means in fact to simulate the work of the automata it consists of. Not the real automata are considered, but their equivalent finite automata, which are represented in the manner described above. A multitude of variables $\mathscr{L}$ is introduced for the simulation of the work of a finite automaton.

arguments a_1	arguments a_2	conjunction	disjunction	nor operation	nand operation	exclusive or	exlusion of a_1
0	0	0	0	1	1	0	1
0	1	0	1	0	1	1	1
1	0	0	1	0	1	1	0
1	1	1	1	0	0	1	0
0	Δ	0	Δ	Δ	1	0	1
Δ	0	0	Δ	Δ	1	Δ	Δ
1	Δ	Δ	1	0	Δ	Δ	0
Δ	1	Δ	1	0	Δ	Δ	Δ
Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ

Fig. 9

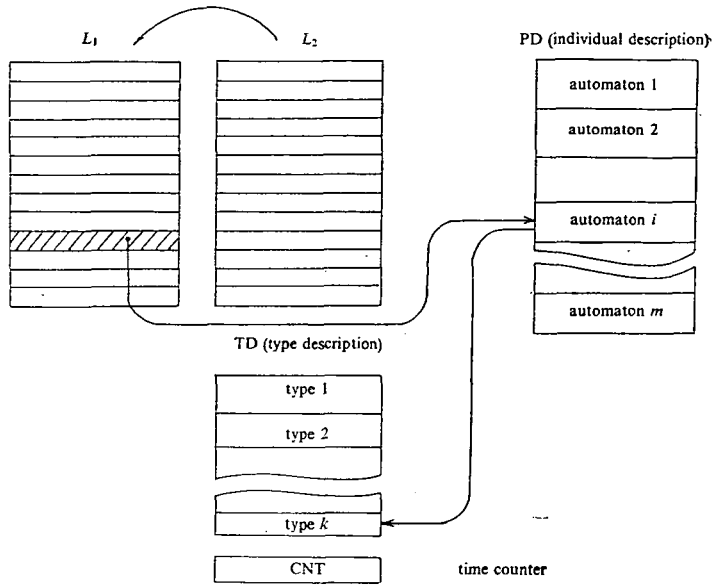


Fig. 10

These variables correspond to the input, output and internal states of the automaton and will be called *internal variables* of the automaton. The variable representing the input state can have many values as is the number of the symbols of the input alphabet of the automaton and its value in any given moment corresponds to the actual input symbol. The internal and the output states are represented in the same way. The multitude of the internal variables includes also variables corresponding to the delay factors, as well as service variables required by the simulating system. As the functioning of every type of automata is individually simulated, it is necessary to define all its stable and quasistable states, the incorrect input affectations, which will be looked for, when the simulation will be carried out and also to define the internal variables of the automata and to work out the algorithm of their changes. For the simulation of a logical network all the automata it consists of should be described and all their connections should be pointed out. It is also necessary to determine the initial state of the network. This presents a difficult problem when complex logical networks are considered. That is why this method accepts that the input (and output) symbols are three : 0, 1 and Δ. The Δ state is undetermined (0 or 1) and it is assign-

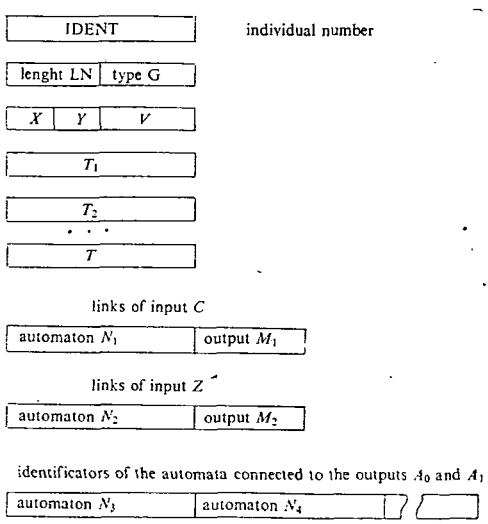


Fig. 11

ed to any input (or output) whose state cannot be precisely specified, as well as to the outputs of the automata, which have received an incorrect input affectation (on the graph on fig.5 and fig.6 this state is marked by Y_n). The introduction of this third state demands that the truth table of the logical functions should be changed (we accept that all logical functions are represented in a classical *and, or, not* basis). The truth table of the elementary functions of two arguments shown on fig.9 illustrates this.

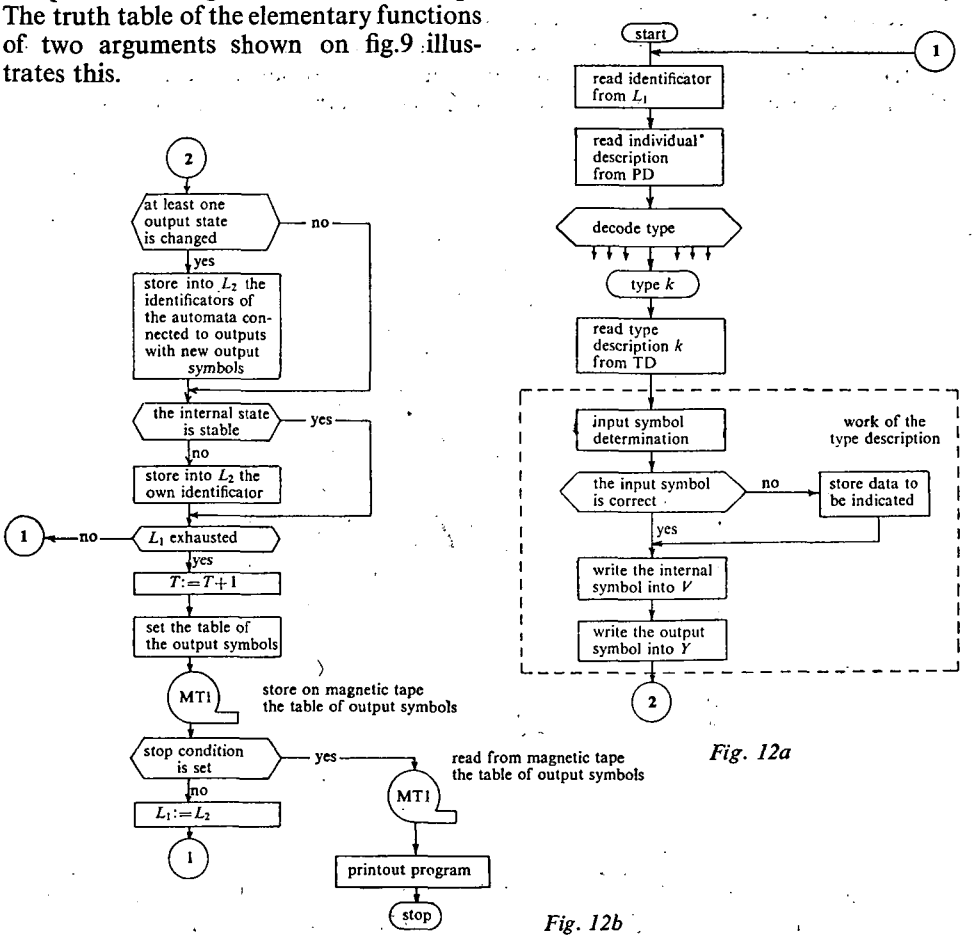


Fig. 12a

Fig. 12b

Structure of the simulating system

This paper considers the simulating system carrying out the described ideas. The system includes the files TD and PD (fig.10), the lists L_1, L_2 and the counter CNT. It makes possible the simulation of the logical network, built of a finite number of types of automata, which have preliminarily assigned internal variables and algorithms. The programmes simulating the functioning of each type of automata are worked out according to their algorithms and are grouped into the file *type descrip-*

tions (TD). The internal variables of all automata, taking part in the network, form the *individual descriptions* of the automata and are grouped into the file PD. Besides that, every individual description includes the length factor (LN), indicating its own length, the identificator IDENT, showing the individual number of the automaton and the type factor G, denoting the type of the automaton (fig.11). The connections between the components of the simulated network are shown by the identificators of the automata and by the numbers of the outputs connected to each input. (See fig.11.) In order to examine a network it is necessary to define precisely the input affectations for every elementary time interval. This method achieves that by describ-

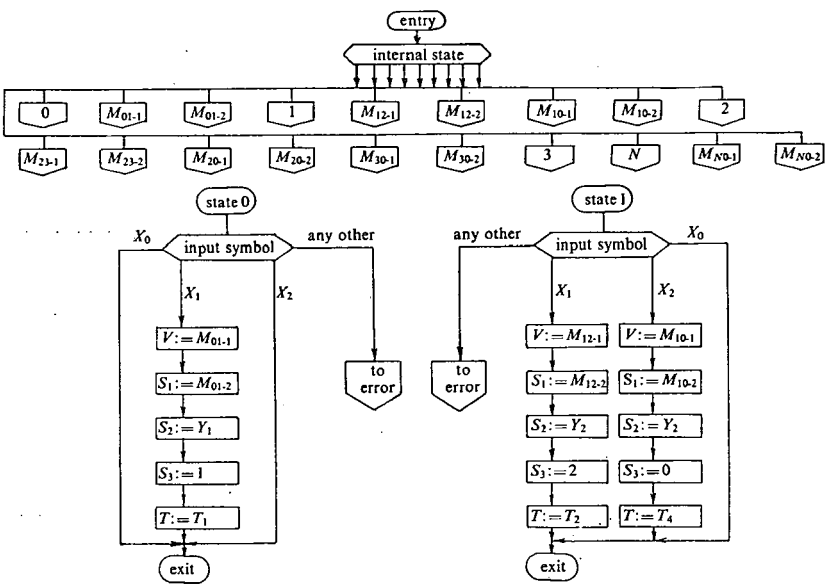


Fig. 13a

ing all input variables as autonomous automata, according to the definition given in [6]. The time counter CNT contains the number of the actual time interval. At the initial moment its value is 0, while all inputs and outputs are in the Δ state. The list L_1 includes the identificators of the autonomous automata, representing the input variables. The programme reads an identificator of an automaton from L_1, takes out its individual description from the file PD, decodes its type and reads its type description from the file TD. During the first elementary time interval this automaton will be an autonomous one whose output will be set in 0 or 1. After that the identificators of all automata connected to the output which has changed its state are put in L_2. In case the automaton has got into a quasistable state, its own identificator will be put in L_2. This process continues till the list L_1 is exhausted. Then the contents of L_2 is put in the place of L_1, "1" is added to the time counter and the study of the behaviour of the network for the next time interval begins. The list L_1 contains now the

identificators of all the automata to whose inputs new symbols have been assigned, as well as the identificators of the automata being in a quasistable state. The rest automata are not studied because their states (input and internal) remain unchangeable. The general algorithm of the simulating programme is given on fig.12. After every elementary time interval "the complete table of output states" is filled with the new data, composed of the identificators and of the states of the outputs of all the automata the network consists of. This table is the result of the simulation and is

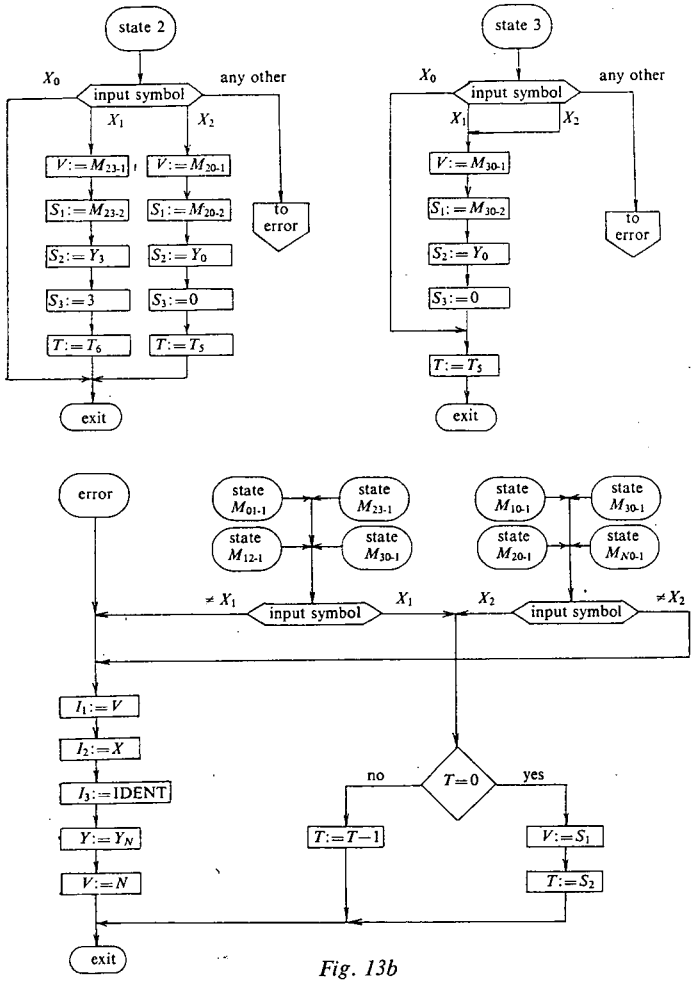


Fig. 13b

stored on a magnetic tape. The simulation will be carried out till some preliminarily specified conditions are detected. (Certain states of some automata or a given value of the time counter CNT.) The structure of the simulated network and the algorithm of the automata for each type are described in a proper language, which is not discussed in this paper.

Let us consider a counter, for example. Its time-diagrams (real and approximated) are shown on fig.3 and fig.4. The graph of the finite automaton, approximating the counter, is given on fig.8. (Only the stable states 1 and 2 and the quasistable ones linking them are shown; the other branches are similar to them.) The multitude of the internal variables becomes of the kind

$$\mathscr{L} \equiv \{V, X, Y, T_1, T_2, \ldots, T_i, T\}$$

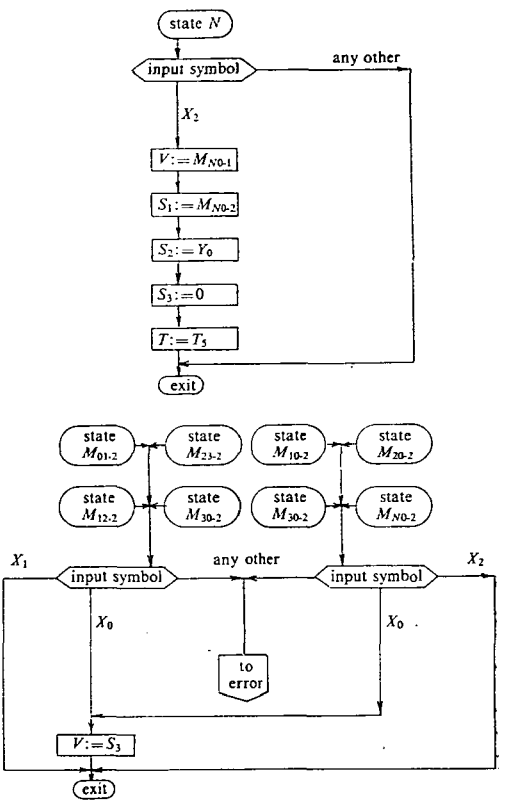


Fig. 13c

where:

V — is the variable of the internal state, it contains the number of the actual internal state;

X — is the variable of the input state, it points out the actual input symbol;
Y — is the variable of the output state, it points out the actual output symbol;
T_i, $i=1, 2, \ldots$ are delay factors;
T — is the variable of the actual delay.

input symbols	binary values of the input signals	
	C	Z
X_0	0	0
X_1	0	1
X_2	1	0
X_3	1	1
X_4	0	Δ
X_5	Δ	0
X_6	Δ	1
X_7	1	Δ
X_8	Δ	Δ

output symbols	binary values of the output signals	
	A_0	A_1
Y_0	0	1
Y_1	0	1
Y_2	1	0
Y_3	1	1
Y_4	0	Δ
Y_5	Δ	0
Y_6	Δ	1
Y_7	1	Δ
Y_N	Δ	Δ

Fig. 14

note: the symbols Y_4, Y_5, Y_6, Y_7 are not allowed

Examining the behaviour of the counter we accept that the input affectations cannot be shorter than t_1, t_2, t_4 (fig.4). Any input affectations which do not meet this requirement will be considered to be incorrect. Let us suppose that the counter is in the stable state 1 (fig.8). It will stay in it as long as the input symbol remains X_0. When the input symbol becomes X_1, the internal state will get into the quasistable state M_{12-1} (see fig.8), the variable of the actual delay (T) receives the value T_2 and the output symbol remains unchangeable. Being in the state M_{12-1} the counter analyses its input symbol, whose value should remain invariably X_1. Meanwhile "1" is subtracted from the current delay during every elementary time interval. When the value of T becomes 0 the counter gets from the state M_{12-1} into the state M_{12-2} and the output symbol changes from Y_1 to Y_2. The change of the input symbol while the internal state is still M_{12-1} means that the input affectation is shorter than T_2. In such a case the next internal state of the automaton will be N and the variable of the output state will receive the value Y_N. The automaton will remain in the state M_{12-2} as long as the input symbol is X_1. It will get into the state 2, when the input symbol is X_0. Any other input symbol will lead into the state N. The maximum length of the input affectation is not checked during the time interval t_3, because it is not limited in time. So far we have considered the branch of the graph connecting the stable states 1 and 2. All other branches are similar.

The algorithm of the counter described above is shown on fig.13. The variables S_1, S_2, S_3, I_1, I_2 and I_3 are service variables. They are not required by the graph but their use makes the algorithm simpler.

Conclusions

The method considered in this paper makes possible the simulation of logical networks, composed of large and expansible set of components. The simulating programme does not depend on the structure of the network and on the kind of the components it consists of. On one hand, the examined network could consist of only one algorithmically described automaton (except the automata representing the input variables). On the other hand, the structure of the network could be described in full details (it can be accepted that the network is composed of elements carrying out the elementary logical functions). This method permits the combination of these two extreme cases. Complex digital automata can be simulated in this way and the detailed examination of some of their parts is possible. Descriptions of new types of components can be easily added to the simulating programme system.

A full information about the functioning of the simulated automata is obtained as a result of the work of the programme. Moreover, race simulation or oscilation of the tested network can be registrated. Finally, by the lists L_1 and L_2 we can judge how often the various components of the tested nerwork are used.

The authors are indebted to ass. prof. D. Dobrev from the Institute of Mathematics and Mechanics of the Bulgarian Academy of Sciences for his valuable recommendations.

BULGARIAN ACADEMY OF SCIENCES
INSTITUTE OF MATHEMATICS AND
MECHANICS WITH COMPUTER CENTER
SOFIA, BULGARIA

References

[1] Боянов, К. & Т. Величков, Моделирование логических схем на цифровой вычислительной машине, *Доклады БАН*, т. 22, № 5, 1969.
[2] Breuer, M. A., Functional partitioning and simulation of digital circuits, *IEEE Trans. Computers*, v. C19, N11, 1970.
[3] Breuer, M. A., Techniques for the simulation of computer logic, *Comm. ACM*, v. 7, 1964, pp. 443—446.
[4] Shalla, L., Automatic analysis of electronic digital circuits using list processing, *Comm. ACM*, v. 9, 1966, pp. 372—380.
[5] Глушков, В. М., *Синтез цифровых автоматов*, Физматгиз, 1969.
[6] Трахтенброт, Б., *Конечные автоматы (поведение и синтез)*, Наука, Москва, 1969.

(Received April 13, 1974)

INDEX—TARTALOM

Felelős szerkesztő és kiadó: Kalmár László
A kézirat a nyomdába érkezett: 1976. június hó
Megjelenés: 1976. december hó
Példányszám: 1000. Terjedelem: 5,2 (A/5) ív
Készült monószedéssel, íves magasnyomással
az MSZ 5601 és az MSZ 5602-55 szabvány szerint
76-2550 — Szegedi Nyomda — F. v.: Dobó József ig.