MACROFORT : a Fortran code generator in MAPLE

Claude Gomez

UNITÉ DE RECHERCHE
INRIA-ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

Rapports Techniques

N° 119

Programme 5
Automatique, Productique,
Traitement du Signal et des Données

MACROFORT : A FORTRAN CODE GENERATOR IN MAPLE

Claude GOMEZ

Mai 1990

RT_0119

MACROFORT : a FORTRAN code generator in MAPLE

MACROFORT : un générateur de code FORTRAN dans MAPLE

Claude Gomez

INRIA
Domaine de Voluceau
Rocquencourt - BP105
78153 Le Chesnay Cedex
FRANCE

Abstract

MACROFORT is a package added to MAPLE which permits complete FORTRAN 77 code generation. While using MACROFORT, the user stays in MAPLE. He no longer deals with FORTRAN label numbering. Complicate FORTRAN loops are easy to do because `while` and `until` capabilities exist in MACROFORT. The generated code can be optimized. Two examples are given. In the first one, MAPLE is not able to solve numerically a problem using its own functions. In the second one, a naturally recursive problem is easily solved by recursively generating a FORTRAN program.

Résumé

MACROFORT est un nouveau package de MAPLE qui permet de générer du code FORTRAN 77 sans sortir de MAPLE. Les étiquettes FORTRAN deviennent inutiles et il est possible de réaliser des boucles compliquées à l'aide des boucles `while` et `until` fournies par MACROFORT. Le code FORTRAN généré peut être optimisé. Nous donnons deux exemples d'utilisation de MACROFORT. Dans le premier, MAPLE ne peut pas résoudre le problème en utilisant ses propres fonctions et dans le deuxième un problème récursif est facilement résolu par un programme FORTRAN.

1 Why generating FORTRAN ?

MAPLE is a computer algebra system and is well suited to handle symbolic computations. It is possible to perform numerical calculations in MAPLE as well with floating-point numbers with a fixed number of digits (which can be as big as we want). A few MAPLE functions deal with numerical computing, e.g. **evalf** and **fsolve** which permit numerical integrations and numerical solutions of equations. But the running time of MAPLE in pure numerical computing is not as good as the one of FORTRAN code. Moreover, if we want to write portable numerical software and give it to another user who does not have MAPLE, a good tractable language for industry is FORTRAN. Also, there are numerical problem that MAPLE fails to solve (see 4). Then, it would be nice to gather MAPLE and FORTRAN in order to use the capabilities of MAPLE for computer algebra and the capabilities of FORTRAN for numerical computing.

We can use MAPLE as a preprocessor for numerical analysis. For instance when we have to do a lot of symbolic manipulations as performing derivatives, integrals, before writing FORTRAN code.

There are problems which are naturally described in a recursive way. If the problem is tail recursive, it is easy to solve it in FORTRAN by using iterations. If it is not, it is very complicated to do that (see 5).

2 Why using MACROFORT ?

There is a MAPLE procedure called **fortran**. It takes as argument a MAPLE expression and returns it in FORTRAN syntax.

For instance we have :

```
> e:=sin(tan(x^(y+1))-log(sin(x-1/y)))+Pi*x^(x^x)/(1.23-abs(y))+expand((1+x)^5);
>
                                                                    x
                                                                 (x )
                                (y + 1)                       Pi x
    e := - sin(- tan(x       ) + ln(sin(x - 1/y))) + ------------- + 1 + 5 x
                                                      1.23 - abs(y)

              2      3      4    5
        + 10 x  + 10 x  + 5 x  + x

> fortran([exp=e]);
      exp = -sin(-tan(x**(y+1))+alog(sin(x-1/y)))+Pi*x**x**x/(0.123E1-ab
     +s(y))+1+5*x+10*x**2+10*x**3+5*x**4+x**5
```

So, we could use this function to generate the FORTRAN statements which do the symbolic calculations into a file and then merge it with the remaining part of the FORTRAN code. But we find this process not very easy because we have to use MAPLE, to create intermediate files, to write FORTRAN code in another file and to merge the files. It would be nice to stay in MAPLE for the whole process. This can be done with MACROFORT.

3 MACROFORT

Actually, we have made a kind of macro FORTRAN we have called MACROFORT. A MAPLE user can perform symbolic calculations and write FORTRAN code without leaving MAPLE. The

original version of MACROFORT was made for the computer algebra system MACSYMA (see [1]).

To generate FORTRAN code by using MACROFORT, we have to build a MAPLE list. Each element of which corresponds to a single FORTRAN statements or to a few FORTRAN statement we call a macro FORTRAN statement.

The syntax of a FORTRAN statement or of a macro FORTRAN statement in MACROFORT is a MAPLE list where the first element is a keyword describing the statement, the optional other elements are relevant arguments. A keyword is made from the FORTRAN instruction name (when it exists) with a `f` at its end for a single FORTRAN statement or a `m` at its end for a macro FORTRAN statement. These keywords correspond to what we call a MACROFORT single instruction or a MACROFORT macro instruction.

3.1 MACROFORT single instructions

The MACROFORT single instructions are the following:

`[callf,name,list]`	generates	`call name (list)`
`[closef,unit]`	generates	`close (unit)`
`[commentf,string]`	generates	`c string`
`[commonf,name,list]`	generates	`common /name/ list`
`[continuef,label]`	generates	`label continue`
`[declaref,type,list]`	generates	`type list`
`[dof,label,index,start, end]`	generates	`do label, index=start, end`
`[dof,label,index,start, end,step]`	generates	`do label, index=start, end, step`
`[elsef]`	generates	`else`
`[endf]`	generates	`end`
`[endiff]`	generates	`endif`
`[equalf,variable,expression]`	generates	`variable=expression`
`[formatf,label,list]`	generates	`label format (list)`
`[functionf,type,name,list]`	generates	`type function name (list)`
`[gotof,label]`	generates	`goto label`
`[if_goto_f,condition,label]`	generates	`if (condition) goto label`
`[if_then_f,condition]`	generates	`if (condition) then`
`[openf,unit,file,status]`	generates	`open (unit=unit, file='file', status='status')`
`[parameterf,list]`	generates	`parameter (list)`
`[programf,name]`	generates	`program name`
`[readf,file,label,list]`	generates	`read (file, label) list`
`[returnf]`	generates	`return`
`[subroutinef,name,list]`	generates	`subroutine name (list)`
`[writef,file,label,list]`	generates	`write (file, label) list`

The arguments of MACROFORT instructions are MAPLE names and you have to quote them if necessary.

When "condition" appears as an argument of a MACROFORT instruction, you only have to write it with MAPLE syntax and MACROFORT will do FORTRAN translation. For instance:

`[if_then_f,a>=b]`.

When you want to introduce logical operators **not**, **and** and **or** in a condition, you have to use the names `NOT`, `AND` and `OR` with a functional notation. For instance:

`[if_then_f,OR(a=b,NOT(c<d))]`.

The label numbers are automatically generated by MACROFORT. When "label" appears as an argument of a MACROFORT instruction, you have to put a MAPLE name. A same name corresponds to a same label and MACROFORT will generate the label number. In fact we will see latter (see 3.2) that we can always avoid using labels.

When "list" appears as an argument of a MACROFORT instruction, it corresponds to an argument FORTRAN list which you have to write as a MAPLE list. For instance:

`[callf,foo,[a,b,c]]`

or

`[formatf,['2x,e14.7'],[x,y]]`.

All this is also available for MACROFORT macro instructions. But for them no label is needed.

3.2 MACROFORT macro instructions

The MACROFORT macro instructions are the following:

`[dom,index,start,end, step,do_list]`	generates	`do label, index=start, end, step`
		`do_list`
		`label continue`

`[dom,index,start,end, do_list]`	generates	`do label, index=start,end`
		`do_list`
		`label continue`

`[functionm,type,name,list, body_list]`	generates	`type function name (list)`
		`body_list`
		`end`

`[if_then_else_m,condition,then_list, else_list]`	generates	`if condition then`
		`then_list`
		`else`
		`else_list`
		`endif`

`[if_then_m,condition,then_list]`	generates	`if condition then`
		`then_list`
		`endif`

`[programm,name,body_list]`	generates	`program name`
		`body_list`
		`end`

`[openm,unit,file,status, body_list]`	generates	`open (unit=unit, file='file',`
		`status='status')`
		`body_list`
		`close (unit)`

`[readm,file,format_list, var_list]`	generates	`read (file,label) var_list` `label format (format_list)`

`[subroutinem,name,list,body_list]`	generates	**subroutine** name (list) body_list **end**
`[writem,file,format_list, var_list]`	generates	**write** (file,label) var_list label **format** (format_list)

There are also the following macro instructions:

`[commonm,name,list]`	generates	**common** /name/ list
`[declarem,type,list]`	generates	type list

The only difference with **commonf** and **declaref** single instructions is that you can put these macros everywhere in the list describing the program and MACROFORT put them at the right place in the generated FORTRAN code. This permits to declare a variable only when it is used in the body of the program. These macros only work within a **programm**, **functionm** or **subroutinem** macro instruction, otherwise there are ignored.

There are other very important macro instructions. We do not give the corresponding generated FORTRAN code because it is complicated.

First there are two macro instructions corresponding to WHILE and UNTIL loops. We explain below their semantic with a PASCAL like syntax:

`[whilem,condition,init_list,while_list,while_max]` :
  <init_list>
  while condition **do**
    <while_list>
  end.

`[untilm,condition,init_list,until_list,until_max]` :
  <init_list>
  do <until_list>
    until condition
  end.

"max_while" and "max_until" arguments denote the maximum number of iterations that the WHILE loop or the UNTIL loop will execute. When the maximum is reached, the loop stops and a message is issued.

"do_list", "then_list", "else_list", "body_list", "init_list", "until_list" and "while_list" arguments must be MAPLE lists describing FORTRAN statements with MACROFORT syntax.

You can nest as many loops as you want.

`[matrixm,variable,matrix]` is another very useful macro instruction. It is used to make assignations of the elements of a matrix. "variable" is the name of a FORTRAN matrix and "matrix" is the name of a MAPLE matrix.

For example, after:

```
a:=array([[x^2,x-y],[x/y,x^2-1]]);
```

$$a := \begin{bmatrix} x^2 & x-y \\ x/y & x^2-1 \end{bmatrix}$$

`[matrixm,v,a]` generates:

```
v(2,2) = x**2-1
v(1,2) = x-y
v(2,1) = x/y
v(1,1) = x**2
```

An example of the use of **whilem** and **matrixm** is given in section 4.

In fact, a great number of single instructions (for instance **dof**, **if_then_f**, **writef** ...) are never used because macro instructions are better and by using them it is never necessary to use labels.

3.3 Flags and global variables

There are global variables defined by MACROFORT.

comment is a logical variable. When it is **true** (default value) MACROFORT generates automatically FORTRAN comments.

input is the logical unit number of standard input (the default is 5).

output is the logical unit number of standard output (the default is 6).

optimized is a logical variable. When it is **true** (the default value is **false**) MAPLE optimizer (**optimize** MAPLE function) is used when FORTRAN is generated.

precision specifies single or double precision in the generated FORTRAN program. It can be **single** (the default value) or **double**.

When double precision is used with MAPLE optimizer, you must know that MAPLE optimizer generates variables beginning with the letter "t". So you can add a `implicit doubleprecision(t)` declaration in the FORTRAN code by using:

`[declaref,'implicit doubleprecision',['(t)']]`.

3.4 Using MACROFORT

You have to load MACROFORT before using it by issuing the commands:

```
readlib(macrofort):
with(macrofort);
```

Then, three new functions are available:

pushe is a new function for adding an element to the end of a list. `pushe(<element>,'<list>');` do the job. Note that you have to quote <list> because it is an argument that is modified in the function.

init_genfor must be used without argument before a new FORTRAN code generation. It gives their default values to the global variables used by MACROFORT and it initializes the various counters used internally by MACROFORT (for label generation for example).

genfor is the function which really generates the FORTRAN code. Its argument is the MAPLE list describing the FORTRAN program.

In summary, to generate FORTRAN code in a file, you have to executes the following MAPLE commands:

```
flist:=[...]; # LIST DESCRIBING THE FORTRAN PROGRAM
writeto('foo.fortran');
init_genfor(); # INITIALIZATION OF MACROFORT
genfor(flist);
writeto(terminal);
```

4 Example 1: Generalized Newton method

4.1 Newton algorithm

We want to solve a non-linear system with n equations and n unknowns:

$$\begin{cases} f_1(x_1,\ldots,x_n) = 0 \\ \vdots \qquad\qquad \vdots \\ f_n(x_1,\ldots,x_n) = 0 \end{cases}$$

with $x_1,\ldots,x_n$ belonging to $\mathbb{R}$. We use the matrix notation defining the vectors X and $F(X)$, and the Jacobian matrix of the system $F'(X)$—a n by n matrix—as :

$$X \stackrel{\text{def}}{=} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \qquad F(X) \stackrel{\text{def}}{=} \begin{pmatrix} f_1(x_1,\ldots,x_n) \\ \vdots \\ f_n(x_1,\ldots,x_n) \end{pmatrix} \qquad F'(X) \stackrel{\text{def}}{=} \left(\frac{\partial f_i}{\partial x_j}\right)_{n,n}$$

The well-known algorithm of Newton method is—in Pascal like syntax— :

$X := X0$ { where X0 is an arbitrary vector }
while $NORM(F(X)) > EPS$ **do** { where $NORM$ computes the norm of $F(X)$ }
  < find Y solution of the linear system $F'(X).Y = -F(X)$ >
  $X := Y + X$
end.

The problem when encoding this algorithm in FORTRAN code is to compute the jacobian.

4.2 MACROFORT program

We give below a MAPLE function which generates a general FORTRAN program for solving a non-linear system by using MACROFORT.

`resol` is a FORTRAN subroutine solving a linear system of equations.

```
# GENERALIZED NEWTON ALGORITHM
#  F[I] IS THE TABLE OF THE FUNCTIONS WITH THE VARIABLES X[J]
#  X IS THE GENERIC NAME OF THE VARIABLES X[J]
#  N IS THE NUMBER OF FUNCTIONS AND VARIABLES

gen_newton := proc(ff,x,n)
  local ii, f, jj, m, jac, pg, linit, lwhile, fo, zj;
# main program
# jacobian matrix
  jac := array(1..n,1..n);
```

```
  for ii from 1 to n do
    for jj from 1 to n do
      jac[ii,jj] := diff(ff[ii],x[jj])
  od od;
# MAPLE list describing the main program is in the variable pg
  pg:=[[declaref,real,[f(n),zj(n,n),x(n)]],
       [writem,output,[''eps = ''],[]],
       [readm,input,['e14.7'],[eps]]];
# while instruction
  linit:=[[dom,i,1,n,
          [[writem,output,[''x(',i3,') = ''],[i]],
           [readm,input,['e14.7'],[x(i)]]]]];
  for ii from 1 to n do pushe([equalf,f(ii),ff[ii]],'linit') od;
  lwhile:=[[matrixm,zj,jac],
              [callf,resol,[zj,f,n]],
              [dom,i,1,n,[equalf,x(i),-f(i)+x(i)]]];
  for ii from 1 to n do pushe([equalf,f(ii),ff[ii]],'lwhile') od;
  pg:=[op(pg),
            [[whilem,znorm(f,n) >= eps,linit,lwhile,1000],
             [writem,output,['(2x,e14.7)'],[x]]]];
 pg:=[programm,newton,pg];

# MAPLE list describing the subroutine computing the quadratic norm of f
# is in variable fo
  fo:=[[declaref,real,[f(m)]],
      [equalf,znorm,0.],
      [dom,i,1,n,[equalf,znorm,znorm+f(i)**2]],
      [equalf,znorm,sqrt(znorm)]];
  fo:=[functionm,real,znorm,[f,m],fo];

# FORTRAN code generation
  writeto('newton.f');
  init_genfor();
  genfor(pg);
  genfor(fo);
  writeto(terminal);
end:
```

4.3 Application to a steel rolling problem

The system we want to solve is described by the equations:

$$f_1(F,h_2,\Phi) = h_2 - S - \frac{F + a_2(1-e^{a_3 F})}{a_1}$$

$$f_2(F,h_2,\Phi) = F - lkR\left(\frac{1}{2}\pi\sqrt{\frac{h_2}{R}}\arctan\sqrt{r} - \frac{\pi\xi}{4} - \ln\left(\frac{h_N}{h_2}\right) + \frac{1}{2}\ln\left(\frac{h_1}{h_2}\right)\right) + \frac{R\xi T_1}{h_2}$$

$$f_3(F,h_2,\Phi) = \arctan\left(\Phi\sqrt{\frac{R}{h_2}}\right) - \frac{1}{2}\sqrt{\frac{h_2}{R}}\left(\frac{\pi}{4}\ln\left(\frac{h_2}{h_1}\right) + \sqrt{\frac{R}{h_2}}\arctan\sqrt{r} - \frac{T_1}{klh_1} + \frac{T_2}{klh_2}\right)$$

with

$$r = \frac{h_1 - h_2}{h_2} \qquad \xi = \sqrt{\frac{h_1 - h_2}{R}} \qquad h_N = h_2 + R\Phi^2$$

and with the following values of the parameters: $a_1 = 610$, $a_2 = 648$, $a_3 = -.00247$, $l = 1250$, $k = .0014$, $R = 360$, $T_1 = 12$, $T_2 = 35$, $h_1 = 24$ and $S = 12$.

This set of equations is the simplified description of the behavior of a steel strip in a stand of a hot strip rolling mill.

If you want to solve it by using pure MAPLE, you cannot use **solve** to obtain the analytic solution because the equations are too complicated. Then, you can use **fsolve** to solve numerically the problem. But fsolve gives a bad solution which has no physical meaning. So you are obliged to use a FORTRAN program.

The MAPLE program which generates the corresponding FORTRAN code is:

```
# STEEL ROLLING
#   solution by Newton method

# numerical values of parameters
data:=[a1=610,a2=648,a3=-.00247,l=1250,k=1.4*10^(-2),gr=360,t1=12,
  t2=35,h1=24,s=12]:

# equations
exp1:=h2-s-(f+a2*(1-exp(a3*f)))/a1:
exp2:=f-l*k*gr*(Pi*sqrt(h2/gr)*arctan(sqrt(r))/2-Pi*csi/4-log(hn/h2)+
              log(h1/h2)/2)
      +gr*csi*t1/h2:
exp3:=arctan(phi*sqrt(gr/h2))-sqrt(h2/gr)*(Pi*log(h2/h1)/4+sqrt(gr/h2)*
     arctan(sqrt(r))-t1/k/l/h1+t2/k/l/h2)/2:
r:=(h1-h2)/h2:
csi:=sqrt((h1-h2)/gr):
hn:=h2+gr*phi^2:

# table of equations with the good variables
f[1] := subs([op(data),f=x[1],h2=x[2],phi=x[3]],exp1):
f[2] := subs([op(data),f=x[1],h2=x[2],phi=x[3]],exp2):
f[3] := subs([op(data),f=x[1],h2=x[2],phi=x[3]],exp3):

# loading MACROFORT
readlib(macrofort):
with(macrofort):

# reading and executing the MAPLE procedure generating the FORTRAN code
read gen_newton:
gen_newton(f,x,3);
```

4.4 Generated FORTRAN program

We give below the FORTRAN program generated by MACROFORT.

```
      program newton
        real f(3),zj(3,3),x(3)
        write(6,1000)
 1000   format('eps = ')
        read(5,1001) eps
 1001   format(e14.7)
c
c         WHILE  (eps<=znorm(f,3)) DO <WHILE_LIST> (1)
c
c         WHILE LOOP INITIALIZATION
            maxwhile1 = 1000
            nwhile1 = 0
c
          do 1002, i=1,3
            write(6,1003) i
 1003       format('x(',i3,') = ')
            read(5,1004) x(i)
 1004       format(e14.7)
 1002     continue
c
          f(1) = x(2)-3984.0/305.0-x(1)/610+324.0/305.0*exp(-0.247E-2*
     +x(1))
          f(2) = x(1)-0.315E4*0.3141593E1*sqrt(x(2))/sqrt(360.0)*atan(
     +sqrt(24-x(2))/sqrt(x(2)))+0.1575E4*0.3141593E1*sqrt(24-x(2))/sqrt(
     +360.0)+0.63E4*alog((x(2)+360*x(3)**2)/x(2))-0.315E4*alog(24/x(2))+
     +12*sqrt(360.0)*sqrt(24-x(2))/x(2)
          f(3) = atan(x(3)*sqrt(360.0)/sqrt(x(2)))-sqrt(x(2))/sqrt(360
     +.0)*(0.3141593E1*alog(x(2)/24)/4+sqrt(360.0)/sqrt(x(2))*atan(sqrt(
     +24-x(2))/sqrt(x(2)))-0.2857143E-1+0.2E1/x(2))/2
c
c         WHILE LOOP BEGINNING
 1005     continue
c
c         WHILE LOOP TERMINATION TESTS
          if (eps.le.znorm(f,3)) then
            if (nwhile1.le.maxwhile1) then
c
c             NEW LOOP ITERATION
              nwhile1 = nwhile1+1
c
c             <WHILE_LIST>
              zj(3,2) = -x(3)*sqrt(360.0)/sqrt(x(2)**3)/(1+360*x(3)**2/x
     +(2))/2-1/sqrt(x(2))/sqrt(360.0)*(0.3141593E1*alog(x(2)/24)/4+sqrt(
     +360.0)/sqrt(x(2))*atan(sqrt(24-x(2))/sqrt(x(2)))-0.2857143E-1+0.2E
     +1/x(2))/4-sqrt(x(2))/sqrt(360.0)*(0.3141593E1/x(2)/4-sqrt(360.0)/s
```

```
     +qrt(x(2)**3)*atan(sqrt(24-x(2))/sqrt(x(2)))/2+sqrt(360.0)/sqrt(x(2
     +))*(-1/(sqrt(24-x(2)))/sqrt(x(2))/2-sqrt(24-x(2))/sqrt(x(2)**3)/2)
     +/(1+(24-x(2))/x(2))-0.2E1/x(2)**2)/2
              zj(3,3) = sqrt(360.0)/sqrt(x(2))/(1+360*x(3)**2/x(2))
              zj(2,1) = 1
              zj(1,3) = 0
              zj(1,1) = -1.0/610.0-0.2623869E-2*exp(-0.247E-2*x(1))
              zj(3,1) = 0
              zj(2,3) = 0.4536E7*x(3)/(x(2)+360*x(3)**2)
              zj(2,2) = -0.1575E4*0.3141593E1/sqrt(x(2))/sqrt(360.0)*ata
     +n(sqrt(24-x(2))/sqrt(x(2)))-0.315E4*0.3141593E1*sqrt(x(2))/sqrt(36
     +0.0)*(-1/(sqrt(24-x(2)))/sqrt(x(2))/2-sqrt(24-x(2))/sqrt(x(2)**3)/
     +2)/(1+(24-x(2))/x(2))-0.7875E3*0.3141593E1/sqrt(24-x(2))/sqrt(360.
     +0)+0.63E4*(1/x(2)-(x(2)+360*x(3)**2)/x(2)**2)/(x(2)+360*x(3)**2)*x
     +(2)+0.315E4/x(2)-6*sqrt(360.0)/sqrt(24-x(2))/x(2)-12*sqrt(360.0)*s
     +qrt(24-x(2))/x(2)**2
              zj(1,2) = 1
              call resol(zj,f,3)
c
              do 1006, i=1,3
                x(i) = -f(i)+x(i)
 1006         continue
c
              f(1) = x(2)-3984.0/305.0-x(1)/610+324.0/305.0*exp(-0.247E-
     +2*x(1))
              f(2) = x(1)-0.315E4*0.3141593E1*sqrt(x(2))/sqrt(360.0)*ata
     +n(sqrt(24-x(2))/sqrt(x(2)))+0.1575E4*0.3141593E1*sqrt(24-x(2))/sqr
     +t(360.0)+0.63E4*alog((x(2)+360*x(3)**2)/x(2))-0.315E4*alog(24/x(2)
     +)+12*sqrt(360.0)*sqrt(24-x(2))/x(2)
              f(3) = atan(x(3)*sqrt(360.0)/sqrt(x(2)))-sqrt(x(2))/sqrt(3
     +60.0)*(0.3141593E1*alog(x(2)/24)/4+sqrt(360.0)/sqrt(x(2))*atan(sqr
     +t(24-x(2))/sqrt(x(2)))-0.2857143E-1+0.2E1/x(2))/2
            goto 1005
            else
c
c             WHILE LOOP TERMINATION :
c             BYPASSING THE MAXIMUM ITERATION NUMBER
              write(6,1007)
 1007         format(' maxwhile1 ')
            endif
c
c         NORMAL WHILE LOOP TERMINATION
          endif
c         WHILE LOOP END (1)
          write(6,1008) x
 1008     format((2x,e14.7))
      end
      real function znorm(f,m)
```

```
      real f(m)
      znorm = 0
c
      do 1009, i=1,3
        znorm = znorm+f(i)**2
 1009 continue
c
      znorm = sqrt(znorm)
      end
```

5 Example 2: A recursive function on a tree

It is possible to write FORTRAN program solving recursive problems.

But it can be very tedious and very complicated to do it when the recursion is complicated.

When the recursion is not so complicated, we can have other problems.

For example, we consider a binary tree with nodes labeled by a couple of integers (i, j). $(1, 1)$ is the root of the tree, $(2, 1)$ and $(2, 2)$ are the children nodes of the root and recursively $(i+1, 2j-1)$ and $(i+1, 2j)$ are the children nodes of node (i, j). Node (i, j) is at level i.

Assume we have the following sequence:

$$f_{1,1} \text{ given}$$
$$f_{i,j} = \begin{cases} g(f_{i-1,\frac{j}{2}}) & \text{if } j \text{ is even} \\ g(f_{i-1,\frac{j+1}{2}}) & \text{if } j \text{ is odd} \end{cases}$$

where g is a given function.

We want to compute the values of the sequence up to a given level N, i.e. the 2^{N-1} values $f(N,1) \ldots f(N,2^{N-1})$.

To write the corresponding FORTRAN code, you only have to write two loops:

```
      real f(n,m)
      do 1,i=1,n
        do 2,j=1,2**(n-1)-1,2
            f(i,j)=g(f(i-1,(j+1)/2))
 2      continue
        do 3,j=2,2**(n-1),2
            f(i,j)=g(f(i-1,j/2))
 3      continue
 1    continue
```

but the dimension `m` of the array `f` is 2^{N-1} ! We have to keep the storage for $N \times 2^{N-1}$ real values instead of $2^N - 1$ real values (5 times more storage for N equal to 10).

A way to solve this problem is to have an array for each level, i.e. to have in the FORTRAN program arrays `f1(1)`, `f2(2)`, `f3(4)` and so on. But now it is very tricky to write the FORTRAN program.

It is easy to generate this program with MACROFORT. We suppose that a FORTRAN function `g` has already been defined. The MAPLE function which generates the FORTRAN program is:

```
gen_func := proc(n)
  local i,j,pg;
  pg:=[];
```

```
# declaration of the arrays
  for i from 1 to n do
    pushe([declaref,real,[f.i(2^(i-1))]],'pg');
  od;
# loops for each array
  for i from 2 to n do
    pushe([dom,j,1,2**(i-1)-1,2,
           [equalf,f.i(j),g(f.(i-1)('(j+1)'/2))]],'pg');
    pushe([dom,j,2,2**(i-1)-1,2,
           [equalf,f.i(j),g(f.(i-1)(j/2))]],'pg');
  od;
  pg:=[programm,f,pg];
  writeto('func.f');
  init_genfor();
  genfor(pg);
  writeto(terminal);
end:
```

The generated FORTRAN program for N equal to 4 is:

```
      program func
        real f1(1)
        real f2(2)
        real f3(4)
        real f4(8)
c
        do 1000, j=1,1,2
          f2(j) = g(f1((j+1)/2))
 1000   continue
c
c
        do 1001, j=2,2,2
          f2(j) = g(f1(j/2))
 1001   continue
c
c
        do 1002, j=1,3,2
          f3(j) = g(f2((j+1)/2))
 1002   continue
c
c
        do 1003, j=2,4,2
          f3(j) = g(f2(j/2))
 1003   continue
c
c
        do 1004, j=1,7,2
          f4(j) = g(f3((j+1)/2))
```

```
 1004   continue
c
c
        do 1005, j=2,8,2
          f4(j) = g(f3(j/2))
 1005   continue
c
      end
```

Together with the previous problem of storage, we can have a much more complicated recursion. For example the g function can be dependent of the node of the tree where we compute the value of the sequence. Such a problem occurs in multiscale statistical signal processing when using wavelet transforms and we want to compute modeling filters by Schur-Levinson recursions (see [2]). Investigations to solve this problem are undertaken.

6 Conclusion

MACROFORT has been described. We have shown that it is possible to make FORTRAN code within MAPLE using it as a preprocessor for numerical analysis. Moreover MACROFORT has shown its capability for:

- solving numerical problems that MAPLE cannot manage
- doing mathematical computations before numerical computation
- solving storage problems in FORTRAN code for classes of recursive algorithms
- solving numerically recursive algorithms.

References

[1] Chancelier J.P., Gomez C. and Quadrat J.P., MACROFORT : a FORTRAN Code Generator in MACSYMA, MACSYMA Newsletter, 1987.

[2] Basseville M. and Benveniste A., Multiscale Statistical Signal Processing, INRIA report 970, 1989.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique