STUDIA

UNIVERSITATIS BABEŞ-BOLYAI

MATHEMATICA

3

1993

CLUJ-NAPOCA

Anul XXXVIII 1993

STUDIA
UNIVERSITATIS BABEŞ-BOLYAI

MATHEMATICA

3

Redacţia : 3400 CLUJ-NAPOCA str. M. Kogălniceanu nr.1 ► Telefon . 116101

SUMAR-CONTENTS-SOMMAIRE

Aniversări - Anniversaries - Anniversaires

STUDIA UNIV BABEŞ-BOLYAI, MATHEMATICA, XXXVIII, 3, 1993

ON SOME MOMENTS OF COMPUTER SCIENCE EVOLUTION IN ROMANIA

by

Emil Muntean

In the fifties, a group of researchers from the Institute for Atomics Physics, Bucharest built up the first romanian electronic computer machine, due to an initiative of Acad Gr. Moisil This computer, named "Computer of the Institute for Atomics Physics" (CIFA-1), was designed and implemented under the co-ordination of Eng Victor Toma, in 1954 On that occasion, at the same institute in Bucharest, a new research group aimed to work in the field of computer software programming, is formed

After a short time, in 1957, at Cluj, is founded the first Romanian Institute, having Acad Tiberiu Popoviciu as supervisor Founded on the 1st of April, 1957 and called the Computer Institute of the Romanian Academy, his activity was based on that of the Numeric Analysis Department of the Cluj branch of the Romanian Academy This institute has been oriented to fields much more related to those considered today as part of Computer Science

This institute, founded by Acad T Popoviciu in Cluj, represented at that time an exceptional organizatoric achievement There were very few such institutes in the whole world, and in Eastern Europe the research in Cybernetics and Computer Science was neither encouraged nor recognised In Romania, since the foundation of the Institute in Cluj, ten years were necessary for the totalitary goverment to oficially promote the interests in the field of Computer Science and to found, in 1968, in Bucharest, the Research Institute for Electronic Computers (known later as the Computer Technique Institute, I T C)

The first romanian transistorized computer, DACICC-1 (Automatic Computing Device of the Computer Technique Institute, Cluj), was built at the Computer Institute from Cluj, in 1961 The research groups from the Department for Computer Machines of the same Institute, start the design of some complex applications, both tehnical and economical As a consequence, different industrial companies in Cluj introduced computer technique the shoe factory Clujana, the Railway Company, the Company for freezing equipment The research is oriented towards optimization problems, linear programming, transport problems There were formed some research groups specialized on different fields hardware design, software design, tehnical and scientifical applications, economical applications These structures, founded between 1960 and 1965 at the Computer Technique Institute in Cluj, have tipical Computer Technique and Computer Science interests During the same period, due to the influence generated by the Computer Technique Institute, the Department for Computer Machines is founded at the Faculty of Mathematics This department will have prepared many generations of computer scientists Ten years will pass from the foundation of this department, until it will have a computer for the students activity and for the teachers research in Computer Science

Less than ten years after the foundation of the Computer Institute in Cluj, the design of a complex project at that time has begun After a lot of complicated efforts to find the nongoverment financial support, in 1967, the design of the DACCIC-2 computer started The DACCIC-2 design project had contained a lot of new elements, introduced at that time as

inovations by the big computer companies, especcialy by 113M through 360 serie

The DACCIC-2 computer had

- word length on 32 bytes,
- memory adress on octets,
- interrupts handling,
- some parallel treatment (statements preparing and execution),
- the speed of the central unit was 200,000 operations/sec
- a kernel of the operating system which achieves the peripherals management, the interrupts handling, the programs management in multiprogramming, compiler, assembler, library and loader for FORTRAN programming language,
- a tehnological approach for a serial production

The design this project on an industrial scale hasn't been achieved Under the pressure of the world development and the initiatives from the neighbourn countries, the political leading decided to buy a license, to organise a computer production and to concentrate all the research forces in a national institute of a ministerial rank (nonacademical), with branches in Cluj and Timişoara, where a lot of valuable research in Computer Science had been developed This had taken part between 1968 and 1970

After a few years, the results seemed good. a lot of equipment had been introduced in the centralized economy, applications were developed, especially for management, after the principles of the state economy

The licence copyright and the attempt to develope it improved the scientific research, solving some of the major problems in Computer Science

At Cluj, the I T C branch had concentrated the research in the domains as programming languages, databases in peripherals design, personal computers and so on Interesting implementations were designed for the Romanian computers arhitecture, developing the licence, for almost every standardized programming language: FORTRAN, FORTRAN-77, COBOL, PASCAL, C, ADA and CHILL Prototypes were obtained for peripherals, which, later, had known a large serial production displays, plotters, digitizers and personal computers During this period, new research groups were formed, which worked, from a organizatorical point of view, on the same principles as the teams from the computers companies

The concept of "Regional Computer Centre " apeared in Romania, in the seventies, as the principal user of the computers This regional Center co-ordinated the computer science activity in a region, and all of these centres were co-ordinated by the Central Institute for Computer Science (I C I), which, for many years, directed even the necessary of computer equipments of all the companies and enterprises in Romania

After 13 or 15 years, one may clearly realised that Romania could not face the development rate in computers, that the tehnology obtained by licence had grew older very fast and that a new one hadn't appeared The research developed in the eighties, in the domain of computer arhitecture in the whole world and, especcialy, in high tehnology, had the effect that the Romanian products as minicomputers, personal computers and peripherals became unfeasible and uncompetitive

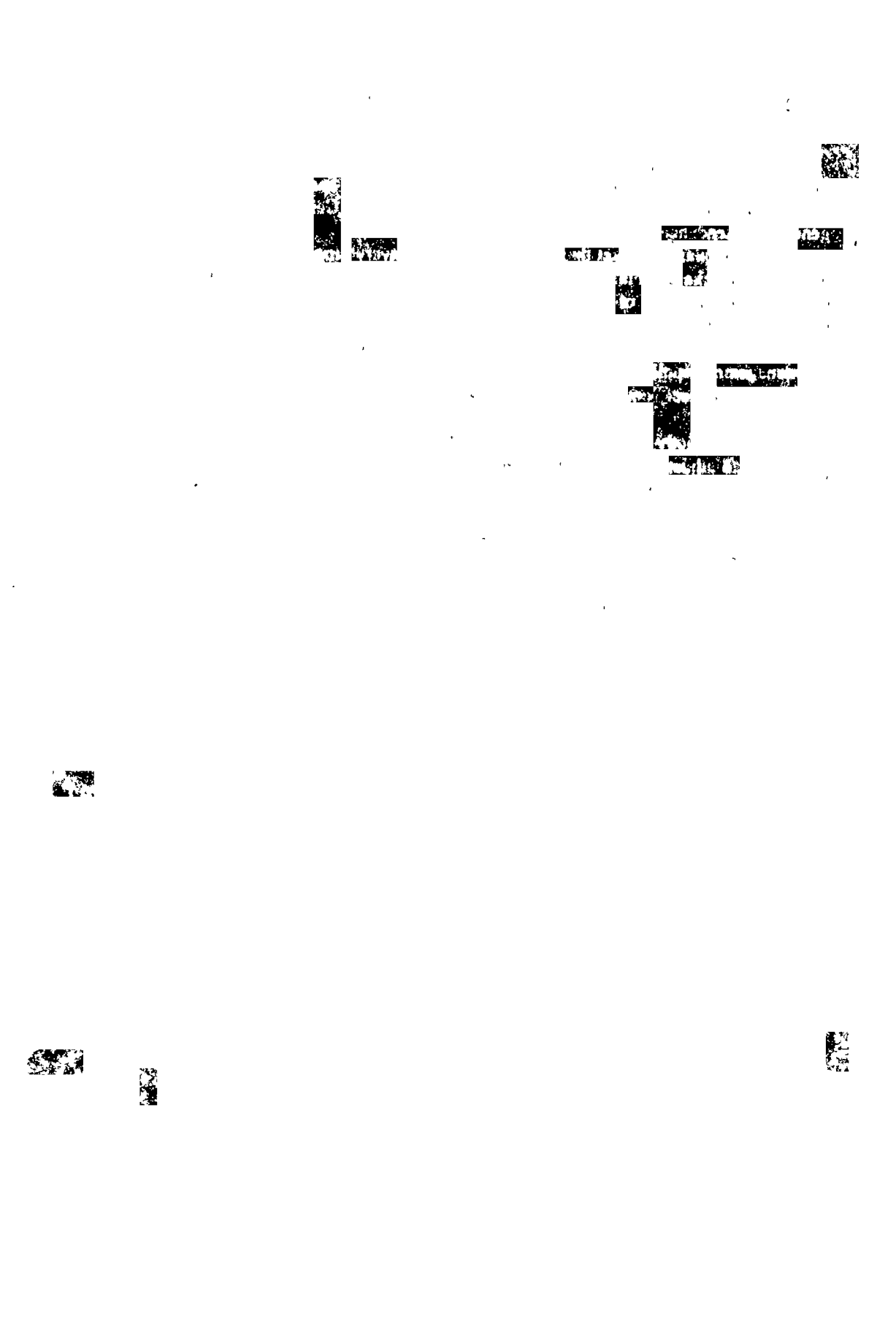
The world tendence in Computer Science was a descentralized one, was in a process

of "democracy". In Romania, the industrial companies could hardly develope their particular applications since the Regional Computer Centre and the Central Institute for Computer generated a tendence of hypercentralization.

That explains the fact that, after 1989, almost everything in Computer Science had to be taken from the beginning, especially concerning the equipment availability, applications design and the training of the operative personnel. Some good experience has been gained during the period of assimilation and development of the licences. But these was an old one. Also, a lot of people gained experience in using the medium computers and minicomputers for management applications, but even this one had the dezavantage of beeing related to a hypercentralized economy, based on laws completly different than those necessary for a market economy.

In a completly new situation, different from that before 1989, the Romanian computer scientists had adjust very quickly, understanding that Romania represents a large computer market. As a consequence, a lot of comercial companies, with state and private fundings, had invested in computer equipment, from private firms. A lot of computer companies had been founded, increasing the quality in software design and computer service.

We hope that in the forthcoming future will bring an explosive increase of computers users, comparative with that in computer equipment. Of course, this fact is seriously affected by the economical restructure and development



STUDIA UNIV BABEŞ-BOLYAI, MATHEMATICA, XXXVIII, 3, 1993

AN IMPLEMENTATION SCHEME FOR THE PARBEGIN-PAREND CONSTRUCTION

Florian Mircea BOIAN and Alexandru VANCEA*

Dedicated to Professor Emil Muntean on his 60th anniversary

Received February 25, 1994
AMS subject classification 68Q45, 68Q10

REZUMAT. - O schemă de implementare pentru construcţia PARBEGIN-PAREND. Lucrarea prezintă o schemă de translatare orientată spre sintaxă pentru construcţia PARBEGIN-PAREND, schemă pe baza căreia se poate construi uşor un translator care generează cod în limbajul C sub sistemul de operare UNIX.

The construction PARBEGIN $P_1 \mid P_2 \mid . \mid P_n$ PAREND [3] describes the simultaneous execution of the processes P_1, P_2, ,P_n and their parallel evolution until all of them terminate The n processes begin their execution at the same time and they function synchronously

This control structure contains a single entry (PARBEGIN) and a single exit (PAREND) and it is a *static* control structure, this meaning that all processing decisions are taken at compile time

The *fork-join* instructions are frequently used in UNIX, these being implemented by means of a *fork-wait* mechanism These instructions provide a direct mechanism for *dynamic process creation* and the possibility of multiple activations of the same process

The execution of a child process is made by calling the fork function which creates

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

the child process by duplicating the father's image Fork returns in the father process the child's PID and zero in the child.

The UNIX fork-wait mechanism [2] allows the synchronization of a father process with its sons The wait function blocks the calling process until one of its childs terminates If at the moment of the call one of its childs it's already terminated the returning is immediate The value returned by wait is an integer representing the terminated child's PID

p = wait (&status)

where status is an integer providing information about the process status.

The synchronization with a certain child (let's say with the one having PID=pid1) can be done in the following way

while (wait(&status) != pid1),

These functionalities suggest the possibility of expressing a PARBEGIN-PAREND construction by means of the fork-wait mechanism

Let's consider the independent processes P_1, ,P_n as the subjects of a PARBEGIN-PAREND instruction, with the syntax

PARBEGIN P_1 PAR PAR P_n PAREND

(we introduced the word PAR instead of |, because the latter may be confused with the C bitwise OR operation)

In these conditions the PARBEGIN entry point has its equivalent in the sequence

```
                    if (fork()) == 0)  { P1; exit(0), },
    else    if (fork()) == 0)  { P2, exit(0), };
    else
else    if (fork()) == 0)  { Pn, exit(0), },
    else    for (i=1, i<=n, i++) wait(&status),
```

Having these, we can express the PARBEGIN-PAREND construction through the following syntax-directed translation scheme [1]

(1) <PARBEGIN_constr> .= **PARBEGIN process** <tail>,
if (fork()==0) {process; exit(0);} <tail>

(2) <tail> = **PAR process** <tail>,
else if (fork()==0) {process; exit(0);} <tail>

(3) <tail> · = **PAREND**,
for (i=1; i<=n; i++) wait(&status);

where we put the nonterminals between brackets

The **process** terminal designates one of the P_1, P_2, ,P_n processes

One of the issues that arise relatively to this scheme is how to handle nested PARBEGIN-PAREND constructs The answer is simple once the deeper construct has been identified and translated, it becomes a **process**

Production (1) will generate process P_1 The rest of the processes are generated by production (2), which also increments the number of processes by one Production (3) uses the number of processes for generating the PAREND waiting point correctly It's easy to write a translator for this mechanism

Let's see a generation example with two processes

(<PARBEGIN_constr>,
<PARBEGIN_constr>) ==>

(**PARBEGIN process** <tail>,
if (fork()==0) {process; exit(0);} <tail>) ==>

(**PARBEGIN process PAR process** <tail>,
if (fork()==0) {process; exit(0);} else if (fork()==0)
{process; exit(0);} <tail>) ==>

```
( PARBEGIN process PAR process PAREND ,
  if (fork()==0) {process; exit(0);} else if (fork()==0)
  {process; exit(0);} else for (i=1; i<=n; i++) wait(&status);)
```

REFERENCES

1 Aho A V , Ullman J D - The Theory of Parsing, Translation and Compiling, Prentice Hall, 1973

2 Rochkind M J - Advanced Unix Programming, Prentice Hall, 1985

3 Tanenbaum A S - Modern Operating Systems, Prentice Hall, 1992

STUDIA UNIV BABEŞ-BOLYAI, MATHEMATICA, XXXVIII, 3, 1993

SERIAL AND PARALLEL ALGORITHMS FOR SOLVING A PROBLEM OF CONVECTION IN POROUS MEDIUM

Ioana CHIOREAN*

Dedicated to Professor Emil Muntean on his 60th anniversary

Received August 5, 1993
AMS subject classification 65Y05, 68Q22

REZUMAT. - Algoritmi seriali şi paraleli pentru rezolvarea unei probleme de convexitate în mediu poros. Scopul acestei lucrări este să se facă o comparaţie între algoritmii seriali şi paraleli, pentru a rezolva o problemă dată în mediu poros Sunt studiate în lucrare performanţele algoritmilor paraleli care au ca scop creşterea vitezei de calcul şi a eficienţei lor

Abstract. The main purpose of this paper is to make a comparison between a serial and a parallel algorithm for solving a given problem of convection in porous medium The performances of the parallel algorithm, established by means of speed-up and efficiency, are studied

NOMENCLATURE

g	gravitational acceleration
V	velocity of the fluid
p	pressure of fluid
T	temperature of fluid
K	permeability of the saturated porous medium
k	thermal conductivity of porous medium
S	rate of internal heat generation of porous medium
Ra	internal Rayleigh number
L	characteristic length of the porous medium
t	time

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

u,v	velocity components
x,y	coordinates

Greek symbols

ρ	density of fluid
μ	viscosity of fluid
$(\rho c)_f$	heat capacity of fluid
$(\rho c)_p$	heat capacity of porous medium
β	thermal expansion coefficient
ψ	dimensionless stream function
ϕ	angular coordinate

Superscripts

dimensional variables

Subscripts

0	value at reference temperature and density

1. Introduction. The problem under consideration is that of $2D$ steady laminar convection in a porous layer bounded by an inclined squre box with four rigid walls of constant temperature (fig 1) Heat is generated by a uniformly distributed energy sources within the cavity The porous layer is isotropic, homogeneous and saturated with an incompressibile fluid The heat generation creates a temperature gradient across the layer, and thereby provides a driving mechanism for natural convection within the cavity

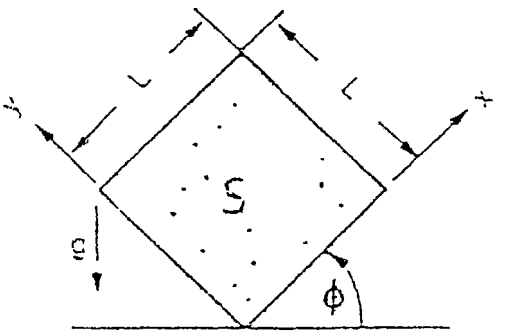


Fig 1 Schematic diagram of the enclosure

In the present study, the saturated porous medium is treated as a continuum, with the

solid and fluid phases in local thermodynamic equilibrium Also, the saturated fluid and the porous matrix are supposed incompressible and all physical properties of the medium, except the fluid density are taken to be constant

2. Converning equations. The fluid motion obeys the equations Darcy-Oberbeck-Boussineq For the case of volumetric heating considered here, the governing equations can be written as

$$\nabla \cdot V' = 0, \tag{1}$$

$$V' = \frac{K}{\mu}(\rho' g - \nabla p'), \tag{2}$$

$$(\rho c)_p \frac{\partial T'}{\partial t'} + (\rho c)_f (V' \cdot \nabla) T' = k \nabla^2 T' + S', \tag{3}$$

$$\rho' = \rho_0' [1 - \beta (T' - T_0')] \tag{4}$$

The four equations may be written

$$\frac{\partial u'}{\partial x'} + \frac{\partial v'}{\partial y'} = 0, \tag{1'}$$

$$u' = \frac{k}{\mu}\left(-\rho' g \sin\phi - \frac{\partial p'}{\partial x'}\right), \tag{2'}$$

$$v' = \frac{K}{\mu}\left(-\rho' g \cos\phi - \frac{\partial p'}{\partial y'}\right), \tag{2"}$$

$$(\rho c)_p \frac{\partial T'}{\partial t'} + (\rho c)_f \left(u' \frac{\partial T'}{\partial x'} + v' \frac{\partial T'}{\partial y'}\right) = k\left(\frac{\partial^2 T'}{\partial x'^2} + \frac{\partial^2 T'}{\partial y'^2}\right) + S', \tag{3'}$$

$$\rho' = \rho_0 [1 - \beta (T' - T_0')] \tag{4'}$$

Derivating (2') after y' and (2") after x' and taking into account that the temperature function has the form $T'(x', y')$, it is obtained

$$\frac{\partial u'}{\partial y'} = \frac{K}{\mu}\left(g \sin\phi \, \rho_0' \beta \frac{\partial T'}{\partial y'} - \frac{\partial^2 p'}{\partial x' \partial y'}\right), \tag{5}$$

$$\frac{\partial v'}{\partial x'} = \frac{K}{\mu}\left(g\cos\phi\,\rho_0'\beta\frac{\partial T'}{\partial x'} - \frac{\partial^2 p'}{\partial x'\,\partial y'}\right) \tag{6}$$

Subtracting (6) from (5) we get

$$\frac{\partial u'}{\partial y'} - \frac{\partial v'}{\partial x'} = \frac{K}{\mu}g\rho_0'\beta\left(\sin\phi\frac{\partial T'}{\partial y'} - \cos\phi\frac{\partial T'}{\partial x'}\right) \tag{7}$$

Using the dimensionless variables

$$t = \frac{kt'}{(\rho c)_\rho L^2}; u = \frac{(\rho c)_f L u'}{k}, v = \frac{(\rho c)_f L v'}{k}, x = \frac{x'}{L}, y = \frac{y'}{L}, T = \frac{k(T' - T_0')}{S'L^2}$$

(7) becomes

$$\frac{\partial u}{\partial y} - \frac{\partial v}{\partial x} = \frac{KL^3 S'\rho_0' g\beta(\rho c)_f}{\mu k^2}\left(\sin\phi\frac{\partial T}{\partial y} - \cos\phi\frac{\partial T}{\partial x}\right) \tag{7'}$$

Taking $Ra = \frac{KL^3 S' g\beta}{\alpha\nu k}$, where $\nu = \mu/\rho_0'$ and $\alpha = (\rho c)_f/k$ as the Rayleigh number, (7') becomes

$$\frac{\partial u}{\partial y} - \frac{\partial v}{\partial x} = Ra\left(\sin\phi\frac{\partial T}{\partial y} - \cos\phi\frac{\partial T}{\partial x}\right) \tag{7"}$$

Analogously, using the dimensionless variables, (1') and (3') become

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0, \tag{1"}$$

$$\frac{\partial T}{\partial t} + u\frac{\partial T}{\partial x} + v\frac{\partial T}{\partial y} = \nabla^2 T + 1 \tag{4"}$$

Equation (4") is verified by the streamfunction ψ where

$$u = \frac{\partial\psi}{\partial y}, \quad v = -\frac{\partial\psi}{\partial x} \tag{8}$$

So, introducing (8) in (4") and (7") we get the finally system of two equations with two unknowns (the temperature function T and the stream function ψ)

$$\begin{cases} \dfrac{\partial T}{\partial t} + \dfrac{\partial\psi}{\partial y}\dfrac{\partial T}{\partial x} - \dfrac{\partial\psi}{\partial x}\dfrac{\partial T}{\partial y} = \nabla^2 T + 1, \\ \nabla^2\psi = Ra\left(\sin\phi\dfrac{\partial T}{\partial y} - \cos\phi\dfrac{\partial T}{\partial x}\right) \end{cases} \tag{9}$$

We solve this system beeig situated in an enclosure with unit square section ($L = 1$), eith the

initial conditions

$$T_0 = \psi_0 = 0 \tag{10}$$

and the boundary conditions

$$T = \psi = 0 \quad \text{for } x = 0 \text{ and } 1,\ y = 0 \text{ and } 1 \tag{11}$$

Numerical results.

3.1. The Steady Problem In the steady case, our system of equations is

$$\begin{cases} \dfrac{\partial \psi}{\partial y}\dfrac{\partial T}{\partial x} - \dfrac{\partial \psi}{\partial x}\dfrac{\partial T}{\partial y} = \nabla^2 T + 1, \\ \nabla^2 \psi = Ra\left(\sin\phi \dfrac{\partial T}{\partial y} - \cos\phi \dfrac{\partial T}{\partial x}\right) \end{cases} \tag{12}$$

In order to obtain the solution for the system (12) with the conditions (10) and (11), we used the Multigrid method [4] with a Gauss-Seidel smoother The space derivatives were approximated in the following manner the first order derivatives with the Euler forward formula and the second order derivatives with the centered differences, accordind to [6] The discretized solution for the temperature and stream functions was obtained working on an equidistant grid Ω_l (where l indicates the level of grid), defined in the following manner

$$\Omega_l = \left\{ (ih_l, jh_l) \mid 0 \le i, j \le N_l, h_l = 1/N_l, N_l = 2^{l+1} \right\}$$

Denoting $T_{i,j} = T(ih_l, jh_l)$, $\psi_{i,j} = \psi(ih_l, jh_l)$ for every $0 \le i, j \le N_l$, l being one grid the system becomes

$$\begin{cases} \dfrac{\psi_{i,j+1}-\psi_{i,j}}{h_l}\dfrac{T_{i+1,j}-T_{i,j}}{h_l} - \dfrac{\psi_{i+1,j}-\psi_{i,j}}{h_l}\dfrac{T_{i,j+1}-T_{i,j}}{h_l} = \dfrac{T_{i+1,j}+T_{i-1,j}+T_{i,j+1}+T_{i,j-1}-4T_{i,j}}{h_l^2}+1, \\ \dfrac{\psi_{i+1,j}+\psi_{i-1,j}+\psi_{i,j+1}+\psi_{i,j-1}-4\psi_{i,j}}{h_l^2} = Ra\left(\sin\phi\dfrac{T_{i,j+1}-T_{i,j}}{h_l} - \cos\phi\dfrac{T_{i+1,j}-T_{i,j}}{h_l}\right) \end{cases} \tag{13}$$

The solution of system (13) was obtained in two ways first, as the output of an serial algorithm and second, as the output of a parallel algorithm.

3.1.1. The serial algorithm The algorithm which solves (13) by means of up to seven grids ($l = 7$) contains the following steps

1 Solve the first equation of system (13) using (10) and (11); results T new;

2 Solve the second equation of system (13) using T new just determined, ψ_0 and ψ at the boundary; results ψ new;

3 Solve the first equation using ψ new and (11); results T new;

4 Repeats Steps 3 and 4 until "CONDITION" (When it is accomplish, the steady solution is obtained)

Note In our case, "CONDITION" means that the difference between two succesive approximation is less that 10^{-6} In other words, if we denoted, e g. F^{old} anf F^{new} two succesive approximations (where F represents T or ψ), "CONDITION" will be

$$\| F^{new} - F^{old} \| \leq 10^{-6}$$

where $\|\cdot\|$ denotes the Euclidean norm [4]. Fig 2a and b indicate the decreasing of error during ten repetitions of Steps 3 and 4 (Fig 2b detailes more the error at temperature function)

Concerning the results, our observations are the followings: the steady temperature has form like in Fig 3 and is not influenced by Ra number or ϕ angle Also, the general shape of the function (and this note is valable for the stream function, too) does not change with the numbers of grid points

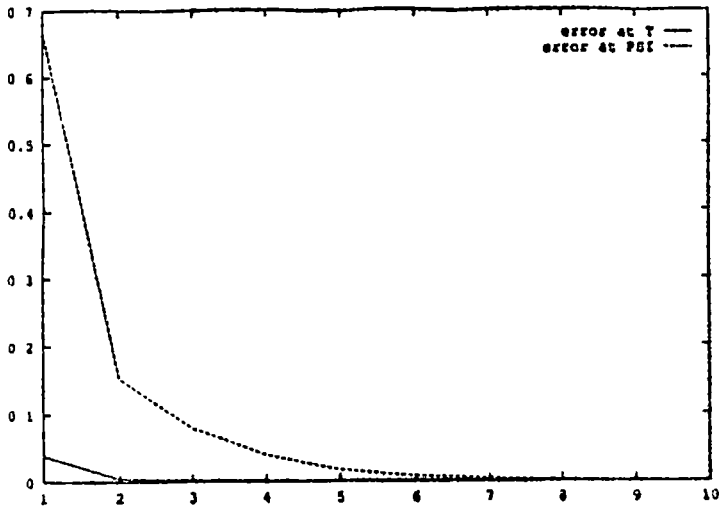


Fig 2a. The error

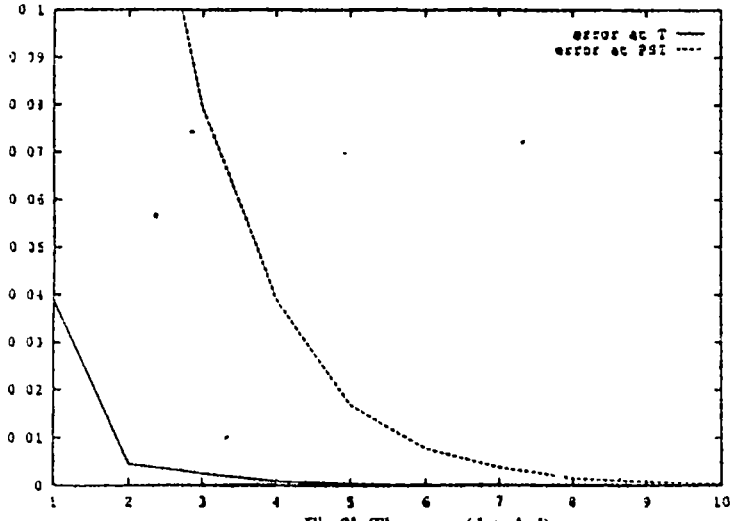


Fig.2b The error (detailed)

The stream function modifies according with the Rayleigh number and has the shape as in Fig 4

The stream function modifies also according with the angle of enclosure (see Fig 5a-c)

3.1.2. The parallel algorithm. The parallel algorithm was implemented on the INMOS Transputer System from University of Heidelberg, under PARIX operating system The main

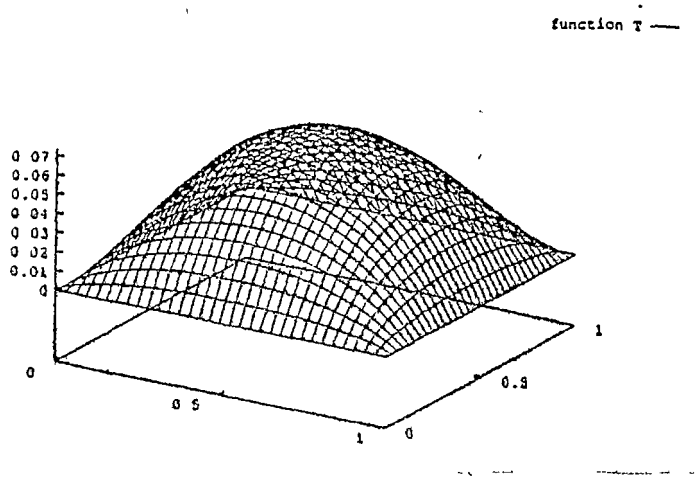


Fig 3 Temperature in steady case with Ra=500 and ϕ=0

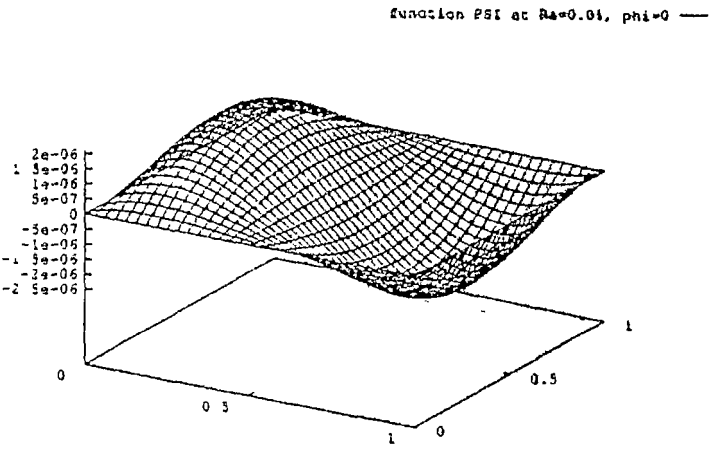


Fig.4 Stream function in steady case with Ra=0 01 and ϕ=0

ideea in solving our problem is that of [3], but with chages due to the convective terms (first equation) and the right-hand-side (second equation) from (12) We use a rectangular grid with $(N_l - 1) \times (N_l - 1)$ unknowns, then each processor is assigned to a subset of unknowns (data partitioning) In an one-dimensional arrengement of n processors caled a ring configuration of length n, processor $p, p \in \{0, \ , n-1\}$ is assigned to the grid points $\{(i,j) | \max(1, pN_l/n) \le i \le (p+1)N_l/n, 1 \le j \le N_l\}$ If the sidelength of the grid is not divisible by the number of

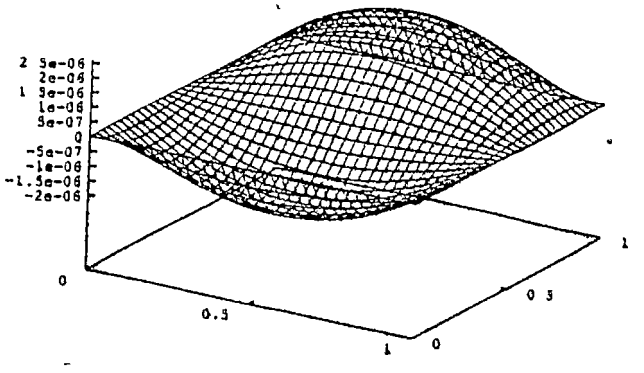


Fig 5a Stream function, with Ra=0 01 and ϕ=90

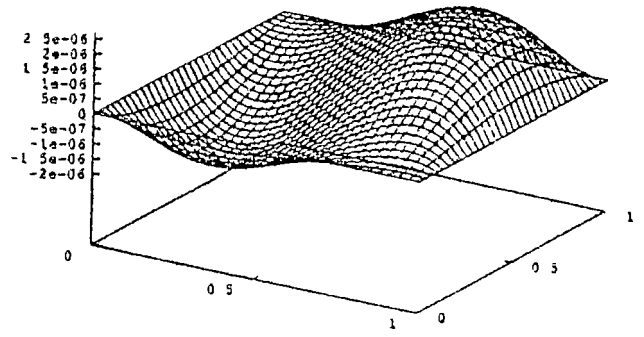


Fig 5b Stream function with Ra=0 01 and ϕ=145

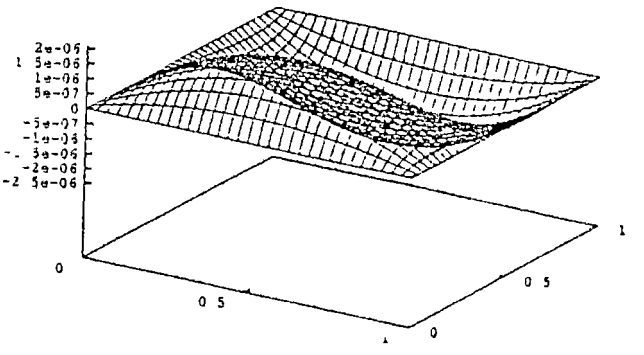


Fig 5c Stream function with Ra=0 01 and ϕ=270

processors, then some of them will be assigned more unknowns than others, generating an unequal load balance, which is one source for loss of efficiency Taking into account the way of disposing the grid points on processors and denoting by $x_{min}(p)$ and $x_{max}(p)$ the leftmost, respectively the rightmost grid point column stored by processor p, each processor will executes simultaneously the following steps

1 Computes the convective terms for the first equation of (12),

2 In case of an overlapping, sends values to the leftside processor (if it exists) and receives values from the righside processor (if it exists),

3 For every j from 1 to n_j do

3 1 Receives values from the leftside processor (if it exists),

3 2 For every i from $x_{min}(p)$ to $x_{max}(p)$ do

Computes Gauss-Seidel iterations,

3 3 Sends values to the rightside processor (if it exists)

After processing the previous steps, with step 3 repetead till the steady solution for temperature is obtained (we have noticed that it happened after 10 iterations), we proceed analogously to solve the second ecuation of (12).

In order to compare the results obtained with the serial and the parallel code, we used, like in [1] and [3], *the speed-up*, defined as

$$S(n) = \frac{T_{Mono}}{T_{Multi}(n)} \tag{14}$$

where T_{Mono} is the time needed for obtaining the solution the with the serial code and T_{Multi} is the time took by the parallel code, using n processors, and *the efficiency* which is defined

$$E(n) = \frac{S(n)}{n} \tag{15}$$

Table 1 presents the execution times (in sec) for the serial and the parallel code, when a different number of processors was used So, we can notice that the increasing of time for the serial code is deeply connected with the numbers of grid points (on a coarse grid, the execution takes a few seconds, the execution takes a few seconds, on a fine grid it takes more that an hour!) and the execution time decreases according with the number of processors used, with the observation that for the coarse grid 32 x 32 the situation is like in Fig 7

Table 1 Execution time

Nr proc/Nr pc	32×32	64×64	128×128	256×256	321×321
1	16 6029	67 9188	276 725	1116 65	1677 73
7	7 86234	19 2012	53 5307	167 708	240 032
11	7 472	17 1044	43 5171	126 039	173 89
15	7 59328	16.6198	39 6177	107 014	141 814
19	7 40141	16 0309	36 4428	95.6118	128 211
23	7 49709	15 6065	35 5842	89 002	116 689

Fig 6 visualises the information from Tabel 1, meanwhile Fig 7 indicates only an unconcludent situation when more that one processors are used

Table 2 Speed-up

Nr proc/Nr pc	32×32	64×64	128×128	256×256	312×312
7	2 11	3 53	5 16	6 65	6 98
11	2 21	3 97	6 35	8 85	9 5
15	2 18	4 08	6 95	10 43	11 8
19	2 24	4 23	7 59	11 67	13 08
23	2 21	4 35	7 76	12 54	14 3

The speed-up for all operations carried out on a fixed grid depends heavily on the number of unknowns per processor, because a larger proportion of computing time is spent on communication and the effects of unequal load distribution are more pronounced if the number of grid points per processor is small This means that a high speed-up can be

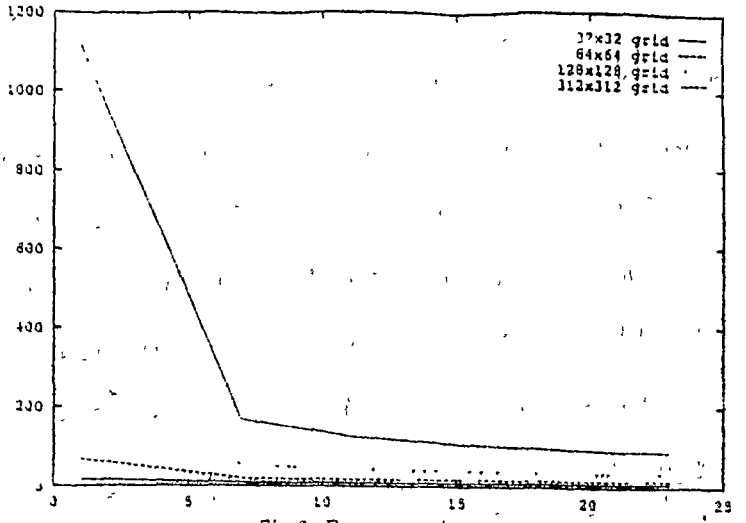


Fig 6 Execution time

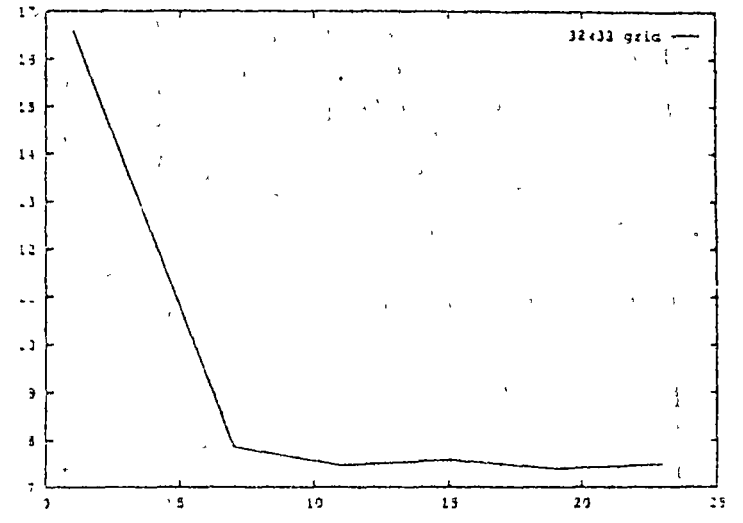


Fig 7 Execution time working with several processors on a coarse grid

achieved on the fine grids (assuming a large number of grid points per processor on the fine grids) like in Fig.9 whereas the speed-up deteriorates on the coarser grids (see Fig 10) Table 2 contains the values which sustained these observations and on which Fig 8 and 9 are based

Working with several processors on a coarse grid, the improving of speed-up is not

concludent, as we can see from Fig 10 Next, accordind with (15), Table 3 contains the values

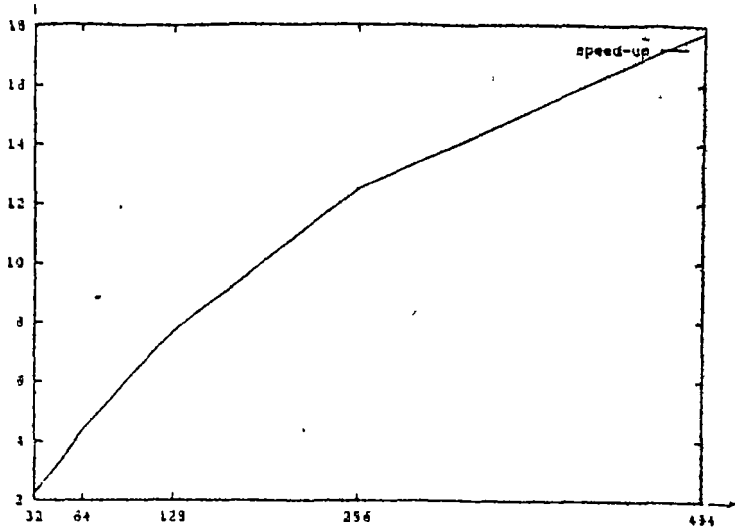


Fig 8 Speed-up (from coarser to finer grids with p=23)

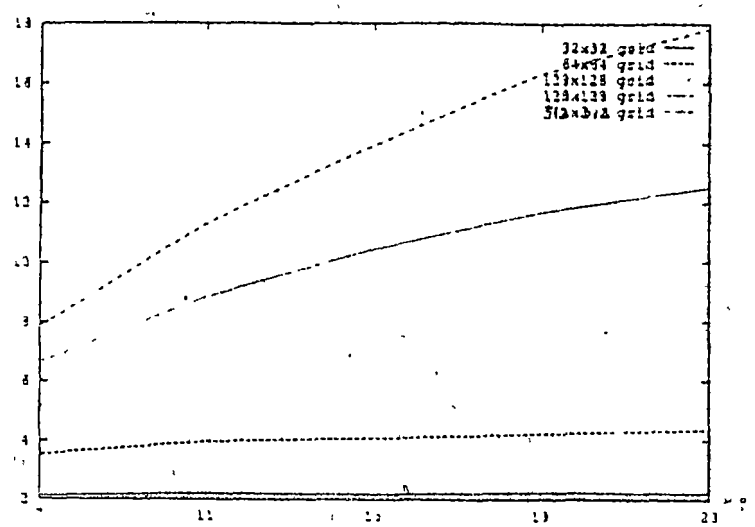


Fig 9 Speed-up

which indicate how efficiency depends on the number of processors and on the number of grids points

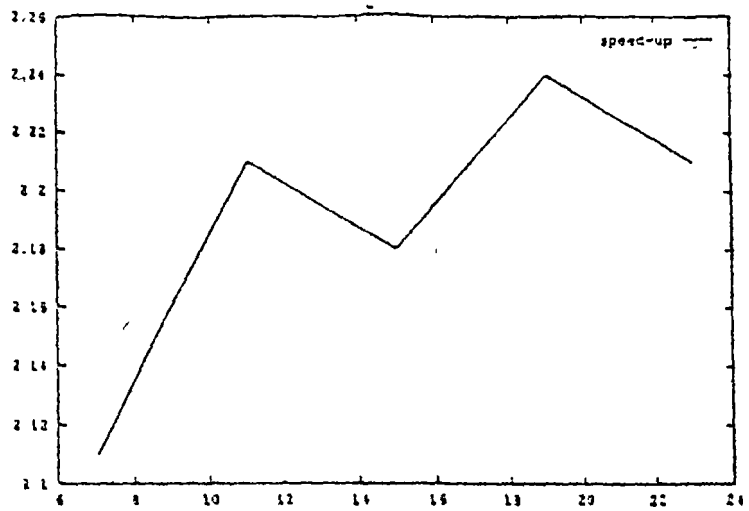


Fig 10 Speed-up on a coarse grid, with several processors

Table 3 Efficiency evolution

Nr proc/Nr pc	64×64	128×128	256×256	312×312
7	0 50	0 73	0 95	0 99
11	0 36	0 57	0 80	0 86
15	0 27	0 46	0 69	0 78
19	0 22	0 34	0 61	0 68
23	0 18	0 33	0 54	0 62

Based on Table 3 , Fig 11 shows the increasing of efficiency when finer grids are used

3.2.The Unsteady Problem Solving the unsteady problem means to solve the system in the original form (9) In order to do this, we use the same finite difference formulas to discretize the space derivatives, as in 3 1 The time derivative will be discretized with the backward Euler formula ([6)] We denote by dt the timestep, which is considered fix, by $L^{(l)}$ the Laplace operator and by G_x and G_y the gradient operators ([2]) Let T^k be the temperature function at the moment of time $t_k = kdt$ Then the first equation of system (9) can be written

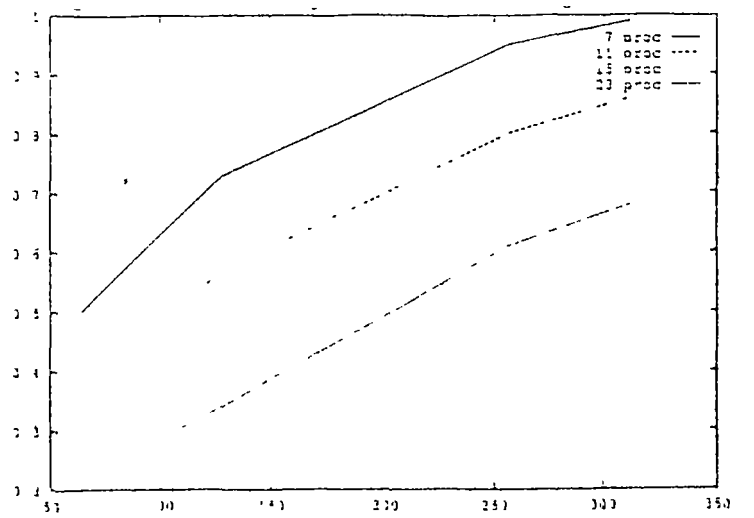


Fig 11 Efficiency

in the following manner

$$\frac{T^{k+1}-T^{k}}{dt}+G_y\psi^{k+1}G_x T^{k+1}-G_x\psi^{k+1}G_y T^{k+1}=L^{ell}T^{k+1}+1 \qquad (16)$$

For a fixed time interval $[t_k,t_{k+m}]$, denoting with I the Identity operator and based on (16), to solve the parabolic equation of system (9) means to solve the following bidiagonal blok-structured system

$$As=b$$

where

$$A=\begin{bmatrix} \frac{1}{dt}I+G_y\psi G_x\cdot+G_x\psi G_y\cdot-L^{ell}\cdot & 0 & \\ \frac{-1}{dt}I & \frac{1}{dt}I+G_y\psi G_x\cdot+G_x\psi G_y\cdot-L^{ell}\cdot & \\ 0 & \frac{-1}{dt}I & \frac{1}{dt}I+G_y\psi G_x\cdot+G_x\psi G_y\cdot-L^{ell}\cdot \end{bmatrix}$$

$$s = \begin{bmatrix} T^{k+1} \\ T^{k+2} \\ \\ T^{k+m} \end{bmatrix} \quad \text{and} \quad b = \begin{bmatrix} 1+\frac{1}{dt}IT^{k} \\ 1 \\ \\ 1 \end{bmatrix}$$

We observe that at every moment of time the relation which gives the temperature function is fully implicit and we have to solve, as the first equation of system (9), the following

$$\frac{T_{i,j}^{k+1}}{dt} + \frac{\psi_{i,j+1}^{k+1} - \psi_{i,j}^{k+1}}{h_i}\frac{T_{i+1,j}^{k+1} - T_{i,j}^{k+1}}{h_i} - \frac{\psi_{i+1,j}^{k+1} - \psi_{i,j}^{k+1}}{h_i}\frac{T_{i,j+1}^{k+1} - T_{i,j}^{k+1}}{h_i} -$$

$$- \frac{T_{i+1,j}^{k+1} + T_{i-1,j}^{k+1} + T_{i,j+1}^{k+1} + T_{i,j-1}^{k+1} - 4T_{i,j}^{k+1}}{h_i^2} = 1 + \frac{T_{i,j}^{k}}{dt} \qquad (17)$$

Equation (17) together with the second equation of system (13) will form the problem we have to solve in this case As in the paragraph 3 1 , the Multigrid method was used and the general scheme of solving is the following

Step 1 Solve equation (17) at the moment of time t^{k+1} based on T^k (where T^0, the initial temperature is given), results T^{k+1}

Step 2 Solve the second equation of system 9130 at the moment of time t^{k+1} based on T^{k+1} just determined results ψ^{k+1}

Step 3 Repeat Steps 1 and 2 until "CONDITION 1"

Note "CONDITION 1" indicates the number of time steps we have to execute until the steady solution is obtained, normally, thi depends on the value of dt For instance, if dt = 0 1, the steady solution is attain in mostly 10 steps, but for dt = 0 001 we need almost 180 time iterations to get it Fig 12a-2 show the evolution in time of the temperature function, for Ra = 500 , ϕ = 0 and dt = 0 001 After 180 time steps, the temperature is stationary (in

order to compare, see Fig 3)

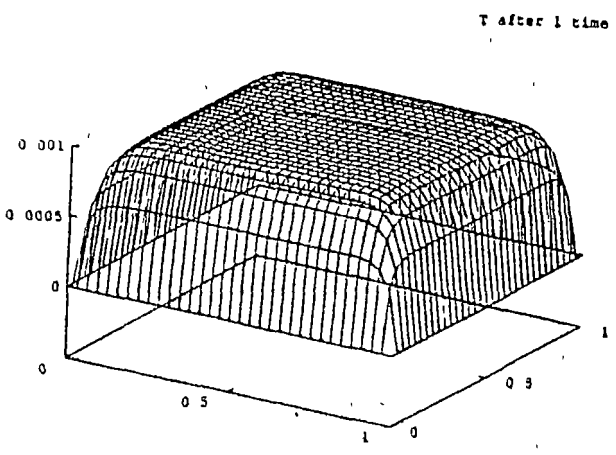


Fig 12a The temperature after 1 time step

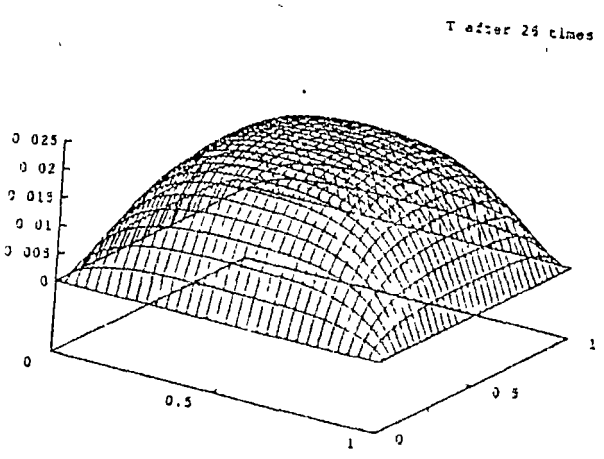


Fig 12b The temperature after 26 time steps

The following graphics show how the temperature function evoluates up to the steady case

In the same conditions (but for Ra = 125), Fig 13a-c present the evolution in time of the stream function

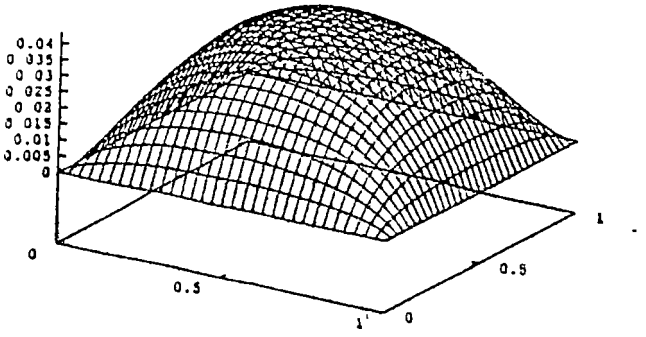


Fig 12c The Temperature after 51 time steps

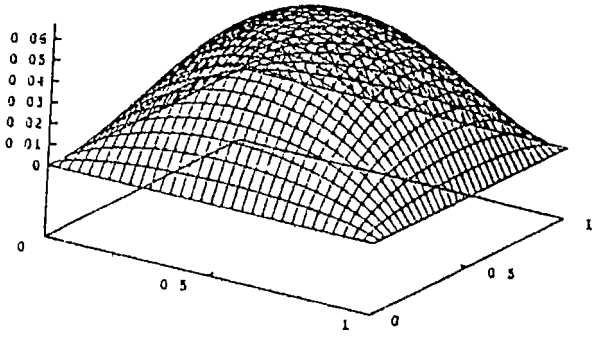


Fig 12d The Temperature after 131 time steps

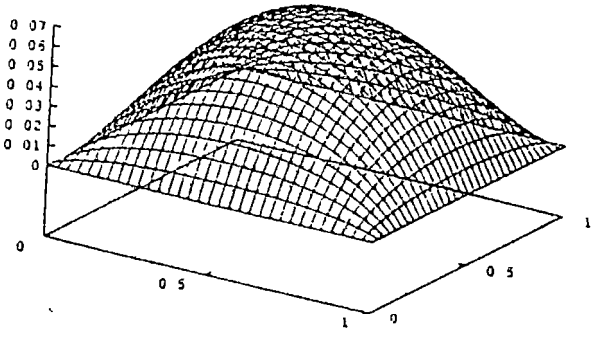


Fig 12e The Temperature after 180 time steps

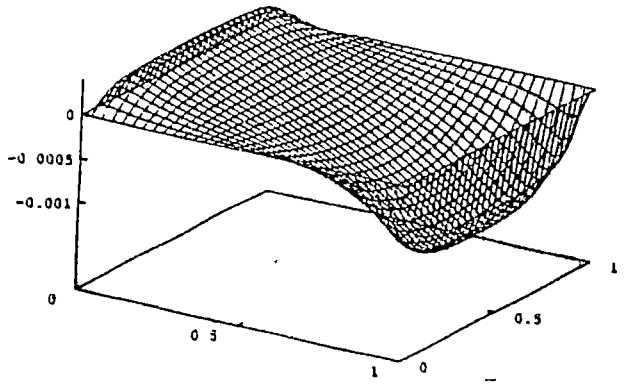


Fig 13a The Stream Function after 1 time step

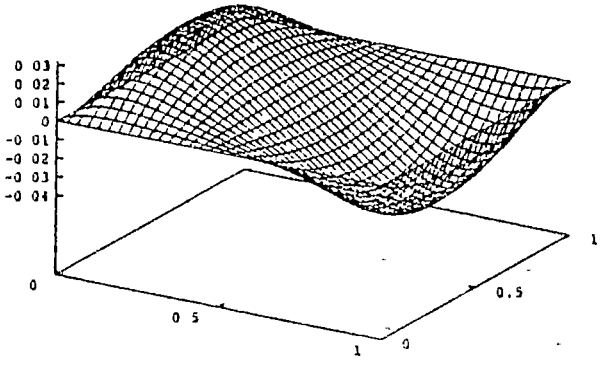


Fig 13b The Stream Function after 26 time steps

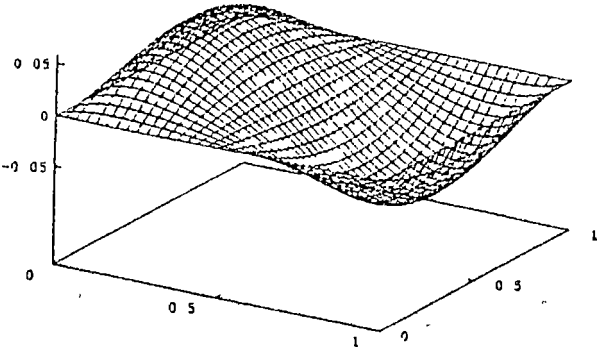


Fig 13c The Stream Function after 51 time steps

After 180 time steps, the stream function becomes steady (Fig 4), as we can see from the following graphics

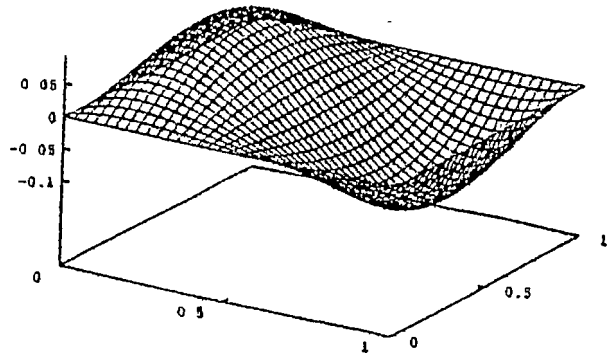


Fig 13d The Stream Function after 131 time steps

PSI after 180 time steps

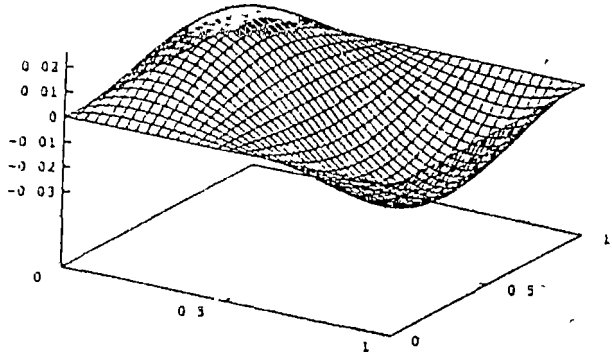


Fig 13e The Stream Function after 180 time steps

4.Conclusions. The main goal of this research was to show that transputer system can efficiently solve large computational problems with good performance We made study on a

problem of interest in the computational fluid dynamics field, which generated a parabolic problem expressed by a PDE system In order to verify the results, we solve first, in serial and in parallel, the steady problem The outputs of this two different codes were almost the sama Based on the steady solution, we solved then the original problem, indicating by means of many graphics the evolution in time, up to the steady state, of the solution functions

Acknowledgements. The author would like to especially thank to Prof Willi Jäger for his constant support Also, expresses her thanks to Peter Bastian and all the coleagues from IWR, University of Heidelberg, for their help in elaborating this paper

REFERENCES

1 Bader,G , Gehrke E , *On the performance of transputer networks for solving linear systems of equations*, Parallel Computing 17(1991), pp 1397-1407

2 Bastian,P , Burmeister,J , Horton,G , *Implementation of a parallel multigrid method for parabolic partial differential equations*, Proceeding of the Sixth GAMM-Seminar, Kiel, Jan 19-21, 1990, pp 18-27

3 Bastian,P , Horton,G , *Parallelization of Robust Multigrid Method ILU Factorization and Frequency Decomposition Method*, SIAM J SCI STAT COMP , vol 12, No 6, pp 1457-1470, nov 1991

4 Hackbusch,W , *Multi-Grid Methods and Applications*, Springer Verlag, Berlin, Heidelberg, 1985

5 May,H O , *A numerical study on natural convection in an inclined square enclosure containing internal heat sources*, Int J Heat Mass Transfer, Vol 34, No 4, pp 919-928/1991

6 Roache,J ,P , *Computational Fluid Dynamics*, Hermosa, Albuquerque, New-Mexico, 1985

7 Vasseur,P , Hung Nguyen,T , Robillard,L , Tong Thi,V K , *Natural convection between horizontal concentric cylinders filled with a porous layer with internal heat generation*, Int J Heat Mass Transfer, vol 27, no 3, pp 337-349/1984







STUDIA UNIV BABEŞ-BOLYAI, MATHEMATICA, XXXVIII, 3, 1993

GENERATING CONTROL STRUCTURES

V. CIOBAN, S. MOTOGNA, V. PREJMEREAN*

Dedicated to Professor Emil Muntean on his 60th anniversary

Received November 18, 1992
AMS subject classification 68N20, 68Q52

REZUMAT. - Generarea structurilor de control. Lucrarea prezintă o modalitate de a defini specificațiile formale cu ajutorul unei gramatici necontextuale

1. Introduction. The aparition of the programming environments generates an accentuated grow of programmers productivity With such a software instrument many actions can be performed editing a source file, compiling and linkediting of a program, execution, debugging even others facilities for files viewed as entities In fact, the aparition of microcomputers and programming environments made a combination of the programming work with the operating work in a calculus system The abandon of the "batch" working style and working interactively impose a specific training in operating a computer If the first programming environment have had restricted functions, the recent ones, as TURBO PASCAL or BORLAND C (considered in top of the classification), are very complex and are few specialists who can handle them completely However, the programming languages from these environments (PASCAL, C, C++) may be considered universal languages (solve a great number of problems technical, scientifical problems, problems which had to work with many informations and so, with files, graphical problems, object-oriented programming) and, that's

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

why handling all of the language facilities became difficult From another point of view languages as PASCAL, C++, COBOL or DBASE IV have thicker instructions, from the syntactical aspect, as FORTRAN We though that an instrument for automatic generation of control structures in a fixed language may be added as an important function in a programming environment

The problem of automatic generation of programs is not recent, and program generators exist in some systems and software products As an example we mention DBASE IV system which has a program generator based on graphical specification

We propose a model for generating some control structures of a program using context free grammars (1) A problem which hasn't been solved efficently is the specification of the structures

2. Control structures. For Dijkstra structures (see for example (2))and for other structures we will introduce the following operators

a) $C(s_1,s_2)$ - operator for concatenation structures s_1 and s_2 in this order ,

b) $\Delta(b,s_1,s_2)$ - operator associated to the complete alternative structure (complete IF) with the semnification

IF b THEN
s_1
ELSE
s_2
ENDIF,

c) $\Delta(b,s)$ - operator associated to the alternative structure with one alternative (simple IF) with semnification IF b THEN s ENDIF,

d) $*(b,s_1, \ldots ,s_n)$ - operator associated to the generalized alternative structure (CASE)

e) $\mho(b,s)$ - operator associated to pretested loop with the semnification

WHILE b DO
s
ENDWHILE,

f) $\Omega(s,b)$ - operator associated to posttested loop with the semnification

REPEAT
s
UNTIL b,

Are required some explanations

- the three Dijkstra are D={ C, Δ, $\mho$} and are considered fundamental, with them any algorithm can be described,
- we asociate operators for structures D'={ C, Δ, ⊾, *, $\mho$, Ω} which are in fact the structures from the PASCAL language,
- any other structure to which a similar operator can be asociated may be simulated with D or D' (for example LOOP-EXIT or LOOP-EXITIF-ENDLOOP structures),
- we may introduce the λ symbol for the empty structure

3. Proprieties of the asociated operators

1 $C(s_1,s_2) \neq C(s_2,s_1)$ - concatenation of structures s_1 and s_2 isn't comutative

2 $C(s_1,C(s_2,s_3)) = C(C(s_1,s_2),s_3)$ - concatenation is asociative

3 $C(s,\lambda) = C(\lambda,s) = s$ - the symbol of the empty structure is playing the role of the neutral element for concatenation

4 $C(\Delta(b,s_1,s_2),s_3) = \Delta(b,C(s_1,s_3),C(s_2,s_3))$ - concatenation is right distributed to alternative

structure

5 $C(s_1,\Delta(b,s_2,s_3)) = \Delta(b,C(s_1,s_2),C(s_1,s_3))$ - concatenation is distributed to left to alternative structure if and only if s_1 structure doesn't have any effect on b predicat

6 $\mho(b,s) = \Delta(b,C(b,\mho(b,s)),\lambda) = \Delta(b,C(s,\Delta(b,C(s,\mho(b,s)),\lambda)),\lambda) =$ - this propriety shows that the three D structure can be reduced to only two structures concatenation and the alternative structure

7 Reducing D' structures to D structures

a) $\text{ь}_{\cdot}(b,s) = \Delta(b,s,\lambda)$

b) $\text{ь}_{-}(b,s) = \Delta(b,s,\mho(c,s))$

c) $*(b,s_1, \ s_n) = \Delta(b_1,s_1,\Delta(b_2,s_2,\Delta(\ ,\Delta(b_{n-1},s_{n-1},s_n)\ )$

where b is formed from $b_1, \ ,b_{n-1}$

d) $\Omega(s,b) = C(s,\mho(\neg b,s))$, where $\neg b$ is the negation of b

8. Some equivalence proprieties

a) $\Delta(b,s_1,s_2) = C(b_1='T',C(\mho(b\wedge b_1,C(b_1='F',s_1)), \mho(b\wedge\neg b_1,C'(b_1='F',s_2))))$

Δ could be reduced to the operators C by introducing a new boolean variable b_1 ('T' is the value TRUE and 'F' is the value FALSE)

b) $\Delta(b,s_1,s_2) = C(\text{ь}_{-}(b,s_1),\text{ь}_{-}(\neg b,s_2))$ mentioning that s_1 doesn't modify b

4. Generating grammars for control structures. With the introduced notation we try to define a grammar which generates programms containing only control structures whose associated operators have been described One may give more than one grammar but we'll reffer only to the structures C, Δ, ь, $\mho$ and Ω

Having n structures s_1, ,s_n (which may be considered the simplest ones, namely attributing) and 2k predicates b_1, b_k and $\neg b_1$, ,$\neg b_k$ we give a grammar which generates all programms over the objects considered above

Let G = (N,Σ,P,S), where

N = {S,B} is the neterminals set

Σ = {C Δ Ь Ʊ Ω (,) s_1 s_n b_1 b_k $\neg b_1$ $\neg b_k$ }

is the alphabet of the grammar

P S --> C(S,S)|Ʊ(B,S)|Ω(S,B)|Ь(B,S)|Δ(B,S,S)|s_1| |s_n

B --> b_1| |b_k|$\neg b_1$| |$\neg b_k$

is the set of production rules

S - is the source symbol, S ∈ N

We consider the following examples

Example 1 The word

C(s_1,C(s_2,C(Ь(b$_1$,s_3),C(s_2,Ʊ($\neg b_2$,s_4)))))

which belongs to L(G) over s_1,s_2,s_3,s_4,b_1,b_2,$\neg b_1$,$\neg b_2$ may be obtained through "=>" in this way

S => C(S,S) => C(S,C(S,S)) => C(S,C(S,C(S,S))) =>

C(S,C(S,C(Ь(B,S),C(S,S)))) => C(s_1,C(s_2,C(Ь(b$_1$,s_3),C(s_2, Ʊ($\neg b_2$,s_4)))))

and it is equivalent with the following program

s_1,
s_2,
IF b_1 THEN s_3,
s_2,
WHILE $\neg b_2$ DO
s_4
ENDWHILE,

Example 2 Let's consider the following word

$$C(s_1, \Delta(b_1, (b_2,s_2), \Omega(s_3,\neg b_2)))$$

$\in$ L(G), which is obtained in this way

$$S \Rightarrow C(S,S) \Rightarrow C(S,\Delta(B,S,S)) \Rightarrow C(S,\Delta(B, (B,S),\Omega(S,B))) \Rightarrow$$
$$\Rightarrow C(s_1,\Delta(b_1, (b_2,s_2),\Omega(s_3,\neg b_2)))$$

and it is equivalent to the following program

```
s1,
IF b1 THEN
      WHILE b2 DO
          s2
      ENDWHILE
   ELSE
      REPEAT s3
      UNTIL ¬b2
ENDIF
```

The introduced grammar has the following properties ,

- is a simple precedence grammar

- there are no conflicts in grammar

We may prove that for any program (written in any language) only with structures C, Δ, $\llcorner$, $\mho$ and Ω exists one single word from L(G), which reproduces the program through operators

Different generators may be construct now having as input a word from L(G) and as output a program written in PASCAL, C, C++, COBOL, FORTRAN and so on The problem which hasn't been solved properly is the specification of the word from L(G) at input

REFERENCES

1 Aho,A V and Ullman,J D - The Theory of Parsing, Translation and Compiling, vol 1 and 2 Englewood Cliffs, New Jersey, Prentice Hall,1972

2 Dahl,O J , Dijkstra E W, Hoare,C A R - Structured programming Academic Press, London, 1972

STUDIA UNIV BABEŞ-BOLYAI, MATHEMATICA, XXXVIII, 3, 1993

SURFACES GENERATED BY BLENDING INTERPOLATION

Gh. COMAN*, I. GÂNSCĂ*, L. ŢÂMBULEA*

Dedicated to Professor Emil Muntean on his 60th anniversary

Received January, 21 1994
AMS subject classification 65D05, 65Y25

REZUMAT. - Suprafeţe generate prin interpolare blending. Folosind proprietatea funcţiei interpolatoare blending de a coincide cu funcţia pe care o interpolează pe puncte, segmente sau arce de curbă situate în domeniul de definiţie al funcţiei, sunt generate suprafeţe controlate de valori ale funcţiei şi derivate ale acestora de gradul I sau II

The blending interpolation has many practical applications As it is well know, blending interpolation is the interpolation at an infinite set of points segments, curves, etc Thus, if one gives the contour of an object by such elements (points, segments, curves) using a blending interpolation, we can generate a surface that contains the given contour Hence, we can construct a surface (a blending function interpolant) which mach a given function and certain of its derivatives on the boundary of a plan domain (rectangle, triangle, etc)

Using such a surface fitting technique it was constructed the roof surfaces for large halls (industrial halls, exposition halls, public buildings) [4,5,6,7,8]

Our goal is to construct some new such surfaces using Lagrange's, Hermite's and Birkoff's interpolatory operators

Let $T_h = \{(x,y) \in \mathbb{R}^2 \mid x \geq 0,\ y \geq 0,\ x+h \leq h\}$ be the standard triangle and $f\ T_h \to \mathbf{R}$ a given function

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

The operators used are.

1) Lagrange's operators L_1^x, L_1^y and L_1^{xy} defined by

$$\left(L_1^x f\right)(x,y) = \frac{h-x-y}{h-y} f(0,y) + \frac{x}{h-y} f(h-y,y)$$

$$\left(L_1^y f\right)(x,y) = \frac{h-x-y}{h-x} f(x,0) + \frac{y}{h-x} f(x,h-x)$$

$$\left(L_1^{xy} f\right)(x,y) = \frac{x}{x+y} f(x+y,0) + \frac{y}{x+y} f(0,x+y)$$

each of them interpolating the function f on two of the sides of T_h

2) Hermite's operators H_3^x, H_3^y and H_3^{xy} corresponding to the double nodes

$$\left(H_3^x f\right)(x,y) = \frac{(h-x-y)^2(h+2x-y)}{(h-y)^3} f(0,y) + \frac{x(h-x-y)^2}{(h-y)^2} f^{(1,0)}(0,y) +$$
$$+ \frac{x^2(3h-2x-3y)}{(h-y)^3} f(h-y,y) + \frac{x^2(x+y-h)}{(h-y)^2} f^{(1,0)}(h-y,y)$$

$$\left(H_3^y f\right)(x,y) = \frac{(h-x-y)^2(h-x+2y)}{(h-x)^3} f(x,0) + \frac{y(h-x-y)^2}{(h-x)^2} f^{(0,1)}(x,0) +$$
$$+ \frac{y^2(3h-3x-2y)}{(h-x)^3} f(x,h-x) + \frac{y^2(x+y-h)}{(h-x)^2} f^{(0,1)}(x,h-x)$$

$$\left(H_3^{xy} f\right)(x,y) = \frac{y^2(3x+y)}{(x+y)^3} f(0,x+y) + \frac{xy^2}{(x+y)^2}(f^{(1,0)} - f^{(0,1)})(0,x+y) +$$
$$+ \frac{x^2(x+3y)}{(x+y)^3} f(x+y,0) - \frac{x^2 y}{(x+y)^2}(f^{(1,0)} - f^{(0,1)})(x+y,0)$$

3) Birkhoff's operators B_1^x and B_1^y defined by

$$\left(B_1^x f\right)(x,y) = f(0,y) + (x+y-h) f^{(1,0)}(h-y,y)$$

$$\left(B_1^y f\right)(x,y) = f(x,0) - (x+y-h) f^{(0,1)}(x,h-x)$$

4) Birkhoff's operators B_3^x and B_3^y with

$$\left(B_3^x f\right)(x,y) = f(0,y)+\frac{x(x^2-3\lambda x+6h\lambda-3h^2)}{3h(2\lambda-h)}f^{(1,0)}(0,y)+$$
$$+\frac{x^2(2x-3h)}{3(2\lambda-h)}f^{(2,0)}(\lambda,y)+\frac{2x^2(3\lambda-x)}{3h(2\lambda-h)}f^{(1,0)}(h,y)$$

$$\left(B_3^y f\right)(x,y) = f(x,0)+\frac{y(y^2-3\gamma y+6h\gamma-3h^2)}{3h(2\gamma-h)}f^{(0,1)}(x,0)+$$
$$+\frac{y^2(2y-3h)}{3(2\gamma-h)}f^{(0,2)}(x,\gamma)+\frac{2y^3(3\gamma-y)}{3h(2\gamma-h)}f^{(0,1)}(x,h)$$

for $\lambda,\gamma\in[0,h]$

1 For the beginning we construct a scalar interpolating formula generated by the operators L_1^x, L_1^y and H_3^x, H_3^y and H_3^{xy}, using two levels of interpolation

First, the function f is approximated by the boolean sum of the operators L_1^x and L_1^y

(1) $$\left(L_1^x\oplus L_1^y f\right)(x,y)=\frac{h-x-y}{h-y}f(0,y)+\frac{h-x-y}{h-x}f(x,0)+\frac{y}{h-x}f(x,h-x)-$$
$$-\frac{h-x-y}{h}f(0,0)-\frac{y(h-x-y)}{h(h-y)}f(0,h)$$

In order to obtain a scalar approximant of f, we use in the second level the following approximations

$$f(0,y)\approx\left(H_3^y f\right)(0,y),\ f(x,0)\approx\left(H_3^x f\right)(x,0) \text{ and } f(x,h-x)\approx\left(H_3^{xy} f\right)(x,h-x)$$

Let

(2) $$f=Pf+Rf,$$

with

(3) $$\left(Pf\right)(x,y)=\frac{(h-x-y)(h^2+hx+hy-2x^2-2y^2)}{h^3}f(0,0)+\frac{x^2(3h-2x)}{h^3}f(h,0)+$$
$$+\frac{y(2hx+3hy-2x^2-2xy-2y^2)}{h^3}f(0,h)+\frac{x(h-x)(h-x-y)}{h^2}f^{(1,0)}(0,0)+$$
$$+\frac{y(h-y)(h-x-y)}{h^2}f^{(0,1)}(0,0)-\frac{x^2(h-x)}{h^2}f^{(1,0)}(h,0)+\frac{x^2y}{h^2}f^{(0,1)}(h,0)+$$
$$+\frac{xy(h-x)}{h^2}f^{(1,0)}(0,h)-\frac{y[y(h-x-y)+x(h-x)]}{h^2}f^{(0,1)}(0,h)$$

be the obtained interpolation formula

Theorem 1 If there exist $f^{(1,0)}(V_i)$ and $f^{(0,1)}(V_i)$, $i=1,2,3$, where V_i are the vertixes of T_h, then Pf interpolates f and its first partial derivatives at V_i, $i=1,2,3$

Also $Pg=g$ for all $g\in P_2^2$, i e the exactness degree of P is two

The proof of the theorem is a straighthforward computation

Theorem 2 If $f\in B_{1,2}(0,0)$ [10] then

$$(Rf)(x,y)=\int_0^h \varphi_{30}(x,y,s)f^{(3,0)}(s,0)ds+\int_0^h \varphi_{21}(x,y,s)f^{(2,1)}(s,0)ds+$$
$$+\int_0^h \varphi_{03}(x,y,t)f^{(0,3)}(0,t)dt+\iint_{T_h} \varphi_{12}(x,y,s,t)f^{(1,2)}(s,t)dsdt,$$

where

$$\varphi_{30}(x,y,s)=\frac{(x-s)_+^2}{2}-\frac{x^2(3h-2x)}{h^3}\frac{(h-s)^2}{2}+\frac{x^2(h-x)}{h^2}(h-s)$$

$$\varphi_{21}(x,y,s)=y(x-s)_+-\frac{x^2y}{h^2}(h-s)$$

$$\varphi_{03}(x,y,t)=\frac{(y-t)_+^2}{2}-\frac{y(2hx+3hy-2x^2-2xy-2y^2)}{h^3}(h-t)^2+$$
$$+\frac{y[y(h-x-y)+x(h-x)]}{h^2}(h-t)$$

$$\varphi_{12}(x,y,s,t)=(x-s)_+^0(y-t)_+$$

The proof follows by Peano's theorem for a triangular domain [2]

The approximation formula (2) is tested on the function $f(x,y)=1/(x^2+y^2+1)$ The graphs of the function f and of the approximation Pf are given in Fig 1 and Fig 2.

Remark Such an interpolation formula can be used to obtain a cubature formula over a triangle

2 Next, it will be used the given interpolatory operators to generate some surfaces on

the domain $D_h=\{(x,y)\in\mathbb{R}^2 \mid |x|+|y|\le h\}$

Such a surface is constructed first on the triangle T_h, after that is extended by symmetry with respect to the coordinate axes on all D_h

First examples of such surfaces are obtained from the approximation function Pf (3), for

(A) $f(0,0)=4, f(h,0)=f(0,h)=f^{(1,0)}(0,0)=f^{(0,1)}(0,0)=$
$=f^{(1,0)}(h,0)=f^{(0,1)}(0,h)=0$

and $f^{(1,0)}(0,h)=f^{(0,1)}(h,0)=-0\,5$ (Fig 3)

respectively

(B) $f(0,0)=4, f(h,0)=f(0,h)=f^{(1,0)}(h,0)=f^{(0,1)}(0,h)=0,$

$f^{(1,0)}(0,0)=f^{(0,1)}(0,0)=-1$ and

$f^{(1,0)}(0,h)=f^{(0,1)}(h,0)=-0\,25$ (Fig 4)

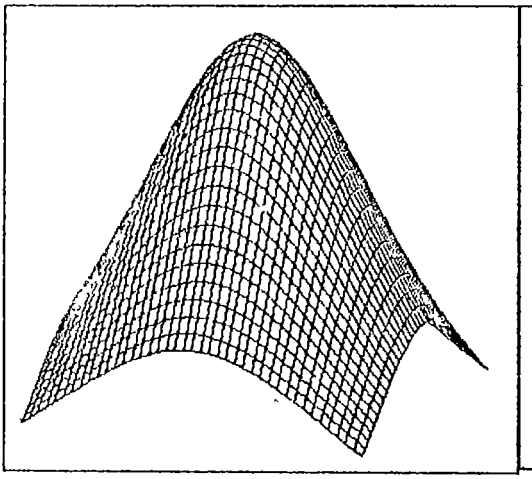


Fig 1

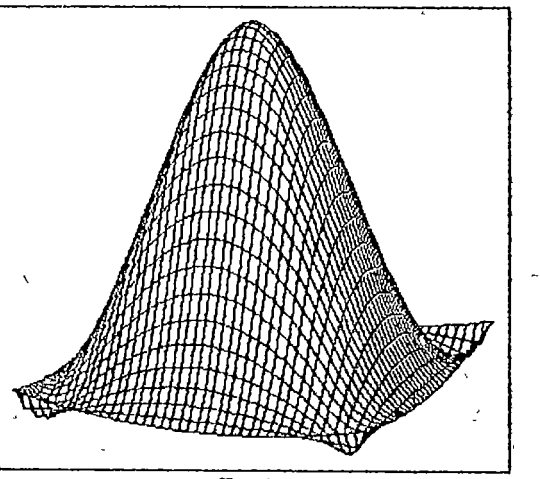


Fig 2

Now one supposes that the function f take the value zero on the border of D, i e $f|_{\partial D}=0$ This is equivalent with the condition $f(x,h-x)=0$ for $x\in[0,h]$ Using this condition

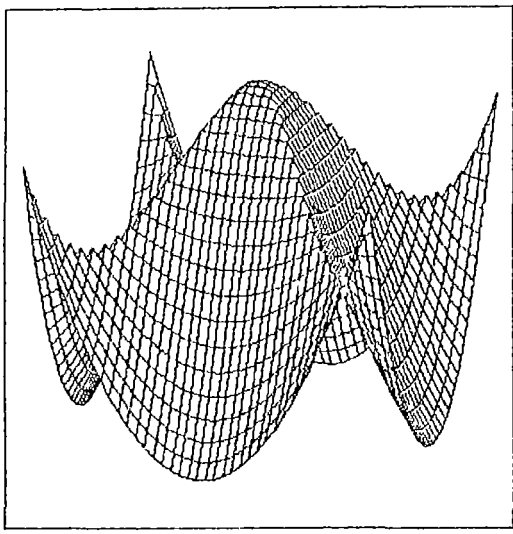


Fig 3

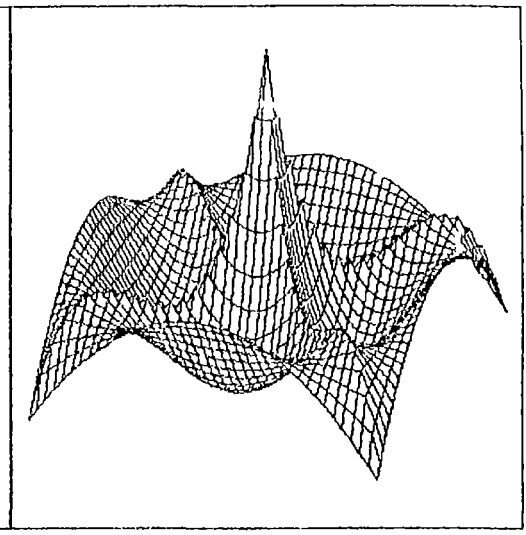


Fig 4

from (1) one obtains

$$L(x,y)=\frac{h-x-y}{h-y}f(0,y)+\frac{h-x-y}{h-x}f(x,0)-\frac{h-x-y}{h}f(0,0)$$

Taking $f(0,h)=\left(H_1^y f\right)(0,y)$ and $f(x,0)=\left(H_1^x f\right)(x,0)$, in the same condition $f(x,h-x)=0$ for all $x\in[0,h]$, one obtains the class of surfaces

$$H(x,y)=\frac{h-x-y}{h^3}\left[(h^2+hx-2x^2-2y^2)f(0,0)+hx(h-x)f^{(1,0)}(0,0)+\right.$$
$$\left.+hy(h-y)f^{(0,1)}(0,0)-hx^2f^{(1,0)}(h,0)-hy^2f^{(0,1)}(0,h)\right],$$

which depends on the data

$$(f(0,0),f^{(1,0)}(0,0),f^{(0,1)}(0,0),f^{(1,0)}(h,0),f^{(0,1)}(0,h))$$

For the data (4,-1,-1,-1,-1) one obtains the surface from the Fig 5

Another class of surfaces is given by the boolean sum of the operators G_3^x and G_3^y obtained from H_3^x respectively H_3^y in the conditions $f(x,h-x)=f^{(1,0)}(x,h-x)=f^{(0,1)}(x,h-x)=0$ for all $x\in[0,h]$, i e

$$\left(G_3^x f\right)(x,y)=\frac{(h-x-y)^2(h+2x-y)}{(h-y)^3}f(0,y)+\frac{x(h-x-y)^2}{(h-y)^2}f^{(1,0)}(0,y)$$

$$\left(G_3^y f\right)(x,y)=\frac{(h-x-y)^2(h-x+2y)}{(h-x)^3}f(x,0)+\frac{y(h-x-y)^2}{(h-x)^2}f^{(0,1)}(x,0)$$

We have

$$\left(G_3^x \oplus G_3^y f\right)(x,y)=(h-x-y)^2\left[\frac{h+2x-y}{(h-y)^3}f(0,y)+\frac{x}{(h-y)^2}f^{(1,0)}(0,y)+\right.$$
$$+\frac{h-x+2y}{(h-x)^3}f(x,0)+\frac{y}{(h-x)^2}f^{(0,1)}(x,0)-\frac{h^2+2hx+2hy+6xy}{h^4}f(0,0)-$$
$$\left.-\frac{x(h+2y)}{h^3}f^{(1,0)}(0,0)-\frac{y(y+2x)}{h^3}f^{(0,1)}(0,0)-\frac{xy}{h^2}f^{(1,1)}(0,0)\right]$$

Now, for

$$f(0,y)=\left(B_1^y f\right)(0,y)=f(0,0)+(y-h)f^{(0,1)}(0,h)$$

$$f(x,0)=\left(B_1^x f\right)(x,0)=f(0,0)+(x-h)f^{(1,0)}(h,0)$$

and

$$f^{(1,0)}(0,y)=\left(L_1^y f^{(1,0)}\right)(0,y)=\frac{h-y}{h}f^{(1,0)}(0,0)+\frac{y}{h}f^{(1,0)}(0,h)$$

$$f^{(0,1)}(x,0)=\left(L_1^x f^{(0,1)}\right)(x,0)=\frac{h-x}{h}f^{(0,1)}(0,0)+\frac{x}{h}f^{(0,1)}(h,0)$$

one obtains

$$G(x,y)=(h-x-y)^2\left\{\left[\frac{h+2x-y}{(h-y)^3}+\frac{h-x+2y}{(h-x)^3}-\frac{h^2+2hx+2hy+6xy}{h^4}\right]f(0,0)+\right.$$
$$+\frac{xy(2y-h)}{h^3(h-y)}f^{(1,0)}(0,0)+\frac{xy(2x-h)}{h^3(h-x)}f^{(0,1)}(0,0)-\frac{xy}{h^2}f^{(1,1)}(0,0)-$$
$$-\frac{h-x+2y}{(h-x)^2}f^{(1,0)}(h,0)+\frac{xy}{h(h-y)^2}f^{(1,0)}(0,h)-$$
$$\left.-\frac{h+2x-y}{(h-y)^2}f^{(0,1)}(0,h)+\frac{xy}{h(h-x)^2}f^{(0,1)}(h,0)\right\}$$

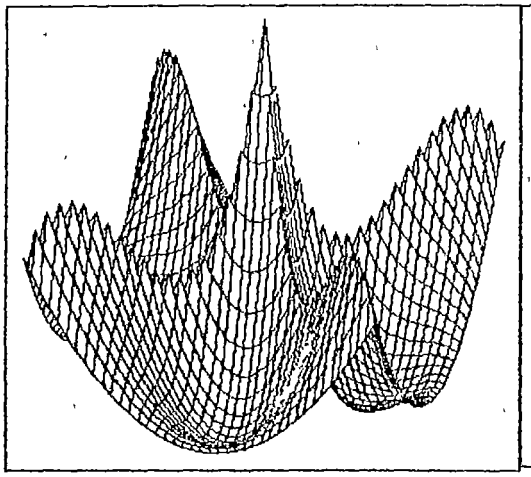


Fig 5.

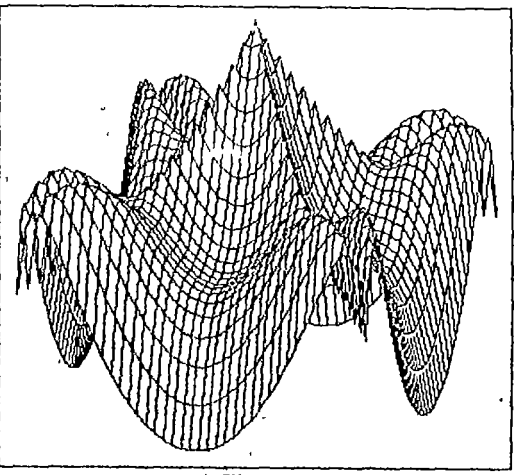


Fig 6

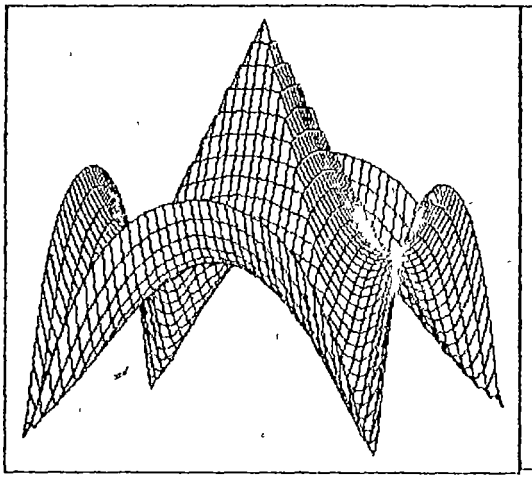


Fig 7.

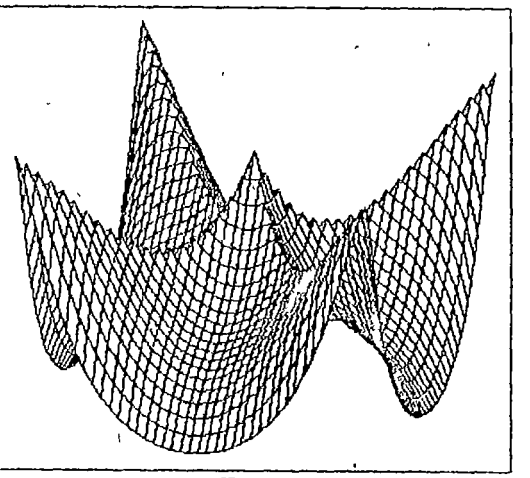


Fig 8

This surfaces depend on the data

$$(f(0,0),f^{(1,0)}(0,0),f^{(0,1)}(0,0),f^{(1,1)}(0,0),$$
$$f^{(1,0)}(h,0),f^{(1,0)}(0,h),f^{(0,1)}(0,h),f^{(0,1)}(h,0))$$

As an example (Fig 6) is given the surface obtained for the data (4,-1,-1,1,0,5,0,5)

The last class of surfaces is generated using the Fejer's type operators F_3^x and F_3^y obtained from H_3^x and H_3^y for

$$f^{(1,0)}(0,y)=f^{(1,0)}(h-y,y)=f^{(0,1)}(x,0)=f^{(0,1)}(x,h-x)=0$$

Taking into account the general condition that $f(x,h-x)=0$ for $x\in[0,h]$, one obtains

$$\left(F_3^x \oplus F_3^y f\right)(x,y)=(h-x-y)^2\left[\frac{h+2x-y}{(h-y)^3}f(0,y)+\right.$$
$$\left.+\frac{h-x+2y}{(h-x)^2}f(x,0)-\frac{(h+2x-y)(h+2y)}{h^3(h-y)}f(0,0)\right]$$

In order to control the inflexion points we take

$$f(0,y)=\left(B_3^y f\right)(0,y)$$
$$f(x,0)=\left(B_3^x f\right)(x,0)$$

One obtains

$$F(x,y)=(h-x-y)^2\left[\frac{h+2x-y}{(h-y)^3}\left(B_3^y f\right)(0,y)+\right.$$
$$\left.+\frac{h-x+2y}{(h-x)^2}\left(B_3^x f\right)(x,0)-\frac{(h+2x-y)(h+2y)}{h^3(h-y)}f(0,0)\right]$$

that depends on

$$(f(0,0),f^{(1,0)}(0,0),f^{(0,1)}(0,0),f^{(1,0)}(0,h),$$
$$f^{(0,1)}(0,h),f^{(2,0)}(\lambda,0),f^{(0,2)}(0,\gamma)),$$

where $\lambda,\gamma\in[0,h]$

Two example are taken here, for the data (4,-1,-1,0,0,0,0) with $\lambda=\gamma=5$ (Fig 7) and (4,-0 75,-0 75,0,0,2,2) with $\lambda=\gamma=15$ (Fig 8)

Finally, we remark that for any of the presented classes of surfaces, for convenable data, can be obtained a large variety of surfaces

REFERENCES

1 Barnhill, R E , Birkhoff, G , Gordon, W J , Smooth interpolation in triangles J Approx Theory, 8, 1973, 114-128

2 Barnhill, R E , Mausfield, L , Error bounds for smooth interpolation in triangles J Approx Theory, 11(1974), 306-318

3 Bohmer, K , Coman, Gh , Smooth interpolation schemes in triangle with error bounds Mathematica, 18(41), 1976, 15-27

4 Coman, Gh., Multivariate approximation schemes and the approximation of linear functionals Mathematica, 16(39), 1974, 229-249

5 Coman, Gh., Gânscă, I , An application of blending interpolation. Itinerant seminar of functional equations, approximation and convexity Cluj-Napoca 1983, Preprint nr.2, 1983, 29-34

6 Coman, Gh , Gânscă, I , Some practical applications of blending approximation Proceedings of the Colloquium on Approximation and Optimization, Cluj-Napoca, October 25-27, 1984.

7 Coman, Gh., Gânscă, I , Some practical applications of blending approximation II Itinerant seminar of functional equations, approximation and convexity. Cluj-Napoca 1986, Preprint nr 7, 1986, 75-82

8 Coman, Gh , Gânscă, I , Ţâmbulea, L , Some practical applications of blending approximation III Itinerant seminar of functional equations, approximation and convexity. Cluj-Napoca 1989, Preprint nr 7, 1989, 5-22

9 Coman, Gh , Gânscă, I , Ţâmbulea, L , Some new roof-surfaces generated by blending interpolation technique Studia Univ Babeş-Bolyai, Mathematica, XXXVI, 1, 1991, 119-130

10 Gordon, W J , Distributive lattices and the approximation of multivariate functions In "Approximation with special emphasis on spline functions" (ed by I J Schoenberg). Academic Press, New York and London, 1969, 223-227

11 Sard, A , Linear approximation AMS 1963

12 Stancu, D D , Generalizarea unor formule de interpolare pentru funcţii de mai multe variabile şi unele consideraţii asupra formulei de integrare numerică a lui Gauss Buletin St Acad R P Române, 9, 2, 1957, 287-313

13 Stancu, D D , The remainder of certain linear approximation formulas in two variables J SIAM, Numer Anal , 1, 1964, 137-163

14 Steffensen, J F , Interpolation Baltimore, 1950

STUDIA UNIV BABEŞ-BOLYAI, MATHEMATICA, XXXVIII, 3, 1993

PROGRAMMING PROVERBS REVISITED

M. FRENŢIU and B. PÂRV*

Dedicated to Professor Emil Muntean on his 60th anniversary

Received February 17, 1994
MS subject classification 68N05

REZUMAT. - Proverbe ale programării revăzute. În lucrare se prezintă metode, principii şi reguli considerate importante în activitatea de programare Se subliniază importanţa acestora în orice curs de învăţare a programării

Computer programming is still in a state of crisis, at least for two reasons the hardware changes, and the appearance of new problems which can be solved by computer The complexity of programs is increasing continuously, and it generates major changes in program design techniques The notion of "good program" can be considered from two different points of view programmer's view, and user's one From the user's point of view one can distinguish 10 so-called "external quality factors" [10] correctness, robustness, extensibility, reusability, compatibility, efficiency, portability, verificability, integrity, and ease of use From programmer's viewpoint, one can enumerate two major criteria for a good program modularity, and complete documentation Of course, the external quality factors must be taken into account as final goals in the software development process

All these quality criteria must find their place in the formation of new programmers There is a continuous need to teach programming for obtaining a better productivity, i e to

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

teach the students the methods that allow us to obtain correct programs from the first execution As Floyd [4] pointed out in his Turing Award Lecture, there "is possible to explicitly teach a set of systematic methods for all levels of program design" Methods, principles, and rules considered important in programming are given below Also the bibliographical source is indicated in the brackets

1 *Define the problem completely* [7, 9] One cannot write a correct program if the problem to be solved is not known exactly By this we mean to write the specifications of the problem As it is known [11], this is not an easy problem, but a very serious one Often the beginners start to write the program but they do not know what are the results that must be obtained

2 *Think first, program later* [9] This may be interpreted to design the algorithms correctly Think to them, try to prove their correctness, and write the program later, when you are sure that everything is correct

3 *Use Top-Down Design* [4, 7, 9] This is a very well known, and important programming paradigm [4] It is also met as *step-wise refinement* method [13], or *Divide and conquer* principle [7]

4 *Use Modularity as much as possible* [9] A function, a procedure, a Turbo-Pascal unit, a Modula module, or an Ada package are considered modules Each module of a program is more understandable than the entire program Also, using modules, the logical structure of the program is improved Build up libraries of your modules for reusability

5 *Use library routines whenever it is possible* [9] This rule is a consequence of rule 4 Certainly, the existing routines are ready to be used, no time needed for writing and testing

these routines Thus the productivity, and the probability of correctness will increase

6 *Design the algorithms by Structured Programming paradigm* [2, 4, 9, 13] This rule asks to design first the algorithms in a Pseudocode language, and only then to translate them in a programming language Also, it requires to think to the structure of the product, at each level

7 *Define a new data type as an Abstract Data Type (ADT)* [6] The above rule asks the designer to think generally, not in the context of the solved problem An ADT may be viewed as a module that defines a data structure and the operations on this structure This independence of the context has beneficial effect for the reusability of modules Also, an ADT is an open system, i e one can add new operations, not affecting the old ones, and not affecting the programs that already use this ADT

8 *Design input-output routines for each abstract data type* [5, 6] These Input-Output operations are very useful in general Often, when a standardized interface is recommended, for these operations one uses videoformat, such as Turbo Vision from Borland This rule is one way of achieving rule 21

9 *Use object-oriented design* [1, 10] This technique permits to obtain flexible, and easy modifiable programs The programs obtained by this technique are easy to maintain since, by using the hierarchy of classes in libraries of components, a massive reusability of these components becomes possible Also, adding new components does not affect the programs that already use the old components On the other side, the other feature of object-oriented programming, the polymorfism, simplifies communication protocol between objects A program in OOP sense is considered as a structured collection of objects which

communicate by message passing

10 *Strive for continuing invention, and elaboration of new paradigms to the set of your own ones* [4] This idea, due to Floyd, is very well presented in his Turing Lecture He recommended to "identify the paradigms you use, as fully as you can, then teach them explicitly"

11 *Prove the correctness of algorithms during their design* [7] The errors must be eliminated as soon as possible. Trying to prove correctness, some wrong parts may be discovered And this can be done much earlier than running it on a computer Also, if we succeeded to prove it, the confidence in its correctness grows up significantly Gries [7] insists on developing correct programs from the beginning His words are "A program and its proof should be developed hand-in-hand, with the proof usually leading the way"

12 *Concentrate to the important things of the moment, postpone the details later* [9, 13] This rule is connected to the stepwise refinement method But it has some other aspects At all levels give attention to the main things, for example do not lose time to print the results nicely if you are not sure these results are correct

13 *Nevertheless the details are important* [6, 13] First, the software products must respect rigorously the specifications Second, the form of the printed results are more important for users than the entire work done for developing the product These must please the users !

14 *Choose suitable and meaningful names for variables* [7, 13] The readability of a program may be one of its very important quality It is very useful during maintenance phase, when many other programmers have to work on the program More, Gries [7] recommends

to define rigorously the meaning of a variable by an assertion that remains true during the execution of the algorithm

15 *For every variable of a program make sure that it is declared, initialised, and used* [12] A variable may appeared in a program accidentally, other variable may not be initialised since a line of a program was not typed

16 *Use symbolic constants* [6, 13] This rule is a consequence of a Murphy like rule

The constants must be considered variables !

One recommends to define symbolic constants at the beginning of a program (module) procedure and to use the names inside Any modifications means small changes in the definition of the constants, and eliminates further errors

17 *Use names for all data types of the program* [6] We consider that all properties of a type are concentrated in its name Using names, the modifiability of the program is easier Also, the clarity is higher

18 *Use intermediate variables only if it is necessary* [9]

The uncontrolled utilization of auxiliary variables, by breaking expressions, just complicated, in subexpressions assigned to new variables, diminishes the clarity of the program, and makes more difficult the program verification

19 *Declare all auxiliary variables of a procedure as local variables* [6] This rule is connected with the autonomy of the corresponding procedure It offers the following advantages easier testing of the procedure, procedure independence of the context in which it is used, no secondary effects due to unexpected changing of the values of global variables

20 *Be careful at the parameters of the called procedure* [6, 13] Each module must

be used only through its interface, that is, the actual parameters passed to the module, which must correspond to the formal parameters (dummy variables) Respect their meanings, and be careful to the correct usage of the procedure calling mechanism

21. *Verify the value of a variable immediately it was obtained* [6] A variable receives a value by an assignment or by an input operation. In both cases the value must be correct, it is worthwhile to check it Especially for input operation, a variable must be protected from wrong values

22 *Think to pretty writing the text of the program* [9,13].

Most of the programming languages allow free format, i e. the blank spaces may be used freely Use them when writing the text of the program, to improve the clarity of this text It must leap to the eyes the beginning and the end of each statement Use indentation for this purpose Make the structure of your program visible.

23 *Use the FOR statements properly; do not change the value of the counting variable, or the limits inside the cycle* [9] This rule ask to respect the semantics of the For statement Do not use For when Repeat or While control structures are most appropiate Changing the limits, or the value of the counting variable may cause invisible errors, very difficult to discover

24 *Do not leave a FOR cycle through a Goto statement* [9] This rule is specific to Fortran programmers, but may be met in those languages that possess GOTO statements The reasons for respecting this rule are the same as for the rule 23

25 *Avoid GOTO statements* [3] The Goto controversy [3, 8] is well known Using unrestricted Goto statements destroys the good structure of that module These statements

must be used only if the programming language does not possess the standard computing structures

26 *Avoid tricky programming* [9] A program must be maintained, oftenly, by other persons different from the people who wrote it And tricks are not compatible with good structure, clarity, and flexibility. Also, for the portability of the program, one must avoid the implementation dependent features.

27. *Use comments* [9, 13]. The text of a program (module) must be understood easily and unambiguously by all the other programmers who have to read it For this purpose the comments can be very useful. We think that each module must contain comments saying at least what it is doing, i e the specifications of the module, and the meaning of the used variables

28 *Verify (test) the correctness of a module soon after it was obtained* [7]. The rule 10 ask us to prove formally the correctness of a program (module) But, just if we have done it, we still have to test this module After all, the proof may be wrong, or the implementation of a correct algorithm may be incorrect. Ledgard [9] recommended "to hand-check the program before running it" We find this very useful for the beginners, some students better understand their errors running themselves their wrong programs

29 *At each phase verify the programm correctness* [6, 13] The verification of program correctness means the verification of specifications, the formal proof of algorithm correctness, the inspection of the text of the program, and the testing of it Remove any error as soon as possible !

30 *Use assertions to document programs and verify their correctness during*

debugging process [7] If one has proved the partial correctness of the algorithms he has used assertions in some points of the algorithms These assertions must be invariantly true during execution They reflect the meaning of the corresponding variables. In the debugging process verify their correctness. If they are not true some errors have occured, and they must be eliminated

31. *Write good documentation simultaneously with program building* [13] The users need a documentation manual, and the maintenance activities need information about all levels of program development Often, there is no documentation at all. The above mentioned rule asks to write the documentation simultaneously with the development of the program. The program itself must be selfdocumented by comments But it is not enough There must be written documents that show all the decisions at each level of the development process. There must exist documents for specification, design, implementation, and testing. Also, a user manual is needed

32. *Use the existing debugging techniques* [9] We hope to obtain error-free programs But errors may arise, and finding and correcting these errors is an important, and very often, an unpleasant job Every operating system has built in it some debugging aids Use them to assist you in finding the errors

33 *Ask for computer assisted software development* [7, 13] Computers can help people to carry out their unpleasant works Particularly, they can help in program development in different ways Many of them are mentioned in the excellent book of Schach [13] planning the activities, and many activities done by Software Development Environments, known as CASE (Computer-Aided Software Development) There are many

activities that have to be performed during the development of the program, such as computations [4], or various decisions

34 *Think to the program portability* [9] A program must be portable, i e to be able to be run directly on a different machine, other than the original one Portability is not usually an issue to worry about But it may be an important quality of a program Isolate into modules those parts of the program that usually change from computer to computer (such as input/output operations). All other modules can be built portable, using statements corresponding to the "standard specification" of the implementation language, and avoiding the particular extensions which are dependent on the compiler implementation

REFERENCES

1 Coad,P , Yourdon,E *Object-oriented Design*, Prentice-Hall, 1991
2 Dahl,O J , E W Dijkstra, C A R Hoare. *Structured programming*, Academic Press, London, New-York, 1972
3 Dijkstra,E W *GOTO Statement Considered Harmful*, Comm A C M , 11(1968),no 3, p 148
4 Floyd,R W *The Paradigms of Programming*, Comm A C M , 22(1979), no 8, pp 455-460
5 Frenţiu,M , B Pârv, V Prejmereanu *Abstract data types for increasing the productivity in programming*, Seminar on Computer Science, "Babeş-Bolyai" University, Preprint no 5, 1992, pp 8-13
6 Frenţiu,M , and B Pârv, *Metode şi tehnici în elaborarea programelor*, Promedia, Cluj-Napoca, 1994
7 Gries,D *The Science of Programming*, Springer Verlag, Berlin, 1985.
8 Knuth,D *Structured Programming with GO TO Statements*, A C M Computing Surveys, 6(1974), no 12, pp 261-301
9 Ledgard,H F , *Programming proverbs for Fortran programmers*, Hayden Book Company, Inc , New-Jersey, 1975
10 Meyer,B *Object Oriented Software Construction*, Prentice-Hall, Englewood Cliffs, 1988
11 Myers, A *A Controlled Experiment in Program Testing and Code Walkthroughs Inspection*, Comm A C M , 21(1978), no 9, pp 760-768 .
12 Naur,P *Proof of algorithms by general snapshots*, BIT, 6(1966), pp 310-316
13 Schach,S R *Software Engineering*, IRWIN, 1990, U S A

STUDIA UNIV BABEŞ-BOLYAI, MATHEMATICA, XXXVIII, 3, 1993

CONSEQUENCES OF THEOREMS CONCERNING THE CONVERGENCE OF CHORD METHOD

Sever GROZE* **and Ioana CHIOREAN***

Dedicated to Professor Emil Muntean on his 60th anniversary

Received February 21, 1994
MS subject classification 65C20

REZUMAT. - Consecinţe ale teoremelor privind convergenţa metodei coardei. Lucrarea îşi propune de a pune în evidenţă câteva consecinţe ale unei teoreme de convergenţă ale metode coardei

$$x_{n+1} = x_n - \Lambda_n P(x_n)$$

metodă folosită în rezolvarea ecuaţiei $P(x) = \theta$, unde $P\ X \to Y$, X, Y fiind spaţii Fréchet

1. In this paper some consequences of the convergence of Chord method are given

Let be the equation

$$P(x) = \theta \tag{1}$$

where $P\ X \to Y$ is a continuous nonlinear mapping, X and Y Fréchet spaces [3], $\theta \in Y$ the null element of the space

Let be any $x_0, x_{-1} \in D \subset X$ and $\Lambda_n = \left[x_n, x_{n-1}, P\right]^{-1}$ the generalized divided quotient [2] of P

Starting from the initial approximation x_0, x_{-1} and using the algorithm

$$x_{n+1} = x_n - \Lambda_n P(x_n) \tag{2}$$

known as "the Chord method", the sequence (x_n) is generated, each term of it being an

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

approximate of the solution of (1)

Obviously, the Chord method cannot be applied in the following two situations

a) applying the algorithm (2), x_n terms of sequence which are not in D are generated,

b) the mapping $[x_n, x_{n-1}, P]^{-1}$ does not exist

To apply the iterative method (2) at each step, the mapping $[x_k, x_{k-1}, P]^{-1}$ is needed To avoid this inconvenient, a "modified" method may be applied

$$x_{n+1} = x_n - [x_0, x_{-1}; P]^{-1} P(x_n) \tag{2'}$$

which, to generate the (x_n) approximations, uses only the mapping

$$A_0 = [x_0, x_{-1}, P]^{-1}$$

Although it gives a "weaker" approximation that (2), it is often use in practice

We mention that both the Chord method (2) and the modified one (2') applied to the approximative solving of equation (1) are identical with the succesive approximations method

$$x_{n+1} = A(x_n) \qquad (n = 0, 1, \ ) \tag{3'}$$

applied to the equations equivalent with (1), respectively

$$x = x - [x^{(1)}, x^{(2)}, P]^{-1} P(x) \tag{$\tilde{3}_1$}$$

and

$$x = x - [x_0, x_{-1}, P]^{-1} P(x) \tag{3_2}$$

Concerning the convergence of Chord method, in [1] the following theorem is proved

THEOREM A *If the following conditions are satisfied for initial approximates* $x_0, x_{-1} \in X$

1 $A_0 = [x_0, x_{-1}, P]^{-1}$ *exists,*

2 $)| A_0 P(x_i) | (\leq \eta_i, \ i = 0, -1$ *and* $\eta_0 < 1/4\, \eta_{-1}$;

3 $)|\Lambda_0[u,v,w,P]|(\le \tilde{K},\ \forall\ u,v,w \in S(x_0, 5/4\eta_{-1})$,

4 $\tilde{h}_0 = \tilde{K}\eta_{-1} \le 1/4$,

then the equation (1) *has at least one solution* $x^* \in S$, *which is the limit of sequence* (x_n) *generated by* (2), *the order of convergence being*

$$)|x^* - x_n|(\le \frac{1}{2^{s_n-1}} q^{s_{n-1}} (4h_0)^s \eta_0 \tag{4}$$

where $0 < q < 1$, *and* s_n *is the general term of the sequence of partial summas of Fibonacci sequence, with* $u_1 = u_2 = 1$

2. In the following, we modify the hypothesis concerning the existence of mapping $\Lambda_0 = [x_0, x_{-1}, P]^{-1}$, using another mapping, connected with it

We probe the following

THEOREM 1 *Supposing the existence of any continuous linear mapping* $\Lambda \in (Y,X)^*$ *which has an inverse and the following conditions fulfilled for initial approximates* $x_0, x_{-1} \in S \subset X$

1^0 $)|\Lambda P(x_i)|(\bar{\eta}_i,\ \ i = 0,-1$ *and* $\bar{\eta}_0 \le 1/4\,\bar{\eta}_{-1}$,

2^0 $)|\Lambda[x_0, x_{-1}\ P]-I|(\le a < 1$, *I beeing the identical mapping,*

3^0 $)|\Lambda[u,v,w,P]|(\le \bar{K}, \forall\ u,v,w \in S(x_0, 5/4\,\bar{\eta}_{-1})$,

4^0 $\bar{h}_0 = \dfrac{\bar{K}\eta_{-1}}{(1-a)^2} \le 1/4$

then the equation (1) *has a solution* $x^* \in S$, *which is the limit of sequence* (x_n) *generated by* (2), *the order of convergence being*

$$)|x^* - x_n|(< \frac{1}{2^{s_n-1}} q^{s_n} (4\bar{h}_0)^{s_n} \bar{\eta}_0 \tag{5}$$

where s_n *and* q *has the significance given bellow*

Proof We show that, from the hypothesis of theorem 1, the conditions of theorem A follow

Hypothesis 2^0 of theorem 1 implies, based on Banach's theorem, the existence of mapping

$$H = (\Lambda[x_0, x_{-1}, P])^{-1} \tag{6}$$

which for

$$)|H|(\leq \frac{1}{1-a}$$

It follows the existence of

$$H\Lambda = \Lambda_0 = \left[x_0, x_{-1}, P\right]^{-1}$$

so the condition 1^0 of theorem A is verified

To fulfill the condition 2^0 of the same theorem, we consider

$$)|\Lambda_0 P(x_i)|(\ =\ )|H\Lambda P(x_i)|(\ \leq$$

$$\leq)|H|(\cdot)|\Lambda P(x_i)|(\leq \frac{\bar{\eta}_i}{1-a},\quad i = 0,-1$$

Changing η_i respectively with $\frac{\eta_i}{1-a}$, $i = 0,-1$, we obtain te condition 1^0 of theorem A

In order to obtain the condition 3^0 of theorem A, we have

$$)|\Lambda_0\left[x^{(1)}, x^{(2)}, x^{(3)}; P\right]|(\leq)|H\Lambda\left[x^{(1)}, x^{(2)}, x^{(3)}, P\right]|(\leq$$

$$\leq \frac{\bar{K}}{1-a}$$

so $\tilde{K}$ corresponds to $\frac{\bar{K}}{1-a}$

According with the expressions for K and η_{-1} we may evaluate $\bar{h}_0$, so the condition 4 of theorem A

Then due to theorem A, it results the existence of solution for equation (1), which is

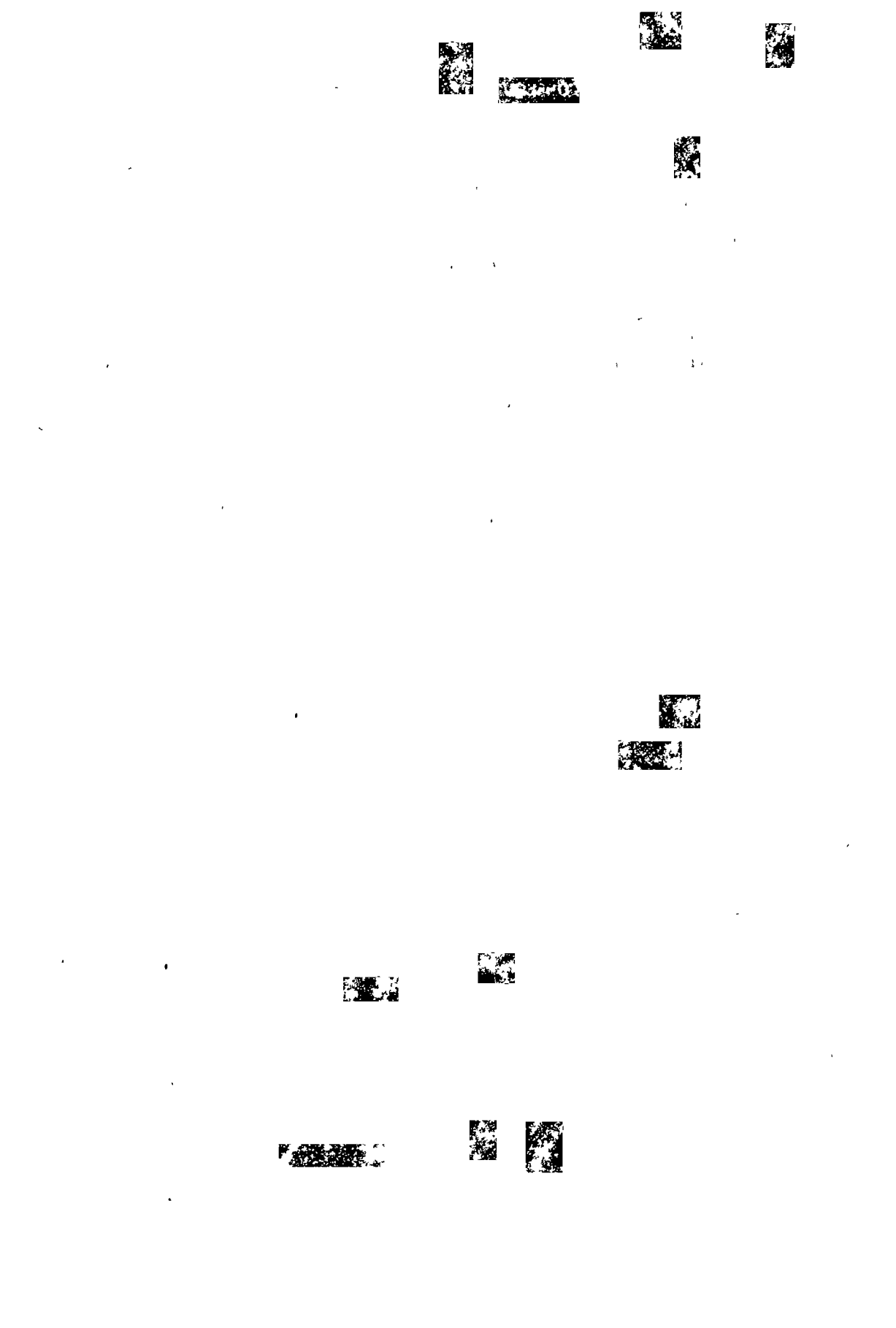
the limit of sequence generated by (x_n), the rapidity of convergence being given by (5)

REFERENCES

1 Groze,S , Goldner,G , Jankó,B , *Asupra metodei coardei în rezolvarea ecuațiilor operaționale definite în spații supermetrice*, Studii şi Cercet Mat , 5, Tome 23, 1971, pp.719-725

2 Groze,S , Innkó,B , *Asupra diferențelor divizate generalizate*, Anal Univ Al I.Cuza, Iaşi, Sect I, XVIII, 1971, fasc 2, pp 375-379

3 Rolewicz,S , *Metric Linear Spaces*, P W N Warszawa, 1972



STUDIA UNIV BABEŞ-BOLYAI, MATHEMATICA, XXXVIII, 3, 1993

FORMAL SPECIFICATION FOR SMALLTALK THROUGH LAMBDA-CALCULUS. A COMPARATIVE STUDY

Simona MOTOGNA*

Dedicated to Professor Emil Muntean on his 60th anniversary

Received January 31, 1994
MS subject classification 68N05, 68Q55, 68Q60

REZUMAT. - Specificarea formală prin lambda-calcul a limbajului Smalltalk. Studiu comparativ. În această lucrare sunt discutate două modele de specificații prin lambda-calcul ale limbajului Smalltalk. Prin considerarea unei ierarhii în mediul Smalltalk au fost comparate cele două modele din punctul de vedere al criteriilor pe care o specificație trebuie să le respecte

Introduction. Denotational semantics based on lambda-calculus has been a very used specification method in some models for formalization of the object oriented languages Cardelli [1] stated that the only notion critically associated with object oriented programming is inheritance This paper tends to present a comparative study of some denotational specification models for inheritance. All the models presented are based on the object oriented language Smalltalk so the study will be somehow easily

Inheritance is the possibility to define a new class (named subclass) using the definition of one or more existing classes (named superclass) A subclass can inherit instance variables or methods from the parent class The meaning of this property can be understand using a "look-up" method Suppose a message, containing the call of a method, is sent to an object Then the look-up method search the class containing the method.

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

```
        procedure lookup (name, class)
    if name = local_name then do local_action
      else if (inherited_module= NIL) then undefined_name
          else lookup (name, inherited_module)
```

In Smalltalk there are two special variables which can appear in a message These two variables are ***self*** and ***super*** When the message contains the variable **self** the search begins in the instance class

lookup (name, instance class)

and if the message contains the variable **super** then the search begins in the superclass of the instance class (which contains the method)

lookup (name, superclass of the instance class)

The mechanism of **self** and **super** supports the access of the methods which have the same names either from the superclass and the subclass, although they have a different action If a subclass redefines a method which was defined in superclass then this mechanism became very useful

Kamin's specification model

In [5], Samuel Kamin proposes a denotational definition for Smalltalk The major characteristic of this definition is the simple way in which inheritance is handled and the paper contains an version of this semantics in Standard ML which can be executed

The Smalltalk defined by Kamin has some modifications.

- only a few primitives are defined,
- the only literals which are permitted in the language are the integers and the arrays,
- the *pools* variables are omitted, excepting class variables,

- contexts are not objects,

- methods are not objects, so it isn't possible to create methods dynamically,

- there is a special way in which the array constants are handled any time when an array constant is evaluated a new array is created;

The definition is based on some semantic maps which assign meaning to syntactic entities These maps models the hierarchy, the inheritance and message passing mechanisms

Let's consider now the following example in Smalltalk. The hierarchy H contains two classes Point and Point1, where Point1 is a subclass of class Point In Point are defined two methods, the first being redefined in Point1 and the second method invokes the first one

```
      class Point
instanceVariablesNames
  ' x y'
method DistFrom Orig
 sqrt(self x^2 + self y^2)
method CloserToOrig(p)=
 (self DistFromOrig < p.DistFromOrig)
      Point superclass  Point1
      method DistFromOrig
 (self x + self y)
```

Let H be the hierarchy containing Point and Point1 For an easy reading, we will denote

m1 = method DistFromOrig

m2 = method CloserToOrig and

R = C[H]

In this example D[H] = YR = sup{ ⊥, R⊥, R(R⊥), .}

For a complete understanding of the example we shall recall the notations used in the specification model Kamin has defined some semantic maps to specify the behavior of the object oriented mechanisms Inheritance is modeled by the two semantic maps C and D

D Hier --> Env
C Hier --> Env --> Env

D[H] = **Y(C[H])**
C[H]ρ
= λ<c,m>
let H(c) = C S w x F
in if F(m) = no-def **then** ρ<S,m> **else M[**F(m)**]**ρ

where **C[H]** defines an application from the environment (meaning of the hierarchy H) to the environment (noted Env) which executes an "inheritance step" For example, if H is a hierarchy containing the class Point1 and it's superclass Point, m2 is an attribute not defined in Point1 and ρ(<Point,m>) is defined, then (**Point1[H]**ρ)(<Point1,m>) will be defined equivalent with ρ(<Point,m>) So, Point1 has "inherited" the definition of m2 from Point **Point1[H]** executes only an inheritance step· if D is a subclass of Point1, which doesn't define the attribute m2, then (**Point1[H]**ρ)(<D,m>) is not defined, but (**Point1[H](Point1[H]**ρ))(<D,m>) is All the inheritances are resolved here

We use ⊥ to denote the primitive routines (e g machine arithmetic).

We will construct some of these environments to understand the inheritance mechanism

R⊥ = { <Point, m1> -> ⊥,
<Point, m2> -> ⊥,
<Point1, m1> -> ⊥,
<Point1, m2> -> ⊥,
<Smallinteger, +> -> . , }

R(R⊥) = { <Point, m1> -> euclidian distance,
<Point, m2> -> if the arguments are from the class
Point then compare the euclidian
distance, else ⊥,
<Point1, m1> -> distance,
<Point1, m2> -> R⊥(<Point,m2>) = ⊥, .}

At first, all the methods are undefined After one step (see R⊥) are defined only those methods which send no messages (like * or +) or invoke primitive methods After two steps (see R(R⊥)), in addition to R⊥, are defined the two versions of method DistFromOrig and the method CloserToOrig only for the class Point After three steps CloserToOrig is defined because it can see the definition of the method DistFromOrig from class Point1 (at this step the method can be applied only for arguments from the Point class - the method is inherited) After four steps Point1 has inherited the complete definition of CloserToOrig (it can be applied for arguments from Point or Point1)

We will transcribe the denotational definition given above in Standard ML

```
val no_methods  Methods = fn m => no_def,

val H0   Hierarchy =
      fn P => (P, "Object", [], [], no_methods),

val psi0 = (fn obj => (simple (intval 0),"Object"),
        fn P => null env),

val Point_methods   Methods =
 fn "m1"=>normal("m1",[],[],literal(intconst 10,10))=>no_def,
 fn "m2"=>normal("m2",[],[], call(self,"m1",inconst 15,15))

   val Point_class   ClassDef =
   ("Point", "Object", [], [], Point_methods)

   val Point1_methods   Methods =
 fn "m1"=>normal("m1",[],[],literal(intconst 20 20))=>no_def,

   val Point1_class   ClassDef =
   ("Point1", "Point", [], [], Point1_methods),

   val H   Hierarchy = H0 mod ("Point" --> Point_class)
                  mod ("Point1" --> Point1_class),

   val prog   Prg = (call(new "Point", "m2", []), H),
```

pp prog psi0,

val prog . Prg = (call(new "Point1", "m2", []), H),

pp prog psi0,

This example illustrates how inheritance works In ML syntax **fn x** represents a lambda abstraction If this program is executed and prog1 is evaluated then it returns (10,10) because m2 representing the method CloserToOrig compare the points(10,10) and (15,15) by the euclidian distance from origin The evaluation of prog2 returns (15,15) because 20+20 > 15+15 (the two points are compare by the distance defined in Point1)

Cook's specification model

Cook's definition [2] is based on three essential aspects related to the inheritance mechanism

- the addition of new methods or replacement of the inherited methods,
- the **self** reference must be redirectionated to access the modified methods,
- the **super** reference must be redirectionated to access the original methods

We will describe this definition using the same example The modifications are expressed as a record, **Point** $\oplus$ **Point1** The new methods from class Point1 are combined with the original methods from the parent class Point, such that the method defined in Point1, in this case **DistFromOrig**, substitutes the corresponding method in class Point

The variable **self** is used to refer to the Point1 version of DistFromOrig and **super** can be used to refer to the Point version of the same method So, the modifications can be expressed as a two arguments function, self and super, and returning the record described

above These functions are called wrappers.

Also the self-reference must be changed in the inherited methods These methods are contained in a function named generator The result is a new class definition, namely a new generator This mechanism is provided by [illegible]

The generator associated with Point is .

$$\begin{aligned}&\text{GenPoint}(x,y) = \lambda\,\text{self}\\&\quad\{\ \text{DistFromOrig} \mapsto\\&\quad\ \text{sqrt}(\text{self}\,x^2 + \text{self}\,y^2),\\&\ \text{CloserToOrig} \mapsto\\&\quad\ \lambda p\ (\text{self DistFromOrig} < p\ \text{DistFromOrig})\}\end{aligned}$$

The wrapper associated with Point1 is

$$\begin{aligned}&\text{Point1Wrapper} = \lambda x,y\ \lambda\text{self}\\&\{\ \text{DistFromOrig} \mapsto\\&\quad(\text{self}\,x + \text{self}\,y)\}\end{aligned}$$

The wrapper application will be

$$\begin{aligned}&\text{Point1Wrapper} \triangleright \text{GenPoint}(x,y) =\\&\lambda x,y\ \lambda\text{self}\\&\{\ \text{DistFromOrig} \mapsto\\&\quad(\text{self}\,x + \text{self}\,y)\\&\ \text{CloserToOrig} \mapsto\\&\quad\lambda p\ (\text{self DistFromOrig} < p\ \text{DistFromOrig})\}\end{aligned}$$

After presenting these two models of specification we shall make some comments The greatest advantage of the Kamin's model is the simple treatment of inheritance The related papers appeared before seems to have some disadvantages Kamin resolved them using fixed points to model inheritance He had defined the semantic maps we have talked a little earlier Indeed, for our example it is a nice specification way But what happens when we have a larger hierarchy? The specification will be sometimes not too easy to be followed. On the other hand we haven't used yet the definition of the E map, which is far more complicated.

The model has it's advantage the specification is concentrated on inheritance and it's mechanism is treated very simple and so it's easy to understand Also, Kamin has described all the mechanisms appeared in an object oriented program the meaning of the hierarchy, the inheritance process, the message passing, the methods evaluations, the evaluation of the primitive methods which provide access to low-level operations

What about Cook's model? This model seems easier to understand maybe because it is provided with an intuitive explanation of inheritance as a mechanism for incremental programming The whole specification is based on this motivation Also, Cook proved the correctness of his model demonstrating that it is equivalent with an operational semantics of inheritance based upon a method-lookup algorithm This way of specifying the inheritance shows that this is not only an object oriented features but a general mechanism that can be applied to any form of recursive definition Although Kamin's model is closely related, he described inheritance as a global operation on programs, which blurs scope issues and inheritance Here is the most important difference between the two models

Kamin's model versus Cook's model

Every specification has to respect some well-known criteria We will discuss how these specifications respect them

Formalization verifies if the specification behaves conforming with the implementation

Kamin's model can be transcribed in an executable version in Standard ML so this criterium is easy to verify We must also notice that the language had suffered some modifications and omissions But Kamin's goal was to specify the mechanism of inheritance

and the missing details are not essential related with this concept

Cook proved that his model is equivalent with an operational semantics and it's obvious that it respects this criterium

Constructability A specification must be easy to construct even if the notation used is formal

The omitted details make Kamin's specification easier to build, but even so if the hierarchy is thick then the construction of the C and D maps seems to be hard to follow

Comprehensibility The specification must be easy to understand The specification given by Kamin seems difficult to understand when we have to deal with the maps E and *E*

Minimality All the non-essential details had been omitted (we have already present the omissions and the modifications of the language, because they are not direct related with the inheritance process)

Applicability From the applicability point of view Cook's model seems to be more interesting since his definition of inheritance, although it was developed first for object oriented languages, shows that, in fact, inheritance is a general mechanism that can be applied to any form of recursive definition

The major problem of object oriented languages is that they lack a solid formal fundamentation There have been some attemps in specifying object oriented features in operational, axiomatic, denotational and algebraic semantics We have focus our attention on denotational semantics because it provides a good mathematical instrument for specification based on lambda-calculus and, on the other hand, an instrument which is not such complicated and hard to understand as the algebraic theory used in algebraic specification techniques This comparative presentation of these two model tries to be a study for choosing

the most suitable formal specification

REFERENCES

1 L Cardelli, P Wegner, On Understanding Types, Data Abstraction and Polymorphism, Computing Surveys, vol 17, no 4, Dec 1985, pg.471-522

2 W Cook, J Palsberg, A Denotational Semantics of Inheritance and its Correctness, OOPSLA'89 Proceedings, 1989, pg 433-444

3 W Cook, W L.Hill, P S Canning, Inheritance is not Subtyping, Proceedings of POPL'90, ACM Press 1990

4 A Goldberg, D Robson, Smalltalk-80 The Language and Its Implementation, Addison-Welsey, Reading, MA, 1983

5 S Kamin, Inheritance in Smalltalk-80 A Denotational Definition, Proceedings of the 15th Annual ACM SIGACT - SIGPLAN Symposium on Principles of Programming Languages, San Diego, Jan 1988, pg 80-87

6 Soo Dong Kim, Formal Specification in Object-Oriented Software Development, Ph D Thesis, The University of Iowa, 1991

STUDIA UNIV BABEŞ-BOLYAI, MATHEMATICA, XXXVIII, 3, 1993

FUNCTIONAL AND RELATIONAL PROGRAMMING WITH PSP

Ilie PARPUCEA* and Bazil PÂRV**

Received February 26, 1994
AMS subject classification 68Q05

REZUMAT. - Programare funcţională şi relaţională cu PSP. Articolul prezintă PSP (Procesorul Simbolic Poisson), într-o manieră ce unifică programarea relaţională cu clauze Horn bazate pe predicate cu programarea funcţională bazată pe egalităţi Unificarea pleacă de la o logică minimală, ce posedă atât clauze Horn cât şi egalităţi, numită logica clauzelor Horn cu egalităţi În ipotezele teoremei Church-Roser, semantica operaţională a PSP constituie o logică completă Semantica se bazează pe unificarea a două abordări. una construită pe baza teoriei modelelor, care foloseşte relaţia de satisfacţie între modele şi instrucţiuni, şi una bazată pe teoria demonstrării, care foloseşte relaţia de partiţionare (entailment) între mulţimi şi instrucţiuni PSP posedă tipuri abstracte de date ce se pot defini de utilizator şi care pot fi considerate module generice (parametrizate) Cu ajutorul subsorturilor se pot introduce operatori polimorfici şi o relaţie de moştenire pe tipurile de date Toate aceste caracteristici concură la definirea riguroasă a semanticii cu ajutorul logicii substrat, ilustrată cu câteva exemple

1. Introduction. A main feature of the processor described in this paper, hereafter called PSP, is the practical way in which it unifies relational programming with functional one, by unifying the logics that underlie relational and functional programming, namely first order Horn clause logic and many-sorted equational logic, to get many-sorted first order Horn clause logic with equality [8] In addition, generic modules are available with a rigorous logical foundation, and PSP also has a subsort facility that greatly increases its expressive power

PSP is intended to operate with Poisson series, which are a well-known tool in

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

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

expressing celestial mechanics problems The motion of celestial bodies is described by means of differential equations, in which the right-hand-side terms are in fact Poisson series Usually, the solution of these differential equations cannot be obtained in exact form There are two alternatives numerical integration or analytic construction of an approximate solution (known as "theory of motion") First was used extensively, being a "classical" solution of motion problems The second alternative seems to be more attractive, because one can obtain the solution in analytical form, which provide a qualitative study of motion There are many analytical methods for constructing the approximate solution of differential equations, most of them known as "perturbation theory" methods [2, 14]

The advantages claimed for PSP includes simplicity, clarity, understandability, reusability and maintanability There is another requirement that we argue also be imposed on our symbolic processor every program should have an initial model [10, 12] An initial model is characterized, uniquely up to isomorphism, by the property that only what is provable is true, and everything else is false The initial model provides a foundation for database manipulations, since you know exactly is true

We have found that neither of the approaches, the model-theoretic and the proof-theoretic one, is by itself sufficient to axiomatize our PSP The model-theoretic approach focuses on the satisfaction relation

$$M \models \gamma$$

between a model M and a sentence γ, and the proof-theoretic one tries to axiomatize the entailment relation

$$\Gamma \vdash \gamma$$

between a set of sentences Γ and a sentence γ derivable from Γ The model-theoretic approach is exemplified by Barwise's axioms for abstract model theory [1] The framework of institutions, given by Goguen and Burstall [6, 7] also belongs to this approach The proof-theoretic approach has a long tradition, dating back to work of Tarski [15] on "consequence relations", and of Hertz and Gentzen on the entailment relation $\vdash$.

This paper proposes a practical approach that integrates the two above-mentioned ones (model-theoretic and proof-theoretic aspects) into a single axiomatization The axiomatization in question consists of an "entailment system", specifying an entailment relation $\vdash$, together with a "satisfaction system" (specifically, an institution in the Goguen-Burstall sense), specifying a satisfaction relation $\models$ [11] The entailment and satisfaction relations are then linked by a soundness axiom

The entailment relation $\vdash$ says nothing about the internal structure of a proof To have a satisfactory account of proofs, we use the additional concept of a proof calculus C for a L The same logic may have, of course, many different proof calculi When we wish to include a specific proof calculus as part of a logic, the resulting logic plus proof calculus is called logical system The axioms for a proof calculus C state that each signature in the logic L has an associated space of proofs, which is an object of an appropriate category. From such a space we can then extract an actual set of proofs supporting a given entailment $\Gamma \vdash \gamma$

In order to obtain some efficiency with respect to PSP, we use the more general concept of proof subcalculus, where proofs are restricted to some given class of axioms and conclusions are also restricted to some given class of sentences It is by systematically exploiting such restrictions that the structure of proofs can be simplified In this way, we can

obtain efficient proof theories, which lead to the theoretical concept of variable operational semantics

2. The features of PSP Conceptual clarity and ease of understanding are facilitated by breaking a program into modules This in turn offers support for debugging and reusability When there are many modules, it is helpful to design the structure of module dependencies in an hierarchical manner Whenever one module (client module) uses data (state) or operations (services) declared in a second one (server module), the server must be explicitly imported to the client and also must be defined earlier in the program text A program obtained in this way has the abstract structure of an acyclic graph with modules as vertices and the module dependencies as edges

A PSP program is a sequence of modules (objects) Each module may define one or more new data sorts, together with associated operations that may create, select, interrogate, store, or modify data Such an module may use existing modules with their sorts of data and operations The module concept includes both data types in the programming language sense (that is, a domain of values of variables together with operations that access or modify those values) and algorithms

PSP has the following syntax for import

<importing> <mod_list>,

where **importing** is keyword and <mod_list> is a list o module names By convention, if a module M imports a module M', that imports a module M", then M" is also imported into M, that is, "importing" is a transitive relation

Usually, programming systems provide a number of built-in data types, for example numbers and identifiers PSP has the following built-in modules BOOL, NAT, INT, and RAT BOOL provides the expected syntax and semantics for Booleans NAT, INT, and RAT define natural, integer and rational numbers (the last ones from the integers)

There is much work on providing user-defined abstract data types in programming languages (e g [3, 4, 9]) The essential idea is to allow users to introduce models that define new sorts and their associated functions and give axioms in Horn clause logic with equality or rules of computation It can also be very helpful to have available subsorts and their associated predicates, as we will see later

Note that PSP keywords are written in **bold**, module names are all CAPITALS, while variable names begin with a capital letter and that relation, function and constant names are all lowercase Attributes can be given for operators, for example, **assoc**, **comm**, and **id** indicate that a binary operator is associative, commutative, and idempotent, respectively

PSP mix-fix notation allows any desired ordering of keywords and arguments for operators, this is declared by giving a syntactic form consisting of a string of keywords and underbar character "_" followed by a " ", followed by the arity as a string of sorts, followed by "->", followed by the value sort of the function Similar conventions are used for predicates An expression is considered well-formed in this scheme iff it has exactly one parse, the parser can interactively help the user to satisfy this condition

PSP operates with Poisson series, which are of the form

$$S = \sum_{i=0}^{\infty} C_i \, y_1^{j_1} y_2^{j_2} \; y_m^{j_m} \begin{matrix} \sin \\ \cos \end{matrix} (k_1 x_1 + k_2 x_2 + \; + k_n x_n),$$

where C_i are numerical coefficients, y_1, y_2, ,y_m are monomial variables, x_1, x_2, ,x_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 t covers the set of all possible combinations of the exponents j and coefficients k ($j \in \mathbf{Z}^m$, $k \in \mathbf{Z}^n$, $\mathbf{Z}$ being the set of integers)

In a concise form we write (1) as follows

$$S = \sum_{t=0}^{\infty} T_t,$$

in which T_t is a term of this series

$$T_t = C_t F_t P_t ,$$

where the polynomial part P_t has the form

$$P_t = y_1^{j_1} y_2^{j_1} \quad y_m^{j_n},$$

while the trigonometric part F_t is

$$F_t = \begin{matrix} \sin \\ \cos \end{matrix} (k_1 x_1 + k_2 x_2 + \quad + k_n x_n)$$

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

$$S = \sum_{t=0}^{N} T_t, \quad N \in N$$

The Poisson expression can be defined in an hierarchical way The complete specification of trigonometric and polynomial part of a Poisson term (Ttr, and Ppol, respectively) can be found in [13] Now we define the Poisson term as following

```
psp  TERM is
       importing Rat Ttr Ppol
       sorts Rat Ttr Ppol Term
       op
            _*_*_  Rat Ttr Ppol -> Term [assoc comm]
       _=_  Term -> Bool
```

```
    vars
        X  Rat
        Y  Ttr
        Z  Ppol
        X·Y·Z X'·Y'·Z'  Term
    eq
        0·Y·Z = 0
        X·0·Z = 0
        X·Y·0 = 0
        1/1·Y·Z = Y·Z
        X·1·Z = X·Z
        X·Y·1 = X·Y
        X·Y·Z = X'·Y'·Z'  - X = X', Y = Y', Z = Z'
endpsp.
```

The above keyword importing indicates that the sorts, subsorts, predicates, functions, and axioms of the listed models are imported into the module being defined The equation

$$X \cdot Y \cdot Z = X' \cdot Y' \cdot Z' \; - X = X', Y = Y', Z = Z'$$

is a Horn clause with equality, where "=" represents equality predicate defined on types, respectively

In the same way, we define EXP, that is based upon TERM, and specify the Poisson expression, viewed as a list of terms, in which the symbol "," is separator

```
psp EXP is
    importing Term
    sorts Term NeExp Exp
    subsorts Term < NeExp < Exp
  op
        _+_  Term Term -> Exp [assoc comm id 0]
    _-_  Term Term -> Exp [id 0]
    _*_  Term Term -> Exp [assoc comm id 1]
        _ _  Exp Exp -> Exp [assoc id nil]
        _==_  Exp Exp -> Bool
    head_  NeExp -> Term
    tail_  NeExp -> Exp
    empty?_  Exp -> Bool
  vars
    T  Term
```

E Exp

$$N/M \cdot \begin{Bmatrix} \sin X_1 \\ \cos X_1 \end{Bmatrix} \cdot Y_1 \quad \text{Term}$$

$$P/Q \cdot \begin{Bmatrix} \sin X_1 \\ \cos X_1 \end{Bmatrix} \cdot Y_1 \quad \text{Term}$$

$$P/Q \cdot \begin{Bmatrix} \sin X_2 \\ \cos X_2 \end{Bmatrix} \cdot Y_2 \quad \text{Term}$$

eq

$$N/M \cdot \begin{Bmatrix} \sin X_1 \\ \cos X_1 \end{Bmatrix} \cdot Y_1 \pm P/Q \cdot \begin{Bmatrix} \sin X_1 \\ \cos X_1 \end{Bmatrix} \cdot Y_1 =$$

$$= (N/M \pm P/Q) \cdot \begin{Bmatrix} \sin X_1 \\ \cos X_1 \end{Bmatrix} \cdot Y_1$$

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

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

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

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

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

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

head(T E) = T
tail(T E) = E
empty?_E = E == nil

endpsp.

In addition, we define two modules for differentiating and integrating of Poisson expressions

psp DERIV **is**
importing Exp
sorts Term Exp NeExp
subsorts Term < NeExp < Exp
op

$$\frac{\partial}{\partial\_}\_ \quad \text{Term Set --> Exp}$$

$$\frac{\partial}{\partial\_}\_ \quad \text{Exp Set --> Exp}$$

vars
E Exp
T Term

$$N/M \cdot \left\{ \begin{matrix} \cos(N_1 \cdot X_1 + N_2 \cdot X_2 + \quad + N_l \cdot X_l) \\ \sin(N_1 \cdot Y_1 + N_2 \cdot Y_2 + \quad + N_l \cdot X_l) \end{matrix} \right\} \cdot Y_1^{M1} \cdot \quad \cdot Y_h^{Mh} \cdot \text{ Term}$$

eq

$$\frac{\partial}{\partial Y_k} (N/M \cdot \left\{ \begin{matrix} \cos(N_1 \cdot X_1 + N_2 \cdot X_2 + \quad + N_k \cdot Y_k + \quad + N_l \cdot X_l) \\ \sin(N_1 \cdot X_1 + N_2 \cdot X_2 + \quad + N_k \cdot Y_k + \quad + N_l \cdot X_l) \end{matrix} \right\} \cdot Y_1^{M1} \cdot \quad \cdot Y_k^{Mk} \cdot \quad \cdot Y_h^{Mh}) =$$

$$= (N{*}M_k/M \cdot \left\{ \begin{matrix} \cos(N_1 \cdot X_1 + N_2 \cdot X_2 + \quad + N_k \cdot Y_k + \quad + N_l \cdot X_l) \\ \sin(N_1 \cdot X_1 + N_2 \cdot X_2 + \quad + N_k \cdot Y_k + \quad + N_l \cdot X_l) \end{matrix} \right\} \cdot Y_1^{M1} \cdot \quad \cdot Y_k^{Mk-1} \cdot \quad \cdot Y_h^{Mh} ,$$

$$\mp N{*}N_k/M \cdot \left\{ \begin{matrix} \sin(N_1 \cdot X_1 + N_2 \cdot X_2 + \quad + N_k \cdot Y_k + \quad + N_l \cdot X_l) \\ \cos(N_1 \cdot X_1 + N_2 \cdot X_2 + \quad + N_k \cdot Y_k + \quad + N_l \cdot X_l) \end{matrix} \right\} \cdot Y_1^{M1} \cdot \quad \cdot Y_k^{Mk} \quad \cdot Y_h^{Mh})$$

$$\frac{\partial}{\partial Y_k}(0) = 0$$

$$\frac{\partial}{\partial Y_k}(\text{nil}) = 0$$

$$\frac{\partial}{\partial Y_k}(E) = \frac{\partial}{\partial Y_k}(\text{head } E) + \frac{\partial}{\partial Y_k}(\text{tail } E)$$

endpsp.

In the specification of INTEG module given below, we use the following abreviations

$$I_p = \int N/M \cdot \sin(N_1 \cdot X_1 + N_2 \cdot X_2 + \ldots + N_k \cdot Y_k + \ldots + N_l \cdot X_l) \cdot Y_1^{M_1} \cdot \ldots \cdot Y_{k-1}^{M_{k-1}} \cdot Y_k^{p} \cdot Y_{k+1}^{M_{k+1}} \cdot \ldots \cdot Y_h^{M_h} \, dY_k ,$$

and

$$J_p = \int N/M \cdot \cos(N_1 \cdot X_1 + N_2 \cdot X_2 + \ldots + N_k \cdot Y_k + \ldots + N_l \cdot X_l) \cdot Y_1^{M_1} \cdot \ldots \cdot Y_{k-1}^{M_{k-1}} \cdot Y_k^{p} \cdot Y_{k+1}^{M_{k+1}} \cdot \ldots \cdot Y_h^{M_h} \, dY_k ,$$

where p Int, p≠-1 (the case p=-1 does not preserve the form of Poisson expressions, because the integration leads to logarithms)

```
psp INTEG is
    importing Exp
    sorts Term Exp NeExp
    subsorts Term < NeExp < Exp
    op
        ∫_d_ Term Set --> Exp
        ∫_d_ Exp Set --> Exp
    vars
        E Exp
        T Term
    P Nat
    N/M · sin(N1·X1+N2·X2+ .+Nl·Xl)·Y1^M1· ·Yh^Mh Term
    N/M · cos(N1·X1+N2·X2+ +Nl·Xl)·Y1^M1· ·Yh^Mh Term
  eq
    I0 = (-1/Nk*N/M) · cos(N1·X1+N2·X2+ +Nk·Yk+ +Nl·Xl)·
         M1      Mk-1  0   Mk+1       Mh
        ·Y1 · Yk-1 ·Yk ·Yk+1 · ·Yh

    I1 = ((-1/Nk*N/M) · cos(N1·X1+N2·X2+ +Nk·Yk+ +Nl·Xl)·
         M1      Mk-1  1   Mk+1       Mh
        ·Y1 · Yk-1 ·Yk ·Yk+1 · ·Yh ,

        (1/(Nk*Nk)) · sin(N1·X1+N2·X2+ +Nk·Yk+ +Nl·Xl)·
         M1      Mk-1  0   Mk+1       Mh
        ·Y1 · Yk-1 ·Yk ·Yk+1 · ·Yh )

    Ip = ((-1/Nk*N/M) · cos(N1·X1+N2·X2+ +Nk·Yk+ +Nl·Xl)·
```

$$\cdot Y_1^{M_1} \cdot \ldots \cdot Y_{k-1}^{M_{k-1}} \cdot Y_k^{p} \cdot Y_{k+1}^{M_{k+1}} \cdot \ldots \cdot Y_h^{M_h},$$

$$(N/M^*{}_p/(N_k{}^*N_k))\cdot \sin(N_1\cdot X_1+N_2\cdot X_2+ \ldots +N_k\cdot Y_k+ \ldots +N_l\cdot X_l)\cdot$$

$$\cdot Y_1^{M_1} \cdot \ldots \cdot Y_{k-1}^{M_{k-1}} \cdot Y_k^{p-1} \cdot Y_{k+1}^{M_{k+1}} \cdot \ldots \cdot Y_h^{M_h},$$

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

$$J_0 = (1/N_k{}^*N/M) \cdot \sin(N_1\cdot X_1+N_2\cdot X_2+ \ldots +N_k\cdot Y_k+ \ldots +N_l\cdot X_l)\cdot$$

$$\cdot Y_1^{M_1} \cdot \ldots \cdot Y_{k-1}^{M_{k-1}} \cdot Y_k^{0} \cdot Y_{k+1}^{M_{k+1}} \cdot \ldots \cdot Y_h^{M_h}$$

$$J_1 = ((1/N_k{}^*N/M) \cdot \sin(N_1\cdot X_1+N_2\cdot X_2+ \ldots +N_k\cdot Y_k+ \ldots +N_l\cdot X_l)\cdot$$

$$\cdot Y_1^{M_1} \cdot \ldots \cdot Y_{k-1}^{M_{k-1}} \cdot Y_k^{1} \cdot Y_{k+1}^{M_{k+1}} \cdot \ldots \cdot Y_h^{M_h},$$

$$(1/(N_k{}^*N_k)) \cdot \cos(N_1\cdot X_1+N_2\cdot X_2+ \ldots +N_k\cdot Y_k+ \ldots +N_l\cdot X_l)\cdot$$

$$\cdot Y_1^{M_1} \cdot \ldots \cdot Y_{k-1}^{M_{k-1}} \cdot Y_k^{0} \cdot Y_{k+1}^{M_{k+1}} \cdot \ldots \cdot Y_h^{M_h})$$

$$J_p = ((1/N_k{}^*N/M) \cdot \sin(N_1\cdot X_1+N_2\cdot X_2+ \ldots +N_k\cdot Y_k+ \ldots +N_l\cdot X_l)\cdot$$

$$\cdot Y_1^{M_1} \cdot \ldots \cdot Y_{k-1}^{M_{k-1}} \cdot Y_k^{p} \cdot Y_{k+1}^{M_{k+1}} \cdot \ldots \cdot Y_h^{M_h},$$

$$(N/M^*p/(N_k{}^*N_k))\cdot \cos(N_1\cdot X_1+N_2\cdot X_2+ \ldots +N_k\cdot Y_k+ \ldots +N_l\cdot X_l)\cdot$$

$$\cdot Y_1^{M_1} \cdot \ldots \cdot Y_{k-1}^{M_{k-1}} \cdot Y_k^{p-1} \cdot Y_{k+1}^{M_{k+1}} \cdot \ldots \cdot Y_h^{M_h},$$

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

$$\int 0 \, dX_k = 0$$

$$\int (\mathrm{nil}) \, dX_k = 0$$

$$\int E \, dX_k = \int \mathrm{head}(E) \, dX_k + \int \mathrm{tail}(E) \, dX_k$$

endpsp.

The NORMAL module provides a normal form of Poisson expressions

```
psp NORMAL is
importing Exp
sorts Term Exp
subsorts Term < Exp
op
  normalised_  Exp -> Bool
  normalising_  Exp -> Exp
vars
  T T'  Term
  E E'  Exp
eq
  normalised(nil) = True
  normalised(T) = True
  normalised(2T E)  - T = T', normalised(T' E)
  normalising(E T T' E') = normalising(E 2T E')  - T=T'
  normalising(E) = E  - normalised(E)
endpsp
```

The basic building blocks of parameterized programming are parameterized modules Parameterized programming is a powerful technique for the reliable reuse of software In this technique, modules are parameterized over very general interfaces that describe what properties of an environment are required for the module to work correctly.

Here is an example of a parameterized module, intitulated SUBST, over the theories SORT1 and SORT2 In our example, SUBST module provides the symbolic substitution operation

```
psp SUBST [S1   SORT1 , S2   SORT2] is
    importing Exp
    sorts Term Exp NeExp
    subsorts Term < NeExp < Exp
    op
      _sub_ _  Term S1 S2 -> Term
      -sub_ _  Exp S1 S2 -> Exp
    vars
     E  Exp
     T  Term
     X  S1
     Y  S2
```

```
    eq
      sub(0 X Y) = 0
            sub(nil X Y) = nil
            sub(T X Y) = sub(T X -> Y)
            sub(E X Y) = sub(head E X Y) + sub(tail E X Y)
endpsp.
```

where the meaning of expression sub(T X -> Y) is all occurences of the symbol X are replaced by the symbol Y in term T SORT1 and SORT2 are theories defined as follows

```
Th SORT1 is                    Th SORT2 is
  sorts Sor1                   sorts Sor2
endth.                         endth.
```

The following specification

```
view SUBS is  (Sor1 as Rat
               Sor2 as Set)
```

define a view called SUBS, mapping from the sorts of SORT1 and SORT2 to the other sorts already defined, that preserves the subsort relation, and a mapping from the operations of SORT1 and SORT2 to the operations of Rat and Set, preserving arity, value sort, and attributes

To actually use a parameterized module, it is necessary to instantiate it with an actual parameter The **Make** command applies a parameterized module to an actual one, by use of a **view** For example,

```
Make SUBSTITUTION is  SUBST[SUBS] endm.
```

uses the view SUBS to instantiate the parameterized module SUBST with the actual parameters Rat and Set

In the same way, one can construct new PSP modules, which implements new operations on Poisson series, like power expansion (including exponents integer numbers or

rational numbers of the form 1/M or M/2 with M nonzero integer), inverse of a Poisson series, binomial expansion and so on (see, for example, [2]) Also, on the basis of PSP we can realize new specialized modules, like Kepler or Taylor ones In Keplerian module, for example, the polynomial and trigonometric variables are the well-known elliptic elements For these elements, there are transformation rules, which can be considered, from our point of view, as rewriting rules The next level of abstraction consists of modules for constructing the approximate solution of differential equations up to an desired order One can construct different modules for each "perturbation method", each of them using operations defined in previous modules Using different methods applied to the same problem, one can compare the obtaining solutions, keeping in mind the fact that many of methods are assimptotically equivalent This can be another facility of theorem proving of PSP

3. Concluding remarks PSP is intended to be a symbolic processor, with features of theorem proving, dedicated to the study of the motion of celestial bodies From the implementation point of view, there are some modules that are not so efficient, this difficulty remains to be considered later Taking into account the built-in abstract data types, the denotational semantics of initial models, the operational semantics based on rewriting rules, PSP, considered as open system, can be helpful in other fields, too

REFERENCES

1 Barwise,K J Axioms for Abstract Model Theory Ann.Math Logic, 7, pp 221-265, 1974

2 Brumberg,V A Analytic Algorithms of Celestial Mechanics, Nauka, Moskow, 1980 (russ)

3 Futatsugi,K ,Goguen,J ,Jouannaud,J P ,Meseguer,J Princi-ples of OBJ2 In Proc 1985 Symp on Principles of Programming Languages, ACM, pp 52-66, 1985

4 Goguen,J Parameterized Programming Tech Rep CSLI-84-9, 1984

5 Goguen,J et al Introducting OBJ Oxford University Draft of January, 1993

6 Goguen,J ,Burstall,R Introducing institutions In E Clarke and D Kozen (eds), Logics of Programs, Springer, LNCS vol 164, pp 221-256, 1984

7 Goguen,J ,Burstall,R Institutions Abstract Model Theory for Computer Science Tech Rep CSLI-85-30, Stanford University, 1985

8 Goguen,J ,Meseguer,J Models and Equality for Logical Programming In Proc TAPSOFT87, Springer, LNCS vol 250, pp 1-22, 1987

9 Goguen,J ,Tardo,J An Introduction to OBJ A Language for Writing and Testing Software Specifications In Specifications of Reliable Software, IEEE Press, pp 170-189, 1979

10 Goguen,J ,Thatcher J ,Wagner,E An Initial Algebra Approach to the Specification, Correctness and Implementation of Abstract Data Types, Tech Rep RC 6487, IBM Watson Research Center, 1976

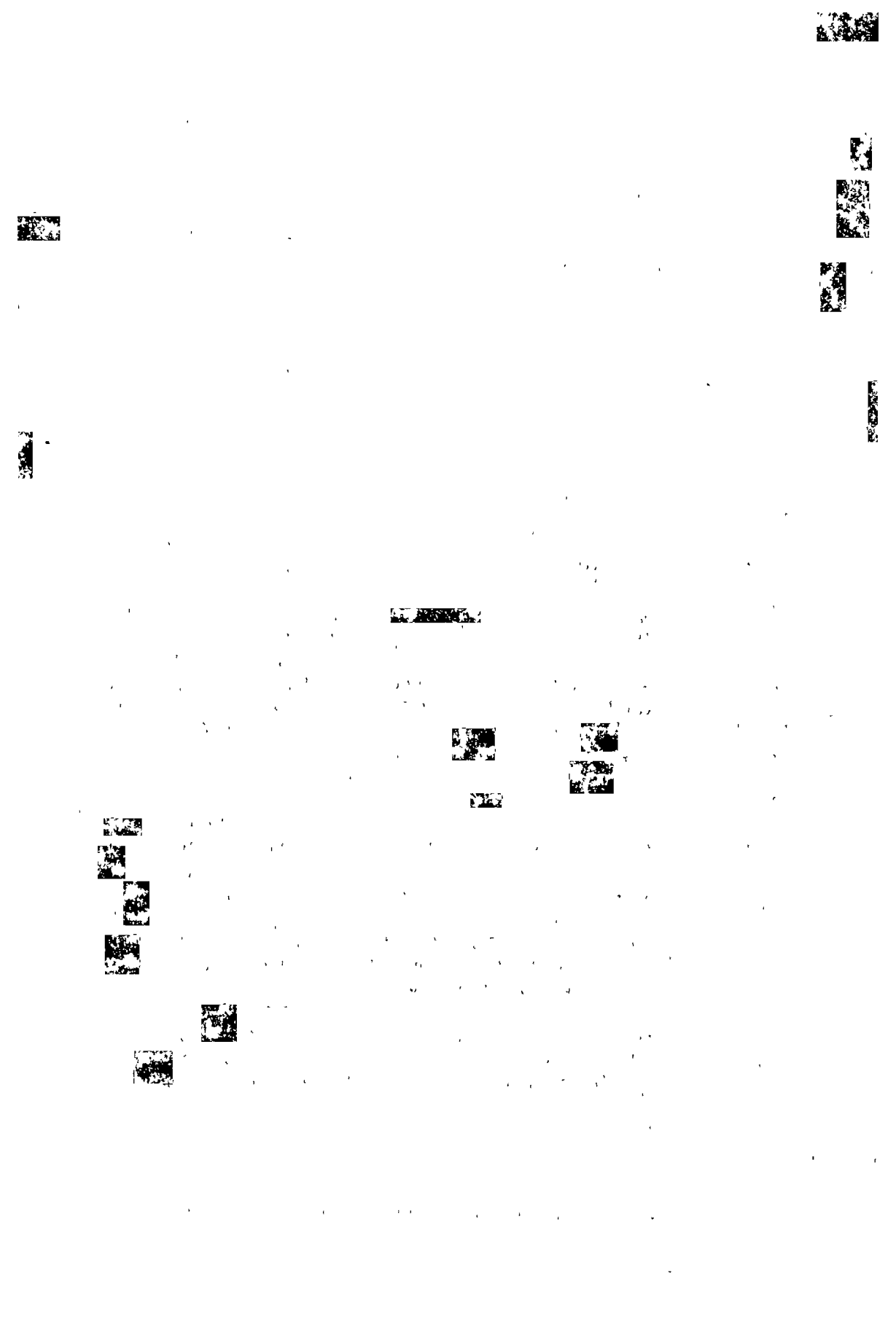
11 Meseguer,J General Logics H-D Ebbinghaus et al (eds), Elsevier B V , 1989

12 Meseguer,J ,Goguen,J Initiality, Induction and Computabi-lity, in Algebraic Methods in Semantics (M Nivat and J C Reynolds eds), Cambridge University Press, 1985

13 Parpucea,I ,Pârv,B Algebraic Specification of PSP, Studia, XXXVII, No 3, pp 99-110, 1992

14 Rand,R ,Armbruster,D Perturbation Methods, Bifurcation Theory and Computer Algebra, Springer, 1987

15 Tarski,A One Some Fundamental Concepts of Metamathematics In Logic, Semantics, Metamathematics, Oxford University Press, pp 30-37, 1956



STUDIA UNIV BABEŞ-BOLYAI, MATHEMATICA, XXXVIII, 3, 1993

A MATHEMATICAL MODEL TO SOLVE THE TIMETABLE PROBLEM USING PROLOG

Dragoş POP*

Dedicated to Professor Emil Muntean on his 60th anniversary

Received January 12, 1994
AMS subject classification 68Q40

REZUMAT. - Un model matematic pentru rezolvarea problemei orarului utilizînd limbajul Prolog. Lucrarea prezintă un model matematic general al problemei orarului precum şi utilizarea acestuia de către un algoritm de rezolvare a problemei Modelul matematic propus asigură descrierea unor restricţii extrem de diverse ce caracterizează soluţiile fiabile Algoritmul propus asigură găsirea soluţiei optime din punct de vedere al mai multor criterii, construind numai soluţiile fiabile susceptibile de a fi soluţii optime

1. Introduction. Timetable problems are by their fundamental nature resource allocation problems, whose solutions represent activity plans Every activity (also called 'meet') needs certain available resources (persons, development places, time, etc) and different conditions for development, depending on the activity itself or other activities (avoiding certain times, activities sequences and certain parallel activity pairs)

These problems depend on the educational systems and they can be mutually very different Some practical requirements cannot be easily caught in mathematical formulas Therefore it is very difficult to model and solve these problems For this reason a model should be universal and very flexible

Timetable problems are known to be **NP** - complete, only some reduced problems are

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

polynomial In our situation, such a special case requires the permanent availability of lecturers and rooms

In this paper we are concentrating on timetables for university faculty and propose a PROLOG rule system which can be run on a microcomputer The mathematical model is extended such that a priori fixed assignements, faculty connections and other miscellaneous constraints are supported

2. Problem formulation Let's assume given the set of faculty teaching staff, the set of student groups (called classes), the set of available rooms and their sizes and the set of activities that have to be scheduled in a certain week

Furthermore, a maximal allowed number of time periods (called hours) is assigned to each day The lunch breaks the day into two daily ammounts of consecutive hours (called daily quantums) with a given extent An activity should not be interrupted by this break

Every professor and class should have a set of unavailable hours due to a priori scheduled activities or other reasons

For the feasibility, the timetable have to satisfy certain requirements The set of all the requirements of the problem is partitioned into three groups, corresponding to the various degrees of strictness ([1])

- hard requirements, which must always be satisfied
- medium requirements, which should be satisfied although they can be relaxed in some cases
- soft requirements, which should be satisfied if the other requirements allow this

The hard requirements should lead to physically feasible timetable and they are expressed as constraints In our case, conflicts due to activities taking place simultaneously but involving classes or professors in common, have to be avoided Other constraints arise from the fact that the activities should not be interrupted and they should not be scheduled in an inadequate room or at hours which are unavailable for one of its participants Also, all the activities have to be scheduled during the time span of one week

The soft requirements deal with preferences and they are modelled as objectives Our objectives are that the timetables for classes and professors are compact, with no time windows, with as many as possible morning courses and other pedagogical recommendations

Every medium requirement can appear either as a constraint or as an objective, depending on the nature or the interpretation of the problem As a medium requirement, two courses of the same topic (called equivalent courses) should not be scheduled in the same day

The objectives are the actual parameters for a given quality function which reflects the preference for one timetable solution over the other It is often difficult to quantify the desirability aspect of timetabling However, a weighted sum of the objectives is a satisfactory, flexible and simple solution and for these reasons we chose it

On these conditions, the problem to solve is to find the feasible optimal solutions for the timetable If there results several optimal solutions, then a decision maker will choose the preferred solution

3. The mathematical model In this section it will be presented a mathematical model for the timetable problem based on the situation in our University, but the model is flexible

enough to allow application to many different situations

We consider the following sets

G - the set of classes

D - the set of teaching staff

P=GUD - the set of participants in teaching activities

S - the set of available rooms

M - the set of teaching activities (courses, seminars, practical training, etc)

H - the set of available hours per week For example, if we have 10 available hours per day, we consider that Monday is represented by the {1 10} hours, Tuesday is represented by the {11 20} hours and so on This convention can be changed in order to satisfy certain interests

and the constants

hdpɛN* - the number of hours per day

mhɛN* - the number of hours of the first daily quantum

(morning hours)

To solve in a smart way the problems of the lunch pauses and the end of a day, we add an hour between the two daily quantums and an hour after each day Then we will mark all classes and professors as unavailable at these fictitious hours Therefore, we avoid the interruption of the activity

Now we can define the time window as the free time period between two occupied time periods, which does not contain fictitious hours

For an easy handle, the rooms and the activities are coded with positive integer numbers Throughout this paper, the activity lists are ordered ascendingly on the codes and the room lists are ordered ascendingly on the room capacity

The following functions give us information about these sets

u $G \to \mathbf{N}^*$ $u(g)$ = the number of students in the group g

v $P \to P(H)$ $v(p)$ = the set of hours when p is available

c $M \to P(S)$ $c(s)$ = the set of suitable rooms for the activity m

p $M \to P(P)$ $p(m)$ = the set of participants in the activity m

r $M \to \mathbf{N}^*$ $r(m)$ = the extent of the activity m

From this information we can determine the function

a $M \to P(H)$ $a(m)$ = the set of hours when the activity m could begin

This function can be calculated using the following formula

$a(m) = \{y \in V \mid y, \ldots, y+r(m)-1 \in V\}$ where

$V = \bigcap v(x)$ represents the hours when all the $x \in p(m)$ participants at the activity m are free

In these conditions we consider a timetable solution as a function $t: M \to H \times S$, $t=(t_1,t_2)$ with the following properties

i) $\{t_1(m), \ldots, t_1(m)+r(m)-1\} \subseteq a(m) \quad \forall m \in M$

ii) $t_2(m) \in c(m) \quad \forall m \in M$

iii) $m \neq n$ and $\{t_1(m), \ldots, t_1(m)+r(m)-1\} \cap \{t_1(n), \ldots, t_1(n)+r(n)-1\} \neq \emptyset \Rightarrow p(m) \cap p(n) = \emptyset$ and $t_2(m) \neq t_2(n) \quad \forall m,n \in M$

(The simultaneous activities must have different participants and must be

scheduled in different rooms)

iv) $m \neq n$ and $p(m)=p(n) \Rightarrow [t_1(m)/hdp] \neq [t_1(n)/hdp]$

(The equivalent activities should not be scheduled in the same day)

Due to the relations between the activities, their different lengths and the problems concerning the rooms which may appear in the general case, the set of activities which could be scheduled at a certain hour depends on the activities scheduled before Therefore we are forced to construct every possible solution, hour by hour, in an heuristic way and then to evaluate its quality Note that there exist papers on the optimal timetable construction in a deterministic way using Operations Research, but they solve the problem only in particular cases ([4])

We understand now that, in the general case, since the timetable problem is **NP**-complete the number of solutions we have to construct is extremely high and it has an exponential growth in the number of participants ([2])

As an interesting particular situation, if all the activities have the same lengths and there are no problems with the rooms, we can consider the graph with the vertex set $V = M \cup H$ and with edges among every different hours, among the activities which have common participants and among the activities and their unsuitable hours In this case, the timetable construction problem is equivalent to the graph colouring problem, with $|H|$ colours

Also, we are very interested in the reduction of the number of constructed solutions There are two posibilities

The first is almost obvious If we evaluate the solution quality during the construction of the solution, we can check at certain moments (at the end of a day, for example) if we are

able to reach a quality which has already been obtained If we don't, it is useless to continue the construction of that solution

With the second method we obtain an extremely important reduction of the number of constructed solutions, with a very low risk of loosing high quality solutions Let's suppose we are constructing the hour h of the timetable and at this hour we can schedule activities from the A set Actually, due to restrictions, we can schedule only subsets of A Let E_h be the set of all these subsets, $E_h \subseteq P(A)$ The idea is to construct only the timetable solutions which have scheduled at the hour h the maximal elements of $(E_h, \subseteq)$

For example, let $\{m_1, \ldots, m_k\}$ be a maximal element of E_h (the activities $m_1, \ldots, m_k$ may be scheduled to begin together at the hour h) A full heuristic algorithm will try to construct solutions with all the following activity sets

$$\emptyset, \{m_1\}, \ldots, \{m_k\}, \{m_1, m_2\}, \ldots, \{m_1, \ldots, m_k\}$$

scheduled to begin at the hour h This means 2^k alternatives

But scheduling only a subset of $\{m_1, \ldots, m_k\}$ implies that some professors and classes will have empty places in their timetable and this fact will decrease the quality of the solution Therefore we can try only with the maximal configuration and the risk to loose this way an optimal solution, is very low

This method also decreases the number of remaining activities and thus it reduces the complexity of the following stages

4. Using the PROLOG programming techniques. The structure of the program

We restrict the presentation and explanation of the program to those elements which

are indispensable for a global comprehension of its ideas and an insight of its structure

The information about faculty teachers, activities, rooms and classes are stored in an internal database

The first module is an initialisation module which creates the information database using the consult predicate to read a given text file containing the input data Then it creates a work database which contains the constraints deduced from the information database These constraints concern the activity pairs that could not be scheduled simultaneously due to theirs common participants, the hours that are inadequate for a certain activity due to the activity length and the suitable rooms for every activity

The target of the main module is the heuristic search of feasible timetable considering all the restrictions, including the one which deals with the size of the rooms and the auxiliary support The best solutions achieved up to the present are stored in a distinct database and their quality is compared with the quality of the latest constructed solution If their quality is equal then the new solution is attached to the database Else, if the quality of the new solution is better, then the solutions stored in the database are removed and the new solution is attached to the database Therefore, at the end of the construction process, we have only the best solutions If the result database contains several solutions, we can apply other constraints concerning pedagogical requirements such as rational distribution of courses and effort during the week or other preferences

The last module is a tool which allows an interactive refinement of the solution The user can choose the preferred solution, if there are several optimal solutions from the given criteria

point of view, and then he may introduce the a priori assignments This module also assists the user to change some assignments preserving the validity of the solution

At the user's choice, this module calls the output module which sends the results to the screen, printer or a given file Among the results there is a new database which represents the updated input database with the new situation of the participants ocupation This database can be useful for unexpected situations which may occur during the semester, and to connect faculties This is an important fact because in general, the professors from a certain faculty assure the majority of courses in the University, which are related to the faculty realm

From all the facts mentioned until now, it results that the algorithm that I propose is semi-heuristic and that it is based on the backtracking mechanism Since the timetable construction problem is suitable for the descriptive programming and the backtracking mechanism is an internal mechanism of the PROLOG language, it becomes clear that the programming effort was considerably reduced this way ([3])

We will present now the predicate for the timetable hours construction which ensures the optimization described above and the predicate for the timetable construction, in a PROLOG-like pseudocode language

constructhour(h,m,A,S,[y|L]) if

$\exists y \in A$, $\exists s_h \in S$ a suitable room for the activity y which is

free between the hours h and h+r(m),

!,

Select $y \in A$, y>m, for which exist suitable rooms where it should be scheduled to begin at the hour h and

let r_y be the most suitable from them,

A1 = A \ l(y),

S1 = S \ {r_y},

constructhour(h,y,A1,S1,L)

constructhour(_,_,_,_,[])

where mεM

A is the ascendingly ordered list of nonscheduled activities

l M -> P(M), l(m) represents the set of activities which have common participants with m,

$$l(m) = \{n \varepsilon M \mid p(m) \cap p(n) \neq \varnothing\}$$

ttconstruct(_,[],_,[],_,[]) if l

ttconstruct(h,A,R,Eq,Ocr,[L|Tt]) if h ≤ the number of hours per week,

Construct Act - the list of activities from A which might be scheduled to begin at the hour h,

Construct AvR - the list of rooms from R which are available at the hour h,

constructhour(h,0,Act,Avr,L),

Create the A1, R1, Eq1, Ocr1 lists as the A, R, Eq and Ocr updated lists,

h1 = h+1,

ttconstruct(h1,A1,R1 Eq1,Ocr1,Tt)

where, in addition to the above described lists

R is the list of available rooms at the hour h

Ocr is the list of occupied rooms at the hour h

(RUOcr is the constant set of all rooms)

Eq is the activity list which should not be scheduled in the present day due to the constraint concerning the equivalent activities

Tt is the constructed solution

Note that these lines must be understood with the PROLOG conventions, i e if an assertion fails, the program will automatically try to find other solutions for the previous assertions in the same clause or if this is impossible, with the following clauses of the predicate

This predicate respects the above considerations on the maximality At a certain moment, there is selected $F=\{m_1, \ , m_i\}$, $F \varepsilon E_h$, where $m_1 < m_2 < \ < m_i$

If $\exists x \varepsilon A \backslash F$, $x>m_i$ such that $FU\{x\} \varepsilon E_h$, then x will be attached to F and the searching process continues Otherwise there are two possible situations

1 If $\nexists x \varepsilon A \backslash F$ such that $FU\{x\} \varepsilon E_h$, then F is a maximal element of $(E_h, \subseteq)$ and the activities from F will be scheduled at the hour h In this case the first clause of the predicate fails and it will be used the second clause

2 If $\exists x \varepsilon A \backslash F$ such that $FU\{x\} \varepsilon E_h$ but $x<m_i$, then F is not a maximal element of E_h and due to the ascending order selection of the activities, $FU\{x\}$ was selected in a previous stage The predicate fails due to the cut backtracking (!) predicate call

5. Concluding remarks and possible extensions Now we can notice the tremendous advantage of considering a relation oriented approach using logical programming techniques The PROLOG programming language is a very fast and flexible prototyping tool Additional

restrictions and situations can be easily included as new predicates or by extension of the existing attribute list.

If this database is too large to fit in the memory, we can use the PROLOG facilities to work with external databases stored on disk

I mention here that I used Borland's Turbo Prolog 2 0. Certainly a PROLOG compiler does not generate very fast programs but Turbo Prolog allows interfacing PROLOG programs with modules written in C or even assembler language for the speed critical parts of the program. Also, knowing that menus and user interface could be programmed more efficient with traditional programming languages, we can use C libraries to do this

I also suggest two useful extensions for this model

- the introduction of the predecessor-succesor relation between the activities as new constraints.
- the introduction of dynamic topics, when we dispose only of the total amount of hours for a certain subject matter, so that we can choose their activity grouping

REFERENCES

1 Eiselt H A , Laporte G , Combinatorial optimization problems with soft and hard requirements, Journal of Operations Research Society 38, 1987

2 Even S , Ilai A , Shamir A , On the complexity of timetable and multicommodity flow problems, SIAM Journal of Computation 5, 1976

3 Hertz A , de Werra D , The Tabu search methaheuristic how we used it, Ann. Math. Artificial Intelligence 1, 1990

4 Mihoc G , Balas E , The problem of optimal timetables, Revue Roumaine de Mathematiques Pures et Appliquees 10, 1965

5 Schmidt G , Strohlein T , Timetable Construction - An annotated bibliography, Computer Journal 23, 1980

STUDIA UNIV BABEŞ-BOLYAI, MATHEMATICA, XXXVIII, 3, 1993

GENERATING FRACTALS OF REGULAR FORM BY PICTURE LANGUAGES

Vasile PREJMEREAN*, Simona MOTOGNA*, Vasile CIOBAN*

Dedicated to Professor Emil Muntean on his 60th anniversary

Received January 31, 1994
MS Subject Classification 68U05, 68U10, 68Q25, 68T10

Rezumat. Generarea fractalilor de formă regulată prin limbaje picturale În această lucrare este prezentată o modalitate de generare a fractalilor de formă regulată, utilizând şiruri de comenzi pentru desenarea acestora Aceste şiruri de comenzi sunt generate prin funcţii care permit dezvoltarea, prelucrarea şi recunoaşterea acestor variaţiuni geometrice numite fractali

The fractal textures are more and more often used in computer graphics since they can model properly 3D-figures and natural forms and they have the advantage of representing the models on a plane surface Fractals can have regular forms, based on repetition of a motif (primary detail) or randomizing forms, which are defined probabilistic They can be built starting from curves, surfaces or figures and they are defined either by a function or by a construction rule [2]

Regular form fractals can be generated using a language of commands for drawings (images) [1] Given a set of graphical primitives (corresponding to a set of drawing commands), from which one or more starting primitives (primary detail) are chosen, we will apply a transformation to this initial set, then to the obtained set of primitives we will apply again the same transformation and so on, as many times as we want Finally, the resulting set

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

of primitives represents the desired fractal, respectively if the result is a string of commands then the fractal will be obtained executing these commands

Let's consider the set $C=\{c_1,c_2, \ ,c_p\}$, containing the graphical primitives which form a fractal family These primitives can be drawn by corresponding graphical commands So, we can achieve a drawing executing a string of such commands

Let $S = \{ x_1x_2 \ x_m / m>0, x_1,x_2, \ ,x_m \in C \}$ be the set of command strings obtained by concatenation of the elements from the set C **A family of variable fractals of regular type** is a set of fractals which have been obtained starting from a primary detail $D \in S$, which is developed step by step according to a transformation rule f C->S, rule which is applied to every graphical primitive (command) This transformation gives the development rule for a primitive and is specific to every fractals family

The development function which allows a fractal to be transformed entirely with an iteration is

$$t \ S \to S, \ t(x_1x_2 \ x_m) = \begin{cases} f(x_1), \text{ if } m=1 \\ f(x_1)f(x_2) \ f(x_m), \text{ if } m>1 \end{cases}$$

(where uv represents the concatenation of u and v)

The development of a fractal after n iterations (n "years") is described by the function

$$t^n \ S \to S \ \ t^n(X) = \begin{cases} t(X), \text{ if } n=1 \\ t^{n-1}(t(X)), \text{ if } n>1 \end{cases}$$

We can say that $t^n(D)$ returns the commands string representing the fractal in the n-th step of the development This observation makes us think that we can define a fractals family specifying the primitives set C, the primary detail D, and the development rule f

$F_n=(C,D,f), n=1,2,$

The definition of the transformation rule f will use the following function

$$\text{Next } C \to C,\ \text{Next}(c_i) = \begin{cases} c_{i+1}, & \text{if } i>p \\ c_1, & \text{if } i=p \end{cases} \quad (\ C = \{c_1, c_2, \ , c_p\}\ )$$

Since this function is bijective, we can construct the reverse function, which will simplify the definition of the function f

$$\text{Next}^{-1}\ C \to C, \quad \text{Next}^{-1}(c_i) = \begin{cases} c_{i-1}, & \text{if } i>1 \\ c_p, & \text{if } i=1 \end{cases}$$

Example 1 Let's consider the following set C={r,u,l,d} where

(r,u,l,d) is (→,↑,←,↓) [3]

and $f(x) = x\ \text{Next}(x)\ x\ \text{Next}^{-1}(x)\ x\ , \quad \forall\ x \in C$

(x concatenated with Next(x), concatenated with x,) and D=r

Then $t^1(D)$, $t^2(D)$, $t^3(D)$, represent the following family

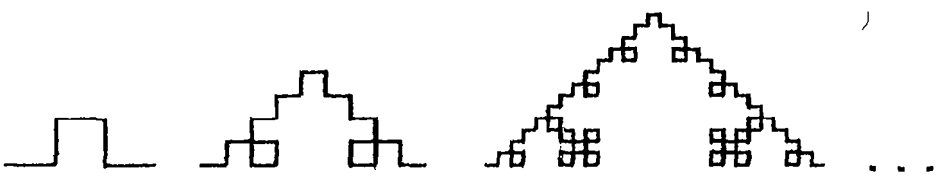


Figure 1

Either in constructing or in recognizing a fractals family defined in the way we described, the following property of the development function is very useful

$$t^n(XY) = t^n(X)\ t^n(Y)$$

The above property can be proved by complete induction as follows

for n=1 $\quad t^1(XY) = t(XY) = t(x_1x_2\ \ x_{m1}y_1y_2\ \ y_{m2}) =$

$= f(x_1)f(x_2)\ \ f(x_{m1})\ f(y_1)f(y_2)\ \ f(y_{m2}) =$

$= t(x_1x_2\ \ x_{m1})\ t(y_1y_2\ \ y_{m2}) = t(X)t(Y)$,

assuming that $\quad t^{n-1}(XY) = t^{n-1}(X)\ t^{n-1}(Y)$

then $t^n(XY) = t^{n-1}(t(XY)) = t^{n-1}(t(X)t(Y)) = t^{n-1}(t(X))t^{n-1}(t(Y))=$

$= t^n(X)t^n(Y)$

This property allows us to draw a fractal succesively on sections and also to analyze a fractal reducing it at its subfractals which compounds it

We consider now another two examples

Example 2 Let C={r,e,u,f,l,g,d,h} be the set with

(r,e,u,f,l,g,d,h) = (→,↗,↑,↖,←,↙,↓,↘) ,

and f(x) = x Next(x) $Next^{-1}$(x) x , and D=urdl

Then $t^1(D)$, $t^2(D)$, $t^3(D)$, represent the following family

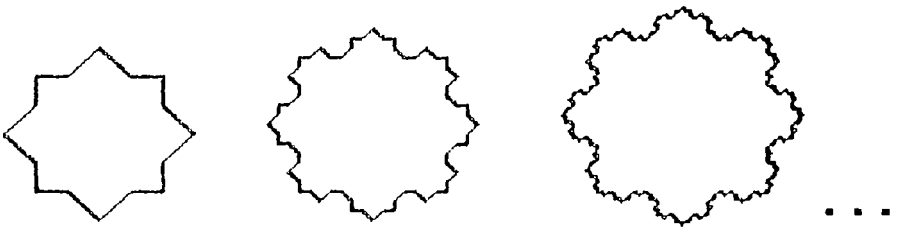


Figure 2

Example 3 Let C={a,b,c} be the set with

(a,b,c) = (↗,↖,←) ,

and f(x) = x Next(x) $Next^{-1}$(x) x x , and D=abc

Then $t^1(D)$, $t^2(D)$, $t^3(D)$, generate the family

The set C might contain even compound primitives (2D-figures or 3D-figures), and in this situation when we represent the fractals we should achieve a projection of the structure

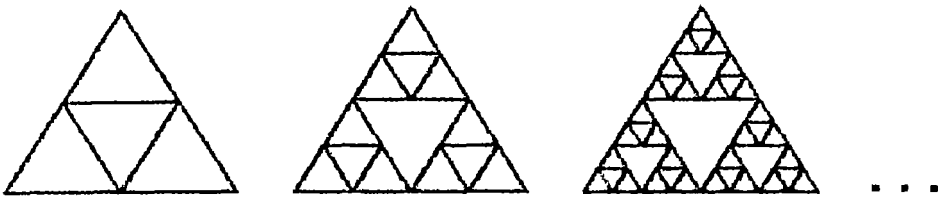


Figure 3

[4]

Some fractal transformations can be achieved modifying the commands' string $t^n(D)$ through some operations

1) **Scaling** the figure can be increased k times, applying the following function

$$s^k \quad S \rightarrow S \ , \ s^k(x_1x_2 \ \ x_m)=x_1^k x_2^k \ \ x_m^k \ ,$$

to the string $t^n(D)$

2) **Translation** a figure can be translated k times on the direction of a primitive c if we apply the function

$$d_c^k(X) = c^kX$$

to the string $X = t^n(D)$

3) **Rotation** the function ρ^k defined below achieves k rotations of the fractal

$$\rho^k(x_1x_2 \ \ x_m)=Next^k(x_1)Next^k(x_2) \ \ Next^k(x_m), \ k\in Z$$

The rotation can be performed with an angle multiple of 45, 60 or 90 grades, depending on the selection of the primitives Since the order of the primitives in the set C is important, we must respect the anticlockwise direction We notice that the function ρ has the following property $\rho^k(t^n(X)) = t^n(\rho^k(X))$ This property simplifies the computation in fractals generation, fractals which have a primary detail compound from more than one primitive

Let's consider again the second example

$$t^n(urdl) = t^n(u)t^n(r)t^n(d)t^n(l) =$$

$$= t^n(u)t^n(Next^{-2}(u))t^n(Next^4(u))t^n(Next^2(u)) =$$

$$= t^n(u)t^n(\rho^{-2}(u))t^n(\rho^4(u))t^n(\rho^2(u)) =$$

$$= t^n(u)\rho^{-2}(t^n(u))\rho^4(t^n(u))\rho^2(t^n(u))$$

We notice that we can compute $t^n(u)$ one time, then the commands are executed, eventually transformed by the rotations ρ^{-2}, ρ^4 and ρ^2

If we have to recognize a fractals family then we will proceed as follows we find the primitives, construct the corresponding commands string, we decompose the commands string in substrings of length equal with f(x), then every substring f(x) is substituted with x We apply the same action to the resulting string until we obtain D [5] The commands string $t^2(D)$ from the second example,

rurdr uluru rurdr drdld rurdr , is transformed by f^{-1} in

rurdr → **r** , **uluru** → **u** , **rurdr** → **r** , **drdld** → **d** and **rurdr** → **r**, then the resulting string **rurdr**, which is f(r), is substituted by **r**

REFERENCES

1 F J Brandemburg, M P Chytil, On Picture Languages Cycles and Syntax - Directed Transformations, Technische Berichte der Fakultat fur Mathematik und Informatik Universitat Passau, MIP-9020, 1990

2 D Dogaru, Metode noi în proiectare, Editura Stiinţifică şi Enciclopedică, Bucureşti 1988

3 H A Maurer, G Rozenberg, E Welzl, Using String Languages to Describe Picture Languages, Information and Control, Vol 54, Nr 3, 1982

4 M Vaida, A Posea, I Nistor şi alţii, Grafică pe calculator limbajele Pascal şi C, Editura Tehnică Bucureşti, 1992

5 R Vancea, S Holban, D Ciubotariu, Pattern Recognition, Editura Academiei, Bucureşti 1989

STUDIA UNIV BABEŞ-BOLYAI, MATHEMATICA XXXVIII, 3, 1993

A NOTE ON NON-MONOTONIC LOGICS

Doina TĂTAR* and Mihaiela LUPEA*

Dedicated to Professor I mil Muntean on his 60th anniversary

Received January 31, 1994
AMS Subject Classification 03B35,68Q40,68T27

Rezumat: Notă asupra logicilor nemonotone. Raţionamentul aproximativ e deosebit de interesant pentru că modelează mai exact reprezentarea şi tratarea cunoştinţelor în cazul informaţiilor incomplete Această lucrare introduce o modalitate de a obţine teoreme pornind de la astfel de cunostinţe (knowledge) incomplete, similar cu deducţiile în cazul clasic al logicii de ordinul întâi Pentru cazul teoriilor normale, se demonstrează că problema e complet reductibilă la cazul clasic

1. Introduction The classical logics are inadequate to capture the tentative nature of human reasoning Since people's knowledge about the world is necessarily incomplete, there will be times when we could be forced to draw conclusions based on an incomplete specification of pertinent details of the situations Under such circumstances, assumptions are made (implicitly or explicitly) about the state of the unknown factors Because these assumptions are not irrefutable, they may have to be withdrawn at some later time,if new evidence prove them invalid If this happens, the new evidence will prevent some assumptions from being made, hence all conclusions which can be arrived at only in conjunction with those assumptions will no longer be derivable

In common-sense reasoning, assumptions are often based on both supporting evidence and the absence of contradictory evidence Traditional logics cannot emulate this form of

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

reasoning, because they lack any tools for considering the absence of knowledge

Non-monotonic logic has been developed to deal with reasoning about incomplete informations There are four major formalizations of non-monotonic reasoning

- McCarty's circumscription [1]
- Moore's autoepistemic logic [4]
- Reiter's default logic [5]
- McDermott and Doyle non-monotonic logic [2],[3]

Reiter's default logic [5] is one of the most proeminent formalizations of non-monotonic reasoning One of the reasons for its attractiveness is the simplicity and naturalness of its underlying idea This logic represents defaults as certain type of inference rules whose applicability does not only depend on the derivability, but also on the underivability of some formulas

Classical logic deals with the formalization of absolutely correct forms of reasoning The aim of this note is to prove that,
in the normal context, the problem is completely reducible to classical case The deductive systems of logic allow us to formalize reasoning of rigurous proof of theorem and to infer conclusions from premises It defines a deduction relation between formulas, denoted by $\vdash$ This relation has the following properties [6]

- reflexivity

  $U_1,U_2, \ldots,U_n,V \vdash V$

- monotonicity

  if $U_1,U_2, \ldots,U_n \vdash V$ then $U_1,U_2, \ldots,U_n,Z \vdash V$

• transitivity

$$\text{if } U_1,U_2, \ldots,U_n \vdash V \text{ and } U_1,U_2, \ldots,U_n,V \vdash Z$$

$$\text{then} \quad U_1,U_2, \ldots,U_n \vdash Z$$

where $U_1,U_2, \ldots,U_n,V,Z$ are the formulas in first-order logic

2. Default logic The property of monotonicity tell us that a derived result cannot be invalidated by further results Also, the inference rules in deductive systems of classical logic are permissive They are always of the form $U_1,U_2, \ldots,U_n \vdash_{r_k} V$ with the significance "If $U_1,U_2, \ldots,U_k$ are theorems, then by rule r_k (of arity k) it results that V is a theorem "

A system which should be able to model non-monotonic resoning should also contain restrictive rules, of the form

" V is a theorem if $U_1,U_2, \ldots,U_k$ are not theorems "

Default logic allows formalizing default reasoning by means of particular inference rules, called **defaults** A **default** has the form $\frac{\alpha M\beta}{\gamma}$ and is interpreted as follows "if one belives α and if is consistent to belive β, then one can also belives γ"

A default theory will comprise, besides the default rules, a set of closed formulas of predicate logic which represent the basic knowledge and are treated as axioms

Definition 1 A **default theory** T is a pair (D,F) where

(i) D is a set of defaults (d) $\frac{\alpha M\beta_1, \ldots,M\beta_m}{\gamma}$, and $\alpha,\beta_1, \ldots,\beta_m,\gamma$ are closed formulas in first-order logic

(ii) F is a set of closed formulas in first-order logic

- α is called the **prerequisite** of default

- γ is called the **consequent** of default

We denote by $Pre(d)$ the prerequisite α of the default $d \in D$,and by $Cons(d)$ the consequent γ of the same d Similary,we introduce $Pre = \bigcup_{d \in D} Pre(d)$

Definition 2 An **extension of default theory** T is any set of all formulas that can be infered by means of the classical inference rules or by means of the defaults We will denote this set by $Th(D,F)$ and we will call them the **set of theorems** of $T=(D,F)$

A default theory can have an empty extension Howevei, it can be proved [5] that a non-empty extension exists for so called normal default theories, which all defaults have the form $\frac{\alpha : M\beta}{\beta}$

By analogy with the definition of a deduction for a formula U, and in accordance with definition 1 and definition 2, we can introduce the

Definition 3 Let $T=(D,F)$ be a default theory, and U and V two set of formulas in the first-order logic We denote $U \vdash V$ (and we call this V is non-monotonic deductible from U) if V is obtained from U either by application of a classical inference rule (like modus ponens, foi example) or by a default rule In this last case, U contains α and V contains β, if the normal default applied is. (d) $\frac{\alpha : M\beta}{\beta}$

We can specify that the default d is applied by denoting

$U \vdash_d V$ or $U \vdash V$ by rule d Now, we are ready to define the concept of a proof for a formula U according to a default theory $T=(D,F)$

Definition 4 A formula U is a **theorem** in a default theory $T=(D,F)$ (or, $U \in Th(D,F)$) if it exists a finite sequence of set of formulas $U_0, U_1, \ldots, U_n$, such that

$U_0 = F$, $U_1 = F \cup \{\alpha\}$, $\alpha \in Pre$, $U \in U_n$ and

a) $U_i \vdash U_{i+1}$, i=1,2, ,n-1

b) U_i is consistent,i=1,2, ,n (therefore U_i does not contain a formula V and his logical negation $\neg V$)

Observation: The sequence $U_0, U_1, ., U_n$ has the property

$U_0 \subseteq U_1 \subseteq \quad \subseteq U_n$

3. The main result Example Let $T=(D,F)$ be the normal default theory having the following set of premises

(i) $F=\{ C \rightarrow D , A \wedge B \rightarrow E , E \vee D , D \rightarrow G \}$ and

(ii) $D=\{ d_1, d_2, d_3, d_4 \}$ as

(d_1) $\dfrac{E \vee G \quad M(A \wedge G)}{A \wedge G}$

(d_2) $\dfrac{A \quad MB}{B}$

(d_3) $\dfrac{A \wedge E \quad MC}{C}$

(d_4) $\dfrac{M \neg E}{\neg E}$

According to definition 4,a proof for U=D may be the following

1) $U_0 = F$,

2) $U_1 = F \cup \{ E \vee G \}$,

3) $U_2 = U_1 \cup \{ A \wedge G \}$, $U_1 \vdash U_2$ by rule d_1,

4) $U_3 = U_2 \cup \{ A , G \}$, $U_2 \vdash U_3$ by rule $\dfrac{A \wedge G}{A, G}$,

5) $U_4 = U_3 \cup \{ B \}$, $U_3 \vdash U_4$ by rule d_2,

6) $U_5 = U_4 \cup \{ A \wedge B \}$, $U_4 \vdash U_5$ by rule $\frac{A,B}{A \wedge B}$,

7) $U_6 = U_5 \cup \{ E \}$, $U_5 \vdash U_6$ by rule $\frac{A \wedge B, A \wedge B \rightarrow E}{E}$,

8) $U_7 = U_6 \cup \{ A \wedge E \}$, $U_6 \vdash U_7$ by rule $\frac{A,E}{A \wedge E}$,

9) $U_8 = U_7 \cup \{ C \}$, $U_7 \vdash U_8$ by rule d_3,

10) $U_9 = U_8 \cup \{ D \}$, $U_8 \vdash U_9$ by rule $\frac{C, C \rightarrow D}{D}$

As $D \in U_9$, U_0, U_1, , U_9 is a proof for D

The following theorem emphasizes a conection between the relation $\vdash$ and the classical relation $\vdash\!\!\!-$ of deductibility in the first-order logic

Theorem. If $T=(D,F)$ is a normal default theory then $U \in Th(D,F)$ iff $F,P \vdash\!\!\!- U$ where P is the set of formulas defined as

"$\alpha \rightarrow \beta \in P$ iff $\frac{\alpha \quad M\beta}{\beta} \in D$"

Proof: The direct implication results by induction about the number k of utilised defaults

If k=0, then we have $F \vdash\!\!\!- U$ and thus $F,P \vdash\!\!\!- U$

Let $U \in Th(D,F)$ such that for U are applied k+1 defaults If the last default is (*d*) $\frac{\alpha \quad M\beta}{\beta}$, then $U(=\beta) \in U_n$,

$U_{n-1} \vdash_d U_n$, and $\alpha \in U_{n-1}$ By induction hypotesis, as for α are applied k defaults, $F,P \vdash\!\!\!- \alpha$ As $\alpha \rightarrow \beta \in P$, we obtain

$F,P \vdash\!\!\!- \beta(=U)$

By analogy,the converse implication can be proved

Observation: If a default theory is normal, then a deduction in this theory can be simulated as usual way in first-order theory

A similar theorem can be proved for the seminormal default theories [5]

REFERENCES

1 McCarty,J , Circumscripuon - A form of non-monotonic reasoning reasoning, Artificial Intelligence, vol 13 (1980),27-39

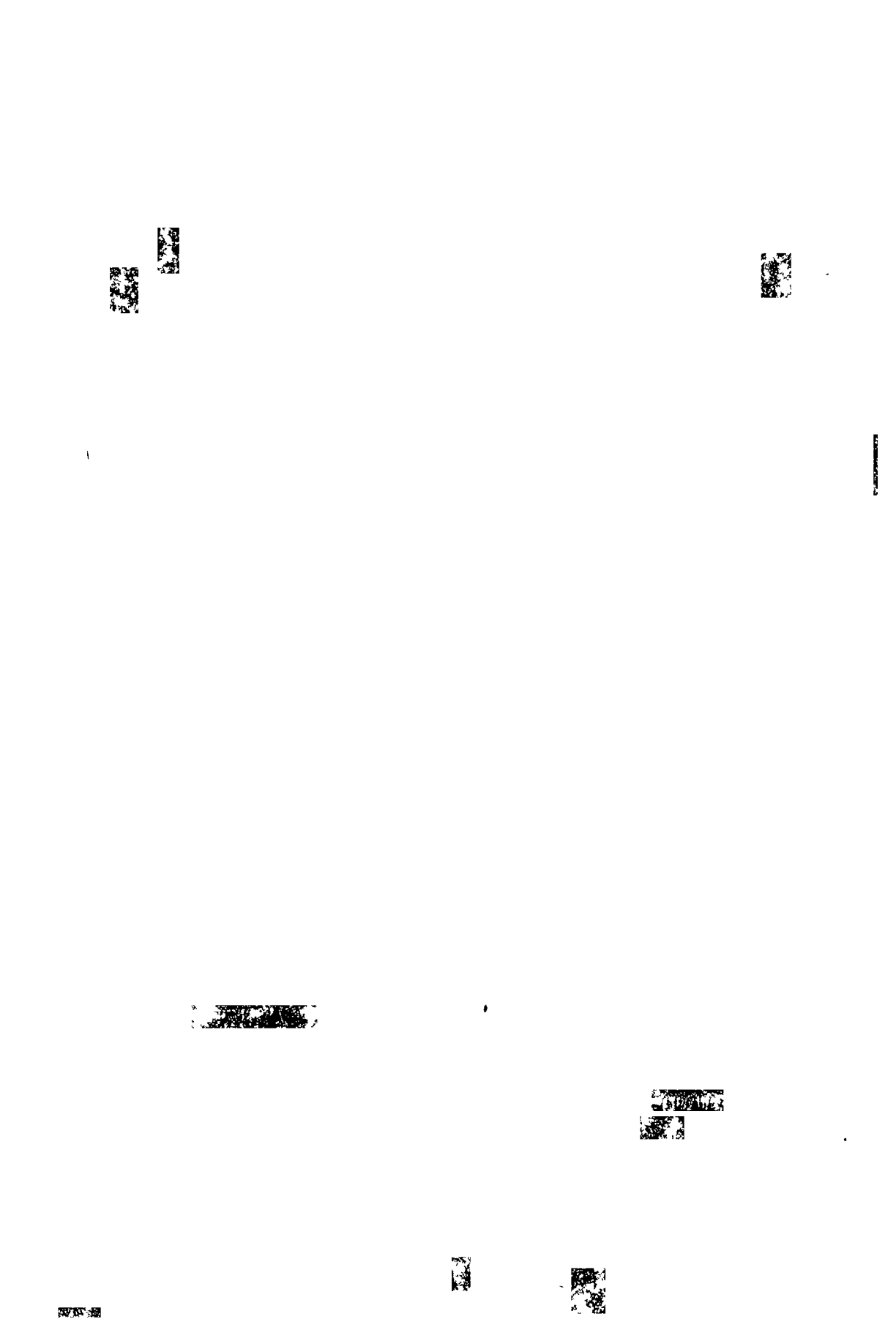
2 McDermott,D , Non-monotonic logic I, Artificial Intelligence Doyle,J vol 13 (1980),41-72

3 McDermott,D, Non-monotonic logic II, J ACM 29(1)(1982),33-57

4 Moore,R C, Semantical considerations on non-monotonic logic, Artificial Intelligence,vol 25 (1985),75-94

5 Reiter,R, A logic for default reasoning, Artificial Intelligence,vol 13 (1980),81-131

6 Thayse,A, From Standard Logic to Logic Programming , John Wiley & Sons,New York (1988)



STUDIA UNIV BABEŞ-BOLYAI, MATHEMATICA XXXVIII 3, 1993

ON SOME MODELS OF PARALLEL PERFORMANCE

Daniela VĂSARU*

Dedicated to Professor Emil Muntean on his 60th anniversary

Received November 29, 1992
AMS Classification 65Y05, 68Q05, 68M20, 68Q22

REZUMAT Articolul **"Asupra unor modele de performanţă paralelă"** prezintă câteva dintre cele mai folosite modele de caracterizare a performanţei algoritmilor paraleli Acestea au ca trăsătură comună folosirea fracţiilor seriale şi paralele în studiul performanţei Paralel cu prezentarea lor sunt discutate atât calităţile şi defectele lor cât şi relaţiile existente între ele În finalul articolului este dat un exemplu de folosire al acestor modele în caracterizarea performanţei paralele

1. Introduction. New requirements in engineering and computational science had lead to a strong interest in constructing a "teraflop" computer Parallel processing is considered to be the great hope in obtaining such a performance Ideally, on n_p processors a program will run n_p times faster than on a single one Unfortunately this is rarely the case One reason is the great disproportion existing between the progress in hardware technology and the methods of programming the parallel computers In what concerns the software part, there are a lot of problems waiting to be solved Two of them are the inexistence of a common complexity model for parallelism and the difficulties encountered in analyzing the performance and corectness of parallel algorithms

This paper presents a number of models of parallel performance, models that have in common the use of serial and parallel fractions in characterizing the parallel algorithms,

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

showing the relations between them and how can we use them in predicting the parallel performance

2. Preliminaries. The most common used measures of parallel performance are **speedup** and **efficiency** [2,3] They are both functions of problem size **n** and number of processors $\mathbf{n_p}$ and formally can be described by

$$S(n,n_p) = \frac{T(n,1)}{T(n,n_p)} \quad , \quad E(n,n_p) = \frac{S}{n_p} \tag{1}$$

T(n,i) is the time spent to solve a problem of size **n** by **i** processors Because of the overhead introduced by parallelization, **T(n,i)** is considered relative to the best serial implementation

The influence that the two parameters **n** and $\mathbf{n_p}$ have on the speedup and efficiency is of great practical importance By varying one or both parameters, different models of parallel performance are coming out

In order to make more readable the article, we will not mention always the parameters of a function For example, we will write S instead of $\mathbf{S(n,n_p)}$ It should be clear from the context on which parameters a function depends In general, all the functions have two parameters In the case that one of them is fixed we will not mention it

3. Amdahl's Law Consider an algorithm solving a problem of given size **n** that has one part intrinsically sequential and the other part, 100% parallelizable, can be distributed equally among the available processors Now, if **s** is the fraction of time spent by a uniprocessor on the serial part of the algorithm (serial fraction) and **p** is the fraction of time

spent on the parallelizable part by the uniprocessor, then the time spent by n_p processors on the same problem will be $(s+p/n_p)*T(1)$ So, the speedup will be given by

$$S = \frac{(s+p)*T(1)}{(s+p/n_p)*T(1)} = \frac{1}{s+(1-s)/n_p} \qquad (2)$$

This is a steep function of s near s = 0 For a fixed number of processors the speedup is increasing unbounded with the decreasing of s This case can be used in selecting the most efficient parallel algorithm (in the sense of efficient use of processors) among different algorithms solving the same problem the one with the minimum s is the best

What's happening if we have a single algorithm for a fixed-problem size and an increasing number of processors? Then the speedup is assimptotically bounded by 1/s

$$S \to 1/s \quad as \quad n_p \to \infty \qquad (3)$$

This is the performance forecast by Amdahl's Law if a computer has two speeds or modes of operation during a given calculation, the slow mode will limit overall performance even if the fast mode is infinitely fast [1,4] It means that if an algorithm has 2% sequential part, speedup greater than 50 one can not obtain even if it has thousands of processors

This result was used by Amdahl as an argument against building massively parallel systems

The limitation of speed given by (3), as we will see in the next sections, is valid only for the case under consideration, i e for fixed-size problems That's the reason why the model discussed is also called the **fixed-size model**

4. Moler's Law Moler was one of the firsts to show that Amdahl's limit can be beated [1] He had proved that parallelism can attain desired speedup for sufficiently large

computations

Instead of considering a fixed size problem and an increasing number of processors, he had study the case of a fixed number of processors and instances of the same problem but with different sizes He had shown that the serial fraction **s** is dependent on the input size **s = s(n)** So, s isn't constant (the main assumption in the fixed-size model) Even if **S** is bounded by **1/s**, this limit isn't fixed He define an effective parallel algorithm as one for which **s(n)** $\rightarrow$ **0** when **n** $\rightarrow \infty$ In this case, for a fixed number of processors n_p, one would obtain

$$S = \frac{1}{s(n)+(1-s(n))/n_p} \rightarrow n_p \text{ , } for\ n \rightarrow \infty \qquad (4)$$

It follows that for problems large enough, it can be obtained the desired speedup (the processors are used efficiently) In practice, **n** cannot grow to infinity but it can be made as big as the available memory allows

5. Sandia's Model The researchers from Sandia Laboratories had studied the variation of speedup starting from the following observation if one has more computing power, he usually don't use it to solve the same problem of fixed size but larger instances of the problem [1,6] The reason is obvious there is no point in using more processors than the concurrency of a problem because then, some of them will remain idle Also, by increasing the number of processors the overhead due to communication is increasing and if the problem size is fixed, than the computational time will remain fixed, while the communication time will grow, affecting the overall performance

By scaling the problem size proportionally with the number of processors, the serial

fraction s can be made as little as we want The serial component of an algorithm is determined by the startup time, serial bottlenecks and I/O, which are not dependent on the problem size The parallelizable part of an algorithm varies with the input size It follows that s can be made to shrink under these circumstances

Adding more processors brings more memory and more speed How do we scale the problem size with memory or with speed? Most scientists scale the problem in order to occupy all the available memory This is called the **scaled model** and it is the one proposed by Sandia They assumed as a first approximation that the parallel part grows proportionally with the number of processors

The model proposed by Sandia as an alternative to the fixed-size model is, in fact, the inverse of the Amdahl's paradigm Instead of asking how fast a given serial program will run on n_p processors, it's asked how long it will take to run a given parallel program on an uniprocessor

If **s'** is the fraction of time spent by a multiprocessor machine with n_p processors on serial parts of a parallel program and **p'** the fraction of time spent by the same multiprocessor on the parallel part, the time to run the program on an uniprocessor will be $(s'+p'*n_p)*T(n_p)$. Then , the scaled speedup will have the form

$$S_w = \frac{(s'+p'*n_p)*T(n_p)}{(s'+p')*T(n_p)} = s' + (1-s')*n_p \qquad (5)$$

It is easy to see that the scaled speedup is a function of moderate slope $1-n_p$ of s' (a line) and it grows with increasing n_p

Another alternative is to scale the problem size in order to maintain constant execution time This is called the **fixed-time model** An example for the use of this model is the

weather prediction It doesn't make sense to have an execution time greater than 24 hours in predicting the time for the next day

To illustrate the difference between these models (in fact, the fixed-time model is intermediary between the fixed-size model and the scaled model), we will present an example For the multiplication of two matrix (with dimensions **nxn**), the memory needed is $\mathbf{O(n^2)}$ but the number of operations is $\mathbf{O(n^3)}$ For the scaled model (problem size scales with memory) $\mathbf{n^2}$ grows proportionally with $\mathbf{n_p}$ but for the fixed-time model, $\mathbf{n^3}$ grows proportionally with $\mathbf{n_p}$ (i e $\mathbf{n^2}$ grows as $\mathbf{n_p^{2/3}}$)

6. General Model of Parallel Performance Carmona and Rice proposed a general model of parallel performance which capture the previous presented models [2]

They use the same criteria of characterizing the parallel algorithms, speedup and efficiency, but with some slights modifications of (1) Instead of considering running time as a measure of the complexity of algorithms, time beeing dependent on the architecture, they use as a measure of work the computational counts or unit counts based on the size of an indivisible task

If **wa** is the **work accomplished** by a parallel program and **we** the **work expended** by the same program, the efficiency can be expressed by $\mathbf{E = wa/we}$

The work accomplished is given by the number of operations done by the best serial implementation and it's not depending on the number of processors, only on the problem size In general, $\mathbf{wa < we}$ because the parallelization introduces some overhead, redundant operations, communication requirements not needed in the serial case

The difference **ww = we-wa** is called the **wasted work** It covers the time needed for the following activities waiting for other tasks to complete work, communication delays and/or memory contention (dependent on the particular architecture and the implementation of the algorithm), operation redundancies introduced by the implementation, including task activation/ termination and synchronization code **Ww** is a function of both problem size and number of processors

Under these considerations, the expressions for efficiency and speedup will be

$$E(n,n_p) = \frac{wa(n)}{we(n,n_p)} = \frac{wa(n)}{wa(n)+ww(n,n_p)} \tag{6}$$

$$S(n,n_p) = E * n_p = \frac{wa(n)}{wa(n)+ww(n,n_p)} * n_p \tag{7}$$

Using these work parameters, Rice and Carmona give also new interpretations for the serial fraction **s** and the scaled serial fraction **s'** From (2) and (7) it follows

$$s = \frac{ww}{wa} * \frac{1}{n_p-1} \quad , (n_p>1) \tag{8}$$

So, **s** can be interpreted as the distribution across the additional processors of the ratio of work wasted to work accomplished Similarly, from (5) and (7)

$$s' = \frac{ww}{we} * \frac{n_p}{n_p-1} \quad , (n_p>1) \tag{9}$$

Therefore, **s'** can be interpreted as a collective wasted effort $\mathbf{n_p}$***s1**, where **s1** is the distribution across the additional processors of the ratio of work wasted to work expended

From eqs (8) and (9) it follows that **s,s',p,p'** are functions both of problem size and the number of processors This modifies the previous points of view, i e **s** was considered constant for fixed-size problems as the number of processors increases, **s'** was considered only for scaled problems, with $\mathbf{n=n(n_p)}$ a increasing function of $\mathbf{n_p}$ These differences appear from the fact that the new definitions of **s** and **s'** incorporate wasted work

It is not difficult to see that the fixed size-model is a particular case of these new definitions if the wasted work has the form $ww = (n_p - 1) * w(n)$, where $w(n)$ is a function only of n, then s will be constant for fixed-size problems Intuitively, ww has this form if each one of the new n_p-1 processors contributes in equal part to the wasted work (with $w(n)$) and these contributions don't depend on the number of processors In a similar way we can show that the other described models are particular cases of this general one

Using eqs (6),(7),(8) and (9),it results the following law·

$$S = \begin{cases} s/s' & , ww>0 \\ n_p & , ww=0 \end{cases}$$
$$E = \begin{cases} p'/p & , ww>0 \\ 1 & , ww=0 \end{cases} \qquad (10)$$

This law relates s and s' for different combinations of n and n_p, while the previous models showed the trend in speedup when s and s' are varied for a given number of processors, or are held fixed and n_p is varied The law (10) also gives an interesting relation between the fixed-size and the scaled model, showing how can one predict the other From (2),(5) and (10) it's easy to derive

$$s' = \frac{s}{s+(1-s)/n_p} \qquad (11)$$

$$s = \frac{s'}{s'+(1-s')*n_p} \qquad (12)$$

These relations can be used in two ways for a given speedup, one can determine from the base scalar fraction the scaled serial fraction (or viceversa), secondly (and more important), from the serial fraction of a base problem s one can derive the scaled serial fraction s' (and, therefore, the scaled speedup) for a larger problem, by simply making $s'=s$ in (5)

The general model proposed by Carmona and Rice is described by a group of assertions, assertions stating how the parameters influence each other on the curves of the form $n = n(n_p)$ These curves represent all possible relations between the problem size and the number of processors Given a function $f(n,n_p)$, the notation $f\uparrow$ (respectively $f\downarrow$) denotes that f increases (decreases) on some fixed curve $n = n(n_p)$ as n_p increases Also, $f\uparrow r$ (respectively $f\downarrow r$) denotes that f approaches the limit r on the curve as $n_p \to \infty$

The performance model is given by the following assertions

A1 $s'\downarrow \Rightarrow s\downarrow \Rightarrow S\uparrow$ (for any curve $n = n(n_p)$)

A2 $s\uparrow \Rightarrow s'\uparrow \Rightarrow E\downarrow$ (for any curve $n = n(n_p)$)

A3 Assume that $n = n(n_p)$ defines a constant s-curve Then $s' = \Theta(1)$ and $s'\uparrow 1$ Furthermore, $S\uparrow 1/c$ and $E\downarrow 0$, where $s(n(n_p),n_p)=c$ (constant s-curve)

A4 Assume that $n = n(n_p)$ defines a constant s'-curve Then $s = \Theta(1/n_p)$ and $s\downarrow 0$ Furthermore, $S = \Theta(n_p)$, $S\uparrow$ and $E\downarrow(1-c)$, where $s'(n(n_p),n_p) = c$ (constant s'-curve)

This general model provides a framework in which the various performance parameters can be compared and contrasted within a single unified view of speedup It is easy to see that assumption A3 is a generalization of the fixed-size model (Amdahl) and the assumption A4 of the scaled model (Sandia)

Now, one question easily arises why these differences between the general model and the previous ones with respect to the number of parameters on which s and s' depend? One reason it was given above The new definitions incorporate wasted work This is due to the fact that in all the other models the speedup was interpreted as the gain in time of a parallel implementation with respect to the serial implementation of the same algorithm, and not over

the implementation of the best serial algorithm that solve the problem, as it is the case in the general model (best serial implementation)

7. **Example** To illustrate the use of these models in predicting the performance of the parallel algorithms, we will give an example The problem to be solved is the evaluation of a polynomial expression at a given point x

$$f(x) = \sum_{i=0}^{n} c_i x^i$$

It is well known that the standard serial algorithm takes 3n-1 unit counts (n additions and 2n-1 multiplications, considering that an addition and a multiplication take each a unit count) The best serial algorithm is the Horner scheme and it takes 2n unit counts (n additions and n multiplications)

A parallel algorithm for solving this problem using p processors, $p \le n/2$, is the following (see [5,7]) each processor i evaluates, using the Horner scheme, the following polynomial

$$g_i(x) = \sum_{j=0}^{[(n-i)/p]} c_{jp+i} x^{jp} \qquad i = 0, \ldots p-1$$

The value of the initial polynomial can be computed from the following expression

$$f(x) = \sum_{i=0}^{p-1} g_i(x) * x^i$$

This parallel algorithm takes (2n/p + 2*log p) unit counts (where the base of the logarithm is 2) For more details on the analysis of the complexity see [5]

In order to study the performance of the algorithm, we have to determine the serial fractions From above and from the general model of performance, we have

wa = 2n,

we = p(2n/p + 2*log p) = 2(n + p*log p) ,

$ww = 2p*\log p$

$s = (p*\log p)/(n*(p-1))$

$s' = (p^2*\log p)/((n + p*\log p)*(p-1))$

$S = n/(n/p + \log p)$

$E = n/(n + p*\log p)$

We can see that the parallel algorithm is efficient in the sense of Molei for a fixed number of processors, $s(n) \to 0$ when $n \to \infty$ and $S \to p$ It depends on our interests and on the available memory how much we will increase the dimension of the problem

From the restriction $p \leq n/2$ it comes that we cannot increase to infinity the number of processors without increasing the dimension of the problem, if we want to make an efficient use of the processors

For a fixed problem size n, the speedup is an increasing function of p, when $1 < p \leq n/2$ (it can be seen by studying the sign of the derivative) It follows that the optimal number of processors (in order to obtain a maximum speedup) is $p = n/2$ and the maximum obtainable speedup for fixed n is $S_{max} = n/(1+\log n)$ and the efficiency will be $E = 2/(1 + \log n)$ This efficiency is not very good, especially for big problems

If we want to find the optimal number of processors in order to obtain a maximum efficiency for a given problem size, we have to study the expression of E It is a decreasing function of p and so, if we want an optimal efficiency, it will be obtained for $p=2$ I this case, $E_{max} = n/(n+2)$ and $S = 2n/(n+2)$

We can see that maximizing the efficiency is not the same thing as maximizing the speedup Sometimes is better to find a way in between these two extremes

If neither the dimension of the problem, nor the number of processors is fixed, we can predict the performance of the parallel algorithm for various relations between these two parameters For example, if n = c*p (with constant c≥2, from the restriction on the number of processors), we obtain

S = n/(c + log (n/c)) and E = c/(c + log(n/c))

It comes that the speedup is increasing with the dimension of the problem and the number of processors, but the efficiency is decreasing

There are many interesting conclusions that can be find out from the expressions above We will conclude with one of them

If we are interested in maintaining a fixed efficiency E, how do the parameters n and p need to be corelated? From the expression of the efficiency it comes out that

n = (E*p*log p)/(1-E)

It means that we have to grow the dimension of the problem proportionally with p*logp (this is the isoefficiency function for the parallel algorithm, as defined in [4]) in order to maintain an efficient use of the processors

8. Final Remarks In conclusion, we will give a summary of the most important applications of these models:

- determining the best parallel algorithm for solving a fixed size problem on a given architecture (the one with the least scalar fraction),
- as the scalar fraction of an algorithm depends on the architecture used, we can determine the most appropriated architecture on which the parallel algorithm should be implemented or

viceversa (finding the minimum s),

- for a fixed size problem we can determine the optimal number of processors to be used in order to maximize the speedup or the efficiency,

- we can find out what relation has to exist between the dimension of the problem and the number of processors in order to maintain a fixed efficiency (called the isoefficiency function)

There are also other models for predicting the parallel performance, for a general view see [4] There isn't a best model, it depends on our interests which one should we use, each is appropriate for a different situation That is the reason why we had choose to present the models that have in common the use of serial fractions in this case, the general model of performance of Rice and Carmona is the best, as it is a generalization of all the others

REFERENCES

1 G F Carey (ed), Parallel Supercomputing Methods, Algorithms and Applications, WILEY Series in Parallel Computing 1989

2 E A Carmona, M D Rice, Modeling the Serial and Parallel Fractions of a Parallel Algorithm, Journal of Parallel and Distributed Computing, vol 13, no 3, p 286-298, 1991

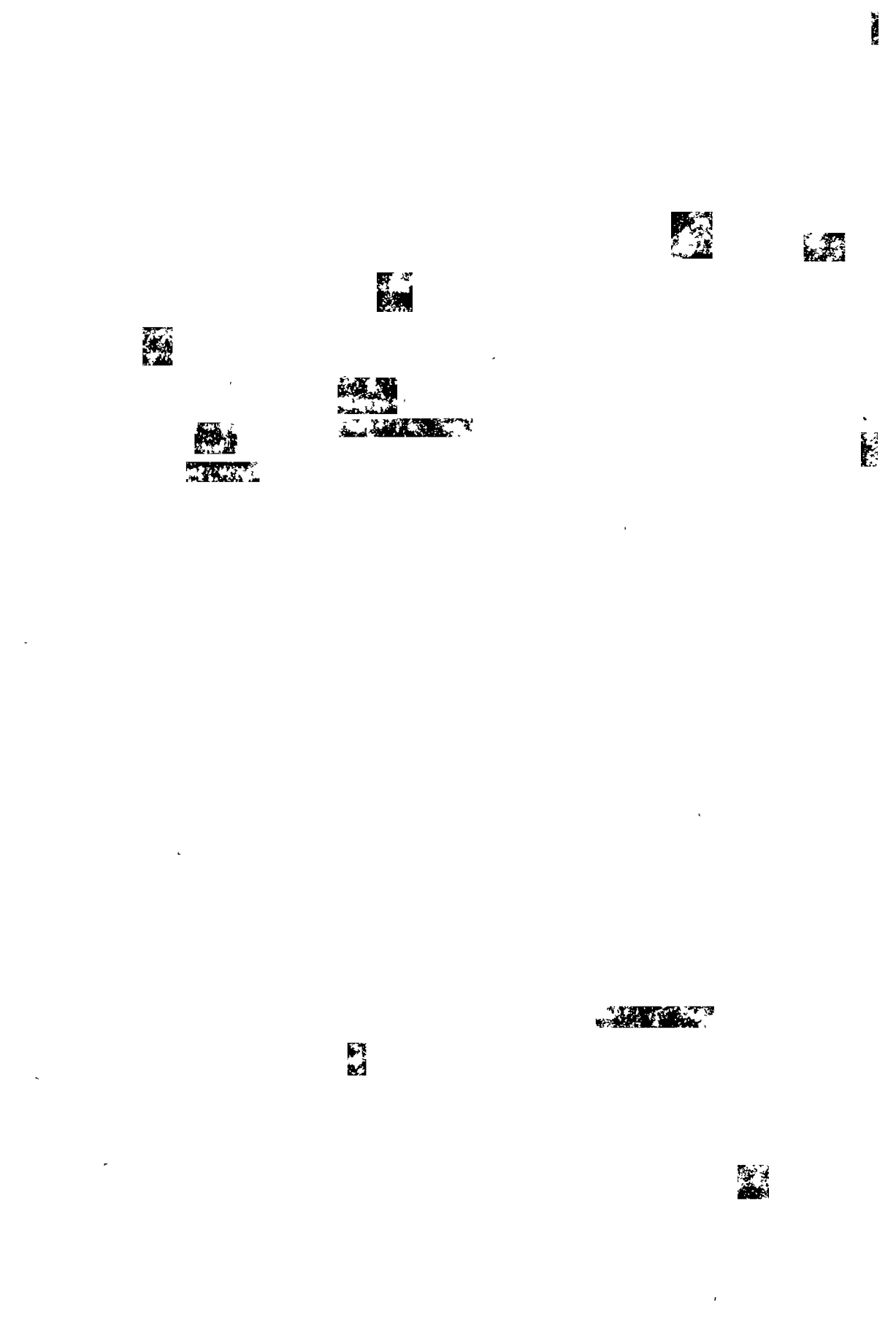
3 Gh Coman, D L Johnson, Complexitatea algoritmilor, Universitatea "Babes-Bolyai", Facultatea de Matematica si Fizica, curs litografiat, 1987

4 V Kumar, A Gupta, Analyzing Scalability of Parallel Algorithms and Architectures, AHPCRC Preprint 92-020

5 D Văsaru, Calcul Paralel, Lucrare de Diploma, 1991, Univ "Babes-Bolyai", Fac de Matem , Cluj

6 H M Wacker, The Use of Amdahl's Scaled Law in Determining the Power of Vector Computers, 1989 CERN School of Computing, p 156-171

7 S Wilson, Numerical Recipes for Supercomputers, in Supercomputational Science, R G Evans,S Wilson (ed), Plenum Press, New-York and London, 1990, pp 81-109



STUDIA UNIV BABEŞ-BOLYAI, MATHEMATICA, XXXVIII, 3, 1993

ANIVERSĂRI

PROFESSOR EMIL MUNTEAN AT HIS 60TH ANIVERSARY

by

Militon FRENŢIU

Professor Emil Muntean was born on July 31, 1933, in Măgura, Hunedoara County After finishing secondary school in 1952, he studied at the University of Cluj-Napoca He graduated in Mathematics from Cluj University in 1957 Still being in the fifth year, he was named at the Computing Institut of Academy Since then, the entire activity of Professor Munteanu is connected with computers He worked to the construction of MARICA (1959), a Romanian computer built from relays, and to the construction (in 1961) of DACICC-1, the first Romanian transistor-based computer Then (1967-1969) he worked to the complex project of building DACICC-200. Also, he had contributed to the realisation of some programs

He obtained his PhD from the SaintPetersburg University, S S S R , in 1964

In 1968 he became the Head of the newly Institut of Computer Technology (ITC) As the Head of the ITC in a pioniering period, he has directed with much competence and inspiration the research activity, to design and implement high level software products He really was a very good organizer

In 1990 he become full professor at our Faculty, Department of Computer Science, but his teaching activity started long time ago He used to teach the students of the Mathematics various subjects connected with computers In the last four years he gave courses in Expert Systems, and Computers Networks His courses were held at a high scientifical and pedagogical level

There are 8 computer scientists who own their PhDegrees to their supervisor, professor Emil Munteanu In the last years he become interested in spreading computer science knowledge, i e. he is today a known editor of books in this area He was the father of Microinformatica, and in the last year he invented Promedia

Professor Emil Muntean is a distingushed pedagogue, very appreciated by his students Also, it is a pleasure for all of us to have such a generous collegue

Now, on celebrating his 60th birthday we wish him "Many Happy Returns of the Day", and a long life in health and happiness to him and his family

ABSTRACT

of the Scientifical Work of Professor Emil Muntean

1 *Monograme de gen zero*, Buletinul ştiinţific al Cercurilor ştiinţifice studenţeşti al Univ din Cluj, 1956

2 *Monograme cu puncte aliniate*, Buletinul ştiinţific al Cercurilor ştiinţifice studenţeşti al Univ din Cluj, 1956

3 *Programarea calculului corijării roţilor dinţate*, Studii şi cercetări de matematică, 1958, pg 1-4

4 *Un algoritm de rezolvare a unei ecuaţii transcendente*, Studii şi cercetări de matematică, Vol XI, 1960, pg 133-139

5 *Programarea calculului unor indici calitativi ai angrenajelor cu roţi dinţate corijate*, Studii şi cercetări de matematică, Vol XI, 1960, pg 133-139

6 *Calculul şarjelor celor mai economice la cuptoarele de topit fontă*, Studii şi cercetări de matematică, Vol XI, 1960, pg 102-109

7 *Programe pentru maşinile de calcul elaborate la Institutul de Calcul din Cluj*, Studii şi cercetări de matematică, Vol XI, 1960, pg 92-101

8 *Cercetarea automată a depăşirii unităţii într-o maşină cu virgulă fixă*, Comunicările Academiei, 1961, nr 12, pg 1455-1457

9 *Uber de Unterschnit und das Ubergangs profil der mit Kegeligen Scheibenfrasor Bearbeiteten Schnecken*, Mathematica, Vol 3, 1961, pg 273-296

10 *Asupra interfeţei şi a profilului de record la melci prelucraţi cu freze conice*, Studii şi cercetări de matematică, Vol XIII, 1962

11 *Analiz algoritmov*, Buletinul Univ A A Jdanov, Leningrad, 1964, pg 102-107

12 *Method analiza graf-schemnih algoritmov*, Mathematica 5(28),1963

13 *Avtomaticeskoe postroienie graf-schemnih*, Buletinul Univ A A Jdanov, Leningrad, 1964, pg 97-106

14 *Analiz graf-schemnih algoritmov*, Buletinul Univ A A Jdanov, Leningrad, 1964

15 *Preobrazovanie algoritmov*, Leningrad, 1964

16 *Asupra unor transformări ale algoritmilor schemă-graf*, Lucrările consfătuirii naţionale de statistică, 1966, pg 153-160

17 *Controle automatique des programmes pour les machines electroniques a calcul*, Mathematica, Vol 9(31), 1966, pg 103-108

18 *Analiza logică a algoritmilor (I)*, Studii şi cercetări de matematică, Vol XVIII, 1966, nr 8, pg 1113-1127

19 *Analyse logique des algorithmes (II)*, Mathematica, Vol 9(32), 1967, nr 1, pg 111-128

20 *Programarea calculatoarelor electronice*, Gazeta matematică, seria A, 1966

21 *Sistemul de operare SERU al calculatorului DACICC-200*, Lucrările colectivului internaţional de aplicaţii ale calculatoarelor, 1967

22 *Calculatorul electronic DACICC-200*, Lucrările sesiunii ştiinţifice ICPUEC, 1969, pg 14-28

23 *Limbajul COBOL*, manual pentru liceele de informatică, ICI, 1972, 236 pagini

24 *Introducere în informatică*, manual pentru liceele de informatică, ICI, 1972, 214 pagini

25 *Noi consideraţii cu privire la limbajele de programare*, Revista de analiză numerică şi teoria aproximaţiei, Vol VI, 1972

26 *Utilizarea calculatoarelor în prelucrarea datelor - Arhitectură şi sisteme de operare*, Ed Dacia, Cluj-Napoca, 1974, 190 pagini

27 *Utilizarea calculatoarelor în prelucrarea datelor - Limbajul COBOL*, Ed Dacia, Cluj-Napoca, 1974, 271 pagini

28 *Programarea în limbajul ASSIRIS*, Ed Tehnică, Bucureşti, 1976, 334 pagini

29 *Înţelere în limbajul ADA*, Ed Tehnică, Bucureşti 1986, 281 pagini

30 *Multiplan-sistem de prelucrare a tabelelor*, Lucrările simpozionului "Informatica şi aplicaţiile sale", Cluj-Napoca, 1985, pg 1-6

31 *Sistemul de operare U (UNIX). Conceptele de bază ale proiectului*, Lucrările simpozionului CONDINF, Cluj-Napoca, 1985

32 *A cincea generaţie de calculatoare*, Lucrările sesiunii ştiinţifice ITC, 1986, pg 126-141

33 *Implementarea portabilă a compilatoarelor, viitorul electronicii şi al informaticii*, Ed Academiei, 1983, pg 121-132

34 *Asupra verificării corectitudinii programelor*, Sesiunea PROCOMP, ITC, 1989

În cel de al XXXVIII-lea an (1993) *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)
educaţie fizică (semestrial)

In the XXXVIII-th year of its publication (1993) *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)
physical training (semesterily)

Dans sa XXXVIII-e année (1993) *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)
éducation physique (semestriellement)