STUDIA UNIVERSITATIS BABEȘ-BOLYAI

MATHEMATICA

3

1989

CLUJ-NAPOCA

ANUL XXXIV 1989

STUDIA UNIVERSITATIS BABEŞ-BOLYAI

MATHEMATICA

3

Redacţia: 3400 CLUJ-NAPOCA, str. M. Kogălniceanu, 1 ● Telefon 1 61 01

SUMAR — CONTENTS

STUDIA UNIV. BABEŞ—BOLYAI, MATHEMATICA, XXXIV, 3, 1989

A LOCAL OPTIMIZATION PROBLEM FOR DATA BASE REDISTRIBUTION IN A COMPUTER NET

GRIGOR MOLDOVAN* and SERGIU DAMIAN**

Received: July 11, 1989

REZUMAT. — O problemă de optimizare locală pentru redistribuirea bazelor de date intr-o reţea de calculatoare. Se consideră o problemă de optimizare pentru baze de date distribuite. Se porneşte de la o reţea de calculatoare în nodurile căreia sint distribuite subbaze de date. Se formulează problema generală a redistribuirii subbazelor de date în reţea, plecînd de la o definiţie locală a gradului de aglomerare. Problema este rezolvată în cazul particular a două lanţuri, momentele de lansare fiind presupuse cunoscute, iar redistribuirea făcindu-se într-un interval de timp dat.

1. **Introduction.** Consider a computer net determined by a connected graph $G = (C, \bar{U})$, where $C = \{C_1, C_2, \ldots, C_n\}$ is the set of computers (which determine the nodes of the net), while $\bar{U}$ is the set of the edges linking the nodes. Suppose that in the nodes of the net the subbases B_k, $k = \overline{1, n}$ are distributed, of a data base denoted by:

$$B = \{B_1, B_2, \ldots, B_n\},$$

such that the subbase B_k lies initially in the node C_k. For simplicity reasons, we shall use hereafter the term *base* instead of *subbase*.

A redistribution of the bases B_k, $k = \overline{1, n}$ in the nodes of the net is determined by the permutation:

$$\sigma = \begin{pmatrix} 1 & 2 & \ldots & n \\ i_1 & i_2 & \ldots & i_n \end{pmatrix}$$

with the significance that the base B_k lying in the node C_k, identified by the index k, is transferred into the node C_{i_k}, identified by the index i_k.

During the redistribution process of the data bases in the net, there appears the phenomenon of crowding in some net nodes while performing the transfer operations. Choosing suitably the instants for launching the transfers, into a given time interval, the crowding in nodes can be diminished or even removed. Some considerations concerning this problem were made in [1—4]. Considering a global measure of the crowding degree with respect to the whole net, we obtained certain results in [5, 6].

The present paper will consider the crowding phenomenon at local level, in every node of the set.

* *University of Cluj-Napoca, Faculty of Mathematics and Physics, 3400 Cluj-Napoca, Romania*
** *University of Cluj-Napoca, Computing Data Center, 3400 Cluj-Napoca, Romania*

2. **General Problem. Definitions. Notations.** We shall introduce the following notations:

$L_k = (k, i_1^k, i_2^k, \ldots, i_{p_k}^k)$ = the chain on which the base B_k travels from the source node k to the destination node $i_{p_k}^k$;

$\mathcal{T}_k = \{t_k, t_k + d_p^k, \ldots, t_k + d_{i_{p_k}}^k\}$ = the set of instants at which the base B_k crosses the nodes of the chain L_k;

t_k = the instant at which the transfer of the base B_k from the node k is launched;

$d_{i_p}^k$ = the transfer duration of the base B_k between the nodes k and i_p^k $(p = \overline{1, p_k})$.

Some specifications must be made here:

— the chains L_k are supposed to be known;

— the station duration of the base B_k in the node i_p^k is assumed to be contained into the value $d_{i_p}^k$;

— the values $d_{i_p}^k$ are considered to be known and liable to a translation with a value $t^k \in [0, T]$, where $[0, T]$ represents a given time interval, such that, after the translation with t^k on the chain L_k, the set of instants at which the base B_k crosses the nodes of the chain L_k becomes:

$$\mathcal{T}'_k = \{t_k + t^k, t_k + d_{i_1}^k + t^k, \ldots, t_k + d_{i_{p_k}}^k + t^k\}.$$

Now, let $\mathcal{L}_k$ be the set of chains on which the transfers are made and which contain the node k. We shall consider further down only those $\mathcal{L}_k$ for which:

— the reunion of the nodes of their chains, on the whole, form a connected component G' of G, with n' nodes $(n' \leqslant n)$ and l chains;

— there exists at least one node k belonging to at least two chains, *i.e.* being crossed by at least two different bases (namely $|\mathcal{L}_k| \geqslant 2$).

Denote by $\mathcal{N}$ the set of nodes of the graph G' endowed with these properties, which provide a meaning to the crowding problem.

The following definitions lead to a local measure of the crowding degree in the redistribution process:

DEFINITION 1. For fixed t^i values $(i = \overline{1, l})$, the *relaxation degree* r_k in the node $k \in \mathcal{N}$ is the minimum value of the distances between the time instants at which two different bases cross the node k, namely:

$$r_k = \min_{i,j} |t_k^i - t_k^j|, \; i \neq j, \; L_i, L_j \in \mathcal{L}_k,$$

where:

$$t_k^i = t_i + d_k^i + t^i \text{ (or } t_k^i \in \mathcal{T}'_i),$$
$$t_k^j = t_j + d_k^j + t^j \text{ (or } t_k^j \in \mathcal{T}'_j).$$

DEFINITION 2. For fixed t^i values $(i = \overline{1, l})$, the *crowding degree* g_k in the node $k \in \mathfrak{N}$ is the inverse of the relaxation degree in the node k, namely:

$$g_k = \begin{cases} 1/r_k, & \text{if } r_k \neq 0; \\ \infty, & \text{if } r_k = 0. \end{cases}$$

DEFINITION 3. For fixed t^i values $(i = \overline{1, l})$, the value $r = \min\limits_{k \in \mathfrak{N}} r_k$ is called *relaxation degree of the net* G' in the redistribution process.

DEFINITION 4. For fixed t^i values $(i = \overline{1, l})$, the value $g = \max\limits_{k \in \mathfrak{N}} g_k$ is called *crowding degree of the net* G' in the redistribution process.

With this, we formulate the following

PROBLEM. *Taking into account the conditions and notations which were introduced above, determine* $t^i \in [0, T]$, $i = \overline{1, l}$, *such that, after translating the initial instants* t_i *with* t^i, *the relaxation degree of the net* G' *in the redistribution process is maximum, or, equivalent, the crowding degree of the net* G' *is minimum; in other words, determine* $t^i \in [0, T]$, $i = \overline{1, l}$, *such that*:

$$\min_{k \in \mathfrak{N}} r_k \to \max$$

or:

$$\max_{k \in \mathfrak{N}} g_k \to \min.$$

We shall give further down a solution of this problem in the peculiar case of two chains of the net G, along which the transfer of two data bases is performed in a given time interval, with the assumption that the chains have at least one common node (hence they constitute a G'-type subnet).

3. **Solution in a Peculiar Case.** Let $L_1 = \{1, i_1^1, i_2^1, \ldots, i_{p_1}^1\}$ and $L_2 = \{2, i_1^2, i_2^2, \ldots, i_{p_2}^2\}$ be two chains along which the contents of the bases B_1 (between the nodes 1 and $i_{p_1}^1$) and B_2 (between the nodes 2 and $i_{p_2}^2$) are respectively transferred. Let also:

$$\mathcal{T}_1 = \{t_1, t_1 + d_{i_1}^1, t_1 + d_{i_2}^1, \ldots, t_1 + d_{i_{p_1}}^1\},$$

$$\mathcal{T}_2 = \{t_2, t_2 + d_{i_1}^2, t_2 + d_{i_2}^2, \ldots, t_2 + d_{i_{p_2}}^2\}$$

be the sets of given time instants at which the bases B_1 and B_2 cross the nodes of the chains L_1 and L_2, respectively.

Considering that the transfers are performed during a given time interval $[0, T_0]$, we have:

$$0 \leqslant t_k < t_k + d_{i_1}^k < t_k + d_{i_2}^k < \ldots < t_k + d_{i_{p_k}}^k \leqslant T_0, \quad k = \overline{1, 2}.$$

Since L_1 and L_2 have at least one common node, we denote by $\mathfrak{N}_c$ the set of common nodes, in which the crowding problem can appear:

$$\mathfrak{N}_c = L_1 \cap L_2 = \{n_1, n_2, \ldots, n_p\}, \qquad 0 < p \leqslant \min(p_1, p_2).$$

Corresponding to the nodes from $\mathfrak{N}_c$, denote by:

$$t^k_{n_i} = t_k + d^k_{n_i}, \quad k = \overline{1, 2}, \; i = \overline{1, p},$$

the time instants at which the base B_k crosses the common node n_i.

Assuming that the elements of $\mathfrak{T}_1$ are fixed, while those of $\mathfrak{T}_2$ are liable to a translation with $t \in [0, T]$, where $T = T_0 - t_2 - d^2_{i_p}$, the problem formulated in Section 2 reduces to the determination of the value $t^* \in [0, T]$, such that, after translating the values from $\mathfrak{T}_2$ with t^*, the minimum distance between the time instants corresponding to the crossing of a node from $\mathfrak{N}_c$ by B_1 and B_2, respectively, is as far as possible maximum.

Hence, we have to determine $t^* \in [0, T]$ such that:

$$\min_{i,t} |t + t^2_{n_i} - t^1_{n_i}| \to \max, \; i = \overline{1, p}, \; t \in [0, T],$$

namely the relaxation degree of the net consisting of L_1 and L_2 (with the above hypotheses) is maximum.

Denoting $d_{n_i} = t^1_{n_i} - t^2_{n_i}$, $i = 1, p$, the problem can be rewritten as follows:

$$\min_{i,t} |t - d_{n_i}| \to \max, \; i = \overline{1, \text{p}}, \; t \in [0, T]. \tag{1}$$

Before performing any translation, the instant at which B_2 crosses the node n_i is earlier than, later than, or coincides with the instant at which B_1 crosses the node n_i, according to the relations $d_{n_i} > 0$, $d_{n_i} < 0$ or $d_{n_i} = 0$, respectively.

Ordering increasingly the values d_{n_i}, we obtain:

$$d_{n_{i_1}} \leqslant d_{n_{i_2}} \leqslant \ldots \leqslant d_{n_{i_p}},$$

or, denoting $d_j = d_{n_{i_j}}$, $j = \overline{1, p}$, we have:

$$d_1 \leqslant d_2 \leqslant \ldots \leqslant d_p. \tag{2}$$

Let us now introduce some functions (all defined on $[0, \infty)$, with values into $[0, \infty)$) as follows:

$$f_j(t) = |t - d_j|, \qquad j = \overline{1, p};$$
$$f(t) = \min_j f_j(t), \qquad j = \overline{1, p};$$
$$F(T) = \max_t f(t), \qquad t \in [0, T].$$

With this, the following proposition holds:

PROPOSITION 1. *The function F is increasing.*

Proof. Let be $0 \leqslant T_1 < T_2$. We have:

$$F(T_1) = \max_{t \in [0,T_1]} f(t) \leqslant \max\left(\max_{t \in [0, T_1]} f(t), \max_{t \in [T_1, T_2]} f(t)\right) = \max_{t \in [0, T_2]} f(t) = F(T_2),$$

therefore Proposition 1 is true. In fact, this proposition represents a general feature of the function max in the case $[0, T_1] \subset [0, T_2]$.

We give further down some propositions which lead to the solution of the problem defined by the formula (1).

PROPOSITION 2. *If* $d_j \leqslant 0$, $j = \overline{1, p}$, *then* $F(T) = T - d_p$ *and* $t^* = T$.

Proof. Since $d_j \leqslant 0$ and $t \geqslant 0$, it result that:

$$f(t) = \min_j |t - d_j| = \min_j (t - d_j) = t - \max_j d_j = t - d_p, \ j = 1, p.$$

The obtained function being strictly increasing with respect to t, it results:

$$F(T) = \max_t (t - d_p) = T - d_p, \quad t \in [0, T],$$

and this relation holds for $t^* = T$, *q.e.d.*

Graphically, this case is exemplified in Fig. 1.

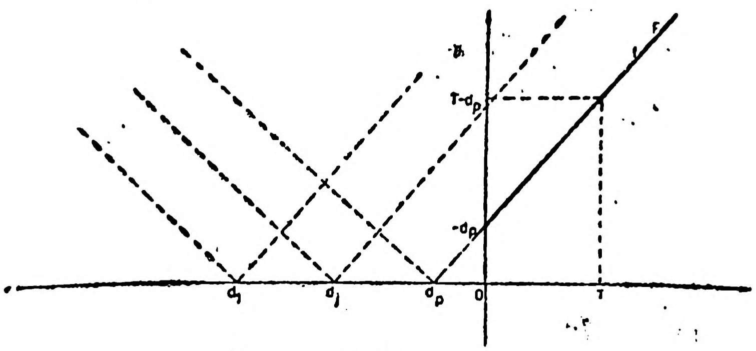


Fig. 1.

For $k = \overline{1, p-1}$, we introduce the notations:

$$\begin{aligned} m_k &= (d_k + d_{k+1})/2, & m_0 &= 0,\ m_p = \infty; \\ l_k &= (d_{k+1} - d_k)/2, & l_0 &= d_1; \\ I_k &= [d_k, m_k], & I_p &= [d_p, \infty); \\ J_k &= [m_k, d_{k+1}], & J_0 &= [0, d_1]. \end{aligned}$$

PROPOSITION 3. *If* $d_j > 0$, $j = \overline{1, p}$, *then*:

$$f(t) = \begin{cases} t - d_k, & \text{if } t \in I_k,\ k = \overline{1, p}; \\ -t + d_{k+1}, & \text{if } t \in J_k,\ k = \overline{0, p-1}. \end{cases}$$

Proof. By (2) we have:

$$-t + d_1 \leqslant -t + d_2 \leqslant \ldots \leqslant -t + d_p$$

and:

$$t - d_1 \geqslant t - d_2 \geqslant \ldots \geqslant t - d_p. \tag{3}$$

Taking into account (2) and (3), we obtain:

a) If $t \in J_0$, then:

$$0 \leqslant -t + d_1 \leqslant \ldots \leqslant -t + d_p,$$

leading to:

$$f_j(t) = -t + d_j, \ j = \overline{1, p};$$

hence:

$$f(t) = \min_j (-t + d_j) = -t + \min_j d_j = -t + d_1, \ \ j = \overline{1, p}.$$

b) If $t \in I_k \cup J_k$, $k = \overline{1, p-1}$, then:

$$d_1 \leqslant d_2 \leqslant \ldots \leqslant d_k \leqslant t \leqslant d_{k+1} \leqslant \ldots \leqslant d_p,$$

leading to:

$$-t + d_1 \leqslant -t + d_2 \leqslant \ldots \leqslant -t + d_k \leqslant 0 \leqslant -t + d_{k+1} \leqslant \ldots \leqslant -t + d_p,$$

and:

$$f_j(t) = \begin{cases} t - d_j, & j = \overline{1, k}; \\ -t + d_j, & j = \overline{k+1, p}, \end{cases}$$

hance:

$$f(t) = \min_{j=\overline{1,p}} f_j(t) = \min \left(\min_{j=\overline{1,k}} (t - d_j), \min_{j=\overline{k+1,p}} (-t + d_j)\right) =$$

$$= \min (t - d_k, -t + d_{k+1}) = \begin{cases} t - d_k, & \text{if } t \in I_k; \\ -t + d_{k+1}, & \text{if } t \in J_k. \end{cases}$$

c) If $t \in I_p$, then:

$$-t + d_1 \leqslant -t + d_2 \leqslant \ldots \leqslant -t + d_p \leqslant 0,$$

leading to:

$$f_j(t) = t - d_j, \ j = \overline{1, p},$$

hence:

$$f(t) = \min_j (t - d_j) = t - \max_j d_j = t - d_p, \ j = \overline{1, p}.$$

From these three results we obtain by reunion Proposition 3. The corresponding graphic exemplification is represented on Fig. 2.

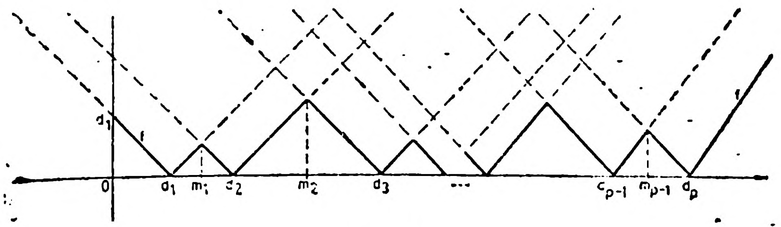


Fig. 2.

PROPOSITION 4. *If $d_j > 0$, $j = \overline{1, p}$, then:*
— *if $T \in J_{k_0}$ (k_0 fixed, $0 \leqslant k_0 \leqslant p$), then:*

$$F(T) = \max_k l_k, \quad k = \overline{0, k_0};$$

— *if $d_{k_0} \leqslant T \leqslant \min (d_{k_0} + \max_k l_k, m_{k_0})$, with $k = \overline{0, k_0 - 1}$ and $1 \leqslant k_0 \leqslant p$, then:*

$$F(T) = \max_k l_k, \quad k = \overline{0, k_0 - 1};$$

— *if $d_{k_0} + \max_k l_k \leqslant T \leqslant m_{k_0}$, with $k = \overline{0, k_0 - 1}$ and $1 \leqslant k_0 \leqslant p$, then:*

$$F(T) = T - d_{k_0}.$$

Proof. From Proposition 3 and the monotonicity of the function f we have:

$$\max_t f(t) = \max_t (t - d_k) = l_k, \quad t \in I_k, \quad k = \overline{1, p - 1};$$
$$\max_t f(t) = \max_t (-t + d_{k+1}) = l_k, \quad t \in J_k, \quad k = \overline{0, p - 1}.$$

If $T \in I_k$, then:

$$\max_t f(t) = \max_t (t - d_k) = T - d_k, \quad t \in [d_k, T], \quad k = \overline{1, p}.$$

If $T \in J_k$, then:

$$\max_t f(t) = \max_t (-t + d_{k+1}) = l_k, \quad t \in [m_k, T], \quad k = \overline{0, p - 1}.$$

Taking into account these estimates, the definition of the function F and the properties of the function max on reunions of intervals, we differentiate the following cases:

a) If $T \in J_0$, it results $[0, T] = [m_0, T] \subset J_0$, therefore:

$$F(T) = \max_t f(t) = l_0, \quad t \in [m_0, T].$$

b) If $T \in J_{k_0}$, $k_0 = \overline{1, p - 1}$, $p \geqslant 2$, k_0 is fixed, we have:

$$t \in [0, T] = \left(\bigcup_{k=0}^{k_0-1} J_k\right) \cup \left(\bigcup_{k=1}^{k_0} I_k\right) \cup [m_{k_0}, T],$$

hence:

$$F(T) = \max \left(\max_{k=\overline{0, k_0-1}} l_k, \max_{k=\overline{1,k_0}} l_k, l_{k_0}\right) = \max_{k=\overline{0,k_0}} l_k.$$

c) If $T \in I_1$, we have $[0, T] = J_0 \cup [d_1, T]$, hence:

$$F(T) = \max \left(\max_{t \in J_0} f(t), \max_{t \in [d_1, T]} f(t)\right) = \max (l_0, T - d_1) =$$
$$= \begin{cases} l_0, & \text{if } d_1 \leqslant T \leqslant d_1 + l_0; \\ T - d_1, & \text{if } d_1 + l_0 \leqslant T. \end{cases}$$

Since $T \in I_1 = [d_1, m_1]$, we obtain:

$$F(T) = \begin{cases} l_0, & \text{if } d_1 \leqslant T \leqslant \min(d_1 + l_0, m_1); \\ T - d_1, & \text{if } d_1 + l_0 \leqslant T \leqslant m_1. \end{cases}$$

d) If $T \in I_{k_0}$, $2 \leqslant k_0 \leqslant p$, k_0 is fixed, we have:

$$t \in [0, T] = \left(\bigcup_{k=0}^{k_0-1} J_k\right) \cup \left(\bigcup_{k=1}^{k_0-1} I_k\right) \cup [d_{k_0}, T],$$

therefore:

$$F(T) = \max\left(\max_{k=\overline{0, k_0-1}} l_k, \max_{k=\overline{1, k_0-1}} l_k, T - d_{k_0}\right) = \max\left(\max_{k=\overline{0, k_0-1}} l_k, T - d_{k_0}\right) =$$

$$= \begin{cases} \max\limits_k l_k, & \text{if } d_{k_0} \leqslant T \leqslant \min(d_{k_0} + \max\limits_k l_k, \mathrm{m}_{k_0}); \\ T - d_{k_0}, & \text{if } d_{k_0} + \max\limits_k l_k \leqslant T \leqslant \mathrm{m}_{k_0}, \end{cases}$$

where $k = \overline{0, k_0 - 1}$ and the intersection with $I_{k_0} = [d_{k_0}, m_{k_0}]$ was performed. Observe that for $k_0 = p$, since $m_p = \infty$, we have:

$$\min(d_p + \max_k l_k, \mathrm{m}_p) = d_p + \max_k l_k, \quad k = \overline{0, p - 1}.$$

Performing the reunion of the explicit results obtained for F in the four above cases, the validity of Proposition 4 is proved.

Now, we introduce the notation:

$$D_k = \max_{i=\overline{0, k-1}} l_k, \quad k = \overline{1, p},$$

by means of which one associates to each interval having the form

$$J_0 = [0, d_1], \; I_{k-1} \cup J_{k-1} = [d_{k-1}, d_k], \; k = \overline{2, p},$$

the value D_k, where $k = \overline{1, p}$.

Observe that:

$$D_{k+1} = \max(D_k, l_k), \; k = \overline{1, p - 1}. \tag{4}$$

With this notation, we can rewrite Proposition 4 as follows:

PROPOSITION 5. *If $d_j > 0$, $j = \overline{1, p}$, then:*

$$F(T) = \begin{cases} D_{k+1}, & \textit{if } T \in J_k, \; k = \overline{0, p - 1}; \\ D_k, & \textit{if } d_k \leqslant T \leqslant \min(d_k + D_k, m_k), \; k = \overline{1, p}; \\ T - d_k, & \textit{if } d_k + D_k \leqslant T \leqslant m_k, \; k = \overline{1, p}. \end{cases}$$

The function F, corresponding to the function f from Fig. 2, is graphically represented in Fig. 3, in which one can notice the monotonicity of the function F.

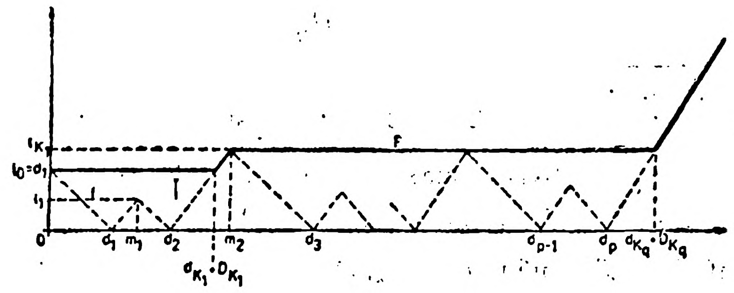


Fig. 3.

Since $D_k \leqslant \max(D_k, l_k) = D_{k+1}$, $k = \overline{1, p-1}$, the following ordering holds:

$$D_1 \leqslant D_2 \leqslant \ldots \leqslant D_p, \tag{5}$$

The strictly increasing values from (5), if they exist, lead to the transcription:

$$\begin{aligned} &D_1 = \ldots = D_{k_1} < D_{k_1+1} = \ldots = D_{k_2} < \\ &< D_{k_2+1} = \ldots = D_{k_q-1} < D_{k_q+1} = \ldots = D_{k_q}, \end{aligned} \tag{6}$$

which emphasizes a division of the values D_k (and, implicitly, a division of the values d_k) and of the intervals corresponding to the values D_k into classes. The number of these classes is $q \geqslant 1$, while k_j, $j = \overline{1, q}$, is the maximum index of the value D_k which determines the class. The same division into classes is determined by the constant branches of the function F.

Denoting such a class by $\mathcal{C}_j$, we have:

$$\mathcal{C}_j = \{D_k | k = \overline{k_{j-1}+1, k_j}\}, \quad j = \overline{1, q}, \quad k_0 = 0,$$

namely a class $\mathcal{C}_j$ is formed by those values D_k for which $D_k = D_{k_j}$.

To consider the classes $\mathcal{C}_j$ leads to the diminution of the number of intervals into which T lies in Proposition 5, according to the more synthetic statement given by:

PROPOSITION 6. *If* $d_i > 0$, $i = \overline{1, p}$, *then*:

$$F(T) = \begin{cases} D_{k_j} & \text{if } m_{k_{j-1}} \leqslant T \leqslant d_{k_j} + D_{k_j},\ j = \overline{1, q}; \\ T - d_{k_j}, & \text{if } d_{k_j} + D_{k_j} \leqslant T \leqslant m_{k_j},\ j = \overline{1, q}. \end{cases}$$

Proof. By Proposition 5 and (6), we have for every $p \geqslant 1$:

$$F(T) = \begin{cases} D_1 = D_{k_1}, & \text{if } T \in J_0; \\ D_p = D_{k_q}, & \text{if } d_p \leqslant T \leqslant d_p + D_p; \\ T - d_p, & \text{if } d_p + D_p \leqslant T \leqslant m_p. \end{cases}$$

For every $p \geqslant 2$ and for every class $\mathcal{C}_j$, $j = \overline{1, q}$, we have:

— If $D_k \in \mathcal{C}_j$ and $T \in J_k$, from Proposition 5 and formula (6) it results:

$$F(T) = D_{k+1} = D_{k_j} \in \mathcal{C}_j,$$

for each $k = \overline{k_{j-1} + 1, k_j - 1}$.

— If $D_k \in \mathcal{C}_j$ and $|\mathcal{C}_j| \geqslant 2$ ($\mathcal{C}_j$ has at least two elements), we have:

$$D_k = D_{k+1} = \max (D_k, l_k) \Leftrightarrow D_k \geqslant l_k \Leftrightarrow$$
$$\Leftrightarrow d_k + D_k \geqslant m_k \Rightarrow \min (d_k + D_k, m_k) = m_k.$$

In this case, if $d_k \leqslant T \leqslant \min (d_k + D_k, m_k)$, from Proposition 5 and formula (6) it results:

$$F(T) = D_k = D_{k_j} \in \mathcal{C}_j,$$

for each $k = \overline{k_{j-1} + 1, k_j - 1}$.

— If $q \geqslant 2$ and $k_j < p$, we have:

$$D_{k_j} < D_{k_j+1} = \max (D_{k_j}, l_{k_j}) \Leftrightarrow D_{k_j} < l_{k_j} \Leftrightarrow$$
$$\Leftrightarrow d_{k_j} + D_{k_j} < m_{k_j} \Rightarrow \min (d_{k_j} + D_{k_j}, m_{k_j}) = d_{k_j} + D_{k_j}.$$

In this case, if $d_{k_j} \leqslant T \leqslant \min (d_{k_j} + D_{k_j}, m_{k_j})$, from Proposition 5 and formula (6) it results:

$$F(T) = D_{k_j},$$

and, if $d_{k_j} + D_{k_j} \leqslant T \leqslant m_{k_j}$, it results:

$$F(T) = T - d_{k_j}.$$

Performing the reunion of the intervals for which F has the same expression, we obtain Proposition 6, which emphasizes the constant branches and, respectively, the strictly increasing branches of the function F.

Consider now the case in which there exist values $d_i \leqslant 0$, $i = \overline{1, p_0}$, and values

$$d_i > 0, \quad i = \overline{p_0 + 1, p}, \quad p \geqslant 2, \quad 1 \leqslant p_0 < p,$$

namely (2) becomes:

$$d_1 \leqslant d_2 \leqslant \ldots \leqslant d_{p_0} \leqslant 0 < d_{p_0+1} \leqslant \ldots \leqslant d_p. \tag{7}$$

With this hypothesis, the following proposition holds:

PROPOSITION 7. a) *If $-d_{p_0} \geqslant d_{p_0} + 1$, then the expression of the function F is given by Proposition 6 applied to the values $d_i > 0$, $i = \overline{p_0 + 1, p}$.*

b) *If $-d_{p_0} < d_{p_0+1}$, then the expression of the function F is given by Proposition 6 applied to the values $d_i > 0$, except the case $0 \leqslant T \leqslant d_{k_1} + D_{k_1}$,* *when we have:*

$$F(T) = \begin{cases} T - d_{p_0}, & \text{if } 0 \leqslant T \leqslant m_{p_0} = (d_{p_0} + d_{p_0+1})/2; \\ D_{k_1}, & \text{if } m_{p_0} \leqslant T \leqslant d_{k_1} + D_{k_1}, \end{cases}$$

where $D_{k_1} = l_{p_0} = (d_{p_0+1} - d_{p_0})/2$.

Proof. a) Since $d_i \leqslant 0$, $i = \overline{1, p_0}$, and $t \geqslant 0$, by applying Proposition 2 to these values, it results:

$$\min_{i=\overline{1,p_0}} f_i(t) = t - d_{p_0}.$$

The condition $-d_{p_0} \geqslant d_{p_0+1}$ leads to:

$$m_{p_0} = (d_{p_0} + d_{p_0+1})/2 \leqslant 0 \leqslant t,$$

or:

$$t - d_{p_0} \geqslant -t + d_{p_0+1}.$$

For $t \in J_{p_0} = [0, d_{p_0+1}]$, by Proposition 3 we have:

$$\min_{i=\overline{1,p_0}} f_i(t) = t - d_{p_0} \geqslant -t + d_{p_0+1} = \min_{i=\overline{p_0+1,p}} f_i(t). \tag{8}$$

For $t \in [d_{p_0+1}, \infty)$, we have:

$$t - d_{p_0} > t - d_{p_0+1} \geqslant 0,$$

therefore, taking into account (7) and Proposition 3:

$$\min_{i=\overline{1,p_0}} f_i(t) = t - d_{p_0} > t - d_{p_0+1} \geqslant \min_{i=\overline{p_0+1,p}} f_i(t). \tag{9}$$

From (8) and (9) it results:

$$f(t) = \min_{i=\overline{1,p}} f_i(t) = \min_{i=\overline{p_0+1,p}} f_i(t).$$

Hence, in the case $-d_{p_0} \geqslant d_{p_0+1}$, the function f has the expression given by Proposition 3, corresponding to the values $d_i > 0$ and $J_0 = J_{p_0}$, $i = \overline{p_0+1, p}$, which leads to the expression of the function F given by Proposition 6 for these values. The graphic representation is given in Fig. 4.

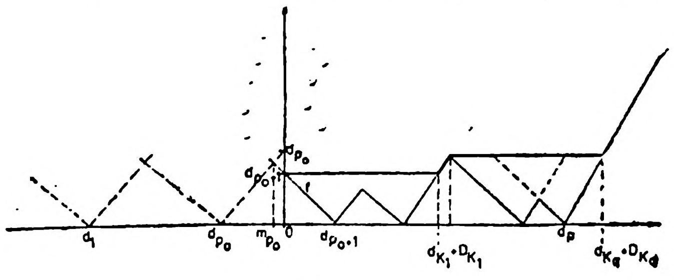


Fig. 4.

b) If $-d_{p_0} < d_{p_0+1}$, then:

$$m_{p_0} = (d_{p_0} + d_{p_0+1})/2 \geqslant 0.$$

For $t \in [0, m_{p_0}]$, it results:

$$t - d_{p_0} \leqslant -t + d_{p_0+1},$$

thus:

$$\min_{i=\overline{1,p_0}} f_i(t) \leqslant \min_{i=\overline{p_0+1,p}} f_i(t). \tag{10}$$

For $t \in [m_{p_0}, d_{p_0+1}]$, it results:

$$t - d_{p_0} \geqslant -t + d_{p_0+1},$$

thus:

$$\min_{i=\overline{1,p_0}} f_i(t) \geqslant \min_{i=\overline{p_0+1,p}} f_i(t). \tag{11}$$

For $t \in [d_{p_0+1}, \infty)$, the relations (9) hold.
From (9), (10) and (11), it results:

$$f(t) = \min_{i=\overline{1,p}} f_i(t) = \begin{cases} \min\limits_{i=\overline{1,p_0}} f_i(t) = t - d_{p_0}, & \text{if } t \in [0, m_{p_0}]; \\ \min\limits_{i=\overline{p_0+1,p}} f_i(t), & \text{if } t \in [m_{p_0}, \infty). \end{cases}$$

Hence, in the case when $-d_{p_0} < d_{p_0+1}$, the function f has the expression given by Proposition 3 for $t \geqslant m_{p_0}$, on the interval J_0, by replacing with the intervals $I_{p_0} = [0, m_{p_0}]$ and $J_{p_0} = [m_{p_0}, d_{p_0+1}]$.

A proof analogous to that of Propositions 4, 5 and 6 leads to the conclusion atated at the point b).

The corresponding graphic representation is given in Fig. 5.

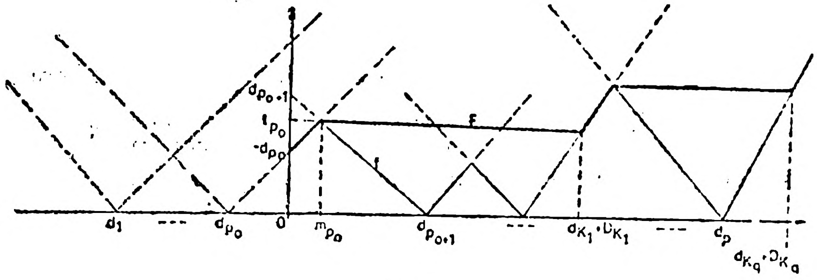


Fig. 5.

Propositions 2, 6 and 7 exhaust all possible cases in the determination of the function F.

Taking into account the monotonicity of the functions which appear while expliciting, the maximum values which determine the function F on different intervals, during the proof of Propositions 2, 4 and 7, are obtained for values $t \in [0, T]$ of the form $t = 0$, $t = T$ or $t = m_k$. The notations we introduced lead to the synthetic formulations of Propositions 6 and 7. The values $t^* \in [0, T]$ which are solutions of the problem formulated in this Section are specified by:

PROPOZITION 8. a) *If* $d_i \leqslant 0$, $i = \overline{1, p}$, *then* $t^* = T$.

b) *If* $d_i > 0$, $i = \overline{1, p}$, *then*:

$t^* = T$, *if* $d_{k_j} + D_{k_j} \leqslant T \leqslant m_{k_j}$;

$t^* = 0$ *or* $t^* = m_{k-1}$, *such that* $l_{k-1} = D_{k_1}$, $k \leqslant k_1$, *and* $m_{k-1} \leqslant T$, *if* $0 \leqslant T \leqslant d_{k_1} + D_{k_1}$;

$t^* = m_{k-1}$, *such that* $l_{k-1} = D_{k_j}$, $k \leqslant k_j$, *and* $m_{k-1} \leqslant T$, *if* $m_{k_{j-1}} \leqslant T \leqslant d_{k_j} + D_{k_j}$.

c) *If there exist* $d_i \leqslant 0$, $i = \overline{1, p_0}$, *and* $d_i > 0$, $i = \overline{p_0 + 1, p}$, *and if* $-d_{p_0} \geqslant d_{p_0+1}$, *then* t^* *is given by the point b), considering only* $d_i > 0$, $i = \overline{p_0 + 1, p}$.

d) *If there exist* $d_i \leqslant 0$, $i = \overline{1, p_0}$, *and* $d_i > 0$, $i = \overline{p_0 + 1, p}$, *and if* $-d_{p_0} < d_{p_0+1}$, *then* t^* *is given by the point c), except the case* $0 \leqslant T \leqslant d_{k+1} + D_{k+1}$, *when we have:*

$t^* = T$, *if* $0 \leqslant T \leqslant m_{p_0}$;

$t^* = m_{k-1}$, *such that* $l_{k-1} = D_{k_1} = l_{p_0}$ *and* $m_{k-1} \leqslant T$, *if* $m_{p_0} \leqslant T \leqslant d_{k_1} + D_{k_1}$.

4. **Comments.** Proposition 8 provides the values $t^* \in [0, T]$ with which, if the initial instants on the chain L_2 are translated, one obtains the maximum relaxation degree or the minimum crowding degree for the net consisting of the chains L_1 and L_2.

Observe that in each case there exists at least one solution; in some cases there exist several solutions for those intervals from the class $\mathfrak{C}_j$ for which $l_{k-1} = D_k$.

If $t^* = 0$, then, in order to obtain the required optimum, no translation is performed on the chain L_2, because the existing initial situation cannot be improved.

If $t^* = T$, the translation is performed with the maximum possible value.

If $t^* = m_{k-1}$, there exists at least the solution $t^* = m_{k_{j-1}}$.

Proposition 8 offers the cases to be analysed in the algorithm for the determination of t^* as function of T, the determination of the strictly increasing sequence D_{k_j} being essential.

REFERENCES

1. L. Gârbacea, Gr. Moldovan, *Distributed Data Bases*, Univ. of Cluj-Napoca, Fac. Math. Res. Sem., Preprint 5 (1984) (Seminar of Models, Structures and Information Processing), 70–90.
2. Gr. Moldovan, *Reorganization of a Distributed Data Base*, Univ. of Cluj-Napoca, Fac. Math. Res. Sem., Preprint 5 (1984) (Seminar of Models, Structures and Information Processing), 3–10.

3. Gr. M o l d o v a n, *O problemă privind bazele de date distribuite*, M.E.I., A VIII-a Consfătuire de lucru şi schimb de experienţă a lucrătorilor din unităţile de informatică. Lucrări prezentate, Constanţa, 29 iulie—4 august 1985, p. 102—103.
4. Gr. M o l d o v a n, *O problemă de optimizare într-un sistem de baze de date distribuite*, Universitatea din Cluj-Napoca, Facultatea de Matematică, Centrul de Calcul Electronic, Simpozionul „Informatica şi aplicaţiile sale", Cluj-Napoca, 20 noiembrie 1986, p. 13—19.
5. Gr. M o l d o v a n, S. D a m i a n, *On some Generalizations of an Optimization Problem for Distributed Data Bases*, Studia Univ. Babeş-Bolyai, Mathematica, *32* (1987), No. 3, 67—76.
6. Gr. M o l d o v a n, S. D a m i a n, *On an Optimization Problem for Distributed Data Bases*, An. Univ. Bucureşti, Mat.—Inf., *37* (1988), No. 2, 82—87.

POISSON SYMBOLIC PROCESSOR

BAZIL PÂRV*

Received: July 7, 1989

REZUMAT. — Asupra unui procesor simbolic Poisson. Lucrarea propune un procesor simbolic pentru manipularea algebrică a expresiilor Poisson. Sint prezentate pe scurt mecanismele interne de reprezentare şi operare. De asemenea, sint descrise operaţiile de implementare şi metafuncţiile de previziune. Acest procesor a fost folosit in unele probleme de mecanică cerească şi poate fi utilizat in orice problemă in al cărei model matematic intervin expresii Poisson.

1. **Introduction.** A qualitative jump in the use of the electronic computer (initially intended to perform only numerical calculations) is actually represented by its use to perform analytical operations. This constitutes a basic component of achieving the so-called intelligent systems.

The first performances in this field were obtained in the seventh decade of our century. For processing the Poisson series, which are of a great interest in celestial mechanics and nonlinear dynamics, constituting the object of this paper, important studies were developed along the last twenty years [1, 2, 4–6, 9–11, 13, 15, 21, 24–25, 27].

Generally, the above mentioned systems are programs written in machine or list-processing languages, whose functions which perform the Poisson series manipulation can be called from programs written in high level languages. Another direction in symbolic processing of Poisson series was the development of specialized programming languages, able to achieve these operations [7, 16, 22–23].

2. **Poisson Series and Expressions.** A Poisson series has the form:

$$S = \sum_{i=0}^{\infty} C_i x_1^{j_1} x_2^{j_2} \dots x_m^{j_m} \begin{matrix}\sin\\ \cos\end{matrix} (k_1 y_1 + k_2 y_2 + \dots + k_n y_n), \tag{1}$$

where: C_i are numerical coefficients; $x_1, x_2, \dots, x_m$ are monomial variables; $y_1, y_2, \dots, y_n$ are trigonometric (angular) variables; $j_1, j_2, \dots, j_m$ and $k_1, k_2, \dots, k_n$ are exponents and, respectively, coefficients. The summation index i covers the set of all possible combinations of the exponents j and coefficients k ($j \in Z^m$, $k \in Z^n$).

The form (1) of the Poisson series can be briefly written as follows:

$$S = \sum_{i=0}^{\infty} T_i, \tag{2}$$

in which T_i is a term of this series:

$$T_i = C_i P_i F_i, \tag{3}$$

* *University of Cluj-Napoca, Computing Centre, 3400 Cluj-Napoca, Romania*

where the monomial part P_i has the form:

$$P_i = x_1^{j_1} x_2^{j_2} \ldots x_m^{j_m}, \tag{4}$$

while the trigonometric part F_i has the form:

$$F_i = \begin{matrix}\sin\\ \cos\end{matrix}(k_1 y_1 + k_2 y_2 + \ldots + k_n y_n). \tag{5}$$

In practice, one does not operate with Poisson series, but with partial sums of these ones, which are usual tools for the analytical theories of celestial mechanics. Such partial sums have the form:

$$S = \sum_{i=0}^{N} T_i, \quad N \in \mathbf{N}, \tag{6}$$

and will be called (Poisson) expressions. The monomial and trigonometric variables and the expressions as well are identified by their names, which are character strings and will be called hereafter symbols.

In what follows we shall present a Poisson symbolic processor (*PSP*) especially devoted to certain problems belonging to celestial mechanics, astrodynamics and related topics.

3. **External Form of Poisson Expression.** The expressions provided by the user to the processor, and those obtained by this last one as a result of the performed operations (expressions provided on output file or listing to the user) as well, must observe some conventions. This represents the so-called external form. *PSP* accepts expressions written in infix notation, in which the operators are: + (addition), − (subtraction), * (multiplication), / (division) and ^ (exponentation). The coefficients C_i are considered to be ordinary fractions:

$$C_i = p_i/q_i, \tag{7}$$

with $p_i \in \mathbf{Z}$, $q_i \in \mathbf{N}^*$. Integer numbers are also considered rational coefficients with the denominator equal to 1. If a term appears as coefficientless, its virtual coefficient is considered 1.

The monomial parts (4) are written as follows:

$$P_i = x_i\,\hat{}\,j_1 * x_2\,\hat{}\,j_2 * \ldots * x_m\,\hat{}\,j_m, \tag{8}$$

where $x_1, x_2, \ldots, x_m$ are the symbols of the monomial variables, while $j \in \mathbf{Z}^*$ (see Section 2). In (8), only the monomial variables with nonzero exponents must be specified.

The trigonometric parts (5) are written as follows:

$$F_i = \begin{matrix}\sin\\ \cos\end{matrix}(k_1 * y_1 + k_2 * y_2 + \ldots + k_n * y_n), \tag{9}$$

where $y_1, y_2, \ldots, y_n$ are the symbols of the trigonometric variables, while $k \in \mathbf{Z}^*$ (see Section 2). In (9), only the trigonometric variables with nonzero coefficients must be specified.

The terms T_i from (3) are written as follows:

$$T_i = C_i * P_i * F_i, \tag{10}$$

where C_i, P_i, F_i have the forms (7), (8) and (9), respectively.

In order to define a term T_i, at least one of its components (C_i, P_i, F_i) must be specified (the others are omitted from the formula (10)).

The Poisson expressions (6) are written under the form of algebraical sums of terms. Such an expression contains at least one term. An expression, in which the terms are arranged in respect to the type of trigonometric function, the set of monomial part exponents, and the set of trigonometric part coefficients is called "normalized".

4. Internal Form of Poisson Expressions. The external form of Poisson expressions represents the interface between the program and the user. In order to operate with these expressions, these ones must have an internal form with the following features [11]:

— unique for a given expression;
— compact (in order to use a small amount of storage);
— rapid in search (a given term may be easy accessed);
— general for such expressions.

For a concrete problem, one must define the monomial and trigonometric variables, and the primitive expressions with which the analytical operations are performed. All expressions with which *PSP* does operate are stored term by term, ordered in respect to their apparition or construction.

For a term of the form (10), the following informations are needed to be stored: its coefficient, the exponents of the monomial variables, the type of the trigonometric function, and the coefficients of the trigonometric variables.

Different methods for representing Poisson expressions are proposed or analysed by [1—2, 5, 11, 16, 21, 24—25]. Generally, in the frame of these methods, the i-th term of a Poisson expression is identified by exponents $j \in \mathbf{Z}^m$, coefficients $k \in \mathbf{Z}^n$, and the type of trigonometric function. In this paper, we considered a representation in which the i-th term of an expression is identified by means of its position in its "normalized" external form. The used method is essentially based on the fact that the coefficients C_i, the exponents j of monomial variables, the coefficients k of trigonometric variables and the terms T_i are stored only once in increasingly ordered lists.

In what follows we shall describe the internal representation manner of the entities above mentioned, starting with the simple types and continuing with the complex structures of data.

The symbols are character strings of at most 4 characters length, which must identify uniquely the represented entities (variable or expression names).

The monomial variables are stored in the list *VP*, ordered with respect to their definition. Every element from this list is a symbol which identifies the respective monomial variable.

The trigonometric variables are stored in the list *VT*, also ordered with respect to their definition. Each element from this list is a symbol, too, which identifies the respective trigonometric variable.

The coefficients C_i are stored in the list TC. An element from this list has the form:

$$C_i : \begin{array}{|c|c|c|}\hline p_i & q_i & Nr \\ \hline\end{array}, \qquad (11)$$

in which p_i and q_i are the numerator and, respectively, the denominator of the coefficient (represented as integers).

The monomial parts (8) are identified by means of the exponents j. They are represented in the list TP as follows:

$$P_i : \begin{array}{|c|c|c|c|c|}\hline j_1 & j_2 & \dots & j_m & Nr \\ \hline\end{array}, \qquad (12)$$

in which the exponents j have integer representation.

The trigonometric parts (9) contain two categories of informations which must be stored: the type of trigonometric function and the coefficients k of the argument of this one. The type of trigonometric function is stored into a byte-type variable (denoted by TF) as follows:

$$TF = \begin{cases} 0, \text{ if } T_i \text{ does not contain the trigonometric part;} \\ 1, \text{ for sine;} \\ 2, \text{ for cosine,} \end{cases} \qquad (13)$$

while the coefficients k are stored into the list TT. Such coefficients are stored into a location of the list as follows:

$$F_i : \begin{array}{|c|c|c|c|c|}\hline k_1 & k_2 & \dots & k_n & Nr \\ \hline\end{array}, \qquad (14)$$

in which the coefficients are represented as integers. We mention that the first nonzero coefficient k_q, $1 \leqslant q \leqslant n$, must be positive.

The terms T_i are stored in the list TV. An element of this list corresponding to the term T_i has the structure:

$$T_i : \begin{array}{|c|c|c|c|c|}\hline TF & AdrTrig & AdrMon & AdrCoef & Nr \\ \hline\end{array}, \qquad (15)$$

where:

$AdrTrig$ = location index from TT where the coefficients of the trigonometric part are stored;

$AdrMon$ = location index from TP where the exponents of the monomial part are stored;

$AdrCoef$ = location index from TC where the coefficient C_i of the term T_i are stored.

All these location indices (pointers) are represented as integers.

The elements from each of the lists TC, TP, TT and TV (elements which will be symbolically denoted by C, P, F and V, respectively) are stored uniquely, by preserving the increasing (lexicographically) ordering of these lists. The field Nr which appears in every above presented data structures signifies the number of apparitions of the respective entity in the stored expressions. This field is represented on one byte.

The expressions are represented by means of two lists. The second such list (hierarchically speaking), symbolically denoted by TE, contains the addresses of the terms of each expression, ordered in respect to their apparition in the "normalized" external form of the respective expression. These addresses are represented as integers constituting the indices of the elements (pointers) from the list TV which are terms of the given expressions. The list TE consists of a set of sublists, every such sublist corresponding to an expression. The elements of each sublist are increasingly ordered. An element from TE will be symbolically denoted by E. In the first list, (denoted by TEV) there is stored a record for each expression, the record having the form:

$$EV : \begin{array}{|c|c|c|}\hline ExprName & FirstAdr & LastAdr \\ \hline\end{array} \tag{16}$$

where:

ExprName = name of expression, represented by the symbol which identifies the expression;

FirstAdr = index of list TE which contains the address of the first term of the expression;

LastAdr = index of list TE which contains the address of the last term of the expression.

FirstAdr and *LastAdr* are represented by integers and are respectively indices of the first and the last location of the sublist from the list TE corresponding to the expression identified by *ExprName*. EV in (16) represents an element of the list TEV.

The structure of above described data is schematically represented in Fig. 1.

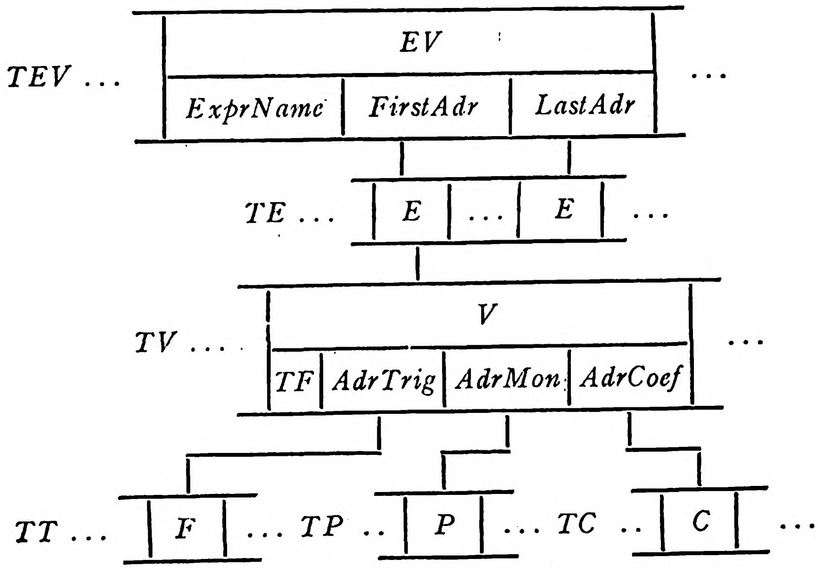


Figure 1. The Hierarchical Structure of Data.

5. **Storage Algorithms.** The data lists presented in Section 4 were ordered as we specified in order to perform a rapid access to the contained informations. In the increasingly ordered lists (or sublists) the searching method we used is the binary search, while in the list *TEV* the search is sequential (since this list generally contains a smaller number of elements than the other lists).

The insertion/deletion of an element in/from the ordered lists is made by means of five basic algorithms, which will be described below.

Algorithm *SEARCH*(L, EL, l, u, i, r) determines the location i of the element EL in the list L. The search is performed between the locations l and u of the list, while r shows whether the element EL is or is not stored in the list L, and has the value:

$$r = \begin{cases} 0, \text{ if } L[i] = EL; \\ 1, \text{ if } L[i] > EL; \\ 2, \text{ if } L[i] < EL. \end{cases} \tag{17}$$

The searching method is the binary search (e.g. [19]), which ensures, for a list containing n elements, the determination of i after at most $\log_2 n$ steps.

Algorithm *MOVER*(L, EL, i, n) inserts a new element (EL) in the list L, of length n, on the location i. The performed operations are:

$$\begin{aligned} &L[q] \rightarrow L[q+1], \; q = n, n-1, \ldots, i; \\ &EL \rightarrow L[i]; \\ &n + 1 \rightarrow n. \end{aligned} \tag{18}$$

Algorithm *MOVEL*(L, i, n) deletes the elements of the location i from the list L (with n elements) as follows:

$$\begin{aligned} &L[q] \rightarrow L[q-1], \; q = i+1, i+2, \ldots, n; \\ &n - 1 \rightarrow n. \end{aligned} \tag{19}$$

Algorithm *INC*(L, LF, i, n) increases by one the values of the fields LF from the elements of the list L (of length n), fields for which their value is greater than or equal to i:

$$L[q]\,.\,LF = \begin{cases} L[q]\,.\,LF, & \text{if } L[q]\,.\,LF < i; \\ L[q]\,.\,LF + 1, & \text{if } L[q]\,.\,LF \geqslant i. \end{cases} \tag{20}$$

Algorithm *DEC*(L, LF, i, n) decreases by one the values of the fields LF from the elements of the list L (of length n), fields for which their value is greater than or equal to i:

$$L[q]\,.\,LF = \begin{cases} L[q]\,.\,LF, & \text{if } L[q]\,.\,LF < i; \\ L[q]\,.\,LF - 1, & \text{if } L[q]\,.\,LF \geqslant i. \end{cases} \tag{21}$$

The manipulation manner (insertion or deletion) of the elements from the precised lists, by means of the above algorithms, will be described below, in a PASCAL-like form.

In what follows, for simplicity, C_i, P_i, F_i, T_i, E_i will respectively be denoted by C, P, F, V, E. The number of elements from the lists TC, TP, TT, TV, TE will respectively be denoted by n_{TC}, n_{TP}, n_{TT}, n_{TV}, n_{TE}, while the current location from these lists will be indicated by i_{TC} and so on.

(a) *Storage of a coefficient.* A coefficient C of the form (11) is stored in the list TC as follows:

```
Procedure AddCoef ($TC, C, n_{TC}, TV, n_{TV}$);
  Begin
    SEARCH ($TC, C, 1, n_{TC}, i_{TC}, r$);
    If $r = 0$
      Then $TC[i_{TC}]\,.\,Nr := TC[i_{TC}]\,.\,Nr + 1$
      Else
        Begin
          If $r = 2$ Then $i_{TC} := i_{TC} + 1$;
          MOVER($TC, C, i_{TC}, n_{TC}$);
          INC ($TV, AdrCoef, i_{TC}, n_{TV}$)
        End;
    End; {AddCoef}
```

(b) *Deletion of a coefficient.* The coefficient C lying on the location i_{TC} will be deleted by using the following procedure:

```
Procedure DelCoef ($TC, n_{TC}, i_{TC}, TV, n_{TV}$);
  Begin
    $TC[i_{TC}]\,.\,Nr := TC[i_{TC}]\,.\,Nr - 1$;
    If $TC[i_{TC}]\,.\,Nr = 0$
      Then
        Begin
          MOVEL ($TC, i_{TC}, n_{TC}$);
          DEC($TV, AdrCoef, i_{TC}, n_{TV}$)
        End;
  End; {DelCoef}
```

The insertion or deletion of monomial (P) or trigonometric (F) parts of a term are performed analogously (observing the corresponding indices and fields), with the procedures *AddMon*, *DelMon*, *AddTrig* and *DelTrig*.

(c) *Storage of a term.* Using the above described (a)-type procedures, the components C, P, F of the term V are stored in the respective lists on the locations i_{TC}, i_{TP}, i_{TT} (or, more rigorously, *AdrCoef*, *AdrMon*, *AdrTrig*), respectively. Using TF from (13) the element V of the list TV is complete. The procedure is the following one:

```
Procedure AddTerm (TV, V, $n_{TV}$, TE, $n_{TE}$);
  Begin
    SEARCH (TV, V, 1, $n_{TV}$, $i_{TV}$, r);
    If $r = 0$
      Then $TV[i_{TV}] . Nr := TV[i_{TV}] . Nr + 1$
      Else
        Begin
          If $r = 2$ Then $i_{TV} := i_{TV} + 1$;
          MOVER (TV, V, $i_{TV}$, $n_{TV}$);
          INC (TE, E, $i_{TV}$, $n_{TE}$)
        End;
  End; {AddTerm}
```

(d) *Deletion of a term.* The term V from the location I_{TV} will be deleted with the following procedure:

```
Procedure DelTerm (TV, V, $n_{TV}$, TC);
  Begin
    $TV[i_{TV}] . Nr := TV[i_{TV}] . Nr - 1$;
    DelCoef(TC, $n_{TC}$, AdrCoef, TV, $n_{TV}$);
    DelMon (TP, $n_{TP}$, AdrMon, TV, $n_{TV}$);
    DelTrig (TT, $n_{TT}$, AdrTrig, TV, $n_{TV}$);
    If $TV[i_{TV}].Nr = 0$
      Then
        Begin
          MOVEL (TV, $i_{TV}$, $n_{TV}$);
          DEC (TE, E, $i_{TV}$, $n_{TE}$)
        End;
  End; {DelTerm}
```

The storage of a new expression is always performed by adding new elements at the end of the lists *TEV* and *TE*, as follows: using the procedure *AddTerm* for each term of the expression, one constitutes a new sublist in *TE* whose limits are stored in the fields *FirstAdr* and *LastAdr* of the new element in *TEV*, together with the name of the expression stored in the field *ExprName*. During the storage of the terms, the similar terms are collected, too, such that, at the end of its storage, the expression has the most concise form. The identification of such terms is made by binary search into the corresponding sublist built upto the respective instant.

The deletion of an expression is achieved in two steps. In the first step, the procedure *DelTerm* is performed for the terms whose addresses lie into the sublist from *TE* corresponding to the expression. In the second step, the element from *TEV* corresponding to the expression is deleted by applying the procedure *MOVEL* (TEV, i_{Ex}, n_{Ex}), where the subscript (Ex) is given with respect to the list *TEV*.

6. **Basic Operations.** All operations (with expressions) which will be described in Section 7 reduce to two basic operations with terms of the respective expressions: addition and multiplication.

The addition of two terms can generate:

— two terms, if they are not similar;

— one term, if they are similar terms, but their coefficients are not opposite;

— no term, if they are opposite.

In the last two cases one performs the collecting like terms.

The multiplication of two terms can generate:

— two terms, if both terms have trigonometric parts (since the product of trigonometric functions transforms into a sum);

— one term, if at least one of the terms has no trigonometric part.

For both basic operations, the most difficult suboperation is the coefficient manipulation; in other words, the rational arithmetic on the computer. The procedures which perform operations with the ordinary fractions use essentially the binary Greatest Common Divisor algorithm (e.g. [20]). All coefficients obtained by rational arithmetic procedures are simplified fractions. The operations involving the transformation of the trigonometric products in sum fulfil the condition mentioned in Section 4, namely: the first nonzero coefficient of the argument of the trigonometric function must be positive (changing accordingly the coefficient sign in the case of the sine function).

7. **Operations with Expressions.** An operation with expressions has the general form:

$$EN = \text{oper}\ (arg_1, arg_2, \ldots, arg_q),\ q \geqslant 2, \tag{20}$$

or:

$$EN = \text{oper}\ (arg_1, arg_2). \tag{21}$$

All implemented operations preserve the form (6) of the Poisson expressions. Since we work with finite expressions, no restriction was formulated as to the number N of terms from (6); the only restrictions are imposed by the limits of the working storage. Also, there are no restrictions as to negligible terms.

With these considerations, we shall describe further the implemented operations with expressions, by using the following symbolic notations:

EN (within or without index) = expression name;

PV = monomial variable name;

TV = trigonometric variable name;

I = natural number ($I \geqslant 1$).

These operations are:

$$EN = \text{add}(EN_1, EN_2, \ldots, EN_q),\ q \geqslant 2, \tag{22}$$

$$EN = \text{sub}\ (EN_1, EN_2), \tag{23}$$

$$EN = \text{mul}\ (EN_1, EN_2, \ldots, EN_q),\ q \geqslant 2, \tag{24}$$

$$EN = \exp\,(EN_1, I), \tag{25}$$
$$EN = \mathrm{dep}\,(EN_1, PV), \tag{26}$$
$$EN = \det\,(EN_1, TV), \tag{27}$$
$$EN = \mathrm{inp}\,(EN_1, PV), \tag{28}$$
$$EN = \mathrm{int}\,(EN_1, TV), \tag{29}$$
$$EN = \mathrm{med}\,(EN_1, TV), \tag{30}$$

or, more explicitly, for each above operation:

$$EN = EN_1 + EN_2 + \ldots + EN_q, \tag{22'}$$
$$EN = EN_1 - EN_2, \tag{23'}$$
$$EN = EN_1 * EN_2 * \ldots * EN_q, \tag{24'}$$
$$EN = (EN1)\hat{}I, \tag{25'}$$
$$EN = \partial(EN_1)/\partial(PV), \tag{26'}$$
$$EN = \partial(EN_1)/\partial(TV), \tag{27'}$$
$$EN = \int EN_1\, d(PV), \tag{28'}$$
$$EN = \int EN_1\, d(TV), \tag{29'}$$
$$EN = (1/2\pi) \int_0^{2\pi} EN_1\, d(TV). \tag{30'}$$

We considered that two differentiation and integration operations are necessary, since the differentiation and integration of a Poisson expression with respect to a monomial or trigonometric variable are esentially different operations.

In its most general form, an expression can be constructed by means of operations (22)—(30) as follows:

$$EN = \mathrm{oper}_1(\ldots) \pm \mathrm{oper}_2(\ldots) \pm \ldots \pm \mathrm{oper}_s(\ldots),\ s \geqslant 1, \tag{31}$$

where oper_l, $1 \leqslant l \leqslant s$, is an operation defined by (22)—(30), of corresponding arguments. We must also mention that even expression names EN are admitted as arguments (EN_1, EN_2 and so on) in the right-hand side of formulae (22)—(30).

8. **Processor Options.** As we showed in Section 4, for a concrete problem, one must define: the monomial and trigonometric variables, the primitive expressions, and the operations with these ones needed by the respective problem. In order to ensure a great flexibility in answering, we foresaw the possibility of interactive request of implemented functions in an order fixed by the user. The functions menu is displayed on the screen and each function is called by pressing a key corresponding uniquely to a given command. These keys are:

I = specification of input file (implicitly keyboard);
O = specification of output file (implicitly CRT);

P = definition of monomial variables;
T = definition of trigonometric variables;
E = definition of primitive expressions;
D = deletion of expressions;
R = construction of new expressions by means of formulae (22)—(30);
S = expression displaying on output file;
M = memory statistics;
N = specification of problem identifier;
Q = exit (stop of execution).

The commands I and O feature the execution environment. The already specified functions are implicitly executed one by one by pressing the corresponding key. In the case when the volume of necessary commands for solving a concrete problem is considerable, it is preferable to leave the interactive mode, specifying (through the function I) the name of an input file containing above described commands of the processor. Each command will be read from the input file, echoed on the screen and then executed. At the end of input file, the keyboard re-acquires the control. On the other hand, if the output results are needed on listing, one may specify (through the command O) an output file (printer or disk file) intended to keep the results.

Another class of commands (P, T, E, D, R) is constituted by the specific commands of the *PSP*. The logical succesion of these commands is (at start) P and/or T, and then E. After E appears, P and/or T can no longer appear. As to E, D and R, they can appear in an arbitrary order.

The commands S, M and N are informational functions. The command S can be used every time when the user wishes the displaying or listing of the stored expressions (at the call instant) on the output file. In this way one can list both intermediary results and the final ones. The command M is purely informative; it allows to the user to know the memory allocation. The command N allows the insertion of a character string (signifying the problem name) in the output file.

Finally, the command Q stops the processor execution and returns the control to the operating system.

9. **Implementation.** The *PSP* was written in *TURBO PASCAL* for *CP/M* and *MS—DOS* machines. The first versions of *PSP* were tested on *CP/M* machines and, after obtaining satisfactory results, transposed on *MS—DOS* machines. This processor will be normally used on *MS—DOS* computers for at least the following reasons:

(a) large working storage;
(b) increased working speed;
(c) use of the rational arithmetics on 32 bits.

Although most of the quoted processors were written in either assembling languages or specialized languages for list processing, we preferred the use of the *PASCAL* language because of the lack of such a specialized list-processing machine. In the implementation of the binary Greatest Common Divisor and *MOVEL*— and *MOVER*—type algorithms, as well, we used the facilities offered by the *TURBO* version of *PASCAL*.

10. **Use of PSP.** Our processor can be used to any kind of problem whose mathematical model resorts to Poisson expressions. In what concern us, this

processor was used to problems belonging to celestial mechanics (perturbed motion in the two-body problem, the two-body problem with variable mass, the restricted three-body problem, all involving the use of averaging-type methods, etc.). As a matter of fact, all processors quoted in this paper are applied to concrete problems of celestial mechanics, in order to both confirm or corroborate classical theories or develop new analytical theories of high order. We mention some such problems: verification of Delaunay's and Hill's theories on Moon's motion [3, 8, 14], computation of general normalized inclination functions [7], computation of f and g functions [26], analytical theories of two Saturn satellites, Enceladus and Dione [17], computation of lunar and solar short-period perturbations of artificial satellites [12], third-order solution of the artificial satellite theory (gravitational perturbations up to the 3-rd power of the 2-nd zonal harmonic coefficient J_2) [18].

REFERENCES

1. Barton, D., Bourne, S. R., Fitch, J. P., *An Algebra System*, Computer J., *13* (1970), No. 1.
2. Barton, D., Bourne, S. R., Horton, J. R., *The Structure of the Cambridge Algebra System*, Computer J., *13* (1970), No. 3.
3. Bourne, S. R., *Literal Expressions for the Co-Ordinates of the Moon. The First Degree Terms*, Celest. Mechanics, *6* (1972), No. 2.
4. Broucke, R., Carthwaite, K., *A Programming System for Analytical Series Expansion on a Computer*, Celest. Mechanics, *1* (1969), No. 2.
5. Brumberg, V. A., *Analytic Algorithms of Celestial Mechanics*, Nauka, Moscow, 1980 (in Russ.).
6. Brumberg, V. A., Isakovich, L. A., *The AMS System for Performing Analytic Operations With Poisson Series on Computer*, Algorithms of Celestial Mechanics, ITA AN SSSR, Leningrade, 1974, No. 1. (in Russ.).
7. Campbell, J. A., *An Exercise in Symbolic Programming: Computation of General Normalized Inclination Functions* Celest. Mechanics, *6* (1972), No. 2.
8. Chapront, J., Mangeney, L., *Application du programme d'opérations sur les séries littérales a la théorie de la Lune*, Colloque sur l'utilisation des calculateurs électroniques pour les développements analytiques en Mécanique Céleste, Prague, 1967.
9. Cherniack, J. R., *A More General System for Poisson Series Manipulation*, Celest. Mechanics, 7 (1973), No. 1.
10. Dasenbrock, R. R., *Algebraic Manipulation by Computer*, Naval Research Laboratory Report No. 7564, 1973.
11. Fateman, R. J., *On the Multiplication of the Poisson Series*, Celest. Mechanics *10* (1974), No. 2.
12. Fisher, D., *Analytic Short-Period Lunar and Solar Perturbations of Artificial Satellites*, Celest. Mechanics *6* (1972), No. 4.
13. Hall, N. M., Cherniack, J. R., *Smithsonian Package for Algebra and Symbolic Mathematics*, SAO Spec. Rep. No. 291, 1969.
14. Henrard, J., *Hill's Problem in Lunar Theory*, Celest. Mechanics, *17* (1978), No. 2.
15. Jefferys, W. H., *A FORTRAN—Based List Processor for Poisson Series*, Celest. Mechanics *2* (1970), No. 4.
16. Jefferys, W. H., *A Precompiler for the Formula Manipulation System TRIGMAN*, Celest. Mechanics *6* (1971), No. 1.
17. Jefferys, W. H., Ries, L. M., *Theory of Enceladus and Dione*, Astron. J. *80* (1975), No. 10.
18. Kinoshita, H., *Third—Order Solution of an Artificial—Satellite Theory*, SAO Spec. Rep. No. 379, 1977.
19. Knuth, D. E., *Tratat de programarea calculatoarelor. Sortare și căutare*, (transl. from English), Ed. Tehnică, București, 1976.

20. Knuth, D. E., *Tratat de programarea calculatoarelor. Algoritmi seminumerici,* (transl. from English), Ed. Tehnică, Bucureşti, 1983.
21. Kovalevsky, J., *Revue de quelques méthodes de programmation des calculs littéraux en mécanique céleste,* Colloque sur l'utilisation des calculateurs électroniques pour les développements analytiques en Mécanique Céleste, Prague, 1967.
22. Le Shack, A. R., Sconzo, P., *FORMAC Language and its Application to Celestial Mechanics,* Astron. J. *73* (1968), No. 3.
23. Mathlab Group, *MACSYMA Reference Manual,* Version Nine, MIT, 1977.
24. Rom, A., *Mechanized Algebraic Operations (MAO),* Celest. Mechanics, *1* (1970), No. 3—4.
25. Rom, A., *Echeloned Series Processor (ESP),* Celest. Mechanics, *3* (1971), No. 3.
26. Sconzo, P., Le Shack, A. R., Tobey, R., *Symbolic Computation of* f *and* g *Series by Computer,* Astron. J., *70* (1965), No. 4.
27. Vasilieva, A. V., *The ALITA System for Performing Analytic Operations with Poisson Series on Computer,* Algorithms of Celestial Mechanics, ITA AN SSSR, Leningrad, 1974, No. 1 (in Russ.).

STUDIA UNIV. BABEŞ—BOLYAI, MATHEMATICA, XXXIV, 3, 1989

BASIC ELEMENTS CONCERNING THE DEFINITION OF SOME PROCESSORS INCORPORATING DYNAMIC EXTENSIBILITY FACILITIES

ILIE PARPUCEA*

Received: June 2, 1989

REZUMAT. — Elemente de bază privind definirea unor procesoare încorporînd facilităţi de extensibilitate dinamică. În prima parte a lucrării se prezintă cîteva aspecte teoretice privind extensibilitatea dinamică a limbajelor de programare. Utilizînd aparatul algebric al ierarhiilor HAS, s-a generalizat noţiunea de tip de dată din limbajul de programare. În partea a doua se fac cîteva referiri legate de implementarea practică a procesoarelor matematice.

1. Algebraic Concept of Programming Language Definition Including Dynamic Extensibility Facilities. The problem of programming language extensibility appears as a need in the programming efficiency growth at the user's disposal. The manners to extend the languages known till now allow a so-called extensibility in statical meaning. This facility emphasizes the fact that the grammar as syntactic specification model is fixed at the soft product implementation, allowing not changes (adaptation or extension) to the user, corresponding to real programming needs.

The programming language extensibility at grammatical level does not constitute a new problem. But the implementation manner of this one on concrete cases is still far from exploiting all offered advantages. This extension mode constitutes an intricate and difficultly applicable problem.

An adequate development of a mathematical device concerning the HAS hierarchy allowed the creation of a formal mechanism for the specification of a concept of abstract calculation system, A, structurally developed in order to be considered as semantic support for a programming language. Accordingly, the objects belonging to such an abstract calculation system are represented as formal expressions. These ones are organized into an algebra of words, W. Such specification model is minutely presented in T. R u s' [4] textbook.

The notion of synamic extensibility impose a "dynamic" character to the structures A and W, allowing the definition of new semantic forms for enriching the collection of semantic forms A. According to these semantic forms, and taking into account their representation, the syntactic correspondent which will enrich the algebra W will be generated.

Between the two algebraic structures, an evaluation morphism:

$$f: W \to A$$

is inductively defined [2].

* *University of Cluj-Napoca, Computing Centre, 3400 Cluj-Napoca, Romania*

In the programming language associated for specification, every element $w \in W$ represents either a program, command, set of commands comprised between *begin* and *end*, or a calculation process. An element w is of the form $w_1 w_2 \dots w_n \circ$, structured on depth levels, where $w_1, w_2, \dots, w_n$ are subwords of the word w, while $\circ$ is an operation for the composition of the words from W. The evaluation process, by means of the morphism f, involves a detailed analysis of w, the separation of the component parts at the subword level upto the free generator level. In other words, an analysis for the identification of all syntactic components is made. After establishing them, one associates to each syntactic component the semantic correspondent from A.

In the algebraic model for the programming language specification, a program is an algebraic expression. That is why the term of program evaluation is more adequate that the term of program parsing. The algebraic expression evaluation algorithm must find out firstly the free generators, which are immediately calculable expressions. After evaluating the free generators, the identification of subexpressions being composed of them imposes to these subexpressions the character of new generators [5].

We shall summarily present hereafter the algebraic model for hierarchized specification of the data types characterizing a programming language [2]. We choose as zero level of the HAS hierarchy the following homogeneous algebra :

$$A_0 = \{D_0,\ \Omega_0,\ F_0 : D_0 \to I\},$$

where :

D_0 = the set of primitive data types of a language ;

Ω_0 = a set of operations defined on the support D_0 ;

F_0 = a function which associates to every element $x \in D_0$ its representation length in standard storage units ;

I = a subset of the natural numbers, which is the set of values of the function F_0 and will constitute the index set for the next level of the hierarchy.

The level 1 of the HAS hierarchy is defined on the basis of the zero level and has the following form :

$$A_1 = \{D_1 = (D_i)_{i \in I},\ (\Sigma S\circ)_{\circ \in \Omega_0},\ F_0 : D_1 \to I,\ F_1\},$$

where :

D_1 = a partition of D_0 into classes of data types, the partitioning criterion being the representation length ;

$\Sigma S\circ$ = the set of operation schemes corresponding to the definition of the new data types.

For $\circ \in \Omega_0$, the function m points out the n-arity : $m(\circ) = n$. If $b = b_1 b_2 \dots \dots b_n \circ$, then one associates to this one an operation scheme $\sigma = (n, \circ, F_0(b_1) F_0(b_2) \dots F_0(b_n) F_0(b))$. When $\circ$ covers the domain Ω_0, while for a fixed $\circ$ it is $(b_1, b_2, \dots, b_n) \in D_0^n$ which varies, the result is the set of all operation schemes which can be defined in the frame of the level 1 of the hierarchy.

F_0 is a function which allows to obtain all operation schemes σ, while F_1 is a function associating to each operation scheme σ a heterogeneous operation scheme specific to the level 1. If $\sigma = (n, \circ, F_0(b_1) F_0(b_2) \dots F_0(b_n) F_0(b))$

is an operation scheme, then $F_1(\sigma)$ is a specific operation in A_1, defined as follows:

$$F_1(\sigma) : D_{F_0(b_1)} \times D_{F_0(b_2)} \times \ldots \times D_{F_0(b_n)} \rightarrow D_{F_0(b)},$$

The domain and co-domain of the operation $F_1(\sigma)$ are inherited from the preceding level; it is its action manner which remains specific in the new level of the hierarchy.

In order to define the level 2 of the hierarchy, one will take into account the data interpretation mode. This level is organized into:

$$A_2 = \{D_2 = (D_{ij})_{i \in N,\, j \in M},\ (\Sigma\, S\, o_{ij}),\ F_2 : D_1 \times M \rightarrow N \times M,\ F_2'\},$$

where:

D_{ij} = the support of the data type of lenght i, interpreted in the mode j;

$\Sigma\, S\, o_{ij}$ = the set of operation schemes corresponding to the definition of the new data types or calculation types, in the frame of the existing ones of length i, interpreted in the mode j;

F_2 = a function which establishes by its values the index set for the family of sets D_2;

$$\forall\ (b, j) \in D_1 \times M\ ;\ F_2(b, j) = (F_0(b), j) \cdot a_{F_0(b)}\ ;$$

F_2' = the symbol of a function which associates to every scheme $\sigma = (n, o, b_1 b_2 \ldots b_n b)$ and to an interpretation mode j a heterogeneous operation scheme specific to the level 2 of the hierarchy.

The support of the heterogeneous algebra A_2 being established, we can reconsider the indexation of the family of sets $(D_{ij})_{i \in N,\, j \in M}$ with a single index i, for simplifying the definition mode of the function F_2'. The re-defined analytical expression for the function F_2 is:

$$\forall\ (b, j) \in D_1 \times M\ ;\ F_2(b, j) = (m(F_0(b) - 1) + j)\ a_{F_0(b)j},$$

where the definition elements are those previously defined.

For a given σ, $F_2'(\sigma)$ will represent a specific operation in A_2 defined as follows:

$$F_2'(\sigma, j) : D_{F_2(b_1, j)} \times D_{F_2(b_2, j)} \times \ldots \times D_{F_2(b_n, j)} \rightarrow D_{F_2(b, j)},$$

where the domain and co-domain are inherited from the preceding level, and the action manner will be specific to the new level of the hierarchy.

The level 3 of the HAS hierarchy will keep its support from the preceding level, being enriched only with a series of operations characteristic to the new defined data types. This level of the hierarchy is defined as follows:

$$A_3 = \{D_3 = (D_i)_{i \in I},\ (\Sigma\, S_i),\ F_3'\},$$

where the components of the heterogeneous algebra have significances similar to those corresponding to the level 2 of the HAS hierarchy. This level allows the definition of data types as RECORD, FILE, SET, known in the high level programming languages as, for instance, PASCAL and C.

The mathematical device of the HAS hierarchy allows the construction of new objects in the frame of a language, structured on different levels of the hierarchy.

2. **Practical Looks Concerning the Definiton of some Mathematical Processors.** On the basis of the theoretical specification (with algebraic contents) of the programming languages, we shall define further down some elements concerning the definition of a mathematical processor. The model we construct does not propose itself to cover all defining and implementation aspects of such a processor.

The zero level of the hierarchy will be constructed with the elements which will be defined further down and will constitute the defining elements of the following algebra:

$$B = (\Delta_B, \Omega_B, G, D),$$

where:

Δ_B = the support of the algebra B; in the present case this support is chosen to be the real number set, denoted by R;

Ω_B = the set of the operations which will be defined further down over the support Δ_B;

G = a function defined on Ω_B with values into the set of the parts (subsets) of Δ_B, which we shall denote $\mathfrak{P}(\Delta_B)$, and which provides the co-domain for every element from Ω_B;

D = a function allowing to obtain the definition domain of an operation, function defined on Ω_B with values into $\mathfrak{P}(\Delta_B)$.

These two functions, G and D, are constructed as new elements are introduced into the base. A first category of operations defined on Δ_B are the simple (primitive) operations. These ones are:

— the addition operation "$+$":

$G(+) = \Delta_B$; $\forall (x, y) \in \Delta_B \times \Delta_B$; $+ : \Delta_B \times \Delta_B \to G(+)$; $+(x, y) = x + y$; $D(+) = \Delta_B$;

— the subtraction operation "$-$":

$G(-) = \Delta_B$; $\forall (x, y) \in \Delta_B \times \Delta_B$; $- : \Delta_B \times \Delta_B \to G(-)$; $-(x, y) = x - y$; $D(-) = \Delta_B$;

— the multiplication operation "$-$":

$G(*) = \Delta_B$; $\forall (x, y) \in \Delta_B \times \Delta_B$; $* : \Delta_F \times \Delta_B \to G(*)$; $*(x, y) = x * y$; $D(*) = \Delta_B$;

- the raise to power operation "$**$":

$G(**) = \Delta_B$; $\forall (x, n) \in \Delta_B \times N$; $** : \Delta_B \times N \to G(**)$; $**(x, n) = x^n$; $D(**) = \Delta_B \setminus \{0\}$;

where N is the natural number set;

— the division operation "$/$":

$G(/) = \Delta_B$; $\forall (x, y) \in \Delta_B \times (\Delta_B \setminus \{0\})$; $/ : \Delta_B \times (\Delta_B \setminus \{0\}) \to G(/)$; $/(x, y) = x/y$; $D(/) = \Delta_B$;

— the extraction of root operation "$\sqrt{\ }$":

$G(\sqrt{\ }) = \Delta_{B+}$; $\forall x \in \Delta_{B+}$; $\sqrt{\ } : \Delta_{B+} \to G(\sqrt{\ })$; $\sqrt{\ }(x) = \sqrt{x}$; $D(\sqrt{\ }) = \Delta_{B+}$;

— the negation operator "$-$":

$$G(-) = \Delta_B ;\ \forall\, x \in \Delta_B ;\ - : \Delta_B \to G(-) ;\ -(x) = -x.$$

The symbol "$-$" is overloaded, but its significance can be deduced from the context. The expressions $x + y$, $x - y$, $x * y$, x^n, x/y and $\sqrt{x}$ are expressions whose evaluation is made in the real number set.

The second category of elements from Ω_B, also defined into the base, consists of the following elementary trigonometric functions:

— the function "sin":

$G(\sin) = [-1, 1]$; $\forall\, x \in \Delta_B$; $\sin : \Delta_B \to G(\sin)$; $\sin(x) = \sin x$;
$D(\sin) = \Delta_B$;

— the function "cos":

$G(\cos) = [-1, 1]$; $\forall\, x \in \Delta_B$; $\cos : \Delta_B \to G(\cos)$; $\cos(x) = \cos x$;
$D(\cos) = \Delta_B$.

The list of definitions for the level zero can be continued, but this is sufficient for examining further down the possibility to define some composed operations, on the basis of those defined in HAS_0.

It is known that every relation $w \in \Omega_B$ in the structure HAS_0 becomes at the next level a relation scheme which specifies by factorization a family of relations. Consider the following algebraic expression $\sqrt{x} * \cos x$, constructed with elements from the base. The analysis shows that its components are $\sqrt{x}$, $\cos x$ and the multiplication operation "$*$" between the two previous ones, which are found as defined in Ω_B. For every element $x \in D(\sqrt{\ }) \cap D(\cos)$ one defines the following composed operation w_1:

$$w_1 : D(\sqrt{\ }) \cap D(\cos) \to G(*, (G(\sqrt{\ }), G(\cos))),$$

according to the law $w_1(x) = \sqrt{x} * \cos x$, while $D(w_1) = D(\sqrt{\ }) \cap D(\cos)$.

Now the function G is defined as follows:

$$G : \Omega_B \times \mathfrak{L}(\Delta_B) \times \mathfrak{L}(\Delta_B) \to \mathfrak{L}(\Delta_B),$$

while for $(w, M_1, M_2) \in \Omega_B \times \mathfrak{L}(\Delta_B) \times \mathfrak{L}(\Delta_B)$ we have $G(w, M_1, M_2) = M_1 \circ M_2$, where the operation $\circ$ is defined as function of w. In the case of the above example, $\circ$ is defined as the $\cap$ operation (intersection).

We defined in this way a new operation w_1, which is in fact a multiplication operator, too, as that defined on the level HAS_0, but having another definition domain. The multiplication operation at the level 1 is in fact a restriction of the multiplication operation corresponding to the zero level. If the above algebraic expression is considered with two variables, $\sqrt{x} * \cos y$, then the new composed operation w_1 will have as definition domain the Cartesian product $D(\sqrt{\ })\times D(\cos)$, while its co-domain remains the same, as well as the analytical expression.

Now we shall define some differentiation operations belonging to the level 1 of the hierarchy. For this purpose, consider the derivative of the trigonometric function $f(x) = \cos x$, namely $f'(x) = -\sin x$. The formal expressions of the derivatives of the elementary functions will belong to the knowledge base of the processor. The differentiation operator transforms the function $f(x) = \cos x$ into a new formal expression for which one has to evaluate the domain and co-domain corresponding to the associated function. In our case, for $f'(x) = -\sin x$, one recognizes the elements already defined in HAS_0.

If we consider the function $f(x) = x^n * \cos x$, whose derivative is:

$$f'(x) = n * x^{n-1} * \cos x - x^n * \sin x,$$

one observes immediately that in the frame of the formal expression of the derivative operations defined in the base can be identified. Step by step one defines firstly a multiplication operatilon in HAS, corresponding to the forme expression $x^n * \sin x$, then a multiplication operation, too, corresponding to the formal expression $n * x^{n-1} * \cos x$, and then, on their basis, a subtraction operation.

Based on the multiplication, addition and raise to power operations, polynomial formal expressions at the level 1 of the hierarchy can be evaluated. A rational function having the following form: $f(x) = \frac{P(x)}{Q(x)}$, where $P(x)$ and $Q(x)$ are polynomial formal expressions recognized in HAS_1, implies the definition of a division operator at the level 2 (HAS_2).

The semantic and syntactic analysis of the expressions to be evaluated imposes, according to the characteristics of the elements from Ω_B, the calculation with formal expressions, too. The numerical or formal evaluation of the expressions can constitute an option for the processor. The formal expressions for the elementary functions and the definition elements of the functions G and D as well will constitute components of a knowledge base open at any time to enrichment by means of the processor itself (self-learning). The enrichment of the base HAS_0 with the corresponding informations, and of the knowledge base, too, will allow to the processor to recognize more complex and various expressions. The self-learning mode of the processor, algebraically defined as an HAS hierarchy, does not constitute a rigid definition, fixed at the implementation of the processor. The construction on hierarchical levels involves by definition a dynamic extensibility of the processor at the user's disposal.

REFERENCES

1. G r ä t z e r, G., *Universal Algebra*, D. van Nostrand Comp. Inc., Princeton, 1968.
2. P a r p u c e a, I., *On the Extensibility of Programming Languages in Dynamic Meaning*, Studia Univ. Babeş-Bolyai, Mathematica, XXXIII (1988), No. 4.
3. P u r d e a, I., P i c, G., *Tratat de algebră modernă*, Vol. 1, Ed. Acad. R.S.R., Bucureşti, 1977.
4. R u s, T., *Mecanisme formale pentru specificarea limbajelor*, Ed. Acad. R.S.R., Bucureşti, 1983
5. R u s, T., *Parsing Languages by Pattern Matching*, IEEE, April, 1988.

STUDIA UNIV. BABEŞ—BOLYAI, MATHEMATICA, XXXIV, 3, 1989

CHARACTERISTIC CONSTANTS OF PROGRAM PRODUCTS

SABIN GORON*

Received : August 3. 1989

REZUMAT. — Constante caracteristice ale rezultatelor unui program. Se propune un model general de creştere a fiabilităţii, unde se ţine seama de toate funcţiile care determină fiabilitatea rezultatului unui program exprimat prin mărimi cu interpretări naturale. În model se iau in considerare atît timpul necesar identificării eorrilor, cit şi timpul efectiv de corectare a lor. Cheltuielile de creştere a fiabilităţii — funcţia de disponibilitate şi funcţia de non-disponiblitate —, care depind de aceste mărimi, se exprimă in diferite forme.

The efficiency coefficient K of the programmer was defined [3] as being the product $K = d(1 - r)$, where d is the probability of detecting an error, while $1 - r$ is the probability of its correct removal. If τ is the available time for adjusting the program product in n stages of running, while τ_0 is the time needed by a programmer of efficiency coefficient K for one running, then $\tau_0 = a/K$, where a is a porportionality coefficient. For $K = 1$ we have $\tau_0 = a = \tau_1$, namely $\tau_0 = \tau_1/K$. Obviously, τ_1 differs from a work (program) to another, being a quantity which features the complexity degree of a work: it is the time needed by the ideal programmer ($K = 1$) for a cycle of adjustment of the respective work. For the n stages of program adjusting, we shall have $n = \tau K/\tau_1$. For the other quantities featuring the reliability one obtains:

$$\lambda = T_0^{-1}(1 - K)^{\tau K/\tau_1}, \tag{1 a}$$

$$R = \exp\left(-\int_0^t \lambda\, dt\right), \tag{1 b}$$

$$T = \int_0^t \lambda t \exp(-\lambda)\, dt, \tag{1 c}$$

where λ is the error rate, R is the probability for the product to be working at a given moment t, while T is the cumulative time of inter-error good functioning (hereafter CTGF).

The time τ_0 needed for a cycle of program adjustment is in its turn composed by the time τ_g for the correction elaboration:

$$\tau_g = (\tau_1/K^2)(1 - (1 - K)/\exp(t)), \tag{2 a}$$

which increases with the time, due to the increase of the difficulty degree

* *University of Cluj-Napoca, Faculty of Economic Sciences, 3400 Cluj-Napoca, Romania*

of the still undetected errors, and the time τ_e of the effective correction:

$$\tau_e = \tau_1/K. \tag{2 b}$$

Cumulating the mean time of good functioning $\bar{T}'_0$ and the times necessary for correction after a fall, we obtain for the physical time:

$$t = n(\bar{T}_0 + \tau_c + \tau_g), \tag{3}$$

or, taking into account τ_0:

$$t = \tau(\exp(t)(K^2\bar{T}_0 + K\tau_1 + \tau_1) + \tau_1(K-1))/(\tau_1 K \exp(t)). \tag{4}$$

Expressing now the quantities with respect to the physical time, one obtains:

$$\tau(t) = (K\tau_1 t \exp(t))/(\exp(t)(K^2\bar{T}_0 + K\tau_1 + \tau_1) + \tau_1(K-1), \tag{5}$$

$$n(t) = K^2 t \exp(t)/(\exp(t)(K^2\bar{T}_0 + K\tau_1 + \tau_1) + \tau_1(K-1)). \tag{6}$$

One also obtains corresponding relations for λ, T_0 and R:

$$\lambda = T_{01}^{-1}(1-K)^{n(t)}, \tag{7}$$

$$T_0 = \int_0^t \lambda(t) \exp(-\lambda(t))\, dt, \tag{8}$$

$$\cdot\ R = \exp\left(-T_{01}^{-1}\int_0^t (1-K)^{n(t)}\, dt\right), \tag{9}$$

where $n(t)$ is considered to be given by (6).

We introduce the notions of restoring intensity, μ, availability function, $A(t)$, and unavailability function, $U(t)$, analogously to these of the products [2], but taking into account the repair time of the program product. So, for the restoring intensity we give the relation:

$$\mu(t) = \tau_0^{-1}(1-(1-K)^{n(t)}), \tag{10}$$

with $n(t)$ given by (6). Since the repair time is $T_r = 1/\mu$, we obtain with (10):

$$T_r = \tau_0/(1-(1-K)^{n(t)}). \tag{11}$$

For $A(t)$ and $U(t)$ we have the relations:

$$A(t) = \mu/(\lambda+\mu) + (\lambda/(\lambda+\mu))\exp(-(\lambda+\mu)t), \tag{12}$$

$$U(t) = (\lambda/(\lambda+\mu))(1-\exp(-(\lambda+\mu)t)), \tag{13}$$

where λ and μ are respetively given by (7) and (10).

For the cost of the program product adjustment we consider relations similar to those given in [3], where c_f and c_v also denote the fixed unitary costs and, respectively, the variable unitary costs, but in which n is given by (6). So, for the total cost C for n running cycles, we have the relation:

$$C = c_f n + c_v E(1 - (1 - K)^n), \tag{14}$$

where E is the number of initially existing errors.

The cost at the n-th repair step, C_n, is given by:

$$C_n = c_f + c_v EK(1 - K)^{n-1}. \tag{15}$$

For the Asher—Feingold model of reliability growth [1]:

$$dT_0(t)/dt = K_2(K_3 - T_0(t)), \tag{16}$$

taking into account the interpretation of the constant K_3 considered in [4] and the new constant τ_1 (defined at the beginning of this paper), one obtains the relation:

$$K_3 = \lim_{t \to \infty} T(t) = (1 - K)^{-\tau K/\tau_1}. \tag{17}$$

Integrating (16) with K_3 given by (17), one obtains for the CTGF:

$$T_0(t) = (1 - K)^{-\tau K/\tau_1}(1 - K_1 \exp(-K_2 t)). \tag{18}$$

For the reliability slow growth models, modified in [4], one obtains analogously:

$$T_0(t) = (\exp(K_2(t + 1)\exp(-t) + K_2) - 1)(1 - K)^{-\tau K/\tau_1}/(\exp(K_2) - 1), \tag{19}$$

for the first model, and:

$$T_0(t) = (\exp(K_2(1 - \exp(t))) - 1)(1 - K)^{-\tau K/\tau_1}/(\exp(K_2) - 1), \tag{20}$$

for the second model.

For the Lloyd—Lipow and Argef models, modified in [4], one obtains new relations, respectively:

$$T_0(t) = (1 - K)^{-\tau K/\tau_1} - K_2/t, \tag{21}$$

$$T_0(t) = (1 - K)^{-\tau K/\tau_1} - \exp(-K_2/t). \tag{22}$$

The model proposed by us expresses the reliability growth in time and does not introduce arbitrary constants:

$$dT_0(t)/dt = T_0^{-1}(1 - K)^{\tau K/\tau_1}\, t \exp(-T_0^{-1}(1 - K)^{\tau K/\tau_1}), \tag{23}$$

where T_0 is the CTGF at the beginning of the program product adjustment.

Taking into account the above results, we can formulate some conclusions. The proposed general model of reliability growth expresses all functions which determine, in a wya or another, the reliability of the program product by quantities with natural interpretation, according to the structures which determine (by projection, construction and adjustment/repair) the program product:

K = programmer s efficiency coefficient;
τ = working time for the adjustment ot the program product;
τ_0 = time need lor the running of a cycle of adjustment (eror ieentification, conection, test);
n = number of adjustment cycles;
τ_1 = time necessary to the ideal programmer ($K = 1$) for the running of a repair cycle; it features uniquely the complexity degree of a program product.

The model considers both the time needed by the error identification and the effective correction time, both functions of K and τ.

The reliability growth costs, the availability function and the unavailability one, expressed under different forms, depend on the same quantities.

In the case of the work on the program product, one takes into account the mean time of good functioning of this one at the beginning of the adjustment.

REFERENCES

1. Asher, H., Feingold, H., *Reiparable System Reliability*, Marcel Dekker, Inc., New York, Basel, 1984.
2. Baron, T. *et al.*, *Calitate şi fiabilitate*, Vol. 1, Ed. Tehnică, Bucureşti, 1988.
3. Goron, S., *The Cost of Informatics Works and Programmer's Efficiency*, in *Method, Models and Techniques in Physics and Related Fields*, University of Cluj-Napoca, Romania, 1987, p. 129–134.
4. Goron, S., *The Cost of Program Reliability*, communication held at the V-th International Symposium on Numerical Methods in Engineering, Lausanne, Switzerland, 1989.

STUDIA UNIV. BABEŞ—BOLYAI, MATHEMATICA, XXXIV, 3, 1989

ON AN IMPLEMENTATION OF COMMON LISP FOR 16 BIT MINICOMPUTERS

M. E. BALÁZS* and D. NEAMŢU*

Received: July 4, 1989

REZUMAT.— Asupra unei implementări de COMMON LISP pentru minicalculatoare de 16 biţi. În lucrare sint analizate unele aspecte ale implementării de COMMON LISP, pentru minicalculatoarele de 16 biţi. Se fac referiri la cerinţele implementării, la structura sistemului, precum şi la implementarea ca atare, în final dindu-se citeva concluzii utile.

The last years brought along a considerable increase in the number of *AI* applications. This tendency calls for developping new tools suitable for this area of *CS*. Since 16 bit minicomputers are still largely used, it seems sound not to neglect this type of hardware. The obtainable configurations (storage and peripherals) also justify this idea.

The present paper discusses some features of an implementation of *COMMON LISP* for computers of this class. The coice of *COMMON LISP* as a developping tool for *AI* applications is due to its specification as a potential *LISP* standard.

1. **Implementation requirements.** An implementation should meet requirements that stem either from the language specification or the applications to come.

First of all a quick addressing of a large memory is needed, in order to make possible the processing of a great amount of object representations.

The typical processings which imply a great frequency of function calls ask for a proper mechanism to handle these operations.

CL has been defined as an environment for developping and running applications. As such it includes predefined functions like the compiler, the editor and several debugging tools. From the user's point of view, these functions must behave exactly as the ones he writes. On the other hand, the system must support compiled and interpretted functions as well. This calls for compatibility between the representations of the function objects.

Implementations also should meet certain requirements on the dimensions and domains of various object types. Thus the magnitude of integer values to be supported may only depend on the size of the available storage, the maximal rank of an array must not be less then seven; each function should allow at least fifty parameters, and so on.

1.1. *System structure.* Since *LISP* is a functional programming language, any *LISP* system must contain an object space and evaluating mechanism.

* *Research Institute for Computer Techniques and Infornatics (I.T.C.I.), 3400 Cluj-Napoca, Romania*

The object space is made up as a collection of object representations. It is partitioned into a set of object classes called types. A main characteristic of the object space is its dynamic changing. Initially it is composed of a certain set of objects (including standard symbols, functions, etc.) and a set of predefined types. During a working session, new objects are added and new types may be defined.

In comparison with other dialects, *COMMON LISP* provides a great variety of predefined object types. These types are organized into a network of subtypes and supertypes. New types may be added to this network, which are to be treated exactly as the standard ones.

Predefined types are divided into atoms and lists. The atomic types include symbols, numbers, characters, strings, arrays, hash-tables, packages, pathnames, streams and random states.

The basic means to refer objects are the variables. A *COMMON LISP* variable is a binary relation between symbols and objects. There are various kinds of variables (which from now on will also be referred as bindings). Thus there are global, special, lexical and "frozen" variables. The description of these kinds of variables will be given later. At any moment there may be more variables having the same symbol component, but of different kinds. Referring an object by a symbol naming it is done by selecting a variable depending on the referring context.

The evaluator is the mechanism which interprets objects belonging to certain types according to some well-described rules. These objects are called evaluable, and they might be: numbers, characters, strings, bit vectors, symbols, calls.

Objects of the first four types evaluate to themselves. These are called self-evaluable objects.

When a symbol is evaluated, the variable is identified according to the context. The result of the evaluation is the value component of this variable.

By call we mean either applying a function or lambda expression to certain parameters, or the expansion of a macro-call.

Evaluating function calls is the main job of the evaluator.

A function is an boject as any other object in the object space. A function object includes components used for parameter passing, the function body and a documentation string. When evaluating a function call, its actual parameters are first evaluated. The results obtained are bound to the formal parameters. This way a new context is bulit, in which the body will be evaluated. Evaluating the body means running the code for compiled functions and evaluationg in sequence each form of the body list for interpretted ones.

2. **The implementation.** 2.1. *System structure.* The storage used by the implementation is divided into three areas. These hold the kernel, the stack and the object space respectively.

The kernel contains the evaluator and managing mechanisms for the other two areas. The laters are mainly constructors and accessors for different kinds of objects.

The stack is used for solving the calls as well as a scratchpad for various processings. It is organized into two levels. The first one corresponds to the bottom of the stack and if it is bulit, it is kept on a mass storage device. The other level holds that part of the stack which can be represented in the main storage. Passing a part of the stack's content from one level to the other is performed asyncronously with the processing when over/undergoing a certain limit.

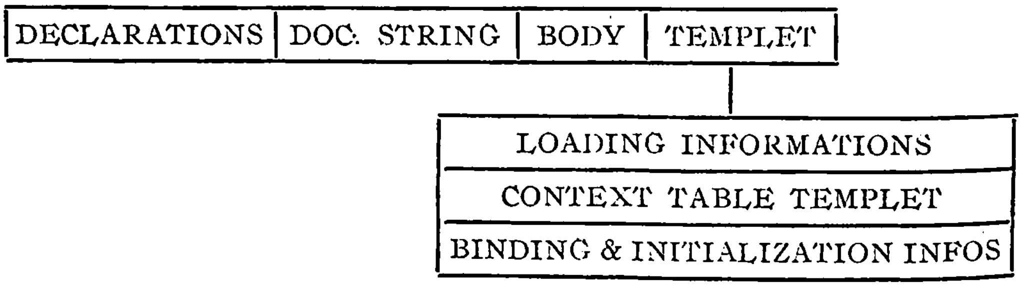
This implementation of the stack makes its size to be limited only by the available capacity of the mass storage. Equally, the time for pushing or popping the stack is comparable to that for stacks entirely kept in the main storage. Also it leads to a memory saving which is very important for computers in the considered class.

The object space is divided into fixed lenght pages. The object space management dynamically extends the representation of the object space to the size of the available part of the main storage. If there is no more space in the main storage, the object space is extended to a disk by implementing a virtual memory mechanism. This solution for the object space management allows the system to run on minicomputers having different main storage capacities.

The discipline for moving pages between the main storage and the disk can be altered either by fixing some of them into the main storage or by changing the swapping algorithm. These facilities allow a kind of tailoring of the system for the applications to be run.

2.2. *The calling mechanism.* The nature of *LISP* makes function calls to be very frequent. Due to the complexity of the parameters passing mechanism, each call requires a relatively great number of operations to be executed. For these reasons the main concern was to devise a fast calling mechanism.

A function call means binding the parameters, evaluating the body and unbinding the parameters. To meet the requirements of the specification, the following representation was chosen for function objects.



The context table templet looks like a symbol table which holds entries for the locally referenceable bindings. Such an entry may have one of the following structures:

$\boxed{\text{SYMBOL} \mid \text{VALUE}}$ — for lexical variables;

$\boxed{\text{SYMBOL} \mid \text{VALUE} \mid \text{ADDRESS}}$ — for references to special variables;

$\boxed{\text{SYMBOL} \mid \text{VALUE} \mid \text{CHAINING}}$ — for definitions of special variables;

$\boxed{\text{SYMBOL} \mid \text{VALUE} \mid \text{VALUE ADDRESS}}$ — for "frozen" variables.

To each of these entry types a field may be added to hold type information.

When calling a function, the *TEMPLET* is pushed onto the stack, running and chaining informations being substituted for the loading ones.

Before the binding, the forms passed as actual parameters are evaluated in the calling context. During the binding process the context table entries are filled with the appropiate values. These are obtained either by the above mentioned evaluations or by evaluating the corresponding initialization forms.

After the binding is completed, the initialization informations are popped from the stack. Their presence on the stack during the binding is necessary for avoiding a great number of accesses to the page containing the function object.

Now the evaluation of the body follows. Being treated as any other object, a function may be built any place in the object space. To make this possible, compiled function bodies are relocated before loading.

During the evaluation of the body all the references to variables are made in the new context.

For lexical variables, accessing the value means accessing the value field of the corresponding entry. Since this kind of variable is the most used, this solution considerably improves execution speed.

In our example, symbol *d* in function *f4* references a lexical binding.

For special variables the value is accessed by indirection, using the address field of the corresponding entry of the context table. This field is filled when pushing the templet onto the stack, with the address of the value field of entry from the last special definition of the same symbol.

Because is possible for a symbol to have more special definitions along the same calling path, the entries corresponding to these definitions are chained.

These solutions for implementing special variables, besides bringing an access speed comparable to that for lexical ones, also renders the shadowing and unbinding mechanisms simple.

When a special reference has no corresponding special definition, it resumes to the global binding if it exists.

In the given example, symbol *c* references special bindings.

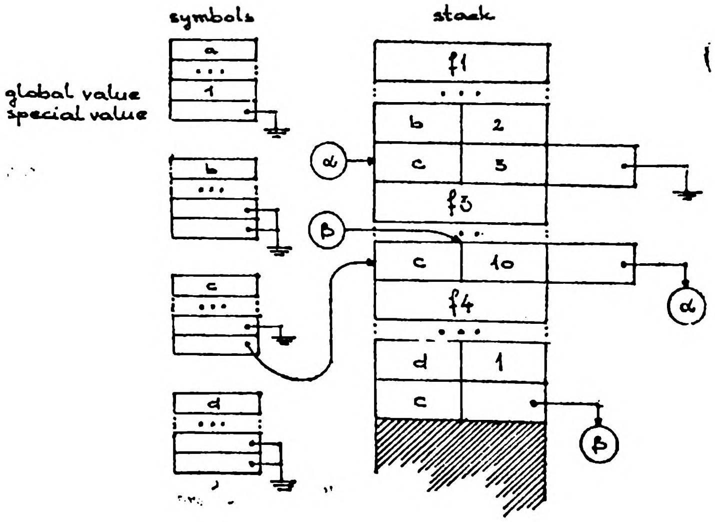
Specifications require for functions defined in a non null context to store (freeze) the local bindings of that context. For "frozen" bindings the value field in the templet of function is used as a local memory. Reading acces to such a binding means accessing the corresponding value field in the context table on the stack, while changing its value requires a modification in the templet as well.

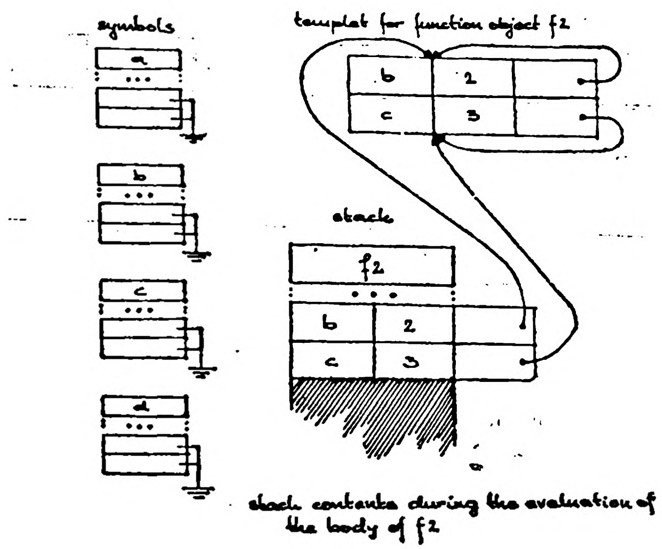
Bindings of symbols *b* and *c* have been frozen when defining function *f*2 in our example.

Example:

```
: (defvar a1)
A
: (defun f1 (b c)
     (declare (special c))
     (defun f2) ()
          (list b c))
     (f3 10))
F1
: (defun f3 (c)
     (declare (special c))
     (list (f4 a) c))
F3
: (defun f4 (d)
     (declare (special c))
     (list c d))
F4
: (f1 2 3)
((10 1) 10)
: (f2)
(2 3)
```

Another advantage of the described solutions is that unbinding is only necessary for special bindings, which is simply solved.




symbols
a
b
c
d
templat for function object f2
b
2
c
3
stack
f2
b
2
c
3
stack contents during the evaluation of the body of f2

3. **Miscellaneous features.** In this section two solutions which come to improve performances are presented.

As mentioned above, the object space is divided into pages. At any moment the number of pages kept in the main storage depends on the available memory size. Objects represented in a page from the main storage are accessed through the memory management unit by mapping a window over that page. In the other case, mapping must be preceded by page swapping. Obviously much time is spent doing these operations.

In the folowings, a solution for reducing this time is given.

Since type checking is a rather frequent operation, keeping the type of every object in its representation would require at least one mapping for every such operation. To avoid this, a referencing object called pointer, which also contains the object type, has been designed. It has the folowing structure:

object type	page number	offset in page

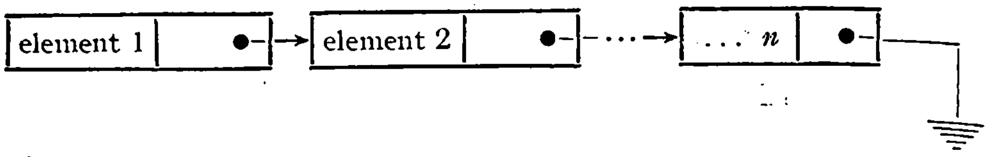
The type field indicates either one of the predefined types or a user defined type. The other two fields are used in a natural way to access the object.

Specific to *LISP* is the great number of operations on lists. Usual list representations and operations require much space and long time. To improve list processing, the refered implementation includes the possibility of compact (vector—like) representation of lists. Thus a list may be represented in one of the following forms:

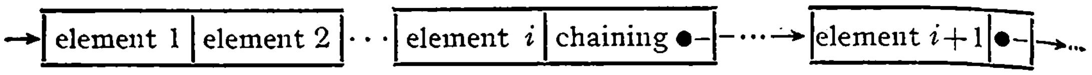
1. Compact list:

element 1	element 2	...	element *n*	list end

2. List of consens:



3. Mixed lists:



Using compact lists drastically reduces space requirements for list representation. The time needed to step trough a compact list is also much shorter then that for consens, because of the elimination of the repeated mappings.

In the presented implementation a compact list is constructed whenever it is possible. Thus the function read produces compact lists. List copying operation produce compact list as well. This is not possible only when the vector used to represent the list would be larger than the size of a page.

Mixed lists may be produced by functions which operate by altering list structures.

4. **Conclusions.** The above described features are parts of an implementation under *MIX* (*RSX*—11*M* compatible) operating system. The implementation language is *C*. This choice was made aiming the implementation for other computers and operating systems.

Because of limited space, only a few ideas about the implementation were presented. Any further information can be obtained from the authors.

REFERENCES

G. L. Steele jr., *COMMON LISP — The Language*, Digital Press, 1984.

STUDIA UNIV. BABEŞ—BOLYAI, MATHEMATICA, XXXIV, 3, 1989

PARALLEL EXECUTABLE SEQUENCES IN SERIAL PROGRAMS

F. BOIAN*, M. FRENŢIU* and Z. KÁSA*

Received: June 2, 1989

REZUMAT. — **Secvenţe executabile în paralel în programe seriale.** Lucrarea se ocupă de posibilitatea detectării secvenţelor de instrucţiuni în programele seriale care se pot executa în paralel.

0. **Introduction.** The development of high speed parallel computers makes it necessary to change the well-known methods to solve complex problems or to develop new algorithms for these computers.

At present many algorithms are written in sequential programming languages like Fortran. It is possible to construct a compiler to detect the parts of the algorithm that can be executed in parallel and to generate code for parallel computers from serial languages.

The present paper deals with a mechanism which detects the parts of a serial algorithm that can be executed in parallel.

1. **The usage of AFT sequences to detect the statements executable in parallel.** For a program scheme S it is important to detect those parts of S that can be executed in parallel, during the compilation process. The AFT sequences offer such a possibility.

The definitions of AFT sequences and of static words are those given in [1], [2]. Informally an AFT sequence is a maximal sequence of assignment type statements (without tests) followed (maybe) by a test statement. A static word is the sequence of the statements of the program in their static order.

Let $a_1 a_2 \ldots a_n$ be an AFT sequence. We denote by $R(a_i)$ the set of variables which are inputs for a_i, and by $W(a_i)$ the set of variables that receive values in the statement a_i.

There are three possibilities to execute the statements a_i and a_j $(i < j)$:

a) the serial execution;

b) the execution of a_i in the same time with a_j (i.e. after $a_{i+1}, a_{i+2}, \ldots$ $\ldots, a_{j-1}$);

c) the execution of a_j in the same time with a_i (i.e. before $a_{i+1}, a_{i+2}, \ldots$ $\ldots, a_{j-1}$).

DEFINITION 1. The statements a_i and a_j are *executable in parallel* if the execution b) or the execution c) gives the same result as the execution a).

We are interested to find if a_i and a_j may be executed in parallel. When $j = i + 1$ (i.e. the statements are consecutive) a condition is given in [2]. Suppose now that $i < j$. The statement a_j may be executed in the same time with a_i, i.e. before $a_{i+1}, a_{i+2}, \ldots, a_{j-1}$ if the values of the variables of $R(a_j)$

*University of Cluj-Napoca, Faculty of Mathematics and Physics, 3400 Cluj-Napoca, Romania

are not changed by these statements and the variables of $W(a_i)$ do not appear in these statements. It results the following.

PROPOSITION 1. *The statement a_j may be simultaneously executed with a_i (before the execution of $a_{i+1}, a_{i+2}, \ldots, a_{j-1}$) iff the following conditions hold:*

$$R(a_j) \cap W(a_k) = \emptyset \tag{1}$$
$$W(a_j) \cap R(a_k) = \emptyset \tag{2}$$
$$W(a_j) \cap W(a_k) = \emptyset \tag{3}$$

for all $k = i, i + 1, \ldots, j - 1$.

Example 1. Let us consider the AFT: $a_1a_2a_3a_4$, where

$$\begin{aligned} a_1 &\text{ is } x = f_1(x) \\ a_2 &\text{ is } y = f_2(x, y) \\ a_3 &\text{ is } z = f_3(t) \\ a_4 &\text{ is } u = f_4(y). \end{aligned}$$

The statement a_3 may be simultaneously executed with a_1, but a_4 may not, since a_2 changes the value of y, needed in a_4, i.e. $R(a_4) \cap W(a_2) = \{y\} \neq \emptyset$.

The statement a_i may be executed after the execution of $a_{i+1}, a_{i+2}, \ldots, a_{j-1}$, simultaneously with a_j, if the variables of a_i are not changed by a_k, for $k = i + 1, i + 2, \ldots, j - 1$, and the variables of $W(a_i)$ do not appear in the statements a_k, $k = i + 1, i + 2, \ldots, j$, i.e. we have the following

PROPOSITION 2. *The statement a_i may be simultaneously executed with a_j (after the execution of $a_{i+1}, a_{i+2}, \ldots, a_{j-1}$) iff the following conditions hold:*

$$R(a_i) \cap W(a_k) = \emptyset \tag{4}$$
$$W(a_i) \cap R(a_k) = \emptyset \tag{5}$$
$$W(a_i) \cap W(a_k) = \emptyset \tag{6}$$

for all $k = i + 1, i + 2, \ldots, j$.

In the AFT of example 1, a_1 cannot be simultaneously executed with a_3 (after the execution of a_2), since a_2 needs the new value of x, i.e. $W(a_1) \cap$ $\cap R(a_2) = \{x\}$.

2. **Precedence relations between statements.** Let A be the set of the statements of an AFT (or program). We define the precedence relation $\pi \subset A \times A$, where $a \pi b$ means that a must be executed before b. Let a_i and a_j be two statements. $a_i \pi a_j$ if and only if the following conditions hold:

$$i < j \tag{7}$$
$$R(a_i) \cap W(a_j) \neq \emptyset \text{ or } R(a_j) \cap W(a_i) \neq \emptyset \text{ or } W(a_i) \cap W(a_j) \neq \emptyset \tag{8}$$

Let us define the relation $\rho \subset A \times A$ with $a \rho b$ iff $a \pi b$ or $b \pi a$.

To test if two statements are in the relation ρ we can use boolean matrices. Let $v_1, v_2, \ldots, v_m$ be the variables of the AFT (or program), and $a_1a_2 \ldots a_n$

the corresponding static word. We define the matrices:

$$R = (r_{ij})_{i=\overline{1,n};\, j=\overline{1,m}}$$

with

$$r_{ij} = \begin{cases} 1, \text{ if } v_j \in R(a_i), \\ 0, \text{ otherwise,} \end{cases}$$

and

$$W = (w_{ij})_{i=\overline{1,n};\, j=\overline{1,m}}$$

with

$$w_{ij} = \begin{cases} 1, & \text{if } v_j \in W(a_i), \\ 0, & \text{otherwise.} \end{cases}$$

Let W^t be the transposed matrix of W, and let S be the matrix

$$S = (s_{ij})_{i=\overline{1,n};\, j=\overline{1,m}}$$

defined by

$$S = R \cdot W^t + W \cdot R^t + W \cdot W^t$$

(The operator $+$ is the boolean addition: $0+0=0$, $0+1=1+0=1+1=1$). Then $a_i \rho a_j$ iff $s_{ij} = 1$, and $a_i \pi a_j$ iff $i < j$ and $a_i \rho a_j$.

The graph of the relation π is called the precedence graph. If two statements a_i and a_j, in this order, are on the same path in this graph it means that a_i must be executed before a_j, i.e. they are not executable in parallel. It results the following

PROPOSITION 3. *Two statements of a program (or AFT) are executable in parallel iff they aren't on the same path in the corresponding precedence graph.*

The transitive closure of S is denoted by $S^* = (s^*_{ij})_{i=1,n;\, j=1,n}$ If $s^*_{ij} = 0$ then the statements a_i and a_j may be executed in parallel.

Example 2. Let us consider the AFT: $a_1 a_2 a_3 a_4 a_5$ where

$$\begin{aligned} a_1 &\text{ is}: x_1 = f_1(x_2) \\ a_2 &\text{ is}: x_3 = f_2(x_1, x_6) \\ a_3 &\text{ is}: x_4 = f_3(x_4, x_5) \\ a_4 &\text{ is}: x_6 = f_4(x_7) \\ a_5 &\text{ is}: x_4 = f_5(x_4, x_8) \end{aligned}$$

Then we have

$$R = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

and

$$W = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

hence

$$R \cdot W^t = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix} \quad W \cdot R^t = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix}$$

and

$$W \cdot W^t = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix}$$

It results

$$S = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix}$$

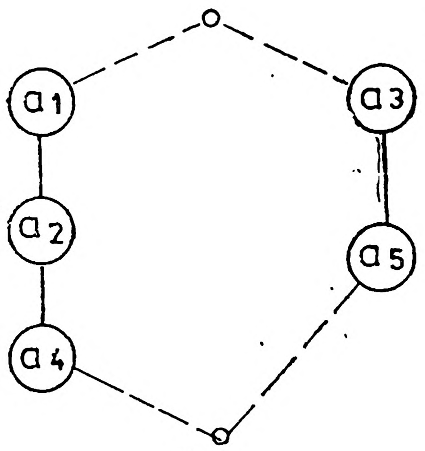


Fig. 1.

The corresponding precedence graph is given in fig. 1.

After the transitive closure we obtain

$$S^* = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix}$$

Then the following pairs of statements may be executed in parallel:

$$(a_1, a_3),\ (a_1, a_5),\ (a_2, a_3),$$
$$(a_2, a_5),\ (a_3, a_4),\ (a_4, a_5).$$

3. Binary trees and expressions.

Let e be a correct arithmetic expression, with the operands

$$a_1, a_2, \ldots, a_q$$

and some binary operators from the set

$$OP = \{o_1, o_2, \ldots o_p\}.$$

Suppose that the operators have the same priority, and the order of executions is established using only the parantheses "(" and ")".

It is known [3] that a binary tree, having the operands $a_1, a_2, \ldots, a_q$ as leaves and the internal nodes are from the set OP, can be attached to the expression e. In this binary tree, we denote by $S(e)$ the set of its proper subtrees. Let x and y be two subexpressions. The subexpression x is included into y iff we have

$$S(x) \subset S(y)$$

Analogously, x and y are disjuncts iff

$$S(x) \cap S(y) = \emptyset$$

In the following we can identify the arithmetic expressions with their binary trees. Fro example, the following expression denoted by e_1:

$$((((((((a + b) + c) + d) + e) + f) \cdot ((g + h) + (i + (j + k)))) \cdot$$
$$\cdot (l + (m + (n + (o + (p + (q + r)))))) \cdot (s + (t + u))) \cdot (v + (w + (x + (y + z)))))$$

has the tree given in fig. 2. For simplicity, we denote:

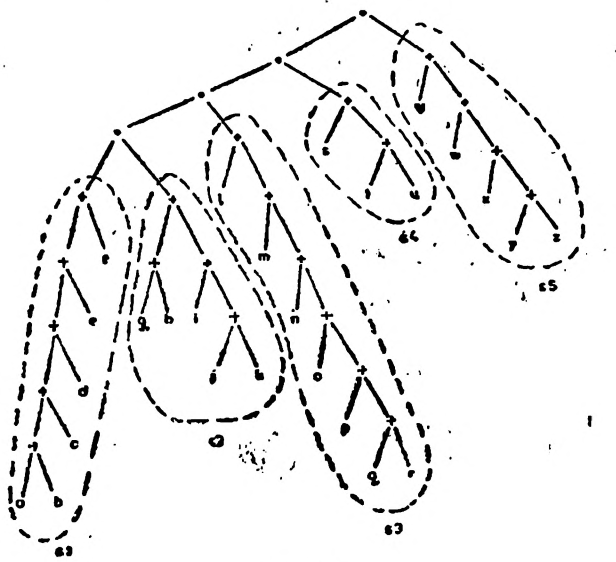


Fig. 2.

$$s_1 = (((((a + b) + c) + d) + e) + f)$$
$$s_2 = ((g + h) + (i + (j + k)))$$
$$s_3 = (l + (m + (n + (o + (p + (q + r))))))$$
$$s_4 = (s + (t + u))$$
$$s_5 = (v + (w + (x + (y + z))))$$

With these notations the expression e_1 is

$$e_1 = ((((s_1 \cdot s_2) \cdot s_3) \cdot s_4) \cdot s_5)$$

Suppose that for computing the value of the expression e_1 we have a computer system with unlimited number of processors. It is obvious that both expressions x and y of $S(e_1)$, with $S(x) \cap S(y) = \emptyset$, may be computed in the same time, i.e. they may be computed in parallel. Thus, the total computing time of the evaluation of the value of e_1 is reduced.

For example, for the expression s_2 two procesors are needed, and for s_1, s_3, s_4 and s_5 only a single one.

In the following we will ignore the operating system overhead time, used for switching between processors and processes. Also, we consider that for each $\circ \in OP$, the evaluation time is independent of the values of its operands. Let T be the function

$$T : OP \to R_+$$

where $T(\circ)$ is the evaluation time of $\circ$, for each $\circ \in OP$.

At last, suppose that the time for reading one value from a memory location and the time needed for transmitting it to a processor is a constant value M, independent of the location and of the processor.

For each $s \in S(e)$, we denote by $L(s)$ and $R(s)$ the left and right subtree, respectively.

DEFINITION 2. The *depth* of $s \in S(e)$, denoted by $H(s)$, is defined as follows:

$$H(s) = \begin{cases} M, & \text{if } s \text{ is a leaf (i.e.an operand)} \\ T(\circ) + \max\{H(L(s)),\ H(R(s))\}, & \text{if } s = (L(s) \circ R(s)) \end{cases}$$

The depth $H(e)$ is the necessary time for the evaluation of "e", if as many as necessary number of processors exist.

Example 3. Suppose that $M = 1$, $T(+) = 1$ and $T(.) = 1$. Then, for the expression e_1 given in fig. 2 we have:

$$H(s_1) = 6, \quad H(s_2) = 4, \quad H(s_3) = 7,$$
$$H(s_4) = 3, \quad H(s_5) = 5, \quad H(e_1) = 10.$$

4. Optimal trees for associative and commutative operators. In the following, we suppose that each operator $\circ$ of OP is associative and commutative. We denote by $[e]$ the set of expressions derived from e using these two properties. Of course, $[e]$ is a finite set.

Our purpose is to find an expression e^* of $[e]$ such that:

$$H(e^*) = \min\{H(a) \mid a \in [e]\}$$

Suppose that $e = s_1 \circ s_2 \circ \ldots \circ s_n$, $n > 1$, and any root of $S(s_1)$, $S(s_2)$, ..., $S(s_n)$ is not marked by the operator $\circ$.

We denote by $BT(\circ, s_1, s_2, \ldots, s_n)$ the set of the binary trees having $s_1, s_2, \ldots, s_n$ as leaves and having $n - 1$ interval nodes marked by the operator $\circ$. Of course, each set BT is a finite set. Let

$$H^*(\circ, s_1, s_2, \ldots, s_n) = \min\{H(a) \mid a \in BT(\circ, s_1, s_2, \ldots, s_n)\}$$

and let

$$OT(o, s_1, \ldots, s_n) = \{a \in BT(o, s_1, \ldots, s_n) \mid H(a) = H^*(o, s_1, \ldots, s_n)\}$$

be the set of optimal trees.

Due to associativity and commutativity, we may suppose that

$$H(s_1) \leqslant H(s_2) \leqslant \ldots \leqslant H(s_n).$$

This hypothesis holds in the following.

It is clear that for each $1 \leqslant i < j \leqslant n$, we have

$$H((s_i \circ s_j)) = T(o) + H(s_j).$$

Also, for each $1 \leqslant i < j \leqslant k < n$ we have

$$H((s_1 \circ s_2)) \leqslant H((s_i \circ s_j)) \leqslant H((s_k \circ s_n))$$

LEMMA 1. *If* $H(s_1) \leqslant H(s_2) \leqslant \ldots \leqslant H(s_i)$,

and $H(s_i) = H(s_{i+1}) = \ldots = H(s_{j-1}) = H(s_j)$,

and $H(s_j) \leqslant H(s_{j+1}) \leqslant \ldots \leqslant H(s_n)$,

and $a \in OT(o, s_1, \ldots, s_n)$, *and* $(s_l \circ s_r) \in S(a)$ *for* $i < l \leqslant j$, *then for each* $k : i \leqslant k \leqslant j$, *there exists* $b \in OT(o, s_1, \ldots, s_n)$ *such that* $(s_k \circ s_r)$ *is in* $S(b)$.

Informally, in each optimal tree a, two elements of $S(a)$ can be changed if they have the same depth. The proof of this lemma is obvious.

LEMMA 2. *If* $H(s_1) \leqslant H(s_2) \leqslant \ldots \leqslant H(s_k)$,

and $H(s_k) < H(s_{k+1})$,

and $H(s_{k+1}) \leqslant H(s_{k+2}) \leqslant \ldots \leqslant H(s_n)$,

and $2 \leqslant k < n$

then it exists $a \in OT(o, s_1, \ldots, s_n)$ *and there exist* i and j, $1 \leqslant i < j \leqslant k$, *such that* $(s_i \circ s_j) \in S(a)$.

Proof. Let b be an optimal tree.

Case 1: There exists $(s_i \circ s_j) \in S(b)$ such that $i < j \leqslant k$. Then lemma is true.

Case 2: For each $(s_i \circ s_r)$ of $S(b)$, we have $i < k$ or $r < k$. Due to commutativity, we may suppose that $i < r$.

Case 2a: For each $(s_i \circ s_r)$, we have $i < k$ and $r < k$. Because $i < k \geqslant 2$, there exits z, t, such that $(s_1 \circ z)$ and $(s_2 \circ t) \in S(b)$. If $H(z) \leqslant H(t)$, then in b we can change $(s_1 \circ z)$ and $(s_2 \circ t)$ with $(s_1 \circ s_2)$ and $(z \circ t)$, and lemma is proved. If $H(z) < H(t)$ we can change $(s_1 \circ z)$ and $(s_2 \circ t)$ with $(t \circ z)$ and $(s_2 \circ s_1)$. Due to commutativity $(s_2 \circ s_1)$ can be changed with $(s_1 \circ s_2)$ and lemma is proved.

Case 2b: It exists $(s_i \circ s_r) \in S(b)$ with $i \leqslant k$ and $r < k$. Because $k \geqslant 2$, it exists z and $(s_j \circ z) \in S(b)$, with $j \leqslant k$. If $H(s_r) \leqslant H(z)$, then $(s_i \circ s_r)$ and $(s_j \circ z)$ can be changed with $(s_i \circ s_j)$ and $(s_r \circ z)$, and lemma is proved. If $H(s_r) < H(z)$, we change $(s_i \circ s_r)$ and $(s_j \circ z)$ with $(z \circ s_r)$ and $(s_j \circ s_i)$. If $i > j$

then lemma is proved. If $i < j$, then due to commutativity, $(s_j \circ s_i)$ can be changed into $(s_i \circ s_j)$, and the lemma is proved, too.

In all cases, all changes keep the depth in the optimal tree.

LEMMA 3. *If* $H(s_1) \leqslant H(s_2) \leqslant \ldots \leqslant H(s_k)$,

$$\textit{and } H(s_k) < H(s_{k+1}),$$
$$\textit{and } H(s_{k+1}) \leqslant H(s_{k+2}) \leqslant \ldots \leqslant H(s_n),$$
$$\textit{and } 2 \leqslant k < n,$$

then there exists $a \in OT(\circ, s_1, \ldots, s_n)$ *and there exists* i, $1 \leqslant i \leqslant k$ *such that* $(s_1 \circ s_i)$ *is in* $S(a)$.

Proof. From lemma 2, it results that it exists an optimal tree b and it exists $(s_i \circ s_j) \in S(b)$, such that $1 \leqslant i < j \leqslant k$.

Case 1: $i = 1$, then this lemma is obvious.

Case 2: $i > 1$, then there exists z and $(s_1 \circ z) \in S(b)$. If $H(z) \leqslant H(s_i)$, then $(s_i \circ s_j)$ and $(s_1 \circ z)$ can be changed into $(z \circ s_j)$ and $(s_1 \circ s_j)$, and lemma is proved. If $H(s_i) < H(z) \leqslant H(s_j)$, then $(s_i \circ s_j)$ and $(s_1 \circ z)$ can be changed into $(s_1 \circ s_j)$ and $(s_i \circ z)$, and lemma is proved. If $H(z) > H(s_j)$, then $(s_i \circ s_j)$ and $(s_1 \circ z)$ can be changed into $(s_i \circ s_1)$ and $(s_j \circ z)$. Due to commutativity, $(s_i \circ s_1)$ can be changed into $(s_1 \circ s_i)$, and the lemma is proved, too.

LEMMA 4. *If* $H(s_1) < H(s_2)$,

$$\textit{and } H(s_2) \leqslant H(s_3) \leqslant \ldots \leqslant H(s_n),$$

then there exists $a^* \in OT(\circ, s_1, \ldots, s_n)$ *such that* $(s_1 \circ s_2)$ *is in* $S(a^*)$.

Proof. If $H(s_2) = H(s_n)$, then applying lemma 1, lemma 4 results immediately. If it exists $k > 2$ such that $H(s_k) < H(s_{k+1})$, then applying lemmas 2 and 3, lemma 4 is proved, too.

Now, from the lemmas 1, 2, 3 and 4, it results the following:

THEOREM 1. *For each* $a \in BT(\circ, s_1, \ldots, s_n)$ *with*

$$H(s_1) \leqslant H(s_2) \leqslant \ldots \leqslant H(s_n),$$

there exists $a^* \in OT(\circ, s_1, \ldots, s_n)$ *such that* $(s_1 \circ s_2) \in S(a^*)$.

This theorem offers an algorithm to construct an optimal tree for an arithmetic expression, when it uses only one associative and commutative operator, and the operands $s_1, s_2, \ldots, s_n$ have the depth $H(s_1), H(s_2), \ldots, H(s_n)$.

```
ALGORITHM 1
Input: The expression $e = s_1 \circ s_2 \circ \ldots \circ s_n$.
Output: An optimal expression $f$ equivalent to $e$.
Step 1: While $n > 2$ do
    Step 1a: Sort the operands such that
                 $H(s_1) \leqslant H(s_2) \leqslant \ldots \leqslant H(s_n)$
    Step 1b: Define the expression $z = (s_1 \circ s_2)$ as a new operand and
             replace $s_1$ and $s_2$ by $z$.
    Step 1c: Rename the operands $z, s_3, \ldots, s_n$ into $s_1, s_2, \ldots, s_{n-1}$ res-
             pectively; Let $n := n - 1$;
```

Step 2: Let $f := (s_1 \circ s_2)$; Stop.

This algorithm is analogous to Huffman's algorithm [3] for finding a binary tree having a minimal weighted lenght.

Now, for finding an optimal tree for an arithmetic expression e having commutative and associative operands from the set OP, one can apply the following algorithm.

```
ALGORITHM 2.
Input: An expression $e$, with commutative and associative operators from
    a set $OP$ and the operands $a_1, a_2, \dots, a_q$.
Output: An optimal expression $f$ equivalent to $e$.

Step 1: Define $H(a_1) = H(a_2) = \dots = H(a_q) = M$.
Step 2: Construct a tree $b$ equivalent to $e$.
Step 3: While $S(b) \neq \emptyset$ do
    Step 3a: Traverse $b$ in postorder, to find the maximal elements $s_1$,
             $s_2, \dots, s_n$ of $S(b)$ such that in each $s_i$ only the same ope-
             rator can appear.
    Step 3b: For $i$ from 1 to $n$ do apply the algorithm 1 to $s_i$ and the
             result is denoted also by $s_i$.
    Step 3c: Replace each subtree $s_i \in b$ with the leaf $s_i$ which have
             the depth $H(s_i)$.
```

5. **An example.** Let us consider the expression given in fig. 2. After applying the steps 1, 2 and 3a of the algorithm 2, the expressions s_1, s_2, s_3, s_4 and s_5 have to be optimized.

Now, at step 3b, we apply the algorithm 1 to

$$(((((a + b) + c) + d) + e) + f)$$

We have $H(a) = H(b) = H(c) = H(d) = H(e) = H(f) = 1$. Using the step 1b of the algorithm 1, we define $z_1 = (a + b)$ with $H(z_1) = 2$. After the step 1c, we obtain the expression:

$$((((c + d) + e) + f) + z_1)$$

Now, we have $n = 5$, $H(c) = H(d) = H(e) = H(f)$ and $H(z_1) = 2$. Then we define $z_2 = (c + d)$ to obtain:

$$(((z_2 + e) + f) + z_1)$$

Now we have $n = 4$, $H(z_2) = H(z_1) = 2$, $H(e) = H(f) = 1$. Then we define $z_3 = (e + f)$ and we obtain:

$$((z_3 + z_2) + z_1)$$

Now we have $n = 3$, $H(z_3) = H(z_2) = H(z_1) = 2$. Then we define $z_4 = (z_3 + z_2)$ with $H(z_4) = 3$, and we obtain the optimal expression

$$s_1 = ((a + b) + ((e + f) + (c + d)))$$

having the optimal depth $H^*(s_1) = 4$.

Analogously, we obtain the following optimal expressions:

$$s_2 = ((g + h) + (k + (i + j))) \text{ and } H^*(s_2) = 4$$
$$s_3 = (((n + o) + (l + m)) + (r + (p + q))) \text{ and } H^*(s_3) = 4$$
$$s_4 = (s + (t + u)) \text{ and } H^*(s_4) = 3$$
$$s_5 = ((v + w) + (z + (x + y))) \text{ and } H^*(e_5) = 4$$

Now, after applying the step 3c of the algorithm 2, we have for optimization the following arithmetic expression:

$$(((((s_1 \cdot s_2) \cdot s_3) \cdot s_4) \cdot s_5)$$

After sorting, we define $z = (s_4 \cdot s_1)$, and $H(z) = 5$, and we obtain the expression:

$$((((s_4 \cdot s_1) \cdot s_2) \cdot s_3)) \cdot s_5)$$

Next, we have $n = 4$, and after sorting we define $z = (s_2 \cdot s_3)$ and $H(z) = 5$. The following expression is obtained:

$$(((s_2 \cdot s_3) \cdot s_5) \cdot (s_4 \cdot s_1))$$

After two steps, we obtain the optimal expression:

$$e_1 = ((s_5 \cdot (s_2 \cdot s_3) \cdot (s_4 \cdot s_1)))$$

having $H^*(e_1) = 7$.

The complete expression is:

$$e_1 = ((((v + w) + (z + (x + y))) \cdot$$
$$\cdot (((g + h) + (k + (i + j))) \cdot$$
$$\cdot (((n + o) + (1 + m)) + (r + (p + q))) \cdot$$
$$\cdot ((s + (t + u)) \cdot$$
$$\cdot ((a + b) + ((e + f) + (c + d))))))$$

and its binary tree is shown in the fig. 3.

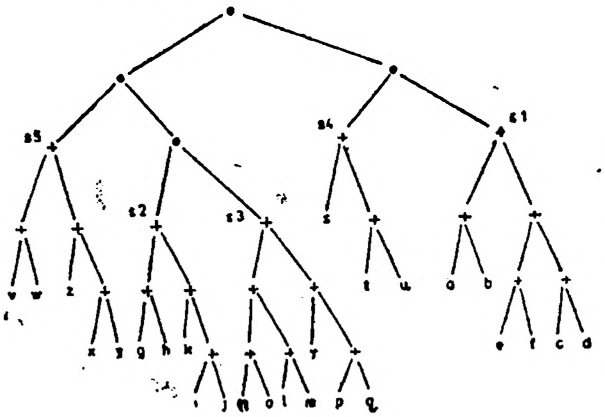


Fig. 3.

6. **Conclusions.** This method can do equivalent transformations over arithmetic expressions, such that it can be executed with maximum parallelism. In this method, the priority order between the elements of OP is not used. Also, other properties of operators are not used too. Of course, using other properties to some particular cases one can obtain better results.

For example, let us consider the expression [5]:

$$e = ((a \cdot b + c) \cdot d + e) \cdot f + g$$

It is an optimal expression using only the associative and the commutative properties of the addition, having $H^*(e) = 7$. But, using the distributivity of multiplication over addition, as in [5], one can obtain the following equivalent expression:

$$e = a \cdot b \cdot d \cdot f + c \cdot d \cdot f + c \cdot f + g$$

having $H^*(e) = 5$.

Also, using a performant compiler, the subexpressions (f) and $(d \cdot f)$ can be evaluated only once.

REFERENCES

1. B o i a n, F., *Reversible Execution with Loop—Exit Schemes*, „Studia Univ. Babeş-Bolyai, Math.", 32, 3 (1987) 29—36.
2. B o i a n, F., F r e n ţ i u, M., K á s a Z., *Parallel Execution in Loop—Exit Schemes*, Babeş-Bolyai University, Faculty of Math., Research Seminars, Preprint No. 9, 1988, pp. 3—16.
3. K n u t h, D. E., *The Art of Computer Programming, vol. I. Fundamental Algorithms*, Wesley, Reading (Massachusetts), 1968.
4. K u n g, H. T., *Computational Models for Parallel Computers*, CMU—CS—88—164, Carnegie Mellon University.
5. S c h e n d e l, U., S c h y s k a, M., *Parallel Computation and Supercomputers and Applications*, Freie Universität Berlin, Fachbereich Mathematik, Seria A, A—88—03.

STUDIA UNIV. BABES—BOLYAI, MATHEMATICA, XXXIV, 3, 1989

R- CONTRACTIONS

IOAN A. RUS*

Received: April 12, 1989

REZUMAT. — R-contracţii. În [10] am dat diverse generalizări ale unui rezultat obţinut de S. Eilemberg. În prezenta lucrare ne propunem să dăm diverse teoreme discrete de : punct fix, surjectivitate şi coincidenţă. Se dau, de asemenea, unele aplicaţii in analiza neliniară.

1. **Introduction.** Let X be a set, $f: X \to X$ a mapping and $R := (R_n)_{n\geq 0}$, $R_n \subset X \times X$ a sequence of equivalence relations. The following theorem is due to S. Eilemberg (see [4], [10]).

THEOREM 1.1. *We suppose that*

(i) $X \times X = R_0 \supset R_1 \supset \ldots \supset R_n \supset \ldots$;

(ii) $\bigcap R_n = \Delta(X)$ *(the diagonal in* $X \times X$*)*;

(iii) *if* $(x_n)_{n\geq 0}$ *is any sequence in* X *such that* $(x_n, x_{n+1}) \in R_n$ *for each* n, *then there exists* $x \in X$ *such that* $(x_n, x) \in R_n$ *for all* $n \in \mathbb{N}$;

(iv) *for all* $n \in \mathbb{N}$, $(x, y) \in R_n$ *implies* $(f(x), f(y)) \in R_{n+1}$.

Then

(a) $F_f = \{x^*\}$,

(b) $(f^n(x_0), x^*) \in R_n$ *for all* $x_0 \in X$ *and* $n \in \mathbb{N}$.

In [10] some discrete fixed point theorems of Eilenberg type are given. For example, the following result is given in [10].

THEOREM 1.2. *Let* X *be a set,* $f: X \to X$ *a mapping and* $R_n \subset X \times X$, $n \in \mathbb{N}$ *a sequence of symmetric binary relations in* X. *We suppose that:*

(1) *the conditions* (i), (ii) *and* (iv) *in the Theorem* 1.1. *are satisfied*;

(2) *if* $(x_n)_{n\geq 0}$ *is any sequence in* X *such that* $(x_n, x_{n+1}) \in R_n$ *for each* n, *then there is a unique* $x^* \in X$ *such that* $(x_n, x^*) \in R_n$ *for all* $n \in \mathbb{N}$.

Our purpose in this paper is to give some discrete fixed point theorems, discrete surjectivity theorems and discrete coincidence theorems. Some aplications in nonlinear analysis are given.

2. **R-contractions.** Let X be a nonempty set and $R = (R_n)_{n\geq 0}$, $R_n \subset X \times X$, a sequence of symmetric binary relations in X. Throughout this paper we suppose that:

(C$_1$) $X \times X = R_0 \supset R_1 \supset \ldots R_n \supset \ldots$,

(C$_2$) $\bigcap_0^\infty R_n = \Delta(X)$,

(C$_3$) if $(x_n)_{n\geq 0}$ is any seuqence in X such that $(x_n, x_{n+p}) \in R_n$ for all n, $p \in \mathbb{N}$, then there is a unique $x^* \in X$ such that $(x_n, x^*) \in R_n$ for all $n \in \mathbb{N}$.

* *University of Cluj-Napoca, Faculty of Mathematics and Physics, 3400 Cluj-Napoca, Romania*

DEFINITION 2.1. A mapping $f: X \to X$ is called *R-contraction* if for all $n \in \mathbf{N}$, $(x, y) \in R_n$ implies $(f(x), f(y)) \in R_{n+1}$.

DEFINITION 2.2. A mapping $f: X \to X$ is called *R-nonexpansive* if for all $n \in \mathbf{N}$, $(x, y) \in R_n$ implies $(f(x), f(y)) \in R_n$.

DEFINITION 2.3. A mapping f is *R-continuous* if $(x_n, x^*) \in R_n$, for all $n \in \mathbf{N}$ imply $(f(x_n), f(x^*)) \in R_n$ for all $n \in \mathbf{N}$.

We have

THEOREM 2.1. *If $f: X \to X$ is a R-contraction, then:*

(i) $F_f = \{x^*\}$,

(ii) $(f^n(x_0), x^*) \in R_n$, *for all* $x_0 \in X$ *and* $n \in \mathbf{N}$.

Proof. (i) + (ii). Let $x_0 \in X$. We have $(x_0, f^p(x_0)) \in R_0$, for all $p \in \mathbf{N}$. This implies that $(f^n(x_0), f^{n+p}(x_0)) \in R_n$, for all n and $p \in \mathbf{N}$. From (C_3) there exists a unique $x^* \in X$ such that $(f^n(x_0), x^*) \in R_n$, for all $n \in \mathbf{N}$. But this implies $(f^{n+1}(x_0), f(x^*)) \in R_{n+1}$, i.e., $(f^{(n)}(x_0), f(x^*)) \in R_n$ for all $n \in \mathbf{N}$. From (C_3) it follows that $x^* = f(x^*)$. Let now to prove (i). Let $x^*, y^* \in F_f$. From $(x^*, y^*) \in R_0$ and the R-contractivity of f we have $x^* = y$.

THEOREM 2.2. *Let $f: X \to X$ be a mapping. If there exists $k \in \mathbf{N}^*$ such that f^k is a R-contraction, then $F_f = \{x^*\}$.*

Proof. The proof follows from the Lemma 1.3.3. in [9] and the Theorem 2.1.

THEOREM 2.3. *Let X be a set, Y a nonempty subset of X $\rho: X \to Y$ a retraction and $f: Y \to X$ a mapping. We suppose that:*

(1) ρ *is R-nonexpansive,*

(2) f *is a R-contraction,*

(3) f *is retractible onto Y by means of ρ.*

Then $F_f = \{x^*\}$.

Proof. Let $(x, y) \in R_n$. From (2), $(f(x), f(y)) \in R_{n+1}$, and from (1), $(\rho(f(x)), \rho(f(y))) \in R_{n+1}$, i.e. $\rho \circ f$ is a R-contraction. From the Theorem 2.1. it follows that. $F_{\rho \circ f} = x^*$. From (3) we have $F_f = \{x^*\}$.

Remark 2.1. From the Theorem 2.1. we have the Theorem 1.1. of E i l e n b e r g (see [4], pp. 17—18), and the Theorem 5 in [10].

Example 2.1. Ordered vector spaces. Let $(X, +, R, \leqslant)$ be an ordered vector space. If X is a relatively σ-complete lattice, then X is called a σ-complete vector lattice. If $x \in X$, let $|x| := x \vee (-x)$ be the modulus of x. By definition a sequence $(x_n)_{n \geqslant 0}$ of elements in X is (o)-convergent to an element x^* if there exist two sequences $(a_n)_{n \geqslant 0}$ and $(b_n)_{n \geqslant 0}$ in X such that

(i) $(a_n)_{n \geqslant 0}$ is increasing and $x^* = \bigvee\limits_{n \in \mathbf{N}} a_n$,

(ii) $(b_n)_{n \geqslant 0}$ is decreasing and $x^* = \bigwedge\limits_{n \in \mathbf{N}} b_n$,

(iii) $a_n \leqslant x_n \leqslant b_n$ for all $n \in \mathbf{N}$.

Let $Y \subset X$ be a bounded and (o)-closed subset of X. Let $a \in]0,1[$ and $M_0 \in X$ such that, $|y_1 - y_2| \leqslant M_0$, for all $y_1, y_2 \in Y$. Let $R = (R_n)_{n \geqslant 0}$, $R_n \subset Y \times Y$,

$$R_n := \left\{(x, y) \,\middle|\, |x - y| \leqslant \frac{a^n}{1 - a} M_0,\ x, y \in Y\right\}.$$

The sequence $(R_n)_{n\geq 0}$ satisfies (C_1), (C_2), (C_3) (see [2] p. 135). Let $f: Y \to Y$ be such that $|f(x) - f(y)| \leq a|x-y|$, for all $x, y \in Y$. Then f is a R-contraction. From the Theorem 2.1. we have

THEOREM 2.4. (see [13]). *Let X be a σ-complete vector lattice, $Y \subset X$, o-closed subset of X and $f: Y \to Y$. We suppose that*

(1) *there exists $M_0 \in X$ such that $|y_1 - y_2| \leq M_0$ for all $y_1, y_2 \in Y$,*

(2) *there exists $a \in [0,1[$ such that*

$$|f(x) - f(y)| \leq a|x-y|, \text{ for all } x, y \in X.$$

Then

(i) $F_f = \{x^*\}$,

(ii) $f^n(x_0) \xrightarrow{(0)} x^*$ *as* $n \to \infty$, *for all* $x_0 \in Y$,

(iii) $|f^n(x_0) - x^*| \leq \frac{a^n}{1-a} M_0$.

Example 2.2. Metric spaces. Let (X, d) be a bounded complete metric space and $f: X \to X$ a (δ, a)-contraction (see [9]). Let

$$R_n := \left\{(x,y) \in X \times X \,\middle|\, d(x,y) \leq \frac{a^n}{1-a}\delta(X)\right\}.$$

The sequence $R = (R_n)_{n\geq 0}$ satisfies (C_1), (C_2) and (C_3) and f is R-contraction. From the Theorem 2.1. we have

THEOREM 2.5 (see [9]). *Let (X, d) be a bounded complete metric space and $f: X \to X$ a (δ, a)-contraction. Then $F_f = \{x^*\}$, and for all $x_0 \in X$, $f^n(x_0) \to x^*$ as $n \to \infty$.*

Example 2.3. Generalized metric space. Let (X, d) be a complete generalized metric space, where $d(x, y) \in \mathbb{R}^n$ (see [8]). Let $Y \subset X$ be a bounded closed subset of X and $f: Y \to Y$ a mapping .We suppose that there exists a non-negative matrix A whose spectral radious $\gamma(A)$ is less than 1 such that.

$$d(f(x), f(y)) \leq A\, d(x, y), \text{ for all } x, y \in X. \qquad (*)$$

Let $R_n := \{(x,y) \in Y \times Y \mid d(x,y) \leq (I-A)^{-1} A^n \delta(Y)\}$.
Then $R = (R_n)_{n\geq 0}$ satisfies (C_1), (C_2) and (C_3) and f is a R-contraction. From the Theorema 2.1. we have.

THEOREM 2.6. (see [11]); see also [8] and [7]). *If f satisfies $(*)$ then*

(i) $F_f = \{x^*\}$,

(ii) $f^n(x_0 \to x^*$ *as* $n \to \infty$,

(iii) $d(f^n(x_0), x^*) \leq (I-A)^{-1}A^n\delta(Y)$.

3. Surjectivity theorems. Let $(X, +)$ be an abelian group. Let $R_n \subset X \times X$, $n \in \mathbb{N}$, be a sequence of symmetric binary relations in X.

In what follows we suppose that $R = (R_n)_{n\geq 0}$ satisfies (C_1), (C_2), (C_3) and

(C_4) $(z \in X, (x,y) \in R_n, n \in \mathbb{N})$ implies $((x+z, y+z) \in R_n)$.

We have

THEOREM 3.1. *Let $f: X \to X$ be a R-contraction. Then $1_X - f: X \to X$ is surjective.*

Proof. Let $y \in X$. Let $x_1, x_2 \in X$. Then we have $(x_1, x_2) \in R_n$ implies $(f(x_1), f(x_2)) \in R_{n+1}$. From (C_4) it follows $(f(x_1) + y, f(x_2) + y) \in R_{n+1}$. The mapping $g : X \to X$, $x \mapsto f(x) + y$ is a R-contraction. The proof follows from the theorem 2.1.

Remark 3.1. From the Theorem 3.1. we have some surjectivity theorems given in [8], [9] and [13].

4. **Coincidences.** 4.1. Let X and Y be two nonempty sets, $f, g : X \to Y$ two mappings. An element $x^* \in X$ is a coincidence point of f and g iff $f(x^*) = g(x^*)$. Let $C(f, g)$ be the set of all coincidence point of the pair f, g.

We have

THEOREM 4.1. *Let X and Y be two sets, $f, g : Y \to X$ and $R = (R_n)_{n \geqslant 0}$, $R_n \subset X \times X$ which satisfies (C_1), (C_2) and (C_3). We suppose that*

(i) *g is surjective,*

(ii) *$y_1, y_2 \in Y$, $(g(y_1), g(y_2)) \in R_n$ implies $(f(y_1), f(y_2)) \in R_{n+1}$, for all $n \in \mathbb{N}$.*

Then $C(f, g) \neq \emptyset$.

Proof. Let $x_1, x_2 \in X$. From (i) there exist $y_1, y_2 \in Y$ such that $x_1 = g(y_1)$, $x_2 = g(y_2)$. Let $g_r^{-1} : X \to Y$ be such that $g \circ g_r^{-1} = 1_X$. We have that $(x_1, x_2) \in R_n$ implies $(f \circ g_r^{-1}(x_1), f \circ g_r^{-1}(x_2)) \in R_{n+1}$, for all $n \in \mathbb{N}$, i.e., $f \circ g_r^{-1}$ is a R-contraction. By the Theorem 2.1., there exists $x^* \in X$ such that $f(g_r^{-1}(x^*)) = x^*$. Let $y^* = g_r^{-1}(x^*)$. We have $y^* \in C(f, g)$.

Remark 4.1. From the Theorem 4.1. we have

THEOREM 4.2. *Let (X, d) be a bounded complete metric space and $f, g : X \to X$ a (δ, a)-contraction pair* (see [9]). *If g is surjective then $C(f, g) \neq \emptyset$.*

Proof. Let $a \in]0,1[$. Let

$$R_n : = \left\{(x, y) \in X \times X \,\middle|\, d(x, y) \leqslant \frac{a^n}{1-a}\,\delta(X)\right\}.$$

We have

$$\delta(f(A)) \leqslant a\delta(g(A)), \text{ for all } A \subset X.$$

This implies that we have the condition (ii) in the Theorem 4.1. The proof follows from the Theorem 4.1.

4.2. Let X be a nonempty set. Let $f, g : X \to X$ be two mappings. We denote

$$Y_1 : = f(X) \cap g(X), \qquad Z_1 : = f^{-}(Y_1) \cap g^{-1}(Y^1),$$
$$Y_2 : = f(Z_1) \cap g(Z_2) \qquad Z_2 : = f^{-1}(Y_2) \cap g^{-1}(Y_2),$$
$$\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots$$
$$Y_n : = f(Z_{n-1}) \cap g(Z_{n-1}), \qquad Z_n : = f^{-1}(Y_n) \cap g^{-1}(Y_n)$$
$$\cdots\cdots\cdots\cdots\cdots$$

We have

THEOREM 4.3. *If*

(i) $\bigcap_{n\in\mathbb{N}} Z_n = \{x^*\}$,

(ii) $Z_n \times Z_n \subset R_n$, *for all* $n \in \mathbb{N}$,

(iii) f *and* g *are R-continuous,*

then $C(f, g) \neq \emptyset$.

Proof. We remark that, $X \supset Z_1 \supset \ldots \supset Z_n \supset \ldots$ Let $x_n \in Z_n$. Then $(x_n, x^*) \in R_n$ for all $n \in \mathbb{N}$. On the other hand there exist $x_n \in Z_n$ and $y_n \in Z_n$ such that $f(x_n) = g(y_n)$. From (i) and (iii) we have $f(x^*) = g(x^*)$, i.e., $x^* \in C(f, g)$.

Remark. 4.2. From the Theorem 4.3. we have a theorem by H o l o d o v s k i (see [8]).

R E F E R E N C E S

1. R. C r i s t e s c u, *Topological vector spaces*, Editura Acad., Bucureşti, Noordhoff, Leyden, 1977.
2. R. C r i s t e s c u, *Structuri de ordine în spaţii liniare normate*, Editura Ştiinţifică şi enciclopedică, Bucureşti, 1983.
3. E. D u b i n s k y, *Fixed points in non-normed spaces*, Ann. Acad. Sc. Fennicae, Series A, Nr. 331, 1963.
4. J. D u g u n d j i, A. G r a n a s, *Fixed point theory in topological vector spaces*, P.W.N., Warszawa, 1982.
5. S. H e i k k i l ä, S. S e i k k a l a, *On fixed points in uniform spaces with applications to probabilistic metric spaces*, Acta Univ. Oulnensis, Series A1, nr. 18, 1978.
6. K. I s e k i, P. L. S h a r m a, B. K. S h a r m a, *Contraction type mapping on 2-metric space*, Math. Japonicae, 21 (1976), 67–70.
7. O k a n o, *On a class of complete spaces and some fixed point theorems*, Math. Japonicae, 21 (1976), 179–185.
8. I. A. R u s, *Principii şi aplicaţii ale teoriei punctului fix*, Editura Dacia, Cluj-Napoca, 1979.
9. I. A. R u s, *Generalized contractions*, Univ. of Cluj-Napoca, Preprint Nr. 3, 1983, 1–130.
10. I. A. R u s, *Discrete fixed point theorems*, Studia Univ. Babeş-Bolyai, Math., 33 (1988), fasc. 3, 61–64.
11. T. S h i b o t a, *On Matkowski's fixed point theorem*, TRU Math., 18–1 (1982), 57–60.
12. H. V. T r o t h a, *Contractivity in certain D–R spaces*, Math. Nachr., 101 (1981), 207–213.
13. F. V o i c u, *Teoreme de punct fix şi ecuaţii operatoriale în spaţii liniare ordonate*, Ph. D. disertation, Univ. of Bucharest, 1988.

STUDIA UNIV. BABEŞ—BOLYAI, MATHEMATICA, XXXIV, 3, 1989

CLASSIFICATION WITH FUZZY RELATIONS II

CRISTIAN LENART*

Received: April 18, 1989

REZUMAT. — Clasificare cu relaţii nuanţate II. Articolul de faţă constituie partea a doua a lucrării [3]. În prima parte s-au dat două teoreme de existenţă a celei mai bune aproximări a unei relaţii nuanţate prin relaţii clasice de echivalenţă în raport cu norma Cebişev şi norma integrală. În lucrarea de faţă vom demonstra două teoreme analoage care ne permit să ne apropiem de elementul de cea mai bună aproximare, aproximind numai pe mulţimi finite de puncte. De asemenea, vom da un algoritm pentru determinarea celei mai bune aproximări pe mulţimi finite în raport cu norma Cebişev. Definiţiile şi notaţiile sint cele din [3].

Introduction. This paper is the second part of the paper [3]. We proved there the existence of the best approximation of a fuzzy relation with classical equivalence relations with respect to the Tchebishew and the integral norm. Finding the best approximation of a fuzzy relation on a set X is a rather difficult problem when X is infinite. Therefore, we now give two theorems, corresponding to those in [3], showing how to approach the best approximation, approximating only on finite sets. An algorithm for the best approximation on finite sets with respect to the Tchebishew norm is also given. The notations are those in [3].

4. **Approximation on finite sets with respect to the Tchebishew norm.** We now consider X as the metric space with the metric d (which induces a metric on $X \times X$). As in [3], $\mathfrak{R}(X)$ is the subspace of the space of bounded functions on $X \times X$ equiped with the Tchebishew norm.

THEOREM 3. *Let $S \subset X$, $\bar{S} = X$ and $S = \bigcup_{m=1}^{\infty} S_m$ where S_m are finite sets satisfying $S_m \subseteq S_{m+1}$, $(\forall)\ m \in \mathbf{N}$. If $R \in \mathfrak{R}(X)$ is continuous on $X \times X$ then:*

$$\inf_{R^* \in \mathfrak{R}_0(X)} \| R - R^* \| = \lim_{m \to \infty} \inf_{R_m \in \mathfrak{R}_0(S_m)} \| R - R_m \|,$$

where the second norm is taken in $\mathfrak{R}(S_m)$.

Proof. Note with ρ the left hand of the equality to prove. Obviously:

$$\rho = \inf_{R^* \in \mathfrak{R}_0(X)} \| R - R^* \| \geqslant \inf_{R_m \in \mathfrak{R}_0(S_m)} \| R - R_m \| \tag{1}$$

Choose arbitrarily $\varepsilon > 0$ and $x_0, \ldots, x_n \in X$ so that $R(x_0, x_1) \geqslant \rho - \frac{\varepsilon}{2}$, $\ldots$, $R(x_{n-1}, x_n) \geqslant \rho - \frac{\varepsilon}{2}$, $R(x_0, x_n) \leqslant 1 - \rho + \frac{\varepsilon}{2}$, which exist according to

* *University of Cluj-Napoca, Faculty of Mathematics and Physics, 3400 Cluj-Napoca, Romania*

theorem 1. Consider the points $y_i \in S$, $i = 1, \ldots, n$ sufficiently close to the corresponding ones above, so that $R(y_0, y_1) \geqslant \rho - \varepsilon, \ldots, R(y_{n-1}, y_n) \geqslant \rho - \varepsilon$, $R(y_0, y_n) \leqslant 1 - \rho + \varepsilon$ (which is also possible because $\bar{S} = X$ and R is continuous on $X \times X$). Recalling that $S = \bigcup_{m=1}^{\infty} S_m$ and $S_m \subseteq S_{m+1}$, $(\forall)\ m \in \mathbf{N}$, we get $\{y_0, \ldots, y_n\} \subseteq S_m$, for every $m > m_\varepsilon$. But then

$$\inf \{r \mid [R(y_0, y_1) \geqslant r, \ldots, R(y_{n-1}, y_n) \geqslant r, y_0, \ldots, y_n \in S_m] \Rightarrow R(y_0, y_m) > 1 - r\} \geqslant \rho - \varepsilon. \tag{2}$$

Theorem 1 now gives us:

$$\inf_{R_m \in \mathcal{R}_0(S_m)} \| R - R_m \| \geqslant \rho - \varepsilon,\ (\forall)\ m > m_\varepsilon. \tag{3}$$

Since ε was chosen arbitrarily the conclusion of the theorem follows from (1) and (3).

5. **Approximation on finite sets with respect to the integral norm.** In order to give a correspondant of the theorem 2, we have to limit the frame. Thus we consider $X \subseteq \mathbf{R}^s$ a Jordan measurable set of finite measure and, as in [3], $\mathcal{R}_m(X)$-a subset of $L_2(X \times X)$.

THEOREM 4. *Let X as specified above. If $R \in \mathcal{R}_m(X)$ is uniformly continuous on $X \times X$ then there exist some finite sets $S_m \subset X$, $m \in \mathbf{N}^*$ and*

$$R_m^* \in \text{conv}\,(\mathcal{R}_0(X) \cap \mathcal{R}_m(X)) \ \textit{with}\ R_m^* |_{S_m}$$

a best approximation of $R|_{S_m}$ in conv $\mathcal{R}_0(S_m)$ *with respect to the square deviation, so that R_m^* converges to R^* a.e. on $X \times X$ (R^* is the best approximation of R from theorem 2).*

Proof. Let us consider a grid of hypercubes in $\mathbf{R}^s$ generated by equidistant hyperplans paralel to the coordinate hyperplans. Select only those included in X. There exists an array of finite sets of k_m hypercubes of side r_m so that $k_m r_m^s \rightarrow \mu(X)$ and $r_m \rightarrow 0$. From now on we abandon the indexation by m and note with $x^1, x^2, \ldots, x^{k_m}$ the centers of the hypercubes, by $X_1, \ldots, X_{k_m}$ the hypercubes and by S_m the set of centers.

Let R_m^* be the best approximation of $R|_{S_m}$ in conv $\mathcal{R}_0(S_m)$ with respect to the square deviation. We extend R_m^* to $X \times X$ as follows:

$$R_m^*(x, y) = \begin{cases} R_m^*(x^i, x^j), \text{ if } (x, y) \in X_i \times X_j \\ 0, \text{ if } (x, y) \notin \bigcup_{i,j} X_i \times X_j \text{ and } x \neq y \\ 1, \text{ if } (x, y) \notin \bigcup_{i,j} X_i \times X_j \text{ and } x = y. \end{cases}$$

It is easy to observe that $R_m^* \in \text{conv}\,(\mathcal{R}_0(X) \cap \mathcal{R}_m(X))$ because $R_m^*|_{S_m} \in$ $\in \text{conv}\,\mathcal{R}_0(S_m)$. We note with $T_m = X \times X \setminus \bigcup_{i,j} X_i \times X_j$.

Let us now evaluate $\|R - R^*\|$ and $\|R - R^*_m\|$. Recalling that R^*_m is a best approximation, the meanvalue theorem for the Riemann integral (R being continuous is Riemann integrable), the uniform continuity of R and $r_m \to 0$, we obtain:

$$\int_{X\times X} (R(x,y) - R^*(x,y))^2 dx\,dy \leqslant \int_{X\times X} (R(x,y) - R^*_m(x,y))^2 dx\,dy =$$

$$= \sum_{i,j} \int_{X_i\times X_j} (R(x,y) - R^*_m(x,y))^2 dx\,dy + \int_{T_m} (R(x,y) - R^*_m(x,y))^2 dx\,dy \leqslant$$

$$\leqslant \sum_{i,j} \int_{X_i\times X_j} (R(x,y) - R^*_m(x,y))^2 dx\,dy + \mu(T_m) = \sum_{i,j} (R(x'^k, y'^k) -$$

$$- R^*_m(x^{k'}, y^{k'}))^2 r_m^{2s} + \mu(T_m) \leqslant \sum_{i,j} (R(x^k, y^k) - R^*_m(x^k, y^k))^2 r_m^{2s} + \varepsilon k_m^2 r_m^{2s} +$$

$$+ \mu(T_m),\ (\forall) m > m_1.$$

Similarly:

$$\int_{X\times X} (R(x,y) - R^*(x,y))^2 dx\,dy = \sum_{i,j} \int_{X_i\times X_j} (R(x,y) - R^*(x,y))^2 dx\,dy +$$

$$+ \int_{T_m} (R(x,y) - R^*(x,y))^2 dx\,dy \geqslant \sum_{i,j} (R(x''^k, y''^k) - R^*(x''^k, y''^k))^2 r_m^{2s} \geqslant$$

$$\geqslant \sum_{i,j} (R(x^k, y^k) - R^*(x^k, y^k))^2 r_m^{2s} - \varepsilon k_m^2 r_m^{2s} \geqslant$$

$$\geqslant \sum_{i,j} (R(x^k, y^k) - R^*_m(x^k, y^k))^2 r_m^{2s} - \varepsilon k_m^2 r_m^{2s},\ (\forall) m > m_2.$$

Considering the extreme terms in the above inequalities and combining them, we get:

$$|\,\|R - R^*\|^2 - \|R - R^*_m\|^2| \leqslant 2\varepsilon k_m^2 r_m^{2s} + \mu(T_m),$$
$$(\forall)\ m > m_0 = \max(m_1, m_2)$$

Passing to the superior limit and using $\mu(T_m) \to 0$, $k_m r_m^s \to \mu(X)$, $m \to \infty$, it results:

$$\overline{\lim_{m\to\infty}}\ |\,\|R - R^*\|^2 - \|R - R^*_m\|^2| \leqslant 2\varepsilon\mu^2(X)$$

As ε was chosen arbitrarily we let $\varepsilon \searrow 0$ and obtain:

$$0 \leqslant \underline{\lim}\ |\,\|R - R^*\|^2 - \|R - R^*_m\|^2| \leqslant \overline{\lim}\ |\,\|R - R^*\|^2 - \|R - R^*_m\|^2| \leqslant 0,$$

hence $\|R - R^*_m\| \to \|R - R^*\|$, which means that R^*_m is a minimizing array for R^*. From the proof of Beppo–Levi theorem (see for instance [4]) it follows that R^*_m is fundamental, hence convergent in the norm of $L_2(X \times X)$ and its limit is R^*. This means that $R^*_m \to R^*$ a.e. on $X \times X$.

6. An algorithm for the evaluation of the best approximation on finite sets with respect to the Tchebishew norm. Consider $X = \{x_1, \ldots, x_n\}$ a finite set and $R \in \mathfrak{R}(X)$ a fuzzy similarity relation. R may be identified with its associated matrix (r_{ij}), $r_{ij} = R(x_i, x_j)$, $i, j = \overline{1, n}$.

An algorithm for the evaluation of the best approximation R^* of R in $\mathfrak{R}_0(X)$ with respect to the Tchebishew norm is given bellow. This algortihm is inspired from the proof of theorem 1.

```
Step 1. For $i := 1$ to $n$, $j := 1$ to $n$ do $r^*_{ij} := -1$;

Step 2. $\rho := 1$;

Step 3. For $i, j := 1$ to $n$ do if $r_{ij} = \rho$ then $r^*_{ij} := 1$;
                                  if $r_{ij} = 1 - \rho$ then $r^*_{ij} := 0$;

Step 4. For $i := 1$ to $n$, $j := 1$ to $n$, $k := 1$ to $n$, do
            if $r^*_{ik} = 1$ and $r^*_{kj} = 1$ then $r^*_{ij} := 1$;
            if $(r^*_{ki} = 1$ and $r^*_{kj} = 0)$ or $(r^*_{jk} = 1$ and $r^*_{ik} = 0)$
                then $r^*_{ij} := 0$;

Step 5. $\rho := \sup(\{r_{ij} | r_{ij} < \rho, i, j = \overline{1, n}\} \cup \{1 - r_{ij} | r_{ij} > 1 - \rho,$
                    $i, j = \overline{1, n}\})$
            (if the set between paranthesis is nonempty)

Step 6. Repeat steps 3), 4), 5) while there exist elements in $R^*$ with
        the value $-1$ and elements with the value 1 are not bound
        to become 0, respectively 0 to become 1.

Step 7. For $i := 1$ to $n$, $j := 1$ to $n$ do
            if $r^*_{ij} = -1$ then $r^*_{ij} := 0$;

Step 8. Output $R^*$, $\|R - R^*\| = \rho$. Stop.
```

We shall now discuss the algorithm. It is obvious that there exist i, j so that $\|R - R^*\| \in \{r_{ij}, 1 - r_{ij}\}$, since X is finite. At the beginning all r^*_{ij} are -1; ρ will keep the value of $\|R - R^*\|$ at a given iteration, which will descend discretly according to the remark given above. Let ρ be fixed; the condition $\|R - R^*\| < \rho$ imposes $r^*_{ij} = 1$ for all i, j for which $r_{ij} \geqslant \rho$ and $r^*_{ij} = 0$ for all i, j for which $r_{ij} \leqslant 1 - \rho$ (indeed, if $r_{ij} \geqslant \rho$ and $r^*_{ij} = 0$, then $\|R - R^*\| \geqslant |r_{ij} - r^*_{ij}| = r_{ij} \geqslant \rho$ and if $r_{ij} \leqslant 1 - \rho$ and $r^*_{ij} = 1$ then $\|R - R^*\| \geqslant |r_{ij} - r^*_{ij}| = 1 - r_{ij} \geqslant \rho$). These inequality conditions may be converted to equality ones because those r^*_{ij} for which $r_{ij} > \rho$ or $r_{ij} < 1 - \rho$ were already given values. The same way, the transitivity of R^* imposes step 4. At step 5 the following less discrete value for ρ is computed. Steps 3, 4 and 5 are repeated until the appearence of contradictions at step 3 or 4.

Eventually, all -1 elements of R^* are given the value 0 which doesn't contradict the transitivity.

Example. Let $X = \{x_1, x_2, x_3, x_4\}$ and

$$R = \begin{bmatrix} 1 & 0.3 & 0.6 & 0 \\ 0.3 & 1 & 0.7 & 0 \\ 0.6 & 0.7 & 1 & 0.2 \\ 0 & 0 & 0.2 & 1 \end{bmatrix}$$

We observe that $R \in \mathcal{R}(X)$ but $R \in \operatorname{conv} \mathcal{R}_0(X)$, hence the convex decomposition algorithm given in [1] cannot be applied. Our algorithm gives the following results.

At the first iteration, with $\rho = 1$, we have:

$$r^*_{11} = r^*_{22} = r^*_{33} = r^*_{44} = 1$$
$$r^*_{14} = r^*_{24} = r^*_{41} = r^*_{42} = 0.$$

At the second iteration, with $\rho = 0.8$, we have:

$$r^*_{34} = r^*_{43} = 0.$$

At the third iteration, with $\rho = 0.7$, we have:

$$r^*_{12} = r^*_{21} = 0$$
$$r^*_{23} = r^*_{32} = 1$$

and at step 4 we put $r^*_{13} = r^*_{31} = 0$.

At the fourth iteration with $\rho = 0.6$ all elements of R^* are already given values. Hence we have;

$$R^* = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$\| R - R^* \| = 0.6.$

The matrix of R^* induces the following partition of X: $\{x_1\}$, $\{x_2, x_3\}$, $\{x_4\}$.

REFERENCES

1. Bezdek J. C., Harris J. D., *Fuzzy Partitions and Relations: an Axiomatic Basis for Clustering*. Reprint from J. Fuzzy Sets and Systems 1 (1978), 111–127.
2. Lenart C., *Pattern Recognition Algorithms in A.I.*, Thesis, University of Cluj-Napoca, 1988.
3. Lenart C., *Classification with Fuzzy Ralations*, Studia, nr. 3 (1988).
4. Muntean I., *Functional Analysis*, Lecture Notes, Univ. of Cluj-Napoca, 1974.

RESOLUTION OF SOME SYMMETRIC FUZZY RELATION EQUATIONS ON RESIDUATED STRUCTURES

ANTONIO DI NOLA*, ADA LETTIERI* and SALVATORE SESSA*

Received: April 24, 1989

REZUMAT. — Rezolvarea unor ecuaţii relaţionale fuzzy, simetrice, în structuri reziduale. Fie (L, $\wedge$, $\leqslant$, α, 1) o semilatice Brouweriană şi a, x, b $\in$ L. Se notează cu $\mathfrak{S}$ mulţimea tuturor soluţiilor x $\in$ L ale ecuaţiei simetrice (aαx) $\wedge$ (xαa) = b. Se demonstrează că dacă $Z \neq \emptyset$, atunci (aαb) $\wedge$ (bαa) este cel mai mare element al lui Z. Rezultate asemănătoare sînt date pentru ecuaţii relaţionale fuzzy intr-o latice Brouweriană.

1. The theory of fuzzy relation equations begun by Sanchez [6] in 1976 is an extension of the classical theory of the Boolean equations [5]. Sanchez [6] solved sup-inf fuzzy relation equations on complete Brouwerian lattices. It is worth mentioning that this theory has provided many applications, mainly in the Knowledge Engineering setting [3]. In [1], a symmetric fuzzy relation equation was defined on finite sets and solved on the unit interval, in view of some applications presented in [2]. We first solve a similar equation on a Brouwerian semilattice, further we give some conditions for the resolution of an analogous fuzzy relation equation on Brouwerian lattices.

2. We recall that a Brouwerian semilattice $(L, \wedge, \leqslant, \alpha, 1)$ is an algebra such that the structure $(L, \wedge, \leqslant, 1)$ is a meet-semilattice with the largest element 1 and where "α" is the residuation operator, i.e. the equivalence: $c \leqslant a\,\alpha\, b$ iff $a \wedge c \leqslant b$, where $a, b, c \in L$. It is well known that the following properties hold for all $a, b, c \in L$:

$$1\,\alpha\, a = a,\ a\,\alpha\, a = 1.$$
$$a \leqslant b \text{ iff } a\,\alpha\, b = 1.$$
$$a\,\alpha\,(b\,\alpha\, c) = (a \wedge b)\,\alpha\, c.$$
$$a\,\alpha\, b \geqslant b.$$
$$a \wedge (a\,\alpha\, b) = a \wedge b.$$
$$a\,\alpha\,(b \wedge c) = (a\,\alpha\, b) \wedge (a\,\alpha\, c).$$
$$(a\,\alpha\, b)\,\alpha\, b \geqslant a.$$
$$a\,\alpha\, c \geqslant b\,\alpha\, c \text{ and } c\,\alpha\, a \leqslant c\,\alpha\, b \text{ if } a \leqslant b.$$

If "$\leqslant$" is a linear ordering, then $a\,\alpha\, b = b$ if $a \wedge b$.

Without special reference, these properties shall be used in the sequel. Let $a, b \in L$ and $\mathfrak{S} = \mathfrak{S}(a, b)$ be the set of all the solutions $x \in L$ of the

* Università di Napoli, Facoltà di Architettura, Istituto Matematico, Via Monteoliveto 3, 80134 Napoli, Italy

following equation:

$$(a \& x) = (a \alpha x) \wedge (x \alpha a) = b. \tag{1}$$

For sake of completeness, we recall the derived operation "&" is known as biresiduation (e.g., [4]). We first show some easy Lemmas.

LEMMA 1. *If $\mathfrak{S} \neq \emptyset$, then we have $a \alpha x = a \alpha b$ for any $x \in \mathfrak{S}$.*

Proof. Let $x \in \mathfrak{S}$, then we have $a \wedge (a \alpha x) \wedge (x \alpha a) = a \wedge b$, i.e. $a < x = = a \wedge x \wedge (a\alpha x) = a \wedge b$. Thus $a\alpha(a \wedge x) = a\alpha(a \wedge b)$, i.e. $(a\alpha a) \wedge (a\alpha x) = = (a\alpha a) \wedge (a\alpha b)$ and therefore the thesis follows.

LEMMA2. *We have $a\&x \geqslant b$ iff $a \wedge b \leqslant x \leqslant b\alpha a$.*

Proof. Let $x \in L$ such that $(a\alpha x) \wedge (x\alpha a) \geqslant b$. Then $b \leqslant a\alpha x$ iff $x \geqslant \geqslant a \wedge b$ and $b \leqslant a\&x$ iff $x \wedge b \leqslant a$ iff $x \leqslant b\alpha a$.

LEMMA3. *If $a\&b$, then we have $x \leqslant a\alpha b$.*

Proof. It suffices to observe that $a \wedge x \leqslant a\&x \leqslant b$.

The following result gives a simple criterion for the resolution of the Equation (1).

THEOREM 1. *$\mathfrak{S} \neq \emptyset$ iff $a\&b \in \mathfrak{S}$. Further, we have $a\&b \geqslant x$ for any $x \in \mathfrak{S}$.*

Proof. Of course, we prove only the nontrivial implication. Let $\mathfrak{S} \neq \emptyset$ and x be an arbitrary element of $\mathfrak{S}$. Then we have

$$a\alpha(a\&b) = [a\alpha(a\alpha b)] \wedge [a\alpha(b\alpha a)] = [(a \wedge a)\alpha b] \wedge [(a \wedge b)\alpha a] = = (a\alpha b) \wedge 1 = a\alpha b.$$

By LEMMA 1, we deduce

$$a\alpha(a\&b) = a\alpha x \tag{2}$$

for any $x \in \mathfrak{S}$. We also have by LEMMAS 2 and 3,

$$x \leqslant a\&b. \tag{3}$$

From (2) and (3), then it follows that

$$a\&(a\&b) = [a\alpha(a\&b)] \wedge [(a\&b)\alpha a] \leqslant (a\alpha x) \wedge (x\alpha a) = b. \tag{4}$$

On the other hand, we have $a \wedge b \leqslant a\&b \leqslant b\alpha a$ and hence by Lemma 2,

$$a\&(a\&b) \geqslant b. \tag{5}$$

Then the thesis follows from (4) and (5).

3. Here we assume $(L, \wedge, \vee, \leqslant, \alpha, 1)$ to be a complete Brouwerian lattice. We recall that L is Brouwerian iff

$$a \wedge (\bigvee_{i\in I} x_i) = \bigvee_{i \in I} (a \wedge x_i)$$

for all $a, x_i \in L$, being I any set of indices. Let X, Y be two nonempty sets (not necessarily finite), $A : X \to L$, $B : Y \to L$ be two assigned fuzzy sets and $R : X \times Y \to L$ be an unknown fuzzy relation such that the following equation holds:

$$\bigvee_{x\in X} [A(x) \& R(x, y)] = B(y) \tag{6}$$

for any $y \in Y$. Note that if $L = [0, 1]$ with the usual lattice operations induced by the natural ordering on the reals (more generally, if L is a linear lattice) and X, Y are finite sets (in this case, the completeness of L is not necessary), we obtain the equation considered in [1], where a characterization of the maximal solutions is presented. Denoting by $\mathfrak{S} = \mathfrak{S}(A, B)$ the set of all the solutions R satisfying the Equation (6), we give the following iff condition.

THEOREM 2. *Let $(A\alpha B)$ be the fuzzy relation pointwise defined as $(A\alpha B)\cdot(x, y) = A(x)\alpha B(y)$ for all $x \in X$ and $y \in Y$. Then $(A\alpha B) \in \mathfrak{S}$ iff $\bigvee_{x\in X} A(x) \geqslant \bigvee_{y\in Y} B(y)$. Further, we have $R \leqslant (A\alpha B)$ for any $R \in \mathfrak{S}$.*

Proof. We have for all $x \in X$ and $y \in Y$:

$$\begin{aligned} A(x)\&[A(x)\alpha B(y)] &= \{A(x)\alpha[A(x)\alpha B(y)]\} \wedge \{A(x)\alpha B(y)]\alpha A(x)\} \\ &= [A(x)\alpha B(y)] \wedge \{[A(x)\alpha B(y)]\alpha A(x)\} \\ &= [A(x)\alpha B(y)] \wedge A(x) = A(x) \wedge B(y). \end{aligned}$$

Since L is Brouwerian, then we deduce for any $y \in Y$:

$$\begin{aligned} B(y) = \bigvee_{x\in X} \{A(x)\&[A(x)\alpha B(y)]\} &= \bigvee_{x\in X} [A(x) \wedge B(y)] \\ &= B(y) \wedge \{\bigvee_{x\in X} A(x)\}, \end{aligned}$$

i.e.

$$(A\alpha B) \in \mathfrak{S} \text{ iff } \bigvee_{x\in X} A(x) \geqslant B(y) \text{ for any } y \in Y.$$

Let $R \in \mathfrak{S}$ and since $A(x) \& R(x, y) \leqslant B(y)$, we deduce $R(x, y) \leqslant A(x)\alpha B(y)$ for all $x \in X$ and $y \in Y$ by LEMMA 3, i.e. $(A\alpha B) \geqslant R$ for any $R \in \mathfrak{S}$.

Remark 1. Note that if the Equation (6) is defined on finite sets, in Theorem 2 the completeness of L is not a necessary hypohesis.

Remark 2. In Theorem 2, the condition $\bigvee_{x\in X} A(x) \geqslant \bigvee_{y\in Y} B(y)$ is clearly satisfied if A and B are normal fuzzy sets, i.e. $A(x') = B(y') = 1$ for some $x' \in X$ and $y' \in Y$ (cfr. Theroem 3 of [1]).

Now we assume that X, Y are finite sets and L is a linear lattice (not necessarily complete). If $B(y) = 1$ for any $y \in Y$, then $\mathfrak{S} \neq \emptyset$ since it suffices to assume $R(x, y) = A(x)$ for all $x \in X$ and $y \in Y$. If $Y \neq B^{-1}(1) = \{y \in Y : B(y) = 1\}$, we can give the following characterization:

THEOREM 3. *Let $Y \neq B^{-1}(1)$. Then $\mathfrak{S} \neq \emptyset$ iff for any $y \in Y - B^{-1}(1)$, there exists a $x \in X$ such that $A(x) \geqslant B(y)$.*

Proof. If $\mathfrak{S} \neq \emptyset$, let $R \in \mathfrak{S}$ and $y \in Y - B^{-1}(1)$. Since X is finite, we have for some $x_0 \in X$:

$$[A(x_0)\alpha R(x_0, y)] \leq [R(x_0, y)\alpha A(x_0)] = B(y). \tag{7}$$

By LEMMA 1, we deduce $A(x_0)\alpha R(x_0, y) = A(x_0)\alpha B(y)$. If $A(x) < B(y)$ for any $x \in X$, then $A(x_0)\alpha R(x_0, y) = 1$ which implies $R(x_0, y)\alpha A(x_0) = B(y) < 1$ by (7), i.e.
$A(x_0) = R(x_0, y)\alpha A(x_0) = B(y)$, a contradiction.
Vice versa, we have $\bigvee_{x\in X} A(x) \geqslant B(y)$ for any $y \in Y - B^{-1}(1)$. Then, defining (cfr. Theorem 2 of [1]) the fuzzy relation $(A\xi B)$ as

$(A\xi B)(x,y) = A(x)\alpha B(y)$ if $B(y) < 1$ and (if $B^{-1}(1) \neq \emptyset)(A\xi B)(x,y) = A(x)$ if $B(y) = 1$, it is easily seen, reasoning as in Theorem 2, that $(A\xi B) \in \mathfrak{S}$, i.e. $\mathfrak{S} \neq \emptyset$.

Remark 3. Theorem 3 does not hold, in general, if L is a Brouwerian lattice as is proved in the following example:

Example. Let $(N, \wedge, \vee, \leqslant, 1)$ be the lattice of the non-negative integers, where "$\wedge$" (resp. "$\vee$") is the least (resp. greatest) common multiple (resp. divisor) and "$\leqslant$" is defined as $a \leqslant b$ iff a is multiple than b, $a, b \in N$. It is known [7, p. 84] that N is a (complete) Brouwerian lattice with maximum 1 (and minimum 0).

Let $X = \{x_1, x_2, x_3\}$, $Y = \{y_1, y_2\}$ and A, B be defined as $A(x_1) = 2$, $A(x_2)=4$, $A(x_3) = 6$, $B(y_1) = 1$, $B(y_2) = 3$. Let R be defined as $R(x_i, y_1) = A(x_i)$ for $i = 1, 2, 3$, $R(x_1, y_2) = R(x_2, y_2) = 3$, $R(x_3, y_3) = 2$.

It is easily seen that $R \,\&\, A = B$, i.e. $\mathfrak{S} \neq \emptyset$ but $A(x_i) \ngeqslant B(y_2)$ for $i = 1, 2, 3$.

Returning to the case that L is a complete Brouwerian lattice and X, Y are nonempty sets (not necessarily finite), a further result is the following:

THEOREM 4. *Let $(A \,\&\, B)$ be the fuzzy relation pointwise defined as $(A \,\&\, B)\cdot(x,y) = A(x) \,\&\, B(y)$ for all $x \in X$ and $y \in Y$. Then $(A \,\&\, B) \in \mathfrak{S}$ iff there exists $R \in \mathfrak{S}$ such that*

$$B(y) \wedge R(x,y) \leqslant A(x) \qquad)$$

and

$$A(x)\alpha B(y) = A(x)\alpha R(x,y) \qquad (9)$$

for all $x \in X$ and $y \in Y$. Further, $(A \,\&\, B) \geqslant R$ for any $R \in \mathfrak{S}$ satisfying conditions (8) and (9).

Proof. (cfr. also proof of Theroem 1). We have for any $y \in Y$:

$$\begin{aligned}
&\bigvee_{x\in X} \{A(x) \,\&\, [A(x) \,\&\, B(y)]\} = \\
&= \bigvee_{x\in X} \{[A(x)\alpha B(y)] \wedge \{\{[A(x)\alpha B(y)] \wedge [B(y)\alpha A(x)]\}\alpha A(x)\}\} = \\
&= \bigvee_{x\in X} \{[A(x)\alpha B(y)] \wedge \{[A(x)\alpha B(y)]\alpha\{[B(y)\alpha A(x)]\alpha A(x)\}\}\} = \\
&= \bigvee_{x\in X} \{[A(x)\alpha B(y)] \wedge \{[B(y)\alpha A(x)]\alpha A(x)\}\}. \qquad (10)
\end{aligned}$$

Let $(A \,\&\, B) \in \mathfrak{S}$ and by setting $R = A \,\&\, B$, we get:

$$\begin{aligned}
B(y) \wedge [A(x) \,\&\, B(y)] &= B(y) \wedge [A(x)\alpha B(y)] \wedge [B(y)\alpha A(x)] \\
&= B(y) \wedge [B(y)\alpha A(x)] \leqslant A(x)
\end{aligned}$$

and

$$A(x)\alpha[A(x) \,\&\, B(y)] = A(x)\alpha B(y)$$

for all $x \in X$ and $y \in Y$, i.e. conditions (8) and (9) are satisfied.

Vice versa, let $R \in \mathfrak{S}$ such that (8) and (9) hold. Then (8) implies that $R(x,y) \leqslant B(y)\alpha A(x)$ for all $x \in X$ and $y \in Y$. Hence, using (5), (9) and (10), we deduce:

$$B(y) \leqslant \bigvee_{x \in X} \{A(x) \,\&\, [A(x) \,\&\, B(y)]\}$$
$$\leqslant \bigvee_{x \in X} \{[A(x)\alpha R(x, y)] \wedge [R(x, y)\alpha A(x)]\} = B(y)$$

for any $y \in Y$, i.e. $(A \,\&\, B) \in \mathfrak{F}$. Since (9) (or Lemma 3) implies $A(x)\alpha B(y) \geqslant$ $\geqslant R(x, y)$ for all $x \in X$ and $y \in Y$, the thesis is completely proved.

REFERENCES

1. Di Nola, A., Pedrycz, W., Sessa, S., *Fuzzy relation equations with equality and difference composition operators*, Fuzzy Sets and Systems 25, 1988, pp. 205—215.
2. Di Nola, A., Pedrycz, W., Sessa, S., *Modus ponens for fuzzy data realized via equations with equality operators*, Internat. J. Intell. Systems, to appear.
3. Di Nola, A., Pedrycz, W., Sanchez, E., Sessa, S., *Fuzzy Relation Equations and Their Applications to Knowledge Engineering*, D. Reidel Publ. Co., to appear.
4. Pavelka, J., *On fuzzy logic II*, Zeitschr. f. Math. Logik and Grundlagen d. Math. 25, 1979, pp. 119—134.
5. Rudeanu, S., *Boolean Functions and Equations*, North-Holland Publ. Co., Amsterdam, 1974.
6. Sanchez, E., *Resolution of composite fuzzy relation equations*, Inform. and Control 30, 1976, pp. 38—48.
7. Szász, G., *Introduction to Lattice Theory*, Academic Press, New York, 1963.

STUDIA UNIV. BABEŞ—BOLYAI, MATHEMATICA, XXXIV, 3, 1989

USE OF DEEP CUTS IN KHACHIYAN'S ALGORITHM

TEODOR TOADERE*

Received: June 12, 1989

REZUMAT. — **Folosirea secţiunilor adînci în algoritmul lui Khachiyan.** Lucrarea reprezintă o dezvoltare a lucrărilor [5] şi [6] ale autorului, prin îmbunătăţirea şi corectarea unor propuneri pentru alegerea sferei de la care se pleacă în algoritmul lui K h a c h i y a n [2], precum şi prin modificarea numărului maxim de iteraţii necesare a fi efectuate. De asemenea, se face un studiu comparativ între patru variante ale algoritmului, două dintre acestea folosind secţiunile adinci propuse în [3].

1. **Introduction.** In order to establish whether the system:

$$a_i^T x \leqslant b_i, \ a_i \in \mathbf{Z}^n, \ b_i \in \mathbf{Z}, \ i = 1, 2, \ldots, m, \tag{1}$$

is consistent or not (and, in the affirmative case, to obtain a solution), K h a c h i y a n gave in 1979 a polynomial algorithm. The performances of this algorithm were improved in several papers, among which [2, 3, 6]. In this paper we propose a new manner if choice for the initial sphere for the algorithm, while in Section 4 we present some numerical results obtained by using the computer for four variants of K h a c h i y a n 's algorithm. We must mention that the modifications proposed in [5, 6] and in this paper assume that the solution of the system (1) is performed for $x \geqslant 0$. If, for instance, this condition is not required for the component x_i, it can be obtained by replacing x_i by $x_i' - x_i''$, where $x_i', x_i'' \geqslant 0$.

2. **Khachiyan's Algorithm.** Let $T = \{x \in \mathbf{R}^n \mid a_i^T x \leqslant b_i, \ i = 1, 2, \ldots, m\}$ be the set of the solutions of (1), possibly empty. The algorithm consists of the construction of a sequence of at most N ellipsoids with the property:

$$E_0 \supset E_1 \supset \ldots \supset E_k \supset \ldots \supset E_N \supset (T \cap E_0). \tag{2}$$

An ellipsoid E_k is defined by a vector $x_k \in \mathbf{R}^n$, which is its centre, and a matrix B_k, such that:

$$E_k = \{y \in \mathbf{R}^n \mid y = x_k + B_k z, \ \|z\| \leqslant 1\}, \tag{3}$$

or:

$$E_k = \{y \in \mathbf{R}^n \mid (y - x_k)^T B_k^{-1} (y - x)_k \leqslant 1\}; \tag{4}$$

that is why it is also denoted by the pair (x_k, B_k). The form (3) for which the condition $\det(B_k) \neq 0$ or $\mathrm{Vol}(E_k) \neq 0$ is required was used by K h a c h i y a n, while the form (4) for which B_k must be positively defined was used by G á c s — L o v á s z and is the most often used. We must emphasize

* University of Cluj-Napoca, Faculty of Mathematics and Physics, 3400 Cluj-Napoca, Romania

that for the same ellipsoid the matrices B_k from the two forms are not identical; there exists between them a well-specified relationship.

As initial ellipsoid one chooses a sphere of centre $x_0 = 0$ and radius R, containing solutions of the system (1) if this one is consistent. In the initial variant, $R = 2^L$, where L is the lenght of the input data code and is obtained from the relation:

$$L = [\sum_i \sum_j \log_2 (|a_{ij}| + 1) + \sum_i \log_2 (|b_i| + 1) + \log_2 (nm)] + 1.$$

This choice is disadvantageous for the solutions of the problems by computer. As it results from [5], even for small dimensions ($n = 3$, $m = 4$) problems, the great value of L leads to upper floating overflow. Khachiyan [2] specified that, without other modifications, the radius value can be chosen $\Delta\sqrt{n}$, where Δ is a majorant for the absolute values of the minors of the extended matrix of the system (1), while the value of L is chosen as being $[\log_2(2R\sqrt{n})] + 1$. In [4] and other papers, the author of the present paper gave some manners of choice for the values of Δ and the radius of the initial sphere from which Khachiyan's algorithm is starting. A majoration of Δ is obtained using Hadamard's inequality (the absolute value of every determinant is at most the product of the norms of its lines) as being $\alpha := $ the product of the greatest $p = \min\{n + 1, m\}$ norms of the lines of the extended matrix of the system (1). In [4, 6], considering the system (1) together with the restrictions $x \geqslant 0$, there was proposed the finding of edges, both inferior x_j^i and superior x_j^s, for each component x_j of the solution vectors for the system (1) from the initial ellipsoid. Subsequently, the author ascertained that the expression for x_j^i is erroneous; but this fact does not affect the numerical results presented in [5].

The values x_j^s can be obtained, as it is presented in [6], too, as follows:

— initialize $x_j^s := \alpha$ for $j = 1, 2, \ldots, n$.

— for every restriction of the system (1) with $a_i \geqslant 0$ and $b_i > 0$, calculate $x_j^s = \min\{x_j^s, b_i/a_{ij}\}$ for $j = 1, 2, \ldots, n$ and $a_{ij} \neq 0$.

Considering $x^i = (0, 0, \ldots, 0)$, $x^s = (x_1^s, x_2^s, \ldots, x_n^s)$, the following lemma holds:

LEMMA 1. *If the system (1) with the conditions $x \geqslant 0$ is consistent, then it has solutions into the sphere of centre:*

$$x_0 = (x^i + x^s)/2 \tag{5}$$

and radius:

$$R_1 = \| x^s - x_0 \| . \tag{6}$$

A change of the values x^i and x^s which diminishes the radius of the initial sphere is presented further down. We initialize:

$$x_j^i = 0,\ x_j^s = \alpha,\ j = 1, 2, \ldots, n.$$

If the system (1) contains at least one restriction with $a_i \geqslant 0$ and $b_i > 0$, then for every $a_{ij} > 0$ we obtain:

$$x_j \leqslant (b_i - \sum_{k \neq j} a_{ik} x_k)/a_{ij} \leqslant (b_i - \sum_{k \neq j} a_{ik} x_k^i)/a_{ij} =: \gamma_j,$$

which allows the choice:

$$x_j^s = \min \{x_j^s, \gamma_j\} \text{ for } a_{ij} > 0, \quad j = 1, 2, \ldots, n. \tag{7}$$

If the system (1) contains at least one restriction with $a_i \leqslant 0$ and $b_i < 0$, then for every $a_{ij} < 0$ we obtain:

$$x_j \geqslant (b_j - \sum_{k \neq j} a_{ik} x_k)/a_{ij} = b_i/a_{ij} - \sum_{k \neq j} (a_{ik}/a_{ij})\, x_k \geqslant$$

$$\geqslant b_i/a_{ij} - \sum_{k \neq j} (a_{ik}/a_{ij})\, x_k^s = (b_i - \sum_{k \neq j} a_{ik} x_k^s)/a_{ij} =: \delta_j.$$

and hence the following actualizations can be made:

$$x_j^i = \max \{x_j^i, \delta_j\} \text{ for } a_{ij} < 0, \quad j = 1, 2, \ldots, n. \tag{8}$$

Then we shall repeat the calculations from (7) and (8) until either all values x^i or x^s remain unchanged, or we have $x_j^s < x_j^i$ for a certain index j, that is the system (1) is inconsistent. If at the end of calculations for x^i and x^s one obtains $x_j^i = x_j^s$, then the j-th component of every solution point must be x_j^s, and therefore the number of un knowns decreases by replacing x_j by x_j^s and renumbering the unknowns. Using this calculation mode for x^i and x^s, Lemma 1 can be applied and we shall obtain a value R_2 for the radius of the sphere which will contain solution of the system (1), with the conditions $x \geqslant 0$, if this one is consistent. In Section 4 one can observe the differences between the values of R_1 and R_2, and the effect of their use for Khachiyan's algorithm as well.

As it is known, the algorithm requires at every step to verify whether x_k (centre of the current ellipsoid) is solution of the system (1). If the ellipsoid E_{k+1} is not constructed such that contain the semiellipsoid $E_k/2$ obtained by cutting the ellipsoid E_k by a hyperplane parallel to the restriction violated by x_k and passing through x_k (also called, for this reason, central cut). The calculation expressions for x_{k+1} and B_{k+1} and the complete algorithm as well can be found, for instance, in [6]. An important part in the proof of the algorithm and establishing the maximum number of iterations after which one can state whether the system (1) is consistent or not is played by the ratio of the volumes for two consecutive ellipsoids. So, Khachiyan proves:

$$\mathrm{Vol}\,(E_{k+1})/\mathrm{Vol}\,(E_k) \leqslant (n/(n+1))(n^2/(n^2-1))^{(n-1)/2} =$$
$$= (1 - 1/(n+1))(1 + 1/(n^2-1))^{(n-1)/2} \leqslant$$
$$\leqslant e^{-1/(n+1)} \left(e^{1/(n^2-1)}\right)^{(n-1)/2} = e^{-1/(2(n+1))}.$$

Using the fact that $2 < e$ and hence the following inequality $\mathrm{Vol}(E_{k+1})/\mathrm{Vol}(E_k) < 2^{-1/(2(n+1))}$ holds, K h a c h i y a n fixed the value of N to $9n^2L$, while in the variant G á c s – L o v á c s it reached $6n(n+1)L$ [1]. But $2^4 < e^3$, and the value of N becomes $9n(n+1)L/2$.

3. **Deep Cuts.** In order to accelerate the algorithm, some authors [1, 3] proposed the use of other cuts than the central ones; according to these cuts, the ellipsoid E_{k+1} contains by its construction less than $E_k/2$; for this reason they are called deep cuts.

We confine ourselves further down to the presentation of the algorithm proposed by K ö n i g and P a l l a s c h k e [3].

```
1. Calculate $(x, B) \sim E_0$
   $d_j = a_j^T B a_j,\ e_j = a_j^T x - b_j,\ j \in \{1, 2, \ldots, m\} =: I_a$
2. $\xi = \max\{c_j/d_j,\ j \in I_a\}$, $k$ = index where max is attained
   if $\xi \leqslant 0$ go to 7
   if $\xi > 1$ system (1) is unsolvable
3. $z = Ba_k/d_k,\ f_j = a_j^T z,\ j \in I_a$
4. Let $\eta = 1$;
   For all $j \in I_a$
       if $e_j \geqslant 0$ next $j$
         else:
         if $e_j < -d_j$ $I_a := I_a - \{j\}$, next $j$
           else:
           if $e_j > f_j$ let $s_j = f_j/d_j,\ t_j = e_j/d_j$
                          $u_j = s_j t_j + ((1 - s_j^2)(1 - t_j^2))^{1/2}$
                  if $u_j \geqslant \eta$ next $j$
                     else $\eta = u_j$, next $j$
             else:
             if $f_j \leqslant 0$ next $j$
               else let $u_j$ as above
                    if $u_j < \xi$ next $j$
                      else $I_a := I_a - \{j\}$
5. $\beta^2 = (n^2/(n^2 - 1))(1 - (\eta^2 + \xi^2)/2 + (((\eta^2 - \xi^2)/2)^2 + (1 - \eta^2)(1 - \xi^2)/n^2)^{1/2})$,
   $\alpha = (\eta - \xi)/((1 - (1 - \eta^2)/\beta^2)^{1/2} + (1 - (1 - \xi^2)/\beta^2)^{1/2})$,
   $\gamma = \xi + \alpha(1 - (1 - \xi^2)/\beta^2)^{1/2}$
6. $x := x - \gamma z$
   $B := \beta^2 B - (\beta^2 - \alpha^2) z \cdot z^T$
   $d_j := (\beta^2 d_j^2 - (\beta^2 - \alpha^2) f_j^2)^{1/2}$
   $c_j := e_j - \gamma f_j$
   if number of iterations $< N$ go to 2
     else stop system (1) is unsolvable
```

7. If all eliminated hyperplanes, $j \in \{1, 2, \ldots, m\} - I_a$, are satisfied, x solves (1) if one of the eliminated hyperplanes is violated, system (1) is unsolvable.

4. **Numerical Results.** In this section we shall present some results obtained by using a CORAL 4030 computer belonging to the Computing Data Centre of the University of Cluj-Napoca. We mention that the data representation in double precision was used. There were programmed both the algorithm which uses the central cuts and that which uses the deep cuts described in Section 3. For each variant the initial sphere was chosen in two manners, firstly with the radius R_1 (variants a, respectively b from Table 1), then with the radius R_2 (variants c and d, respectively, from the same Table 1).

Table 1

n	Case	Variant	L	Radius	N	k	Time (sec)
2	A	a	9	142.128	243	0	21
		b	9	142.128	243	0	10
		c	1	0.707	27	0	10
		d	1	0.707	27	0	15
	B	a	12	1414.920	324	0	9
		b	12	1414.920	324	0	9
		c	1	0.707	27	0	8
		d	1	0.707	27	0	111
3	A	a	10	260.673	540	74	29
		b	10	260.673	540	25	43
		c	2	0.866	108	2*	11
		d	2	0.866	108	1*	13
	B	a	14	2598.942	756	126	42
		b	14	2598.942	756	33	68
		c	2	0.866	108	2*	11
		d	2	0.866	108	1*	12
4	A	a	11	401.000	990	148	71
		b	11	401.000	990	48	91
		c	2	1.000	180	4*	20
		d	2	1.000	180	1*	45
	B	a	14	4001.000	1260	233	97
		b	14	4001.000	1260	62	106
		c	2	1.000	180	4*	1
		d	2	1.000	180	1*	15
5	A	a	12	560.135	1620	5*	
		b	12	560.135	1620	79	117
		c	3	1.118	405	5*	23
		d	3	1.118	405	1*	100
	B	a	15	5591.288	2025	305	188
		b	15	5591.288	2025	102	228
		c	3	1.118	405	5*	11
		d	3	1.118	405	1*	15

* The solution vector has all components equal; this is due to the fact that all cuts were madeusing the last inequation of the instance.

** Although the computer worked more than 8 hours, the algorithm did not find a solution of the system.

(contd.

n	Case	Variant	L	Radius	N	k	Time (sec)
10	A	a	14	1582.720	6930	937	2847
		b	14	1582.720	6930	347	982
		c	4	1.581	1980	17*	51
		d	4	1.581	1980	1*	24
	B	a	17	15812.969	8415	1388	2505
		b	17	15812.969	8415	441	2013
		c	4	1.581	1980	17*	60
		d	4	1.581	1980	1*	25
16	A	a	15	3202.000	18360	34*	162
		b	15	3202.000	18360	939	5377
		c	4	2.000	4896	34*	205
		d	4	2.000	4896	1*	60
	B	a	18	32002.000	22032	3985	19775
		b	18	32002.000	22032	1179	6522
		c	4	2.000	4896	34*	206
		d	4	2.000	4896	1*	59
20	A	a	16	4474.372	30240	**	
		b	16	4474.372	30240	1504	18825
		c	5	2.236	9450	47*	540
		d	5	2.236	9450	1*	70
	B	c	5	2.236	9450	47*	404
		d	5	2.236	9450	1*	88
30	A	c	5	2.738	20925	82*	861
		d	5	2.738	20925	1*	159
	B	c	5	2.738	20925	82*	2019
		d	5	2.738	20925	1*	194

The specified variants were applied to some instances obtained for the system of inequations of the form:

$$x_i \geqslant a, \; i = 1, 2, \ldots, n,$$

$$x_1 + x_2 + \ldots + x_n \leqslant na + b,$$

where for each value of n there were considered the following cases: A ($a=100$, $b = 1$) and B ($a = 1000$, $b = 1$).

Using the notations: N for the necessary maximum number of iterations $= 9n(n + 1)L/2$, and k for the number of iterations performed until the finding of a solution of the considered instance, Table 1 presents the obtained results.

As to the time given in Table 1, we must emphasize that it is orientative, sin e the computer we used is multiuser and worked only a part of the time indicated for the solution of the problem (other part of the time-being used for other tasks, simultaneously run).

We must also emphasize the fact that, using the new manner of calculating the radius, the values of both this one and L or N (hence the calculation accuracy and the algorithm complexity) do not change by passing from case A to case B for the same value of n; on the other hand, these ones are smaller than in other variants. By using the deep cuts, the complexity of an iteration increases, nevertheless the small necessary number of iterations leads to a smaller complexity of the algorithm for the considered instances than in the case of central cuts.

Finally we specify that the maximum number of iterations needed for establishing the consistence of the system is the same; it does not depend on the type of used cuts. Therefore, theoretically speaking, the complexity for the variant which uses the deep cuts is greater.

REFERENCES

1. Bland, R. G., Goldfarb, D., Todd, M. J., *The Ellipsoid Method. A Survey*, Operations Research, *29* (1981), 1039–1091.
2. Khachiyan L. G., *On the Exact Solving of Linear Inequation Systems and Linear Programming Problem*, Ž. Vycisl. Mat. i Matem. Fiz., *22* (1982), 999–1002 (Russ.).
3. König, H., Pallaschke, D., *On Khachiyan's Algorithm and Minimal Ellipsoids*, Numer. Math., 36 (1981), 211–223.
4. Toadere, T., *On Khachiyan's Algorithm for Polynomial Solving of the Linear Programming Problem*, Proc. Coll. on Approximation and Optimization, Cluj-Napoca, October 25–27, 1984, 339–342.
5. Toadere, T., *Numerical Experiments with Khachiyan's Algorithm*, Univ. of Cluj-Napoca, Res. Sem., Seminar on Computer Science, Report No. 5, 1987, 55–64.
6. Toadere, T., *A Modification of the Initial Sphere Choice for Khachiyan's Algorithm*, Studia Univ. Babeş-Bolyai, Mathematica, 33 (1988), No. 1, 46–49.

RECENZII

A. Di Nola, A. G. S. Ventre (Eds.), **The Mathematics of Fuzzy Systems**, Verlag TÜV Rheinland, Köln, 1986

Offering an excellent survey of current research in the mathematics of fuzzy concepts, this invaluable collection reflects the growing interest in this new field. The volume containg the contributions of many leading specialists in the area, provides not only the state of the art in the theory of Fuzzy Sets but suggest as will directions for future research.

From the contents we remember: Semantical structure of fuzzy logics (G. Cattaneo and G. Nisticò), Categorial foundations of fuzzy set theory with applications to algebra and topology (U. Cerruti and U. Höhle), Towards an algebraic setting of measure of fuzziness (A. Di Nola and A. Ventre), Fuzzy set theory with T-norm and Ø-operators (S. Gottwald), Relations between probability and fuzzy theory (A. Muir).

The volume is strongly recommended to all interested in Fuzzy Set Theory as well as a reference book.

D. DUMITRESCU

INTREPRINDEREA POLIGRAFICĂ CLUJ,
Municipiul Cluj-Napoca, Cd. nr. 450/1989

In cel de al XXXIV-lea an (1989) *Studia Universitatis Babeş—Bolyai* apare în specialităţile:

matematică
fizică
chimie
geologie-geografie
biologie
filosofie
ştiinţe economice
ştiinţe juridice
istorie
filologie

In the XXXIV-th year of its publication (1989) *Studia Universitatis Babeş—Bolyai* is issued as follows:

mathematics
physics
chemistry
geology-geography
biology
philosophy
economic sciences
juridical sciences
history
philology

Dans sa XXXIV-e année (1989), *Studia Universitatis Babeş—Bolyai* paraît dans les spécialités:

mathématiques
physique
chimie
géologie-géographie
biologie
philosophie
sciences économiques
sciences juridiques
histoire
philologie

43 875

Abonamentele se fac la oficiile poştale, prin factorii poştali şi prin difuzorii de presă, iar pentru străinătate prin „ROMPRES-FILATELIA", sectorul export-import presă, P. O. Box 12—210, telex. 10376 prsfir, Bucureşti Calea Griviţei nr. 64—66.

Lei