STUDIA
UNIVERSITATIS BABEȘ-BOLYAI

MATHEMATICA

3

1992

Anul XXXVII 1992

S T U D I A
UNIVERSITATIS BABEŞ-BOLYAI

MATHEMATICA

3

R e d a c ţ i a : 3400 CLUJ-NAPOCA str. M. Kogălniceanu nr.1 ▸ Telefon: 116101

S U M A R - C O N T E N T S - S O M M A I R E

STUDIA UNIV. BABEŞ-BOLYAI, MATHEMATICA, XXXVII, 3, 1992

ON AN EXPERIMENT IN SOLVING EQUATIONS USING GENETIC ALGORITHMS

Márton-Ernö BALÁZS[*]

Received: January 15, 1993
AMS Subject Classification: 68T01

REZUMAT.- Asupra unui experiment de rezolvare a ecuaţiilor folosind algoritmi genetici. Articolul prezintă concluziile unui experiment de utilizare a unui algoritm genetic pentru rezolvarea ecuaţiilor. Rezultatele sînt comentate în paralel cu prezentarea paşilor unui algoritm genetic general. Sînt de asemenea arătate unele soluţii specifice clasei de probleme abordate precum şi sugerate posibilităţi de îmbunătăţire ale rezultatelor.

0. Introduction. Solving equations has been for a long time a central problem in numerical applications. The most important keywords in the field of equation solving are the *domain of convergence, rate of convergence* and *complexity* of the methods. Although there are many nice results in developing methods of rather high rate of convergence with acceptable complexity, most of them are strongly limited in what concerns their domain of convergence. Nevertheless in many such methods testing for the convergence domain is of a comparable complexity to the computation itself. Another inconvenience of most of these methods is that of converging to a single solution (that is obviously induced by their nature) whereas there might be many other viable solutions around. These deficiencies of the classical equation solving methods gave place to the searching for methods with "large" convergence domains (see [1]). Even if these type of methods are usually of a great complexity they may be used at least for isolating to a certain amount the solutions,

[*] *"Babeş-Bolyai" University, Department of Mathematics and Computer Science, 3400 Cluj-Napoca, Romania*

giving then place to the "specialized" fast converging ones. Beside this approach in many cases these mathods can be improved by using some kind of domain specific properties as shown in [5]. We also should mention that the developing of these "unorthodox" methods was greatly encouraged by the development of computing techniques towards parallel processing. In the present paper we shall make some remarks on an experience in solving equations using *genetic algorithms*.

1.A short introduction to genetic algorithms. One of the striking ideas for designing optimum searching methods is that of using principles of organic evolution [3].The approaches emerging from this idea (*genetic algorithms* and *evolution strategies*) rely upon imitating the adaptation and selection capacity of natural populations. Although the two main approaches are similar to a large extent in the followings we shall discuss in the terms of genetic algorithms.

1.1. General structure of genetic algorithms. Omitting the details we can say that genetic algorithms (GA) operate on a fixed size *population* of individuals of the same type creating new *generations* of it in successive cycles.

Analogously with natural populations the creation of a newgeneration is directed by a *fitness criterion* which conditions the perpetuation of the most viable individuals. In GA-s this is carried out through the following operations:

a.) *evaluation of the fitness* of each individual of the current population;

b.) *formation of a gene pool* built up from the most fit individuals;

c.) creation of the new generation by *recombination* and *mutation* of elements in the gene pool.

The process of creating new generations is iterated until the average fitness of the population ceases to improve, by which time the population has converged to a few kinds of well performing individuals.

As stated above the creation of a new generation is carried out by recombination and mutation of elements in the current gene pool. Individuals contribute to the gene pool in a number depending on their fitness. The elements of the gene pool are codings of the best performing individuals (according to the fitness criterion) of the current population. Such a coding has to be a condensed representation of individuals which captures their defining features.

Recombination is performed through the crossover operator which randomly selects a fixed *number of parent genes* from the gene pool and creates an *offspring gene* by combining a randomly chosen part of them. Usually this combination is done either by combining the "tails" of the parent genes (one point crossover), or by combining some internel part of them (two point crossover). We should mention here that this crossover is not merely a random creation of a new gene since it preserves features of the parent genes.

Mutation is meant for preserving the diversity of the population which prevents the algorithm from converging to local solutions as well as from omitting some solutions in the search space. It is esentially carried out by randomly changing some portion of a gene.

Now we can present an outline of a GA (after [3] and [5]). Let *N* be the fixed size of the population and *f* the function which measures the fitness of an individual (*fitness function*).

```
procedure GA
   begin
     t:=0;    { t is the number (moment) of the generation }
     create an initial population P*t(;
     evaluate the fitness of each individual in P*t( using f;
     while termination condition not satisfied do
       begin
         create a gene pool G*t( bz coding the most fit
                individuals of P*t(;
         t:=t+1;
         select parent genes from P*t-1(;
         recombine selected genes and build P*t(;
         evaluate the fitness of each individual in P*t(
                  using f;
       and
   end.
```

In the followings we shall make our references to this description.

1.2. A short overview of some results concerning GA-s. We shall mention the results and make some comments following the steps in the algorithm outlined in the previous section. *Creation of the initial population*. Although it is not explicitly required most impelmentations of genetic algorithms begin with a uniformly distibuted random initial population. This approach is reasonalble since it intuitively prevents a possibly long-drawn phase of diversification of individuals before convergence, but in some specific problems it has been shown that it is not absolutely necessary. *Evaluation of fitness of individuals*. The most common way of treating fitness evaluation is by simple function evaluation. To do this the objective function o (the function to be optimized) has to be transformed by composition to a nonnegative number: $f(.)=u(o(.))$. This transformation may damage some desirable properties of the GA (see [5]).

Creation of the gene pool. In the creation of the gene pool one should handle the following problems: the selection of individuals from the current population, establishing the *contribution* of each selected individual to the gene pool and *coding* of the resulting set of individuals.

Let us start with the *coding*. As it results from the literature many contributors to GA-s' theory argue for binary coding but there are some who consider other representation more suited for special classes of problems [5]. Choosing binary coding, beside other features is apealing for its simplicity in implementing recombination on digital computers.

As already stated *selection* and establishing the *contribution* of individuals to be used for building the gene pool depends on the fitness computed in the previous step of the algorithm. There are several selection algorithms (which also establish the contributions) of which the most used seems to be *the proportional selection algorithm* which computes the number of contributions of an individual as ratio of the individual's fitness and the average fitness of the current population. For this selection algorithm Holland gave a characterisation of the number of classes of individuals in the next generation (the Schema Theorem, see [5]). We omit further discussions here.

Selection of parent genes. Parent *genes* are selected randomly from the gene pool. The number of parents used for generating an offspring gene depends on the model used. As presented in [5] GA-s may use either a *one parent* or a *manyparents* model.

Recombination of genes. We have already mentioned that recombination of genes is performed through *crossover* and *mutation*. Crossover means combination (tipically exchanging) of a randomly chosen part of the parent genes. This operation is the main mechanism which creates new candidates for solution deriving them from old ones by preserving a significan part of their properties. In order this latter statement to be true the size of gene portions to be combined is kept small. In the previous section we gave the two alternative crossover types: *one point* and *two point* crossover.

It is not our goal to go in more details about GA-s here, the interested reader is suggested to read [5].

2. Notes on an experiment in solving equations. The main goal of the present paper is to report on some notes of the author made during an experiment in using a GA for solving various equations. In the followings we shall present some of them.

Our purpose was to try to develop a GA type method for approximatively solving equations of the form $f(x)=0$, where $f:R^n \to R$. Obviously this problem can easily be transformed into another, suited for application of a GA: *Approximate* the *minimum points of function* $g:R^n \to R$ where $g(x)=|f(x)|$ of which only those solutions x^* are solutions of the original problem for which $g(x^*)=0$. So we applied a GA to this problem. For the beginning we only studied one dimensional problems which also raised some difficulties.

In our implementation the *population was made up* of real numbers of the domain in which solutions were sought (i.e. an interval of the real axis). We varied the *population size* in different runnigns ranging from 10 to 200 individuals, noting that the convergence of the algorithm improved for population sizes over 100, by which time it correctly made distinction between more solutions.

Whether the the *initial population* was built by uniformly distributing numbers on the search segment or not seemed in our experiment of no significance - the number of steps in which the desired precision was achieved was about the same.

The *coding* used is a two level one which first mappes the search domain into the segment [0,1], then considered the first *k* significant binary digits of the representation, where *k* depends on the desired precision. This coding is a simple one to one mapping from individuals to their representation which also has the advantage of easy reconstruction of an individual from a code. As all binary codings it also has the advantage of simplicity of implementing crossover.

For *selection* the proportional selection algorithm was used, with randomly distributing the unassigned positions in the gene pool to individuals in the current population. The only model tried was the two parents one.

The most problems in tuning the algorithm were raised by the recombination of genes and building of the new population.

Implementing *crossover* and *mutation* was thechnically speaking simple due to the binary coding used. A two point crossover was tried with length ranging from 1 to 8 bits of the representation. Here we can make the following notes:

a.) The *convergence speed of the algorithm* was improved by making the probability with which the portion of genes to be combined was selected depend on the generation number in such a way that in "advanced" generations less significant digits were combined with greater probability than the more significant ones. This observation sounds intuitively fair since in advanced stages of the search the algorithm should work harder on improving the precision of the already approximated solutions but without excluding the possibility of finding new candidates.

b.) When selecting a parent gene with a probability *p* it is essential *to select the coparent gene* with a probability 1-*p* in order the two parents to be different. Probably this note isnot valid if crossover is done with other combination than simple exchanging.

For *building the new population* from the gene pool many variants have been tried.

At the beginning the new population was entirely constructed from the gene pool. With this approach for all the equations tried the method converged to just one (of the known) solutions even if in early stages of search all the solutions have had some approximation.

This fenomenon was eliminated when a generation was built up by preserving the most fit individuals from the previous one and generating just some new individuals from gene pool. This is however a known practice [5] even if we do not have sufficient data to support the *one fifth principle*.

In the figure below we present the outlines of the results obtained with our implementation for one of the equations studied.

Evolution of number of individuals (real numbers) around the solutions of equation **sin 2x** over (1.4) for a population size of 100.

	1.57...	3.14...	Total
Generation: 1	7.00%	7.00%	14.00%
Generation: 2	17.00%	26.00%	43.00%
Generation: 3	41.00%	8.00%	49.00%
Generation: 4	32.00%	14.00%	46.00%
Generation: 5	33.00%	32.00%	65.00%
Generation: 6	47.00%	20.00%	67.00%
Generation: 7	59.00%	25.00%	84.00%
Generation: 8	64.00%	29.00%	93.00%
Generation: 9	69.00%	24.00%	93.00%
Generation:10	62.00%	31.00%	93.00%
Generation:11	65.00%	32.00%	97.00%

3. Conclusions. From the experience presented in the previous section we conclude that GA-s can be successfully used for solving equations although much care has to be taken in designing the implementation. The expected advantage of this approach is that of being applyabble even for equations which are defined by non continuous functions (not to speak about not differentiable ones), a quality which is not common for other known methods.

Obviously the implicit parallelism of the method is an other appealing property which should be exploited in implementations. As we did it in [1] we suggest the combining of the GA with specialized, fast solving methods in order to obtain both generality and speed.

In the followings we intend to work on improving the selection methods and building of the fitness function using, where possible properties of the equation defining function.

R E F E R E N C E S

1. M.E.Balázs - *A Heuristic Search Type Algorithm for Solving Nonlinear Equation Systems,* Studia Univ. "Babeş-Bolyai" Mathematica, XXXIII, 3, 1988.
2. J.J.Grefenstette, J.E.Baker - *How Genetic Algorithms work: A Critical Look at Implicit Parallelism,* Proceedings of ICGA'89, ed. J.D.Schaffer, (1989).
3. F.Hoffmeister - *A Survey of Evolution Strategies,* roceedings of ICGA'91, ed. R.K.Blew, (1991).
4. J.H.Holland - *Adaptation in Natural and Artificial Systems,* University Michigan Press (1975).
5. G.E.Liepins, M.R.Hilliard - *Genetic Algorithms: Foundations and Applications,* Annals of Operation Research, 21 (1989).

STUDIA UNIV. BABEŞ-BOLYAI, MATHEMATICA, XXXVII, 3, 1992

EXTENDED B-TREE

F. M. BOIAN[*]

Received: January 3, 1993
AMS Subject Classification: 68P10, 68P20

REZUMAT. B-arbore extins. In lucrare se prezintă o extindere a conceptului de *B*-arbore. Prin această extindere se permite accesul relativ în acest tip de structură de date. Prin acces relativ se înţelege posibilitatea de deplasare optimală în *B*-arbore peste *n* chei faţă de cheia curentă.

DEFINITIONS. In [3] B-tree was formally defined. We denote by m the order of the B-tree, and we denote by e the number of keys from the current node from B-tree. By p, p_0, p_1, p_2 etc. We denote some pointers to nodes from B-tree. At last, by K, with possible subscripts, we denote value(s) of key(s) from B-tree.

If p is a pointer to a node from B-tree, we denote by $S(p)$ the sub B-tree having the root in the node pointed by p.

DEFINITION 1. *The possession of $S(p)$ is the total number of keys from $S(p)$. We denote this number by $Z(p)$.*

Let $a = K_{i+1}K_{i+2}\ldots K_{i+r}$ be the r succesive keys from the same node of B-tree. Let p_i, p_{i+1}, p_{i+2}, $\ldots$, p_{i+r} be the neighbours pointers for the keys from a.

Notations. By $S(a)$ we denote the sub B-tree which has in its root only the keys from a and the descendents $S(p_i)$, $S(p_{i+1})$, $S(p_{i+2})$, $\ldots$, $S(p_{i+r})$.

We denote by $Z(a)$ the possession of $S(a)$.

By $|\ a\ |$ we denote the number r (the number of keys from a).

[*] *University of Cluj-Napoca, Department of Computer Science, 3400 Cluj-Napoca, Romania*

THEOREM 1. *The following relations (with the above notations), holds:*

$Z(a) = r + Z(p_i) + Z(p_{i+1}) + Z(p_{i+2}) + \ldots + Z(p_{i+r})$

For each j *from* 1 *to* r,

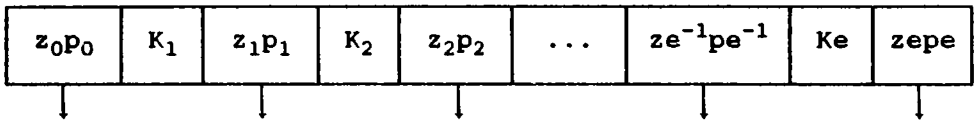
$Z(a) = Z(K_{i+1}\ldots K_{i+j}) + Z(K_{i+j+1}\ldots K_{i+r}) - Z(p_{i+j})$ *and*

$Z(a) = Z(K_{i+1}\ldots K_{i+j-1}) + 1 + Z(K_{i+j+1}\ldots K_{i+r})$

The *proof* of this theorem immediately follows from the definition of possession.

With these considerations, we continue to define an extended *B*-tree.

DEFINITION 2. *An* ***Extended B-tree*** [1] *is a B-tree having in its nodes the following information:*



where $z_i = Z(p_i)$, $i = 0, 1, \ldots, e$

An example. In fig. 1, an extended *B*-tree is presented. In each node, only values of keys are presented. For leafless nodes, there are two arrows near each key: one on the left and the other on the right. On the left of each arrow, in brackets, the value of possession appears, and on the right, the value of the pointer (here is the number of the node) appears.

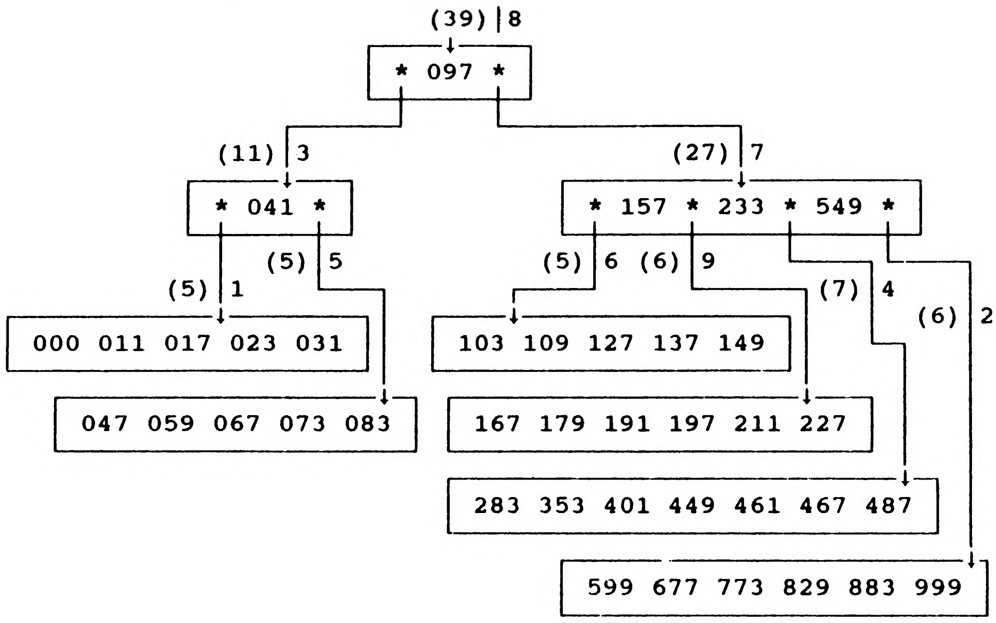


Figure 1. An extended *B*-tree

For *example*, *S*(7) has the nodes 7, 6, 9, 4, 2, and *Z*(7) = 27. If *a* = "157 233", then *S*(*a*) is *S*(7) without the key "549" and without the node 2, and *Z*(*a*) = 20.

Extended *B*-tree transformation. The operations with *B*-tree are presented in [3]. In [2] and [1] we have described some ideas to implement an extended *B*-tree. In figures 2, 3 and 4 three pairs of transformations are presented: rotate left /right, transform a node into two or reverse, transform two nodes into three or reverse. In these figures, we note by lower case (*a, b, ..., h, i*) the sequences, possibly empty, from consecutive keys (from the same node), and by an uppercase (*C, E, G*) a key from a node.

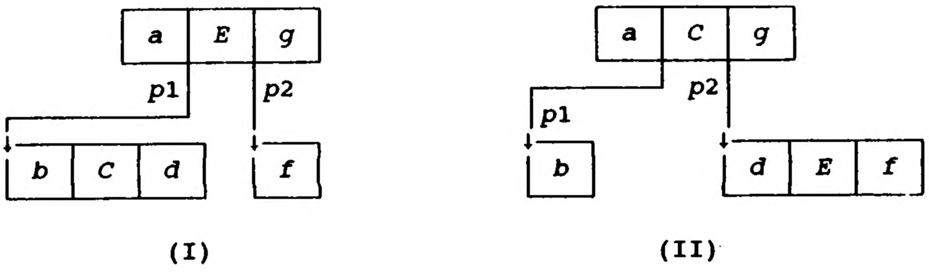


Figure 2 Rotate left / right

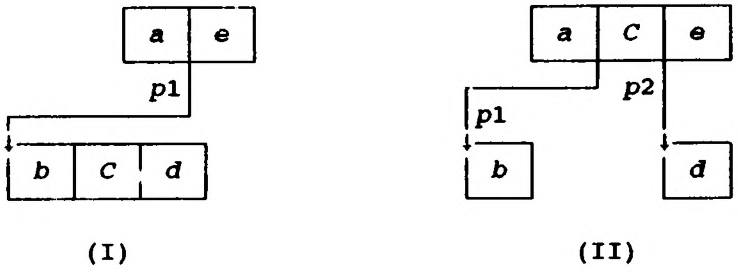


Figure 3 Transformation between one node - two nodes

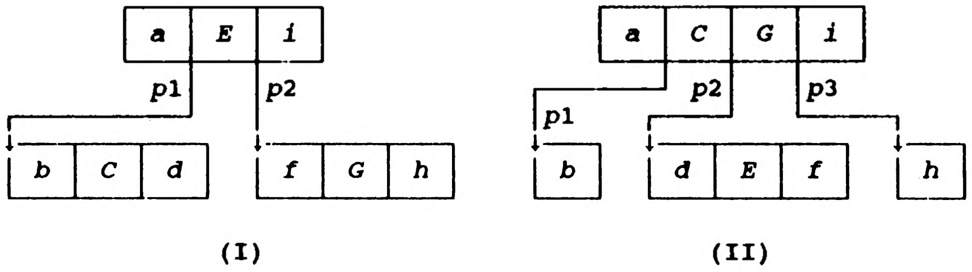


Figure 4 Transformation between two nodes - three nodes

When a transformation is applied, the possessions for new nodes must be computed only from the olds, without to see the others nodes from *B*-tree. In the following, for the fourth usualy transformations, the new possessions are:

1) From (I) to (II) of the fig. 2 (rotate to right):

$Z(p_1) := Z(b);$

$Z(p_2) := Z(d) + 1 + Z(f);$

2) From (II) to (I) of the fig. 2 (rotate to left):

$Z(p_1) := Z(b) + 1 + Z(d);$

$Z(p_2) := Z(f);$

3) From (II) to (I) of the fig. 3 (fusion of two nodes into one):

$Z(p_1) := Z(b) + 1 + Z(d);$

4) From (I) to (II) of the fig. 4 (transformations two nodes into three):

$Z(p_1) := Z(b);$

$Z(p_2) := Z(d) + 1 + Z(f);$

$Z(p_3) := Z(h).$

We have used these four transformatios in [1] for implementation. If only these are used, at most two nodes are necessary for operations with *B*-tree.

When these transformations must be applieds? From [3] these are applied, possibily, after deleting a key or inserting a key, if after that the number of keys from the current node are less than m / 2 or great m. If after a deletion, in node remain less than m / 2 keys, then this event is called **undersized**. If after a insertion, in the node there are great m keys, then this event is called **overflow**. The following rules are applied, in this order:

```
1 If (|bCd| = m+1 (overflow) and |f| < m) or
      (|f| = m / 2 - 1 (undersize) and |bCd| > m / 2)
   then
      b and d are choisen so that | |bCd| - |f| | ≤ 1
      and rotations on the right are applied (see I to II in
fig 2).
2 If (|dEf| = m+1 (overflow) and |b| < m) or
      (|b| = m / 2 - 1 (undersize) and |dEf| > m / 2)
```

then
d and f are choisen so that $|\ |dEf| - |b|\ | \leq 1$
and rotations on the left are applied (see II to I in fig. 2).

3 If undersize and $|b| + |d| < m$
then
two node join into one (see II to I in fig. 3).

4 If overflow and $|bCd| + |fGh| = 2m+1$
then
transform two node into three other, with (approximate) the same numbers of keys: b, d, f and h are choise so that:
$||b|-|dEf|| \leq 1$ and $||dEf|-|h|| \leq 1$ and $||b|-|h|| \leq 1$
(see I to II in fig. 4).

Relative access in extended B**-tree**. Let K_c (current key) and K_t (target key) two keys from a B-tree. Suppose that between K_c and K_t, in ascendent order, there are other n-1 keys. The problem is to construct an algorithm so that to minimize the number of moves in B-tree to find K_t when K_c is the current key.

Let p be the pointer to the nearest node so that both K_c and K_t can be accessed from it. This node is called **common ancestor** from both keys. Its clear that all the n-1 keys between K_c and K_t are in $S(p)$. Because each other ancestor of K_c and K_t is ancestor for their common ancestor, it results that minimal number of moves is from common ancestor to K_t.

To find common ancestor between current key and any other key, we purpose to create and update a stack. When a key K_c is found, for each ancestor of K_c an record is pushed in this stack. An record from stack has the following structure:

$$(p_i,\ j_i,\ zl_i,\ zr_i,\ Kl_i,\ Kr_i)$$

where:

i is the current level in B-tree (the root has the level 1);

p_i is the pointer to the node;

In the following, we suppose that the node p_i has the form:

$$z_{i0}p_{i0}\ K_{i1}\ \ldots\ K_{ij}\ z_{ij}p_{ij}\ \ldots\ K_{ie}\ z_{ie}p_{ie}$$

j_i is the index of the key K_c, if $K_{ij} = K_c$, or K_c is in $S(p_{ij})$, if $K_{ij} \neq K_c$;

$zl_i = Z(K_{i1}...K_{ij}) - Z(p_{ij}) - 1$ (the possession to left of K_{ij});

$zr_i = Z(K_{ij}...K_{ie})$ (the possession to right of K_{ij});

Kl_i is the minimum value from $S(p_i)$;

Kr_i is the maximum value from $S(p_i)$;

For example, in the B-tree from fig. 1, if $K_c = 211$, the stack is:

i	$p\ i$	$j\ i$	$zl\ i$	$zr\ i$	$Kl\ i$	$Kr\ i$
1	8	1	11	28	-oo	+oo
2	7	1	5	22	097	+oo
3	9	5	4	2	157	233

The fields of this stack can be completed during the search for a key. All informations for a record are known from the current node or from its father. The last record from stack corresponds to a node having the current key in it. The maximum size of this stack is very small (see [3] for details).

Now, suppose that the current key is K_c and we want to skip over n keys (forward or backward if $n < 0$). For that, we pop from stack until $n \leq zr_i$ when $n \geq 0$, or until $-n \leq zl_i$ when $n < 0$. The p_i from top of stack pointed to common ancestor to K_c and K_t over n keys over K_c.

This stack helps to reduce the nùmber of nodes accessed when looking for a key having a value. For that, its suffices to pop from stack until the value of the new key is between Kl_i and Kr_i. In the most cases, the search a new key begins instead the root with an it's descendant for a same level.

R E F E R E N C E S

1. Boian F. M., *Sistem de fişiere bazat pe B-arbori,* în Lucrările celui de-al VII-lea colocviu naţional de informatică, INFO-IASI, 1989, pp. 33-40.
2. Boian F. M., *Căutare rapidă în B-arbori,* în Lucrările simpozionului "Informatica şi aplicaţiile sale", Zilele academice Clujene, Cluj-Napoca, 1989.
3. Knuth D. E., *Tratat de programarea calculatoarelor;* vol III, Sortare şi căutare. Ed. Tehnică, Bucureşti, 1976.

STUDIA UNIV. BABEŞ-BOLYAI, MATHEMATICA, XXXVII, 3, 1992

PROGRAM TESTING IN LOOP-EXIT SCHEMES

F.M.BOIAN* and M.FRENŢIU*

Received: March, 3, 1993
AMS Subject Classification: 68Q50, 68Q60

REZUMAT. - Testarea schemelor Loop-Exit. În această lucrare se introduce noţiunea de drum complet într-o schemă Loop-Exit şi se arată importanţa drumurilor complete într-o schemă program pentru testarea programelor. De asemenea, se construieşte un limbaj care generează mulţimea drumurilor complete.

1. Introduction. In this paper we consider the Loop-Exit Schemes as they were defined in [2]. Nevertheless, we impose a minor condition: there is an initial assignement a_0 just at the begining (after START block in the corresponding flowchart), and a final assignment a_f at the end (in front of the STOP block). A and T are the sets of assignment and test symbols, respectively, and $M = A \cup T$. Also, we denote by $SW(S)$ the skeleton word associated to S, and we denote by $D(x\alpha y)$ the direct word from x to y (as in [4]).

To each Loop-Exit Scheme S a language $L(S)$ may be associated. More exactly, we have the following definition:

DEFINITION 1. *The language $L(S)$ associated to the Loop-Exit Scheme S is generated by the following context free grammar* [1,2,3]:

$$G(S) = (\mathrm{N}, \Sigma, \mathcal{P}, \nabla)$$

where

$$\mathrm{N} = \{\nabla\} \cup \{I_j | j>0\} \cup \{L_k | k>0\},$$

$$\Sigma = M \cup \{+,-\}$$

I_j is a nonterminal for IF$_j$, and L_k and B_k are two nonterminals

* University of Cluj-Napoca, Departament of Computer Science, 3400-Cluj-Napoca, Romania

for LOOP$_k$ from the definition of the Loop-Exit Scheme S, ∇ is a new symbol - the axiom of $G(S)$, and the set $\mathcal{P}$ of the productions is constructed by the following rules:

a) ∇ ---> SW(S)

b) the following productions

b1) I_j ---> b-

b2) I_j ---> b+SW(α) only if α has not the form α'EXIT$_k$

are in $\mathcal{P}$ if

IF$_j$ b THEN$_j$ α ENDIF$_j$;

is in S.

c) the productions

c1) I_j ---> b+SW(α) if $\alpha \neq \alpha$'EXIT$_k$;

c2) I_j ---> b-SW(β) if $\beta \neq \beta$'EXIT$_k$;

are in $\mathcal{P}$ if

IF$_j$ b THEN$_j$ α ELSE$_j$ β ENDIF$_j$;

is in S.

d) if

LOOP$_k$ $\alpha_1\alpha_2\delta$ ENDLOOP$_k$;

is in S then the productions

d1) L_k ---> SW($\alpha_1\alpha_2\delta$)L_k

d2) B_k ---> SW($\alpha_1\alpha_2\delta$)B_k | ϵ

d3) L_k ---> D(LOOP$_k$ α_1 IF$_j$) b+SW(β)

if

α_2 = IF$_j$ b THEN$_j$ β EXIT$_k$; ENDIF$_j$;

or

α_2 = IF$_j$ b THEN$_j$ β EXIT$_k$; ELSE$_j$ γ ENDIF$_j$;

d4) L_k ---> D(LOOP$_k$ α_1 IF$_j$) b-SW(β)

if

$\alpha_2 =$ IF$_j$ b THEN$_j$ γ ELSE$_j$ β EXIT$_k$; ENDIF$_j$;

are in $\mathcal{P}$.

Intuitively, $L(S)$ contains the set of all sequences which can be met during the execution of the scheme.

2. The complete paths in a Loop-Exit Scheme. An important problem in software development is program testing. Testing may be done starting from the specification of the resolved problem, or starting from the text of the program. In the second alternative it is important to know all the paths from the START block to the STOP block of the corresponding flow chart. For this purpose we introduce the notion of complete path in a Loop-Exit Scheme.

DEFINITION 2. *A word* $z = a_{i_1}X_{i_1}a_{i_2}X_{i_2}\ldots a_{i_s}X_{i_s}$ *is a section for S if and only if there is* $w \in L(S)$ *such that:*

a) $w = xzy$

b) $i_j < i_{j+1}$ *for* $j=1,2, \ldots, s-1$

c) *if* $x \neq \epsilon$ *then* $x = x'a_{i_0}X_{i_0}$ *with* $i_0 > i_1$

d) *if* $y \neq \epsilon$ *then* $y = a_{i_{s+1}}X_{i_{s+1}}y'$ *with* $i_s > i_{s+1}$.

The set of all sections is denoted by SEC(S).

The following theorem is proved in [2]:

THEOREM 1. *For each S we have $L(S) \sqsubset (\mathrm{SEC}(S))$.*

DEFINITION 3. *A word $z \in \mathrm{SEC}(S)$ is a branch for S if and only if there is $w \in L(S)$ such that $w=zy$. The set of all branches of S is denoted by* BRA(S).

Next, an algorithm to construct the set BRA(S) is given.

Algorithm 1. Which constructs the set BRA(S), has the following steps:

Step 1. The grammar G_1 has the productions obtained from the productions of $G(S)$ by replacing the productions

$$B_k \dashrightarrow \alpha B_k \mid \epsilon,$$

with the production $B_k \dashrightarrow \alpha$ and in all the other productions which have not this form the metasymbol B_k is replaced by ϵ.

Step 2. Putting off the inaccesible and unseful metasymbols of G_1 we obtained the grammar G_2 [1];

Step 3. The grammar G_3 is obtained from the grammar G_2 by replacing the productions of the form

$$L_k \dashrightarrow \alpha\ L_k$$

by the productions

$$L_k^a \dashrightarrow \alpha$$

where L_k^a is a new metasymbol associated to L_k ;

Step 4. The grammar G_4 is constructed from the grammar G_3 by adding to the productions of G_3 some new productions. If L_k is a recursive symbol in G_2 and $A \dashrightarrow \alpha\ L_k\ \beta$ is in G_3 then add the production $A \dashrightarrow L_k^a$ to G_4. Here L_k^a is the symbol associated

to L_k.

Step 5. One computes BRA(S) = $L(G_4)$.

To each metasymbol A of a grammar

$$G=(N,\Sigma,\mathcal{P},\nabla)$$

one can associate the grammar

$$G_A = (N,\Sigma,\mathcal{P},A)$$

which has the metasymbol A as the axiom.

If BRA(A) is the result of the application of the algorithm 1 to the grammar G_A then the following theorem holds [1].

THEOREM 3. *If S is a Loop-Exit Scheme then*

$$\mathrm{SEC}(S) = \mathrm{BRA}(S) \cup \{\mathrm{BRA}(A) \mid A \textit{ is recursive in } G_S^r\},$$

where G_S^r is the reduced grammar of the scheme S.

DEFINITION 4. *For each $xy^nz \in L(S)$ with $n \geq 0$ and $y \in \mathrm{SEC}(S)$ the words $w_1=xz$ and $w_2=xyz$ with $x=a_0x_1$ and $z=z_1a_f$ (i.e. which contains the assignments a_0 and a_f) are called complete paths of the Loop-Exit Scheme. The set of all complete paths of S is denoted by $CP(S)$.*

THEOREM 4. *Let G_p be the grammar obtained from G in the following way: if A is a recursive symbol in G and*

$$A \dashrightarrow \alpha A \mid \beta_1 \mid \beta_2 \ldots \mid \beta_k$$

are all the A-productions of G then the A-productions of G_p are

$$A \dashrightarrow \beta_1 \mid \beta_2 \ldots \mid \beta_k \mid \alpha\beta_1 \mid \alpha\beta_2 \ldots \mid \alpha\beta_k$$

The language generated by the grammar G_p generates CP(S).

The proof of this theorem follows imediatelly from the definition 4.

To ilustrate these we consider the following Loop-Exit Scheme:

```
$a_1$ $a_2$
LOOP$_1$
      IF$_1$ $a_3$ THEN$_1$ EXIT$_1$ ENDIF$_1$
      IF$_2$ $a_4$ THEN$_2$ $a_5$
              ELSE$_2$ $a_6$ $a_7$ $a_8$ ENDIF$_2$
ENDLOOP$_1$
$a_9$
```

The grammar $G(S)$ and the reduced grammar G_S^r are

$G(S)$	G_S^r
$v \dashrightarrow a_1\ a_2\ L_1\ a_9$	$v \dashrightarrow a_1\ a_2\ L_1\ a_9$
$L_1 \dashrightarrow I_1\ I_2\ L_1 \mid a_3+$	$L_1 \dashrightarrow I_1\ I_2\ L_1 \mid a_3+$
$B_1 \dashrightarrow I_1\ I_2\ B_1 \mid \epsilon$	$I_1 \dashrightarrow a_3-$
$I_1 \dashrightarrow a_3-$	$I_2 \dashrightarrow a_4+a_5 \mid a_4-a_6a_7a_8$
$I_2 \dashrightarrow a_4+a_5 \mid a_4-a_6a_7a_8$	

For this Loop-Exit Scheme we have

$$\mathrm{BRA}(S) = \{\ a_1a_2a_3+a_9\ ,\ a_1a_2a_3-a_4+a_5\ ,\ a_1a_2a_3-a_4-a_6a_7a_8\ \}$$

and

$$SEC(S) = BRA(S) \sqcup \{ a_3{+}a_9 \ , \ a_3{-}a_4{+}a_5 \ , \ a_3{-}a_4{-}a_6a_7a_8 \}$$

The grammar G_p has the following productions:

$\nabla \dashrightarrow a_1\ a_2\ L_1\ a_9$

$L_1 \dashrightarrow I_1\ I_2\ a_3{+} \ \mid\ a_3{+}$

$I_1 \dashrightarrow a_3{-}$

$I_2 \dashrightarrow a_4{+}a_5 \ \mid\ a_4{-}a_6a_7a_8$

and the set of the complete paths is

$$CP(S) = \{ a_1a_2a_3{+}a_9 \ , \ a_1a_2a_3{-}a_4{+}a_5a_3{+}a_9 \ , \ a_1a_2a_3{-}a_4{-}a_6a_7a_8a_3{+}a_9 \}.$$

3. Testing a Loop-Exit Program Scheme. Similarly to [6] any Loop-Exit Scheme becomes a Program Scheme if the assignments and test symbols are defined as follows.

Let

$$V = \{v_1, v_2, \ldots, v_m\} = I \sqcup W \sqcup O$$

be a set of variables, where I is the set of input variables, W is the set of working variables, and O is the set of the output variables. We may suppose, as in [9], that the set I, W and O are mutually disjoint. Let

$$F = \{f_1, f_2, \ldots, f_n\}$$

be a set of functional symbols. We suppose that each assignment $a \in A$ is of the form

$$v := f(y_1, y_2, \ldots, y_k)$$

where $f \in F$, $k \geq 0$, $y_1, y_2, \ldots, y_k \in I \sqcup W$, and $v \in W \sqcup O$.

Further, let

$$T = \{t_1, t_2, \ldots, t_r\}$$

be a set of test symbols. We suppose that each test symbol of the

Loop-Exit Scheme is of the form

$$t(y_1, y_2, \ldots, y_k)$$

where $t\epsilon T$, $k\geq 0$, and $y_1, y_2, \ldots, y_k \epsilon I \sqcup W$.

DEFINITION 5. *A Loop-Exit Scheme S is a Loop-Exit program Scheme if the symbols* $a\epsilon M$ *are defined as above, and for any* $w\epsilon L(S)$ *and any* $v\epsilon W$ *if* $w=w_1aXw_2$ *and a is of the form* $t(\ldots,v,\ldots)$ *or* $u{:=}f(\ldots,v,\ldots)$ *then there is* $a'\epsilon A$ *of the form* $v{:=}f(y_1, y_2, \ldots, y_k)$ *such that* $w=w'a'w''aXw_2$.

As an example, from the Loop-Exit Scheme given above we obtain the following Program Scheme:

```
d:=n1;  i:=n2;
LOOP_1
     IF_1 d=1 THEN_1 EXIT_1 ENDIF_1
     IF_2 d>i THEN_2 d:=d-i
            ELSE_2 t:=i; i:=d; d:=t ENDIF_2
ENDLOOP_1
div:=d
```

In other words, the definition 5 asks that any working variable is first initialized and then this variable may be used in computation.

The condition of the definition 5, taken from [6], is very strong. An example of a Loop-Exit Program Scheme which do not satisfy this condition but all variables receive their values before their use, is given in [4]. Also, in [4] is shown that a scheme S is a Program scheme if and only if this condition holds for any $z\epsilon$BRA(S). It follows that if a variable does not satisfy this condition for every $z\epsilon$BRA(S) then it is certainly an uninitialised variable. This fact is very important for the verification of the program corectness. Also, it is important for the programmer to be informed about all the uninitialised variables on some branches of the program.

Testing a program [7] means to observe the results obtained if the program is run for some testing data. A run is needed for each complete path. Therefore, for program testing it is very important to know all of its complete paths.

Knowing a complete path is also useful for choosing the coresponding testing data. If the input variables receives these data the program follows this path. That is, all test conditions met in this path are satisfied.

R E F E R E N C E S

1. Aho A.V., Ullman J.D., *The theory of Parsing, Translation and Compiling*, Prentice Hall Inc., 1972-1973.
2. Boian F.M., *Sisteme conversaţionale pentru instruire în programare*, Teză de doctorat, Cluj-Napoca, 1986.
3. Boian F.M., *Loop-Exit Schemes and Grammars: Properties Flowchartablies*, Studia Universitatis "Babeş-Bolyai", Math.(1986), n0.3, pp. 52-57.
4. Boian F.M., M.Frenţiu, and Z.Kasa, *Parallel execution in Loop-Exit Schemes*, Seminar on Computer Science, Preprint no.9, 1988, pp.3-16.
5. Floyd,R.W. (1967), *Assigning meanings to programs*, in Proc. Symposium App.Math., XIX,(J.T.Schwartz ed.), Providence, Am.Math.Soc.
6. Greibach S., *Theory of Program Structures: Schemes, Semantics, Verification* (Lecture Notes in Computer Science), Springer-Verlag, 1975.
7. J.C.King, *Symbolic Execution and Program Testing*, Comm. ACM, 19 (1976), 7, 385-394.
8. S.Katz, Z.Manna, *Logical Analysis of Programs*, CACM 19(1976), 4, 188-206.
9. Manna, Z. (1974) *Mathematical Theory of Computation*, New York: McGrawHill.

STUDIA UNIV. BABEŞ-BOLYAI, MATHEMATICA, XXXVII, 3, 1992

PARALLEL PRE-PROCESSING IN EXTRAPOLATION METHODS FOR SOLVING ORDINARY DIFFERENTIAL EQUATIONS

O. BRUDARU[*] and G.M. MEGSON[**]

Received: December 1, 1992
AMS Subject Classification: 68Q22, 58F99

Abstract: - The work deals with the parallel implementation of the pre-processing stage of the polynomial/rational extrapolation techniques for solving initial value problems in ordinary differential equations. The multiple use of the Euler method is analysed in the case of three step number sequences (n_k): $n_k=ak+b$, $n_k=2{*}{*}k$, and $n_{k+1}=n_k+n_{k-1}$, $k=0,..,K$. We make use of the PRAM model of parallel computation and propose a specific parallel algorithm for each type of sequence. It is assumed that the number of processors is a factor of K. The performances concerning the parallel processing time and the effectiveness of processor utilization are established. Some conditions ensuring the optimality of the proposed algorithms are also given.

Keywords: ordinary differential equations, polynomial and rational extrapolation, parallel algorithms.

1. Introduction. The parallel implementation of the polynomial and rational extrapolation techniques ([10], [12-13]) for solving initial value problems in ordinary differential equations (ODE's) is discussed in [8] where two types of systolic arrays are proposed.

We consider here a complementary task, namely, the design of parallel algorithms for computing the initial data input to the systolic arrays in [8].

The stages involved by the extrapolation methods for solving initial value problems in ODE's are described in section 2. In section 3, we present three parallel algorithms to compute the data entering the extrapolating process, and consider Euler's

[*] Universitatea "Al. I. Cuza", Seminarul Matematic "A. Myller" 6600-Iaşi, Romania

[**] Computing Laboratory, University of Newcastle-Upon-Tyne Claremont Tower, Claremont Rd, Newcastle-Upon-Tyne, NE1 7RU, U.K.

method applied to three types of step number sequences. The performances concerning the parallel processing time and the effectiveness of processor utilization are established. Some comments are given in the last section.

2. Extrapolation methods for solving ODE's. Consider the initial value problem

$$y'=f(t,y),\quad t_0\le t\le b,\quad y(t_0)=y_0; \tag{2.1}$$

and let us suppose that we need to compute $y(t_0+H)$, $t_0+H\le b$. Let $y(t)$ be the true solution of (2.1) and $y(t,h)$ be the approximate solution obtained by using step length h, $h\le H$, and a suitable numerical method which is applied many times. Let us take some step-number sequence $n_0<n_1<n_2<\ldots$, put $h_k=H/n_k$ and define

$$Y(k,0)=y(t_0+H,h_k), \tag{2.2}$$

the numerical solution obtained by performing n_k steps with step size h_k, $k=0,2,\ldots,K$. The computation of the $Y(k,0)$-values represents the pre-processing stage of the extrapolation.

Let us suppose that $y(t,h)$ admits an asymptotic expansion in h of the following form

$$y(t,h)=y(t)+d_1h^{r(1)}+d_2h^{r(2)}+\ldots+d_mh^{r(m)}+\ldots \tag{2.3}$$

where $0<r(1)<r(2)<\ldots<r(m)$, $d_1,d_2,\ldots$ are independent of h.

2.1. Polynomial and rational extrapolation. If we consider $r(i)=ir$ in (2.3) then following [13] the polynomial extrapolation involves the computing of the Y-values table given by

$$Y(k,m)=Y(k+1,m-1)+[Y(k+1,m-1)-Y(k,m-1)]/[(n_{k+m}/n_k)^r-1] \tag{2.4}$$

for $k=0,1,\ldots,K-m$ and $m=1,2,\ldots,K$. The computation given by (2.4) is represented (for $K=5$ and $m=5$) in Figure 1.

For $n_k=H/(h_0b^k)$, $b\epsilon(0,1)$, (2.4) becomes

$$Y(k,m)=Y(k+1,m-1)+[Y(k+1,m-1)-Y(k,m-1)]/[1/b^{mr}-1], \qquad (2.5)$$

$k=0,1...,K-m$ $m=1,2,...,K$.

Usually, $r=1$ or $r=2$. The process is stopped ([13]) when

$$\text{abs}(Y(0,m)-Y(0,m-1))\leq\text{tol}, \qquad (2.6)$$

where tol is a prescrbied tolerance and $Y(0,m)$ approximates $y(t_0+H)$.

The rational extrapolation ([10]) is defined by

$$R(k,-1)=0\ ;\ R(k,0)=Y(k,0)\ ; \qquad (2.7)$$

$$R(k,m)=R(k+1,m-1)+[R(k+1,m-1)-R(k,m-1)]/\{(h_k/h_{k+m})^2[1-(R(k+1,m-1)-R(k,m-1))/(R(k+1,m-1)-R(k+1,m-2))]-1\} \qquad (2.8)$$

for $m\geq1$. This scheme is illustrated (for $K=5$) in Figure 2. The process is stopped ([10]) when

$$\text{abs}(R(k-m,m)-R(k-m+1,m))<\text{tol} \qquad (2.9)$$

for some K and m. If (2.9) holds then the result $R(k-m+1,m)$ could be accepted.

2.2. Basic pre-processing methods. In the case of non-stiff equations ([10], [13]), for r=1 the basic method to compute the $Y(k,0)$-values is the Euler method

$$y_{i+1}=y_i+h_kf(t_i,y_i),t_{i+1}=t_i+h_k,\ i=0,...,n_k-1, \qquad (2.10)$$

$$Y(k,0)=y_{nk}.$$

For $r=2$, a numerical integration formula for which (2.3) holds is the second order Gragg's method described by

$$z(t_0,h_k)=y_0, \qquad (2.11a)$$

$$z(t_1,h_k)=y_0+h_kf(t_0,y_0);$$

$$t_i=t_0+ih_k, \quad z(t_{i+1},h_k)=z(t_{i-1},h_k)+2h_kf(t_i,z(t_i,h_k)); \qquad (2.11b)$$

$$i=1,\ldots,n_{k-1}:$$

$$Y(k,0)=[z(t_nk-1,h_k)+z(t_nk,h_k)+h_kf(t_nk,z(t_nk,h_k))]/2$$

Extrapolation of implicit methods can be used for stiff equations. Some symmetric and non-symmetric methods which are of interest are described in [12]. In this case, for each $Y(k,0)$-value, we must solve n_k nonlinear equations, and consequently the computing time cannot be predicted. However, a lower bound to this time could be given by the time needed by Gragg's method. In what follows this lower bound will be used instead of the exact time.

3. Parallel pre-processing algorithms. This section deals with the parallel computation of $Y(k,0)$, $k=0,\ldots,K$. We consider the case of explicit methods. The entire computational diagraph of the polynomial extrapolation is illustrated for $K=5$ in Figure 3. The computation required by $Y(k,0)$ is represented by the sequence of vertices denoted by seq(k). Each vertex in this sequence represents just one step of the Euler method, while an arc (v,v') denotes the sending of the y-value from v to v'. The first vertex in each sequence requires Y_0 as the starting value.

For each fixed k, $Y(k,0)$ is obtained by performing n_k steps with the step size h_k. It is reasonable to suppose that each step requires an amount of time which depends on the applied explicit method and the complexity in evaluating $f(t,y)$, and does not depend on the step size h_k.

Let $t(O)$ be the time to perform the operation $O\epsilon(+,*,/)$ and define the time unit (TU) as $t(/)$. Let us denote by c the time to execute just one step of the Euler's method (2.10). The dominant part of c is represented by the time to compute f. In the case of formula (2.10), $Y(k,0)$ requires cn_k TUs.

If Gragg's method is used then $Y(k,0)$ is computed in $c+(n_k-1)[c+t(+)]+(c+t(+)+t(*))=n_k+1$ TUs, because $t(+)<<c$. Therefore, the use of the Gragg's method for computing the initial Y-values, can be studied by considering $n_k=n_k+1$. In the case of $n_k=ak+b$ this can be simply done by taking $b:=b+1$. Some problems arise for $n_k=2^k$ and $n_{k+1}=n_k+n_{k-1}$. In each case, n_k exponentially increases and n_k does not differ significantly from n_k. So, we can reduce the discussion to Euler's method applied to the above step number sequences. Also, we can suppose that $c=1$.

We make use of the idealized model of parallel computation known as the PRAM ([11]). We shall suppose that S processors, P_s, $s=1,...,S$, are available. We consider the parallel execution of the sequences seq(k), $k=0,...,K$, where $K+1>S$. The processors execute the same program implementing Euler's method for the same function and initial condition, but for different step sizes and number of steps. We remark that the use of the PRAM model is motivated by the necessity to consider functions of arbitrary complexity and not by the intrinsec structure of the numerical method computing the $Y(k,0)$-values. We note that if f is a polynomial both in t and y, or is given by a small size arithmetic expression including elementary functions, then a soft/hard-systolic implementation is also possible ([2-6], [14-15], [17-18]).

3.1. Case $n_k=ak+b$. Let Q and Q_0 be two positive integers so that $K+1=(2S)Q+Q_0$, with $Q_0<2S$.

The $K+1$ sequences are organized in Q bands $B(q), q=1,\dots,Q$, of 2S succesive sequences and the band $B(Q+1)$ of Q_0 sequences. The bands $B(q)$, $q=1,\dots,Q+1$, are processed in a serial fashion, while the sequences of each band are processed in parallel, as it is illustrated in Figure 4 for $S=4$. The q-th band is formed by seq$((q-1)2S+j-1)$, $j=1,\dots,2S$, $q=1,\dots,Q$.

The processors $P_1,\dots,P_S$ begin the execution of $B(q)$ at the same time, say ts(q). P_r executes seq$((q-1)2S+r-1)$ and then seq$((q-1)2S+2S-r)$, $r=1,\dots,S$, and this computation takes $Tp(q)=t((q-1)2s+r-1)+t((q-1)2s+2s-r)=a[4(q-1)s+2s-1]+2b$ TUs.

This time does not depend on r, thus the processors terminate the processing of $B(q)$ at the same time, $ts(q)+Tp(q)$. A serial algorithm takes $Ts(q)=S*Tp(q)$, then the efficiency of processor utilization is $Ec(q)=Ts(q)/(STp(q))=1$ i.e. the strategy to process each band is optional.

The execution of $B(q+1)$ starts at $ts(q+1)=ts(q)+Tp(q)$ and is done in the same manner. If $Q_0=0$, then the parallel processing time for Q bands is

$$Tp=ts(1)+Tp(1)+\dots+Tp(Q)=Q[2b+a(2S-1)+2aS(Q-1)], \qquad (3.1)$$

for $ts(1)=0$. The total time required by a serial algorithm is

$$Ts=Ts(1)+\ldots+Ts(Q)+S[Tp(1)+\ldots+Tp(Q)]=S*Tp, \qquad (3.2)$$

therefore the global efficiency $Ec=1$.

Now, let us suppose that $Q\neq 0$ and $B(Q+1)$ is processed like the previous bands.

First, if $1\leq Q_0\leq S$ we use only Q_0 processors to process $seq(2QS-1+j)$, $j=1,\ldots,Q_0$. Therefore

$$Tp'(Q+1)=t(2QS-1+Q_0)=t(K)=aK+b \ TUs,$$

while

$$Ts'(Q+1)=t(2QS)+t(2QS+1)+\ldots+t(2QS+Q_0-1)=$$

$$=[a(2QS-1)+b]Q_0+aQ_0(Q_0+1)/2.$$

Thus, we obtain that the effectiveness of processor utilization for the last band is

$$Ec'(Q+1)=1-a(Q_0-1)/[2(aK+b)].$$

On the other hand,

$$Tp'=Tp+Tp'(Q+1),$$

$$Ts'=Ts+Ts'(Q+1),$$

and from (3.1) and (3.2) we obtain the effectiveness of processor utilization for the entire process as

$$Ec'=Ts'/(STp')=$$

$$=1-[aQ_0^2/2-Q_0(aK+b+a/2)+S(aK+b)]/S(Tp+aK+b)]<1,$$

and Ec' has the greatest value for $Q_0=S$ when

$$Ec'=1-a(S-1)/[2(Tp+aK+b)].$$

Second, if $S+1\leq Q_0\leq 2S-1$, we have that S processor execute the sequences $seq(2QS-1+j)$, $j=1,\ldots,Q_0$. Let us take $Q_\delta\in(1,\ldots,S-1)$, so that $Q_0=S+Q_\delta$. For the sake of regularity we use the same strategy for peocessing this last band. Therefore, P_j performs only $seq(2SQ-1+j)$, $j=1,2,\dots,S-Q_\delta$ while P_h executes $seq(2QS-1+h)$ and then $seq(2QS+2S-h)$, $h=S-Q_\delta+1,\dots,S$. Consequently, the parallel processing time for the last band is

$$Tp''(Q+1)=t(2QS-1+S)+t(2QS+S)=a(4QS+2S-1)+2b \text{ TUs},$$

while the corresponding sequential time is

$$Ts''(Q+1)=Ts'(Q+1)$$

and we obtain that

$$Ec''(Q+1)=(Q_0/S)[a(K+1-Q_0-1)+b+a(Q_0+1)/2]/[a(2(K+1-Q_0)+2S-1)+2b].$$

In this case, the total parallel processing time is

$$Tp''=Tp+Tp''(Q+1),$$

and the corresponding sequential time is

$$Ts''=Ts+Ts''(Q+1),$$

and from (3.1) and (3.2), the obtained effectiveness of processor utilization for the entire process is

$$Ec''=Ts''/(S*Tp'')=1-\{a[-2S^2(1+2Q)+S(1+2QQ_0)+Q_0(Q_0-1)/2]+b(-2S+Q_0)\}/$$
$$[S[Tp+a(4QS+2S-1)+2b]\}.$$

As a conclusion of the above analysis we can state

THEOREM 1. *Under the above assumptions, if $n_k=ak+b$, $k=0,\dots K$ and S processors are used to compute $Y(k,0)$, $k=0,\dots,K$, then the following assertions are true:*

(i) *if $K+1=2SQ$ then the algorithm is optional with respect to processor utilization and the parallel processing time is*

$$Tp=Q[2+a(2S-1)+2aS(Q-1)];$$

(ii) *if $Q_0=K+1-2SQ$ and $1\le Q_0\le S$ then the parallel processing time is*

$$Tp'=Tp+aK+b$$

and the effectiveness of processor utilization is

$Ec'=1-[aQ_0^2/2-Q_0(aK+b+a/2)+S(aK+b)]/[S(Tp+aK+b)]$

$\leq 1-a(S-1)/[2(Tp+aK=b)]$,

while the equality holds for $Q_0=S$;

(iii) *if* $Q_0=K+1-2SQ$ *and* $S+1\leq Q_0\leq 2S-1$ *then the parallel processing time is*

$$Tp''=Tp+a(4QS+2S-1)+2b,$$

and the effectiveness of processor utilization is

$Ec''=1-\{a[-2S^2(1+2Q)+S(1+2QQ_0)+Q_0(Q_0-1)/2]+b(-2S+Q_0))/$

$\{S[Tp+a(4QS+2S-1)+2b]\}$.

The above discussion could be extended in the following way. Let m be a positive integer, $1<m<K+1$, with $K+1=m*N$, and define $C_p=\{seq(p+jm)/j=0,\ldots,N-1\}$, $p=0,\ldots,m-1$. Also, consider the positive integers a_p and b_p, $p=0,\ldots,m-1$. A mixed pre-processing strategy is to compute $Y(k,0)$ in accordance to the step number sequence a_p*k+b_p, as it is required in the case of stiff equations ([12]). A desirable property of the parallel processing scheme is that whenever $k<k'$, the $Y(k,0)$-value is obtained before $Y(k'.0)$, because in this case the stopping condition (2.6) can be efficiently used to save time and hardware. On the other hand, the precedence constraints appearing in the extrapolation stage do not require such a property (see Figures 1-2). It results that we could relaxe the above condition by requiring that it holds only for a given number of succesive $Y(k,0)$-values. So, an acceptable compromise is to assign $seq(p+jm)$, $p=0,\ldots,m-1$, to processor p_{j+1}, $j=0.,,,.S-1$, $S>N$, and to continue with P_1,

$P_2, \ldots$ for $j=S, S+1, \ldots$ and so on. This solution is compatible with the necessity to adopt a local synchronization technique ([19], [16]) for interfacing the set of S processors and the systolic arrays in [8].

3.2. Case $n_k=2^k$. If S processors are available to compute seq(k), $k=0,\ldots,K$, the obtained efficiency is

$$Ec(S)=N(K)/(S*Tp(S)),$$

where $N(k)=n_0+\ldots+n_k$ and $Tp(S)$ is the corresponding parallel processing time. Since $N(k)=2n_k-1$ and

$$Tp(S)\geq\max\{n_k/k=0,1,\ldots,K\}=n_K,$$

we obtain

$$Ec(S)=(2n_K-1)/(Sn_K)\leq(2n_K-1)/(2n_K).$$

This upper bound does not depend on S and because

$$Ec(2)=(2n_K-1)/(2n_K)$$

we conclude that it is fruitless (from the efficiency point of view) to use more than two processors to execute seq(k), $k=0,1,\ldots,K$. If the processors are P_1 and P_2, then P_1 executes seq(k), $k=0,\ldots,K-1$ in $N(K-1)$ *TUs* and P_2 performs seq(K) in $n_K=N(K-1)+1$ *TUs* as it is illustrated in Figure 5. If we need to extend the activity of P_i, $i=1,2$, to seq(k), $k=K+1,\ldots,K+R+1$, and P_1 executes seq(k), $k=K+1,\ldots,K+R$, while P_2 acts on seq($K+R+1$), then the obtained efficiency is

$$Ec(2)=n_{K+1}N(R)/(2n_{K+R+1})=1-1/2^{R+1}.$$

Now, P_1 works in $n_{K+1}N(R-1)=2^{K+R+1}-2^{K+1}$ *TUs*, while P_2 needs n_{K+R+1} *TUs*.

These results suggest the following strategy. Let us suppose that $S=2S'$ and the sequence seq(k), $k=0,\dots,K$ are divided into M bands $B(m)$, $m=1,\dots,M$ of equal size, and band $B(m)$ is divided into S' subbands $SB(m,s)$, $s=1,\dots,S$ of equal size R, i.e. $K+1=MS'R$. Therefore B(m) contains seq$((m-1)RS'+j-1)$, $j=1,\dots,RS'$, and $SB(m,s)$ consists of seq$((m-1)RS'+(s-1)R+r-1)$, $r=1,\dots,R$, $s=1,\dots,S'$, $m=1,\dots,M$. The processing begins with $B(1)$ so that the S' pairs of processors start at the same time. The s-th pair of processors, say $(P(s,1),P(s,2))$, acts on $SB(1,s)$, $s=1,\dots,S'$ so that $P(s,1)$ executes the first $R-1$ sequences and $P(s,2)$ the R-th sequence. As soon as $P(s,j)$ terminates its work for $SB(m,s)$ it passes to the corresponding sequence(s) of $SB(m+1,s)$, and so on. This strategy is illustrated in Figure 6. It results that $P(s,1)$ executes seq$((m-1)RS'+(s-1)R+r-1)$, $r=1,\dots,R-1$, while $P(s,2)$ acts on seq$((m-1)RS'+sR-1)$, $m=1,\dots,M$, $s=1,\dots,S'$. Let t_0 be the time when $P(s,j)$, $s=1,\dots,S'$, $j=1,2$, start the activity for $B(1)$. Also, let us denote by $tf(k)$ $(tin(k))$ the time when the execution of *seq*(k) is terminated (initiated). Clearly, for each $s\epsilon\{1,\dots,S''\}$, from the activity of $P(s,2)$, we have that

$$tf(sR-1)=t_0+2^{sR-1},$$

$$tf((h-1)RS'+sR-1)=tf((h-2)RS'+sR-1)+2^{(m-1)RS'+sR-1},\quad h=2,\dots,m,$$

and consequently, we obtain

$$tf((m-1)RS'+sR-1)=t_0+2^{sR-1}(2^{mRS'}-1)/(2^{RS'}-1),\quad m=1,\dots,M.$$

On the other hand,

$$tin((m-1)RS'+sR-1)=tf((m-1)RS'+sR-1)-2^{(m-1)RS'+sR-1}=$$
$$tf((m-2)RS'+sR-1), \quad m=1,\ldots,M.$$

Now, let us analyse the activity of $P(s,1)$, $s\epsilon\{1,\ldots,S'\}$. This processor executes in the order $((\text{seq}((m-1)RS'+(s-1)R+r-1), r=1,\ldots,R-1), \quad m=1,\ldots,M)$. Let $TB(s,m)$ be the time in which $P(s,1)$ executes the sequences belonging to $B(m)$, $m=1,\ldots,M$. Clearly,

$$TB(s,m) = \sum_{r=1}^{R-1} 2^{(m-1)RS'+(s-1)R+r-1} =$$
$$= (2^{R-1}-1)\ 2^{(m-1)RS'+(s-1)R}$$

Also, let $TF(s,m)$ be the time when the last sequence of $B(m)$ associated to $P(s,1)$ is terminated. Clearly,

$$TF(s,1)=t_0+TB(s,1),$$
$$TF(s,h)+TF(s,h-1)+TB(s,h),\ h=1,\ldots,m.$$

Thus,

$$TF(s,m) = t_0 + \sum_{h=1}^{m} TB(s,h)$$

$$t_0+2^{(s-1)}\ (2^{R-1}-1)(2^{mRS'}-1)/(2^{RS'}-1).$$

If $TS(s,m)$ denotes the time when $P(s,1)$ begins the execution of its first sequence from $B(m)$, then $TS(s,m)=TF(s,m)-TB(s,m)$.

For a better analysis, we compute $tf((m-1)RS'+(s-1)R+r-1)$, $r=1,\ldots,R-1$, $m=1,\ldots,M$. Clearly, we have while

$$tin((m-1)RS'+(s-1)R)=TS(s,m)$$

$$tf((m-1)RS'+(s-1)R+r-1)=TS(s,m)+\sum_{h=1}^{r} 2^{(m-1)RS'+(s-1)R+h-1} \qquad (3.3)$$

$$=TS(s,m)+(2^{r}-1)2^{(m-1)RS'+(s-1)R},$$

and

$tin((m-1)RS'+(s-1)R+r-1)=tf((m-1)RS'+(s-1)R+r-2)$, $r=2,\dots,R-1$.

Now, let us compute the effectiveness of processor utilization, $Ec(S',R)$.

The time Ts required by a sequential algorithm is

$$Ts=n_0+\dots+n_K=2^{K+1}.$$

The number $Np=2S'$ of processors leads to the parallel processing time Tp whose value is obtained by analysing the activity of (i) $p(s,1)$, $s=1,\dots,S'$ and (ii) $P(s,2)$, $s=1,\dots,S'$.

(i) From (3.3) we obtain the equivalent form of the final time $tf((m-1)RS'+(s-1)R+r-1)=t_0+(2^{R-1}-1)(2^{mRS'}-1)2^{(s-1)R}/(2^{RS'}-1)-$

$$(2^{R-1}-1)2^{(m-1)RS'+(s-1)R}+$$

$$(2^{r}-1)2^{(m-1)RS'+(s-1)R},$$

$$r=1,\dots,R-1,\ m=1,\dots,M.$$

The last task executed by $P(s,1)$ is obtained for $m=M$ and $r=R-1$ and has the ending time $tf_{s,1}$ given by

$$tf_{s,1}=tf((M-1)RS'+(s-1)R+R-2)=$$

$$t_0+(2^{R-1}-1)(2^{K+1}-1)2^{(s-1)R}/(2^{RS'}-1),$$

because $K+1=MRS'$.

(ii) The last task executed by $P(s,2)$ has an end time $tf_{s,2}$ obtained from $tf((m-1)RS'+sR-1)$ by taking $m=M$, i.e.

$tf_{s,2}=t_0+(2^{K+1}-1)2^{sR-1}/(2^{RS'}-1)$.

But

$Tp(S',R)=\max\{tf_{s,1},tf_{s,2}/s=1,\dots,S'\}-t_0=\max\{tf1,tf2\}$, where

$$tf1=\max\{tf_{s,1}/s=1,\ldots,S'\}-t_0=tf_{S',1}-t_0=$$
$$(2^{R-1}-1)(2^{K+1}-1)2^{(S'-1)R}/(2^{RS'}-1),$$

while

$$tf2=\max\{tf_{s,2}/s=1,\ldots,S'\}-t_0=tf_{S',2}-t_0=$$
$$(2^{K+1}-1)2^{RS'-1}/(2^{RS'}-1),$$

and finally, we obtain

$$Tp(S',R)=(2^{K+1}-1)2^{RS'-1}/(2^{RS'}-1).$$

Therefore,

$$Ec(S',R)=Ts/(NpTp(S',R))=(1-1/2^{RS'})/S'. \tag{3.4}$$

As it was expected, by taking $S'=1$ and $R=K+1$ in (3.4), it results

$$Ec(1,K+1)=1-1/2^{K+1}$$

and

$$Ec(1,K+1)\geq EC(S',R)S'.$$

Also, for $M=1$ it holds $Ec(1,K+1)=Ec(S',R)S'$.

On the other hand, $Tp(1,K+1)=2^K$ and

$$Tp(S',R)/Tp(1,K+1)=(1-1/2^{K+1})/(1-1/2^{(K+1)/M}). \tag{3.5}$$

By taking $M=1$ in (3.5), $Tp(S',R)=Tp(1,K+1)$ results. If $M>1$ consider $f(x)=(1-x^M)/(1-x)$, with $0<x<1/2$. A simple calculation shows that $1<f(x)<2-1/2^{M-1}$ and therefore we obtain that

$$Tp(1,K+1)<Tp(S',R)<Tp(1,K+1)(2-1/2^{M-1}).$$

In fact, we have proved the following

THEOREM 2. *Under the above assumptions if $n_k=2^k$, $k=0,\ldots,K$, and S processors are used to compute $Y(k,0)$, $k=0,\ldots,K$, then the following assertion are true:*

(i) *if $Ec(S)$ is the effectiveness of processor utilization* *for an arbitrary parallel algorithm then* $Ec(S) \leq Ec(2)$, *for* $S \geq 2$;

(ii) *if* $S=2S'$ *and* $K+1=MRS'$ *and the above strategy is used then*

(ii.i) *the parallel processing time is*

$$Tp(S',R)=(2^{K+1}-1)2^{RS'-1}/(2^{RS'}-1);$$

(ii.ii) *the effectiveness of processor utilization is*

$$Ec(S',R)=(1-1/2^{RS'})/S';$$

(ii.iii) *if* $M=1$ *then*

$$Tp(1,K+1)=Tp(S',R)$$

and

$$Ec(1,K+1)=Ec(S',R)S';$$

(ii.iv) *if* $M>1$ *then*

$$Tp(1,K+1)<Tp(S',R)<Tp(1,K+1)(2-1/2^{M-1})$$

and

$$Ec(1,K+1)>Ec(S',R)S'.$$

3.3. Case $n_{k+1}=n_k+n_{k-1}$. Let us consider the case when (n_k) is a Fibonacci sequence, for some prescribed n_0 and n_1. The reason to consider this sequence is that it is possible to determine a, b, n_0 and n_1 such that $ak+b \leq n_k \leq 2^{**}k$ holds for $k \geq k_0$, where k_0 is a prescribed rank. We suppose again that $S=2S'$. We split the set of $K+1$ sequences into S' bands $B(s)$, $s=1,\ldots,S'$. The band $B(s)$ splits into the subbands $SB(s,r)$, $r=1,\ldots,R$, where $SB(s,r)$ consists of three succesive sequences, i.e. $K+1=3RS'$ and $SB(s,r)=\{seq((s-1)3R+3(r-1)+h)/h=0,1,2\}$ as it is shown in Figure 7(a). The pair of processors $(P(s,1),P(s,2))$ acts on $B(s)$, $s=1,\ldots,S'$ and all pairs begin the activity at the same time t_0

(Figure 7(b)). The processor $P(s,1)$ executes the sequences $((seq((s-1)3R+3(r-1)), seq((s-1)3R+3(r-1)+1), r=1,\dots,R)$ in this order, while $P(s,2)$ executes $(seq((s-1)3R+3(r-1)+2), r=1,\dots,R)$ in this order. Therefore $P(s,1)$ needs $t(s,1)$ time, where

$$t(s,1) = \sum_{r=1}^{R} (n_{(s-1)3R+3(r-1)} + n_{(s-1)3R+3(r-1)+1}),$$

while the time required by $P(s,2)$ is

$$t(s,2) = \sum_{r=1}^{R} n_{(s-1)3R+3(r-1)+2}, \tag{3.6}$$

and because (n_k) is a Fibonacci sequence it results that $t(s,1)=t(s,2)$. Therefore, the time to execute $B(s)$ is $t(s,2)$ and the parallel processing time is

$$Tp=\max\{t(s,2)/s=1,\dots,S'\}=t(S',2)=$$

$$\sum_{r=1}^{R} n_{(s'-1)3R+3r-1}TUs.$$

If q_1 and q_2 are the roots of the equation $x^2-x-1=0$, then

$$n_k=c_1q_1{}^k+c_2q_2{}^k, \tag{3.7}$$

where c_j, $j=1,2$, are determined by n_0 and n_1. Using this form of n_k and taking into account that $q_j{}^3-1=2q_j$, $j=1,2$, from (3.6) we obtain

$$\begin{aligned} t(s,2) &= \sum_{i=1}^{2} c_i \sum_{r=1}^{R} q_i^{(s-1)3R+3(r-1)+2} = \\ & 0.5 \sum_{i=1}^{2} c_i q_i^{(s-1)3R+1}(q_i^{3R}-1). \end{aligned} \tag{3.8}$$

Therefore, it is obtained that

$$Tp=0.5\sum_{i=1}^{2} c_i q_i^{(S'-1)3R+1}(q_i^{3R}-1)\,.$$

On the other hand, the time required by a sequential algorithm is

$$Ts = 2\sum_{s=1}^{S'} t(s,2)\,,$$

and from (3.8) we produce

$$Ts = \sum_{i=1}^{2} c_i q_i (q_i^{K+1}-1)\,,$$

while from (3.7) and the fact that $K+1=3RS'$, we obtain

$$Tp=0.5(n_{K+2}-n_{K+2-3R})$$

and

$$Ts=n_{K+2}-n_1.$$

Thus, the effectiveness of processor utilization is

$$Ec=Ts/(S*Tp)=3R(n_{K+2}-n_1)/[(K+1)(n_{K+2}-n_{K+2-3R})].$$

Let us remark that $S'=1$ yields $k+1=3R$ and $n_{K+2-3R}=n_1$, consequently $Ec=1$.

As a consequence of the above analysis we can state

THEOREM 3. *Under the above assuptions, if $\{n_k\}$ is a Fibonacci sequence, a number of $S=2S'$ processors are used to compute $Y(k,0)$, $k=0,\ldots,K$, and the above parallel algorithm is used, then:*

(i) *the parallel processing time is*

$$Tp=0.5(n_{K+2}-n_{K+2-3R})\ TUs;$$

(ii) *the effectiveness of processor utilization is*

$Ec=3R(n_{K+2}-n_1)/[(K+1)(n_{K+2}-n_{K+2-3R})]$;

(iii) *if* $S=2$ *then the parallel algorithm is optimal with respect to the processor utilization.*

Now, we are able to give a better motivation to our work. A serial implementation of the extrapolation stage requires $O(K^2)$ amount of time. The pre-processing stage needs $O(cK^2)$ time for $n_k=ak+b$, and $O(cq^K)$ time, where $q=2$ for $n_k=2^k$ and $q=\max(q_1,q_2)$ for Fibonacci sequence, where c includes the time complexity in the evaluation of f. The comparison still indicates that a parallel approach of the pre-processing stage is well suited. A similar situation appears in the case of Romberg's extrapolation method for numerical integration ([1], [7]).

4. Final remarks. There are two contradictory aspects in any attempt to parallelize the extrapolation methods for solving initial value problema in ODE's.

The first aspect refers to the adaptive feature of the method which allows us to stop the first stage computation as soon as it is obtained the desired accuracy. A considerable amount of time can be saved in this way. Clearly, the best way to accomplish the adaptive task is to use a serial algorithm, which starts the computation of $Y(k,0)$ value only if $Y(h,0)$, $h=0,\ldots,k-1$, do not lead to a true value of the convergence test.

The second aspect concerns the fact that if we wish to obtain a short parallel computing time, then the parallel algorithm must anticipate the computation of some $Y(k,0)$-values before knowing that these values are necessary or not to obtain the desired tolerance. If they aren't then an useless computation was performed, and this comes as a price. In compensation, a certain gain of accuracy could be obtained if the extrapolation stage continues while it consumes the $Y(k,0)$ values whose computation was already started and does not need much time to be finished.

The proposed algorithms accomplished a serial processing of the bands, while the parallel processing addresses to the tasks within of each band. For this reason, they seem to be a good compromise between opposing restrictions.

R E F E R E N C E S

1. Brudaru, O., *Systolic arrays for numerical integration with Romberg's formula,* Analele Ştiinţifice ale Universităţii "Al. I. Cuza" din Iaşi, Informatica, no. 35, 1989, pp. 367-374.
2. Corbaz, G., Duprat, J., Hochet, B., Muller, J.M.,*Implementation of VLSI polynomial evaluator for real-time applications,* RR 91-07, LIP, ENSL, France, 1991.
3. Darte, A., Risset, T., Robert, Y., *Synthetizing systolic arrays: some recent developments,* RR 91-09, LIP, ENSL, France, 1991.
4. Duprat, J., Muller, J.-M., *Evaluation of polynomials and elementary functions by integrated circuits,* RR 698-I, TIM3, IMAG, France, 1988.
5. Duprat, J., Muller, J.-M., *Hardwired polynomial evaluation,* Journal of Parallel and Distributed computing, no. 5, 1988, pp. 291-309.
6. Evans, D. J., Margaritis, K., Bekakos, M.P., *Systolic and holographic pyramidal soft-systolic designs for succesive matrix powers,* Parallel Computing, no. 9, 1989, pp. 373-387.
7. Evans, D.J., Megson, G.M., *Romberg integration on systolic arrays,* Parallel Computing, no. 3, 1986, pp. 289-304.
8. Evans, D.J., Megson, G.M., *Construction of Extrapolation Tables by Systolic Arrays for Solving Ordinary Differential Equations,* Parallel Computing, 410.1, 1987, pp. 33-48.
9. Fisher, A.L., Kung, H.T., *Synchronizing large VLSI processor arrays,* IEEE Transactions on Computers, vol. 34, no. 8, 1985, pp. 734-740.
10. Gear, W.G., *Numerical Initial Value Problems in Ordinary Differential Equations,* Prentice-Hall, Inc., Englewood Cliff, New Jersey, 1971.
11. Gibons, A., Rytter, W., *Efficient Parallel Algorithms,* Cambridge University Press, Cambridge, 1988.
12. Hairer, E., Wauner, G., *Solving Ordinary Differential Equations II-Stiff and Differential-Algebraic Problems,* Springer Verlag, Berlin, 1991.
13. Jain, M.K., *Numerical Solution of Differential Equation,* Wiley Eastern Limited, New Delhi, 1984.
14. Kung, H.T., *Let's Design Algorithms for VLSI Systems,* Technical Report CMU-CS-79-151, Carnegie-Mellon University, Computer Science Departament, Pittsburgh, January, 1979.
15. Kung, H.T., *Why Systolic Architectures?,* Computer, 15, 1982, pp. 37-46.

16. O'Leary, D.P., Stewart, G.W., *From Determinancy to Systolic Arrays,*IEEE Transactions on Computers, vol. C-36, no. 11, November, 1987.
17. Privat, G., Renaudin, M., *L'algorithme CORDIC dans les architectures specialisees de traitement numerique du signal,* Traitement du Signal, vol. 5, no. 6, 1988, pp. 421-434.
18. Rajopadhye, S.V., Fujimoto, R.M., *Synthesizing Systolic Arrays from Recurrence Equations,* Parallel Computing, no.14, 1990, pp. 163-189.

$$\begin{tikzcd}
y(t,h_0)=Y(0,0) \arrow[dr] & & & & & \\
& Y(0,1) \arrow[dr] & & & & \\
y(t,h_1)=Y(1,0) \arrow[ur] \arrow[dr] & & Y(0,2) \arrow[dr] & & & \\
& Y(1,1) \arrow[ur] \arrow[dr] & & Y(0,3) \arrow[dr] & & \\
y(t,h_2)=Y(2,0) \arrow[ur] \arrow[dr] & & Y(1,2) \arrow[ur] \arrow[dr] & & Y(0,4) \arrow[dr] & \\
& Y(2,1) \arrow[ur] \arrow[dr] & & Y(1,3) \arrow[ur] \arrow[dr] & & Y(0,5) \\
y(t,h_3)=Y(3,0) \arrow[ur] \arrow[dr] & & Y(2,2) \arrow[ur] \arrow[dr] & & Y(1,4) \arrow[ur] & \\
& Y(3,1) \arrow[ur] \arrow[dr] & & Y(2,3) \arrow[ur] & & \\
y(t,h_4)=Y(4,0) \arrow[ur] \arrow[dr] & & Y(3,2) \arrow[ur] & & & \\
& Y(4,1) \arrow[ur] & & & & \\
y(t,h_5)=Y(5,0) \arrow[ur] & & & & &
\end{tikzcd}$$

Figure 1: Polynomial extrapolation data dependencies.

$$\begin{tikzcd}
Y(0,0)=R(0,0) \arrow[dr] & & & & & \\
R(1,-1)=0 \arrow[r] & R(0,1) \arrow[dr] & & & & \\
Y(1,0)=R(1,0) \arrow[ur] \arrow[dr] \arrow[rr] & & R(0,2) \arrow[dr] & & & \\
R(2,-1)=0 \arrow[r] & R(1,1) \arrow[ur] \arrow[dr] \arrow[rr] & & R(0,3) \arrow[dr] & & \\
Y(2,0)=R(2,0) \arrow[ur] \arrow[dr] \arrow[rr] & & R(1,2) \arrow[ur] \arrow[dr] \arrow[rr] & & R(0,4) \arrow[dr] & \\
R(3,-1)=0 \arrow[r] & R(2,1) \arrow[ur] \arrow[dr] \arrow[rr] & & R(1,3) \arrow[ur] \arrow[dr] \arrow[rr] & & R(0,5) \\
Y(3,0)=R(3,0) \arrow[ur] \arrow[dr] \arrow[rr] & & R(2,2) \arrow[ur] \arrow[dr] \arrow[rr] & & R(1,4) \arrow[ur] & \\
R(4,-1)=0 \arrow[r] & R(3,1) \arrow[ur] \arrow[dr] \arrow[rr] & & R(2,3) \arrow[ur] & & \\
Y(4,0)=R(4,0) \arrow[ur] \arrow[dr] \arrow[rr] & & R(3,2) \arrow[ur] & & & \\
R(5,-1)=0 \arrow[r] & R(4,1) \arrow[ur] & & & & \\
Y(5,0)=R(5,0) \arrow[ur] & & & & &
\end{tikzcd}$$

Figure 2: Rational extrapolation data dependencies.

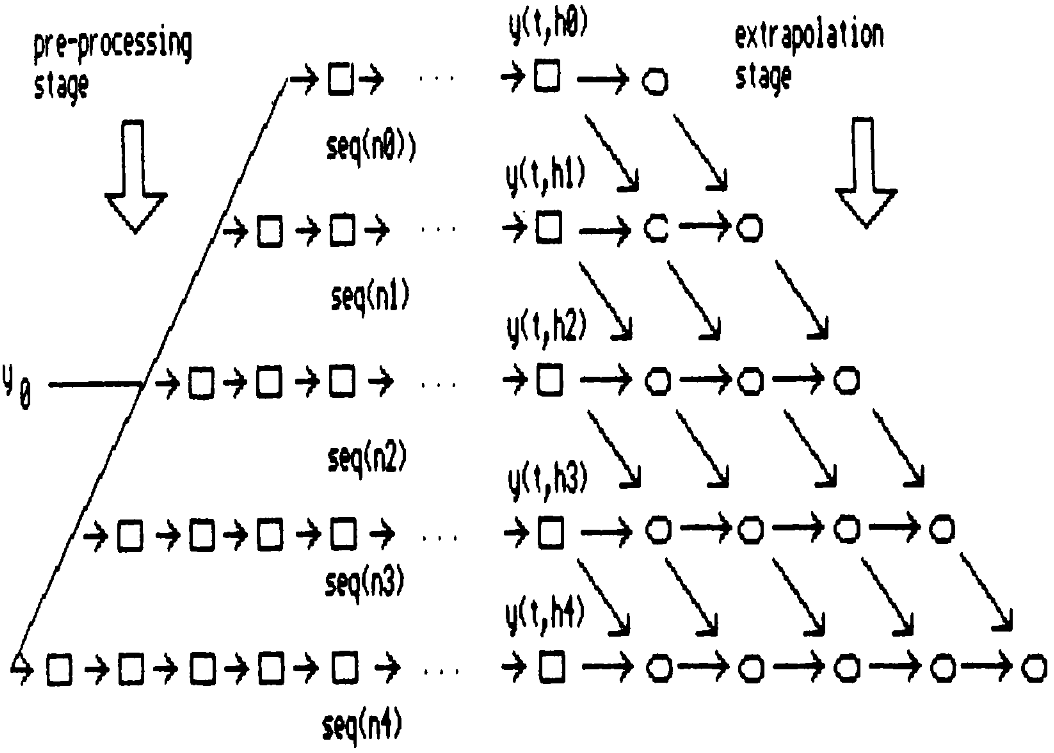


Figure 3: Polynomial extrapolation including the pre-processing stage (K=5).

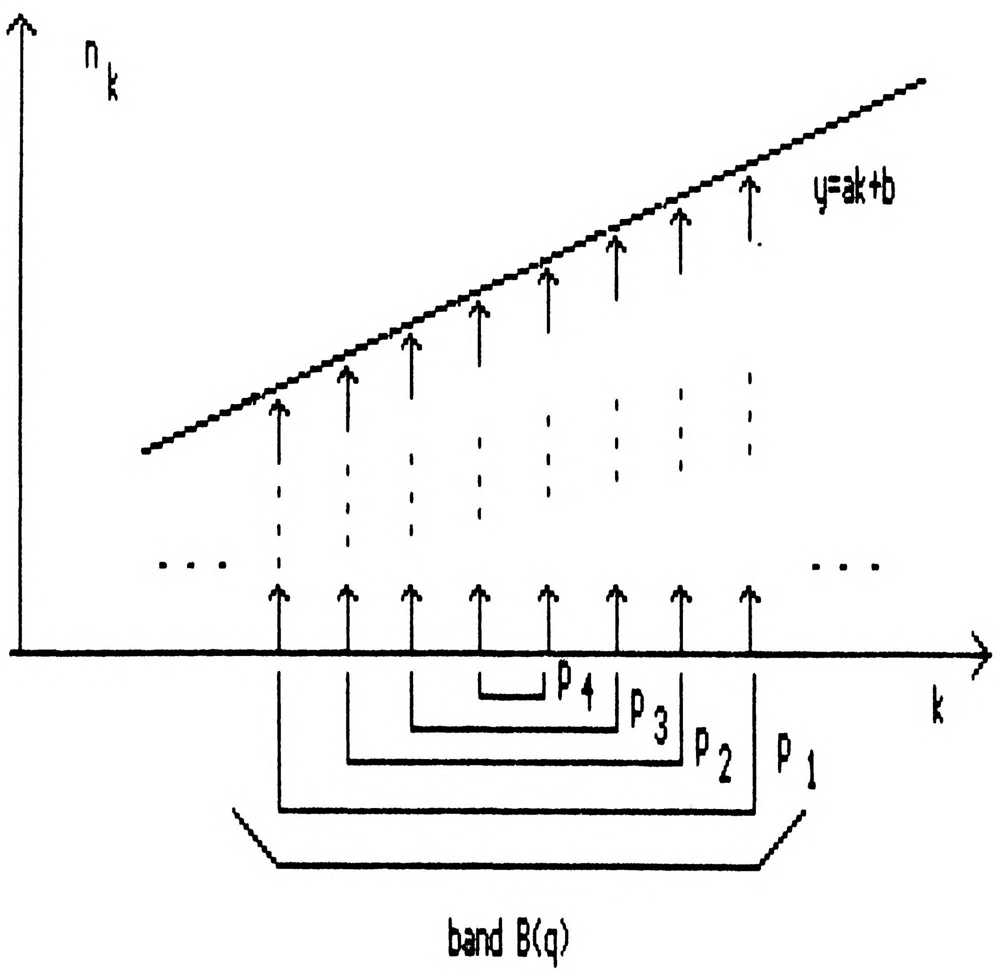


Figure 4: Parallel pre-processing for n_k=ak+b.

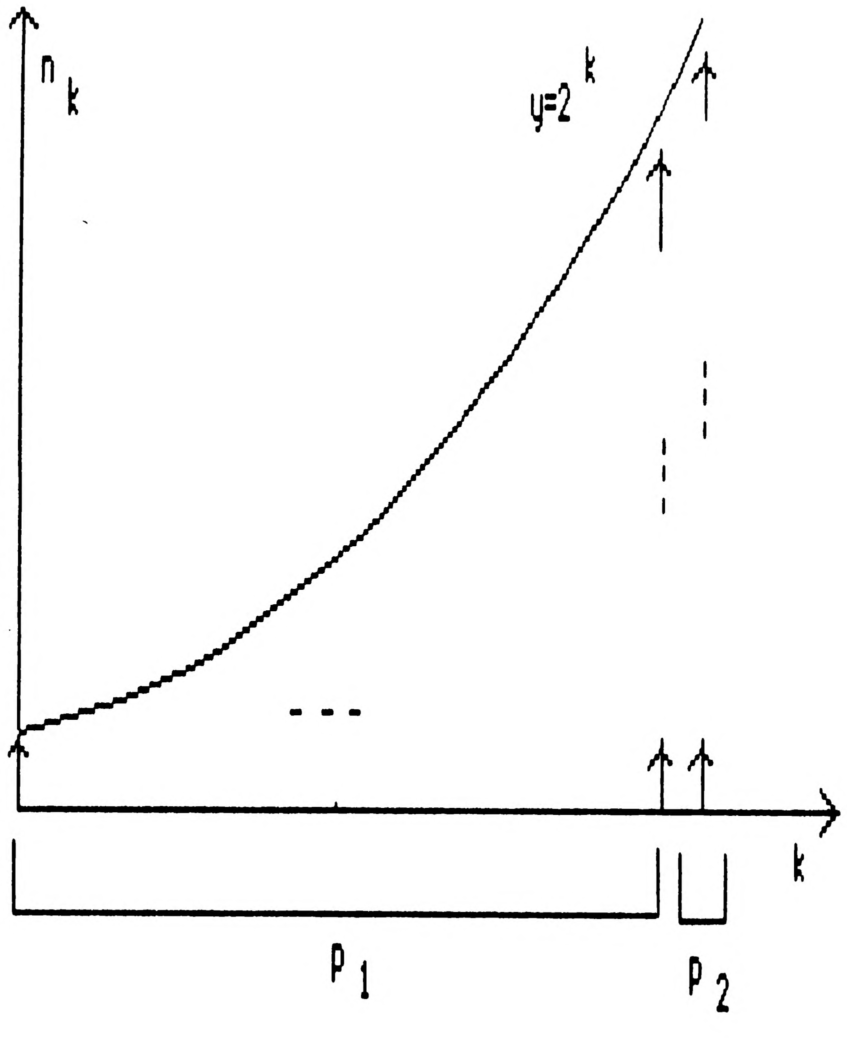


Figure 5: Two processors executing the pre-processing stage for n_k=2**k.

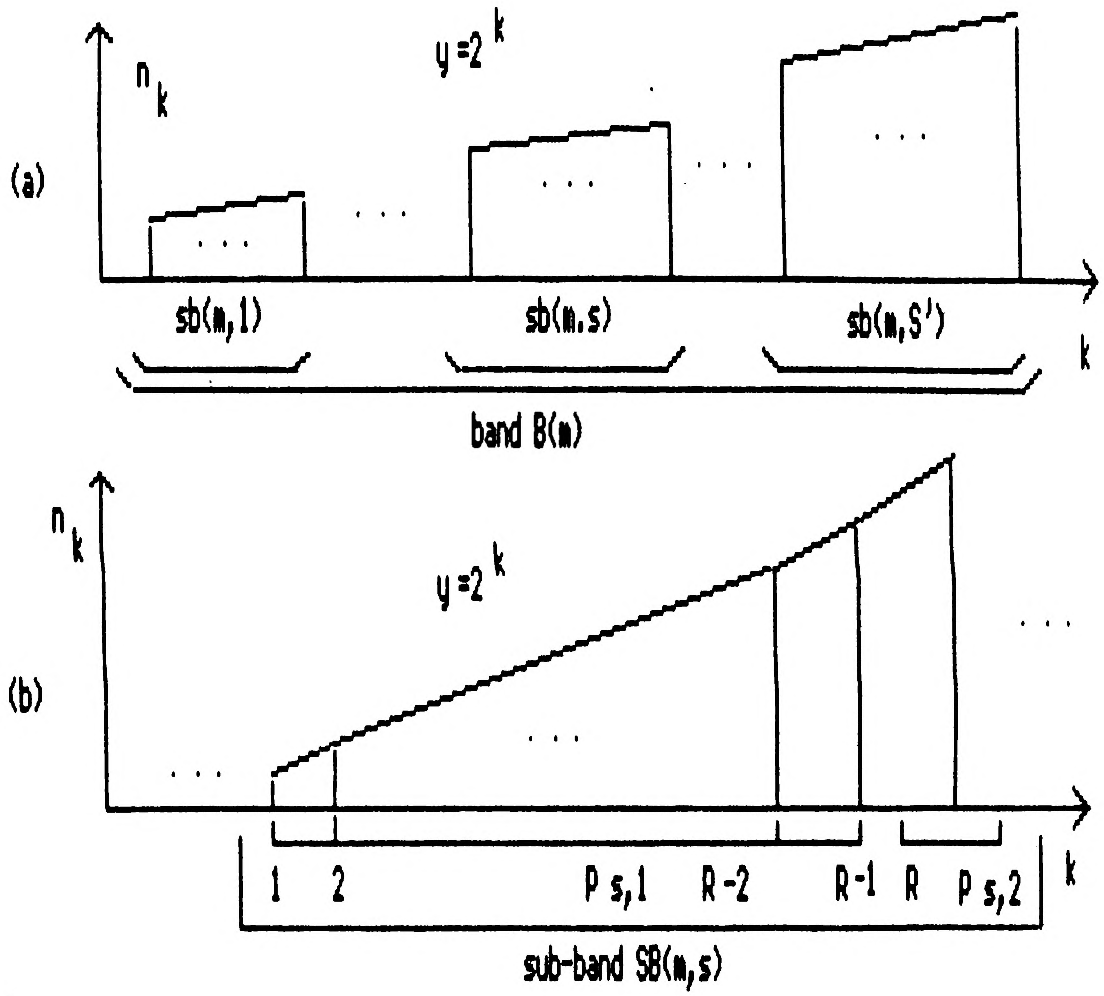


Figure 6: Parallel pre-processing for n_k=2**k:
(a) band structure of the sequence;
(b) the execution of $SB(m,s)$ by the pair $(P(s,1), P(s,2))$.

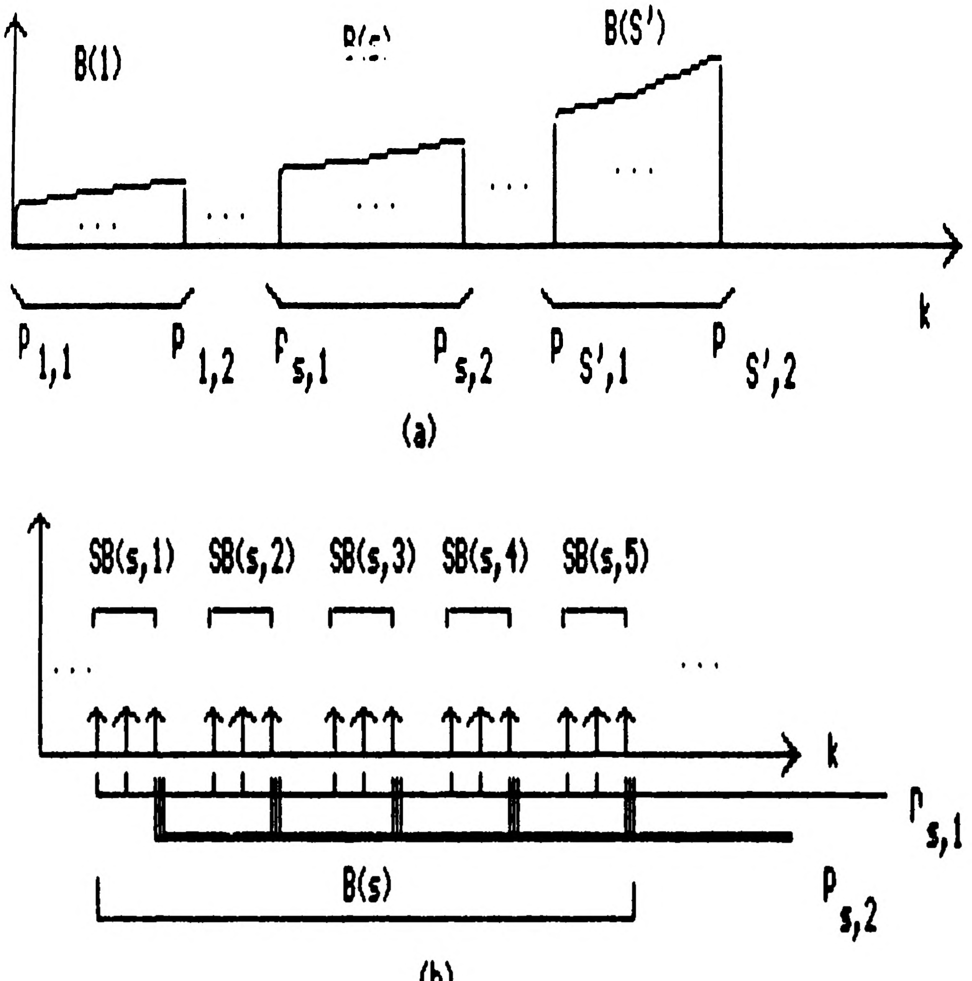


Figure 7: (a) The band structure of the K+1 sequences
(b) The sequences assigned to $P(s,1)$ and $P(s,2)$ (R=5) when $n_{k+1}=n_k+n_{k-1}$.

ON A NONLINEAR-FRACTIONAL OPTIMIZATION PROBLEM IN GRAPHS

Eugenia IACOB[*]

Received: November 1, 1992
AMS Subject Classification: 05C75, 05C38, 90C35

REZUMAT. Asupra unei probleme neliniar-fractionară de optimizare în grafe. Fie $G(N,A)$ un graf finit şi neorientat. Asociem fiecărei muchii a grafului două ponderi pozitive: costul c_{ij} şi capacitatea k_{ij}. Notăm cu Y mulţimea tuturor arborilor de acoperire ai grafului G. Pentru $T \epsilon Y$ definim costul $c(T)$ şi capacitatea $k(T)$ a arborelui T. Scopul acestei lucrări este de a rezolva următoarea problemă de optimizare: Să se determine un arbore $T \epsilon Y$ care minimizează $c(T)/k(T)$ pe mulţimea Y. În articol se prezintă şi un algoritm aproximativ de determinare a soluţiei optimale.

1. General problem. Let $G(N,A)$ be a finite undirected graph, where N is the vertex set and $A \subseteq \{(i,j): i,j \in N\}\setminus\{(i,i): i \in N\}$ is the edge set.

We associate to each edge (i,j) in A two positive integer pounds, that is a cost c_{ij} and a capacity k_{ij} .

Let Y be a given set of subgraphs of $G(N,A)$. For every subgraph T in Y we define:

$$c(T) = \sum_{(i,j)\in T} c_{ij} \quad \text{and} \quad k(T) = \min \{ k_{ij} : (i,j) \in T\},$$

representing, the cost and the capacity of the subgraph T, respectively.

We consider the following general fractional problem:

(P). Find a subgraph T'in Y minimizing the ratio $c(T)/k(T)$ over the set Y, namely:

$\min \{ c(T)/k(T) : T \in Y \}$.

The particular case of problem (P) , when Y is the set of

[*] *University "Babeş-Bolyai" Cluj-Napoca, Department of Computer Science, 3400 Cluj-Napoca, Romania*

paths between two given vertex of the graph G was studied by Martins [3]. In the paper [5], we gave a generalization of Martins algorithm.

The purpose of this paper is to apply this general algorithm for a particular case of the set Y. Namely, we take Y to be the set of all spanning trees of the graph G. In this case, we solve problem (P) by the perturbation of its cost coefficients. We obtain an approximate optimal solution for the initial problem and we give an evaluation of the deviation from the optimal value.

2. Basic properties. First we introduce the "nondomination" relation on the set Y of spanning trees.

DEFINITION 1. *Let $T,T' \in Y$ be two distinguished spanning trees of $G(N,A)$. T dominates T' if and only if $c(T) \le c(T')$ and $k(T) \ge k(T')$ and the strict inequality holds at least once.*

The fact that T dominates T' we denote by $T \; D \; T'$.

Let $Y_D = \{T \in Y : \exists \; T' \in Y$ such that $T' \; D \; T\}$ be the set of dominated spanning trees.

DEFINITION 2. *$Y_N = Y - Y_D$ is the set of nondominated spanning trees.*

From the above definition it results that Y_N can be viewed as a set of optimal solutions for a bicriterion problem associated to (P).

The algorithm that will be presented for solving problem (P), is based on the concept of nondominated spanning tree. This concept is inspired by the procedures of solving the bicriterion problem.

Further on, we present without proof some theorems and propositions which represent the theoretic support for the algorithm. In the general case, when Y is an arbitrary set of subgraphs of G, we allready proved all these results (see, [5]).

Let Y_0 be the set of optimal solutions of problem (P). The following theorem establishes a relationship between Y_0 and Y_N.

THEOREM 1. ([5]) *Any optimal solution for* (P) *is a non-dominated spanning tree, that is* $Y_0 \in Y_N$.

DEFINITION 3. *The subset* Y_N' *of* Y_N *is a selection of* Y_N *if and only if for any* T *in* Y_N *there exists a unique spanning tree* $T' \in Y_N'$ *such that* $c(T)=c(T')$ *and* $k(T)=k(T')$.

PROPOSITION 1. ([5]) *Let* Y_N' *a selection of* Y_N. *If* T' *and* T'' *are two distingueshed spanning trees such that* $T',T'' \in Y_N'$ *then* $c(T')$ *is not equal with* $c(T'')$ *and* $k(T')$ *is not equal with* $k(T'')$.

From Proposition 1, it follows that the number of elements belonging to Y_N' is bounded by the number m' of edges with distinct capacities from $G(N,A)$. Hence, a selection Y_N' can be computed in $O(m'C(m,n))$ time, where m is the number of edges of $G(N,A)$ and $C(m,n)$ is the time needed for determining the minimum cost spanning tree.

It follows that the complete determination of a selection Y' can be done in polynomial time. Therefore, the execution of an exhaustive search for Y_N' (using Theorem 1), in order to compute an optimal solution of (P) is not unrealistic. However, in the algorithm that we present, we need not to find an entire set Y_N'. As a consequence the number of executions of a minimum cost spanning tree algorithm is minimized.

PROPOSITION 2. ([5]) *Let* $Y_N' = \{T_1, T_2, \dots, T_r\}$ $(r < m)$ *be a selection of* Y_N. *Then* Y_N' *can be ordered such that* $c(T_i) < c(T_{i+1})$ *and* $k(T_i) < k(T_{i+1})$ *for* $i = 1, 2, \dots, r-1$.

We consider now the following set:

$Y_N'' = \{T_1, \dots, T_h\} \in Y_N'$ *where* $T_1, \dots, T_h$ are the first h elements of Y_N' and Y_N' is a selection which has the elements ordered in sense of proposition 2.

THEOREM 2. ([5]) *Let* $k' >$ max $\{k(T) : T \in Y\}$. *If the element* $T_j \in Y_N'$ *verifies the following conditions:*

i) $c(T_j)/k(T_j) = \min \{ c(T)/k(T) : T \in Y_N'\}$,

ii) $c(T_h) < [\, c(T_j)/k(T_j) \,]\, k'$,

iii) T_j *is not in* Y_0,

then exists T_s *in* Y_0 *and* $(Y_N' - Y_N'')$ *such that:*

$k(T_s) > [k(T_j)/c(T_j)]\, c(T_h)$.

3. Algorithm for the problem of the spanning tree with minimum cost/capacity ratio. The basic scheme of the algorithm that we propose is the same as the algorithm for the MINSUM-MAXMIN bicriterion problem. Let us assume that $T_s \in Y_N'$ was just determined with such an algorithm. Then, as $k(T_{s+1}) > k(T_s)$, the edges $(i,j) \in A$, for which $k_{ij} < k(T_s)$, can be deleted from $G(N,A)$. In the resulting graph, the subgraph T_{s+1} is determined as the spanning tree with minimum cost and maximal capacity relatively to the set of all spanning trees with minimum cost.

In fact, the difficulty is that we haven't an algorithm to determine the whole set of minimum cost spanning trees. The algorithm of Kruskal find only one of these spanning trees. These difficulty doesn't appear when Y is the set of all paths between two given vertex of $G(N,A)$ or Y is the set of assignements in a bipartite graph (see, e.g. [1], [2], [3]).

We avoid this dificulty by using the following method.

Let $C = \{c_1, c_2, \ldots, c_m\}$ be the sequence of cost values for the edges of $G(N,A)$ in a nondecreasing order, and let $E=\{(i_1,j_1),\ldots,(i_m,j_m)\}$ be the sequence of corresponding edges of $G(N,A)$. Further on we suppose that $c_1 < c_2$. Otherwise we have $c_1 = c_2 = \ldots = c_m$ and (P) degenerates to the usual problem of determining the maximum capacity spanning tree of a graph $G(N,A)$.

The following algorithm generates some perturbation pounds d_{ij} and perturbated pounds $c_{ij}'=c_{ij}+d_{ij}$, ordered in the sequences $D = \{ d_1,\ldots,d_m \}$ and $C' = \{ c_1',\ldots,c_m' \}$, respectively, corresponding to the same edges as the elements of C.

Algorithm 1.

```
Step 1. Take $k = 1$ and $p = 0$.
Step 2. If $k+1 > m$, then $d_m = 0$ and go to Step 11. Otherwise, go to Step 3.
Step 3. If $c_k = c_{k+1}$, go to Step 4. Otherwise, set $d_k = 0$, take $k+1$ instead of $k$ and go to Step 2.
Step 4. Set $d_k = 0$.
Step 5. Take $p+1$ instead of $p$ and go to the next step.
Step 6. If $k+p = m$, then go to Step 10. If $k+p < m$, then go to Step 7.
```

Step 7. If $c_k = c_{k+p}$, then go to Step 5. Otherwise, go to Step 8.

Step 8. For q from 1 to p do $d_{k+q} = q\ 10^{-L}\ (c_{k+p}-c_k)/p$, where L is a natural number choosen such that $10^L > m$.

Step 9. Take $k+p$ instead of k and go to Step 2.

Step 10. For q from 1 to $m-k$ do

$$d_q = q\ 10^{-L}\ (m-k)^{-1} \max \{ c_{h+1}-c_h : h=1,\dots,m-1\}.$$

Step 11. For q from 1 to m do $c_q' = c_q + d_q$.

Step 12. Stop.

This algorithm modifies the costs of those edges from $G(N,A)$ which have the same cost. More exactly, the algorithm adds to the costs of these edges a positive quantity in order to differentiates their costs. In this way, all values of costs associate to edges are different. In this case, the spanning tree of minimum cost is unique.

The following theorem estimates the maximum error, due to this method, in the determination of the spanning tree with minimum cost/capacity ratio.

Let $e = 10^{-L} \max \{ c_{h+1} - c_h : h = 1 ,\dots, m-1 \}$ and

$$g = \min \{ k_{ij} : (i,j) \in A\}.$$

THEOREM 4. *The maximum error caused by Algorithm* 1 *in finding the spanning tree with minimum cost/capacity ratio is* $(n-1)\ e/g$.

Proof. Let suppose that T^* is the spanning tree with minimum cost/capacity ratio. Then, from the definition of e and g, it follows :

$$\frac{c'(T^*)}{k(T^*)} = \frac{\sum\limits_{(i,j)\in T^*} c_{ij}}{\min\{k_{ij} : (i,j)\in T^*\}} = \frac{\sum\limits_{(i,j)\in T^*} (c_{ij} + d_{ij})}{\min\{k_{ij} : (i,j)\in T^*\}} =$$

$$= \frac{\sum\limits_{(i,j)\in T^*} c_{ij}}{\min\{k_{ij} : (i,j)\in T^*\}} + \frac{\sum\limits_{(i,j)\in T^*} d_{ij}}{\min\{k_{ij} : (i,j)\in T^*\}} \le$$

$$\le \frac{c(T^*)}{k(T^*)} + \frac{(n-1)e}{g}$$

Since, for any (i,j) in A, $d_{ij} > 0$, from the above relations, it results:

$$0 \le \frac{c'(T^*)}{k(T^*)} - \frac{c(T^*)}{k(T^*)} \le \frac{(n-1)e}{g}$$

which means that the conclusion of the theorem is true.

Further we will present an adjustement of Martins's algorithm. Theorems 2 and 3 are used as an attempt to decrease the number of spanning trees that have to be determined.

Three working variables are used in the algorithm: T, c^* and z. T' keeps the best spanning tree that was determined until the curent iteration, c^* keeps the value of $(c(T')/k(T'))\ k'$ and z keeps the value of $c(T')/k(T')$. Foregoing k' is a parameter of the algorithm which is fixed such that $k' > \max\{k(T) : T \in Y\}$.

Algorithm 2.

Step 1. Apply the algorithm 1 for $G(N,A)$, and let denote by c_{ij}' the perturbed costs.

Step 2. Set c^*=INF, z = INF and the graph $H(N,B) = G(N,A)$, where INF is integer positive number such that:

INF > 2 max $\{c(T)/k(T) : T \in Y\}$.

Step 3. Find T'' the spanning tree with minimum cost from $Y(H) = \{ T \subset Y : T$ spanning tree for $H(N,B)\}$, that is :

$c'(T'') = \min \{ c'(T) : T \in Y(H) \}$.

Step 4. If $Y(H)$ is an empty set or $c'(T'') > c^*$, then finish the algorithm. In the opposite case go to Step 5.

Step 5. If $c'(T'')/k(T'') < z$, then perform the following operations:

i) Take $T' = T''$.

ii) Take $c^* = [c'(T'')/k(T'')]\ k'$ and $z = c'(T'')/k(T'')$.

iii) Take $x = k(T'')$ and go to Step 7.

If $c'(T'')/k(T'') > z$, then go to Step 6.

Step 6. Set $x = c'(T'')/z$ and go to Step 7.

Step 7. Eliminate from $H(N,B)$ all the edges (i,j) which has $k_{ij} < x$. After this the new set of edges B is :

$B := B - \{(i,j) \in A : k_{ij} < x \}$.

Go to Step 2.

In order to clarify the implications of theorems 2 and 3 we explain the algorithm step by step.

The first step modifies, if it is necessary, the costs associated to edges of $G(N,A)$ such that these became distinct.

In the second step the working variables c^* and z are initialized .

In the third step we determined a nondominated spanning tree T''. This step is the most complex step of the algorithm because each iteration requires the solving of an optimization problem on the spanning trees set $Y(H)$. We can use in this step Kruskal's algorithm (see, e.g., [4], [2]).

In the fourth step, we check a stop condition of the algorithm. Thus the algorithm stops with an optimal solution T' when either in $Y(H)$ there is no elements or it is verified the condition of theorem 2.

Step 5 is performed when the spanning tree T'' determined in Step 3 is "beter" then the best spanning tree determined until that moment. Also in this step we actualize T', c^* and z. If T'' is worse than T', Step 6 is performed. In Step 5 is used Theorem 3, to justify the elimination (in Step 6) of some edges from the auxiliary graph $H(N,B)$. In this way, it can be possible to avoid some nondominated spanning trees which not belong to Y_0 .

In Step 7 the edges having the capacities smaller than x are eliminated from $H(N,B)$. We must note that $x=k(T')$ when the Step 4 is performed. But when Step 5 is executed the value of x is determined by the Theorem 3.

R E F E R E N C E S

1. Burkard, R.E., Hahn, W., Zimmerman, U.,: *An algebraic approach to assignement problem,* Math. Programming, 12 (1977), pp.219-232.
2. Gondran, M., Minoux, M., *Graphes et algorithmes,* Editions Eyrolles, Paris, 1979.
3. Kruskal, H.B., *On the shortest spanning subtree of a graph and the traveling salesman problem,* Proc. Am. Math. Soc., 71 (1956), pp.48-50.
4. Martins, E.Q.V., *An algorithm to determine a path with minimal cost/capacity ratio,* Discrete Applied Math., 8 (1984), pp. 189-194.
5. Tigan E., *Algorithms to minimize the cost/capacity ratio,* Econ. Comp. and Econ. Cyb. Studies and Res., 4 (1989), pp.53-59.

STUDIA UNIV. BABEŞ-BOLYAI, MATHEMATICA, XXXVII, 3, 1992

FILES D'ATTENTE DANS DES SYSTÈMES DISTRIBUÉES DES SERVICES

Gr. MOLDOVAN[*] and I. RÂP[*]

Received: February 21, 1993
AMS Subject Classification: 68M20, 60K25

REZUMAT. - Cozi de aşteptare în sisteme distribuite de servicii. Se consideră un sistem de aşteptare distribuit ce are o topologie particulară determinată de matricea de rutaj dată. În anumite ipoteze ale sistemului de aşteptare au loc teoremele 1 şi 2 ce dau evaluări pentru unele caracteristici numerice mai importante.

L'étude des systèmes distribueés des services exécutées à l'aide des ordinateurs a actuellement une importance de plus en plus grande. La réalisation des certains services distribuées à l'aide des ordinateurs est possible grace au développement de la technologie de construction des ordinateurs basée sur l'utilisation de microprocesseurs de plus en plus performants. Plusieurs systèmes distribuées des services correspondent aux réseaux des ordinateurs ayant des topologies connues.

Un système distribué des services exécutées à l'aide des ordinateurs suppose l'existence des certaines composants physiques (réseaux des ordinateurs) et des programmes assurant le traitement le transfert, le contrôle et la protection des données. Dans le transfert des donnés et des informations, qui est fait par certaines procédées de comutation, l'exécution des services suppose la consideration du caractère aléatoire de la réalisation des certains événements.

Dans ce qui suit on suppose que l'unité qui doit être traîté dans un noeud du système distribué est le job.

[*] *University of Cluj-Napoca, Department of Computer Science, 3400 Cluj-Napoca, Romania*

Nous faisons les hypothèses suivantes:

I1. Les jobs arrivent aléatoirement à aux ordinateurs du réseau en formant un flux poissonnéen.

I2. Le temps d'éxecution de n'importe quel job est aléatoire, ayant une répartition exponentielle.

I3. Pour tous les ordinateurs du réseau les temps des services et les inter-arrival temps sont des variables aléatoires indépendantes.

Dans les hypothèses faites les ordinateurs qui éxistent dans les noeuds du réseau des ordinateurs sont les stations de service dans des files d'attente $M/M/1$.

Considérons un système distribué formé par les noeuds C_0, $C_1,\ldots,C_n$ et supposons conues les probabilités p_{ij} q'un job qui a été éxécuté dans le noeud C_i est dirijé vers le noeud C_j, pour $i,j = \overline{0,n}$.

La matrice $P = (p_{ij})$, $i,j = \overline{0,n}$ s'appele la matrice de routage du réseau.

Dans le processus de passage des jobs d'un noeud a l'autre des agglomerations seront crées dans certains noeuds. Pour étudier cettes agglomerations nous considérons le réseau des ordinateurs comme un réseau d'attente exponentiel ayant les stations dans les noeuda et ayant la matrice de routage P. Ce réseau d'attente sera considéré en régime stationaire.

Il est utile de trouver le nombre des jobs qui attends à être éxécutés dans les noeuds du réseau des ordinateurs.

Un état du réseau d'attente sera le vecteur $k = (k_0, k_1,\ldots,k_n)$ où k_i est le nombre des jobs en cours d'éxécution on qui attendent à être éxécutés par l'ordinateur C_i , $i = \overline{0,n}$.

Pour les réseaux exponentielles d'attente a lieu la théorème de forme produit du Jackson ([1], [2]). Donc, la probabilité que le réseau d'attente se trouve dans l'état k est

$$P(k) = \prod_{i=0}^{n} P_i(k_i)$$

où $P_i(k_i)$ est la probabilité que dans le système d'attente de l'ordinateur C_i se trouvent k_i jobs, $i = \overline{0,n}$. Dans le régime stationaire ces probabilités ne dependent pas de temps.

Soit a le paramètre du flux poissonnéen des arrivées des jobs à l'ordinateur C_0, soit b_i le paramètre du temps de service exponentiel de l'ordinateur C_i, $i = \overline{1,n}$ et soit p_i la probabilité que le job éxécuté à l'ordinateur C_i quitte le réseau, $i = \overline{0,n}$.

Içi nous nous limitons au cas $p_{ij} = 0$ pour $|i-j| > 1$ et $p_{i0} = 0$ pour $i = \overline{1,n}$.

Les probabilités ci-dessous satisfont les relations

$$\begin{cases} p_0 = 1 - \sum_{i=1}^{n} p_{0i} \\ p_1 = 1 - p_{12} \\ p_j = 1 - (p_{j,j-1} + p_{j,j+1}) \ , j=\overline{2,n-1} \\ p_n = 1 - p_{n,n-1} \end{cases}$$

Si a_i notte le paramètre du flux d'entrée dans le file d'attente de l'ordinateur C_i, $i = \overline{0,n}$, alors [3]

$$\begin{cases} a_0 = a \\ a_1 = p_{01}a_0 + p_{21}a_2 \\ a_j = p_{0j}a_0 + p_{j-1,j}\, a_{j-1} + p_{j+1,j}\, a_{j+1},\ j=\overline{2,n-1} \\ a_n = p_{0n}\, a_0 + p_{n-1,n}\, a_{n-1} \end{cases}$$

Le facteur de service de l'ordinateur C_i sera

$r_i = \dfrac{a_i}{b_i}$, $i = \overline{0,n}$. L'existence du régime stationaire exige que

$r_i < 1, \quad i = \overline{0,n}$.

Pour ce type de système distribué ont lieu les théorèmes suivants.

THÉORÈME 1. *Dans les hypothèses I_1 - I_3 et en régime stationaire on a*

i) $$P(k_0,k_1,k_2) = \frac{a^{k_0+k_1+k_2}(b_0 - a)}{b_0^{k_0+1}\, b_1^{k_1+1}\, b_2^{k_2+1}\, (1-p_{12}\, p_{21})^2} .$$

$$.\,[b_1\,(1-p_{12}\,p_{21}) - a(p_{01} + p_{02}\,p_{21}\,)]\,.$$

$$.\,[b_2(1 - p_{21}\,p_{12}\,) - a(p_{02} + p_{01}\,p_{12})]$$

ii) *L'espérance mathématique du nombre des jobs à l'ordinateur C_i est*

$$N_i = \frac{r_i}{1 - r_i} \quad , \quad i = 0,1,2$$

iii) *Le facteur de nonutilisation de l'ordinateur C_i est*

$$\theta_i = 1 - r_i \quad , \quad i = 0,1,2$$

iv) *L'espérance mathématique de nombre des jobs dans le réseau est*

$$N = \frac{r_0}{1-r_0} + \frac{r_1}{1-r_1} + \frac{r_2}{1-r_2}$$

v) *Si la discipline d'attente est FIFO, l'espérance mathématique du temps d'attente d'un job à l'ordinateur C_i est*

$$T_i = \frac{1}{b_i(1-r_i)} \quad , \quad i = 0,1,2.$$

THÉORÈME 2. *Dans les hypothèses I_1 - I_3 et en régime stationaire ou*

a

i) $P(k_0, k_1, \ldots, k_n) = \prod_{i=0}^{n} (1-r_i) r_i^{k_i}$

ii) *L'espérance mathématique du nombre des jobs à l'ordinateur C_i est*

$$N_i = \frac{r_i}{1-r_i} \quad , \quad i = \overline{0,n}$$

iii) *Le facteur de nonutilisation de l'ordinateur C_i est*

$$\theta_i = 1-r_i \quad , \quad i = \overline{0,n}$$

iv) *L'espérance mathématique du nombre des jobs dans le réseau est*

$$N = \sum_{i=1}^{n} \frac{r_i}{1-r_i}$$

Si la discipline d'attente est FIFO, l'espérance mathématique du temps d'attente d'un job à l'ordinateur C_i est

$$T_i = \frac{1}{b_i(1-r_i)} \quad i = \overline{0,n}$$

Les approximations données correspondent aux approximations

$$a_i = \frac{A_i}{B_i} \qquad i = \overline{1,n}$$

où

$$A_i = \sum_{j=1}^{n} \ell_{ij} \quad , \quad i = \overline{1,n}$$

avec

$$\ell_{ij} = \begin{cases} p_{0i} \prod_{s=i}^{j+1} p_{s,s-1} & ,pour \quad j<i \\ p_{0i} & ,pour \quad j=i \\ p_{0i} \prod_{s=i}^{j-1} p_{s,s+1} & ,pour \quad j>i \end{cases}$$

et

$$B_i = \begin{cases} 1 - p_{12}\, p_{21} \quad , \quad pour \ i = 1 \\ 1 - p_{i,i-1}\, p_{i-1,i} - p_{i,i+1}\, p_{i+1,i} \quad , \quad pour \ i = \overline{2,n-1} \\ 1 - p_{n,n-i}\, p_{n-1,n} \quad , \quad pour \ i=n \end{cases}$$

Les démonstrations des deux théorèmes consistent dans la résolution des équations de Chapmann-Kolmogoroff d'un file d'attente $M/M/1$ en régime stationaire et la résolution du système des équations satisfaites par a_i , $i = \overline{0,n}$.

Le cas particulier du théorème 1 est presenté en detail dans [4].

B I B L I O G R A P H I E

1. Jackson, R.R.P., *Networks of waiting lines*, JORSA, vol 5, nr. 4, pp. 518 (1957).
2. Jackson, R.R.P., *Job-shop like quening systems*, Management Science, vol. 10, pp. 131-142 (1963).
3. Vijbrands, R. J., *Quening Network Models and Performance Analysis of Computer Systems*, Technische Universiteit Eindhoven, Nederland, 1988.
4. Moldovan, Gr., Râp, I., *Asupra unei probleme de aşteptare în reţele de calculatoare*, Lucrările Conferinţei de Matematică Aplicată şi Mecanică, Cluj-Napoca, 21-23, octombrie 1988, Cluj-Napoca 1989, pp. 493-501.

VARIOUS KINDS OF INHERITANCE

S. MOTOGNA* and V. PREJMEREAN*

Received: November 18, 1992
AMS Classification:68Q65, 68Q99

REZUMAT. Diverse tipuri de moştenire. Lucrarea de faţă se vrea un mic studiu comparativ, în funcţie de avantajele şi dezavantajele lor, a diverselor tipuri de moştenire implementate în limbajele orientate obiect. Studiul s-a făcut în principal pe limbajele Smalltalk, Beta şi CLOS deoarece exemplifică cel mai bine moştenirea simplă, multiplă şi în CLOS introducerea claselor abstracte numite mixin.

1. Introduction : A variety of inheritance mechanisms have been developed for object-oriented programming languages. These mechanisms range from Smalltalk single inheritance to the complex and powerful multiple inheritance combination of CLOS.

These languages have similar object models and also share the view that inheritance is an incremental modification mechanism, but they differ widely in the kind of incremental changes supported.

Inheritance is a hierarchical incremental modification mechanism that transform a parent P with a modifier M into a result R : $R = P + M$ as in figure 1.

$$\left.\begin{array}{r}\text{Parent } P\\ \text{Modifier } M\end{array}\right\}\quad R = P + M$$

fig. 1

The parent P, modifier M and result R have a set with finite number of attributes :

* "Babeş-Bolyai" University, Department of Computer Science, 3400 Cluj-Napoca, Romania

$P = (P_1, P_2, \ldots, P_p)$

$M = (M_1, M_2, \ldots, M_m)$

$R = (R_1, R_2, \ldots, R_r)$

When the attributes of the modifier M differ from those of the parent P then the result R has $p+m$ attributes, contains the union of the P and M attributes. For the overlapping attributes, the modifier attributes redefine the parent attributes, in the same way as the identifiers declared in an inner encapsulated module redefine those declared outside the module.

The inheritance in Smalltalk, Beta, CLOS are representative of three kinds of inheritance. The inheritance mechanisms seem to be very different but they have a common structure. This mechanism combines two sets of attributes P and M such that duplicate attribute definitions are given a value from one set.

2. Single Inheritance : If we consider single inheritance, each class has at most one superclass, the determination of the class procedure list is trivial : we only need to traverse a linear path to the most general superclass of its inheritance hierarchy.

Inheritance in Smalltalk is a mechanism for incremental derivation of classes, it is a single inheritance and was adopted from Simula.

In Beta the inheritance is single, too and is designed to provide security from substituting of a method by a completely different method. Inheritance is supported by prefixing of definitions.

These two mechanisms are the same, only the direction of modification is different. In Smalltalk the new attributes are favored and may replace the inherited ones, in Beta the original attributes are favored.

3. **Multiple Inheritance:** The real world has in many situations to deal with multiple inheritance. The natural inheritance comes from two parents more than from one.

Let's consider a class T which inherits from superclasses $T_1, T_2, \ldots, T_n$:

class T inherits ($T_1, T_2, \ldots, T_n$) in T - body.

Some multiple inheritance systems, as CLOS extend the inheritance hierarchies, i.e. by ordering $T_1, \ldots, T_n$ in a linear order from left to right.

The problem of the multiple inheritance is at the moment of invoking a method. If a method is defined in more than one superclass which of them should be invoked? There must be no conflict between characteristics inherited from independent classes (even if these characteristics have the same name).

There are some algorithms for linearization the hierarchy, for reaching a method but each of them has some disadvantages.

There are two kinds of strategies to solve the conflict problem in multiple inheritance : linear strategy and graph-oriented strategy.

The principle of linear strategy is that the inheritance graph should be transformed to a linear structure, without duplicates and treat the resulting graph as a single inheritance one. This goal is realised ordering the superclasses list of a class using depth-first search or breadth-first search.

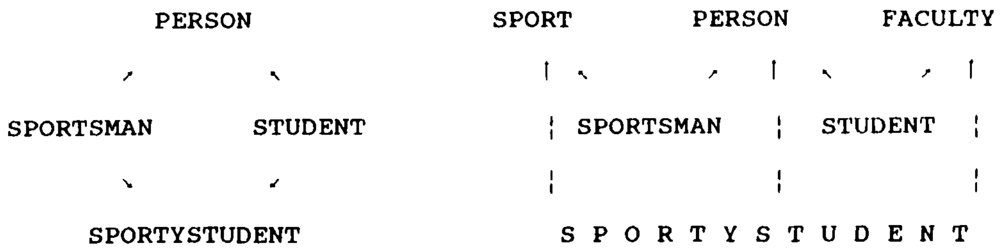
The graph-oriented strategy works directly on the inheritance graph without modifying it, allowing to access each inherited characteristic. When a conflict occurs the superclass from which we want to inherit must be specified . We mention that this class is not necessary the one which defines the characteristic. A technic which is used in extended Smalltalk is the selector composition. A composed selector is a selector preceded by the class name.

Using linearization, a CLOS multiple inheritance hierarchy could be reduced to a collection of inheritance chains, each of which can be interpreted using single inheritance.

4. **Mixins** : A mixin is an abstract subclass that may be used to specialize the behavior of a variety of parent classes. In contrast to classes, mixins are no objects which can create own instances. In mixins we may define new methods that perform some actions and then call the coresponding parent methods, but these methods are only textually. When a class is defined, the methods of its mixins will be defined for that class.

Stroustrup[6] had the following argument for using multiple inheritance : "it might be useful to have class *B* inheriting from two classes A_1 and A_2 ...". This is not possible using mixins but if we use factorization we obtain a solution for this.

VARIOUS KINDS OF INHERITANCE



using multiple inheritance using mixins and factorization

fig. 2

CLOS supports mixins and it seems to be not so difficult to extend Beta and Smalltalk to support mixins and generalized inheritance.

R E F E R E N C E S

1. Wegner, P. and Zdonik, S. B., *Inheritance as an Incremental Modification Mechanism or What Like Is and Isn't Like,* in ECOOP'88 Proceedings,pp 55-77.
2. Ducournau, R. and Habib, M., *On Some Algorithms for Multiple Inheritance in Object Oriented Programming,* in ECOOP'87, pp 243-252
3. Bracha, G. and Cook, W., *Mixin-based Programming,* in ECOOP/OOPSLA'90 Proceedings, pp 303-311.
4. Bretthauer, H., Christaller, T. and Kopp, J., *Multiple vs. Single Inheritance in Object-oriented Programming Languages*. What Do We really Want, 1989.
5. Wegner, P., *Concepts and Paradigms of Object Oriented Programming* - Expansion of Oct. 4 OOPSLA'89 Keynote Talk, OOPS Messenger, vol 1, no 1, pp 7-87, Aug 1990.
6. Stroustrup,B., *The C++ Programming Language*. Addison Welsey, 1986.

STUDIA UNIV. BABEŞ-BOLYAI, MATHEMATICA, XXXVII, 3, 1992

ALGEBRAIC SPECIFICATION OF PROGRAMMING LANGUAGE SUBSETS

Ilie PARPUCEA[*]

Received: March 1, 1993
AMS Subject Classification: 68Q05

REZUMAT. Specificarea algebrică a subseturilor unui limbaj de programare. În această lucrare autorul încearcă aplicarea unui model algebric de specificare, în definirea subseturilor unui limbaj de programare. Modelul se bazează pe ierarhia algebrelor eterogene. Cîteva rezultate teoretice cît şi un exemplu concret de specificare sînt redate amănunţit pe parcursul întregii lucrări.

1. Introduction. The algebraic modelling of programming language specification is subjected to an intense research, with significant results. Partial or "relatively total" solutions were found, having the category theory [ADJ73, ADJ77], partial or total heterogeneous universal algebras [BRO82, BRO87], algebras with operator schemes, or even context-free algebras [RUS72] as algebraic departure point. These solutions concerned many times only certain aspects of the programming language specification.

The ADJ group of authors attempts to develop a concept of programming language within the framework of the category theory. It is interesting as to the mathematical object in its own self, constituting a strong source of inspiration for subsequent results. In [WAG89] the author presents a model of algebraic specification of a language for abstract data type specification. This model of language belongs to the object-oriented language set.

The total or partial heterogeneous algebras seem to be ever

[*] *Universitatea "Babeş-Bolyai" Cluj-Napoca*
Facultatea de Ştiinţe Economice
P-ţa Ştefan cel Mare 1, 3400 Cluj-Napoca, România

more used to the complex approach of programming language specification. Lots of papers (e.g. [BRO82, BRO87]) constitute interesting approaches, from both theoretical and practical viewpoints. They present algebraic models for defining abstract types of partial data and hierarchical data types. Very interesting is the pure algebraic model concerning the specification of programming language semantics, using particular models of abstract data types [BRO87].

Both the heterogeneous algebras [BIR70] and those with scheme of operators [HIG63] constituted the main theoretical source for the elaboration of HAS hierarchy [RUS80] and introduction of context-free algebras [RUS72]. It is to be noticed that the HAS hierarchy concept allows a unitary approach of the programming language specification, from both semantic and syntactical viewpoint.

The specification of a programming language implies the existence of a set of data (which constitutes the data base of the language), and a set of operation defined on this data base (which constitutes the instruction set). The data base and the instruction set associated to a language constitute the so-called "computing reality". The concept of program will allow the identification of that subset from the "computing reality" which needs transformations. Within the framework of a specified programming language, a program will constitute a well defined entity from two points of view: semantic and syntactical.

We shall consider a language which gathers together all "computing realities" corresponding to a given set of languages.

This language will be called universal language (UL). Its restriction to a certain "computin reality" will constitute a subset of the universal language. Such an approach allow to consider the programming languages hierarchically as to their specification and implementation.

2. Example of simple language. In order to exemplify our model, we shall consider the following simple language. This language includes three data types: numerical, array and record. On the numerical type there are defined the arithmetic operations (+,-,*,/,**) and the relational operations (<,<=,>,>=,==,/=). One defines the instruction IF and the assignment instruction. The syntax of data types and arithmetic expressions is given in BNF by:

```
<TYPE_DECLARATION>::=type <TYPE_NUME> is <TYPE_DEFINITION>
<TYPE_DEFINITION>::=<NUMERICAL_TYPE>/<ARRAY_TYPE>/<RECORD_TYPE>
<CONSTRAINT>::= range <RANGE>
<RANGE>::=<SIMPLE_EXPRESSION>..<SIMPLE_EXPRESSION>
<SIMPLE_EXPRESSION>::=<FACTOR>{<ARITHMETICAL_OPERATOR><FACTOR>}
<FACTOR>::=<VARIABLE>/<CONSTANT>
<VARIABLE>::=<IDENTIFIER>/<INDEXED_COMPONENT>/
             <SELECTED_COMPONENT>
<RELATION>::=<SIMPLE_EXPRESSION><RELATIONAL_OPERATOR>
             <SIMPLE_EXPRESSION>
<ARITHMETICAL_OPERATOR>::=+/-/*///**
<RELATIONAL_OPERATOR>::=</<=/>/>=/==//=
<SUBTYPE_DEFINITION>::=subtype<SUBTYPE_NAME>
```

```
                         is<TYPE_DESIGNER>[<CONSTRAINT>]
<TYPE_DESIGNER>::=<TYPE_NAME>/<SUBTYPE_NAME>/
                    <PREDEFINITION_TYPE>
<NUMERICAL_TYPE>::=<CONSTRAINT>/new<PREDEFINITION_TYPE>
                   <CONSTRAINT>
<PREDEFINITION_TYPE>::=<INTEGER>/<FLOAT>
<OBJECT_DECLARATION>::=<IDENTIFIER_LIST>:<TYPE_DESIGNER>
                    [:=<SIMPLE_EXPRESSION>];
<IDENTIFIER_LIST::=<IDENTIFIER>{,<IDENTIFIER>}
<RECORD_TYPE>::=record<COMPONENT_LIST> end record.
<COMPONENT_LIST>::=<OBJECT_DECLARATION>{,<OBJECT_DECLARATION>}
<ARRAY_TYPE>::=array(<INDEX>{,<INDEX>}) of <TYPE_DESIGNER>
<INDEX>::=<RANGE>/<INTEGER>
```

The syntax of assignment and IF instructions is given also in BNF by:

```
<ASSIGNMENT_STATEMENT>::=<VARIABLE>:=<SIMPLE_EXPRESSION>
<EL_IF>::=<RELATION> then <SEQUENCE>
<IF_LIST>::=<EL_IF>/<IF_LIST>elseif<EL_IF>
<IF>::=if<IF_LIST>endif/if<IF_LIST>else<SEQUENCE>endif
<SEQUENCE>::=<ASSIGNMENT_STATEMENT>{;<ASSIGNMENT_STATEMENT>}
```

Remark. The symbols +, -, *, /, ** correspond to the arithmetic operations of addition, substraction, multiplication, division, powering, while the symbols <, <=, >, >=, ==, /= correspond to the relational operations of smaller than, smaller than or equal to, greater than, greater than or equal to, equal to, different from, respectively.

In the above defined language the numerical types <INTEGER>

and <FLOAT> are considered to be predefined. We renounced the definition of the following syntactical categories: <INDEXED-COMPONENT>, <SELECTED-COMPONENT>, <IDENTIFIER>, <CONSTANT>, <TYPE-NAME> and <SUBTYPE-NAME>. Neither the theoretical aspect, nor the examples we shall present will be altered by these restrictions.

A context-free grammar is considered to be a quadruple $\underline{G} = < V_n \cup V_t \ , \ V_t \ , \ P \ , \ V_n >$, *where*

$$P = \{ \ A \rightarrow s_0A_1s_1A_2s_2\ldots s_{n-1}A_ns_n \mid s_0, s_1, \ldots, s_n \in V_t^*, \\ A, \ A_1, \ A_2, \ldots, A_n \in V_n \}$$

V_n is the nonterminal symbol set, while V_t is the terminal symbol set. A specification base $\underline{B} = < V_n \cup V_t \ , \Sigma \ S>$ is made to correspond to this context-free grammar, where

$$\Sigma \ S = \{ \ \delta = <n, s_0s_1\ldots s_n, A_1A_2\ldots A_nA> \mid A \rightarrow s_0A_1s_1\ldots s_{n-1}A_ns_n \in P \}.$$

The context-free algebra [RUS83] specified by $\underline{B}$ can be written in the form of the triple

$$\underline{A} = < \ (W)_{A\in V_n} = W_A(V_t, \Sigma S)_{A\in V_n}, \Sigma S_{\underline{B}}, \ F >,$$

where $W_A(V_t,\Sigma S) = \{x\in V_t^* \mid A \overset{*}{\rightarrow} x\}$, and for every $w_k \in W_{A_k}(V_t,\Sigma S)$,

$k=1,2,\ldots,n$, and $\delta = <n, s_0s_1\ldots s_n, A_1A_2\ldots A_nA>$ we have $F_\delta(w_1,w_2,\ldots,w_n) = s_0w_1s_1\ldots s_{n-1}w_ns_n \in W_A(V_t,\Sigma S)$.

F is the symbol of a function which associates to every operation scheme $\delta \in \Sigma S$ a heterogeneous operation specific to the context-free algebra $\underline{A}$. If $\delta=<n,s_0s_1\ldots s_n, A_1A_2\ldots A_nA>$, then F_δ is the function

$$F_\delta : W_{A_1} \times W_{A_2} \times \ldots \times W_{A_n} \to W_A.$$

The action law of the function is the above mentioned one. We present further down the operation schemes $\delta \in \Sigma S$:

σ_1 = <2, type is,TYPE_NAME TYPE_DEFINITION TYPE_DECLARATION>

σ_2 = <1, ,NUMERICAL_TYPE TYPE_DEFINITION>

σ_3 = <1, ,ARRAY_TYPE TYPE_DEFINITION>

σ_4 = <1, ,RECORD_TYPE TYPE_DEFINITION>

σ_5 = <1, range ,RANGE CONSTANT>

σ_6 = <2, .. ,SIMPLE_EXPRESSION SIMPLE_EXPRESSION RANGE>

σ_7 = <1, ,FACTOR SIMPLE_EXPRESSION>

σ_8 = <3, ,FACTOR ARITHMETICAL_OPERATOR SIMPLE_EXPRESSION SIMPLE_EXPRESSION>

σ_9 = <1, ,VARIABLE FACTOR>

σ_{10}= <1, ,CONSTANT FACTOR>

σ_{11}= <1, ,IDENTIFIER VARIABLE>

σ_{12}= <1, ,INDEXED_COMPONENT VARIABLE>

σ_{13}= <1, ,SELECTED_COMPONENT VARIABLE>

σ_{14}= <3, ,SIMPLE_EXPRESSION RELATIONAL_OPERATOR SIMPLE_EXPRESSION RELATION>

σ_{15}= <0, + ,ARITHMETICAL_OPERATOR>

σ_{16}= <0, - ,ARITHMETICAL_OPERATOR>

σ_{17}= <0, * ,ARITHMETICAL_OPERATOR>

σ_{18}= <0, / ,ARITHMETICAL_OPERATOR>

σ_{19}= <0, ** ,ARITHMETICAL_OPERATOR>

σ_{20}= <0, < ,RELATIONAL_OPERATOR>

σ_{21}= <0, <= ,RELATIONAL_OPERATOR>

σ_{22}= <0, > ,RELATIONAL_OPERATOR>

σ_{23}= <0, >=   ,RELATIONAL_OPERATOR>

σ_{24}= <0, ==   ,RELATIONAL_OPERATOR>

σ_{25}= <0, /=   ,RELATIONAL_OPERATOR>

σ_{26}= <2,subtype is ,SUBTYPE_NAME TYPE_DESIGNER SUBTYPE_DEFINITION>

σ_{27}= <3,subtype is ,SUBTYPE_NAME TYPE_DESIGNER CONSTRAINT SUBTYPE_DEFINITION>

σ_{28}= <1,   ,TYPE_NAME TYPE_DESIGNER>

σ_{29}= <1,   ,SUBTYPE_NAME TYPE_DESIGNER>

σ_{30}= <1,   ,PREDEFINITION_TYPE TYPE_DESIGNER>

σ_{31}= <1,   ,CONSTRAINT NUMERICAL_TYPE>

σ_{32}= <2, new   ,PREDEFINITION_TYPE CONSTRAINT NUMERICAL_TYPE>

σ_{33}= <1,   ,INTEGER PREDEFINITION_TYPE>

σ_{34}= <1,   ,FLOAT PREDEFINITION_TYPE>

σ_{35}= <2, :   ,IDENTIFIER_LIST TYPE_DESIGNER OBJECT_DECLARATION>

σ_{36}= <3, : :=   ,IDENTIFIER_LIST TYPE_DESIGNER SIMPLE_EXPRESSION OBJECT_DECLARATION>

σ_{37}= <1,   ,IDENTIFIER IDENTIFIER_LIST>

σ_{38}= <2, ,   ,IDENTIFIER_LIST IDENTIFIER IDENTIFIER_LIST>

σ_{39}= <1,record end record, COMPONENT_LIST RECORD_TYPE>

σ_{40}= <1,   ,OBJECT_DECLARATION COMPONENT_LIST>

σ_{41}= <2, ,   ,COMPONENT_LIST OBJECT_DECLARATION COMPONENT_LIST>

σ_{42}= <1,   ,INDEX INDEX_LIST>

σ_{43}= <2, ,   ,INDEX_LIST INDEX INDEX_LIST>

σ_{44}= <2,array( ) of ,INDEX_LIST TYPE_DESINGER ARRAY_TYPE>

σ_{45}= <1,   ,RANGE INDEX>

σ_{46}= <1,   ,INTEGER INDEX>

σ_{47}= <2, :=   ,VARIABLE SIMPLE_EXPRESSION ASSIGNMENT_STATEMENT>

σ_{48}= <2, then   ,RELATION SEQUENCE EL_IF>

σ_{49}= <1,   ,EL_IF IF_LIST>

σ_{50}= <2, elsif   ,IF_LIST EL_IF IF_LIST>

σ_{51}= <1,if endif,IF_LIST IF>

σ_{52}= <2, if else endif,IF_LIST SEQUENCE IF>

σ_{53}= <1,   ,ASSIGNMENT_STATEMENT ASSIGN_LIST>

σ_{54}= <2, ,   ,ASSIGN_LIST ASSIGNMENT_STATEMENT ASSIGN_LIST>

σ_{55}= <1,   ,ASSIGN_LIST SEQUENCE>

3. Basic concepts. Let $\underline{B} = \langle I \cup S,\ \Sigma S,\ \alpha\rangle$ be the specification base for an arbitrary programming language. $\underline{B}$ is organized under the form of a homogeneous algebra. I is the set of all object names, S is the reserved word set, such that $S \cap I = \emptyset$; ΣS is the set of operation schemes represented as triples: $\Sigma S = \{\sigma = \langle n,\ s_0 s_1 \ldots s_n,\ i_1 i_2 \ldots i_n i\rangle\}$; α is the set of axioms given as couples of words (w_1, w_2). These words are generated by the operations specified by the operation schemes $\sigma \in \Sigma S$. For an operation scheme $\sigma \in \Sigma S$, $\sigma = \langle n, s_0 s_1 \ldots s_n,\ i_1 i_2 \ldots i_n i\rangle$, the function

$$F_\sigma : A_{i_1} X A_{i_2} X \ldots X A_{i_n} \rightarrow A_i$$

specifies a signature for a heterogeneous operation in the language specified by $\underline{B}$. Consider an arbitrary subset $J \subseteq I$, to which we shall refer in what follows.

DEFINITION 1. *The operation scheme* $\sigma' = \langle n, s_0 s_1 \ldots s_n,$

$j_1 j_2 \ldots j_n i>$, $j_k \in J$, $k=1,2,\ldots,n$, is called restriction of the operation scheme $\sigma \in \Sigma S$, $\sigma = <n, s_0 s_1 \ldots s_n, i_1 i_2 \ldots i_n i>$ to the subset J if there exists the function $F_{\sigma'} : A_{j_1} X A_{j_2} X \ldots X A_{j_n} \to A_i$ which is a restriction of the function $F_\sigma : A_{i_1} X A_{i_2} X \ldots X A_{i_n} \to A_i$.

For a given subset J, an operation scheme σ can have one or several restrictions. We denote by $\sigma|_J$ the set of all restrictions of σ to J, namely:

$$\sigma|_J = \{ \sigma' | \sigma' \text{ is a restriction of the operation scheme } \sigma \text{ to the subset } J \}.$$

Remark. If $\sigma' = <n, s_0 s_1 \ldots s_n, j_1 j_2 \ldots j_n i>$ is a restriction of the operation scheme $\sigma = <n, s_0 s_1 \ldots s_n, i_1 i_2 \ldots i_n i>$ to the subset J, then $A_{j_k} \subseteq A_{i_k}$, $\forall_{j_k}$, $k=1,2,\ldots,n$.

DEFINITION 2. *Let $\Sigma' = \{\sigma_1, \sigma_2, \ldots, \sigma_m\}$ be a set of operation schemes. A restriction of Σ' to the subset J is defined as follows:*

$$\sum{}'|_J = \{\sigma_i' | \sigma_i' \in \sigma_i|_J \text{ , } i=\overline{1,m} \}$$

DEFINITION 3. *The restriction of the specification base $\underline{B} = <I \cup S\, \Sigma S, \alpha>$ to the subset J is the specification base $\underline{B}|_J = <J \cup S, \Sigma S|_J, \alpha|_J>$, where $\alpha|_J$ is that subset of the axiom set α which consists of the axioms satisfied by the operations specified by $\Sigma S|_J$ (restriction of the operation scheme set ΣS to the subset J).*

DEFINITION 4. *An interpretation* [RUS83] *of the specification base $\underline{B} = <I \cup S, \Sigma S, \alpha>$ is the triple $<A, \rho, \Psi>$, where $A = (A_i)_{i \in I}$ is a family of sets indexed by the set I of the supports of $\underline{B}$; φ*

is a function $\varphi : I \to A$ *such that* $\varphi(i)$ *is an element of A for every* $i \in I$; Ψ *is the function* $\Psi : \Sigma S \to OP(A)$, *where* $OP(A)$ *is the set of all operations defined on A.*

The interpretation $\underline{A} = \langle A, \varphi, \psi \rangle$ of the specification base $\underline{B} = \langle I \cup S, \Sigma S, \alpha \rangle$ fulfils the following conditions:

C1. $\varphi(i) = A_i$ for every $i \in I$; A_i is called the support set for the name of the object i.

C2. For every $\sigma \in \Sigma S$, $\sigma = \langle n, s_0, s_1 \ldots s_n, i_1 i_2 \ldots i_n i \rangle$, $\Psi(\sigma) : \varphi(i_1) \times \varphi(i_2) \times \ldots \times \varphi(i_n) \to \varphi(i)$, or in other words $\Psi(\sigma) : A_{i_1} x A_{i_2} x \ldots x A_{i_n} \to A_i$, such that if $\sigma = \langle 0, s, i \rangle$ then $\Psi(\sigma) : \phi \to \varphi(i)$ or $\Psi(\sigma) : \phi \to A_i$, namely $\Psi(\sigma)$ is in this case the injection of the element $\Psi(\sigma)$, which will be denoted by s, in the set A_i.

C3. Every axiom (W_1, W_2) is considered to be satisfied in the interpretation $\underline{A} = \langle A, \varphi, \psi \rangle$; this means that it is satisfied by operations from $OP(A)$.

We shall hereafter denote by $\underline{I}(\underline{B})$ the set of interpretations of a specification base $\underline{B} = \langle I \cup S, \Sigma S, \alpha \rangle$.

If $\underline{A} \in \underline{I}(\underline{B})$, then it can be denoted $\underline{A} = \langle A = (A_i)_{i \in I}, \Sigma S, \psi \rangle$, where $\varphi(i) = A_i$ for every $i \in I$, and

$\psi(\sigma) : \varphi(i_1) \; x \; \varphi(i_2) x \ldots x \; \varphi(i_n) \to \varphi(i)$ or $\psi(\sigma) : A_{i_1} \; x \; A_{i_2} x \ldots x \; A_{i_n} \to$

for every $\sigma \in \Sigma S$, $\sigma = \langle n, s_0 s_1 \ldots s_n, i_1 i_2 \ldots i_n i \rangle$. In other words, every interpretation $\underline{A}$ of the specification base $\underline{B}$ is a heterogeneous algebra.

DEFINITION 5. *Let $\underline{B}$ =$<I \cup S, \Sigma S, \alpha>$ be a specification base, and let $\underline{A} = \langle A, \varphi, \psi \rangle$ be an interpretation; then $\underline{A}|_J = \langle A|_J, \varphi|_J, \psi|_J \rangle$ is called restriction of the interpretation $\underline{A} \in \underline{I}(\underline{B})$ to the subset J, where: $A|_J$ is the family of sets $(A_j)_{j \in J}$; $\varphi|_J$ is the restriction of the function φ to the subset J; $\Psi|_J$ is the restriction of the function Ψ to the operation scheme set $\Sigma S|_J$.*

PROPERTY 1. *Let $\underline{B}$ =$<I \cup S, \Sigma S, \alpha>$ be a specification base, and let $\underline{A} \in \underline{I}(\underline{B})$ an arbitrary interpretation . Then the restriction of $\underline{A}$ to the subset J, $\underline{A}|_J = \langle A|_J, \varphi|_J, \psi|_J \rangle$, is an interpretation of the restriction of $\underline{B}$ to the subset J, $\underline{B}|_J = \langle J \cup S, \Sigma S|_J, \alpha|_J \rangle$.*

Proof. The proof results immediately from Definitions 3,4 and 5.

For a given specification base $\underline{B}$ =$<I \cup S, \Sigma S, \alpha>$, an important interpretation is the so-called word-algebra or the heterogeneous *W*-algebra [RUS83] specified by the base $\underline{B}$. This is obtained as follows:

(i) One constructs the family of all words specified by the base $\underline{B}$ in the form $W=(W_i)_{i \in I}$, where every W_i is defined as follows:

r1. If $\sigma=<0,s,i>, \sigma \in \Sigma S$, then $s \in W_i$.

r2. If $\sigma \in \Sigma S$, $\sigma=<n, s_0 s_1 \ldots s_n, i_1 i_2 \ldots i_n i>$, and if $w_1, w_2, \ldots, w_n$ are words of the types $i_1, i_2, \ldots, i_n$, respectively, namely $(w_1, w_2, \ldots, w_n) \in W_{i1} x W_{i2} x \ldots x W_{in}$, then

$w = s_0w_1s_1, \ldots, s_{n-1}w_ns_n \in W_i$.

r3. The rules r1 and r2 describe all the words of the type i, which will sometimes be denoted by $W(i)$, too.

(ii) For the given specification base $\underline{B}$ one constructs the following interpretation:

p1. For every $i \in I$, $\varphi(i) = W(i)$.

p2. For every $\sigma \in \Sigma S$, $\sigma = <n, s_0s_1 \ldots s_n, i_1i_2 \ldots i_ni>$, the function $\Psi(\sigma) : W(i_1) x W(i_2) x \ldots x W(i_n) \to W(i)$ acts according to the law: if $w_k \in W(i_k)$, $k = 1, 2, \ldots, n$, then $\Psi(\sigma)(w_1, w_2, \ldots, w_n) = s_0w_1s_1 \ldots s_nw_n \in W(i)$.

(iii) Interpreting the specification axioms as formal identities [RUS83], follows that the family $W = (W(i))_{i \in I}$ together with the above defined functions (φ, ψ) form an interpretation of the considered specification base. This interpretation is written under the form

$$\underline{W}(\underline{B}) = <W = (W(i))_{i \in I}, \Sigma S, \psi>$$

Let $\underline{A}_1 = <A = (A_i)_{i \in I}, \Sigma S, \Psi_1>$, $\underline{A}_2 = <B = (B_i)_{i \in I}, \Sigma S, \Psi_2>$ be two similar heterogeneous algebras such that $\underline{A}_1, \underline{A}_2 \in \underline{I}(\underline{B})$.

DEFINITION 6. *A family of functions $f = (f_i)_{i \in I}$, $f_i : A_i \to B_i$, indexed by the set I is called similarity morphism or homomorphism from $\underline{A}_1$ to $\underline{A}_2$ if for every operation scheme $\sigma \in \Sigma S$, $\sigma = <n, s_0s_1 \ldots s_n, i_1i_2 \ldots i_ni>$ and for every*

$$(a_1, a_2, \ldots, a_n) \in A_{i_1} x A_{i_2} x \ldots x A_{i_n}$$

we have

$$f_i(\psi_1(\sigma)(a_1, a_2, \ldots, a_n)) = \psi_2(\sigma)(f_{i_1}(a_1), f_{i_2}(a_2), \ldots, f_{i_n}(a_n)).$$

DEFINITION 7. *Let $\underline{B} = <I \cup S, \Sigma S, \alpha>$ be a given specification base. A model of $\underline{B}$ is an interpretation $\underline{W} \in \underline{I}(\underline{B})$ with the* *property that for every other interpretation $\underline{A} \in \underline{I}(\underline{B})$ there exists exactly one homomorphism $f:\underline{W} \to \underline{A}$.*

In [RUS83] there is shown that for every specification base $\underline{B} = \langle I \cup S, \Sigma S, \alpha \rangle$ there exists a model of this base. This model is the word algebra $\underline{W}(\underline{B}) = \langle W=(Wi))_{i \in I}, \Sigma S, \Psi \rangle$. If $\underline{W}_1(\underline{B})$ and $\underline{W}_2(\underline{B})$ are two models of the base $\underline{B}$, then they are isomorphic.

Since the model $\underline{W}(\underline{B})$ belongs to the set of interpretations of $\underline{B}$, according to Definition 5 one can consider the restriction of the model $\underline{W}(\underline{B})$ to the subset J, written as

$$\underline{W}(\underline{B})|_J = \langle W|_J = (W(j))_{j \in J}, \Sigma S|_J, \Psi|_J \rangle .$$

Property-1 also holds for the model $\underline{W}(\underline{B})$, but may be reformulated as follows:

PROPERTY 2. *Let $\underline{B}=\langle I \cup S, \Sigma S, \alpha \rangle$ be a specification base. Then the restriction $\underline{W}(\underline{B})|_J$ of the model is equal to the model of the restriction of $\underline{B}$, $\underline{W}(\underline{B})|_J)$.*

THEOREM 1. *Let $\underline{B}=\langle I \cup S, \Sigma S, \alpha \rangle$ be a specification base, and let $\underline{W}(\underline{B})$ be a model of this base. For every $J \in \mathcal{P}(I)$, $J \neq \emptyset$, there exists a restriction $\underline{B}|_J$ of $\underline{B}$ whose model is $\underline{W}(\underline{B})|_J$ (here $\mathcal{P}(I)$ stands for the set of the subsets of I).*

Proof. Consider $J \in \mathcal{P}(I)$, $J \neq \emptyset$, and construct the restriction $\underline{B}|_J$. Let $\underline{W}(\underline{B})$ be the model of the specification base $\underline{B}$, and let $\underline{W}(\underline{B})|_J$ be the restriction of the model $\underline{W}(\underline{B})$ to the subset J. According to Property 2, follows that $\underline{W}(\underline{B})|_J = \underline{W}(\underline{B}|_J)$.

For a given specification base $\underline{B} = \langle I \cup S, \Sigma S, \alpha \rangle$, the construction of an interpretation $\underline{A} \in \underline{I}(\underline{B})$ supposes the choice of both the family of sets A and the operation set OP(A) defined on this family. The construction of an interpretation for a specification base corresponds to the construction of the level one (HAS_1) from the HAS (heterogeneous algebraic structure) hierarchy [RUS80] of a language specification. According to the principles of construction of a HAS hierarchy, the level one (HAS_1) will constitute the specification base for the level two (HAS_2). The transit from level one to level two is performed by constructing a new interpretation of the specification base HAS_1. This process can continue until the specified language satisfies the requests of the specifier. The new theoretical concepts presented in this paper are valid for every level HAS_i from a language specification. For simplicity reasons we discussed only the level one from the HAS hierarchy.

Let $LB_1=\langle A_1,\varphi_1,\psi_1\rangle$, $LB_2=\langle A_2,\varphi_2,\psi_2\rangle$ be two programming languages specified by the same base $\underline{B}$.

DEFINITION 8. *The language LB_1 is subset of LB_2 if $A_1 \subseteq A_2$, and, if $w_1 \in OP(A_1)$, then either $w_1 \in OP(A_2)$ or w_1 is a restriction of an operation $w_2 \in OP(A_2)$.*

COROLLARY. *Let $=\underline{B}=\langle I\cup S, \Sigma S, \alpha\rangle$ be a specification base. The language $LB|_J$ specified by the restriction $\underline{B}|_J$ of the base $\underline{B}$ is a subset of the language LB specified by the base $\underline{B}$.*

Proof. This is evident by virtue of Theorem 1.

One can construct in this way a large specification base $\underline{B}$ able to generate a subset of the object specified by $\underline{B}$, by means of the operation of restriction to a given subset J. Denoting by UL the language (object) specified by the base $\underline{B}$, a subset of UL can be obtained as an object specified by a restriction of $\underline{B}$.

4. Examples. The examples which follow concern the simple language specified from a syntactical point of view in Section 2. The set I consists of all words written in capitals from the context-free grammar of the language, while the set S consists of all words written in small letters from the same grammar.

Example 1. Consider J_1 = I\{PREDEFINITION_TYPE}. Remove from ΣS the operation schemes σ_{33}, σ_{34}, which are PREDEFINITION_TYPE. Replace the operation scheme σ_{30} by a restriction of this one to the subset J_1, denoted σ_{30}'=<1, ,INTEGER TYPE_DESIGNER>, and replace the operation scheme σ_{32} by a restriction of this one to the same subset J_1, denoted σ_{32}'=<2, new , INTEGER CONSTRAINT NUMERICAL_TYPE>. The restriction of the specification base $\underline{B}$ to the subset J_1 is $\underline{B}|_{J1}$ = $<J_1 \cup S$, $\Sigma S|_{J1}, \alpha|_{J1}>$, where $\Sigma S|_{J1}$ = $(\Sigma S \setminus \{\sigma_{30}, \sigma_{32}, \sigma_{33}, \sigma_{34}\}) \cup \{\sigma_{30}', \sigma_{32}'\}$. The following numerical-type declaration

type REAL_MIC is new FLOAT range -10.0..10.0

is correct in the language LB specified by $\underline{B}$, but not in the language $LB|_{J1}$ specified by $\underline{B}|_{J1}$. This last language admits the following numerical-type declaration

type INTEGER_MIC is new INTEGER range -10 .. 10

which is a restriction of the first declaration admitted by LB. Neither the operation scheme σ_{30} nor its restriction σ_{30}' contribute directly to the specification of objects of LB or $LB|_{J1}$, respectively. The objects of the type TYPE_DESIGNER specified by σ_{30}' will contribute to the specification of objects of the type ARRAY_TYPE, as we shall see in the next example. The axiom set $\alpha|_{J1}$ remains the same as the axiom set α. In these conditions the language $LB|_{J1}$ is a subset of the language LB.

Example 2. Consider $J_2 = J_1\backslash\{\text{TYPE\_DESIGNER}\}$. Remove from $\Sigma S|_{J1}$ the operation schemes σ_{28}, σ_{29}, which are TYPE_DESIGNER. We choose the following restrictions for σ_{26}

σ_{26}'=<2, subtype is ,subtype_NAME INTEGER SUBTYPE_DEFINITION>,

and for σ_{27}

σ_{27}' =<3, subtype is ,subtype_NAME INTEGER CONSTRAINT
SUBTYPE_DEFINITION>.

For σ_{35} and σ_{36} we choose respectively the restrictions

σ_{35}'=<2, : ,INTEGER_LIST INTEGER OBJECT_DECLARATION>,

σ_{36}'=<3, : := ,IDENTIFIER_LIST INTEGER SIMPLE_EXPRESSION
OBJECT_DECLARATION>.

Lastly, one chooses only one restriction for σ_{44}

σ_{44}'=<2, array( ) of ,INDEX_LIST INTEGER ARRAY_TYPE>.

In these conditions, $B|_{J_2} = \langle J_2\cup S, \sum S|_{J_2}, \alpha|_{J_2}\rangle$, *where*

$$\sum S|_{J_2} = (\sum S|_{J_1}\backslash \{\sigma_{26},\sigma_{27},\sigma_{28},\sigma_{29},\sigma_{35},\sigma_{36},\sigma_{44}\}\cup\{\sigma_{26}',\sigma_{27}',\sigma_{35}',\sigma_{36}',\sigma_{44}'\}.$$

The following array-type declaration

type ARR_TY is array (1..10, 1..10) of FLOAT

is correct in the languages LB and $LB|_{J1}$, but not in $LB|_{J2}$. The language $LB|_{J2}$ admits the definition of an array-type restriction (reduced from) whose elements are of integer-type. So, above array-type definition reduces in the language $LB|_{J2}$ to

type ARR_TY is array (1..10, 1..10) of INTEGER.

The following record-type declaration

type COMPLEX is

record

```
            REAL : FLOAT;
            IMAG : FLOAT;
        end record.
```

is correct in the languages LB and $LB|_{J1}$. The language specified by J_2 ,$LB|_{J2}$, admits only a restriction of the object COMPLEX, namely

```
        type COMPLEX is
          record
            REAL : INTEGER;
            IMAG : INTEGER;
        end record.
```

Also in this example the axiom sets $\alpha|_{J2}$ and α coincide. In these conditions the language $LB|_{J2}$ is a subset of the language $LB|_{J1}$.

Example 3. Consider $J_3 = J_2\backslash\{IF\_LIST\}$. Remove from $\Sigma S|_{J2}$ the operation schemes σ_{49} and σ_{50}, which are of the type IF_LIST. We choose respectively for σ_{51} and σ_{52} the restrictions

$\sigma_{51}{}' =$ <1, if endif, EL_IF IF>,

$\sigma_{52}{}' =$ <2, if else endif, EL_IF SEQUENCE IF>.

In these conditions $B|_{J_3} = \langle J_3 \cup S, \sum S|_{J_3}, \alpha|_{J_3}\rangle$, where $\sum S|_{J_3} = (\sum S|_{J_2}\backslash\{\sigma_{49}, \sigma_{50}, \sigma_{51}, \sigma_{52}\}) \cup \{\sigma_{51}', \sigma_{52}'\}$. An instruction of the form

```
    if RELATION then
                SEQUENCE
    {elsif RELATION then
                   SEQUENCE}
    [else
                SEQUENCE]
    endif.
```

is admitted in the languages LB, $LB|_{J1}$, $LB|_{J2}$. The language $LB|_{J3}$ admits the following reduced forms

```
    if RELATION then SEQUENCE endif,
    if RELATION then SEQUENCE
                     else SEQUENCE
                          endif.
```

The axiom set $\alpha|_{J3}$ coincides with α, too. In these conditions the language $LB|_{J3}$ is a subset of the language $LB|_{J2}$.

R E F E R E N C E S

[ADJ73] J.A. Goguen, J.W. Thatcher, G.Wagner, J.B. Wright: *A junction between computer science and category theory, Basic definitions and examples, Part 1,* IBM Research Report RC-4526, 1973.

[ADJ77] J.A. Goguen, J.W. Thatcher, E.G. Wagner, J.B. Wright: *Initial Algebra for Computing Machinery,* vol. 24, no.1, 1977.

[BRO82] M. Broy, M. Wirsing,: *Partial Abstract Types,* Acta Informatica, 18, 47-64 (1982).

[BRO87] M. Broy, M.Wirsing, P.Pepper: *On the Algebraic Definition of Programming Languages,* ACM Transactions on Programming Languages and Systems, vol 9., no.1, 1987.

[RUS72] T. Rus: *ΣS-Algebra of a Formal Language,* Bul. Math. de la Soc. de Science, Bucharest, 15(63):2, 227-235, 1972.

[RUS80] T. Rus: HAS-Hierarchy: *A natural tool for Languag specification,* Ann. Soc. Math. Polonae, Series IV. Fundamenta Informaticae, 3:3,269-274, 1980.

[RUS83] T. Rus: *Mecanisme formale pentru specificarea limbajelor,* Editura Academiei R.S.R., 1983, Bucuresti.

[HIG63] P.J. Higgins: *Algebras with a scheme of operators,* Math. Nachr., 27, 115-132, 1963/64.

[BIR70] G. Birkoff, J.D. Lipson: *Heterogeneous algebras,* Journal of Combinatorial Theory, 8, 115-132, 1970.

[WAG89] E.G. Wagner: *An Algebraically Specified Language for Data Directed Design,* Proceedings of the First International Conference "Algebraic Methodology and Software Technology", May 22-24, 1989, IOWA, U.S.A.

STUDIA UNIV. BABEŞ-BOLYAI, MATHEMATICA, XXXVII, 3, 1992

ALGEBRAIC SPECIFICATION OF PSP

Ilie PARPUCEA[*] **and Bazil PÂRV**[**]

Received: March 1, 1993
AMS Subject Classification: 68Q05

REZUMAT. - Specificarea algebrică a RSP. În lucrare se prezintă specificarea algebrică a tipurilor de date folosite în procesorul simbolic Poisson PSP ([Pâr89]). Se foloseşte un model ierarhic, bazat pe semantica denotaţională (algebrică).

0. Introduction. Algebraic specification of PSP (Poisson Symbolic Processor, see [Pâr89]) data types is made by means of an hierachical model, from simple to complex types. The abstract data structure of PSP is viewed as a hierachy of abstract data types.

A **Poisson series** have 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, ..., x_m are monomial variables; y_1, y_2, ..., y_n are trigonometric variables; j_1, j_2, ..., j_m and k_1, k_2, ..., 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 ($j \in \mathbf{Z}^m, k \in \mathbf{Z}^n$).

The form (1) of 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:

---

[*] *"Babeş-Bolyai" University, Department of Economic Sciences 3400 Cluj-Napoca, Romania*

[**] *"Babeş-Bolyai" University, Faculty of Mathematics and Computer Science, 3400 Cluj-Napoca, Romania*

$$T_i = C_i \; P_i \; F_i \; ,$$

where the *polynomial part* P_i has the form:

$$P_i = x_1^{j_1} x_2^{j_2} \ldots x_m^{j_m}, \tag{3}$$

while the *trigonometric part* F_i is:

$$F_i = {\sin \atop \cos} (k_1 y_1 + k_2 y_2 + \ldots + k_n y_n) . \tag{4}$$

In practice, one does not operate with Poisson series, but with partial sums of these ones, called *Poisson expressions*, of the form:

$$S = \sum_{i=0}^{N} T_i, \quad N \in \mathbf{N}. \tag{5}$$

PSP operates with Poisson expressions of the form (5). The hierarchical model of its algebraic specification consists of five levels:

1) numerical coefficients specification;
2) trigonometric part specification;
3) polynomial part specification;
4) term specification;
5) Poisson expression specification.

1. Numerical coefficients specification. The coefficients C_i from (1) are considered rational numbers, of the form M/N, with $M,N \in \mathbf{Z}$. The definition of the abstract data type RAT follows the chain:

BOOL --> NAT --> INT --> RAT

where:

BOOL - represents the primitive boolean type;

NAT - represents the hierarchical natural type (including zero value);

INT - represents the hierarchical integer type;

RAT - represents the hierarchical rational type.

In the specification of the NAT type we use the NAT* subtype of NAT, which corresponds to the natural numbers without zero value. The above listed types are specified as follows:

1.1. The primitive BOOL type

```
type BOOL ≡
     SORT bool,
     CONS
          true: --> bool,
          false: --> bool,
     OPNS
          .not.: bool --> bool,
          .and.: bool, bool --> bool,
          .or.:  bool, bool --> bool,
     VARS
           X,Y: bool,
     AXIOMS
           true ≠ false,
           .not. true = false,
           .not. false = true,
           true .and. X = X,
           false .and. X = false,
           X .or. Y = .not.((.not. X) .and. (.not. Y))
endoftype.
```

1.2. The NAT and NAT* types

The NAT type uses the primitive type BOOL:

```
type NAT ≡
     SORT bool, nat,
     CONS
          zero: --> nat,
          succ: nat --> nat,
          pred: (nat x: x noteq zero) --> nat,
     OPNS
          *: nat, nat --> nat,
          eq: nat, nat --> bool,
          +: nat, nat --> nat,
          noteq: nat, nat --> bool,
          < : nat, nat --> bool,
          [_,_]: nat, (nat x: x noteq zero) --> nat,
          GCD(_,_): nat, nat --> nat,
     VARS
          R, P, M, N: nat,
     AXIOMS
     *    N eq M  =  M eq N,
      *    zero eq zero  = true,
      *    zero eq succ(N)  =  false,
      *    succ(N) eq succ(M)  =  N eq M,
      *    N noteq M  =  not (N eq M),
      *    pred(succ(N))  =  N,
           [N,1]  =  N,
           (∃ R, P: nat : R < M, N = P * M + R) ==> [N , M]  = P,
           GCD(N, 0)  =  N,
           U < V ==> GCD(V, U)  =  GCD(U, V - U * [V , U]),
           GCD(U, V)  =  GCD(V, U),
      *    N * 0  =  0  =  0 * N,
      *    N * succ(M)  =  (N * M) + N  =  succ(M) * N,
      *    N * pred(M)  =  (N * M) - N  =  pred(M) * N,
      *    N + 0  =  0 + N  =  N,
      *    N + succ(M)  =  succ(N + M)  =  succ(M) + N,
      *    N + pred(M)  =  pred(N + M)  =  pred(M) + N,
           (0 < succ(0))  =  true,
           (pred(N) < succ(N))  =  true,
           (N < succ(N))  =  true,
           (pred(N) < N)  =  true,
           N < M  ==>  (succ(N) < succ(M)  =  true),
           N < M  ==>  (pred(N) < pred(M)  =  true),
      *    N - 0  =  N,
      *    IF  M < N  THEN  N - succ(M)  =  pred(N - M),
      *    IF  M < N  THEN  N - pred(M)  =  succ(N - M),
endoftype.
```

We define NAT*, subtype of the NAT type:

```
type NAT* =
     SORT bool, nat*,
     CONS
          one: --> nat*,
          succ: nat* --> nat*,
          pred: (nat* x: x noteq one) --> nat*,
     OPNS
          +: nat* --> nat*,
          -: nat* --> nat*,
          *: nat*, nat* --> nat*,
          eq: nat*, nat* --> bool,
          noteq: nat*, nat* --> bool,
     VARS
          M, N: nat*,
     AXIOMS
          The axioms of NAT type marked with * are axioms of the
           subtype NAT*, with the following modifications:
          one eq one  =  true,
          one eq succ(N)  =  false,
          N * 1  =  1 * N  =  N,
          N + 1  =  1 + N  =  succ(N),
          IF  1 < N  THEN  N - 1  =  pred(N),
endoftype.
```

1.3. The INT type

The specification of INT type uses BOOL and NAT as primitive types:

```
type INT ≡
     SORT bool, int, nat,
     CONS
          zero: --> int,
          succ: int --> int,
          pred: int --> int,
     OPNS
          eq: int, int: --> bool,
          noteq: int, int --> bool,
          +: int, int --> int,
          -: int, int --> int,
          *: int, int --> int,
          <: int, int --> bool,
          |_|: int --> nat,
     VARS
          N, M: int,
     AXIOMS
          N eq M  =  M eq N,
          zero eq zero  =  true,
          succ(N) eq succ(M)  =  N eq M,
          N noteq M  =  not (N eq M),
          pred(succ(N))  =  N,
          succ(pred(N))  =  N,
          | 0 |  =  0,
          (pred(0) < 0)  =  true,
          The axioms of INT which refer to the operations +, -
             and < are inherited from NAT type;
endoftype.
```

Remark. The INT type contains the *zero* value and two unary operations, *succ* and *pred*. The integer number n (n>0) is obtained by n successive applications of the *succ* operation to 0. Analogously, the integer number -n (n>0) is constructed with *pred*, starting with 0.

1.4. The RAT type

The specification of RAT type uses INT and NAT* as primitive types:

```
type RAT ≡
     SORT int, nat*, rat,
     CONS
```

```
          _/_: int, nat* --> rat,
     OPNS
          _+_: rat, rat --> rat,
          _*_: rat, rat --> rat,
          _:_: rat, rat --> rat,
          _-_: rat, rat --> rat,
     VARS
          M/N, P/Q: rat,
          S, T: nat*,
     AXIOMS
          (M/N) + (P/Q)  = ((M * Q) + (P * N)) / (N * Q),
          (M/N) * (P/Q)  = (M * P) / (N * Q),
          (M/N) : (P/Q)  = (M * Q) / (N * P),
          (M/N) - (P/Q)  = ((M * Q) - (P * N)) / (N * Q),
          (∃ S: nat*, ∃ T: nat*, :
             M  =  S * GCD(|M|,|N|) ∧ N  =  T * GCD(|M|,|N|)) →
             → M/N = S/T
endoftype.
```

2. Trigonometric part specification. The definition of the trigonometric parts F_i from (4) follows the chain:

SET --> FLIN --> TTR

where:

SET - represents a symbol set;

FLIN - represents the abstract type of linear forms;

TTR - represents the abstract type of trigonometric parts.

2.1. The SET type

The SET type consists of a set of symbols. This type is used as primitive type for the specification of FLIN type.

```
type SET =
     SORT set,
     CONS
          X1 : --> set,
          X2 : --> set,
          - - - - - -
          Xn : --> set,
          Y1 : --> set,
          Y2 : --> set,
          - - - - - -
          Ym : --> set
endoftype.
```

2.2. The FLIN type

The FLIN type defines the linear forms over the types SET and INT.

```
type FLIN =
     SORT int, set, flin,
     CONS
          _·_ (concatenation) : int, set --> flin,
     OPNS
          _+_: flin, flin --> flin,
          _-_: flin, flin --> flin,
          _*_: int, flin --> flin,
     VARS
          X, Y, Z: set,
          M, N, K: int,
     AXIOMS
          X·1  =  1·X  =  X,
          X·0  =  0·X  =  0,
          N·X  =  X·N,
          X·(N + M)  =  (N + M)·X  =  N·X + M·X,
          N·X + M·Y  =  M·Y + N·X,
          (N·X + M·Y) + K·Z  =  N·X + (M·Y + K·Z),
          N·X + 0  =  0 + N·X  =  N·X,
          N * 0  =  0  =  0 * N,
          1 * X  =  X * 1  =  X,
          N * (M·X)  =  (N * M)·X,
          N * (M·X ± K·Y)  =  (N * M)·X ± (M * K)·Y
endoftype.
```

The FLIN type contains linear combinations of the form:

$$N_1 \cdot Y_1 + N_2 \cdot Y_2 + \ldots + N_k \cdot Y_k$$

which can be used as arguments for sine and cosine functions in trigonometric parts F_i. The three operations (+, * and concatenation), together with the axioms, are used for the specification of the trigonometric part.

2.3. The TTR type

```
type TTR =
     SORT flin, ttr,
     CONS
          cos: flin --> ttr,
          sin: flin --> ttr,
     AXIOMS
          cos 0  =  1,
          sin 0  =  0,
endoftype.
```

3. Polynomial part specification. The definition of the polynomial parts P_i from (3) follows the scheme:

MPOL --> PPOL

where:

MPOL - represents the set of monomials;

PPOL - represents the set of polynomials.

3.1. The MPOL type

The MPOL type defines the set of monomials of the form X^N, where X is in SET and N is of INT type.

```
type MPOL =
     SORT int, set, mpol,
     CONS
          _^_ (raise to power) : set, int --> mpol
                   ( X^N  =  X^N )
     VARS
          X: set, N: int,
     AXIOMS
          (X ^ 0)  =  1,
          (X ^ 1)  =  X,
endoftype.
```

3.2. The PPOL type

The PPOL type defines the set of polynomials of the form (3):

```
type PPOL =
     SORT mpol, ppol
     CONS
          _*_: mpol, mpol --> mpol,
     VARS
          X^M , X^N , Y^M , Z^K : ppol,
     AXIOMS
          X^M  * X^N   =  X^N  * X^M   =  X^(M+N)  ,
          X^N  * Y^M   =  Y^M  * X^N  ,
          X^N  * 1  =  X^N  =  1 * X^N ,
          X^N  * (Y^M  * Z^K )  =  (X^N  * Y^M ) * Z^K
endoftype.
```

4. Term specification. The terms T_i from (2) are defined as follows:

```
type TERM ≡
     SORT rat, ttr, ppol, term,
     CONS
          · (concatenation) : rat, ttr, ppol --> term,
     VARS
          N/M   : rat, cosY, sinY: ttr, X: ppol,
* * * We denote with Y the linear form and with X the polynomial
* * * form
     AXIOMS
                    sinY                   sinY
          (N/M · {      }) · X = N / M · ({      }) · X  ,
                    cosY                   cosY
                   sinY         sinY                sinY
          N/M · {      } · X = {      } · N/M · X = {      } · X · N/M
                   cosY         cosY                cosY
                                                 sinY
                                   = X · N/M · {      } ,
                                                 cosY
          0 · 0 · X = 0 ,
          1/1 · 1 · X = X ,
               sinY
          0 · {      } · X = N/M · 0 · X = 0
               cosY
endoftype.
```

5. Poisson expression specification. Taking into account the above definitions, the Poisson expressions (5) will be defined in the following form:

```
type EXP ≡
     SORT term, exp,
     CONS
          _+_: term, term --> exp,
          _-_: term, term --> exp,
          _*_: term, term --> exp,
     OPNS
          ∂ (differentiation) : exp, set --> exp,
          ∫ (integration) : exp, set --> exp,
     VARS
          N/M , P/Q : rat,
          Y, Y1 , Y2 : flin,
          X, X1 , X2  : mpol,
     AXIOMS
                   sinY                sinY
          N/M · {      } · X ± P/Q · {      } · X =
                   cosY                cosY
```

$$= (N/M \pm P/Q) \cdot X \cdot \left\{ \begin{matrix} \sin Y \\ \cos Y \end{matrix} \right\} ,$$

$$(N/M \cdot \cos Y_1 \cdot X_1) * (P/Q \cdot \sin Y_2 \cdot X_2) =$$

$$= (1/2 * N/M * P/Q) \cdot X_1 \cdot X_2 \cdot \sin(Y_1+Y_2) +$$

$$+ (1/2 * N/M * P/Q) \cdot X_1 \cdot X_2 \cdot \sin(Y_2-Y_1) ,$$

$$(N/M \cdot \sin Y_1 \cdot X_1) * (P/Q \cdot \sin Y_2 \cdot X_2) =$$

$$= (1/2 * N/M * P/Q) \cdot X_1 \cdot X_2 \cdot \cos(Y_1-Y_2) -$$

$$- (1/2 * N/M * P/Q) \cdot X_1 \cdot X_2 \cdot \cos(Y_1+Y_2) ,$$

$$(N/M \cdot \cos Y_1 \cdot X_1) * (P/Q \cdot \cos Y_2 \cdot X_2) =$$

$$= (1/2 * N/M * P/Q) \cdot X_1 \cdot X_2 \cdot \cos(Y_1+Y_2) +$$

$$+ (1/2 * N/M * P/Q) \cdot X_1 \cdot X_2 \cdot \cos(Y_1-Y_2) ,$$

$$\frac{\partial}{\partial Y_k} (N/M \cdot \left\{ \begin{matrix} \cos(N_1 \cdot Y_1+N_2 \cdot Y_2+\ldots+N_k \cdot Y_k+\ldots+N_l \cdot Y_l) \\ \sin(N_1 \cdot Y_1+N_2 \cdot Y_2+\ldots+N_k \cdot Y_k+\ldots+N_l \cdot Y_l) \end{matrix} \right\} \cdot$$

$$\cdot X_1^{M1} \cdot \ldots \cdot Y_k^{Mk} \cdot \ldots \cdot X_h^{Mh}) =$$

$$= N*M_k/M \cdot \left\{ \begin{matrix} \cos(N_1 \cdot Y_1+N_2 \cdot Y_2+\ldots+N_k \cdot Y_k+\ldots+N_l \cdot Y_l) \\ \sin(N_1 \cdot Y_1+N_2 \cdot Y_2+\ldots+N_k \cdot Y_k+\ldots+N_l \cdot Y_l) \end{matrix} \right\} \cdot$$

$$\cdot X_1^{M1} \cdot \ldots \cdot Y_k^{Mk-1} \cdot \ldots \cdot X_h^{Mh}) \mp$$

$$\mp N*N_k/M \cdot \left\{ \begin{matrix} \sin(N_1 \cdot Y_1+N_2 \cdot Y_2+\ldots+N_k \cdot Y_k+\ldots+N_l \cdot Y_l) \\ \cos(N_1 \cdot Y_1+N_2 \cdot Y_2+\ldots+N_k \cdot Y_k+\ldots+N_l \cdot Y_l) \end{matrix} \right\} \cdot$$

$$\cdot X_1^{M1} \cdot \ldots \cdot Y_k^{Mk} \cdot \ldots \cdot X_h^{Mh}) ,$$

$$I_0^{N_k} = \int N/M \cdot \sin(N_1 \cdot Y_1+N_2 \cdot Y_2+\ldots+N_k \cdot Y_k+\ldots+N_l \cdot Y_l) \cdot$$

$$\cdot X_1^{M_1} \cdot \ldots . X_{k-1}^{M_{k-1}} \cdot Y_k^{0} \cdot X_{k+1}^{M_{k+1}} \cdot \ldots \cdot X_h^{M_h} \, dY_k =$$

$$= (-1/N_k*N/M) \cdot \cos(N_1 \cdot Y_1+N_2 \cdot Y_2+\ldots+N_k \cdot Y_k+\ldots+N_l \cdot Y_l) \cdot$$

$$\cdot X_1^{M_1} \cdot \ldots . X_{k-1}^{M_{k-1}} \cdot Y_k^{0} \cdot X_{k+1}^{M_{k+1}} \cdot \ldots \cdot Y_h^{M_h}$$

$$I_1^{N_k} = \int N/M \cdot \sin(N_1 \cdot Y_1+N_2 \cdot Y_2+\ldots+N_k \cdot Y_k+\ldots+N_l \cdot Y_l) \cdot$$

$$\cdot X_1^{M_1} \cdot \ldots . X_{k-1}^{M_{k-1}} \cdot Y_k^{1} \cdot X_{k+1}^{M_{k+1}} \cdot \ldots \cdot X_h^{M_h} \, dY_k =$$

$$= (-1/N_k*N/M) \cdot \cos(N_1\cdot Y_1+N_2\cdot Y_2+\ldots+N_k\cdot Y_k+\ldots+N_l\cdot Y_l)\cdot$$

$$\cdot X_1^{M_1}\cdot\ldots\cdot X_{k-1}^{M_{k-1}}\cdot Y_k^{1}\cdot X_{k+1}^{M_{k+1}}\cdot\ldots\cdot X_h^{M_h} +$$

$$+ (1/(N_k*N_k)) \cdot \sin(N_1\cdot Y_1+N_2\cdot Y_2+\ldots+N_k\cdot Y_k+\ldots+N_l\cdot Y_l)\cdot$$

$$\cdot X_1^{M_1}\cdot\ldots\cdot X_{k-1}^{M_{k-1}}\cdot Y_k^{0}\cdot X_{k+1}^{M_{k+1}}\cdot\ldots\cdot X_h^{M_h}$$

$$I_p^{N_k} = \int N/M \cdot \sin(N_1\cdot Y_1+N_2\cdot Y_2+\ldots+N_k\cdot Y_k+\ldots+N_l\cdot Y_l)\cdot$$

$$\cdot X_1^{M_1}\cdot\ldots\cdot X_{k-1}^{M_{k-1}}\cdot Y_k^{p}\cdot X_{k+1}^{M_{k+1}}\cdot\ldots\cdot X_h^{M_h}\, dY_k =$$

$$= (-1/N_k*N/M) \cdot \cos(N_1\cdot Y_1+N_2\cdot Y_2+\ldots+N_k\cdot Y_k+\ldots+N_l\cdot Y_l)\cdot$$

$$\cdot X_1^{M_1}\cdot\ldots\cdot X_{k-1}^{M_{k-1}}\cdot Y_k^{M_k}\cdot X_{k+1}^{M_{k+1}}\cdot\ldots\cdot X_h^{M_h} +$$

$$+(N/M*p/(N_k*N_k))\cdot\sin(N_1\cdot Y_1+N_2\cdot Y_2+\ldots+N_k\cdot Y_k+\ldots+N_l\cdot Y_l)\cdot$$

$$\cdot X_1^{M_1}\cdot\ldots\cdot X_{k-1}^{M_{k-1}}\cdot Y_k^{p-1}\cdot X_{k+1}^{M_{k+1}}\cdot\ldots\cdot X_h^{M_h} -$$

$$- (N/M*p/N_k*(p-1)/N_k) \cdot I_{p-2}^{N_k}$$

endoftype.

Remark. The EXP type must also contain the axioms referring to the recurrent computation of the integral:

$$J_p^{M_k} = \int N/M \cdot \cos(N_1\cdot Y_1+N_2\cdot Y_2+\ldots+N_k\cdot Y_k+\ldots+N_l\cdot Y_l)\cdot$$

$$\cdot X_1^{M_1}\cdot\ldots\cdot X_{k-1}^{M_{k-1}}\cdot Y_k^{p}\cdot X_{k+1}^{M_{k+1}}\cdot\ldots\cdot X_h^{M_h}\, dY_k =$$

$$= (-1/N_k*N/M) \cdot \sin(N_1\cdot Y_1+N_2\cdot Y_2+\ldots+N_k\cdot Y_k+\ldots+N_l\cdot Y_l)\cdot$$

$$\cdot X_1^{M_1}\cdot\ldots\cdot X_{k-1}^{M_{k-1}}\cdot Y_k^{p}\cdot X_{k+1}^{M_{k+1}}\cdot\ldots\cdot X_h^{M_h} +$$

$$+(N/M*p/(N_k*N_k))\cdot\cos(N_1\cdot Y_1+N_2\cdot Y_2+\ldots+N_k\cdot Y_k+\ldots+N_l\cdot Y_l)\cdot$$

$$\cdot X_1^{M_1}\cdot\ldots\cdot X_{k-1}^{M_{k-1}}\cdot Y_k^{p-1}\cdot X_{k+1}^{M_{k+1}}\cdot\ldots\cdot X_h^{M_h} -$$

$$- (N/M*p/N_k*(p-1)/N_k) \cdot J_{p-2}^{p}$$

CONCLUDING REMARKS. This paper specifies the data types defined in PSP (implemented in Pascal) in a hierarchical way. Some of the advantages of algebraic specifications are used, especially those involving the correctness of the defined operations. In such a way the singular cases (as non-determinations or exceptions) are avoided. The authors intention is to simulate the operational semantics of PSP by using terms rewrite rules. PSP can also be specified in the specification and programming language OBJ3 (see [Gog88] and [Kir87]).

R E F E R E N C E S

[Wir82] Wirsing,M., Broy,M.: *An analysis of semantic models for algebraic specification,* in Broy,M., Schmidt,G.(eds.), Theoretical foundations of programming methodology, Reidel, Dordrecht, 1982, 351-412.

[Wir83] Wirsing,M.,Pepper,P.,Partsch,H.,Dosch,W.,Broy,M.: *On hierarchies of abstract data types,* Acta Informatica, 20, 1983, 1-33.

[Kir87] Kirchner,C.,Kirchner,H.,Meseguer,J.: *Operational semantics of OBJ3,* Technical Report, SRI International, 1987.

[Gog88] Goguen,J.,Winkler,T.: *Introducing OBJ3,* Technical Report, SRI International, 1988.

[Pâr89] Pârv,B.: *Poisson Symbolic Processor,* Studia, Mathematica, XXXIV, 1989, No.3, 17-29.

STUDIA UNIV. BABEŞ-BOLYAI, MATHEMATICA, XXXVII, 3, 1992

CORP MODELLING USING FORMAL LANGUAGES

V. PREJMEREAN, V. CIOBAN and S. MOTOGNA[*]

Received: November 11, 1992
AMS Subject Classification : 68N20, 68Q52

REZUMAT. Modelarea corpurilor folosind limbajele formale. Definirea unui corp tridimensional se poate realiza de multe ori mai natural şi mai eficient utilizînd comenzi de deplasare a "cursorului" (vîrf de creion imaginar) în diverse direcţii. În acest fel se pot defini muchiile sau chiar suprafeţele unui corp, prin "plimbarea" cursorului pe direcţi paralele cu axele de coordonate. Pentru aceasta am definit un limbaj de comandă şi o serie de transformări elementare : translaţie, rotaţie, scalare, simetrie şi proiecţie pentru reprezentarea corpurilor descrise.

Solid corps modelling (tridimensional graphic objects) through front specification can be realised covering (describing) the edges ("wire-frame" method). This cover may be realised by moving commands of the cursor on a tridimensional frame this way: *up, down, right, left, front, back*. These moving commands compound the alphabet needed in the object outline description, which we want to model :

$$\Sigma = \{u, d, r, l, f, b\}$$

$w \in \Sigma^*$ is called command word of the cursor.

We will use for object definition an infinite frame of points obtained through intersection of the parallel plans with xOy, xOz and yOz at distance equal to 1. Two points $P(x,y,z)$ and $P'(x',y',z')$ are "neighbours" if we have the following relation :

$$|x-x'|+|y-y'|+|z-z'| = 1 \qquad (*)$$

This relation can be generalized :

$$x' = x + dx$$

$$y' = y + dy$$

[*] *"Babeş-Bolyai" University, Department of Computer Science, 3400 Cluj-Napoca, Romania*

$z' = z + dz$

where dx, dy, $dz \in \{-1,0,1\}$, $|dx|+|dy|+|dz| > 0$, which need an extension of the alphabet Σ by definition of other commands, obtained through composition of the existing commands.

We will use definition (*) to simplify grammars.

One command word $w \in \Sigma^*$ is, in fact, a succession of commands to the cursor and defines a "walk on the frame".

For a point $P(x,y,z)$ we define its succesors on the six directions this way :

- up : $SUC((x,y,z),u) := (x,y,z+1)$
- down : $SUC((x,y,z),d) := (x,y,z-1)$
- left : $SUC((x,y,z),l) := (x,y-1,z)$
- right : $SUC((x,y,z),r) := (x,y+1,z)$
- front : $SUC((x,y,z),f) := (x+1,y,z)$
- back : $SUC((x,y,z),b) := (x-1,y,z)$

The design specified in this way by a command word $w \in \Sigma^*$ is defined as follows :

$SUC : Z^3 \times \Sigma^* \dashrightarrow Z^3$

$SUC((x,y,z),\epsilon) = (x,y,z)$

and

$SUC((x,y,z),aw) = SUC(SUC(x,y,z),a),w)$

We notice that the cursor movements are executed from left to right. The cursor movement may be defined initializing the cursor in $P_0(x_0,y_0,z_0)$ and then moving through a command word
$w \in \Sigma^*$: $MOV(w) = SUC((x_0,y_0,z_0),w)$.

A spatial graph unoriented described by

$w = a_1 \ldots a_n$, ($a_i \in \{u,d,l,r,f,b\}$, $i=1,n$)

is $g((x_0,y_0,z_0),w) = (V,A)$,

where $V = \{ \mathrm{MOV}(a_1 \ldots a_i) /\ i=1,n\}$

and

$A = \{(\mathrm{MOV}(a_1 \ldots a_{i-1}), \mathrm{MOV}(a_1 \ldots a_i) /\ i=1,n\}$.

We suppose that $\mathrm{MOV}(\epsilon) = (x,y,z)$.

A language formed of command words is called language of solid object description, and the grammar which generates the language is called solid forms grammar.

To represent the objects described by such grammars we will define functions for spatial transformations.

TRANSLATION :

$$\mathrm{TR}(g((x_0,y_0,z_0),w),(dx,dy,dz)) = g((x_0+dx,y_0+dy,z_0+dz),w) =$$

$$= g((x_0,y_0,z_0), S_x^{|dx|} S_y^{|dy|} S_z^{|dz|} w)$$

$$S_x = \begin{cases} f & dx>0 \\ \varepsilon & dx=0 \\ b & dx<0 \end{cases}$$

$$S_y = \begin{cases} l & dy<0 \\ \varepsilon & dy=0 \\ r & dy>0 \end{cases}$$

$$S_z = \begin{cases} u & dz>0 \\ \varepsilon & dz=0 \\ d & dz<0 \end{cases}$$

So, if $V = (dx,dy,dz)$ is the translation vector then we may say that $TR(g(w),V) = g(S_x^{|dx|}\ S_y^{|dy|}\ S_z^{|dz|}\ w)$.

For rotations with 90°, 180°, 270° or even with 45°, 135°, 225° and 315° we can apply command word transformations depending on the axis of the rotation Ox, Oy or Oz. For example, rotation around Ox will change the word *w* so :

f, *b* - remain unchanged

$u \dashrightarrow r$

$d \dashrightarrow l$

$l \dashrightarrow u$

$r \dashrightarrow d$.

In the same way the substitutions applied to the productions rules of the grammar, for the other angles and axis are defined.

SCALATION :

A point $P(x,y,z)$ through scalation become $P'(x',y',z')$, where:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = f_g \begin{bmatrix} f_x & 0 & 0 \\ 0 & f_y & 0 \\ 0 & 0 & f_z \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

If $f_g f_x$, $f_g f_y$, $f_g f_z \geq 1$ then the image grows and we will consider these scalar factors natural.

$$\text{If } f_g \cdot f_x,\ f_g \cdot f_y,\ f_g \cdot f_z < 1$$
$$f_g \cdot f_x = \frac{1}{f'_x}\ ,\quad f_g \cdot f_y = \frac{1}{f'_y}\ ,\quad f_g \cdot f_z = \frac{1}{f'_z}\ \text{and}$$
$$f'_x\ ,\ f'_y\ ,\ f'_z \in \mathbb{N}$$

then the scalars applied to a command word $w = a_1 \ldots a_n$ will modify the word as follows :

$a_i \dashrightarrow a_i^f$, where

$$f = \begin{cases} f_g \cdot f_z & if\ a_i \in \{u, d\} \\ f_g \cdot f_y & if\ a_i \in \{l, r\} \\ f_g \cdot f_x & if\ a_i \in \{f, b\} \end{cases}$$

These transformations may be applied to the grammars which generate the command language.

SYMETRIES :

Reflections in planes xOy, xOz, yOz or in axis Ox, Oy, Oz are realised through such substitutions :

reflection in xOy: $u \leftrightarrow d$

xOz: $l \leftrightarrow r$

yOz: $f \leftrightarrow b$.

$SIM_{Oz}(g((x_0,y_0,z_0),w)) = g((-x_0,-y_0,z_0),w')$, where w' is obtained by substituting $l \leftrightarrow r$ and $f \leftrightarrow b$.

In the same way we construct reflections in Ox and Oy.

PROJECTIONS :

Projection parallel to Oz on xOy is defined as :

$$PR_{xOy}(g(w)) = \begin{cases} g(d^{z_0} w') & if\ z_0 > 0 \\ g(u^{z_0} w') & if\ z_0 < 0 \end{cases}$$

where w' is obtain by substituting "u" or "d" with ϵ.

We can realise images with solid forms grammars extending the alphabet Σ with simbols representing commands to choose (select) colours, where the null colour , "dark", shift the cursor without effective translation :

$\Sigma' = \Sigma \cup \{0,1,\ldots,7\}$.

In this case we consider that the initial position of the cursor is $O(0,0,0)$ and the initial colour is null. Then the word

$$(S_x^{|x_0|} s_y^{|y_0|} S_z^{|z_0|} cw)$$

draw the object $g(w)$ from the initial position (x_0,y_0,z_0) in the colour "c".

R E F E R E N C E S

1. F. J. Brandenburg, J. Dassow, *Reduction of Picture Words*, MIP 8905, Universität Passau, 1989.
2. H. A. Maurer, G. Rozenberg, E. Welzl, *Using string languages to describe picture languages*, Inf. and Control 54, 1982.
3. A. Rosenfeld - Picture Languages - *Formal Models for Picture Recognition*, Academic Press, 1979.
4. F. J. Brandenburg, M. P. Chytil, *On picture languages: Cycles and syntax-directed transformations*, MIP 9020 Universität Passau, 1990.

TERM REWRITING SYSTEMS AND COMPLETION THEOREMS PROVING: A SHORT SURVEY

Doina TĂTAR[*]

Received: February 15, 1993
AMS Subject Classification: 68T15

REZUMAT. - Sisteme de rescriere a termenilor şi demonstrarea teoremelor prin algoritmi de completare. În acest articol sunt prezentate principalele rezultate privind problema cuvântului pentru o teorie ecuaţională, tratată ca sistem de rescriere a termenilor (TRS), inclusiv o versiune sintetică a algoritmului de completare Knuth-Bendix.

Abstract. In this paper we survey the main results concerning equations and the methods for reasoning about them like term rewriting systems (TRS). This TRS are used to reduce expressions to canonical form, if this form is unique. A simplified version of Knuth-Bendix completion algorithm is presented.

Like most surveys, ours does not contain any new results, but it gives an idea on the application of TRS to theorems proving. This paper is justified by the interest of this subject and it presents the most important things in the completion idea. Since the pioneering paper (Knuth and Bendix, 1970), which indroduced the algorithm Knuth-Bendix, and the influential papers (Huet and Oppen, 1980), there has been a great deal of research in this field. For an excellent survey see (Avenhaus, Madlener, 1990).

This paper is organized as follows: Chapter 1 presents

[*] *Univ. "Babeş-Bolyai" of Cluj-Napoca, Dept. of Computer Science 3400 Cluj-Napoca, România*

equational systems and TRS, Chapter 2 the "critical-pair" idea and the "critical-pair" completion algorithm, and Chapter 3 some of the examples.

1. Introduction.

DEFINITION: An equational theory (F, V, E) consists of

- . a set F of function symbols or operators (with the same sort, for simplicity).
- . a set V of variables. Let $T(F,V)$ be the set of terms built from F and V.
- . a set E of pair of equations, $s=t$, $s,t \in T(F,V)$.

The set of equations E defines a syntactical equality relation $=_E$ on $T(F,V)$, usualy defined by the concept of "replacing equals by equals". One has also a semantical (logic) definition in equational theory E denoted by: $E \models s=t$.

The theorem of Birkhoff (1935) assures that both notions coincide: $t_1 =_E t_2 \leftrightarrow E \models t_1=t_2$.

A fundamental problem is the "validity problem" or "word problem", which is undecidable in general:

"Give $s,t \in T(F,V)$, does $s =_E t$?"

Obviously, the undecidability (more precisely, the semi-decidability) of the "word problem" is transfered on the approach by TRS. But this approach is, on the our opinion, more algorithmicaly.

DEFINITION: A TRS R is a set of rules:

$R = \{l \to r \mid l, r \in T(F,V)$ every variable occuring in term r also occurs in term $l\}$.

It defines a rewrite relation $\xrightarrow[R]{}$:

DEFINITION: $s \xrightarrow[R]{} t$ iff there is a rule $l \to r \in R$, an occurence p in s, such that:

$$s/p=\sigma(l), \quad t=s[p\leftarrow\sigma(r)].$$

for some substitution σ .

The relation $\xrightarrow[R]{}$ is compatible with the term structure in $T(F,V)$ (i.e. $s\to t$ implies $t_1[p\leftarrow s] \xrightarrow[R]{} t_1[p\leftarrow t]$) and with the substitutions (i.e. $s\to t$ implies $\sigma(s)\xrightarrow[R]{}\sigma(t)$ for each σ).

We denote by $\overset{*}{\underset{R}{\to}}$, $\overset{*}{\underset{R}{\leftrightarrow}}$ the reflexive - transitive and reflexive - transitive - symmetric closure of $\xrightarrow[R]{}$.

DEFINITION: The transfering of "word problem" to a TRS is:

"Given an equational theory E, compute an R such that $s \underset{E}{=} t$ is equivalent to $s \overset{*}{\underset{R}{\leftrightarrow}} t$ ".

The problem of compute R is a "completion procedure because R is constructed step by step by collecting new rules in R, which is in the same time simplified as much as possible.

Let $t\downarrow R$ (or $t\downarrow$) normal form of t, that is such term with the properties:

1) $t \overset{*}{\underset{R}{\to}} t\downarrow$

2) $t\downarrow$ is irreducible.

If R has the properties that every term has a unique normal form, then:

$$s \overset{*}{\underset{R}{\leftrightarrow}} t \quad iff \quad s\downarrow = t\downarrow .$$

because $s \overset{*}{\underset{R}{\leftrightarrow}} t$ is $s \overset{*}{\underset{R}{\to}} s\downarrow = t\downarrow \overset{*}{\underset{R}{\leftarrow}} t$)

Fact: If in R every term t has a unique normal form then

$$s \underset{E}{=} t \text{ if } s\downarrow \equiv t\downarrow$$

DEFINITION: A term rewriting system R with the property that every term has a unique normal form is *convergent* and it has the properties:

a) R is terminating (or Noetherian) that is it allows no infinite sequences:

$$t_1 \underset{R}{\rightarrow} t_2 \underset{R}{\rightarrow} t_3 \rightarrow \dots$$

(such that every term t has at least a normal form $t\downarrow$)

b) R is confluent, that is: $t_1 \overset{*}{\underset{R}{\rightarrow}} t_2$, $t_1 \overset{*}{\underset{R}{\rightarrow}} t_3$ implies that does exist u such that $t_2 \overset{*}{\underset{R}{\rightarrow}} u$, $t_3 \overset{*}{\underset{R}{\rightarrow}} u$ (every term t has at most a normal form $t\downarrow$).

The properties 1 and 2 are, unfortunately, undecidable (Dershowitz, 1987). However, there are useful tools to prove termination, the most important ones are **reduction orderings** on $T(F,V)$. An ordering $>$ on $T(F,V)$ is a reduction ordering if $>$ is well founded, is compatible with the term structure and is compatible with substitutions.

THEOREM: *A TRS R is terminating iff there is a reduction ordering > such that $\ell > r$ for every rule $\ell \rightarrow r$ in R.*

Due to a result of Newman (Newman, 1972) the confluence is, for terminating TRS, equivalent to the weaker property of local confluence, which is: $t_1 \underset{R}{\rightarrow} t_2$ *and* $t_1 \underset{R}{\rightarrow} t_2$, implies that does exist u such that $t_2 \overset{*}{\underset{R}{\rightarrow}} u$, $t_3 \overset{*}{\underset{R}{\rightarrow}} u$. Hence, a terminating R is confluent iff it is locally confluent.

2. Critical pair completion.

DEFINITION: Let $\ell_1 \to r_1$ and $\ell_2 \to r_2$ be two rules in R. By renaming of variables we may assume that they do not share common variables. If σ_1 and σ_2 are two substitutions, such that:

$$\sigma_1(\ell_1) = \sigma_2\ell_2)$$

then $(\sigma_1(r_1),\ \sigma_2(r_2))$ is a critical pair for R. Let CP(R) be the set of all critical pairs for R as equations. "Critical Pair Lemma" (Knuth and Bendix, 1970) say that:

For any TRS R, if $t_1 \underset{R}{\to} t_2$ *and* $t_1 \underset{R}{\to} t_3$ then either does exist a term u such that $t_2 \overset{*}{\underset{R}{\to}} u,\ t_3 \overset{*}{\underset{R}{\to}} u$ (if R is locally confluent) or $t_1 \underset{CP(R)}{=} t_2$.

Clearly, if for every $(\alpha_1,\ \alpha_2) \in$CP($R$) we have $\alpha_1 \overset{*}{\underset{R}{\to}} \alpha_2$ *or* $\alpha_2 \overset{*}{\underset{R}{\to}} \alpha_1$, then R is locally confluent. This think can be tested. The completion procedure do this. The simplest form of a completion procedure is:

INPUT: A set E of equations, an reduction ordering >.

OUTPUT: a) A TRS R_E convergent, such that

$$\underset{E}{=} = \overset{*}{\underset{R_E}{\leftrightarrow}}$$

b) FAILURE.

c) The algorithm run forever.

The Completion Algorithm:

```
R = ø.
If every equations in E can be oriented
    then
        R:= {ℓ→r| ℓ > r., ℓ = r ∈ E}
    else
```

```
                "FAILURE". STOP.
  while            $CP(R) \neq \emptyset$ do
            $(t_1,t_2)$:= an element in $CP(R)$
            It calculates $t_1\downarrow$ and $t_2\downarrow$.
            If $t_1\downarrow \neq t_2\downarrow$
                then
                If neither $t_1\downarrow > t_2\downarrow$ nor $t_2\downarrow > t_1\downarrow$
                            then
                            "FAILURE" STOP
                            else
                            $CP(R)=CP(R)\setminus\{(t_1,t_2)\}$.
                            $R = R \cup \{t_1\downarrow \to t_2\downarrow\}$.
                                    or
                            $R = R \cup \{t_2\downarrow \to t_1\downarrow\}$
                else
                    $CP(R)=CP(R)\setminus\{t_1, t_2)\}$.
      STOP.
```

Some observations are immediatelly:

a) If $CP(R)$ is transformed in $\emptyset$, then the procedure stop succesfully.

b) The procedure may stop by "FAILURE" if R is not terminating.

c) The procedure may run forever, if $CP(R)$ can be not transformed in $\emptyset$.

($CP(R)$ grows and degrows in a step).

3. Theorem proving examples. 1) Let E be given by $E=\{e*x = x,\ i(x)*x=e,\ x*(y*z) = (x*y)*z\}$, hence E are the axioms for a group.

If the ordering is $i > * > e$, then R is, at beginning:

$$R\begin{cases} r_1:\ e*x \to x \\ r_2:\ i(x)*x \to e \\ r_3:\ (x*y)*z \to n*(y*z) \end{cases}$$

From r_2 and r_3 we obtain:

$$r_4: i(x)*(x*y) \to y.$$

because:

$$(i(x)*x)*y \underset{r_2}{\to} e*y \underset{r_1}{\to} y$$

$$(i(x)*x)*y \underset{r_3}{\to} i(x)*(x*y)$$

Hence, $(i(x)*(x*y),y)$ is a critical pair and r_4 is the new correspondent rule. At the end, we obtain the following TRS convergent:
$R\ \{r_1,\ldots r_{10}\}$, where r_1, r_2, r_3, r_4, are the previously, and:

r_5: $i(e) \to e$

r_6: $x*e \to x$

r_7: $i(i(x)) \to x$

r_8: $x*i(x) \to e$

r_9: $x*(i(x)*y) \to y$

r_{10}: $i(x*y) \to i(y) * i(x)$.

Thus, for example we have:

$i(i(x*y)*e) * i(y*y)=i(y*i(x))$

$i(i(x*y)*e) * i(y*y) \underset{r_{10}}{\to} (i(e) * i(i(x*y)) * i(y*y) \underset{r_5, r_7}{\to}$
$(e*(x*y)) * i(y*y) \underset{r_1}{\to} (x*y) * i(y*y) \underset{r_{10}}{\to} ((x*y) * i(y)) * i(y)$
$(x*y) * (i(y) * i(y)) \underset{r_3}{\to} x * (y*(i(y) * i(y)) \underset{r_9}{\to} x * i(y)$

For second member:

$$i(y*i(x)) \xrightarrow[r_{10}]{*} i(i(x)) * i(y) \xrightarrow[r_7]{*} x*i(y)$$

Thus theorem $i(i(x*y)*e)*i(y*y)=i(y*i(x))$ is proved, as each member have the same normal form $x * i(y)$.

In present, several very strong prover have been developed, such as REVE (Lescanne, 1984), RRL (Kapur et al. 1986) and the Large prover (Garland and Guttag 1989). A catalogue of theorems proved is given in (Hermann 1991). Some experiments with a completion theorem prover was made in (U. Martin and M. Lai, 1989). Another way for using TRS in theorem proving is that suggested by Hsiang (Hsiang 1985, Tătar 1992): The TRS denoted BA for Boolean Algebra. This is a rewrite-based method for first-order predicate calculus.

At the end of this short survey let list the best paper concerned TRS and Theorem proving.

R E F E R E N C E S

1. Avenhaus, J., Madlener, K. : *Term rewriting and Equational Reasoning*, Formal Techniques in A.I., A Sourcebook, R.B.Banerji (ed) 1990.
2. Avenhaus, J. : *On the Descriptive Power of TRS*, J. of Symbolic Computation 2, 1986, pp. 109-122.
3. Avenhaus, J., Denzinger, J., Muller, J. : *THEOPOGLES anefficient Th. Prov. based on Rewr. Technique*, Tech. Rap.,Univ. of Kaiserslautern, 1990.
4. Bachmair, L., Dershowitz, N., Hsiang, J. : *Orderings for equational proofs*, 1st LICS (1986), pp. 346-357.
5. Bellegarde, F., Lescanne, P. : *Transformation orderings*, 12th CAAP, 1987, LNCS 249, pp. 69-80.
6. Ben Cherifa, A., Lescanne, P. : *Termination of TRS by Polynomial Interpretations and its Implementation*, Science of Comp. Programming, 9, 1987.
7. Buchberger, B. : *History and Basic Features of the Critical-Pair/Completion Procedure*, J. Symbolic Comp.3, 1987, pp.3-38.
8. Buchberger, B., Loos, R. : *Algebraic Simplification*, in Computer Algebra-Symb. and Alg. Comp., 1982, pp.11-43.
9. Buchberger, B. : *A critical-pair/completion algorithm in reduction rings*, Technical Rap, CAMP-LINZ, 1983.
10. Bundgen, R. : *Buchberger's Algorithm: The Term Rewriter's point of view*, ICALP'92, to appear.
11. Dershowitz, N. : *Completion and its Applications*, Coll. on the

"Resolution of Eq. in Alg. Structures", vol.2, 1989.

12. Dershowitz, N. : *Termination*, J. Symbolic Comp, 3(1987), pp.69-116.
13. Dershowitz, N., Hsiang, N. : *Refutational Theorem Proving with Oriented Equations*, Coll. on the Res. of Eq. in Alg Struct., 1987.
14. Goguen, J. : *Proving and rewriting,* 2nd International Conf. on Alg. and Logic Programming, oct. 1990.
15. Hermann, M., Kirchner, C., Kirchner, H. : *Implementations of TRS*, Computer Journal 34, 1991, pp. 20-33.
16. Hsiang, J. : *Two results in term rewriting theorem proving*, LNCS 202, pp. 301-324.
17. Hsiang, J. : *Rewrite method for Theorem proving in First Order Theory with Equality*, J. Symb. Comp. 1-2, vol. 3, 1987.
18. Hsiang, J., Rusinowitch, M. : *On word problems in equational theories*, LNCS 267, pp. 54-71.
19. Huet, G., Oppen, D.C. : *Equationa and Rewrite Rules: A Survey,* in "Formal languages: theory, perspective and open problems" (ed. R. Book), 1980.
20. Jantzen. M. : *Confluent string rewriting,* EATCS monographs 11, 1988.
21. Jouannaud, J.P., Lescanne, P. : *Rewriting Systems*, Techhnology and Science of Informatics, 1987, pp. 181-199.
22. Kaplan, S. : *Conditional Rewrite Rules*, Th. Comp. Sci., 33, 1984, pp. 175-193.
23. Knuth, D.E., Bendix, P.P. : *Simple Word Problqms in Universal Algebras*, Comp. Prob. in Abstract Algebra, (ed. J. Leech), 1970.
24. Lescanne, P. : *Term Rewriting Systems and Algebra*, LNCS, 170, 1984.
25. Lescanne, P. : *Computer Experiments with the REVE TRS Generator*, Proc. 10 ACM POPL, Austin, 1983, pp: 99-108.
26. Martin, U., Lai, M. : *Some experiments with a completion theorem prover*, J. Symb. Comp., 13, 1992, pp. 81-100.
27. Muller, J. : *Topics in Completion Theorem Proving*, Technical Rap., Univ. Kaiserslautern, 1990.
28. Newman, M.H.A. : *On theories with a combinatorial Definition of Equivalence*, Ann. of Math., *43*, 1942, pp. 223-243.
29. Rusinowitch, M. : *Path of Subterms Ordering and Recursive Decomposition Ordering Revisited*, J. Symb. Comp., 1987, vol.3, pp. 117-133.
30. Rusinowitch, M. : *Demonstration automatique. Techniques de réécriture*, Inter. Edition, Paris, 1989.
31. Tătar, D. : *A new method for the proof of theorems*, Studia Univ. "Babeş-Bolyai", 1991, pp.
32. Tătar, D.: *Critical pair as a deduction rule,* Proceedings of 3th. Coll. on Logic Lang., Math. Linguisitc, Braşov, mai, 1991.
33. Tătar, D.:*Normalized TRS and applications in the theory of programs*, Analele Univ. Buc., nr.*2*, 1989, pp. 76-80.
34. Winkler, F., Buchberger,B.: *A criterion for eliminating unnecessary reductions in the K-B algorithm* , 983, Coll. Alg. Combin. and Logic, Györ.

STUDIA UNIV. BABEŞ-BOLYAI, MATHEMATICA, XXXVII, 3, 1992

ON THE EQUIDISTANT DIVISION OF TWO-DIMENSIONAL SMOOTH CURVE

Iuliu VLAIC[*]

Received: April, 11, 1993
AMS Subject Classification: 65D30, 68U05

REZUMAT.- Asupra divizării echidistante a unei curbe plane netede. Această notă prezintă o metodă de divizare echidistantă a unei curbe plane netede definită prin ecuaţia sa explicită, parametrică, polară sau printr-un tabel de puncte discrete. În finat este dată o aplicaţie de natură tehnică. Programarea formulei lui SIMPSON, aferentă unor integrări numerice, a fost făcută în limbajul HPL pentru un calculator de tipul HEWLETT-PACKARD 9826.

0. Introduction. Setting of Problem. For a two-dimensional smooth curve given by explicit, parametric, polar equations or by n data points, i.e.:

$$\left|\ y = f(x), \qquad x \in [x_0, x_n],\right. \tag{1}$$

$$\left|\ \begin{aligned} &x = x(t),\\ &y = y(t), \qquad t \in [t_0, t_n],\end{aligned}\right. \tag{2}$$

$$\left|\ r = r(\varphi), \qquad \varphi \in [\varphi_0, \varphi_n],\right. \tag{3}$$

and respective

$$\left|\ \begin{aligned} &(x_0,y_0),\ (x_1,y_1),\ldots,(x_n,y_n),\\ &x_0 < x_1 <\ldots< x_n, \qquad n \in N,\end{aligned}\right. \tag{4}$$

on search points coordinates (α_i, β_i), $i = \overline{0,m}$ which divide this curve in m equal arcs, or the distances between (α_i, β_i) and $(\alpha_{i+1}, \beta_{i+1})$, $i = \overline{0,m-1}$ to be equal.

There are many technical applications of such problems. For example, in the last part of this paper is presented equidistant division of basis ellipse from gearings with elliptical gear wheels.

1. BRIEF THEORETICAL SPECIFICATIONS. SOLUTION OF PROBELM. Let a plane smooth curve describe by the equations (1), (2) or

[*] *Mechanical Enterprise of Cugir, 2566 Cugir, Romania*

(3) be given. For a value of x (t or φ) may be obtained corresponding value of the function. If the curve is given by n data points (see (4)), first it is necessary an interpolation proceeding and for a value x the ordinate y is obtained as the value of the interpolating polinom at x.

Let ℓ be the length of the curve. We have:

$$\ell = \int_{x_0}^{x_n} \sqrt{1 + [f'(x)]^2} \cdot dx = \int_{T_0}^{T_n} \sqrt{[x'(t)]^2 + [y'(t)]^2} \cdot dt = \\ = \int_{\varphi_0}^{\varphi_n} \sqrt{r^2 + [dr/d\varphi]^2} \cdot d\varphi, \tag{5}$$

and respective:

$$\ell = \sum_{k=0}^{n-1} \int_{x_k}^{x_{k+1}} \sqrt{1 + [f'_{s,k}(x)]^2} \cdot dx, \tag{6}$$

where $f_{s,k}$ are elementary functions of cubic spline interpolation for n data points (4), [1], [2], [3], [4]. If it is necessary m division arcs, then length of elementary arc (10) is (l/m). Now we determine coordinates of the division points (α_i, β_i), $i = \overline{0,m}$ The first point of division (α_0, β_0) coincides with (x_0, y_0) and the last one (α_m, β_m) coincides with (x_n, y_n). The estimation of definite integrals and solution of transcendent equation are maked with numerical methods.

Thus, for integration we have 1/3 SIMPSON rule, [1], [3]:

$$\int_b^a f(x) \cdot dx - (h/3) \cdot \{f(a) + 4 \cdot f(a + h) + 2 \cdot f(a + 2h) + \\ + 4 \cdot f(a + 3h) + \ldots + 4 \cdot f[a + (p - 1)h] + f(a + ph)\}, \tag{7}$$

where $h = (b - a)/p$.

This formula, applyed on each interval $[\alpha_i, \alpha_{i+1}]$, $i = \overline{0, m-1}$, with

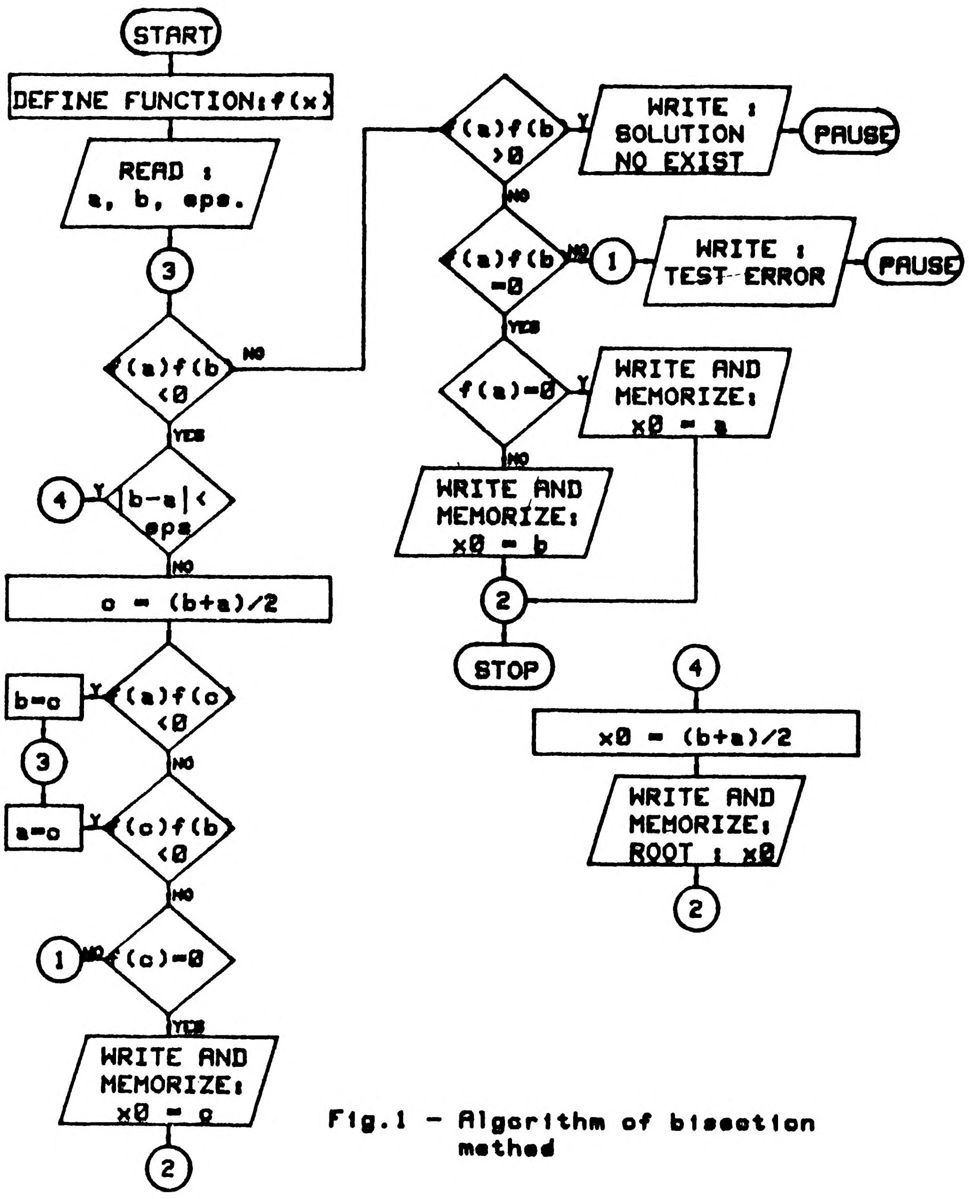


Fig.1 – Algorithm of bisection method

$$\int_{\alpha_i}^{\alpha_{i+1}} f(x)\cdot dx = 10, \qquad i = \overline{0, m-1}$$

and α_{i+1} unknown (initial value is $\alpha_0 = x_0$ and maximum value is $\alpha_m = x_n$), becomes:

$$(h/3)\cdot\{f(\alpha_i) + 4\cdot f(\alpha_i + h) + 2\cdot f(\alpha_i + 2h) + 4\cdot f(\alpha_i + 3h) + \ldots + \\ + 4\cdot f[\alpha_i + (p-1)h] + f(\alpha_i + ph)\} - 10 = 0, \qquad (8)$$

where $h = (\alpha_{i+1} - \alpha_i)/p$, $\alpha_0 = x_0$, $\alpha_m = x_n$, $i = \overline{0, m-1}$ and α_{i+1} is unknown value.

We remark that, in general, the function f has one of the following forms:

$$\sqrt{1 + [f'(x)]^2} \to f(x), \qquad \sqrt{[x'(t)]^2 + [y'(t)]^2} \to f(t),$$

$$\sqrt{r^2 + (dr/d\varphi)^2} \to f(\varphi), \qquad \sqrt{1 + [f'_{s,k}(x)]^2} \to f(x) \text{ and}$$

$$[t_0, t_n] \to [x_0, x_n], \ [\varphi_0, \varphi_n] \to [x_0, x_n] \text{ or } [x_k, x_{k+1}] \to [x_0, x_n]$$

For equation (8) we present a numerical solution by using bisection method where $\alpha_i < \alpha_{i+1} < \alpha_m = x_n$. In Fig. 1 is given the algorithm of this method.

The number of points, p, for SIMPSON rule depends on the ε-precision supplyed by the user. If for the estimation of the curve length ℓ, we impose the precision $\varepsilon/2$ and for each interval $[\alpha_i, \alpha_{i+1}]$, the precision $\varepsilon/(2m)$ both from the estimation of number p and for the solution of the equation (8), the total error will be:

$$\varepsilon/2 + m\cdot\varepsilon/(2\cdot m) = \varepsilon \qquad (9)$$

which may be considered as error for entire proceeding (7)-(8).

```
CO-ORDINATES OF DIVISION POINTS :

X( 0.000 )= 97.594    Y( 0.000 )= 0.000
X( 1.000 )= 96.368    Y( 1.000 )= 12.641
X( 2.000 )= 92.777    Y( 2.000 )= 24.822
X( 3.000 )= 87.055    Y( 3.000 )= 36.160
X( 4.000 )= 79.517    Y( 4.000 )= 46.302
X( 5.000 )= 70.497    Y( 5.000 )= 55.322
X( 6.000 )= 60.305    Y( 6.000 )= 62.899
X( 7.000 )= 49.213    Y( 7.000 )= 69.004
X( 8.000 )= 37.451    Y( 8.000 )= 73.875
X( 9.000 )= 25.217    Y( 9.000 )= 77.203
X( 10.000 )= 12.682    Y( 10.000 )= 79.322
X( 11.000 )=-.000    Y( 11.000 )= 80.000
X( 12.000 )=-12.682    Y( 12.000 )= 79.322
X( 13.000 )=-25.217    Y( 13.000 )= 77.203
X( 14.000 )=-37.451    Y( 14.000 )= 73.875
X( 15.000 )=-49.213    Y( 15.000 )= 69.004
X( 16.000 )=-60.305    Y( 16.000 )= 62.899
X( 17.000 )=-70.497    Y( 17.000 )= 55.322
X( 18.000 )=-79.517    Y( 18.000 )= 46.302
X( 19.000 )=-87.055    Y( 19.000 )= 36.160
X( 20.000 )=-92.777    Y( 20.000 )= 24.822
X( 21.000 )=-96.368    Y( 21.000 )= 12.641
X( 22.000 )=-97.594    Y( 22.000 )= 0.000
```

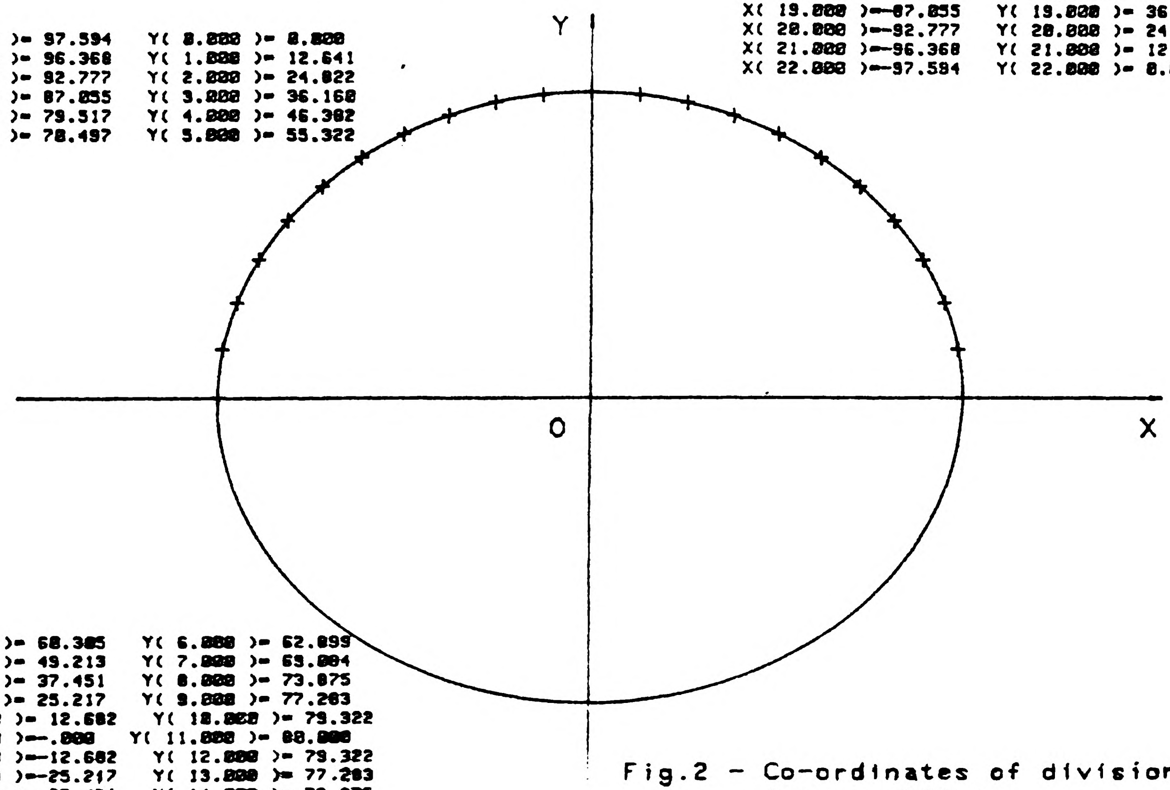


Fig.2 - Co-ordinates of division points for an ellipse

A subroutine for SIMPSON rule is presented by HPL, [5], program given in Table 1.

In this method the stop criterion is to obtain a maximum number of divisions for interval [a,b] or the difference between two successive values is less than ε-tolerance supplyed by the user.

All parameters supplyed as input data for subroutine must be initialized in driver programme:

The user may by joint subroutine with any your programme and solution for integral value is stored in V variable.

Parameters and used variables:

a) Input data:

P - maximum numbers of iterations;

A - lower bounds of definite integral;

B - upper bounds of definite integral;

$f(x) \rightarrow Y$ - which may be given by the user;

E - error tolerance (difference between two successive calculated values of integral.

b) Output data:

P, A, B, $f(x)$, E (see a));

V - approximative value for definite integral: $\int_a^b f(x) \cdot dx$.

c) Working variables:

D - domain of abscissae;

J, $(J < P)$ - index of iterations;

Q - precedent value of integral (used for to stop criterion);

N - current number of intervals (*N* = 2^*J*);

C - size of interval [*C* = (*B* - *a*)/*N*];

X - function argument;

Y - function value {*y* = *f*(*x*)];

R - the first and the last value of the integral.

Table 1 - HPL Programme of SIMPSON rule

```
0: "SIMPSON":0->i;0->V;sfg3;if A>B;prt"error limits";ret
1: A->X;gsb"Function";V+Y->X;gsb"Function";V+Y->V;V+Y->R
2: if (J+1->J)>P;Q->V;ret
3: 2^J->N;((B-A)/N->C)+A->X;gsb"Function";V+4Y->V;A->D
4: if (D+2C->D)<B;gto 8
5: CV/3->V;if flg3;cfg3;gto 7
6: if abs(Q-V)<E;ret
7: V->Q;R->V;gto 2
8:  D->X;gsb"Function";V+2Y->V;D+C->X;gsb"Function";4+4Y->V;
   gto4
9: "Function"
10: 72.5*V(1-X*X/(1041312))->Y;ret
```

2. EXAMPLE. The described proceeding has been applicated for equidistant division of basis ellipse from gearings with elliptical gear wheels in paper industry (gearings axis has been situated in focus of ellipse). Such an example of division is given in Fig. 2, where it is shown coordinates of division points.

R E F E R E N C E S

1. Bakvalov, N., *Méthodes numériques,* Edition Mir, Moscou, 1976.
2. Beckett, R. and Hurt, J., *Numerical Calculations and Algorithms,* New-York, Mc Graw-Hill, 1967, pp. 166-169.
3. Salvadori, M.G., Baron, M. L., *Metode numerice în tehnică,*Editura Tehnică, Bucureşti, 1972.
4. Toma, M., Odăgescu, I., *Metode numerice şi subrutine,*Editura Tehnică, Bucureşti, 1980.
5. Hewlett-Packard-HPL, *Operating Manual for the HP9000 series 200. Model 216/226/236 Computers,* Manual Part. No. 98614-90010, Hewlett-Packard Company, 3404 East Harmony Road, Fort Collins, Colorado 80525, USA.

În cel de al XXXVII-lea an (1992) *Studia Universitatis Babeş-Bolyai* apare în următoarele serii:

matematică (trimestrial)
fizică (semestrial)
chimie (semestrial)
geologie (semestrial)
geografie (semestrial)
biologie (semestrial)
filosofie (semestrial)
sociologie-politologie (semestrial)
psihologie-pedagogie (semestrial)
ştiinţe economice (semestrial)
ştiinţe juridice (semestrial)
istorie (semestrial)
filologie (trimestrial)
teologie ortodoxă (semestrial)

In the XXXVII-th year of its publication (1992) *Studia Universitatis Babeş-Bolyai* is issued in the following series:

mathematics (quarterly)
physics (semesterily)
chemistry (semesterily)
geology (semesterily)
geography (semesterily)
biology (semesterily)
philosophy (semesterily)
sociology-politology (semesterily)
psychology-pedagogy (semesterily)
economic sciences (semesterily)
juridical sciences (semesterily)
history (semesterily)
philology (quarterly)
orthodox theologie (semesterily)

Dans sa XXXVII-e année (1992) *Studia Universitatis Babeş-Bolyai* paraît dans les séries suivantes:

mathématiques (trimestriellement)
physique (semestriellement)
chimie (semestriellement)
geologie (semestriellement)
géographie (semestriellement)
biologie (semestriellement)
philosophie (semestriellement)
sociologie-politologie (semestriellement)
psychologie-pédagogie (semestriellement)
sciences, économiques (semestriellement)
sciences juridiques (semestriellement)
histoire (semestriellement)
philologie (trimestriellement)
théologie orthodoxe (semestriellement)

43 875

Lei 200