ALGORYTMY

Vol. XI • No. 20 • 1974

INSTYTUT MASZYN MATEMATYCZNYCH

A L G O R Y T M Y
Vol. XI N° 20 1974

PL ISSN 0065-6240

Adres Redakcji: ul. Krzywickiego 34, 02-078 Warszawa, tel. 28-37-29

Druk IMM zam. 35/75 n. 600 egz. pap. piśm. kl. III g. 70. GP-II/1416/68

TREŚĆ
СОДЕРЖАНИЕ
CONTENTS

ALGORYTMY, XI, N° 20, 1974, 5-10 517.51:517.949

AN UNBIASED FINITE-DIFFERENTIAL ESTIMATOR OF THE GRADIENT

Ryszard ZIELIŃSKI
Instytut Matematyczny PAN

Received 27th 02. 1974

Estimation of the derivative of a function f at the point x by $(f(x + h) - f(x))/h$ yields the error $O(h)$; estimation by $(f(x + h) - f(x - h))/2h$ yields the error $O(h^2)$. The former method required computation of $k + 1$ values of f, the latter $2k$ values provided f is a function of k variables. In the paper [1] a randomized method was presented which required $k + 1$ points and yielded the expected error $O(h^2)$. In the present note a randomized method is considered which requires $2k$ points and for which the expected error equals to zero(unbiasedness).

1. INTRODUCTION

The paper deals with estimation of the gradient of a given function f at a given point without calculation of the derivatives of the function. To be more precise, we shall give a formula for estimation of the first partial derivative of the function. To simplify notations we shall consider the problem of estimation of the first derivative of a function $f: R \to R$. The estimation of the gradient consists in repeating calculations for each first partial derivative separately. The formula requires a randomization and calcu-

lation of the values of the function at two points; thus estimation of the gradient of a function of k variables will require calculation of the values of the function at $2k$ points.

Throughout the paper is assumed that f has all derivatives $f^{(i)}(x)$, $i = 1,2,\ldots$ and that $f(x + h) =$

$$= \sum_{i=0}^{\infty} (h^i/i!)\, f^{(i)}(x) \quad \text{for} \quad x, h \in R.$$

2. RESULT

Let $U = (u_j)_{j=0}^{\infty}$, $V = (v_j)_{j=0}^{\infty}$ be real sequences such that

$$\sum_{j=0}^{\infty} v_j u_j = \frac{1}{2}; \qquad \sum_{j=0}^{\infty} v_j u_j^{2i+1} = 0 \quad \text{for all } i=1,2,\ldots \tag{1}$$

Let $\mathcal{F}$ be a class of functions f. We define the sequences U and V as feasible ones for $\mathcal{F}$ if

$$\sum_{j=0}^{\infty}\sum_{i=0}^{\infty} \frac{v_j u_j^{2i+1}}{(2i+1)!} f^{(2i+1)}(x) = \sum_{i=0}^{\infty}\sum_{j=0}^{\infty} \frac{v_j u_j^{2i+1}}{(2i+1)!} f^{(2i+1)}(x) \tag{2}$$

for every $f \in \mathcal{F}$.

Theorem

Let J be a random variable such that $P(J = j) = q_j > 0$, $j = 0,1,2,\ldots$. If U and V are feasible sequences for $\mathcal{F}$ then for each $f \in \mathcal{F}$ the random variable

$$Y(x) = \frac{v_J \left[f(x + u_J) - f(x - u_J) \right]}{q_J} \qquad (3)$$

is an unbiased estimator of the first derivative $f(x)$ of the function f at the point x, i.e. $EY(x) = f'(x)$.

An appropriate algoritm may be formulated in the following way:

1. sample the random variable J ;
2. calculate the values of the function f at the points $f(x + u_J)$ and $f(x - u_J)$;
3. estimate the derivative $f'(x)$ by the formula (3).

3. THE CHOICE OF U AND V

It is easy to see that for a given n there exist sequences $u_0, u_1, \ldots, u_n$ and $v_0, v_1, \ldots, v_n$ such that

$$\sum_{j=0}^{n} v_j u_j = \frac{1}{2} \quad \text{and} \quad \sum_{j=0}^{n} v_j u_j^{2i+1} = 0$$

for $i=1,2,\ldots n$

The sequence $u_0, u_1, \ldots, u_n$ may be chosen whichever one likes provided $u_i \neq u_j$ for $i \neq j$. Then $v_0, v_1, \ldots, v_n$ are solutions of the system of linear equations with the matrix

$$M_n = \begin{bmatrix} u_0 & u_1 & \dots & u_n \\ u_0^3 & u_1^3 & \dots & u_n^3 \\ \dots & \dots & \dots & \dots \\ u_0^{2n+1} & u_1^{2n+1} & \dots & u_n^{2n+1} \end{bmatrix}$$

$$\text{Det } M_n = u_0 u_1 \dots u_n \prod_{j>i} (u_j^2 - u_i^2) \neq 0 \quad \text{and for } v_k,$$

$k = 0, 1, 2, \dots, n,$ we have

$$v_k = \left\{ 2u_k \prod_{j=0}^{k-1} \left(1 - \left(\frac{u_k}{u_j}\right)^2\right) \cdot \prod_{j=k+1}^{n} \left(1 - \left(\frac{u_k}{u_j}\right)^2\right) \right\}^{-1} \quad (4)$$

Unfortunately, for $n \to \infty$ it is rather difficult to obtain v_k by (4) with given u_j. By (4) we conclude that for $n \to \infty$ the product $\prod_{j=k+1}^{n} \left[1 - \left(\frac{u_k}{u_j}\right)^2\right]$ should converge so that the series $\sum_{j=k+1}^{n} u_j^{-2}$ should converge and u_j^2 should tend to infinity. Now by (1) we conclude that v_j should tend to zero fast enough.

Another method of construction U and V is following. Let F be a periodic function such that $F(kT) = 0$ for $k = 1,2,\dots$ and $F\left(\frac{T}{k}\right) \neq 0$, say, equals to $\frac{1}{2}$. Suppose that $F(x) = \sum_{j=0}^{\infty} a_j x^j$ for every $x \in R$. Now we can choose $v_j = a_j (T/k^2)^j$ and $u_j = k^j$. Then

$$\sum_{j=0}^{\infty} v_j u_j^{2i+1} = \sum_{j=0}^{\infty} a_j (k^{2i-1}T)^j = F(k^{2i-1}T) = \begin{cases} \frac{1}{2} & \text{for } i=0 \\ 0 & \text{for } i=1,2,\ldots \end{cases}$$

and (1) holds. For example letting $F(x)= \sin x$ we obtain $v_j = (-1)^j (\pi/36)^{2j+1}/(2j+1)!$ and $u_j = 6^{2j+1}$.

4. PROOF OF THE THEOREM

$$EY(x) = \sum_{j=0}^{\infty} q_j E(Y(x) \mid J=j) =$$

$$= \sum_{j=0}^{\infty} v_j \left[f(x+u_j) - f(x-u_j)\right] =$$

$$= \sum_{j=0}^{\infty} v_j \left(2 \sum_{i=0}^{\infty} \frac{u_j^{2i+1}}{(2i+1)!} f^{(2i+1)}(x)\right) =$$

$$= 2\sum_{i=0}^{\infty} \frac{1}{(2i+1)!} f^{(2i+1)}(x) \sum_{j=0}^{\infty} v_j u_j^{2i+1} = f'(x)$$

5. REMARK

By (2) we conclude that for a given function f the sequences U and V should be chosen in a suitable way. For example if f is a polynomial (2) holds for any U and V. Another example: if $f^{(2i+1)}(x)$ are bounded and $\sum_{i,j} v_j u_j^{2i+1}/(2i+1)!$ converges absolutely then (2) holds.

O PEWNEJ NIEOBCIĄŻONEJ SKOŃCZENIE-RÓŻNICOWEJ METODZIE SZACOWANIA GRADIENTU FUNKCJI

Streszczenie

Oszacowanie pochodnej funkcji f w punkcie x za pomocą ilorazu $(f(x+h) - f(x))/h$ daje błąd $O(h)$; oszacowanie $(f(x+h) - f(x-h))/2h$ daje błąd $O(h^2)$. W przypadku szacowania gradientu funkcji k zmiennych pierwsza metoda wymaga obliczania wartości funkcji f w k+1 punktach, druga w 2k punktach. W pracy [1] przedstawiono pewną metodę losowania k+1 punktów tak, żeby oczekiwany błąd był $O(h^2)$. W niniejszej pracy podano 2k-punktowe oszacowanie gradientu, w którym oczekiwany błąd jest równy zeru (estymator nieobciążony).

ОБ ОДНОЙ НЕСМЕЩЕННОЙ ОЦЕНКЕ ГРАДИЕНТА ФУНКЦИИ ПРИ ПОМОЩИ КОНЕЧНЫХ РАЗНОСТЕЙ

Резюме

Оценка производной функции f в точке x при помощи $(f(x+h)-f(x))/h$ имеет ошибку $O(h)$: оценка $(f(x+h)-f(x-h))/2h$ имеет ошибку $O(h^2)$. В случае функций к переменных первая оценка требует вычисления значении функции f в к + 1 точках, вторая - в 2к точках. В статье [1] предложен метод который требует вычисления к + 1 значении функции и имеет ожидаемую оценку $O(h^2)$. В настоящей работе предлагается рандомизированная оценка которая требует вычисления 2к значении функции и имеет нулевую ожидаемую ошибку (оценка несмещенная).

Warszawa, ul. Krzywickiego 34
A.M.

ALGORYTMY, XI, N° 20, 1974, 11-43 330.115:338:339

PROCESSES IN COMPOSED SYSTEMS

by Józef WINKOWSKI
Centrum Obliczeniowe PAN

Received 11th March 1974

The paper refers to methods of characterization of processes which run in composed systems. The methods are considered, where a process running in a system composed of a number of components is characterized by processes which run in the components. The main problem that is considered is to explain to which extent such a characterization determines the characterized process. As in [4], systems, composed systems, and processes are described using simple categorical notions.

The problem of describing of processes which run in composed systems arises often in practice. Such a problem appears, for example, in simulation technique applications. A solution consists usually in describing the processes in the components and relationships among them. The natural question arises about a theoretical base of the method. In the paper a suggestion for such a base is presented. Some simple categorical notions are used for this purpose.

As in [4] , categories are used for systems describing. Objects represent states and morphisms - the actions which

imply state changes. Any composed system is described by a subcategory of the product of the categories which describe components. Processes in the system, as well as in the system components, are described by functors from linear orderings to the suitable categories.

First, some needed notions will be introduced to make the paper selfcontained. Next, in order to support appropriate intuitions, we consider a behaviour of a concrete simple system. Finally, we give more general problem formulation and try to explain its nature with some details.

1. NOTIONS AND DENOTATIONS

A graph A consists of a class $|A|$ and of operations

$$|A| \xrightarrow{\partial_i} |A| \quad , \quad i = 0,1$$

which satisfy the conditions

$$\partial_i \, \partial_j = \partial_j$$

i.e. the conditions

$$\partial_j \, (\partial_i (\alpha)) = \partial_i (\alpha) \tag{1}$$

for every $\alpha \in |A|$, $i = 0,1$, $j = 0,1$.

The elements $\alpha \in |A|$ we call (generalized) edges. The (generalized) edges which satisfy the condition

$$\partial_0 (\alpha) = \alpha$$

we call vertices. Due to (1) , $\partial_0(\alpha)$, $\partial_1(\alpha)$ are vertices for every $\alpha \in |A|$. They will be called beginning and end of α respectively. For simplicity, the beginning will be

denoted by α^- and the end by α^+. The class of all the vertices of A will be denoted by $|A|^0$.

Any non-empty finite sequence of edges of A, where every edge has the end identical with the beginning of the next one, we call path of A.

A category is a graph with a partial binary operation which assignes an edge $\alpha \cdot \beta$ with

$$(\alpha \cdot \beta)^- = \alpha^-, \qquad (\alpha \cdot \beta)^+ = \beta^+$$

to every edges α, β with

$$\alpha^+ = \beta^-$$

in such a way that

$$(\alpha \cdot \beta) \cdot \gamma = \alpha \cdot (\beta \cdot \gamma), \qquad \alpha^- \cdot \alpha = \alpha, \qquad \alpha \cdot \alpha^+ = \alpha$$

Edges and vertices in a category are called morphisms and objects respectively. The morphism $\alpha \cdot \beta$ we call a composition of α, β. We usually express the conditions like

$$a = \alpha^-, \; b = \alpha^+$$

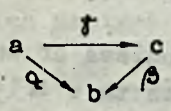
in the form

$$a \xrightarrow{\alpha} b$$

the conditions like

$$a = \alpha^- = \gamma^-, \; b = \alpha^+ = \beta^-, \; c = \beta^+ = \gamma^+$$

in the form



(2)

and so on. Pictures like the above we call diagrams. The diagrams in which all the compositions of the morphisms of arbitrary paths with common beginning and end are identical we

call commutative. For example, the diagram (2) is commutative iff $\gamma = \alpha \cdot \beta$.

If there exists such an object a that for every object b there exists unique morphism

$$a \xrightarrow{\alpha} b$$

then a is called an initial object of the category.

Any quasi-ordering $\leqslant$ with pairs $a \leqslant b$ as morphisms with the beginning $a \leqslant a$ and end $b \leqslant b$ and with the composition

$$(a \leqslant b) \cdot (b \leqslant c) = (a \leqslant c)$$

is a category. Any object $a \leqslant a$ is usually identified with a.

In the following, the natural ordering of the numbers $0,1,2,\ldots$, considered as a category, will be denoted by $\mathcal{N}$.

The class of all the paths of a graph with beginnings and ends defined in the natural way and with the natural composition is a category.

The product of categories is a category. For example, in $A \times B$ we define as usually

$$(\alpha, \beta)^- = (\alpha^-, \beta^-)$$
$$(\alpha, \beta)^+ = (\alpha^+, \beta^+)$$
$$(\alpha_1, \beta_1) \cdot (\alpha_2, \beta_2) = (\alpha_1 \cdot \alpha_2, \beta_1 \cdot \beta_2)$$

A category all the morphisms of which are morphisms of a given category A, and such that α^-, α^+, $\alpha.\beta$ are the same as in A is called a subcategory of A. If it is the smallest subcategory with given morphisms then we call it to be generated by these morphisms. A subcategory of A with exactly the same as in A morphisms between arbitrary two its

objects is called full.

A functor from a category A into another category B is a mapping

$$|A| \xrightarrow{f} |B|$$

which satisfies the conditions

$$f(\alpha^-) = (f(\alpha))^- , \qquad f(\alpha^+) = (f(\alpha))^+$$

$$f(\alpha \cdot \beta) = f(\alpha) \cdot f(\beta)$$

For such a functor we write

$$A \xrightarrow{f} B$$

If there exists the inverse f^{-1} and it is a functor from B into A, then f is called isomorphism and A, B are called isomorphic.

The projections of the product of categories onto the factors are functors. The natural injection of the class $|B|$ of morphisms of a subcategory B of a category A into $|A|$ is a functor. In the case of categories which are quasi-orderings functors correspond simply to isotonic mappings.

A natural mapping of a functor

$$A \xrightarrow{f} B$$

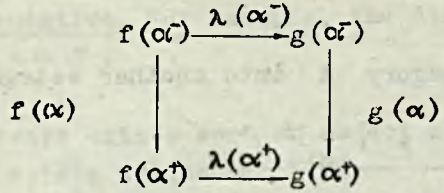
into another functor

$$A \xrightarrow{g} B$$

is an assignement λ which assignes a morphism

$$f(a) \xrightarrow{\lambda(a)} g(a)$$

of B to every object $a \in |A|^0$ in such a way that for every $\alpha \in |A|$ the diagram



is commutative. Such a mapping is usually denoted by

$$f \xrightarrow{\lambda} g$$

2. A PRODUCER-CONSUMER SYSTEM

In order to give an example of a system, let us consider a producer which produces some objects and a consumer which consumes these objects. The producer can not produce and the consumer can not consume more than one object simultaneously. The objects the producer has produced and the consumer has not taken yet form a stock. It may be used by the consumer. The producer, the consumer, and the stock form a system that will be considered in the following.

The producer actions can be identified with the paths of the following graph

$$r \underset{\beta}{\overset{\alpha}{\rightrightarrows}} p$$

where α denotes an object production beginning, β - an object production end, r - a rest state, and p - a production state. These actions and states form a category C_1.

Similarly, the consumer actions can by identified with the paths of the graph

$$w \underset{\delta}{\overset{\gamma}{\rightrightarrows}} c$$

where γ denotes an object consumption beginning, δ - an object consumption end, w - a state of waiting, and c - an

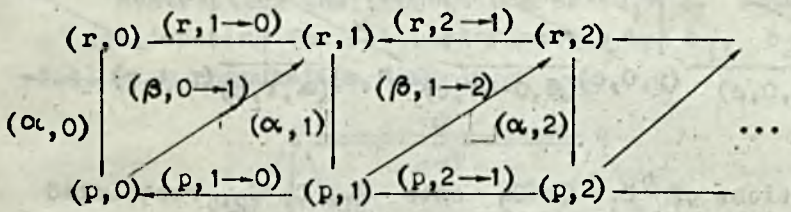
object consumption state. They form a category C_2.

Finally, the stock changes are morphisms of the quasi-ordering C generated by the graph

$$0 \rightleftarrows 1 \rightleftarrows 2 \rightleftarrows \dots$$

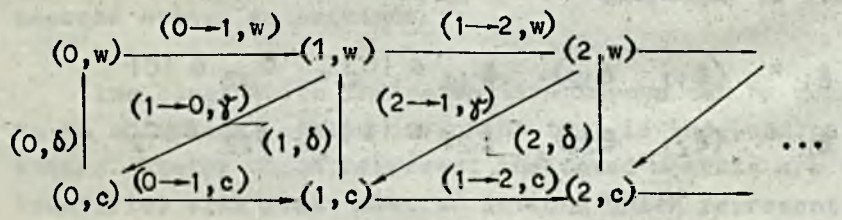
where 0,1,2,... denote stock levels.

The producer affects on the stock. The stock level increases by 1 at any object production end. Otherwise, the stock level does not increase. Hence, the producer-stock subsystem actions are generated by the following morphisms of $C_1 \times C$



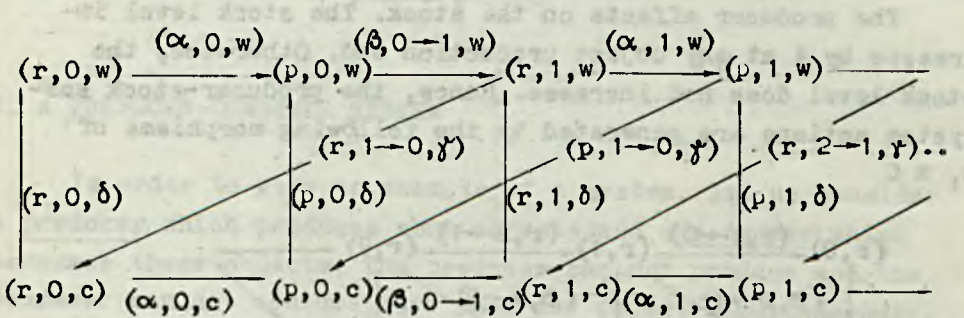
(however, this diagram is not commutative). They form a subcategory Q_1 of $C_1 \times C$. The projections of $C_1 \times C$ onto C_1, C, restricted to Q_1, will be denoted by μ_1, π_1.

The stock affects on the consumer in such a way that the consumer can use the objects of the stock only. On the other hand, the stock level decreases by 1 if the consumer starts to consume an object. Otherwise, the stock level does not decrease. Therefore, the stock-consumer subsystem actions are generated by the following morphisms of $C \times C_2$



They form a subcategory Q_2 of $C \times C_2$. The projections of $C \times C_2$ onto C, C_2, restricted to Q_2, are denoted by $\varkappa_2$, μ_2.

The interaction in the producer-stock-consumer system consists in the above mentioned producer influence on the stock and in the stock-consumer interaction. As result, the actions of the whole system form a subcategory Q of $C_1 \times C \times C_2$ which is generated by the following morphisms



The projections of $C_1 \times C \times C_2$ onto C_1, C, C_2, restricted to Q, are denoted by Q_1, $\varkappa$, Q_2.

Observe that we could get an equivalent description considering the subsystems Q_1, Q_2 as components. It is sufficient then to take into account the fact that the stock is the same in Q_1 and Q_2 (i.e. is a common component of Q_1 and Q_2). We get then a subcategory Q' of $Q_1 \times Q_2$ with morphisms (ξ_1, ξ_2) which satisfy the condition

$$\varkappa_1(\xi_1) = \varkappa_2(\xi_2)$$

i.e. are of the forms

$$\xi_1 = (\xi_{11}, \xi_{12}), \quad \xi_{11} \in |C_1|, \quad \xi_{12} \in |C|$$

$$\xi_2 = (\xi_{21}, \xi_{22}), \quad \xi_{21} \in |C|, \quad \xi_{22} \in |C_2|$$

with

$$\xi_{12} = \xi_{21}$$

The morphism

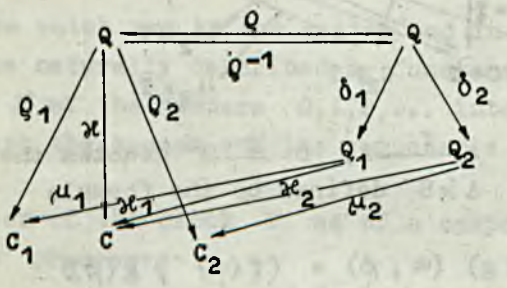
$$\varrho(\xi_1, \xi_2) = (\xi_{11}, \xi_{12}, \xi_{22})$$

of Q can be assigned to each $(\xi_1, \xi_2) \in |Q'|$ and it gives a functor

$$Q' \xrightarrow{\varrho} Q$$

As it is easy to see, it is an isomorphism so that Q, Q' are isomorphic.

Restricting the projections of $Q_1 \times Q_2$ onto Q_1, Q_2 to Q' and denoting the obtained functors by δ_1, δ_2 we get the following commutative diagram



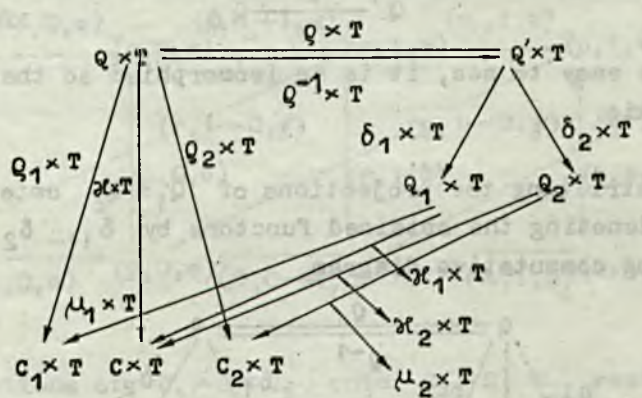
3. PROCESSES IN THE PRODUCER-CONSUMER SYSTEM

Various processes of the above described producer-consumer system are usually characterized by a clock time notion. In fact, some time durations are assigned to the system or its components actions and it characterizes to some extent a process states consequence.

The clock which indicates time lapses can be considered as an additional system component that is independent of the others. States which represent indicated moments are usually identified with real numbers. Actions which represent

indicated time lapses are time intervals of the form $t_1 \leqslant t_2$. Therefore, the clock T can be considered as the natural ordering of real numbers.

The fact that the clock works independently of other components we express by considering the components C_1, C, C_2 and the systems Q_1, Q_2, Q, Q′ together with the clock T as $C_1 \times T$, $C \times T$, $C_2 \times T$, $Q_1 \times T$, $Q_2 \times T$, $Q \times T$, $Q' \times T$ respectively. Then we get the following commutative diagram

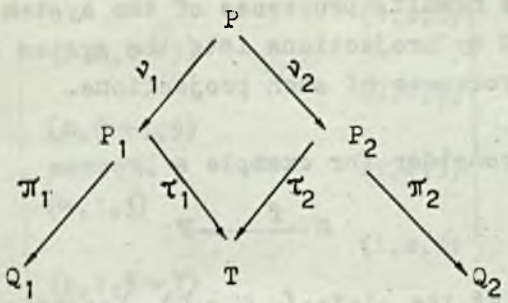


where $f \times g$ for $A \xrightarrow{f} A'$, $B \xrightarrow{g} B'$ denotes the functor from $A \times B$ into $A' \times B'$ defined by the formula

$$(f \times g)\ (\alpha, \beta) = (f(\alpha)\ ,\ g(\beta))$$

and T denotes the identity mapping of $|T|$.

In order to simplify denotations we write P, P_1, P_2 for $Q \times T$, $Q_1 \times T$, $Q_2 \times T$ and ϑ_1, ϑ_2 for $(Q^{-1} \times T) \cdot (\delta_1 \times T)$, $(Q^{-1} \times T) \cdot (\delta_2 \times T)$ respectively and denote the projection of P_1 onto Q_1 by π_1, the projection of P_2 onto Q_2 by π_2, the projection of P_1 onto T by τ_1, and the projection of P_2 onto T by τ_2. Then the following diagram



is commutative and ϑ_1, ϑ_2 determine uniquely an injection

$$P \xrightarrow{\vartheta} P_1 \times P_2$$

and a functor

$$P \xrightarrow{\tau = \vartheta_1 \cdot \tau_1 = \vartheta_2 \cdot \tau_2} T$$

Processes which run in the system and in the system components can be naturally described by functors from the natural ordering $\mathcal{N}$ of the numbers 0,1,2,... into the categories which represent the system and its components.

A function of the clock T as of a component of P, P_1, P_2 results in processes

$$\mathcal{N} \xrightarrow{f} P, \qquad \mathcal{N} \xrightarrow{f_1} P_1, \qquad \mathcal{N} \xrightarrow{f_2} P_2$$

from the fact that non-decreasing moments

$$\tau(f(0)) \leqslant \tau(f(1)) \leqslant \ldots$$
$$\tau_1(f_1(0)) \leqslant \tau_1(f_1(1)) \leqslant \ldots$$
$$\tau_2(f_2(0)) \leqslant \tau_2(f_2(1)) \leqslant \ldots$$

correspond to the consecutive states 0,1,2,... of the processes. Therefore, the consequence of states is characterized to some extent by information on the actions which have been

performed. As result, processes of the system can be often characterized by projections into the system components or even by subprocesses of such projections.

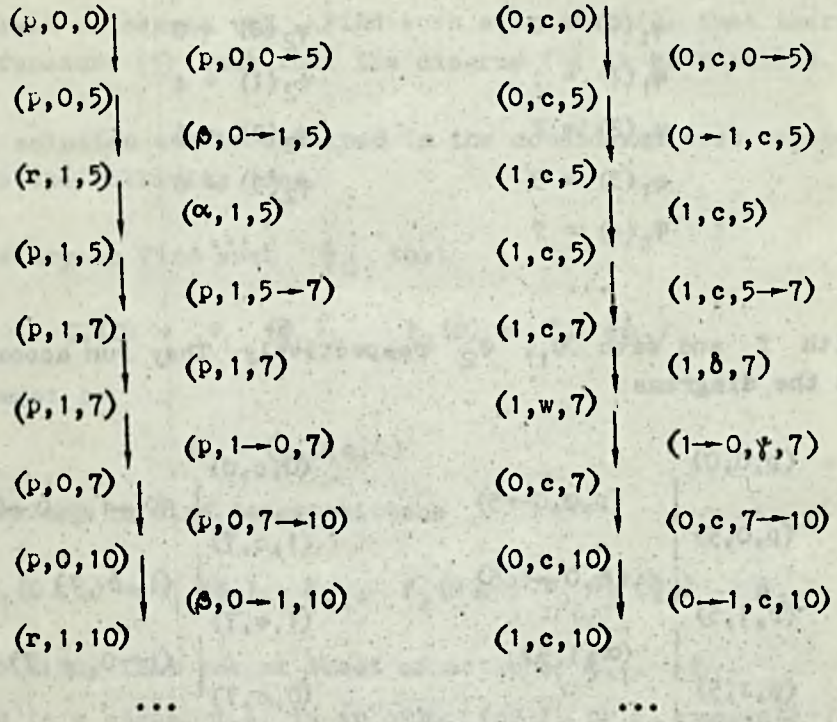
Let us consider for example a process

$$\mathcal{N} \xrightarrow{f} P \qquad (3)$$

which starts at the state $(p,C,c,0)$. Let it will be the process in which the producer produces every object in 5 time units, the consumer consumes every object in 7 time unite and then starts to consume the next one as soon as possible, and the actions α, β, δ need no time at all. This process runs in the following way

$$\begin{array}{ll}
(p,0,c,0) & \\
\downarrow & (p,0,c,0\rightarrow 5) \\
(p,0,c,5) & \\
\downarrow & (\beta,0\rightarrow 1,c,5) \\
(r,1,c,5) & \\
\downarrow & (\alpha,1,c,5) \\
(p,1,c,5) & \\
\downarrow & (p,1,c,5\rightarrow 7) \\
(p,1,c,7) & \\
\downarrow & (p,1,\delta,7) \\
(p,1,w,7) & \\
\downarrow & (p,1\rightarrow 0,\gamma,7) \\
(p,0,c,7) & \\
\downarrow & (p,0,c,7\rightarrow 10) \\
(p,0,c,10) & \\
\downarrow & (\beta,0\rightarrow 1,c,10) \\
(r,1,c,10) & \\
\downarrow & \\
\cdots &
\end{array}$$

Its projections into the components P_1, P_2 can be obtained by composing f with ϑ_1, ϑ_2 respectively. These projections run according to the diagrams



They uniquely determine, action after action, the process (3). However, it appears, some their subprocesses determine its run as well.

Let us consider for example the subprocesses

$$\mathcal{N} \xrightarrow{f_1} P_1, \qquad \mathcal{N} \xrightarrow{f_2} P_2 \tag{4}$$

of the above projections, those obtained by composing the injections

$$\mathcal{N} \xrightarrow{\varphi_1} \mathcal{N} \qquad \mathcal{N} \xrightarrow{\varphi_2} \mathcal{N} \tag{5}$$

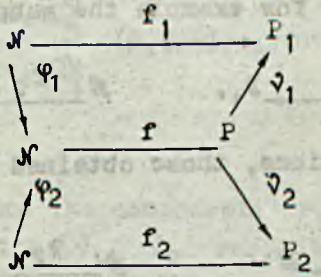
where

$$\varphi_1(0) = 0 \qquad \varphi_2(0) = 0$$
$$\varphi_1(1) = 1 \qquad \varphi_2(1) = 4$$
$$\varphi_1(2) = 2 \qquad \varphi_2(2) = 5$$
$$\varphi_1(3) = 3 \qquad \varphi_2(3) = 6$$
$$\varphi_1(4) = 7 \qquad \ldots$$
$$\ldots$$

with f and with ϑ_1, ϑ_2 respectively. They run according to the diagrams

$$\begin{array}{l} (p,0,0) \\ \downarrow \quad (p,0,0\to 5) \\ (p,0,5) \\ \downarrow \quad (\beta,0\to 1,5) \\ (r,1,5) \\ \downarrow \quad (\alpha,1,5) \\ (p,1,5) \\ \downarrow \quad (p,1\to 0,5\to 10) \\ (p,0,10) \\ \ldots \end{array} \qquad \begin{array}{l} (0,c,0) \\ \downarrow \quad (0\to 1,c,0\to 7) \\ (1,c,7) \\ \downarrow \quad (1,\delta,7) \\ (1,w,7) \\ \downarrow \quad (1\to 0,\gamma,7) \\ (0,c,7) \\ \ldots \end{array}$$

and the diagram



is commutative.

Having the processes (4) we can reset the run of the process (3) . The problem can be formulated as follows.

Given processes (4) . Find such a process (3) that there exist functors (5) such that the diagram (6) is commutative.

A solution can be obtained in the considered case according to the following idea.

We try to find such Φ_0 that

$$f_1(0) = \vartheta_1(\Phi_0), \quad f_2(0) = \vartheta_2(\Phi_0)$$

Here, must be

$$\Phi_0 = (p,0,c,0)$$

Next, we try to find decompositions

$$f_1(0 \leqslant i) = \vartheta_1(\Phi_1) \cdot \xi_{11}, \quad f_2(0 \leqslant 1) = \vartheta_2(\Phi_1) \cdot \xi_{12}$$

in such a way that one at least of actions ξ_{11}, ξ_{12} reduces to a state, i.e. is an object (of P_1 or P_2 respectively) , i,e, $\xi_{11} = f_1(1)$ or $\xi_{12} = f_2(1)$. In this case it should be

$$\Phi_1 = (p,0,c,0 \rightarrow 5), \quad \xi_{11} = (p,0,5) = f_1(1) , \quad \xi_{12} =$$

$$= (0 \rightarrow 1, c, 5 \rightarrow 7)$$

Farther, we try to find some decompositions of $f_1(i \leqslant 2)$ and of ξ_{12} similar like those of $f_1(0 \leqslant 1)$, $f_2(0 \leqslant 1)$ and so on. We get successively

$$f_1(1 \leqslant 2) = \vartheta_1(\Phi_2) \cdot \xi_{21}, \quad \xi_{12} = \vartheta_2(\Phi_2) \cdot \xi_{22},$$

$$\Phi_2 = (p,0 \rightarrow 1, c, 5), \quad \xi_{21} = (r,1,5) = f_1(2) , \quad \xi_{22} = (1,c,5 \rightarrow 7),$$

$f_1(2 \leqslant 3) = \nu_1(\Phi_3) \cdot \xi_{31}$, $\xi_{22} = \nu_2(\Phi_3) \cdot \xi_{32}$,

$\Phi_3 = (\alpha, 1, c, 5)$, $\xi_{31} = (p, 1, 5) = f_1(3)$, $\xi_{32} = (1, c, 5, 7)$,

$f_1(3 \leqslant 4) = \nu_1(\Phi_4) \cdot \xi_{41}$, $\xi_{32} = \nu_2(\Phi_4) \cdot \xi_{42}$,

$\Phi_4 = (p, 1, c, 5 \rightarrow 7)$, $\xi_{41} = (p, 1 \rightarrow 0, 7 \rightarrow 10)$, $\xi_{42} = (1, c, 7) = f_2(1)$,

$\xi_{41} = \nu_1(\Phi_5) \cdot \xi_{51}$, $f_2(1 \leqslant 2) = \nu_2(\Phi_5) \cdot \xi_{52}$,

$\Phi_5 = (p, 1, \delta, 7)$, $\xi_{51} = (p, 1 \rightarrow 0, 7 \rightarrow 10)$, $\xi_{52} = (1, w, 7) = f_2(2)$,

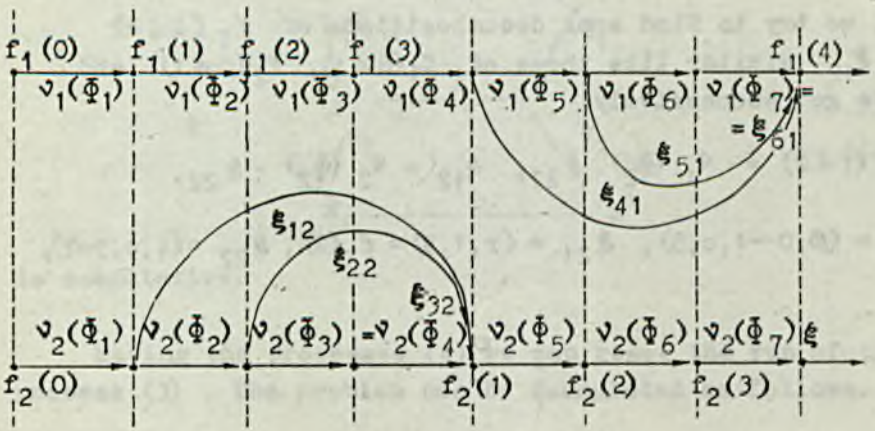
$\xi_{51} = \nu_1(\Phi_6) \cdot \xi_{61}$, $f_2(2 \leqslant 3) = \nu_2(\Phi_6) \cdot \xi_{62}$,

$\Phi_6 = (p, 1 \rightarrow 0, \gamma, 7)$, $\xi_{61} = (p, 0, 7 \rightarrow 10)$, $\xi_{62} = (0, c, 7) = f_2(3)$,

$\xi_{61} = \nu_1(\Phi_7) \cdot \xi_{71}$, $f_2(3 \leqslant 4) = \nu_2(\Phi_7) \cdot \xi_{71}$,

$\Phi_7 = (p, 0, c, 7 \rightarrow 10)$, $\xi_{71} = (p, 0, 10) = f_1(4)$, $\xi_{72} = (0, c, 7 \rightarrow 10)$

and so on, what can be ilustreted graphically as follows



Taking

$$\bar{f}(0) = \Phi_0, \qquad \bar{\varphi}_1(0) = 0, \qquad \bar{\varphi}_2(0) = 0,$$

$$\bar{f}(0 \leqslant 1) = \Phi_1, \qquad \bar{\varphi}_1(1) = 1,$$

$$\bar{f}(1 \leqslant 2) = \Phi_2, \qquad \bar{\varphi}_1(2) = 2,$$

$$\bar{f}(2 \leqslant 3) = \Phi_3, \qquad \bar{\varphi}_1(3) = 3,$$

$$\bar{f}(3 \leqslant 4) = \Phi_4, \qquad \bar{\varphi}_2(1) = 4,$$

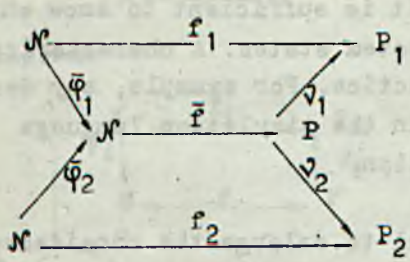
e.t.c. we get a process

$$\mathcal{N} \xrightarrow{\bar{f}} P \tag{7}$$

and functors

$$\mathcal{N} \xrightarrow{\bar{\varphi}_1} \mathcal{N}, \qquad \mathcal{M} \xrightarrow{\bar{\varphi}_2} \mathcal{M} \tag{8}$$

which make commutative the following diagram



(9)

The actions Φ_1, $\Phi_2, \ldots$, ξ_{11}, $\xi_{12}, \ldots, \xi_{21}$, $\xi_{22}, \ldots$ are the only actions which satisfy suitable conditions in this case. As result, the process (7) runs like the first process (3) in such a meaning that its consecutive actions are consecutive actions in (3) . Here, both processes are even identical but it does not need be so in general. For instance, it would not be if the action $(p,0,c,0 \rightarrow 5)$ were composed of the two consecutive actions $(p,0,c,0 \rightarrow 2)$, $(p,0,c,2 \rightarrow 5)$ in (3) . A more precise analysis of this situation will be given in

the next part of the paper.

The above described method of a composed system process reconstruction according to processes of components is applied in various simulation techniques. For instance, to look for decompositions

$$f_1(0 \leqslant 1) = \nu_1(\Phi_1) \cdot \xi_{11}, \quad f_2(0 \leqslant 1) = \nu_2(\Phi_1) \cdot \xi_{12}$$

means to look for the longest action Φ_1 of the whole system, some parts $\nu_1(\Phi_1)$, $\nu_2(\Phi_1)$ of $f_1(0 \leqslant 1)$, $f_2(0 \leqslant 1)$ correspond to which in the components. This action Φ_1 must finish at the earlier of the final moments of $f_1(0 \leqslant 1)$, $f_2(0 \leqslant 1)$, i.e. at the moment 5 when $f_1(0 \leqslant 1)$ finishes. Then we replace the action $f_1(0 \leqslant 1)$ which finishes at the moment 5 by the action $f_1(1 \leqslant 2)$ and try to find the longest action Φ_2 with $\nu_1(\Phi_2)$ being a part of $f_1(1 \leqslant 2)$ and $\nu_2(\Phi_2)$ being a part of ξ_{12} e.t.c. We can do this if we know the processes (4) . In fact, the complete knowledge of (4) is not needed. It is sufficient to know which actions start at concrete system states. A characterization like that is just given in practice. For example, any description of a composed process in the simulation language SIMULA is such a characterization.

It was essential to enlarge the considered system by a clock in the above given case. However, the described method and procedure are valid in general, i.e. for arbitrary systems. It follows from more general facts which will be the matter of the farther considerations.

4. THE GENERAL CASE

In the following, all systems and their components will be treated as categories.

Let P be a system with componets P_i $(i \in I)$. Let $P \xrightarrow{\nu} \prod P_i$ be the injection of P into the product $\prod P_i$ of P_i's and let $P \xrightarrow{i} P_i$ be compositions of ν with the suitable projections $\prod P_i \longrightarrow P_i$.

In general, the problem to characterize a process $\mathcal{N} \xrightarrow{f} P$ by some subprocesses of the process $\mathcal{N} \xrightarrow{f} P \xrightarrow{\nu_i} P_i$ can be formulated as follows.

Given some processes

$$\mathcal{N} \xrightarrow{f_i} P_i \,, \quad i \in I \tag{10}$$

Find such a process

$$\mathcal{N} \xrightarrow{f} P \tag{11}$$

and functors

$$\mathcal{N} \xrightarrow{\varphi_i} \mathcal{N} \tag{12}$$

that all the diagrams

$$\begin{array}{ccc} \mathcal{N} & \xrightarrow{f_i} & P_i \\ {\scriptstyle \varphi_i}\downarrow & & \downarrow{\scriptstyle \nu_i} \\ \mathcal{N} & \xrightarrow{f} & P \end{array} \tag{13}$$

are commutative.

Of course, there may be none such a process (11) with functors (12) or may be many. However, if it exists then the processes (10) satisfy some conditions which will be given now.

For simplicity, we will usually assume that the actions f_i $(m \leqslant n)$ of (10) can not be represented in the form

$$\alpha \,.\, f_i \ (p \leqslant q) \,.\, \beta$$

with $p \leqslant m \leqslant n \leqslant q$ and $p \neq m$ or $q \neq n$. Then the processes (10) will be called non-degenerated.

The product $\prod_{i \in I} \mathcal{N}_i$ with $\mathcal{N}_i = \mathcal{N}$ will be denoted by $\prod \mathcal{N}$. The objects $m \in |\mathcal{N}|^0$ are of the form $(m_i)_{i \in I}$, where $m_i \in |\mathcal{N}_i|^0$ is the projection of m into $\mathcal{N}_i$. The morphisms of $\prod \mathcal{N}$ are of the form $m \leqslant n$ with $m \in |\prod \mathcal{N}|^0$, $n \in |\prod \mathcal{N}|^0$ and $(m_i \leqslant n_i) \in |\mathcal{N}_i|$ for $i \in I$. The projections of the morphisms $\alpha \in |\prod P_i|$ into P_i will be denoted by α_i. Finally, $\prod f_i$ will denote the functor from $\prod \mathcal{N}$ into $\prod P_i$ defined by the formula

$$\prod f_i \ (m \leqslant n) = (f_i \ (m_i \leqslant n_i))_{i \in I}$$

The following theorem holds.

Theorem 1. A functor

$$\prod \mathcal{N} \xrightarrow{\Phi} P \tag{14}$$

with a natural mapping

$$\Phi \cdot \nu \xrightarrow{\lambda} \prod f_i \tag{15}$$

can be assigned to every process (11) and to functors (12) with commutative diagrams (13) in such a way that the following conditions hold:

for every $m \in |\prod \mathcal{N}|^0$ at least one of the projections $\lambda_i(m)$ of $\lambda(m)$ reduces to a state, (16)

for every sequence $m(0) \leqslant m(1) \ldots$ with $m(k) \in |\prod \mathcal{N}|^0$ for which $m(k) \leqslant m \in |\prod \mathcal{N}|^0$ a non-negative integer K exists such that all the actions $\Phi(m(p) \leqslant m(q))$ with $p, q \geqslant K$ reduce to a state. (17)

If the process (10) are non-degenerated then

for every $i \in I$ and non-negative integer p a non-negative integer $M(i,p)$ exists such that for every $m \in |\Pi N|^0$ with $m_i \leqslant p$ and for every $\lambda_j \in I$ for which $\lambda_j(m)$ reduces to a state the inequality $m_j \leqslant M(i,p)$ is satisfied. (18)

Proof. For every $m \in |\Pi N|^0$ such an index i_m exists that

$$m^* = \min_{\text{def } i} \varphi_i(m_i) = \varphi_{i_m}(m_{i_m}) \tag{19}$$

Taking

$$\lambda_i(m) = \nu_i(f(m^* \leqslant \varphi_i(m_i))) \tag{20}$$

we get

$$\lambda_{i_m}(m) = \nu_i(f(m^*))$$

For any $(m \leqslant n) \in |\Pi N|$ the ineguality $m^* \leqslant n^*$ holds. We define

$$\Phi(m \leqslant n) = f(m^* \leqslant n^*) \tag{21}$$

Due to $m^* \leqslant n^* < \varphi_i(n_i)$ and $m^* < \varphi_i(m_i) \leqslant \varphi_i(n_i)$ we have

$$f(m^* \leqslant n^*) \cdot f(n^* < \varphi_i(n_i)) = f(m^* \leqslant \varphi_i(m_i)) \cdot f(\varphi_i(m_i) \leqslant \varphi_i(n_i))$$

But

$$\nu_i(f(\varphi_i(m_i) < \varphi_i(n_i))) = f_i(m_i \leqslant n_i)$$

so that

$$\nu_i(\Phi(m \leqslant n)) \cdot \lambda_i(n) = \lambda_i(m) \cdot f_i(m_i \leqslant n_i)$$

Therefore, the assignement λ defined by the formula

$$\lambda(m) = (\lambda_i(m))_{i \in I}$$

is a natural mapping of $\Phi \cdot \nu$ into Πf_i and satisfies the condition (16).

The property (17) results from the fact that $m(0) \leqslant m(1) \leqslant \ldots \leqslant m$ implies $m^*(0) \leqslant m^*(1) \leqslant \ldots \leqslant m^*$. Namely, an integer K then exists such that $p,q \geqslant K$ implies $m^*(p) = m^*(q)$.

In order to prove (18) observe that each of the sequences $\varphi_i(0)$, $\varphi_i(1)$, $\varphi_i(2)$,... is essentially increasing because of non-degeneracy of (10). Hence, there exists $M(i,p)$ such that $\varphi_j(M(i,p)) > \varphi_i(p)$ for every $j \in I$. If therefore $m_i \leqslant p$ and $m_j > M(i,p)$ for $m \in |\prod \mathcal{N}|^0$ then $\varphi_j(m_j - 1) > \varphi_i(p)$ and $m^* \leqslant \varphi_i(p)$, i.e.

$$m^* \leqslant \varphi_j(m_j - 1) \leqslant \varphi_j(m_j)$$

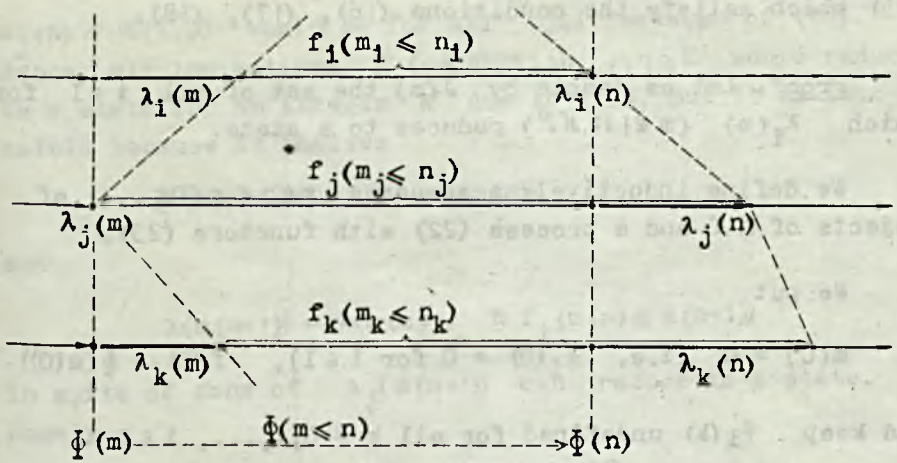
Finally,

$$\lambda_j(m) = \vartheta_j(f(m^* \leqslant \varphi_j(m_j))) =$$

$$\vartheta_j(f(m^* \leqslant \varphi_j(m_j - 1))) \cdot f_j(m_j - 1 \leqslant m_j)$$

and $\lambda_j(m)$ can not reduce to a state because the processes (10) are non-degenerated.*

The functor Φ defined by the formula (21) and the natural mapping λ defined by the formula (20) will be called a <u>standard functor</u> and a <u>standard natural mapping</u> for (11) and (12). Intuitively, $\Phi(m)$ means a state of P that can be considered as the last in (11) before going through any of states $f_i(m_i)$ in processes (10). The morphism $\lambda_i(m)$ can be understood as the action of P_i which should be performed in P_i in order to attain the state $f_i(m_i)$ in the suitable of the processes (10) provided P is in the state $\Phi(m)$. The morphism $\Phi(m \leqslant n)$ means an action of P which moves P from the state $\Phi(m)$ to the state $\Phi(n)$ in (11). All the above can be illustrated graphically as follows.



The condition (17) means that every $\Phi(m \leqslant n)$ is composed at most of a finite number of essentially different non-trivial actions of the form $\Phi(p \leqslant q)$. The condition (18) means that the processes (10) run at comparable rates.

The above theorem 1 can be converted in the following way.

Theorem 2. If the processes (10) are non-degenerated then a process

$$\mathcal{N} \xrightarrow{\bar{f}} P \tag{22}$$

with functors

$$\mathcal{N} \xrightarrow{\bar{\varphi}_i} \mathcal{N} \tag{23}$$

and with commutative diagrams

$$\begin{array}{ccc} \mathcal{N} & \xrightarrow{f_i} & P_i \\ {\scriptstyle \bar{\varphi}_i}\downarrow & & \downarrow{\scriptstyle \nu_i} \\ \mathcal{N} & \xrightarrow{\bar{f}} & P \end{array} \tag{24}$$

can be assigned to every functor (14) with a natural mapping

(15) which satisfy the conditions (16), (17), (18).

Proof. Let us denote by $J(m)$ the set of all $i \in I$ for which $\lambda_i(m)$ $(m \in |\Pi \mathcal{N}|^0)$ reduces to a state.

We define inductively a sequence $m(0) \leqslant m(1) \leqslant \ldots$ of objects of $\Pi \mathcal{N}$ and a process (22) with functors (23).

We put

$$m(0) = 0 \quad (\text{i.e. } m_i(0) = 0 \text{ for } i \in I), \qquad \bar{f}(0) = \Phi(m(0))$$

and keep $\bar{\varphi}_i(k)$ undefined for all $k = 0,1,\ldots$, $i \in I$.

When $m(n) \in |\Pi \mathcal{N}|^0$, $\bar{f}(n) = \Phi(m(n))$ and $\bar{\varphi}_i(k)$ $(0 \leqslant k \leqslant m_i(n)$, $i \in I)$ are defined then we define

$$m_i(n+1) = m_i(n) \quad \text{for} \quad i \notin J(m(n))$$

and

$$m_i(n+1) = m_i(n)+1, \quad \bar{\varphi}_i(m_i(n)) = n \text{ for } i \in J(m(n))$$

and we put

$$m(n+1) = (m_i(n+1)) \quad i \in I$$

$$\bar{f}(n \leqslant n+1) = \Phi(m(n) \leqslant m(n+1))$$

Clearly, all $\bar{\varphi}_i$ can be extended to functors which are defined on some subcategories of $\mathcal{N}$.

As it appears, each $\bar{\varphi}_i$ is defined on the whole $\mathcal{N}$. In other words, for every i,p an unique $n(i,p)$ exists such that $\bar{\varphi}_i(p) = n(i,p)$. The uniqueness is a consequence of the definition of $\bar{\varphi}_i$. The existence results as follows.

For increasing n these of $m_j(n)$ increase for which $j \in J(m(n))$. If none $n(i,p)$ were for the given i,p then

$m_j(n) \leqslant M(i,p)$ would be for all $j \in I$ because of (18). Hence, all the actions $\bar{f}$ $(p \leqslant q)$ with $p,q \geqslant K$ would reduce to a state for an integer K due to (17). But it is impossible because it implies

$$\Phi(m(n) \leqslant m(n+1)) = \Phi(m(n))$$

and

$$\lambda(m(n+1)) = \lambda(m(n)) \cdot \pi f_i(m(n) \leqslant m(n+1))$$

in spite of none of $\lambda_j(m(n+1))$ can reduce to a state. Namely,

$$\lambda_j(m(n)) \cdot f_j(m_j(n) \leqslant m_j(n+1))$$

does not reduce to a state for $j \in J(m(n))$ because the processes (10) are non-degenerated and $\lambda_j(m(n))$ does not reduce to a state for $j \notin J(m(n))$ because of the definition of $J(m)$.

By the definitions of $n(i,p)$ and $\bar{\varphi}_i$ we have

$$m_i(n(i,p)) = p \tag{25}$$

and

$$f_i(p) = \lambda_i(m(n(i,p))) \tag{26}$$

Commutativity of the diagrams (24) results from a simple calculus. Namely,

$$\bar{f}\ (\bar{\varphi}_i(p \leqslant q))) = \bar{f}(n(i,p) \leqslant n(i,q)) = \Phi(m(n(i,p)) \leqslant m(n(i,q)))$$

so that

$$\bar{f}\ (\bar{\varphi}_i(p \leqslant q)) \cdot \lambda(m(n(i,q))) =$$

$$\lambda(m(n(i,p))) \cdot \pi f_i(m(n(i,p)) \leqslant m(n(i,q)))$$

and due to (25)

$$\forall_i\ (\bar{f}\ (\bar{\varphi}_i(p \leqslant q))) = f_i(m_i(n(i,p)) \leqslant m_i(n(i,q))) = f_i(p \leqslant q) \quad *$$

The process (22) and functors (23) defined in the above way will be called standard process and standard functors for Φ , λ. Of course, it does not result from the theorem 2 whether standard functor and standard natural mapping for the process (22) with functors (23) coincide with Φ, λ.

It will be convenient to formulate in the categorical language the facts we shall deal with in the following.

We shall work with the category

$$(P \xrightarrow{\varphi_i} P_i \xrightarrow{f_i} \mathcal{N})_{i \in I} \tag{27}$$

the objects of which are such processes (11) with functors (12) or - in other words - systems

$$(\mathcal{N} \xrightarrow{\varphi_i} \mathcal{N} \xrightarrow{f} P)_{i \in I}$$

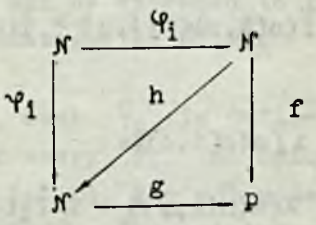
for which the diagrams (13) are commutative and the morphisms

$$(\mathcal{N} \xrightarrow{\varphi_i} \mathcal{N} \xrightarrow{f} P)_{i \in I} \xrightarrow{h} (\mathcal{N} \xrightarrow{\varphi_i} \mathcal{N} \xrightarrow{g} P)_{i \in I}$$

of which are such functors

$$\mathcal{N} \xrightarrow{h} \mathcal{N}$$

for which all the diagrams



are commutative.

According to the theorem 1, a standard functor and its standard natural mapping correspond to every object of the

category (27). As it appears, all the morphisms of the category (27) preserve standard functors and standard natural mappings for all the objects of (27) . Namely, the following theorem holds.

Theorem 3. If there exists a morphism

$$(\mathcal{N} \xrightarrow{\varphi_i} \mathcal{N} \xrightarrow{f} P)_{i \in I} \xrightarrow{h} (\mathcal{N} \xrightarrow{\psi_i} \mathcal{N} \xrightarrow{g} P)_{i \in I} \quad (28)$$

in (27), then the standard functors and natural mappings for $(\mathcal{N} \xrightarrow{\varphi_i} \mathcal{N} \xrightarrow{f} P)_{i \in I}$ and for $(\mathcal{N} \xrightarrow{\psi_i} \mathcal{N} \xrightarrow{g} P)_{i \in I}$ are identical.

Proof. Let Φ, λ be the standard functor and natural mapping for $(\mathcal{N} \xrightarrow{\varphi_i} \mathcal{N} \xrightarrow{f} P)_{i \in I}$ and Ψ, μ the standard functor and natural mapping for $(\mathcal{N} \xrightarrow{\psi_i} \mathcal{N} \xrightarrow{g} P)_{i \in I}$. Using the denotations of the proof of the theorem 1 we get for $(m \leqslant n) \in |\Pi \mathcal{N}|$

$$\Psi(m \leqslant n) = g(\min_i \psi_i(m_i) \leqslant \min_i \psi_i(n_i)) =$$

$$g(\min_i h(\varphi_i(m_i)) \leqslant \min_i h(\varphi_i(n_i))) =$$

$$g(h(\min_i \varphi_i(m_i) \leqslant \min_i \varphi_i(n_i))) =$$

$$f(\min_i \varphi_i(m_i) \leqslant \min_i \varphi_i(n_i)) = \Phi(m \leqslant n)$$

Similarly, for all $m \in |\Pi \mathcal{N}|^0$ and $i \in I$

$$\mu_i(m) = \nu_i(g(\min_i \psi_i(m_i) \leqslant \psi_i(m_i))) =$$

$$\vartheta_i\,(g(\min_i h\,(\varphi_i(m_i)) \leqslant h\,(\varphi_i(m_i))) =$$

$$\vartheta_i\,(g(h(\min_i \varphi_i(m_i) \leqslant \varphi_i(m_i))) =$$

$$\vartheta_i\,(f(\min_i \varphi_i(m_i) \leqslant \varphi_i(m_i))) = \lambda_i(m) \qquad *$$

This theorem implies some consequences for the structure of the category (27). In order to formulate these, let us consider the full subcategory

$$(\Phi,\ \lambda,\ P \xrightarrow{\vartheta_i} P_i \xleftarrow{f_i} \mathcal{N})_{i\in I} \qquad (29)$$

of (27) the objects of which have the common standard functor Φ and standard natural mapping λ. According to the theorem 3 there is no morphism between objects of subcategories (29) with different Φ, λ. Therefore, the category (27) is disjoint union of the categories (29).

Some other subcategories of (27) have more interesting properties.

Let us consider the full subcategory of (27) the objects $(\mathcal{N} \xrightarrow{\varphi_i} \mathcal{N} \xrightarrow{f} P)_{i\in I}$ of which satisfy the following condition:

if $\vartheta_i(f(n \leqslant \varphi_i(p)))$ reduces to a state then $\varphi_i(p) = n$. (30)

It will be denoted by

$$\langle P \xrightarrow{\vartheta_i} P_i \xleftarrow{f_i} \mathcal{N} \rangle_{i\in I} \qquad (31)$$

Its intersections with the categories (29) will be denoted by

$$\langle \Phi,\ \lambda,\ P \xrightarrow{\vartheta_i} P_i \xleftarrow{f_i} \mathcal{N} \rangle_{i\in I} \qquad (32)$$

The category (31) is also disjoint union of (32). Moreover, the following theorem holds.

Theorem 4. If the processes (10) are non-degenerated then each of the non-empty categories (32) has an initial object.

Proof. We shall prove that the standard process (2) and the standard functors (23) for Φ, λ form an initial object of (32).

The process (22) with the functors (23) satisfy (30) because of the construction which was described in the proof of the theorem 2. Namely,

$$\vartheta_i(\bar{f}(n \leqslant \bar{\varphi}_i(p))) = \vartheta_i(\Phi(m(n) \leqslant m(n(i,p)))) =$$

$$\vartheta_i(\Phi(m(n) \leqslant m(n(i,p)))) \cdot \vartheta_i(f(n(i,p))) =$$

$$\vartheta_i(\Phi(m(n) \leqslant m(n(i,p)))) \cdot f_i(p) =$$

$$\vartheta_i(\Phi(m(n) \leqslant m(n(i,p)))) \cdot \lambda_i(m(n(i,p))) =$$

$$\vartheta_i(m(n)) \cdot f_i(m_i(n) \leqslant p)$$

reduces to a state only if $m_i(n) = p$, since the processes (10) are non-degenerated. But $m_i(n) = p$ implies $n = n(i,p) = \bar{\varphi}_i(p)$ according to the definition of $\bar{\varphi}_i$.

If a morphism

$$(\mathcal{N} \xrightarrow{\bar{\varphi}_i} \mathcal{N} \xrightarrow{f} P)_{i \in I} \xrightarrow{h} (\mathcal{N} \xrightarrow{\varphi_i} \mathcal{N} \xrightarrow{f} P)_{i \in I} \qquad (33)$$

there exists then it should be

$$h(\bar{\varphi}_i(p)) = h(n(i,p)) = \varphi_i(p)$$

But

$$\lambda_i(m(n(i,p))) = f_i(p)$$

Therefore,

$$\varphi_i(p) \leqslant \min_j \varphi_j(m_j(n(i,p)))$$

so that $m_i(n(i,p)) = p$ implies

$$\varphi_i(p) = \min_j \varphi_j(m_j(n(i,p)))$$

Hence, must be

$$h(n) = \min_j \varphi_j(m_j(n)) \tag{34}$$

for every $n = n(i,p)$. However, each $n = 0,1,2,\ldots$ is of the form $n(i,p)$ for certain $i \in I$, $p = 0,1,\ldots$. It implies (34) for all $n = 0,1,2,\ldots$.

As result, may be at most one morphism like (33).

Now we shall prove the existence of (33).

Observe, that the function defined by the formula (34) extends uniquely to a functor from $\mathcal{N}$ into $\mathcal{N}$. Therefore, it is sufficient to prove the identity

$$h(\bar{\varphi}_i(p)) = \varphi_i(p) \tag{35}$$

for all $i \in I$ and non-negative interger p, and the identity

$$f(h(n \leqslant n+1)) = \bar{f}(n \leqslant n+1) \tag{36}$$

for all integers $n \geqslant 0$.

Due to $\bar{\varphi}_i(p) = n(1,p)$ we have

$$h(n(i,p)) = \min_j \varphi_j(m_j(n(i,p)))$$

Moreover,

$$\varphi_i(m_i(n(i,p))) = \min_j \varphi_j(m_j(n(i,p)))$$

Indeed,

$$\lambda_i(m(n(i,p))) = \nu_i(f(\min_j \varphi_j(m_j(n(i,p)))) \leqslant \varphi_i(m_i(n(i,p)))$$

and

$$\lambda_i(m(n(i,p))) = f_i(p)$$

reduces to a state as it could not be in the case of inequality because of (30). Therefore, (35) holds for all $n = n(i,p)$, i.e. for all n.

The equality (36) results from the following

$$f(h(n \leqslant n+1)) = \bar{f}(\min_i \varphi_i(m_i(n)) \leqslant \min_i \varphi_i(m_i(n+1))) =$$

$$\Phi(m(n) \leqslant m(n+1)) = f(n \leqslant n+1)$$

Therefore, h is a morphism from $(\mathcal{N} \xrightarrow{\bar{\varphi}_i} \mathcal{N} \xrightarrow{\bar{f}} P)_{i \in I}$ into $(\mathcal{N} \xrightarrow{\varphi_i} \mathcal{N} \xrightarrow{f} P)_{i \in I}$.

Due to the theorem 3, Φ is the standard functor and λ is the standard natural mapping for $(\mathcal{N} \xrightarrow{\bar{\varphi}_i} \mathcal{N} \xrightarrow{\bar{f}} P)_{i \in I}$ so that $(\mathcal{N} \xrightarrow{\bar{\varphi}_i} \mathcal{N} \xrightarrow{\bar{f}} P)_{i \in I}$ is an initial object of (32). *

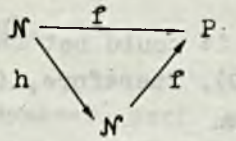
The theorem 4 has a practical significance. Namely, in many cases like that considered in the part 3 of the paper only one functor

$$\Pi \mathcal{N} \xrightarrow{\Phi} P$$

and one natural mapping

$$\Phi \cdot \nu \xrightarrow{\lambda} \Pi f_1$$

there exist which satisfy the condition (16), and these Φ, λ can be obtained by a procedure similar to that described in the part 3. If Φ and λ satisfy the conditions (17) and (18) then the procedure gives the standard process (22) and the standard functors(23)for Φ, λ and all the other processes like (11) and functors like (12) which satisfy the condition (30) can be obtained using decompositions of the form



and taking then

$$\varphi_i = \bar{\varphi}_i \cdot h$$

for $i \in I$.

References

[1] DAHL O.J.: Discrete Event Simulation Languages, in Programming Languages, ed. by F. Genuys, Academic Press, London, 1968

[2] DIJKSTRA E.W.: Cooperating Sequential Processes, in Programming Languages, ed. by F. Genuys, Academic Press, London, 1968

[3] MAC LANE S.: Categories for the Working Mathematician, Springer-Verlag, New York, Heidelberg, Berlin, 1971

[4] WINKOWSKI J.: Processes and Processors, Algorytmy, t. X, nr 17, 1973

PROCESY W SYSTEMACH ZŁOŻONYCH

Streszczenie

Praca dotyczy metod charakteryzacji procesów przebiegających w systemach złożonych. Rozważa się metody, w których proces przebiegający w systemie złożonym z pewnej liczby składowych charakteryzuje się przez procesy przebiegające w składowych. Podstawowy problem, o którym mówi się w pracy, sprowadza się do wyjaśnienia, w jakim stopniu charakteryzacja tego typu wyznacza charakteryzowany proces. Podobnie jak w [4], systemy, systemy złożone i procesy opisuje się za pomocą prostych pojęć teorii kategorii.

ПРОЦЕССЫ В СЛОЖНЫХ СИСТЕМАХ

Резюме

В статье приводятся методы характеризации процессов, протекающих в сложных системах. Рассматриваются методы, в которых процесс протекающий в системе, слогающейся из нескольких компонентов схарактеризован процессами, протекающими в компонентах. Основная проблема, разработанная в статье, состоит в том, чтобы найти ответ на вопрос: в какой степени такого типа характеризация определяет описываемый процесс. Подобно работе [4], системы, сложные системы и процессы описываются с помощью простых понятий теории категории.

ALGORYTMY, XI, N° 20, 1974, 45-57

681.322.06:517.5

PEWNE UWAGI O PROCEDURACH POSZUKIWANIA EKSTREMUM

Krzysztof AMBORSKI
Teresa OSTROWSKA

Instytut Sterowania i Elektroniki
Przemysłowej Politechniki Warszawskiej

Pracę złożono 4.10.1973

W pracy przeprowadzono analizę krytyczną wybranych procedur POWELL i MATRIX poszukiwania ekstremum, opublikowanych w języku ALGOL. Stwierdzono błędy utrudniające lub uniemożliwiające w niektórych przypadkach uzyskanie zbieżności. Podano propozycje środków zaradczych i ich wpływ na działanie procedur.

1. WSTĘP

Algorytmy i programy poszukiwania ekstremum funkcji, zwłaszcza funkcji wielu zmiennych, mają bardzo duże znaczenie w wielu dziedzinach zastosowań, zwłaszcza w teorii sterowania automatycznego.

Zadania identyfikacji, a następnie automatycznej syntezy wielowymiarowych układów sterowania wymagają wielokrotnego obliczania minimum funkcji przy uwzględnieniu ograniczeń nałożonych zarówno na argumenty funkcji, jak i na wartości wynikowe [1].

Przy poszukiwaniu ekstremum często i chętnie wykorzystywana jest metoda Rosenbrocka, jako bardzo pewna w działaniu. Ponieważ jednak istnieje wiele innych metod opisanych algorytmicznie, a nawet oprogramowanych [2], [6] - przy czym niektóre z nich zalecane są jako lepsze - podjęto próbę [5] zastosowania procedur zbudowanych według dwóch takich metod: metody Powella i metody Davidona. Wykonano wiele testów na maszynie cyfrowej GIER wykorzystując, w celu poszukiwania ekstremum warunkowego, procedurę POWELL [2] (nadając jej nazwę POG) zaś w celu poszukiwania ekstremum bezwarunkowego:

- procedurę POWELL , [2]
- procedurę MATRIX , [6](modyfikacja procedury DAVIDON - [2] , [4]).

Wykorzystując oryginalne postacie procedur nie uzyskano zadawalających wyników. Przeanalizowano przyczyny tego stanu rzeczy i wprowadzono modyfikacje, które omówiono poniżej.

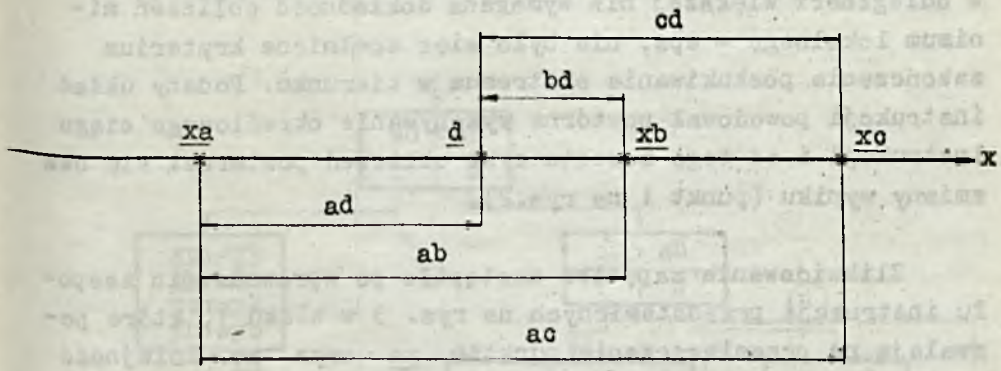
2. MODYFIKACJE PROCEDURY POWELL

W celu poszukiwania minimum w kierunku, w metodzie POWELL [4] po określeniu przedziału, w którym znajduje się punkt ekstremalny, stosowana jest metoda interpolacji kwadratowej. Prawidłowy wybór przedziału ma bardzo istotny wpływ na zbieżność metody.

W testowanej procedurze wybór przedziału następuje w zależności od relacji między odległościami bieżących punktów przedziału obejmującego ekstremum oraz położenia punktu ekstremalnego $\underline{d}$.

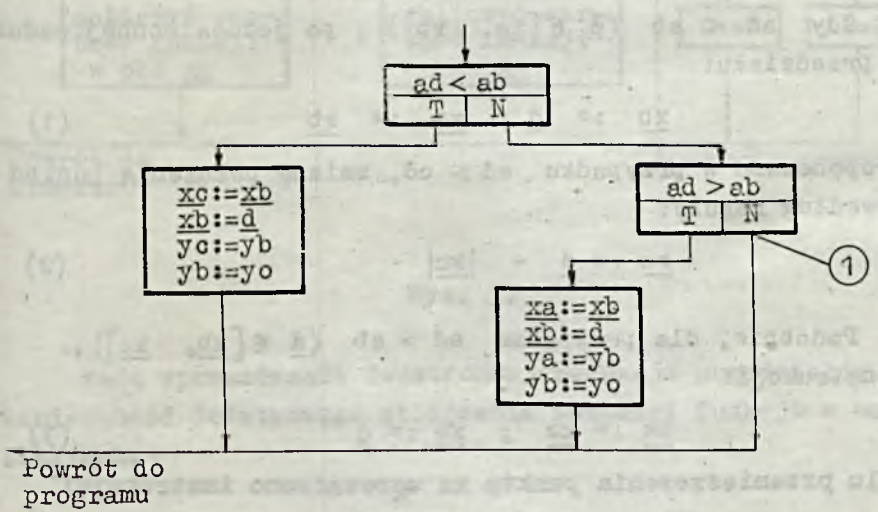
Wielkości te dla przypadku jednowymiarowego przedstawiono na rys.1. W trakcie obliczeń testowych występowały

zapętlenia i nie osiągano zbieżności.



Rys. 1

Na rysunku 2 przedstawiono w postaci schematu blokowego fragment programu realizujący to zadanie [2].



Rys. 2.

Zapętlenia występowały wtedy, gdy ad = ab (punkt, w którym występowało minimum w kierunku, pokrywał się z punktem xb). Jednocześnie punkty xa oraz xc znajdowały się w odległości większej niż wymagana dokładność obliczeń minimum lokalnego - eps, nie było więc spełnione kryterium zakończenia poszukiwania ekstremum w kierunku. Podany układ instrukcji powodował powtórne wykonywanie określonego ciągu instrukcji i od tego momentu cykl obliczeń powtarzał się bez zmiany wyniku (punkt 1 na rys.2).

Zlikwidowanie zapętleń nastąpiło po wprowadzeniu zespołu instrukcji przedstawionych na rys. 3 w bloku 1, które pozwalają na przemieszczenie punktów xa oraz xc. Kolejność przemieszczania uzależniona została od aktualnej wartości wskaźnika u .

Nieznaczna zbieżność metody powodowana była jednostronną redukcją przedziału obejmującego punkt ekstremalny. Wprowadzono więc następujące zmiany:

Gdy ad < ab ($\underline{d} \in [\underline{xa}, \underline{xb}]$) , po jednostronnej redukcji przedziału:

$$\underline{xb} := \underline{d} \; ; \quad \underline{xc} := \underline{xb} \tag{1}$$

zaproponowano w przypadku ad > cd, zmianę położenia punktu xa według reguły:

$$\underline{xa} := \underline{d} - |\underline{xc}| \tag{2}$$

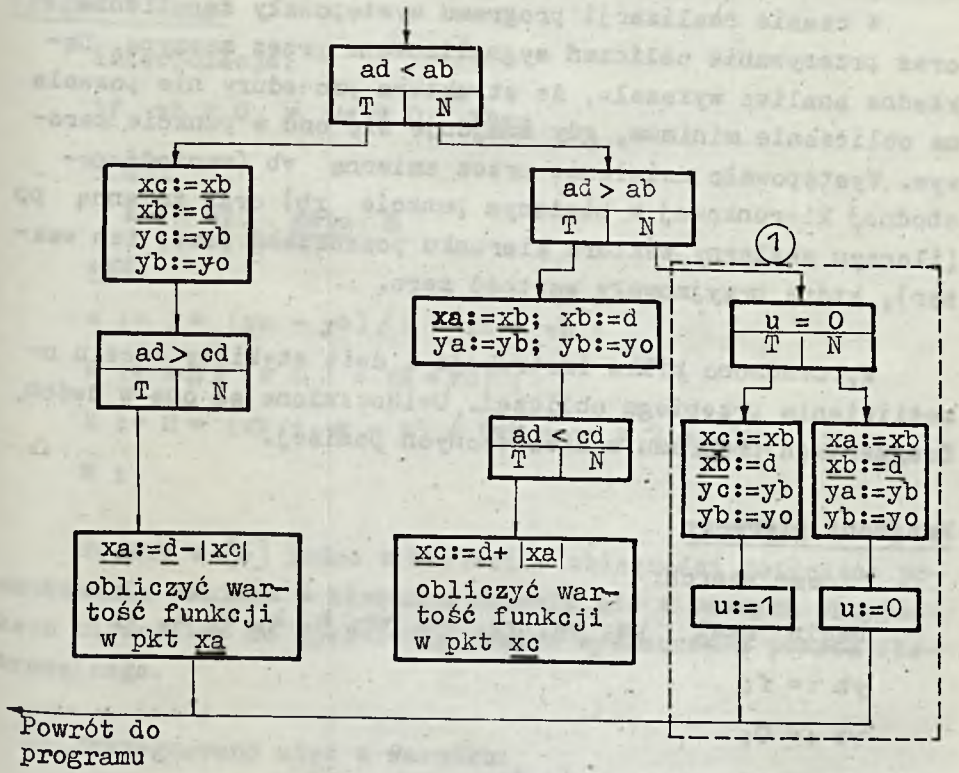
Podobnie, dla przypadku ad > ab ($\underline{d} \in [\underline{xb}, \underline{xc}]$), po instrukcji:

$$\underline{xa} := \underline{xb} \; ; \quad \underline{xb} := \underline{d} \tag{3}$$

w celu przemieszczenia punktu xc wprowadzono instrukcję:

$$\underline{xc} := \underline{d} + |\underline{xa}| \tag{4}$$

Schemat blokowy omówionej części programu wraz z modyfikacjami przedstawiono na rys.3.



Rys. 3.

Wadą wprowadzenia dwustronnej redukcji przedziałów jest konieczność dodatkowego obliczania wartości funkcji w nowych punktach.

3. MODYFIKACJE PROCEDURY MATRIX

Procedura MATRIX [6] oparta jest na metodzie Davidona, ze zmianami mającymi na celu skrócenie czasu obliczeń.

W czasie realizacji programu występowały zapętlenia oraz przerywanie obliczeń sygnalizowane przez maszynę. Dokładna analiza wykazała, że struktura procedury nie pozwala na obliczanie minimum, gdy znajduje się ono w punkcie zerowym. Występowało dzielenie przez zmienną vb (wartość pochodnej kierunkowej w bieżącym punkcie xb) oraz zmienną pp (iloczyn skalarny wektora kierunku poszukiwań przez ten wektor), które przyjmowały wartość zero.

Wprowadzono kilka instrukcji i dwie etykiety w celu umożliwienia przebiegu obliczeń. Uwidocznione są one w dwóch fragmentach programu zamieszczonych poniżej.

Fragment pierwszy

```
Linear search:
begin real  ya, va, yb, vb, vc, h, k, w, z, t;
yb := f;
vb := 0;
if  pp=0  then  goto A;
for  i := 1  step  1  until  n  do
begin
   u := p [i]  := p[i]/ pp;
   vb := vb + u * g[i]
end;
A :
vb := dot (g, p) ;
```

```
if  vb > 0   then  goto  START;
if  vb = 0   then  begin  k := 0,  goto  scale  end;
k := 2 * (est - f) /vb;
```

Fragment drugi

```
interpolacja:
if  vb = 0  ∧  va = 0   then
begin
  k := 0;   goto  B
end;
z := 3 * (ya - yb) / h + va + vb ;
w := sqrt (z ↑ 2 - va * vb) ;
k := h * (vb + w - z) / (vb - va + 2 * w) ;
B :
```

Podane w [6] jedno z kryteriów zbieżności dotyczące poszukiwania minimum w kierunku okazało się w pewnych przypadkach niemożliwe do spełnienia pomimo wyznaczenia punktu ekstremalnego.

Zrezygnowano więc z warunku:

$$(vc)^2 > 10^{-4} * \text{dot}\ (g, g) \qquad (5)$$

gdzie:

vc - wartość pochodnej kierunkowej w punkcie x

dot (g, g) - procedura obliczania iloczynu skalarnego

g - aktualna wartość gradientu funkcji.

Warunek ten nie mógł być spełniony, gdy funkcja badana miała minimum równe zeru w początku układu współrzędnych - wówczas bowiem zmienna vc przyjmowała wartość zero i oczy-

wiście nie mogła być większa od zawsze dodatniej prawej strony nierówności.

Do kryterium wskazującego zakończenie poszukiwania minimum globalnego wprowadzono dodatkowy warunek:

$$\text{abs}\ (vc) < \text{eps} \qquad (6)$$

gdzie eps jest dokładnością zakończenia obliczeń. Spowodowało to skrócenie czasu obliczeń.

4. ZASTOSOWANE TESTY I WYNIKI

Przyjęto następującą postać funkcji celu:

$$f = .2 < \underline{R}, \underline{R} >$$

$\underline{R}$ - wektor n-wymiarowy

$< \underline{R}, \underline{R} >$ - iloczyn skalarny

Przetestowano funkcje jednowymiarowe, dwuwymiarowe i sześciowymiarowe.

Wykorzystując procedurę POWELL dokonano obliczeń dla kilkunastu zestawów danych przyjmując różne wartości parametrów formalnych:

M - wielkość ograniczająca argumenty funkcji minimalizowanej

h - początkowa długość kroku

eps 1 - wymagana dokładność obliczeń minimum globalnego

c - wartość przekroczenia ograniczeń w punkcie startowym

R - arbitralnie wybrany punkt startowy

Przykładowo zamieszczono dane, wyniki oraz krotność

Tabela 1

Seria	Nr przykładu	Dane		I			II		
		h	c	eps	Wyniki obliczeń	Krotność obliczeń funkcji	eps	Wyniki obliczeń	Krotność obliczeń funkcji
A	1	10^{-3}	10	10^{-5}	R=0; f=0	37	10^{-3}	R=0; f=0	37
	2	10^{-1}	10	10^{-5}	wolna zbieżność	143	10^{-2}	R=0; f=0	32
	3	1	10	10^{-5}	zapętlenie	13	10^{-5}	R=0; f=0	32
	4	10	10	10^{-5}	zapętlenie	10	10^{-3}	R=0; f=0	39
	5	10^{2}	10	10^{-5}	zapętlenie	19	10^{-5}	zapętlenie	25
	6	10^{-2}	10^{3}	10^{-5}	zapętlenie	33	10^{-5}	$f=1.2\cdot10^{-41}$	38
B	7	10^{-3}	100	10^{-5}	zapętlenie	50	10^{-5}	$f=1,5\cdot10^{-39}$	60
	8	10^{-2}	100	10^{-5}	$f=7,8\cdot10^{-7}$	38	10^{-5}	$f=7,8\cdot10^{-7}$	38
	9	10^{-2}	10	10^{-5}	$f=7,8\cdot10^{-7}$	38	10^{-5}	$f=7,8\cdot10^{-7}$	38
	10	1	100	10^{-5}	wolna zbieżność	85	10^{-5}	$f=8,8\cdot10^{-3}$	28
	11	10^{-3}	10	10^{-5}	zapętlenie	30	10^{-5}	$f=1,1\cdot10^{-34}$	35

obliczeń funkcji celu dla przypadku funkcji jednowymiarowej. Zadania podzielono na serie A i B. W przykładach serii A współczynnik M = 10, w serii B współczynnik $M = 10^{-3}$. W tabeli 1 podano pozostałe dane oraz wyniki obliczeń. Wyniki oznaczone numerem I dotyczą obliczeń przy zastosowaniu procedury POWELL zaczerpniętej z pracy [2] . Wyniki oznaczone numerem II uzyskano po wprowadzeniu omówionych modyfikacji. W pierwszych dziesięciu przykładach R = 20, w przykładzie 11 natomiast R = 2.

Wykorzystując oryginalną procedurę MATRIX nie uzyskano w żadnym przypadku pozytywnego wyniku. W tabeli 2 zamieszczono wyniki otrzymane po wprowadzeniu modyfikacji oraz wartość parametrów formalnych różnych dla każdego przypadku, gdzie:

est - przybliżona wartość funkcji w punkcie minimum

eps - dokładność zakończona obliczeń.

Tabela 2

Nr przykładu	Dane			Krotność obliczeń funkcji celu	Wyniki obliczeń
	est	eps	punkt startowy		
1	10	10^{-5}	1	8	f=0; R=0
2	1	10^{-2}	2	10	f=0; R=0
3	-1	10^{-3}	5	10	zapętlenie
4	10	10^{-5}	10	24	$f=3,1\cdot10^{-6}$; $R=-13\cdot10^{-4}$

5. WNIOSKI

Po wprowadzeniu modyfikacji liczba iteracji przy wykorzystaniu każdej z metod wynosiła 2.

Z porównania uzyskanych wyników obserwuje się szybszą zbieżność procedury MATRIX (mniejsza liczba obliczeń funkcji celu i wartości ograniczeń).

Szybkość zbieżności procedury MATRIX jest zależna od doboru punktu startowego, który jest pierwszym przybliżeniem punktu optymalnego. Poprawne przyjęcie tego parametru powoduje zmniejszenie liczby obliczeń funkcji celu.

Żadna z zastosowanych metod nie jest całkowicie uniwersalna. Uzyskanie wyniku końcowego w dużej mierze uzależnione jest od liczby argumentów funkcji celu (nie uzyskano pozytywnego wyniku w przypadku problemu sześciowymiarowego).

Na przebieg obliczeń i szybość zbieżności procedury POWELL istotny wpływ ma wartość początkowej długości kroku h . Jest ona uzależniona od zadanej wartości ograniczeń M. Na podstawie uzyskanych wyników stwierdzono, że rząd wielkości współczynnika h powinien być najwyżej równy rzędowi ograniczeń M.

Dobór współczynników c oraz eps jest dowolny.

Literatura

[1] AMBORSKI K.: Synteza liniowych układów regulacji automatycznej za pomocą maszyn cyfrowych, rozprawa doktorska, Politechnika Warszawska, 1972.

[2] Biblioteka programów optymalizacji statycznej, praca zbiorowa pod red. J. Szymanowskiego, Instytut Automatyki Politechniki Warszawskiej, 1970.

[3] DUBNICKI H., ZORYCHTA K.: Metody programowania wypukłego, Uniwersytet Warszawski, 1972.

[4] FINDEISEN W., SZYMANOWSKI J., WIERZBICKI A.: Metody obliczeniowe optymalizacji, Wyd. Politechniki Warszawskiej, 1972.

[5] OSTROWSKA T.: Metody poszukiwania ekstremum przy ograniczeniach niefunkcyjnych, praca magisterska, Instytut Sterowania i Elektroniki Przemysłowej Politechniki Warszawskiej, 1973.

[6] TESSAROWICZ M.: Minimalizacja funkcji wielu zmiennych, Wyd. Uniwersytetu Warszawskiego, zesz. 6, 1967.

НЕКОТОРЫЕ ЗАМЕЧАНИЯ О ПРОЦЕДУРАХ ИСКАНИЯ ЭКСТРЕМУМ

Резюме

В работе проведено критический анализ избранных процедур POWELL и MATRIX искания экстремум, опубликованных на языке АЛГОЛ. Подтверждено ошибки препятствующие или делающие в некоторых случаях невозможным получение конвергенции. Заявлено предложения средств улучшения и их влияние на деятельность процедур.

SOME REMARKS ON EXTREMUM-SEEKING PROCEDURES

Summary

The critical analysis of specific procedures POWELL and MATRIX of search for extremum, published in ALGOL language, is given in the paper. The serious errors, making convergence difficult or impossible in some cases, have been found. Proposals of improvement are given and their result on procedures action is considered.

Warszawa, ul. Krzywickiego 34
A.M.

ALGORYTMY, XI, N° 20, 1974, 59-71

517.11:517.5

WARUNEK ISTNIENIA DEKOMPOZYCJI ILOCZYNOWEJ FUNKCJI BOOLOWSKIEJ

Wojciech ZAWADZKI

Studium Doktoranckie
Politechniki Warszawskiej

Pracę złożono 16.07.1973

W pracy sformułowano warunek konieczny i dostateczny istnienia dekompozycji iloczynowej funkcji boolowskiej. Przedstawiona metoda, oparta na pojęciu tzw. operatora Q, wykazuje wiele zalet w porównaniu z innymi metodami algebraicznymi.

1. WSTĘP

Problemem dekompozycji zajmowało się wielu autorów. Większość prac dotyczyła tzw. dekompozycji dysjunkcyjnej [2, 3, 7, 6]. Z punktu widzenia teorii i zastosowań, równie ważna jest dekompozycja iloczynowa [3, 6].

Niech $X=\{x_1, x_2, \ldots, x_n\}$, oraz A, B, C będą podzbiorami właściwymi X, takimi że:

$$A \cup B \cup C = X, \quad A \cap B = B \cap C = C \cap A = \emptyset, \quad A \neq \emptyset, \quad C \neq \emptyset .$$

Funkcja boolowska $F(X)= F(A, B, C)$ posiada dekompozycję

iloczynową, jeśli może być przedstawiona w postaci:

$$F(A, B, C) = f(A, B) \cdot g(B, C) .$$

W przypadku, gdy $A=\{x_a\}$, $C=\{x_b\}$, to jak wykazano w [3] . warunkiem koniecznym i dostatecznym istnienia dekompozycji $F(X)= f(x_a, B) \cdot g(B, x_b)$ jest, aby:

$$\frac{\partial F}{\partial x_a} \cdot \frac{\partial F}{\partial x_b} = F \cdot \frac{\partial^2 F}{\partial x_a \partial x_b} ,$$ gdzie $\frac{\partial F}{\partial x_a}$ jest pochodną funkcji $F(X)$ względem zmiennej x_a [1] .

Analogicznie dla $\overline{\overline{AUC}} > 2$ wykazano [3] , że na to, aby $F(A, B, C)= f(A, B) \cdot g(B, C)$, potrzeba i wystarcza, żeby spełniony był warunek

$$\frac{\partial F}{\partial A} \cdot \frac{\partial F}{\partial C} = F \cdot \frac{\partial^2 F}{\partial A \partial C} ,$$ oraz istniały wszystkie dekompozycje podziorowe postaci $f(A', B') \cdot g(B', C')$, gdzie $A' \subset A$, $C' \subset C$, $B \subset B'$.

Celem niniejszego artykułu jest uproszczenie przytoczonych powyżej warunków.

2. WARUNEK ISTNIENIA DEKOMPOZYCJI ILOCZYNOWEJ

Niech $A=\{x_1^t, \ldots, x_k^t\}$ i $C=\{x_1^g, \ldots, x_m^g\}$. Wprowadźmy następujące oznaczenia:

$$F_1^A \overset{df}{=} F(\underbrace{1, 1, \ldots, 1}_{k \text{ jedynek}}, B, C) , \quad F_1^C \overset{df}{=} F(A, B, \underbrace{1, 1, \ldots, 1}_{m \text{ jedynek}}) .$$

$F_{01}^{AC} \overset{df}{=} F(\underbrace{0, 0, \ldots, 0}_{k \text{ zer}}, B, \underbrace{1, 1, \ldots, 1}_{m \text{ jedynek}})$ i analogicznie

F_0^A, F_0^C, F_{10}^{AC} itd.

Definicja 1 [6]

a/ $\frac{\partial F}{\partial A} = F_1^A \cup F_0^A$

b/ $\frac{\partial^2 F}{\partial A \, \partial C} = \frac{\partial}{\partial C}\left(\frac{\partial F}{\partial A}\right)$

Z powyższej definicji wynikają natychmiast następujące własności:

$\frac{\partial}{\partial A}(f \cup g) = \frac{\partial f}{\partial A} \cup \frac{\partial g}{\partial A}$; $\frac{\partial}{\partial A}(f(A) \cdot g(B)) = \frac{\partial f(A)}{\partial A} \cdot g(B)$;

$\frac{\partial f(B)}{\partial A} = f(B)$; $\frac{\partial^2 F}{\partial A \, \partial C} = \frac{\partial^2 F}{\partial C \, \partial A}$.

Będziemy korzystać z następującego rozwinięcia funkcji boolowskiej:

$$F(A, B, C) = \bigcup_{i=0}^{H} F_i(B, C) \cdot P_i(A) \qquad (1)$$

gdzie $H = 2^{\overline{A}} - 1$,

$P_0(A) = \bar{x}_1^t \bar{x}_2^t \ldots \bar{x}_{k-1}^t \bar{x}_k^t$,

$P_1(A) = \bar{x}_1^t \bar{x}_2^t \ldots \bar{x}_{k-1}^t x_k^t$,

$$P_2(A) = \bar{x}_1^t \bar{x}_2^t \ldots x_{k-1}^t \bar{x}_k^t ,$$

$$\vdots$$

$$P_{H-1}(A) = x_1^t x_2^t \ldots x_{k-1}^t \bar{x}_k^t ,$$

$$P_H(A) = x_1^t x_2^t \ldots x_{k-1}^t x_k^t .$$

$F_i(B, C)$ jest funkcją, która powstaje z $F(A, B, C)$ przez podstawienie takiej kombinacji wartości zmiennych należących do A, że $P_i(A) = 1$.

Jeśli do funkcji $F_i(B, C)$ zastosujemy rozwinięcie (1) względem zmiennych należących do zbioru C, to otrzymamy następujący wzór:

$$F(A, B, C) = \bigcup_{i=0}^{H} \bigcup_{j=0}^{G} F_{ij}(B) \cdot P_i(A) \cdot P_j(C), \qquad (2)$$

w którym F_{ij}, P_j są zdefiniowane analogicznie jak poprzednio, natomiast $G = 2^{\overline{\overline{C}}} - 1$.

Dowód. Warunek konieczny.

Wyznaczamy funkcje występujące po lewej i po prawej stronie warunku (1) :

$$\frac{\partial F}{\partial x_a} = g(B, x_b) \cdot \frac{\partial f}{\partial x_a} , \quad \frac{\partial F}{\partial x_b} = f(x_a, B) \cdot \frac{\partial g}{\partial x_b} , \quad \frac{\partial^2 F}{\partial x_a \partial x_b} =$$

$$= \frac{\partial f}{\partial x_a} \cdot \frac{\partial g}{\partial x_b} , \qquad (3)$$

skąd $\frac{\partial F}{\partial x_a} \cdot \frac{\partial F}{\partial x_b} = F \cdot \frac{\partial^2 F}{\partial x_a \partial x_b}$, C.B.D.C.

Warunek dostateczny.

Korzystając z definicji 1, warunek (3) można przedstawić w postaci:

$$(F_0^{x_a} \cup F_1^{x_a}) \cdot (F_0^{x_b} \cup F_1^{x_b}) = F \cdot (F_{0\,0}^{x_a\,x_b} \cup F_{0\,1}^{x_a\,x_b} \cup F_{1\,0}^{x_a\,x_b} \cup F_{1\,1}^{x_a\,x_b}) \qquad (4)$$

Stosując do funkcji występujących w powyższej równości rozwinięcia (1) i (2) względem zmiennych $A = \{x_a\}$ i $C = \{x_b\}$ otrzymujemy postać równoważną (4) :

$$(x_b F_{01} \cup \bar{x}_b F_{00} \cup x_b F_{11} \cup \bar{x}_b F_{10}) \cdot (x_a F_{10} \cup \bar{x}_a F_{00} \cup x_a F_{11} \cup \bar{x}_a F_{01}) = (x_a x_b F_{11} \cup \bar{x}_a x_b F_{01} \cup x_a \bar{x}_b F_{10} \cup \bar{x}_a \bar{x}_b F_{00}) \cdot (F_{00} \cup F_{01} \cup F_{10} \cup F_{11}), \qquad (5)$$

gdzie F_{00}, F_{01} ... oznaczają odpowiednio $F_{0\,0}^{x_a x_b}$, $F_{0\,1}^{x_a x_b}$ itd.

Równość (5) zgodnie z założeniem jest prawdziwa dla każdej kombinacji wartości x_a, x_b, skąd otrzymujemy następujący układ równoważny:

$$(F_{00} \cup F_{10}) \cdot (F_{00} \cup F_{01}) = F_{00},$$

$$(F_{01} \cup F_{11}) \cdot (F_{00} \cup F_{01}) = F_{01},$$

$$(F_{00} \cup F_{10}) \cdot (F_{10} \cup F_{11}) = F_{10},$$

$$(F_{01} \cup F_{11}) \cdot (F_{10} \cup F_{11}) = F_{11}.$$

Stosując prawa pochłaniania dostajemy:

$$\begin{aligned} F_{00} \cup F_{01} \cdot F_{10} &= F_{00}, \\ F_{01} \cup F_{00} \cdot F_{11} &= F_{01}, \\ F_{10} \cup F_{00} \cdot F_{11} &= F_{10}, \\ F_{11} \cup F_{01} \cdot F_{10} &= F_{11}. \end{aligned} \tag{6}$$

Układ (6) jest spełniony dla każdej kombinacji wartości zmiennych należących do zbioru B.

Zauważmy, że funkcja $F(x_a, B, x_b)$ posiada dekompozycję $f(x_a, B)$. $g(B, x_b)$ wtedy i tylko wtedy, gdy dla każdej kombinacji wartości zmiennych z B, spełniony jest następujący układ równości:

$$\begin{aligned} F_{00} &= f_0^{x_a} \cdot g_0^{x_b}, \\ F_{01} &= f_0^{x_a} \cdot g_1^{x_b}, \\ F_{10} &= f_1^{x_a} \cdot g_0^{x_b}, \\ F_{11} &= f_1^{x_a} \cdot g_1^{x_b}. \end{aligned} \tag{7}$$

W tabeli 1, dla każdej kombinacji wartości funkcji F_{00} (B),

$F_{01}(B)$... spełniających warunek (6), podano funkcje $f_0^{x_a}(B)$, $g_0^{x_b}(B)$... spełniające (7), co kończy dowód.

Tabela 1

F_{00}	F_{01}	F_{10}	F_{11}	$f_0^{x_a}$	$f_1^{x_a}$	$g_0^{x_b}$	$g_1^{x_b}$
0	0	0	0	0	0	0	0*)
0	0	0	1	0	1	0	1
0	0	1	0	0	1	1	0
0	0	1	1	0	1	1	1
0	1	0	0	1	0	0	1
0	1	0	1	1	1	0	1
1	0	1	0	1	1	1	0
1	0	0	0	1	0	1	0
1	1	0	0	1	0	1	1
1	1	1	1	1	1	1	1

Dla przypadku $A \overline{\cup} C \geqslant 2$ obowiązuje następujące twierdzenie:

Twierdzenie 1

Warunkiem koniecznym i dostatecznym istnienia dekompozycji $f(A, B) \cdot g(B, C)$ funkcji $F(A, B, C)$ jest, aby

$$\frac{\partial F}{\partial A} \cdot \frac{\partial F}{\partial C} = F \cdot \frac{\partial^2 F}{\partial A \, \partial C} \tag{8}$$

oraz aby istniały wszystkie dekompozycje podzbiorowe.

Pełny dowód tego twierdzenia zostanie pominięty, gdyż jest on analogiczny, jak w pracy [3]. Podamy jedynie szkic

*) jedna z tych funkcji może być różna od zera.

dowodu warunku dostatecznego, pokazując w jaki sposób należy wykorzystać założenie o istnieniu dekompozycji podzbiorowych.

Warunek dostateczny - szkic dowodu:

Przedstawiając funkcję F (A, B, C) w postaci $F(A, B, C) = \bigcup_{i=0}^{H} \bigcup_{j=0}^{G} P_i(A)\, P_j(C)\, F_{ij}(B)$ obliczamy kolejno funkcje $\frac{\partial F}{\partial A}$, $\frac{\partial F}{\partial C}$, $\frac{\partial^2 F}{\partial A \partial C}$. W rezultacie warunek (8) przyjmuje postać:

$$\bigcup_{i=0}^{H} \bigcup_{j=0}^{G} P_i(A)\, P_j(C)(F_{0j} \cup F_{Hj})(F_{i0} \cup F_{iG}) = \bigcup_{i=0}^{H} \bigcup_{j=0}^{G} P_i(A)$$

$$P_j(C)(F_{00} \cup F_{0G} \cup F_{H0} \cup F_{HG})\, F_{ij}$$

skąd $$F_{ij}(F_{00} \cup F_{0G} \cup F_{HC} \cup F_{HG}) = (F_{0j} \cup F_{Hj})(F_{i0} \cup F_{iG}) \quad (9)$$

dla każdej pary i, j, oraz dla każdej kombinacji wartości zmiennych należących do zbioru B. Zauważmy, że dla tych kombinacji, dla których $F_{00} \cup F_{0G} \cup F_{H0} \cup F_{GH} = 1$, warunek (9) można zapisać w postaci

$$F_{ij} = (F_{0j} \cup F_{Hj})(F_{i0} \cup F_{iG}),$$

stąd jeśli przyjmiemy $f(A, B) = \bigcup_{i=0}^{H} (F_{i0}(B) \cup F_{iG}(B))$

$P_i(A)$ oraz $g(B, C) = \bigcup_{j=0}^{G} (F_{0j}(B) \cup F_{Hj}(B))\, P_j(C)$,

to $f(A, B) \cdot g(B, C) = \bigcup_{i=0}^{H} \bigcup_{j=0}^{G} F_{ij}(B)\, P_i(A)\, P_j(C)$

i dekompozycja istnieje. W pozostałych kombinacjach, dla

których $F_{00} \cup F_{0G} \cup F_{H0} \cup F_{HG} = 0$, należy ze zbioru A (C) usunąć dowolną zmienną (tzn. dołączyć ją do zbioru B) i powtórzyć ten sam tok postępowania.

Można pokazać, że na każdym etapie otrzymuje się tożsamości analogiczne jak (9), a zmianie ulegają jedynie wartości wskaźników H i G. W efekcie rozpatruje się dekompozycje, w których zbiory A i C zawierają tylko jedną zmienną. Wówczas warunek (8) jest jednocześnie warunkiem dostatecznym, co kończy dowód, gdyż dla każdej kombinacji wartości zmiennych należących do zbioru B, znaleziono funkcje f (A, B) i g (B, C), których iloczyn daje w wyniku F (A, B, C).

Przedstawioną metodę zilustrujemy obecnie na przykładzie:

sprawdźmy, czy dla funkcji $F(X) = x_1 x_2 \bar{x}_3 \cup x_3 x_4$ istnieje dekompozycja $f(x_1, x_2, x_3) \cdot g(x_3, x_4)$.

Badamy warunki na dekompozycje podzbiorowe:

$$\frac{QF}{Qx_1} = x_2\bar{x}_3 \cup x_3x_4, \quad \frac{QF}{Qx_4} = x_1x_2\bar{x}_3 \cup x_3, \quad \frac{Q^2F}{Qx_1\,Qx_4} = x_2\bar{x}_3 \cup x_3,$$

$$\frac{QF}{Qx_1} \cdot \frac{QF}{Qx_4} = x_1x_2\bar{x}_3 \cup x_3x_4 = F \cdot \frac{Q^2F}{Qx_1\,Qx_4},$$

$$\frac{QF}{Qx_2} = x_1\bar{x}_3 \cup x_3x_4, \quad \frac{Q^2F}{Qx_2\,Qx_4} = x_3 \cup x_1\bar{x}_3,$$

$$F \cdot \frac{Q^2F}{Qx_2\,Qx_4} = x_3x_4 \cup x_1x_2\bar{x}_3 = \frac{QF}{Qx_2} \cdot \frac{QF}{Qx_4}.$$

Sprawdzamy warunek konieczny:

$$\frac{QF}{Q\{x_1, x_2\}} = \bar{x}_3 \cup x_3 \bar{x}_4, \quad \frac{Q^2F}{Q\{x_1, x_2\}\, Qx_4} = 1,$$

$$\frac{QF}{Q\{x_1, x_2\}} \cdot \frac{QF}{Qx_4} = x_1 x_2 \bar{x}_3 \cup x_3 x_4 = F \cdot \frac{Q^2F}{Q\{x_1, x_2\}\, Qx_4} \cdot$$

$$F(X) = (x_1 x_2 \cup x_3) \cdot (\bar{x}_3 \cup x_3 x_4) .$$

Powyższy przykład dowodzi, że w przypadku obliczeń "ręcznych", wyznaczanie funkcji $\frac{QF}{QA}$ nie nastręcza większych trudności, zwłaszcza, gdy funkcja dana jest w jednej z postaci kanonicznych: dysjunkcyjnej lub koniunkcyjnej. Można pokazać (dowody pominiemy), że twierdzenia analogiczne do sformułowanych w niniejszym artykule obowiązują dla przyrostów funkcji [5] . Rachunek przyrostów jest szczególnie efektywny dla postaci "suma modulo 2 iloczynów". W realizacji maszynowej napotykamy na innego rodzaju problemy; scharakteryzujemy je w następnym rozdziale.

3. MASZYNOWE METODY WYZNACZANIA DEKOMPOZYCJI.

Problem poszukiwania rozkładów funkcji boolowskiej na funkcje prostsze jest bardzo skomplikowany. Metoda tablic dekompozycji zaproponowana przez Ashenhursta [2] , a rozwinięta przez Curtisa [4] wymaga przetestowania 2^n-2-n tablic, gdzie n jest liczbą zmiennych funkcji. Metoda Akersa [1]prowadzi do warunku algebraicznego o niezwykle skomplikowanej formie. Stosunkowo najprostszy warunek podał Bańkowski w [3] , a także Thayse [6] . Zasadnicza wada obu metod wynika wprost z definicji operatorów $\frac{qF}{qA}$ i $\frac{\partial F}{\partial A}$; mamy bowiem:

$\frac{qF}{qA} = \frac{q}{qx_k}\left(\frac{q}{qx_{k-1}}\left(\cdots\left(\frac{q}{qx_2}\left(\frac{qF}{qx_1}\right)\right)\cdots\right)\right)$ i analogicznie dla $\frac{\partial F}{\partial A}$. Wynika stąd, że aby wyznaczyć $\frac{qF}{qA}\left(\frac{\partial F}{\partial A}\right)$ należy bądź pamiętać $\frac{qF}{q\{A-x_k\}}\left(\frac{\partial F}{\partial\{A-x_k\}}\right)$, bądź też obliczać oba operatory bezpośrednio z funkcji F. W pierwszym przypadku wyniki pośrednie zajmują stosunkowo duże obszary pamięci, w drugim - czas realizacji algorytmu ulega znacznemu wydłużeniu. Wady tej unika się stosując operator Q. Funkcję $\frac{QF}{QA}$ można bowiem znaleźć bezpośrednio z F identycznie jak w [8] obliczano $\frac{\partial F}{\partial x_i}$, uwzględniając jedynie, że:

$\frac{QF}{QA}$ = 1 dla tych i tylko tych kombinacji,

dla których $F_s \cup F_{s'} = 1$, gdzie F_s oznacza

wartość funkcji dla s-tej kombinacji, $s' = s + \sum 2^{v-1}$,

v jest zbiorem indeksów zmiennych z A; natomiast

w [8] korzystano z faktu, że $\frac{\partial F}{\partial x_i}$ = 1 dla takich s,

dla których $F_s \oplus F_{s+2^{i-1}} = 1$.

4. ZAKOŃCZENIE.

W pracy pokazano, w jaki sposób można uprościć badanie dekomponowalności funkcji boolowskich stosując operator Q. Zamieszczony przykład ilustrujący wprowadzoną metodę wykazał, iż można stosunkowo szybko i łatwo znajdować $\frac{QF}{QA}$, gdy funkcja dana jest w postaci dysjunkcyjnej lub koniunkcyjnej. W realizacji maszynowej, przewaga metody Q nad innymi wynika wprost z definicji operatora Q, pozwalając znacznie skrócić czas wyszukiwania wszystkich dekompozycji iloczynowych danej funkcji.

Literatura

[1] AKERS S.B.: On a Theory of Boolean Functions, J.SIAM, t.7, grudzień 1959.

[2] ASHENHURST R.L.: The Decomposition of Switching Functions, Proc. Int. Symp. Theory of Switching, Ann. Computation Lab., Cambridge Mass. Harvard Univ. Press, t.29, 1959, s. 74-116.

[3] BAŃKOWSKI J.: Difference Theorems on Decomposition of Boolean Functions /rękopis/.

[4] CURTIS H.A.: A New Approach to the Design of Switching Circuits, Van nostrand, Princeton, 1962.

[5] KOWALSKI S.: Pochodne formalne funkcji boolowskich i niektóre ich zastosowania w teorii funkcji przełączających, rozprawa doktorska, Gdańsk 1969.

[6] THAYSE A.: Philips Res. Repts 26, 1971, s. 229-246.

[7] THAYSE A.: A Fast Algorithm for the Proper Decomposition of Boolean Functions, Philips Res. Repts 27, 1972.

[8] ZAWADZKI W.: Synteza sieci przełączających na podstawie własności różnicowych funkcji, praca magisterska, Politechnika Warszawska, 1971.

УСЛОВИЕ СУЩЕСТВОВАНИЯ УМНОЖИТЕЛЬНОГО РАЗЛОЖЕНИЯ ФУНКЦИИ

Резюме

В работе представлено необходимое и достаточное условие существования умножительного разложения функции Буля. Представленный метод, основан на понятию так называемого оператора Q указывает много ценностей по сравнению с другими алгебраическими методами.

CONDITION FOR PRODUCT FUNCTION DECOMPOSITION EXISTENCE

Summary

In the work the necessary and sufficient condition for the existence of decomposition of Boolean product function is formulated. Presented method, based on the notion of so-called operator Q, has many advantages in comparison to other algebraic methods.

Warszawa, ul. Krzywickiego 34
A.M.

ALGORYTMY, XI, N° 20, 1974, 73-83 512.25

ZASTOSOWANIE SCHEMATU CHOLESKY'EGO DO ROZWIĄZYWANIA UKŁADÓW RÓWNAŃ LINIOWYCH

Andrzej RAKUS
Centralne Laboratorium
Gazownictwa

Pracę złożono 23.03.1971

Artykuł opisuje metodę rozwiązywania układów równań liniowych w postaci Ax = b, gdzie macierz A zawiera dużą ilość elementów zerowych. Metoda ta wymaga rozwiązywania układu równań algorytmem Cholesky'ego i zastosowanie jej prowadzi do dużej oszczędności miejsc w pamięci maszyny matematycznej oraz czasu liczenia w porównaniu z klasycznymi metodami. W niektórych zastosowaniach (np. przy projektowaniu sieci elektrycznych, gazowych czy wodociągowych) daje oszczędność ok. 70% czasu liczenia.

W wielu zagadnieniach technicznych zachodzi potrzeba rozwiązywania układów równań liniowych Ax = b o szczególnej postaci. Niniejszy artykuł zajmuje się takimi układami, gdzie macierz A jest symetryczna oraz ma dużą liczbę elementów zerowych.

Wiele metod rozwiązywania takich układów równań, mimo ich modyfikacji, często nie wykorzystuje własności takich macierzy. Zagadnienie to ma szczególne znaczenie, kiedy układ postaci Ax = b rozwiązać należy wiele razy, np. przy rozwiązywaniu układów równań nieliniowych, gdzie poprzez linearyzację dochо-

dzi się do układu równań liniowych i jedno rozwiązanie jest tylko jedną z iteracji.

Warto tu rozpatrzyć metodę Cholesky'ego, która po pewnej modyfikacji okazała się bardzo ekonomiczna jeżeli chodzi o miejsce w pamięci maszyny matematycznej oraz szybkość liczenia.

1. SCHEMAT CHOLESKY'EGO

Jeżeli mamy układ równań liniowych:

$$Ax = b \tag{1}$$

to możemy macierz A przedstawić w postaci iloczynu macierzy

$$A = CD \tag{2}$$

gdzie C jest macierzą dolnotrójkątną, a D górnotrójkątną z elementami równymi jedności na głównej przekątnej, a więc elementy tych macierzy będą określone

$$\begin{aligned} c_{ij} &\neq 0 \quad \text{dla} \quad i \geqslant j \\ c_{ij} &= 0 \quad \text{dla} \quad i < j \\ d_{ij} &\neq 0 \quad \text{dla} \quad i < j \\ d_{ij} &= 1 \quad \text{dla} \quad i = j \\ d_{ij} &= 0 \quad \text{dla} \quad i > j \end{aligned} \tag{3}$$

Tak określony rozkład istnieje i jest jednoznaczny, gdy wszystkie główne minory macierzy A są różne od zera, tzn.

$$a_{11} \neq 0, \quad \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \neq 0, \ \ldots\ldots, \det A \neq 0; \tag{4}$$

Ponieważ $x = A^{-1} b$

więc możemy zapisać

$$x = D^{-1} C^{-1} b \tag{5}$$

i wyliczenie macierzy odwrotnych D^{-1} oraz C^{-1} jest dużo łatwiejsze, gdyż są one trójkątne.

Wprowadzając wektor pomocniczy y możemy zapisać

$$y = C^{-1} b, \qquad x = D^{-1} y \tag{6}$$

Wzory wg Cholesky'ego na otrzymanie macierzy C i D oraz wektorów y i x mają postać

$$c_{i1} = a_{i1} \tag{7}$$

$$c_{ij} = a_{ij} - \sum_{k=1}^{j-1} c_{ik} d_{kj}; \quad i \geqslant j > 1$$

oraz

$$d_{1j} = \frac{a_{1j}}{c_{11}} \tag{8}$$

$$d_{ij} = \frac{1}{c_{ii}} \left(a_{ij} - \sum_{k=1}^{i-1} c_{ik} d_{kj} \right); \quad j > i > 1$$

następnie

$$y_1 = \frac{b_1}{c_{11}} \tag{9}$$

$$y_i = \frac{1}{c_{ii}} \left(b_i - \sum_{k=1}^{i-1} c_{ik} y_k \right)$$

$$i = 2, \ldots, n$$

$$x_n = y_n \tag{10}$$

$$x_i = y_i - \sum_{k=i+1}^{n} d_{ik} x_k; \quad i = n-1, n-2, \ldots, 1$$

Gdy macierz A jest symetryczna, tzn.

$$a_{ij} = a_{ji}$$

wówczas

$$d_{ij} = \frac{c_{ji}}{c_{jj}}$$

i możemy wtedy obliczać tylko macierz C pamiętając o tej zależności oraz o tym, że $d_{ii} = 1$.

Załóżmy teraz, że mamy dany tylko górny trójkąt macierzy symetrycznej A.

Możemy otrzymać macierz C^T na miejscu tegoż górnego trójkąta wg wzoru

$$c'_{ij} = a_{ij} - \sum_{k=1}^{i-1} a_{kj} \frac{a_{ki}}{a_{kk}} ; \quad 1 < i \leqslant j \tag{11}$$

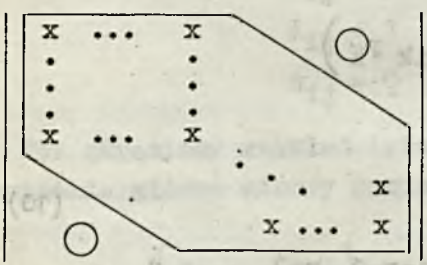
gdyż

$$c'_{1j} = a_{1j}$$

Zwróćmy uwagę teraz na macierze wielodiagonalne. Elementy tych macierzy są zdefiniowane następująco:

$$a_{ij} = \begin{cases} 0 & \text{gdy} \quad |i-j| > m \\ a_{ij} & \text{gdy} \quad |i-j| \leqslant m \end{cases}$$

dla ij = 1, 2,..., n oraz m < n
a więc macierze te mają postać



(13)

Istotnym faktem jest to, iż przy rozkładzie macierzy wielodiagonalnej na iloczyn macierzy C i D, macierze te są również wielodiagonalne, tzn. elementy c_{ij} macierzy C oraz d_{ij} macierzy D są równe zeru, gdy $|i-j| > m$. Tę zależność można wykorzystać. Gdy macierz A jest symetryczna i wielodiagonalna, to możemy ją zapisać w postaci macierzy G o wymiarach $n \times (m+1)$, gdyż wystarczy pamiętać tylko jeden trójkąt.

Załóżmy, że będziemy pamiętać tylko trójkąt górny. Wtedy elementy macierzy A dla $i \leqslant j$ będą zapisane w macierzy G na miejscach

$$[i,\ j-i+1] \tag{14}$$

I tak dla np. $n = 8$, $m = 2$ elementy g_{rk} ($r = 1,2,\ldots, n$; $k = 1,2,\ldots, m$) macierzy G będą odpowiednio równe

$$G = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{22} & a_{23} & a_{24} \\ a_{33} & a_{34} & a_{35} \\ a_{44} & a_{45} & a_{46} \\ a_{55} & a_{56} & a_{57} \\ a_{66} & a_{67} & a_{68} \\ a_{77} & a_{78} & \square \\ a_{88} & \square & \square \end{vmatrix} \tag{15}$$

Miejsca w dolnym prawym rogu są niewykorzystane, lecz nie jest to wielka strata (szczególnie gdy $n \gg m$). Ponieważ macierz C jest trójkątna i wielodiagonalna, więc wg Cholesky'ego można ją otrzymać na tych samych miejscach macierzy G nie rezerwując dodatkowych. Wygodniej jest otrzymać (przy założeniu, że mamy zapisany górny trójkąt macierzy A) macierz C^T na tych samych miejscach, co nie zmienia praktycznie istoty rzeczy.

Wektory y oraz x określone wzorami 6 obliczamy na miejscach wyrazów wolnych (wektora b) wg zależności:

$$y_1 = \frac{1}{a_{11}}\, b_1$$

$$y_i = \frac{1}{a_{ii}} \left(b_1 - \sum_{k=1}^{i-1} a_{ki}\, y_k \right) \tag{16}$$

kolejno dla $i = 2, \ldots, n$

oraz

$$x_n = y_n$$
$$x_i = y_i - \frac{1}{a_{ii}} \sum_{k=i+1}^{n} a_{ik} y_k$$

kolejno dla i = n-1,..., 1

A więc wzory (11), (16) wyznaczają w całości metodę postępowania wg Cholesky'ego bez uwzględnienia zamiany miejsc określonych wg (14), co łatwo jest przekształcić przy realizacji konkretnego algorytmu.

2. PROPONOWANA METODA POSTĘPOWANIA

Zwróćmy uwagę na fakt, iż na podstawie indeksów określających elementy niezerowe, można dokonać przenumerowania zmiennych x_i tak, aby przez zamianę wierszy i kolumn otrzymać macierz A' o postaci wielodiagonalnej. (Artykuł ten nie zajmuje się przenumerowaniem zmiennych, gdyż jest to zagadnienie osobne). Rozpatrzmy to na przykładzie.

Załóżmy, że mamy w pamięci maszyny indeksy określające elementy niezerowe macierzy A, znajdujące się poza główną przekątną ($a_{ii} \neq 0$) np. przy n = 6.

i	j
2	6
1	3
4	5
1	6
5	6
2	4
3	5

Ponieważ macierz A jest symetryczna, tzn. $a_{ij} = a_{ji}$, więc określenie elementów niezerowych w podany sposób jest wystarczające.

Macierz A miałaby wtedy postać:

$$A = \begin{vmatrix} a_{11} & 0 & a_{13} & 0 & 0 & a_{16} \\ 0 & a_{22} & 0 & a_{24} & 0 & a_{26} \\ a_{13} & 0 & a_{33} & 0 & a_{35} & 0 \\ 0 & a_{24} & 0 & a_{44} & a_{45} & 0 \\ 0 & 0 & a_{35} & a_{45} & a_{55} & a_{56} \\ a_{16} & a_{26} & 0 & 0 & a_{56} & a_{66} \end{vmatrix} \qquad (17)$$

Chcąc pamiętać np. górny trójkąt macierzy A musielibyśmy zarezerwować n (n+1) /2 miejsc w tablicy jednowymiarowej. Takie postępowanie jest nieekonomiczne.

Dokonajmy np. następującej zamiany zmiennych

$$\begin{aligned} x'_1 &= x_1 \\ x'_2 &= x_5 \\ x'_3 &= x_2 \\ x'_4 &= x_6 \\ x'_5 &= x_4 \\ x'_6 &= x_3 \end{aligned} \qquad (18)$$

którą można zanotować na n miejscach w postaci wektora c o elementach całkowitych

$$c = \begin{bmatrix} 1 \\ 5 \\ 2 \\ 6 \\ 4 \\ 3 \end{bmatrix} \qquad (19)$$

Takie przenumerowanie łączy się z inwersjami kolumn oraz wierszy macierzy A wraz z wyrazami wolnymi b.

Naszą macierz A' oraz wektor b' możemy wówczas zapisać w postaci:

$$A' = \begin{vmatrix} a_{11} & a_{13} & a_{16} & 0 & 0 & 0 \\ a_{13} & a_{33} & 0 & a_{35} & 0 & 0 \\ a_{16} & 0 & a_{66} & a_{56} & a_{26} & 0 \\ 0 & a_{35} & a_{56} & a_{55} & 0 & a_{45} \\ 0 & 0 & a_{26} & 0 & a_{22} & a_{24} \\ 0 & 0 & 0 & a_{45} & a_{24} & a_{44} \end{vmatrix} \qquad b' = \begin{vmatrix} b_1 \\ b_3 \\ b_6 \\ b_5 \\ b_2 \end{vmatrix} \qquad (20)$$

gdyż elementy a_{ks}, a_{sk} macierzy A będą zapisane odpowiednio na miejscach $[c_k, c_s]$, $[c_s, c_k]$ macierzy A', natomiast elementy b_k na miejscach c_k, gdzie k, s = 1, 2, ..., n.

Jak widzimy macierz A' jest macierzą wielodiagonalną i możemy układ $A'x' = b'$ rozwiązać metodą Cholesky'ego korzystając z przedstawienia macierzy A' wg (15).

W naszym przypadku $m = \max |i' - j'| = 2$ więc możemy macierz A' zapisać w postaci innej macierzy o wymiarach 6 x 3, gdyż wystarczy pamiętać tylko elementy "górnej taśmy" macierzy A', a następnie przekształcamy ją na tych samych miejscach na macierz C^T.

Na tym małym przykładzie nie widać dużych oszczędności jakie się czyni. Dla uwidocznienia tego oprzemy się na przykładzie zagadnień sieciowych, które spotyka się często w praktyce.

Metody rozwiązujące dane zagadnienia bardzo często wykorzystują rozwiązanie układów równań liniowych, których macierze mają elementy niezerowe na głównej przekątnej oraz gdy węzeł i ma połączenie z węzłem j.

W rzeczywistości spotykamy sieci zawierające przeważnie powyżej 150 węzłów, gdzie każdy ma liczbę połączeń z innymi węzłami nie większą od 5.

A więc np. przy n = 150 musielibyśmy korzystać z macierzy n x n = 22500 elementów. W przypadku gdy macierz jest symetryczna korzystalibyśmy z n (n+1) /2 = 11325 elementów (zapisujemy tylko jeden trójkąt macierzy A).

Jeżeli liczba połączeń każdego węzła z innymi nie przekracza 5, to po przenumerowaniu węzłów możemy przeważnie otrzymać macierz wielodiagonalną A', gdzie m jest równe około 11. W takim przypadku możemy na zapis macierzy A' zająć tylko n x (m+1) = 150 x 12 = 1800 miejsc w pamięci maszyny.

Z porównania tych liczb wynikają wyraźnie zalety metody Cholesky'ego ze wstępnym przenumerowaniem, nie mówiąc już o różnicy w liczbie wykonywania działań przy rozwiązywaniu układów równań liniowych w zestawieniu z metodami, które nie uwzględniają miejsc zerowych macierzy.

Algorytm ten można połączyć z wyborem maksymalnego elementu i można to uwzględnić podczas przenumerowania zmiennych. Dla macierzy symetrycznych ten element oczywiście musi znajdować się na głównej przekątnej, gdyż w przeciwnym razie popsuje się symetrię. Jest to oczywiście wadą.

Gdy macierz jest źle uwarunkowana, to niestety zagadnienie jest w dalszym ciągu otwarte. W praktyce nie zawsze jednak jest tak źle. Np. w zagadnieniach sieciowych bardzo często macierze są symetryczne i dobrze uwarunkowane, gdyż jest spełniony warunek

$$a_{ii} \geqslant \sum_{\substack{j=1 \\ j\neq i}}^{n} |a_{ij}| = \sum_{\substack{j=1 \\ j\neq i}}^{n} |a_{ji}|$$

co jest bardzo istotne przy algorytmie Cholesky'ego.

Algorytm rozwiązywania układów równań liniowych, gdźie macierz A jest symetryczna i wielodiagonalna, zapisany w języku ODRA-ALGOL, jako procedura znajduje się w Centralnym Laboratorium Gazownictwa w Warszawie.

Literatura

[1] DEMIDOWICZ B.P., MARON I.A.: Metody numeryczne, PWN, 1965.

ПРИМЕНЕНИЕ СХЕМЫ ХОЛЕСКОГО К РЕШЕНИЮ СИСТЕМ ЛИНЕЙНЫХ УРАВНЕНИЙ

Резюме

В статьи указан метод решения систем линейных уравнений в виде $Ax = b$ для матрицы A содержащей много нулевых элементов. Метод основан на алгоритме Холеского, который в таких случаях можно модифицировать, и на двух следующих замечаниях:

1. Использование свойств разложения матрицы A на произведение матриц CD, где C - матрица нижнетреугольная, D - матрица верхнетреугольная с единичными элементами на главной диагонали. Итак - если матрица A многодиагональная - для произвольного $m > n$ (n - степень системы уравнений) исполнена зависимость $a_{ij} = 0$ для $|i-j| > m$, то матрицы C и D тоже многодиагональные и $c_{ij}=d_{ij}=0$ для $|i-j| > m$.

2. Системы уравнений $Ax = b$ можно привести к виду $A'x' = b'$ (при том A' - многодиагональная матрица) прямым изменением переменных типа $x_i = x_{r_i}$, где r_i это вообще беспорядочное множество индексов составных вектора x. Этот метод дает большую экономию памяти математической машины и заметно сокращает время исчисления.

APPLICATION OF CHOLESKY'S SCHEME FOR SOLVING LINEAR EQUATIONS SYSTEMS

Summary

In the article the method of solving linear equations systems in the form of $Ax = b$, for the matrix A containing many zero elements is described. The method is based on the Cholesky's algorithm, which can be modified in such cases, and on two following observations:

1. Making use of the property of A matrix factorization to matrix product CD where C is downtriangle matrix and D-toptriangle one with main diagonal elements that are equal 1. And so, if the matrix A is multidiagonal matrix, i.e. for any $m<n$ (n - degree of equation system) the condition $a_{ij} = 0$ is satisfied for $|i-j|>m$, then matrices C and D are also multidiagonal and $c_{ij} = d_{ij} = 0$ for $|i-j| > m$.

2. Equation system $Ax = b$ can be reduced to the form $A'x' = b'$ (where A' is multidiagonal matrix) by means of simple transformation of variables of the type $x'_i = x_{r_i}$ where r_i is, in general, a disordered set of indices of vector x elements. The method saves a lot of computer memory as well as shortens time of calculation.

APPLICATION OF CHOLESKY'S METHOD FOR SOLVING LINEAR EQUATIONS ENCOUNTERED IN THE THEORY OF ELASTICITY

Summary

In the article the method of solving linear equations in the form of $Ax = b$, for the matrix A containing many zero elements is described. The method is based on the Cholesky's algorithm, which can be modified in such cases, and on two following observations:

1. Adding one of the property of a matrix transposition to matrix product (in the triangle matrix and Cholesky's one with only diagonal elements that are equal 1), and so: if the matrix A is multiplied-matrix, i.e. for any $a_{ij} = 0$ for $(i - j) > a$, and the matrix B is multiplied-matrix, i.e. for any $b_{ij} = 0$ for $(i - j) > b$, then the matrix $C = AB$ is also multiplied and $c_{ij} = 0$ for $(i - j) > a + b$.

2. Equation system $Ax = b$ can be reduced to the form $A'x = b'$ (where A' is multiplied matrix) by means of simple transformation of variables of the type $ci = x_i$, where ri, in general, a elementary set of indices of vector x elements. The method saves a lot of computer memory as well as shortens time of calculation.

Author's address:
Instytut Budowy Maszyn Rolniczych
Politechnika Warszawska
Warszawa, ul. Nowowiejska 24.

ALGORYTMY, XI, N° 20, 1974, 85-134 681.322.06

METODA WYKORZYSTANIA PROGRAMU-GENERATORA DOWOLNEGO JĘZYKA ZAPISANEGO W NOTACJI BACKUSA JAKO AKCEPTORA TEGO JĘZYKA

Grażyna KUŚMIERZ
Ryszard JANDA

Studium Doktoranckie
Politechniki Warszawskiej

Pracę złożono 10.05.1973

Przedstawiono algorytm, który zawsze umożliwia rozstrzygnięcie, czy wybrany ciąg symboli jest poprawnym zdaniem określonego uprzednio języka bezkontekstowego, czy nie. Język powinien być opisany w notacji Backusa. Notacja ta jest punktem wyjściowym do konstrukcji tzw. programu-generatora, wykorzystywanego później w procesie rozstrzygania.

S p i s t r e ś c i

WSTĘP

Artykuł niniejszy prezentuje metodę wykorzystania programu-generatora dowolnego języka zapisanego w notacji Backusa jako akceptora tego języka.

Program-generator dowolnego języka ma za zadanie wytwarzanie gramatycznie poprawnych zdań tego języka. Może on wygenerować każde poprawne zdanie i żadne inne. O ile w języku występuje więcej niż jedno zdanie poprawne, to w programie-generatorze muszą istnieć przejścia niezdeterminowane.

Program-generator może być także wykorzystany do analizy arbitralnie obranych ciągów symboli w celu stwierdzenia, czy ciągi te są wyrażeniami poprawnymi danego języka i - w przypadku odpowiedzi pozytywnej - do podania rozkładu gramatycznego zdania. Spełnia on w tym przypadku wymienioną już wyżej funkcję tzw. akceptora lub analizatora języka.

Autorzy ograniczają swoje rozważania do języków zdefiniowanych za pomocą notacji Backusa. Notacja ta jest gramatyką języka i musi spełniać wszystkie warunki gramatyki poprawnie zdefiniowanej.
Musi więc podawać:

a/ alfabet końcowy (terminalny) języka,
b/ alfabet pomocniczy gramatyki,
c/ zbiór produkcji gramatyki,
d/ aksjomat (symbol początkowy) gramatyki.

Podstawowym członem zapisu backusowskiego jest tzw. instrukcja podstawienia, np.:

<instrukcja> ::= beg <instrukcja> end | <go-to> | <assn>

określająca pewien niepusty podzbiór zbioru produkcji, do którego należą wszystkie produkcje o tym samym poprzedniku. Wyrażenia ujęte w nawiasy trójkątne nazywają się zmiennymi

języka. Czasami nazywać je będziemy po prostu zmiennymi. Zbiór wszystkich zmiennych tworzy alfabet pomocniczy.

Autorzy zakładają, że w opisie języka istnieją zawsze:

- zmienna <program>, która jest aksjomatem. Określa ona najogólniej całe wyrażenie poprawne języka.
- zmienna <symbol> i instrukcja podstawienia:

<symbol> ::= .....

po której prawej stronie występują wszystkie symbole danego języka, tworzące alfabet końcowy, oddzielone znakami " | " i żadne inne. Pozwala to uniknąć niejasności typu: czy ":=" to dwa różne symbole ":" i "=" napisane obok siebie, czy też jeden nierozdzielny symbol.

Do symboli języka zaliczać będziemy symbol pusty i oznaczać go przez ε.

Załóżmy teraz, że mamy dany pewien ciąg znaków i chcemy zbadać, czy jest on poprawnym zdaniem danego języka. Warunkiem dokonania analizy jest, w myśl założeń tego artykułu, dysponowanie specjalnym programem-generatorem tego języka.

Sama notacja Backusa, aczkolwiek przejrzysta i prosta, nie nadaje się do bezpośredniego wykorzystania w praktyce "maszynowej", chociażby z następujących powodów:

a/ nie wszystkie symbole danego języka są jednolitymi symbolami określonej maszyny cyfrowej,

b/ zdania metajęzykowe nie mają postaci ciągu adresowanych instrukcji, co utrudnia określenie miejsca poddawanego aktualnie analizie.

Trudności tych można łatwo uniknąć używając, zamiast symboli, zmiennych sznurowych reprezentujących je oraz

tworząc na podstawie zapisu backusowskiego odpowiedni pro - gram-generator, będący sekwencją pewnej liczby specjalnych instrukcji, które można etykietować, numerować, itp. Instrukcje te będą podane i opisane niżej.

Przedstawiony tutaj algorytm polega na jednoczesnym przeglądaniu analizowanego ciągu symboli i programu-generatora. C ile w procesie generacji niektóre przejścia od jednej instrukcji do drugiej mogą zachodzić w sposób niejednoznaczny, co powoduje, że sterowanie przebiega jedną z wielu możliwych dróg, o tyle w procesie analizy sterowanie to przebiega wszystkie możliwe drogi w pewien usystematyzowany sposób. Gdy na jakiejś drodze nie ma możliwości wygenerowania aktualnie analizowanego symbolu, lub gdy droga ta jest "za długa" wtedy eliminuje się ją. Akceptacja badanego ciągu symboli jako poprawnego zdania języka zachodzi w chwili napotkania na jakiejś drodze instrukcji "stop" z jednoczesnym osiągnięciem końca tego ciągu. Brak akceptacji następuje w chwili wyeliminowania wszystkich dróg. Algorytm jest skończony, gdyż zawsze zachodzi jedna z tych dwu sytuacji.

1. PROGRAM-GENERATOR

Weźmy teraz pod uwagę następującą instrukcję podstawienia:

$$\langle body\rangle \quad ::= \quad \langle contents\rangle \;|\; \langle body\rangle \quad A \qquad (1)$$

określa ona dwuelementowy podzbiór zbioru produkcji, którego elementy można przedstawić następująco, używając w tym celu zamiast ":::=" innego symbolu podstawienia "⟶":

$$\begin{aligned} \langle body\rangle &\longrightarrow \langle contents\rangle \\ \langle body\rangle &\longrightarrow \langle body\rangle \quad A \end{aligned} \qquad (2)$$

Produkcje te charakteryzują się wspólnym poprzednikiem <body> oraz dwoma różnymi następnikami: <contents> i <body> A. Dla uproszczenia mówić będziemy, że instrukcja podstawienia ze wzoru (1) posiada dwa następniki.

Każdy następnik nazywać będziemy gałęzią. Każdy poprzednik instrukcji podstawienia posiadającej więcej niż jeden następnik nazywać będziemy węzłem. Każdy poprzednik jest oczywiście zmienną języka.

Mówimy, że gałąź wychodzi z węzła, gdy jest ona następnikiem tego węzła.

Przejdźmy teraz do opisu programu-generatora języka. Jest on tworzony na podstawie zapisu tego języka w notacji Backusa. Podstawowym elementem programu-generatora jest instrukcja. Każdej zmiennej języka, która występuje w jakiejś gałęzi lub jest zmienną <program> odpowiada w programie-generatorze procedura o tej samej nazwie, co nazwa zmiennej lecz pisanej dużymi literami i bez nawiasów.

Oprócz procedur program-generator zawiera ponadto tzw. gałąź główną, którą zdefiniujemy dalej. Przyjmujemy, że gałąź główna wychodzi z tzw. węzła zerowego.

Niektóre instrukcje programu-generatora oznaczane są etykietami.
Rozróżniamy przy tym dwa rodzaje tych etykiet:

a/ etykiety ciał procedur, stojące przed pierwszą instrukcją każdego ciała procedury. Nazwy tych etykiet są identyczne z nazwami procedur, które etykietują.

b/ etykiety gałęzi, stojące przed pierwszą instrukcją każdej gałęzi programu-generatora, wychodzącej z węzła programu-generatora. Gałęzią i węzłem programu-generatora nazywamy przy tym ciągi instrukcji odpowiadające odpo -

wiednio gałęzi i węzłowi opisu metajęzykowego. Ponadto do gałęzi i węzłów programu-generatora zaliczamy gałąź główną i węzeł zerowy. Etykiety gałęzi to: etykieta START stojąca na początku gałęzi głównej oraz etykiety A_i ($_i$ = 1,2,3,...) , stojące na początku innych gałęzi, wychodzących z węzłów.

Program-generator dowolnego języka zapisanego w notacji Backusa utworzyć można za pomocą sześciu następujących instrukcji:

1. call (PROC) , gdzie PROC oznacza dowolną nazwę procedury. Instrukcja ta powoduje wejście w ciało procedury PROC, wykonanie jej i powrót do instrukcji występującej bezpośrednio za call (PROC) .

2. jp (A_i) , ($_i$= 1,2,3,...), której parametrem jest dowolna etykieta gałęzi. Instrukcja ta powoduje skok do instrukcji umieszczonej pod etykietą A_i.

3. rnd (n). Instrukcja ta stoi zawsze na początku ciała procedury odpowiadającej zmiennej języka, będącej węzłem. Parametr n równy jest liczbie gałęzi wychodzących z tego węzła (tzw. stopniowi węzła). Instrukcja rnd (n) w czasie generacji programu wybiera niejednoznacznie liczbę naturalną K z przedziału [1, n] i powoduje pominięcie K-1 następnych instrukcji. Instrukcje rnd (n) i jp (A_i) mogą występować jedynie w konfiguracji jak w przykładzie 1.

Przykład 1.

```
        rnd  (n)
       ⎧ jp  (A_j)
       ⎪ jp  (A_j+1)
n razy ⎨  ⋮
       ⎪
       ⎩ jp  (A_j+n-1)
```

Ciąg tych instrukcji jest, jak widać, węzłem programu-generatora.

4. gen (H) , gdzie H oznacza dowolny symbol końcowy języka. W wyniku wykonania instrukcji gen (H) zostaje wygenerowany symbol H.

5. return. Instrukcja ta umieszczona jest na końcu każdej gałęzi z wyjątkiem gałęzi głównej. Powoduje ona powrót z ciała procedury, do którego należy dana gałąź, do instrukcji następującej bezpośrednio za operacją call (.) , która wywołała daną procedurę.

6. stop. Instrukcja ta umieszczona jest na końcu gałęzi głównej i kończy generację zdania.

Gałęzią główną każdego programu-generatora jest następująca sekwencja instrukcji:

START: call (PROGRAM)
stop.

Generacja każdego zdania rozpoczyna się od wykonania instrukcji call (PROGRAM) umieszczonej pod etykietą START. Instrukcja ta powoduje wejście w ciało nieopcjonalnej procedury PROGRAM (odpowiadającej zmiennej <program>), wykonanie tej procedury i powrót do instrukcji stop.

Każde ciało procedury składa się albo z jednej gałęzi programu-generatora, o ile procedura ta odpowiada zmiennej języka nie będącej węzłem, albo z węzła programu-generatora (patrz przykład 1) i wychodzących z niego gałęzi programu-generatora. Procedury mogą być wywoływane rekurencyjnie.

Poniżej podano przykłady opisu pewnego języka w notacji Backusa i odpowiadającego temu opisowi programu-genera-

tora.

Przykład 2

Opis języka:

<program> ::= beg <body> end

<body> ::= <contents> | <body> A

<contents> ::= B | <body>

<symbol> ::= A | B | beg | end | ε

Przykład 3

Program-generator:

```
START     : call (PROGRAM)
            stop.

PROGRAM   : gen  (beg)
            call (BODY)
            gen  (end)
            return.

BODY      : rnd  (2)
            jp   (A1)
            jp   (A2)

A1        : call (CONTENTS)
            return.

A2        : call (BODY)
            gen  (A)
            return.

CONTENTS  : rnd  (2)
            jp   (A3)
            jp   (A4)
```

```
A3    : gen (B)
        return.

A4    : call (BODY)
        return.
```

Zauważmy, że zmienna <symbol> nie występuje w opisie języka w żadnej gałęzi (tzn. jest nieosiągalnym symbolem alfabetu pomocniczego) . Oznacza to, że procedura SYMBOL odpowiadająca tej zmiennej, gdyby istniała, nie byłaby nigdy wywoływana. Dlatego też w powyższym programie-generatorze pominięto ją.

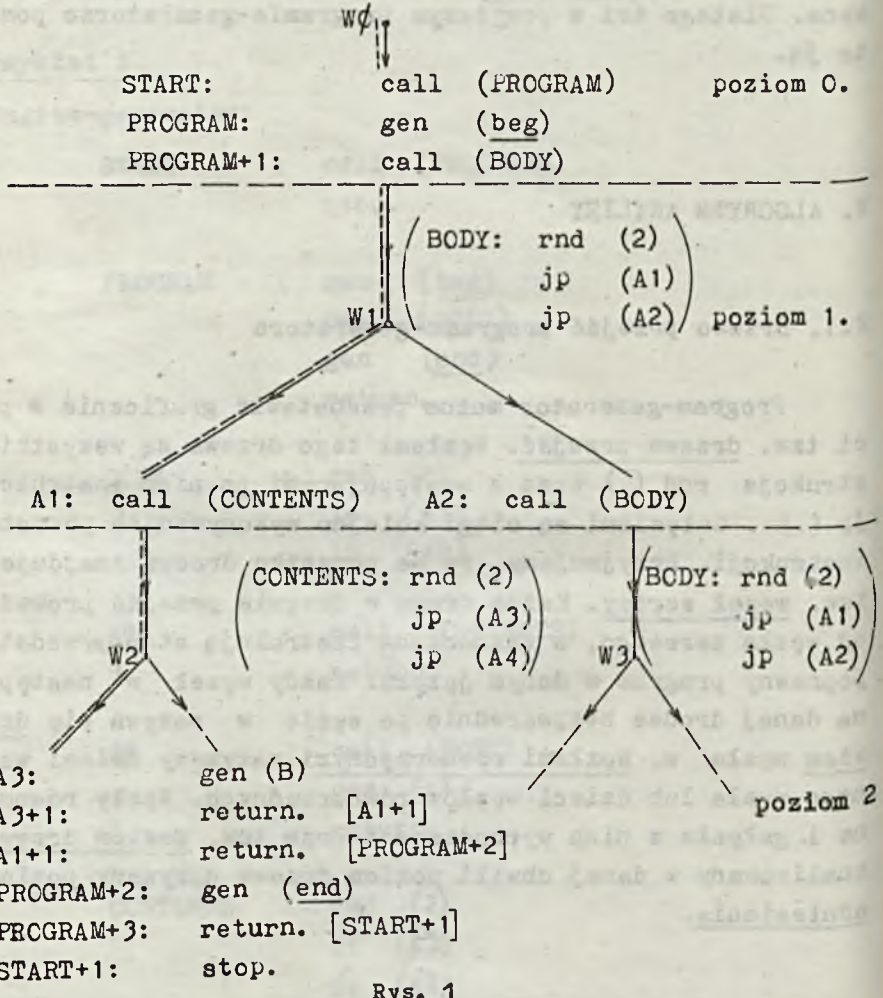
2. ALGORYTM ANALIZY

2.1. Drzewo przejść programu-generatora

Program-generator można przedstawić graficznie w postaci tzw. drzewa przejść. Węzłami tego drzewa są wszystkie instrukcje rnd (.) wraz z następującymi po nich instrukcjami jp (.) . Gałęziami są ciągi kolejno wykonywanych pozostałych instrukcji. Przyjmujemy, że na początku drzewa znajduje się tzw. węzeł zerowy. Każda droga w drzewie przejść prowadząca od węzła zerowego, a zakończona instrukcją stop przedstawia poprawny program w danym języku. Każdy węzeł w' następujący na danej drodze bezpośrednio po węźle w nazywa się dzieckiem węzła w. Węzłami równorzędnymi nazywamy dzieci wspólnego węzła lub dzieci węzłów równorzędnych. Węzły równorzędne i gałęzie z nich wychodzące tworzą tzw. poziom drzewa. Analizowany w danej chwili poziom drzewa nazywamy poziomem odniesienia.

Dzieci węzłów równorzędnych umieszczonych na poziomie odniesienia nazywamy węzłami podrzędnymi. Tworzą one wraz z wychodzącymi z nich gałęziami poziom drzewa o jeden wyższy niż poziom odniesienia. Poziom ten nazywamy przyszłym poziomem odniesienia. Za poziom zerowy przyjmujemy węzeł zerowy i gałąź z niego wychodzącą.

Drzewo przejść ma zwykle postać nieskończoną. Jego "początek" dla programu-generatora z przykładu 3 przedstawiono na rys.1.



Rys. 1

Obok każdej instrukcji return podano w nawiasach kwadratowych adres powrotu. Węzły W2, W3 są węzłami równorzędnymi i podrzędnymi względem węzła W1. Jak łatwo zauważyć, przejściu po drodze oznaczonej dodatkowo linią przerywaną towarzyszy wygenerowanie zdania:

beg B end

które jest najprostszym zdaniem w podanym języku.

Głębokością wywołania procedury nazywamy liczbę otwartych (przez instrukcję call (.)), a jeszcze nie zamkniętych (przez instrukcję return) ciał procedur, liczoną w momencie otwarcia danej procedury na wybranej drodze w drzewie przejść. Takie "niedokończone" procedury tworzą tzw. łańcuch procedur. Otwarcie każdej następnej procedury łańcucha zachodzi w ciele procedury poprzedniej.

2.2. Idea algorytmu analizy

Algorytm polega na poszukiwaniu w drzewie przejść programu-generatora takiej drogi, na której następuje generacja zdania identycznego z badanym ciągiem symboli.

Analizowane są wszystkie drogi w drzewie przejść odcinek po odcinku, przy czym za odcinek przyjmujemy jedną gałąź tego drzewa.

Po przeanalizowaniu odcinka jednej drogi, analizowany jest kolejny odcinek drogi następnej itd. Poniżej przedstawiony zostanie szkicowo jedynie mechanizm "przełączania" sterowa-

nia z jednej drogi na drugą oraz mechanizm kończenia algorytmu. Dokładny opis wszystkich czynności analizatora podany będzie w podrozdziale 2.4.

Analizuje się kolejno wszystkie węzły i gałęzie umieszczone na poziomie odniesienia, począwszy od poziomu 0. Aktualnie analizowany węzeł i aktualnie analizowaną gałąź nazywamy odpowiednio węzłem odniesienia i gałęzią zaktywizowaną.

Proces uznawania węzła lub gałęzi za węzeł odniesienia lub gałąź zaktywizowaną nazywać będziemy odpowiednio aktywizacją węzła lub gałęzi.

Analiza każdego węzła odniesienia polega na:

1. kolejnym aktywizowaniu i analizowaniu wszystkich gałęzi wychodzących z tego węzła,

2. aktywizacji, w przypadku przeanalizowania już wszystkich gałęzi danego węzła, kolejnego węzła równorzędnego umieszczonego na poziomie odniesienia lub - w przypadku gdy wszystkie węzły równorzędne zostały wyczerpane - zwiększeniu poziomu odniesienia o 1 i aktywizacji pierwszego węzła, umieszczonego na nowym poziomie odniesienia.

Analiza każdej gałęzi zaktywizowanej polega na:

3. odtworzeniu stanu analizatora z chwili, gdy węzeł odniesienia, z którego wychodzi dana gałąź, został napotkany w czasie analizy gałęzi doprowadzającej sterowanie do tego węzła, umieszczonej na poziomie drzewa o 1 niższym,

4. kolejnym wykonywaniu instrukcji programu-generatora, należących do danej gałęzi, przy czym:

a/ gdy napotkana zostaje instrukcja stop, to w przypadku przeanalizowania już całego badanego ciągu symboli, następuje akceptacja tego ciągu z podaniem jego rozkładu gramatycznego i zakończenie algorytmu; w przypadku przeciwnym - badana gałąź zostaje wyeliminowana i następuje aktywizacja następnej gałęzi wychodzącej z węzła odniesienia,

b/ gdy napotkana zostaje instrukcja rnd (.) , co oznacza, że napotkany został węzeł podrzędny, na którym dana gałąź się kończy, następuje zapamiętanie stanu analizatora z tego momentu i aktywizacja następnej gałęzi, wychodzącej z węzła odniesienia,

c/ gdy napotkana zostaje instrukcja call (.) , która powoduje wywołanie procedury o zbyt dużej głębokości (patrz twierdzenie o maksymalnej głębokości wywołania procedury), badana gałąź zostaje wyeliminowana i następuje aktywizacja następnej gałęzi, wychodzącej z węzła odniesienia,

d/ gdy napotkana zostaje instrukcja gen (.) , której parametr nie jest identyczny z aktualnie rozpatrywanym symbolem badanego ciągu symboli, to badana gałąź zostaje wyeliminowana i następuje aktywizacja następnej gałęzi, wychodzącej z węzła odniesienia.

We wszystkich ww przypadkach eliminacja gałęzi jest równoznaczna z eliminacją drogi w drzewie przejść, na której dana gałąź się znajduje.

Algorytm zawsze kończy się akceptacją lub brakiem akceptacji badanego ciągu symboli, przy czym:

5. BRAK AKCEPTACJI następuje wtedy, gdy wszystkie gałęzie umieszczone na poziomie odniesienia zostały wyeliminowane,

6. AKCEPTACJA następuje w momencie napotkania w jakiejś gałęzi instrukcji stop z jednoczesnym osiągnięciem końca badanego ciągu symboli.

2.3. Twierdzenie o maksymalnej głębokości wywołania procedury

Języki zdefiniowane w notacji Backusa posiadają zwykle opisy rekurencyjne, przy czym możemy mieć do czynienia z rekursjami lewostronnymi, prawostronnymi, wewnętrznymi i złożonymi.

W poniższym przykładzie zmienna <string> definiowana jest w każdej kolejnej gałęzi, począwszy od gałęzi drugiej, odpowiednio przez rekursje ww typów:

<string> ::= A | <string> B | C <string> | D <string> E | <chain> F

<chain> ::= <string>

W trakcie analizy największą komplikację wprowadzają rekursje lewostronne i złożone. Wykażemy to na poniższym przykładzie.

Chcemy zbadać, czy ciąg symboli:

BA

daje się wywieść ze zmiennej <string> . Analizujemy w tym celu wszystkie gałęzie wychodzące z węzła <string> w kolejności od lewej do prawej. Gałęzie pierwsza, trzecia i czwarta są "zablokowane" odpowiednio przez symbole A,C,D więc nie możemy w nie wejść. "Otwarte" są tylko gałęzie druga i piąta. Wchodzimy w nie i w obu przypadkach napotykamy znowu węzeł <string>, w którym jedynie gałęzie druga i piąta są otwarte.

Proces powyższy powtarza się w nieskończoność. Oznacza to, że postępując jak wyżej nie mamy możliwości sprawdzenia w skończonej liczbie kroków, czy badany ciąg daje się wywieść ze zmiennej <string> , czy nie.

Aby taką możliwość zapewnić autorzy wprowadzają pojęcie tzw. maksymalnej głębokości wywołania procedury, które służy do oszacowania maksymalnej liczby kroków algorytmu analizy.

Twierdzenie:

Załóżmy, że opis rozpatrywanego przez nas języka zawiera g zmiennych.

Gdy przy analizowaniu poprawnego zdania, złożonego z k symboli ($k \geqslant 0$) zajdzie sytuacja, że na pewnej drodze w drzewie przejść głębokość wywołania procedury przewyższy liczbę MAX równą

$$MAX = (k + 1) \cdot g$$

zwaną maksymalną głębokością wywołania procedury, to drogę tę możemy wyeliminować, gdyż akceptacja zdania nastąpi na pewno na innej drodze.

Dowód tego twierdzenia wymaga wprowadzenia wielu pojęć pomocniczych nie używanych w dalszym ciągu tego artykułu. Aby nie tracić ciągłości rozumowania celowe więc było podanie go dopiero w ostatnim rozdziale.

2.4. Opis algorytmu analizy

Algorytm używa dwóch stosów. Są to:

1. stos SYMBOL. Na każdej numerowanej pozycji tego stosu zapisany zostaje przed rozpoczęciem analizy jeden symbol badanego ciągu symboli. Ostatnia pozycja stosu zawiera zawsze zastrzeżony symbol finish, służący do identyfikacji końca tego ciągu. Odwołanie do m-tej pozycji stosu SYMBOL zachodzi za pomocą napisu SYMBOL (m) .

2. stos STACK. Jest to stos roboczy. Na stosie tym zapisywane są tzw. moduły (patrz niżej) , złożone z numerowanych pozycji, z których każda może mieć kilka parametrów, identyfikowanych dużymi literami alfabetu i oddzielonych przecinkami. Odwołanie do n-tej pozycji stosu STACK zachodzi za pomocą napisu STACK(n), zaś odwołanie do parametru X tej pozycji - za pomocą napisu X of STACK(n). Pozycje stosu STACK mogą być czterech rodzajów. Pierwszy parametr każdej pozycji identyfikowany jest literą Y i może posiadać 4 następujące wartości:

$$x, g, <, >$$

które służą do identyfikacji rodzaju pozycji. Rozróżniamy następujące rodzaje pozycji stosu STACK:

a/ węzeł stosu Y,A,B,C,D,E,F,G,H. przy czym Y = x.

b/ generacja Y,I. przy czym Y = g.

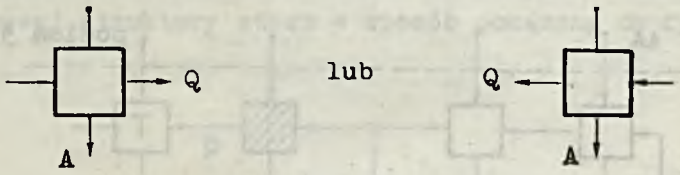
c/ otwarcie procedury Y,I,K,L. przy czym Y = <.

d/ zamknięcie procedury Y. przy czym Y = >.

Stos STACK jest obrazem drzewa przejść. Obraz każdej gałęzi tego drzewa i kończącego ją węzła*) nazywamy modułem.

*) gałąź i węzeł ją kończący występują na dwóch różnych poziomach drzewa

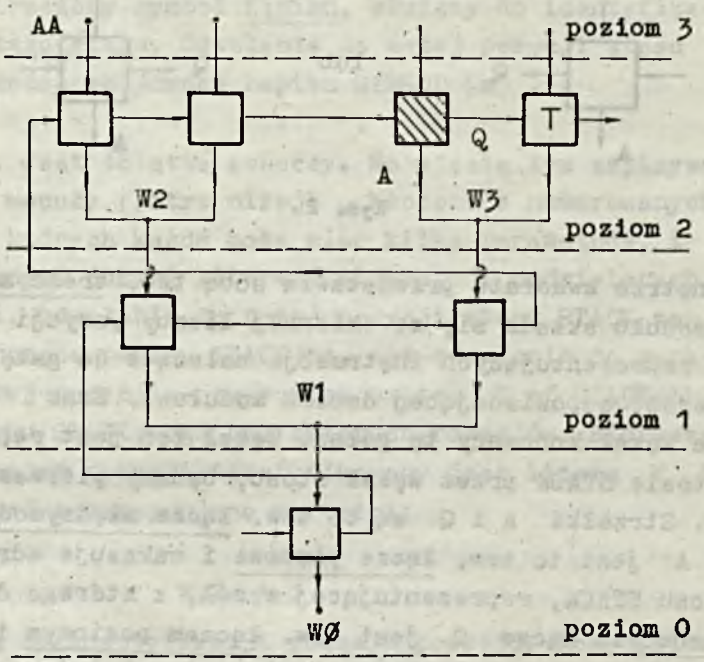
Moduł można przedstawić graficznie w sposób podany na rys.2.



Rys. 2

Wnętrze kwadratu przedstawia sobą tzw. treść modułu. Treść modułu składa się ze zmiennej liczby pozycji stosu STACK, reprezentujących instrukcje należące do gałęzi drzewa przejść, odpowiadającej danemu modułowi. Znak "." przedstawia węzeł kończący tę gałąź. Węzeł ten jest reprezentowany w stosie STACK przez węzeł stosu, będący pierwszą pozycją modułu. Strzałki A i Q są to tzw. łącza międzymodułowe. Łącze A jest to tzw. łącze pionowe i wskazuje adres pozycji stosu STACK, reprezentującej węzeł, z którego dana gałąź wychodzi. Łącze Q jest tzw. łączem poziomym i wskazuje początek następnego modułu w stosie, odpowiadającego analizowanej gałęzi.

Połączenie poszczególnych modułów dla drzew z rys. 1. jest przedstawione na rys.3.

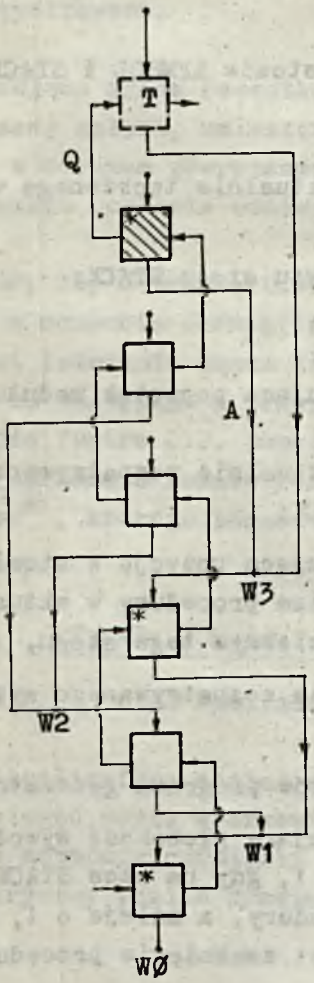


Rys. 3

Załóżmy, że w danej chwili analizowana jest gałąź, odpowiadająca zakreskowanemu modułowi. Łącze A tego modułu wskazuje wtedy węzeł odniesienia W3*), zaś łącze Q - początek nie znajdującego się jeszcze w stosie modułu T.

*) zwraca się uwagę czytelnika na fakt, że węzeł odniesienia nie znajduje się w module odpowiadającym aktualnie analizowanej gałęzi lecz w module wskazywanym przez łącze A, chociaż i węzeł i gałąź leżą na tym samym poziomie odniesienia

Wszystkie moduły połączone są szeregowo łączami poziomymi. Węzły tworzone są za pomocą łącz pionowych. Dzięki temu, rozgałęzioną strukturę drzewa przedstawić można za pomocą szeregowej struktury stosu w sposób pokazany na rysunku 4.



Rys. 4

Moduły połączone łączami pionowymi tworzą tzw. łańcuch dynamiczny stosu. W stosie występować może wiele łańcuchów dynamicznych. Na rysunku 4 moduły oznaczone gwiazdkami tworzą jeden łańcuch dynamiczny. Kolejność modułów w łańcuchu przyjmowana jest od dołu do góry stosu. Łańcuch dynamiczny stosu jest obrazem jednej drogi w drzewie przejść programu-generatora.

Obok omawianych stosów SYMBOL i STACK używane są następujące zmienne stanu:

a/ odnoszące się do aktualnie tworzonego w stosie STACK modułu:

SP - wskaźnik szczytu stosu STACK:
A - łącze pionowe,
Q - łącze poziome,
K - zmienna wskazująca początek modułu.

b/ odnoszące się do aktualnie rozpatrywanego łańcucha dynamicznego stosu STACK:

PP - zmienna wskazująca pozycję w stosie STACK, będącą ostatnim otwarciem procedury w aktualnie rozpatrywanym łańcuchu dynamicznym tego stosu,

AP - adres aktualnie rozpatrywanego symbolu w stosie SYMBOL,

IC - licznik rozkazów programu generatora,

GP - zmienna wskazująca głębokość wywołania procedury. Wzrasta ona o 1, gdy na stos STACK zapisywane jest otwarcie procedury, a maleje o 1, gdy na stos STACK zapisywane jest zamknięcie procedury.

c/ związane z "przełączaniem" sterowania (tzw. zmienne sterujące):

GG - zmienna wskazująca liczbę gałęzi wychodzących z węzła odniesienia lub z węzła aktualnie napotkanego w trakcie analizy,

G - zmienna, wskazująca numer zaktywizowanej gałęzi, wychodzącej z danego węzła. Gdy G = Ø, to żadna gałąź nie jest zaktywizowana,

AA - zmienna wskazująca adres początku modułu, odpowiadającego pierwszej gałęzi, umieszczonej na poziomie odniesienia, a zarazem pierwszemu węzłowi umieszczonemu na przyszłym poziomie odniesienia (patrz rys.3).

Zmienne PP, AP, IC, GP, GG, G tworzą tzw. stan analizatora i muszą być odtwarzane w momencie cofnięcia się do węzła (które jest możliwe dzięki istnieniu łącza A). Dlatego przy końcu tworzenia modułu, co następuje w przypadku napotkania węzła w drzewie przejść (patrz 2.2. punkt 4b) w pierwszej pozycji tego modułu (uprzednio zawsze rezerwowanej) zapisany zostaje węzeł stosu*), którego parametry mają następujące wartości:

x, A, Q, PP, AP, IC, GP, GG, G

łącza stan analizatora

Dzięki temu, że stan analizatora został zapisany w stosie w chwili napotkania jakiegoś węzła w drzewie przejść, możliwe jest odtworzenie go w momencie cofnięcia się do tego węzła, gdy już będzie rozpatrywany poziom drzewa o 1 wyższy.

*) wynika stąd, że pierwsza pozycja modułu odpowiada nie początkowi, lecz końcowi gałęzi, reprezentowanej przez ten moduł. Należy mieć to na uwadze przy studiowaniu dalszego ciągu artykułu

Przedstawiony poniżej akceptor języka składa się z: podprogramu INITIATION, inicjującego analizę, podprogramu sterującego CONTROL, którego zadaniem jest, opisane szkicowo w podrozdziale 2.2., przełączanie sterowania z jednej drogi w drzewie przejść na drugą oraz z podprogramów opisujących działanie poszczególnych instrukcji programu - generatora w czasie analizy (odmienne od podanego wcześniej działania w czasie generacji).

Podamy teraz sieci działań i opisy poszczególnych podprogramów.

1. INITIATION. Przed wykonaniem pierwszej instrukcji programu-generatora następuje inicjacja analizy obejmująca następujące czynności:

1a/ zapełnienie stosu SYMBOL badanym ciągiem symboli,
1b/ umieszczenie na szczycie tego stosu symbolu finish,
1c/ ustalenie maksymalnej głębokości wywołania procedury - MAX,
1d/ nadanie zmiennym stanu wartości początkowych:

```
SP  := 1.
A   := Ø.
Q   := 1.
K   := 1.
PP  := Ø.
AP  := 1.
IC  := START.
GP  := Ø.
GG  := 1.
G   := 1.
AA  := 1.
```

1e/ Wpisanie w zerowej pozycji stosu STACK węzła stosu, co oznacza aktywizację węzła zerowego:

Y,A,B,C,D,E,F,G,H, of STACK(A) :=x,A,Q,PP,AP,IC,GP,GG, G.*)

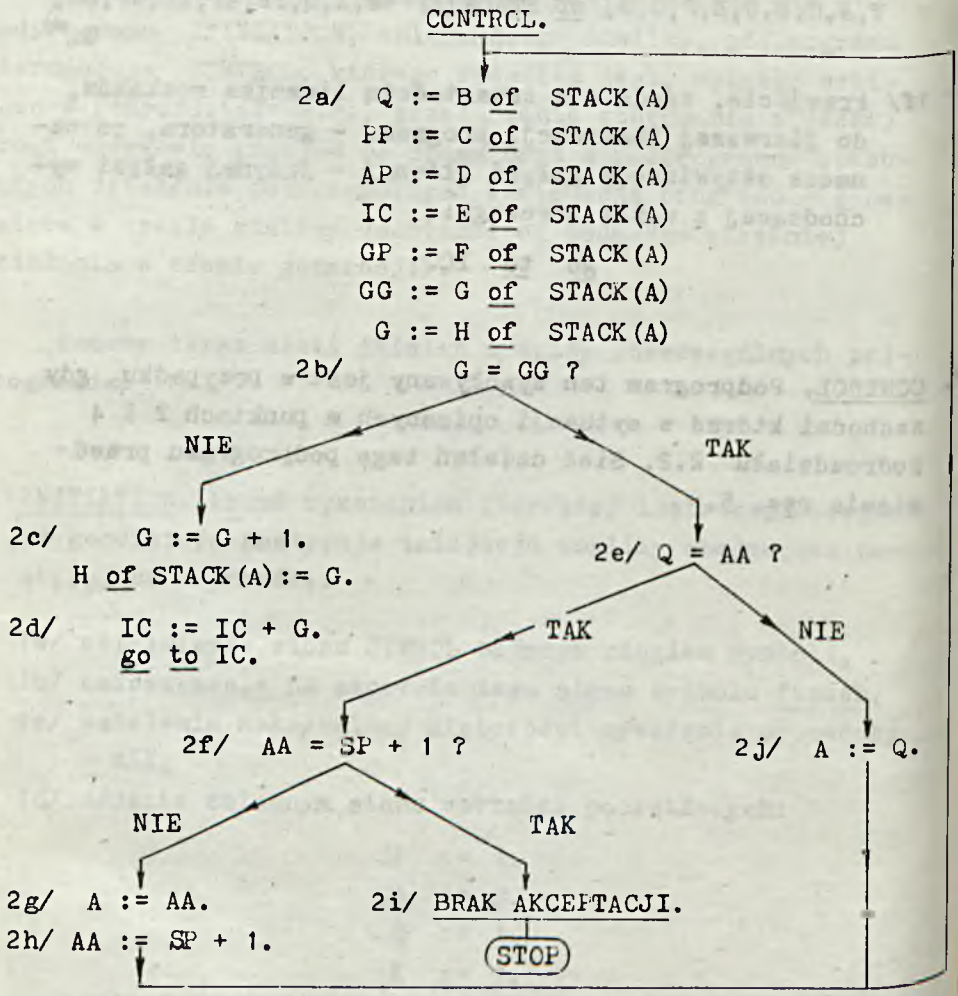
1f/ Przejście, zgodnie z zawartością licznika rozkazów, do pierwszej instrukcji programu - generatora, co oznacza aktywizację gałęzi głównej - jedynej gałęzi wychodzącej z węzła zerowego:

go to IC.

2. CCNTRCL. Podprogram ten wywoływany jest w przypadku, gdy zachodzi któraś z sytuacji opisanych w punktach 2 i 4 podrozdziału 2.2. Sieć działań tego podprogramu przedstawia rys. 5.

*) Jest to skrót następującego zapisu

```
Y of STACK A := x.
A of STACK A := A.
B of STACK A := Q.
  :     :
H of STACK A := G.
```



Rys. 5

Znaczenie poszczególnych czynności jest następujące:

2a/ przywraca stan, odpowiadający momentowi napotkania węzła odniesienia w trakcie analizy gałęzi prowadzącej do tego węzła,

2b/ kontroluje, czy była już badana ostatnia gałąź wychodząca z tego węzła,

gdy NIE, to:

2c/ powoduje aktywizację następnej gałęzi, wychodzącej z węzła odniesienia,

2d/ powoduje przejście do odpowiedniej instrukcji jp (.) - instrukcji skoku do początku zaktywizowanej gałęzi,

gdy TAK, to:

2e/ bada, czy zostały wyczerpane wszystkie węzły na poziomie odniesienia,

gdy NIE, to:

2j/ przechodzi do następnego węzła równorzędnego,

gdy TAK, to:

2f/ bada, czy na danym poziomie drzewa wszystkie gałęzie zostały wyeliminowane,

gdy TAK, to:

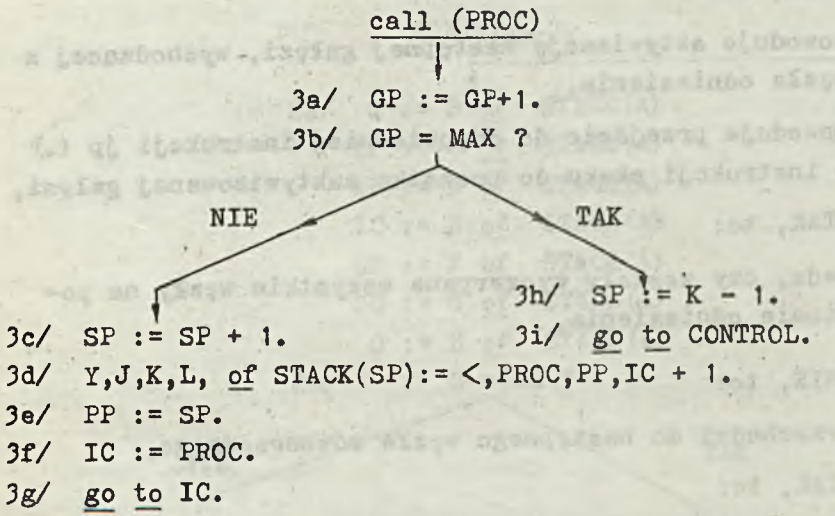
2i/ nie akceptuje badanego ciągu symboli i kończy analizę,

gdy NIE, to:

2h/ zmienia poziom odniesienia,

2g/ aktywizuje pierwszy węzeł, umieszczony na nowym poziomie odniesienia.

3. Call (PROC), gdzie PROC oznacza dowolną etykietę ciała procedury.



Rys. 6

Znaczenie poszczególnych czynności jest następujące:

3a/ zwiększa wskaźnik głębokości procedury

3b/ kontroluje, czy nie nastąpiło zbyt głębokie wywołanie procedury,

gdy NIE, to:

3c,d/ zapisują na stos STACK otwarcie procedury. Poszczególne parametry tej pozycji oznaczają:

- Y - rodzaj pozycji,
- J - nazwę procedury,
- K - adres poprzedniego otwarcia procedury w aktualnie rozpatrywanym łańcuchu dynamicznym stosu STACK,
- L - adres powrotu z ciała otwartej właśnie procedury,

3e,f,g/ zmieniają PP,IC i powodują wejście w ciało procedury

gdy TAK, to:

3h/ skreśla bieżący moduł ze stosu eliminując w ten sposób odpowiednią gałąź drzewa przejść

3i/ przechodzi do wykonania podprogramu CONTROL.

4. rnd(n), gdzie $n \geqslant 2$.

```
rnd(n).
   |
   v
4a/  GG := n.
4b/  G := Ø.
4c/  Q := SP + 1.
4d/ Y,A,B,C,D,E,F,G,H of STACK(K) :=x,A,Q,PP,AP,IC,GP,GG,G.
4e/  go to CONTROL.
```

Rys. 7

Instrukcja rnd(.) (patrz także podrozdział 2.2. punkt 4b) kończy zawsze tworzenie modułu w stosie STACK. Dokonuje także operacji związanych z dojściem w drzewie przejść do węzła podrzędnego.
Znaczenie poszczególnych czynności jest następujące:

4a/ zapamiętuje liczbę gałęzi wychodzących z węzła podrzędnego,

4b/ nadaje wskaźnikowi G wartość zero, przez co nie aktywizuje jeszcze żadnej gałęzi wychodzącej z tego węzła. Nastąpi to dopiero w trakcie analizy następnego poziomu drzewa,

4c/ ustala łącze poziome Q dla kończonego modułu. Q wskazuje teraz początek modułu następnego, jeszcze nie utworzonego,

4d/ zapisuje w pierwszej pozycji kończonego modułu węzeł stosu,

4e/ przechodzi do wykonania podprogramu CONTROL.

5. jp (A_i), gdzie A_i oznacza dowolną etykietę gałęzi.

jp (A_i).

5a/ SP := SP + 1.
5b/ K := SP.
5c/ IC := A_i.
5d/ go to IC.

Rys. 8

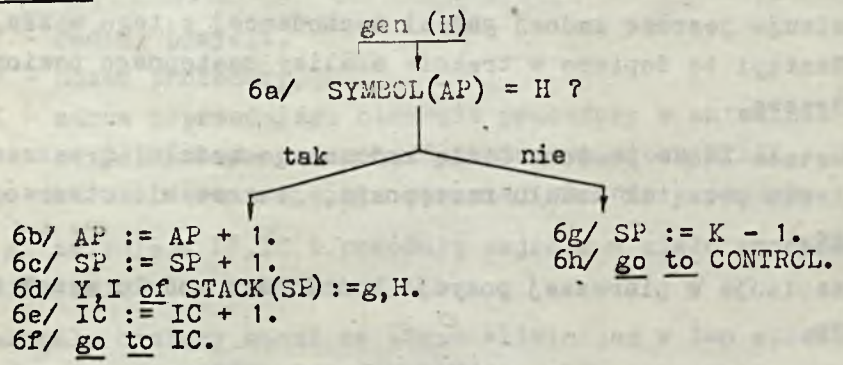
Instrukcja jp (.) rozpoczyna zawsze tworzenie nowego modułu w stosie STACK, co odpowiada wejściu w nową gałąź drzewa przejść.
Znaczenie poszczególnych czynności jest następujące:

5a/ rezerwuje jedną pozycję stosu STACK, w której, w momencie kończenia modułu, zapamiętany zostanie węzeł stosu,

5b/ zmienia wskaźnik początku bieżącego modułu,

5c,d/ przekazują sterowanie do pierwszej instrukcji zaktywizowanej gałęzi.

6. gen (H), gdzie H oznacza dowolny symbol języka.



Rys. 9

Znaczenie poszczególnych czynności instrukcji gen (.) jest następujące:

6a/ sprawdza, czy parametr operacji gen(.) jest zgodny z aktualnie rozpatrywanym symbolem badanego ciągu symboli,

gdy TAK, to:

6b/ przechodzi do rozpatrywania następnego symbolu tego ciągu,

6c,d/ zapisują na stos STACK generację. Poszczególne parametry tej pozycji oznaczają:

Y - rodzaj pozycji,

I - generowany symbol,

6e,f/ przekazują sterowanie do następnej instrukcji programu generatora.

gdy NIE, to:

6g/ skreśla bieżący moduł ze stosu STACK,

6h/ przechodzi do wykonania podprogramu CONTROL.

7. return.

```
                return.
                   |
7a/  GP := GP - 1.
7b/  SP := SP + 1.
7c/  Y of STACK(SP) :=>.
7d/  PP := K of STACK(PP)
     IC := L of STACK(PP)
7e/  go to IC.
```

Rys. 10

Instrukcja return zamyka ciało ostatniej procedury aktualnie rozpatrywanego łańcucha procedur (patrz podrozdział 2.1.). Otwarcie tej procedury jest zapisane w stosie STACK pod adresem PP. Znaczenie poszczególnych czynności jest następujące:

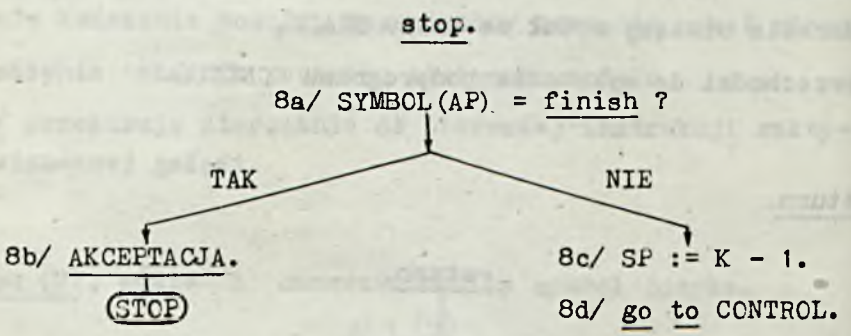
7a/ zmniejsza wskaźnik głębokości wywołania procedury,

7b,c/ zapisują na stos STACK zamknięcie procedury,

7d/ przywraca wartość wskaźnika PP sprzed ostatniego wywołania procedury i nastawia licznik rozkazów IC na adres powrotu z ciała procedury,

7e/ przechodzi do wykonania instrukcji umieszczonej pod tym adresem.

8. stop.



Rys. 11

Znaczenie poszczególnych czynności jest następujące:

8a/ sprawdza, czy osiągnięty został koniec badanego ciągu,

gdy NIE, to:

8c/ skreśla bieżący moduł ze stosu STACK,

8d/ przechodzi do wykonania podprogramu CONTRCL,

gdy TAK, to:

8b/ akceptuje badane zdanie i podaje jego rozkład gramatyczny oraz kończy algorytm.

Rozkład gramatyczny zdania podawany jest na podstawie drogi w drzewie przejść, na której nastąpiła akceptacja. Polega on na wydrukowaniu niektórych pozycji stosu STACK, umieszczonych w łańcuchu dynamicznym stosu, odpowiadającym tej drodze. Pozycje te są drukowane w kolejności od dołu do góry stosu. Węzły stosu są ignorowane.

Wypisywane są jedynie niektóre parametry pozostałych pozycji. Są to odpowiednio:

dla otwarcia procedury

Y,J np. <BODY:

dla zamknięcia procedury

Y tzn. >

dla generacji

I np. beg

za parametrem J (oznacza on nazwę otwieranej procedury) wypisywany jest dwukropek.

Na przykład, dla języka z przykładu 2 rozkład gramatyczny zdania: beg B end jest następujący:

<PROGRAM: beg <BODY : <CONTENTS : B>> end>

Wykazane to będzie w poniższym przykładzie.

Przykład

Mechanizm analizy zostanie prześledzony dla języka z przykładu 2 , programu-generatora z przykładu 3 oraz drzewa przejść z rys.1.

Należy sprawdzić, czy ciąg symboli

beg B end

jest poprawnym zdaniem ww języka.
Śledzenie polegać będzie na przerywaniu pracy analizatora, wypisaniu ciągu instrukcji programu-generatora, wykonanych między dwoma kolejnymi przerwaniami oraz podaniu zawartości stosu STACK i wartości zmiennych stanu w momencie przerwania.

1. INITIATICN
Inicjacja analizy daje w chwili przekazania sterowania do pierwszej instrukcji programu-generatora następujący stan analizatora.

Stos SYMBCL.

4. finish. k = 3.

3. end. g = 3. gdyż nie trzeba uwzględniać zmiennej <symbol>.

2. B.

1. beg. MAX = (k+1) g = 12.

Stos STACK.

1.

ø. x, ø, 1, ø, 1, START, ø, 1, 1.

Wartości zmiennych stanu dla każdego kroku podano w tabeli 1.
Stos SYMBCL oraz liczba MAX nie ulegają zmianie w czasie analizy.

2. START: call (PROGRAM)
Operacja ta zapisuje na stosie STACK pozycję: <, PROGRAM,

Ø, START+1, zmienia wskaźniki GP, SP, PP, IC i przechodzi do wykonania procedury PROGRAM.

Stos STACK.

3.

2. < , PROGRAM, Ø, START+1.

1.

Ø. x, Ø, 1, Ø, 1, START, Ø, 1, 1.

3. PROGRAM: gen (beg)
Operacja ta, po stwierdzeniu, że jej parametr beg jest zgodny z symbolem umieszczonym w pierwszej (AP=1) pozycji stosu SYMBOL, zapisuje na stosie STACK pozycję g, beg i przekazuje sterowanie do następnej instrukcji.

Stos STACK.

4.

3. g, beg.

2. < , PROGRAM, Ø, START+1.

1.

Ø. x, Ø, 1, Ø, 1, START, Ø, 1, 1.

4. PROGRAM+1: call (BODY)

Stos STACK.

5.

4. < , BODY, 2, PROGRAM+2.

3. g, beg.

2. < , PROGRAM, Ø, START+1.

1.

ø. x, ø, 1, ø, 1, START, ø, 1, 1.

5. BODY: rnd (2) (po dojściu do czynności 4e).

Operacja ta, przez odpowiednie ustawienie wskaźnika początku następnego modułu Q oraz zapisanie w pierwszej pozycji (K=1) stosu STACK łącz międzymodułowych oraz stanu analizatora, kończy tworzenie w stosie STACK pierwszego modułu. Zapisana pozycja jest węzłem stosu i odpowiada węzłowi W1 z rys.1.

Stos STACK.

5.
4. <, BODY, 2, PROGRAM+2.
3. g, beg.
2. <, PROGRAM, ø, START+1.
1. x, ø, 5, 4, 2, BODY, 2, 2, ø.

moduł 1.

ø. x, ø, 1, ø, 1, START, ø, 1, 1.

6. BODY: rnd (2) (po wykonaniu podprogramu CCNTRCL).

W trakcie wykonywania podprogramu CCNTRCL przywracany jest stan analizatora z chwili zakończenia inicjacji.
Odpowiada to "cofnięciu" się do węzła zerowego (węzeł WØ na rys.1) , który jest węzłem odniesienia.
Po stwierdzeniu, że wszystkie gałęzie wychodzące z tego węzła zostały wyczerpane (czynność 2b) i że wszystkie węzły na poziomie odniesienia zostały wyczerpane (czynność 2e), poziom odniesienia zostaje zmieniony (czynność 2h).
Następnie aktywizowane są: węzeł BODY (W1 na rys.1), który jest pierwszym węzłem na nowym poziomie odniesienia

(czynności 2g, a) oraz pierwsza wychodząca z niego gałąź (czynność 2c) oraz następuje przejście do instrukcji jp (A1) .

Stos STACK.

5.

4. <, BODY, 2, PROGRAM+2.

3. g, beg.

2. <, PROGRAM, ø, START+1.

1. x, ø, 5, 4, 2, BODY, 2, 2, 1. (zmianie uległ parametr H)

ø. x, ø, 1, ø, 1, START, ø, 1, 1.

7. BODY+1: jp (A1)

A1: call (CONTENTS)

Stos STACK.

7.

6. <, CONTENTS, 4, A1+1.

5.

4. <, BODY, 2, PROGRAM+2. ┐

3. . . ├ b.z.

2. . ┘

8. CONTENTS: rnd (2)

BODY+2: jp (A2)

A2: call (BODY)

Operacja rnd (2) powoduje zapisanie w pozycji 5 stosu STACK węzła stosu, odpowiadającego węzłowi W2 z rys.1

oraz aktywizację drugiej gałęzi wychodzącej z węzła odniesienia W1. Operacja jp (A2) powoduje przejście do wykonania pierwszej instrukcji tej gałęzi tzn. call (BODY). Ta ostatnia instrukcja powoduje rekurencyjne wywołanie procedury BODY i zapisuje na szczycie stosu pozycję: ∠, BODY, 4, A2+1.

Stos STACK.

9.

8. <, BODY, 4, A2+1.

7.

6. <, CONTENTS, 4, A1+1.

5. x, 1, 7, 6, 2, CONTENTS, 3, 2, ø.

4. <, BODY, 2, PROGRAM+2.

3. g, beg.

2. <, PROGRAM, ø, START+1.

1. x, ø, 5, 4, 2, BODY , 2, 2, 2. (zmianie uległ parametr H)

ø. x, ø, 1, ø, 1, START, ø, 1, 1.

9. BODY: rnd (2)

CONTENTS+1: jp (A3)

A3: gen (B)

Operacja rnd (2) powoduje zapisanie w pozycji 7 stosu STACK węzła stosu, odpowiadającego węzłowi W3 z rys.7. Ponieważ wszystkie gałęzie wychodzące z węzła odniesienia W1 zostały przeanalizowane oraz brak jest na poziomie odniesienia innych węzłów, następuje zmiana poziomu odniesienia i aktywizacja węzła W2 i pierwszej wychodzącej z niego gałęzi.

W wyniku tego wykonane zostają instrukcje jp (A3) oraz gen (B).

Stos STACK.

11.

10. g, B.

9.

8. <, BODY, 4, A2+1.

7. x, 1, 9, 8, 2, BODY , 3, 2, Ø.

6. <, CONTENTS, 4, A1+1.

5. x, 1, 7, 6, 2, CONTENTS, 3, 2, 1.(zmianie uległ parametr H)

4. <, BODY, 2, PROGRAM+2.

3. . b.z.

2. .

10. A3+1: return.

Operacja ta zamyka ciało procedury CONTENTS, której otwarcie zapisane zostało w szóstej pozycji stosu STACK (PP=6).

Na szczycie stosu zapisane zostaje zamknięcie procedury, przywracana jest wartość PP sprzed otwarcia procedury CONTENTS oraz następuje powrót z ciała tej procedury pod adres A1+1.

Stos STACK.

12.
11. >.
10. g, B.
9.
8. <, BODY, 4, A2+1. } b.z.
7. .
6. .

11. A1+1: return.
PROGRAM+2: gen (end)
PROGRAM+3: return.

Stos STACK.

15.
14. >.
13. g, end.
12. >.
11. >.
10. .
9. . } b.z.
8. .

12. START+1: stop.

Operacja ta po stwierdzeniu, że osiągnięty został koniec badanego ciągu (czynność 8a) akceptuje ten ciąg i podaje rozkład gramatyczny opierając się na zawartości stosu STACK, którą na poniższym rysunku przedstawiono w nieco zmodyfikowanej formie.

Stos STACK.

	14.	>.	
	13.	g, end.	moduł 4.
	12.	>.	
	11.	>.	
	10.	g, B.	
	9.	nie zapełniona (łącze A=5)	
	8.	<, BODY, 4, A2+1.	moduł 3.
W3.	7.	x, 1, 9, 8, 2, BODY , 3,2, ø.	
	6.	<, CONTENTS, 4, A1+1.	moduł 2.
W2.	5.	x, 1, 7, 6, 2, CONTENTS, 3, 2, 1.	
	4.	<, BODY, 2, PROGRAM+2.	
	3.	g, beg.	moduł 1.
	2.	<, PROGRAM, ø, START+1.	
W1.	1.	x, ø, 5, 4, 2, BODY , 2, 2, 2.	
WØ.	ø.	x, ø, 1, ø, 1, START, ø,1, 1.	węzeł zerowy.

Rys. 12

Moduły 1, 2 i 4 tworzą jeden łańcuch dynamiczny stosu. Odpowiada on drodze w drzewie przejść z rys. 4, oznaczonej linią kreskowaną, która prowadzi do akceptacji badanego zdania. Zawartość ww łańcucha, zredukowana zgodnie z podanymi wcześniej zasadami, tworzy rozkład gramatyczny tego zdania. Rozkład ten jest następujący:

<PROGRAM: beg <BODY: <CONTENTS: B>> end>

Tabela 1

Wartości zmiennych stanu

Numer kroku	SP	A	Q	PP	AP	IC	GP	GG	G	AA	K
1.	1	ø	1	ø	1	START.	ø	1	1	1	1
2.	2	ø	1	2	1	PROGRAM.	1	1	1	1	1
3.	3	ø	1	2	2	PROGRAM+1.	1	1	1	1	1
4.	4	ø	1	4	2	BODY.	2	1	1	1	1
5.	4	ø	5	4	2	BODY.	2	2	ø	1	1
6.	4	1	5	4	2	BODY+1.	2	2	1	5	1
7.	6	1	5	6	2	CONTENTS.	3	2	1	5	5
8.	8	1	5	8	2	BODY.	3	2	2	5	7
9.	10	5	7	6	3	A3+1.	3	2	1	9	9
10.	11	5	7	4	3	A1+1.	2	2	1	9	9
11.	14	5	7	ø	4	START+1.	ø	2	1	9	9

3. DOWÓD TWIERDZENIA O MAKSYMALNEJ GŁĘBOKOŚCI WYWOŁANIA PROCEDURY

Przed przystąpieniem do właściwego dowodu twierdzenia wprowadzimy kilka pojęć pomocniczych, które od razu ilustrować będziemy przykładami.
Weźmiemy pod uwagę dowolny opis języka podany w notacji Backusa, np. opis następujący:

Przykład 4.

<P> ::= <C> <P> - aksjomat
<C> ::= b<T>|<C><L>
<T> ::= <E><A>|b<T> c

<A> ::= <E>
<E> ::= ε
<L> ::= a|d
<symbol> ::= a|b|c|d|ε

1. Zakładamy, że wszystkie gałęzie występujące w metajęzykowej instrukcji podstawienia są ponumerowane w kolejności od lewej do prawej. Mocą zmiennej nazywamy liczbę gałęzi występujących w instrukcji podstawienia odpowiadającej tej zmiennej.

2. Wypisujemy kolejno wszystkie produkcje języka (z wyjątkiem produkcji odpowiadających nieosiągalnej zmiennej <symbol>) i oznaczamy je tzw. oznacznikami produkcji, w których skład wchodzi nazwa zmiennej języka, stojącej po lewej stronie produkcji, oraz numer gałęzi, której ta produkcja odpowiada (w dalszych przykładach dla uproszczenia pomijać będziemy nawiasy trójkątne, co nie prowadzi do niejednoznaczności, gdyż symbole języka z przykł.4 oznaczane są małymi literami, a zmienne języka - dużymi).

Przykład 5.

P1 : P→C
C1 : C→bT
C2 : C→CL
T1 : T→EA
T2 : T→bTc
A1 : A→E
E1 : E→ε
L1 : A→a
L2 : L→d

Produkcje posiadające pusty następnik, albo następnik, w którym występują (być może obok zmiennych) niepuste symbole języka, nazywamy produkcjami tworzącymi. Oznaczamy je przez

podkreślenie oznaczników.
Produkcje posiadające identyczny poprzednik nazywamy jednoimiennymi.

3. Sznurem nazywamy dowolny ciąg zmiennych i symboli.

4. Wywodem równoległym nazywamy ciąg sznurów

$$(X_0, X_1, \ldots, X_n)$$

spełniający następujące warunki:

a/ $X_\emptyset$ jest aksjomatem

b/ X_n jest sznurem złożonym z samych symboli, tzn. nie zawiera zmiennych

c/ $\forall$ (i = 1, 2, ... , n) sznur X_i otrzymany jest przez jednoczesne zastosowanie do wszystkich zmiennych występujących w sznurze X_{i-1} produkcji ze zbioru produkcji języka.

Poszczególne sznury wywodu oddzielamy znakiem "⟶".
Liczbę n nazywamy długością wywodu równoległego.

Dla języka z przykładu 4 wywodem równoległym jest np.

$X_\emptyset$ X_1 X_2 X_3 X_4 X_5 X_6

$$P \longrightarrow C \longrightarrow C\ L \longrightarrow b\ T\ a \longrightarrow b\ E \cdot A\ a \longrightarrow b\ E\ a \longrightarrow b\ a \qquad (3)$$

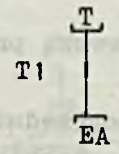
P1 C2 C1 L1 T1 E1 A1 E1

pod zmiennymi każdego sznura wypisano oznaczniki produkcji zastosowanych do tych zmiennych. W sznurach X_2 i X_4 zastosowano jednocześnie dwie produkcje. Znaki ϵ pominięto. Długość powyższego wywodu wynosi 6.

5. Wprowadzimy teraz wygodny w dalszych rozważaniach sposób zapisywania produkcji i wywodów równoległych. Na przykład produkcję:

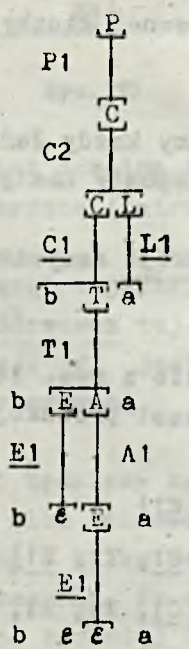
$$T1 : \quad T \longrightarrow EA$$

zapisywać będziemy w postaci:



Rys. 13

zaś wywód ze wzoru (3) w postaci:



Rys. 14

znaki ε będziemy czasami pomijać.

6. Łańcuchem produkcji nazywamy ciąg produkcji wywodu równoległego:

$$(P_1, P_2, \ldots, P_k)$$

spełniający następujące warunki:

1°. Produkcja P_1 jest pierwszą produkcją wywodu.

2°. $\forall (i = 2, 3, \ldots, k)$ poprzednikiem produkcji P_i jest zmienna występująca w następniku produkcji P_{i-1}.

3°. Produkcja P_k jest produkcją tworzącą.

Podłańcuchem nazywamy część łańcucha otrzymaną przez usunięcie z tego łańcucha pewnej liczby początkowych produkcji.

Łańcuchem głównym nazywamy każdy łańcuch, którego ostatnia produkcja posiada niepusty następnik.

Długością łańcucha produkcji nazywamy liczbę produkcji tego łańcucha.

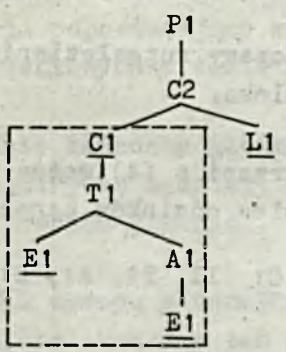
Dla przykładu w wywodzie z rys. 14. rozróżnić można 4 łańcuchy produkcji (zamiast produkcji wypisano ich oznaczniki).

$$(P1, C2, \underline{C1}) \qquad (4)$$
$$(P1, C2, \underline{C1}, T1, \underline{E1})$$
$$(P1, C2, \underline{C1}, T1, A1, \underline{E1})$$
$$(P1, C2, \underline{L1})$$

Łańcuchy pierwszy i czwarty są łańcuchami głównymi. Długość łańcucha trzeciego jest największa i wynosi 6.

Można łatwo wykazać, że długość wywodu równoległego jest równa długości najdłuższego łańcucha wchodzącego w skład tego wywodu.

7. Każdy wywód równoległy można przedstawić jako drzewo wchodzących w jego skład produkcji. Na przykład wywód z rys. 14. jest następującym drzewem produkcji



Rys. 15

Łańcuchem produkcji jest dowolna droga w tym drzewie zakończona produkcją tworzącą (patrz (4)).

Część drzewa "wychodzącą" z jakiejś produkcji wraz z tą produkcją nazywamy poddrzewem tej produkcji.
Np. poddrzewem produkcji C1 jest na rys. 15 część drzewa wydzielona linią przerywaną.

8. Rozgałęzieniem nazywać będziemy każdą produkcję drzewa produkcji, która jest albo produkcją tworzącą, albo posiada następnik złożony z kilku zmiennych.
Rozgałęzienie ma tę własność, że albo kończy jakiś łańcuch, albo należy do co najmniej 2 łańcuchów.
Rozgałęzienie nazywamy istotnym, gdy jest ono produkcją tworzącą lub należy do co najmniej 2 łańcuchów głównych.
W przeciwnym przypadku rozgałęzienie nazywamy nieistotnym.

Na przykład w drzewie produkcji z rys. 15 produkcje C2, C1, L1, E1, E1 są rozgałęzieniami istotnymi. Produkcja T1 jest rozgałęzieniem nieistotnym.

9. Odcinkiem łańcucha nazywamy ciąg produkcji łańcucha zawartych między dwoma rozgałęzieniami istotnymi lub między początkiem łańcucha, a pierwszym rozgałęzieniem istotnym łańcucha.

Do odcinka łańcucha wliczamy rozgałęzienie istotne wyznaczające koniec tego odcinka.

Dla przykładu łańcuch trzeci z (4) można podzielić na następujące odcinki (granice odcinków oznaczono znakiem II)

(II P1, C2 II C1 II T1, A1, E1 II)

Lemat

Z] Niech rozpatrywany język posiada g zmiennych. Dany jest wywód równoległy W_1 kończący się sznurem x złożonym z k symboli $(k \geqslant \emptyset)$.

T] Istnieje wywód równoległy W_2 o długości nie większej niż $g \cdot (k + 1)$, kończący się sznurem x.

D] Niech s_i $(i = 1, 2, \ldots k)$ będą jakimiś niepustymi symbolami języka i niech sznur x ma postać następującą

$$s_1 \quad s_2 \ldots \quad s_j \ldots \quad s_k.$$

Każdy z symboli s_i musi występować w następniku jakiejś produkcji wywodu W_1, przy czym kilka symboli może występować w jednym i tym samym następniku. Każda taka produkcja jest oczywiście produkcją tworzącą i kończy jeden łańcuch główny wywodu W_1.

Stąd łańcuchów głównych może być najwyżej k. Pozostałe łańcuchy (nazwijmy je niegłównymi) kończą się produkcjami o pustym następniku. Usunięcie takich łańcuchów z wywodu (właściwie chodzi tutaj o ich pewne podłańcuchy, sposób usuwania będzie podany niżej) nie zmienia oczywiście sznura końcowego.

Przedstawmy wywód W_1 w postaci drzewa produkcji.

Będziemy przekształcać to drzewo w taki sposób, aby otrzymać drzewo produkcji odpowiadające wywodowi równoległemu W_2 o własnościach wymienionych w tezie lematu.

Ponumerujmy wszystkie łańcuchy główne wywodu W_1 i podzielmy je na odcinki. Rozpatrywać będziemy po kolei wszystkie odcinki kolejnych łańcuchów głównych.

Gdy w ramach odcinka wywodu głównego występuje kilka produkcji jednoimiennych, to skracamy ten odcinek w sposób następujący.

Z drzewa produkcji usuwamy poddrzewo pierwszej produkcji jednoimiennej i zastępujemy je poddrzewem ostatniej produkcji jednoimiennej. Przez takie przekształcenie usuwamy z danego odcinka wszystkie produkcje zawarte między pierwszą produkcją jednoimienną (wraz z tą produkcją) i ostatnią produkcją jednoimienną. Usunięte produkcje nie są rozgałęzieniami istotnymi, a więc żaden łańcuch główny nie zostaje usunięty.

Usunięty fragment odcinka, o ile zawiera rozgałęzienia nieistotne, może dawać początek podłańcuchom zakończonym produkcjami o pustym następniku. Wszystkie te podłańcuchy są także usuwane. Po skróceniu w ten sposób wszystkich odcinków łańcuchów głównych skracamy podobnie wszystkie, nie należące do łańcuchów głównych, części łańcuchów niegłównych.

Części te są podłańcuchami składającymi się z produkcji, z których tylko ostatnia jest rozgałęzieniem istotnym. Wynika stąd, że każdy taki podłańcuch składa się tylko z jednego odcinka.

Po skróceniu wszystkich łańcuchów wywodu W_1 otrzymujemy wywód W_2, w którym wszystkie odcinki łańcuchów nie zawierają produkcji jednoimiennych, a więc mają długość nie większą niż g.
Każdy łańcuch główny tego wywodu zawierać może oczywiście najwyżej k rozgałęzień istotnych, a więc najwyżej k odcinków. Stąd każdy łańcuch główny ma długość nie większą niż k·g.

Każdy łańcuch niegłówny podzielić można na dwie części: należącą i nie należącą do jakiegoś łańcucha głównego. Pierwsza ma długość najwyżej k·g, druga - składająca się z jednego odcinka - najwyżej g.

Stąd każdy łańcuch wywodu W_2 ma długość nie większą niż $(k + 1) \cdot g$. c.b.d.o.

Dla wywodu twierdzenia wystarczy tylko zauważyć, że pojęcie łańcucha produkcji jest analogiczne do pojęcia łańcucha procedur, wywód równoległy jest odpowiednikiem drogi w drzewie przejść programu-generatora, zaś długość łańcucha jest analogonem głębokości wywołania procedury. Z uwagi tej oraz z lematu wynika, że o ile sznur x o długości k jest poprawnym zdaniem języka, to w drzewie przejść programu-generatora istnieje droga akceptująca to zdanie, na której głębokość wywołania procedury nie przekracza liczby $(k + 1) g$. Stąd wszystkie drogi nie spełniające tego warunku mogą być wyeliminowane (chociaż, w przypadku gdy gramatyka języka jest niejednoznaczna, również one prowadzić mogą do akceptacji).

4. UWAGI KOŃCOWE

Zaprogramowanie przedstawionego algorytmu jest pracą stosunkowo łatwą. Główne ograniczenie stanowi duży stos, potrzebny przy analizowaniu skomplikowanych języków. Autorzy zamierzają w najbliższym czasie zaprogramować algorytm w języku wewnętrznym maszyny K-202 i przedstawić otrzymane wyniki w kolejnym artykule.

Algorytm może być wykorzystany przy projektowaniu fazy translatora zwanej analizą syntaktyczną. Jest on szczególnie przydatny w trakcie projektowania języka, gdy następują częste zmiany w jego opisie.
Algorytm jest bardziej efektywny w przypadku opisywania języka za pomocą rekursji prawostronnych. Np. w opisie ALGOL-u podanym w [3] korzystne byłoby, tam, gdzie jest to możliwe, zastąpienie rekursji lewostronnych prawostronnymi.
Eliminowanie dróg w drzewie przejść programu-generatora następowałoby wtedy znacznie szybciej, przy jednoczesnym zmniejszeniu wielkości stosu STACK.

Literatura

[1] BLIKLE A.: Automaty i Gramatyki, PWN, Warszawa, 1971.

[2] RANDELL B., RUSSELL L.J.: ALGOL 60 Implementation, Academic Press, London and New York, 1964.

[3] PASZKOWSKI S.: Język ALGOL 60 - Dodatek A - Opis Języka Algorytmicznego ALGOL 60, PWN, Warszawa, 1965.

МЕТОД ИСПОЛЬЗОВАНИЯ ПРОГРАММЫ-ГЕНЕРАТОРА ПРОИЗВОЛЬНОГО ЯЗЫКА ЗАПИСАННОГО НА ФОРМЕ БАККУСА КАК АКЦЕПТОРА ЭТОГО ЯЗЫКА

Резюме

В работе представлены принципы конструкции и действия программы-генератора произвольного бесконтекстного языка записанного на форме Баккуса. Такой генератор делает возможной генерацию произвольного предложения принадлежащего к такому языку. Затем описан алгоритм, который - пользуя программу-генератор - допускает решить, принадлежит ли произвольный, арбитрарно избранный ряд символов к данному языку, или нет. Процесс такого решения называется процессом акцептации. Последним результатом алгоритма является грамматическое разложение исследоваемого ряда символов или сигнал об его непринадлежности к языку.

Примененный метод в процессе акцептации заключается в ответственном, управленном символами исследоваемого ряда, движении по так называемом дереве переходов программы-генератора. Доказанием ограниченности алгоритма является сформулированная и доказанная теорема о найбольшей глубине вызвания процедуры.

APPLICATION METHOD OF PROGRAM-GENERATOR OF ANY LANGUAGE WRITTEN IN BACKUS NOTATION AS AN ACCEPTOR OF THIS LANGUAGE

Summary

At the beginning of the paper the mechanism of a program-generator of any context-free language written in BNF is presented. Such generator enables generating any sentence that belongs to such a language. Next, the algorithms, that uses mentioned above program-generator, makes possible to decide whether any arbitrary chosen sequence of symbols belongs to a given language or not is described. The process of such deciding is called acceptation process. The final product of the algorithm is parsing of examinated sequence of symbols, or the signal that it does not belong to the language. The method used in acceptation process consists in corresponding, controlled by symbols of examined sequence, moving about so-called transition tree of program-generator. The authors formulated and proved the theorem about maximal depth of procedure call that is a justification of algorithm termination.

Warszawa, ul. Krzywickiego 34
A.M.