СООБЩЕНИЯ
ОБЪЕДИНЕННОГО
ИНСТИТУТА
ЯДЕРНЫХ
ИССЛЕДОВАНИЙ

Дубна

98-377

E2-98-377

M.N.Tentyukov*

AUTOMATION OF FEYNMAN DIAGRAM EVALUATIONS

*E-mail: tentukov@thsun1.jinr.ru

1998

1 Introduction

Recent high precision experiments require, on the side of the theory, high-precision calculations resulting in the evaluation of higher loop diagrams in the Standard Model (SM). For specific processes thousands of multiloop Feynman diagrams do contribute, and it turns out to be impossible to perform these calculations by hand. This makes the request for automation a high-priority task.

Several different packages have been developed with different areas of applicability (see a good review [1]). For example, FEYNARTS / FEYNCALC [2] are MATHEMATICA packages convenient for various aspects of the calculation of radiative corrections in the SM. There are several FORM packages for evaluating multiloop diagrams, like MINCER [3], and a package [4] for the calculation of 3-loop bubble integrals with one non-zero mass. Other packages for automation are GRACE [5] and COMHPEP [6], which partially perform full calculations, from the process definition to the cross-section values.

A somewhat different approach is persued by XLOOPS [7]. A graphical user interface makes XLOOPS an 'easy-to-handle' program package, but is mainly aimed to the evaluation of single diagrams. To deal with thousands of diagrams, it is necessary to use special techniques like databases and special controlling programs. In [8] for evaluating more than 11000 diagrams the special database-like program MINOS was developed. It calls the relevant FORM programs, waits until they finished, picks up their results and repeats the process without any human interference.

The package GEFICOM [9] developed for computation of higher order processes involving a large number of diagrams is based on cooperative usage of several software tools such as Mathematica, FORM, Fortran, etc.

It seems impossible to develop an universal package, which will be effective for all tasks. It appears absolutely necessary that various groups produce their own solutions of handling the problem of automation: various ways will be of different efficiency, have different domains of applicability, and last but not least, should eventually allow for completely independent checks of the final results. This point of view motivated us to seek our own way of automatic evaluation of Feynman diagrams.

Our first step is dedicated to the automation of the muons two-loop anomalous magnetic moment (AMM) $\frac{1}{2}(g-2)_\mu$. For this purpose the package TLAMM was developed [10]. The algorithm is implemented as a FORM-based program package. For generating and automatically evaluating any number of two-loop self-energy diagrams, a special C-program has been written. This program creates the initial FORM-expression for every diagram generated by QGRAF [11], executes the corresponding subroutines and sums up the various contributions. In the SM 1832 two-loop diagrams contribute in this case. The calculation of the bare diagrams is finished.

Our aim is to create some universal software tool for piloting the process of generating the source code in multi-loop order for analytical or numerical evaluations and to keep the control of the process in general. Based on this instrument, we can attempt to build a complete package performing the computation of any given process, at least in the framework of a concrete model.

2 Brief description

The project called DIANA (DIagram ANAlyser) for the evaluation of Feynman diagrams is being finished by our group at present.

For this project we had elaborated a special text manipulating language (TM). The TM language is a very simple TeX-like language for creating source code and organizing the interactive dialog.

The program reads QGRAF output. For each diagram it performs the TM-program, producing input for further evaluation of the diagram. Thus the program:

Reads QGRAF output and for each diagram it:

1. Determines the topology, looking for it in the table of all known topologies and distributes momenta according to the current topology. If we do not yet know all needed topologies, we may use the program to determine missing topologies that occur in the process.

2. Creates an internal representation of the diagram in terms of vertices and propagators corresponding to the Feynman integrand.

3. Executes the TM-program to insert explicitly expressions for the vertices, propagators etc. The TM-program produces input text for some program (FORM e.g.), and executes the latter (optionally).

Executing the TM-program provides apart from the possibility to calculate each diagram using FORM or another formulae manipulating language, to do some numerical calculation by means of FORTRAN, to create a postscript file for the picture of the current diagram, etc.

The program operates as follows: first of all, it reads its configuration file, which may be produced manually or by DIANA as well. This file contains:

1. The information about various settings (file names, numbers of external particles, definition of key words, etc.)

2. Momenta distribution for each topology.

3. Description of the model (i.e., all particles, propagators and vertices).

4. TM-program.

The TM-program is part of the configuration file. It starts with the directive

```
\begin translate
```

Then the program starts to read QGRAF output. QGRAF generates the diagrams in symbolic form in terms of vertices. Fig. 1. shows one of the diagrams with the corresponding QGRAF output.

For each diagram DIANA determines the topology, assigns indices and creates the textual representation of the diagram corresponding to the Feynman integrand. All defined data (masses of particles, momenta on each lines, etc.) are stored in internal tables,

```
. . .
*--#[ d2:
*
   1
   *vx(E(2),e(-1),H(1))
   *vx(E(3),e(4),A(-3))
   *vx(E(-2),e(5),H(6))
   *vx(E(5),e(3),H(1))
   *vx(E(4),e(2),H(6))
*
*--#] d2:
. . .
```

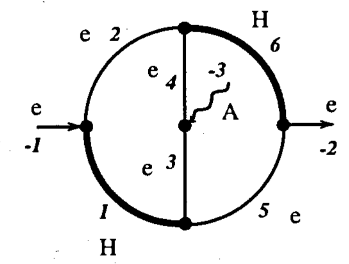


Figure 1: Diagram "number 2". e stand for fermion, E for anti-fermion, H is a scalar, and A the photon. External legs are numbered by negative numbers and vx stands for the vertex for the particles in the argument list.

and may be called by TM-program operators. At this point DIANA performs the TM-program. After that it starts to work with the next diagram.

When all diagrams are processed, the program may performs the TM-program a last time (optionally). This may be used to do some final operations like summing up the results.

Using of the TM language makes DIANA be very flexible. It is easy to build various algorithms of diagram evaluating by specifying settings in configuration file or by a TM program.

The typical flowchart may looks like fig. 2.

There is a possibility to use DIANA to perform TM-program only, without reading QGRAF output. If one specifies in the configuration file

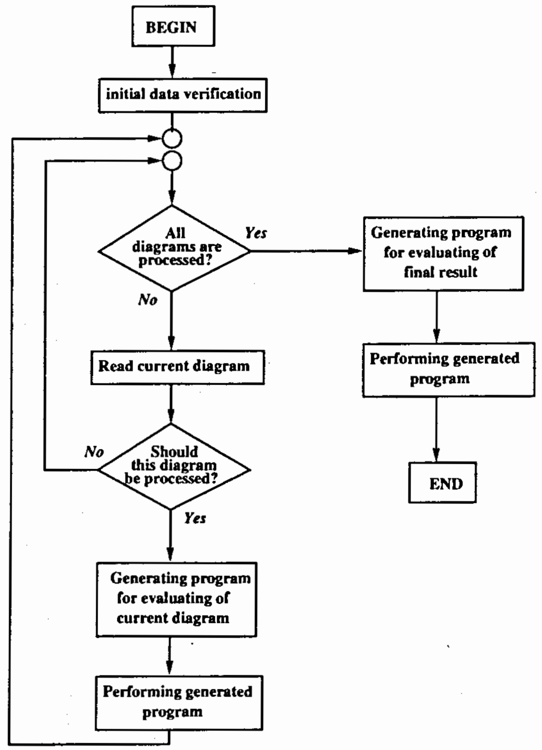
```
only interpret
```

then DIANA will not try to read QGRAF output, but immediately enters the TM - program.

DIANA contains powerful preprocessor. User can creates macros to hide complicated constructions. Now we have special macros to start DIANA. Similar as LaTeX provides the possibility for non-specialsts to typeset high-quality texts using TeX language, these macros permit DIANA to work at very high level. The user can specify the model and the process, and DIANA will generate all necessary files.

3 Topologies and momenta

Topologies are represented in terms of ordered pairs of numbers like (fromvertex,tovertex) (see fig. 3). All external legs have negative numbers. These are supposed to begin with -1. The fist numbers correspond to ingoing particles, the last numbers to outgoing ones. External legs must be connected with vertices of smallest possible identifying number.



BEGIN
initial data verification
All diagrams are processed?
Yes
No
Generating program for evaluating of final result
Performing generated program
END
Read current diagram
Should this diagram be processed?
No
Yes
Generating program for evaluating of current diagram
Performing generated program

Figure 2: Typical flowchart of Feynman diagram evaluations by Diana.

The number of an internal line corresponds to its position in the chain of pairs and the direction from the first to the second number in the pair:

direction:	$5 \to 1$	$1 \to 3$	$3 \to 4$	$4 \to 2$	$2 \to 5$	$5 \to 6$	$3 \to 6$	$4 \to 6$
$(-2,2)(-1,1)$	$(5,1)$	$(1,3)$	$(3,4)$	$(4,2)$	$(2,5)$	$(5,6)$	$(3,6)$	$(4,6)$
number:	1	2	3	4	5	6	7	8

Knowing thus the topology we can assign momenta. Their distribution according to fig. 3 e.g. is added like

```
topology A = (-2,2)(-1,1)(5,1)(1,3)(3,4)(4,2)(2,5)(5,6)(3,6)(4,6):
  p1,p2,p3,p4,p5,p6,p7,p8;
```

This fixes directions and values of all momenta on internal lines. External lines must be known from the process definition.

All topologies should be described in the configuration file. DIANA stores topologies in the internal table in some standard form. After reading new diagram DIANA defines its topology, reduces the topology to the standart form and searches the topology in the table. If DIANA fails in finding the topology, it will not produce the Feynman integrand for the diagram.

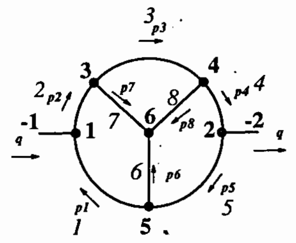


Figure 3: Topology (-2,2)(-1,1)(5,1)(1,3)(3,4)(4,2)(2,5)(5,6)(3,6)(4,6).

4 TM language

Similar to the TeX language, all lines without special escape - characters ("\") are simply typed to the output file. So, to type "Hello, world!" in the file "hello" we may write down the following program:

```
\program
\setout(hello)
Hello, world!
```

The asterisk in the beginning of line is a comment:

```
* This is the comment!
```

Each word the first character of which is the escape character will be considered as a command. This feature makes this language very easy-to-use. In the appendix A there is the example of a simple TM-program and the result of its performing.

DIANA's preprocessor is run on TM program before actual compilation. It permits the user to do textual substitutions (with parameters), to perform conditional compilation, etc. For example, the following preprocessor directive

```
\DEF(macroname)
. . .
\ENDDEF
```

defines the macrodefinition. After this directive you can just write

```
\macroname()
```

to invoke the macro. You can use macro with arguments:

```
\macroname(a,b,c)
```

Using macro, we are able to create high level macrolanguage similar to LaTeX is the set of macrodefinitions under TeX. The basic idea is similar to LaTeX, again. It is the *styles.* Instead of complicated TM programming you can use proper style. The style file containing all macrodefinitions just included in the begin of TM program.

5 Generating configuration files

Now we have several such styles oriented to FORM using. There is very useful style "create". Using this style, the user can specify the model and the process, and DIANA will generate all necessary files.

Here is the example of usage of this style.

Suppose, we have the following file "create.cnf" in current directory:

```
SET _processname = "zbb"
SET _qgrafname = "qgraf"
SET _syspath = "/home/U.Ser/diana/tml/"
system path = "/home/U.Ser/diana/tml/"
only interpret

\openlanguage(create.tml)
\Begin(program)

\Begin(model,gwsmassless.model)
\End(model)

\Begin(process)
ingoing Z(mu;p1);
outgoing b(;k1),B(;k2);
loops = 2;
\End(process)

\Begin(qgrafoptions)
options=onepi,nosnail;
\End(qgrafoptions)

\Begin(tmlprogram,form.prg)
\End(tmlprogram)

\indices(mu,mu1,mu2,mu3,mu4,mu5,mu6,mu7,
mu8,mu9,mu10,mu11,mu12,mu13,mu14,mu15,mu16,mu17,
mu18,mu19,mu20,mu21,mu22,mu23)
\vectors(p1,p2,p3,p4,p5,p6,p,q,q1,q2,k1,k2)

\End(program)
```

This file containes all necessary information to create files for DIANA to proceed the process $Z \to b\bar{b}$ in the frame of the standard model.

The following directive:

```
\openlanguage(create.tml)
```

is equal to two directives

```
\begin translate
\include(create.tml)
```

The file `create.tml` containes all macro definitions and subroutines.

The environment

```
\Begin(model,gwsmassless.model)
\End(model)
```

should define the model description. Instead of explicit definition in this example the file "gwsmassless.model" is used. This is the file containing the simplified Standard Model.

After the user types the command

```
diana -c create.cnf
```

DIANA starts to work. It generates several temporary files, invoke QGRAF, defines all topologies occur in the process..

Three new files appear in the current directory: "config.zbb", "qlist.zbb" and "settings.zbb". File "config.zbb" containes TM program, file "settings.zbb" containes various optional settings, topologies and momenta description, the model and the process definition. File "qlist.zbb" is the QGRAF output file. To start calculation of Feynman diagrams the user can enter the command

```
diana -c config.zbb
```

The file "config.zbb" containes the directive to include the file "settings.zbb", several settings for various options and the draft of a TM-program:

```
* This file is automatically generated by DIANA for the process zbb.
\include(settings.zbb)

*Remove the following line to refuse the protocol file generating:
log file = log.zbb

* Uncomment the following line to get debug information:
*debug on

extra call

\openlanguage(specmode.tml)
\Begin(program,routines.rtn)

\section(common,browser,regular)

\Begin(initialization)
  *This initialization for functions \sift() and \headstr():
  \setcheck(0123456789)
  \Begin(browser,not)
    \export(callform,\ask(\(Call form?(Y/N))))
  \End(browser)
\End(initialization)

\section(regular)
\Begin(output,\askfilename())
   **** d\counter()
   **** (diagram \currentdiagramnumber())
   \blankline()
   * Set here your defines!
   \Begin(foreach,i,1,\numberofinternallines())
      #define mm\i() "\mass(\i())"
      \blankline()
   \End(foreach)

   #define LINE "\numberofinternallines()"
   #define FERMIONLINE "\maxfcount()"
   #define TOPOLOGY "\sift(\topologyid())"
   #define TNAME "\headstr(\topologyid())"
   \blankline()

   functions \functions();
   commuting \commuting();

   1 Rq =\integrand()
   \blankline()
   * Here should be your FORM program!
   .end
\End(output)

\Begin(special,"\import(callform)"eq"Y")
   \execute(\(form -l )!.!)
\End(special)

\section(epilog)
\Begin(browser,not)
   \Begin(output)
     #define NMAX \counter()
   \End(output)
\End(browser)

\End(program)
```

The file "config.zbb" containes the draft of the TM-program. The user should specify all preprocessor definitions instead of the line

```
   * Set here your defines!
```

and set the body of the FORM program itself instead of the line

```
   * Here should be your FORM program!
```

Now the TM-program uses the macro - style "specmode":

```
\openlanguage(specmode.tml)
```

This style defines the special "sections" mechanism: a TM-program should consists of several "sections". Each of them is actually performed only under sertain conditions. So, the common structure of the TM-program looks like follows:

```
\Begin(program)
\section(browser)
   (contents of the section "browser")
\section(regular)
   (contents of the section "regular")
\section(epilog)
   (contents of the section "epilog")
\End(program)
```

Each section will be performed only under proper conditions.

The main section is `\section(regular)`. It is performed to create FORM input for each diagram.

There is possibility to run DIANA without FORM input generating just to obtain some information about diagrams. If the user start DIANA with the option `-l filename`, the program creates the file "filename" and stores in this file some information. The user can use the TM-program to do some actions with information about current diagram. The corresponding part of TM-program is the section

```
\section(browser)
```

The section

```
\section(epilog)
```

will be performed during "extra call" after all diagrams have been processed.

There is the special section `common`. It may accept several extra arguments. These extra arguments are names of the other sections. The "common" section is performed for all these sections. For example,

```
\section(common,browser,regular)
```

defines part of a TM-program wich should be performed both in "browser" and "regular" modes. The special environments are defined with names coincide with sections names. In the section

```
\section(common,browser,regular)
```

the environment

```
\Begin(browser)
 . . .
\End(browser)
```

will be performed only for "browser" mode. And, the environment

```
\Begin(browser,not)
 . . .
\End(browser)
```

will be performed for all modes except the "browser" mode, in this case just in "regular" mode.

The environment

```
\Begin(program,routines.rtn)
```

now containes the argument "routines.rtn". This is the name of a file containing various routines – TM-functions and macros. These are the following functions:
`\functions()` – outputs comma - separated list of all non-commuting functions occur in current diagram.
`\commuting()` – outputs the list of all commuting functions occur in current diagram.
`\sift(text)` – returns all digits occur in argument. For example,

```
\sift(qwerty2w3e4r5t6y78i)
```

returns "2345678".
`\headstr(text)` – returns the head of an argument untill first digit, for example

```
\headstr(qwerty2w3e4r5t6y78i)
```

returns "`qwerty`".

Both the functions `\sift(text)` and `\headstr(text)` accept the digits are set by the built-in operator `\setcheck(0123456789)`. This can be performed only one time in the beginning, before of actual processing of diagrams. That is why it is set in the environment

```
\Begin(initialization)
 . . .
\End(initialization)
```

In this environment the global variable "callform" is defined. The function `\ask(text)` outputs its argument to the screen, waits from the user entering "y" or "n" and returns "Y" or "N", respectively. This function is defined in the style file.

An output inside the environment

```
\Begin(output,filename)
 . . .
\End(output)
```

is directed into the file "filename". In this example, instead of explicit file name we use the macro `\askfilename()` defined in the style file. This macro asks the file name from the user.

For each diagram the file name "`aaa.bbb`" will be expanded to "`aaa#.bbb`", where `#` is the number of the current diagram. Thus, for diagram number 15 the file "`aaa15.bbb`"

will be created. Note, this expansion is performed by the environment, not by the macro `\askfilename()`.

The macro `\askfilename()` asks the file name only one time, in the case when the file name does not defined. After the file name definition the environment just uses the same file name. It is impossible to change the file name once entered.

In the section `epilog` this file name will be used as the output file name without changing.

The macro `\counter()` is expanded into the order number of the processed diagrams. It can differ from the number of diagram returned by the operator `\currentdiagramnumber()`.

The operator `\blankline()` returns the blank line. It must be used to obtain the blank line in produced output because all blank lines you just insert in your TM-program are suppressed.

There is the special environment `foreach` providing the cycles with parameters. A contents of the environment

```
\Begin(foreach,i,1,10)
\End(foreach)
```

will be repeated 10 times. Each time the macro `\i()` expands to the digit counter, i.e 1,2,...10.

The built-in operator `\numberofinternallines()` returns number of internal lines, the built-in operator `\maxfcount()` returns number of continuous fermion lines in current diagram.

The macro `\integrand()` will be expanded in the sequence of TM-operators returning the Feynman integrand for current diagram.

There is the environment `special`. It has the form

```
\Begin(special,<condition>)
 . . .
\End(special)
```

The contents of this environment will be performed only if `<condition>` evaluates in `True`.

In this example such environment is used to call the FORM interpreter if the user want to do it:

```
\Begin(special,"\import(callform)"eq"Y")
  \execute(\(form -l )!.!)
\End(special)
```

The macro `\execute()` using here expands the symbols `!.!` to the full file name. The expression `!.` is expanded to the file name without extension, and `.!` is expanded to the extension. The the macro `\execute()` call the built-in TM-operator `\system()` to execute the FORM. Thus, for the diagram number 15 e.g. the following command will be fulfilled:

```
form -l d15.frm
```

Here we assume the user has entered the file name "d.frm".

The section "epilog" uses the environment "output" without the second argument. The second argument is unnecessary because the file name is already defined.

In the "extra" call the environment "output" does use the file name exactly the user has specified answering the question asked by the macro \askfilename().

6 Conclusion

Any involved calculation in field theory necessarily consists of two parts: 1)automatic generation of Feynman diagrams and source codes and 2)techniques of handling the numerators and evaluating scalar Feynman diagrams.

In this paper we have concentrated on the first part. The software described has to be joined with some subroutines for analytical and/or numerical evaluating. At present, we are working under such conjugation.

Acknowledgements

This work has been supported by Russian Foundation for Basic Research, Grant 98-02-16923.

Appendix A. Example of simple TM-program

In this appendix we consider the "pure" TM-program *without* macrocoding. This should help us to understand some basis of the TM language.

Let us suppose that we use FORM as a formulae manipulating language.

The user types his FORM program and has the possibility to insert in the same line TM-operators as well. For example, the typical part of a TM program looks like follows:

```
\program
\setout(d\currentdiagramnumber().frm)
#define dia "\currentdiagramnumber()"
#define TYPE "\type()"
#define COLOR "\color()"
#define LINES "\numberofinternallines()"
\masses()
#include def.h
l  R=\integrand()
#call feynmanrules{}
#call projection{}
#call reducing{'TYPE'}
#call table{'TYPE'}
#call colorfactor{'COLOR'}
.sort
```

```
drop R;
g dia'dia' = R;
.store
save dia'dia'.sto;
.end
\setout(null)
\system(\(form -l )d\currentdiagramnumber().frm)
```

Some of the TM - commands are just TM-operators while some are functions (returning a value) written in the TM-language itself. The following operators are built-in TM operators:

`\numberofinternallines()` – returns the number of internal lines.
`\currentdiagramnumber()` – returns the order number of current diagram.
`\setout(filename)` – redirects output to the file "filename". There is the special file "null", output redirected to this file never appear. If empty string is used as argument of this operator, the output will appear at the terminal. This operator returns empty string.
`\system(cmd)` – call the operating system to perform the command "cmd". It returns some integer number.

An argument of one operator may containes arbitrary number of other operators. All spaces in the operator arguments are suppressed.

There is the special quotating operator without name, it just returns its argument without any changing: `\( any text)` returns the string "`any text`". In the above example it is used in the argument of the operator `\system()` to keep spaces.

The functions

```
\masses(), \type(), \color() ,\integrand()
```

are TM-language functions. Their code should be plased somewhere before the main program (the latter starts with the operator `\program`). Here is the example of the function "masses":

```
\function masses;\-\let(i,0)
\+\do
#define m\inc(i,1) "\mass(\get(i))"
\while "\numcmp(\get(i),\numberofinternallines())" eq "<" loop
\end
```

This function just outputs the string like

```
#define m1 "mmH"
```

for each internal line. The built-in operator `\mass(1)` returns the mass of particle at line 1.

The TM-language is a very simple language, and only one type of data exists. It is a string. All operators and functions return one text string. They may have one or more arguments, or have not arguments at all. Arguments are separated by commas. Each operator have fixed number of arguments.

Some string may be stored in variable. Operator `\let(var,val)` creates (or resets) variable `var` and stores `val` in it. Further operator `\get(var)` will return string `val`:

```
\program
\let(who,world)
\setout()
Hellow, \get(who)!
```

Note, the operator `\let(var, val)` returns `val`, not empty string!

There are several built-in operators for manipulating integer numbers. In the example of the function `\masses()` the operator `\inc(var,val)`increments value of `var` by `val` and returns new value, and the operator `\numcmp(a,b)` returns "<", ">" or "=", depending on the numerical values of `a` and `b`.

Variables defined by the operator `\let()` are local against TM-function. The user can defines global variables by means the operator `\export(var,val)`. Global variables exist during processing all diagrams. They can be obtained by the operator `\import(val)`.

Unlike built-in operators, the user - defined functions can produce some output to the current output file. The function `\masses()` e.g. returns empty string, but it produces several lines placing into output file. That is why we use derictives `\-` and `\+`: the former just suppress any output while the latter permits output, again.

The default value returned by function is empty string. Functions can return arbitrary string by means of the operator `\return(text)`.

There are several kinds of the "controlling constructions". They are used to change the stream of the control. These are well - known "if", "do", etc. In the function `\masses()` you can see how the post-condition cycle is used.

This TM-program will generate the FORM input for each diagram. For example, the corresponding part of the FORM program generated for diagram number 15 will be placed to the file "d15.frm" and looks like follows:

```
#define dia "15"
#define TYPE "4"
#define COLOR "3"
#define LINES "4"
#define m1 "mmH"
#define m2 "mmW"
#define m3 "mmW"
#define m4 "mmH"
#include def.h
l  R=
    1*V(1,mu1,mu,2)*(-i_)*em^2/2/s*V(2,0)*(-i_)*1/4*em^2/s^2*mmH/mmW*
    V(3,mu2,+q4-(+q3),1)*(-i_)*em/2/s*SS(1,0)*i_*VV(2,mu1,mu2,+q2,2)*i_*
    SS(3,2)*i_*SS(4,0)*i_;
#call feynmanrules{}
#call projection{}
#call reducing{'TYPE'}
#call table{'TYPE'}
#call colorfactor{'COLOR'}
.sort
drop R;
g dia'dia' = R;
```

```
.store
save dia'dia'.sto;
.end
```

After producing, this program will be performed by FORM due to the operator

```
\system(\(form -l )d\currentdiagramnumber().frm)
```

In this case the operator performs the command

```
form -l d15.frm
```

References

[1] R. Harlander and M. Steinhauser, TTP98-41, BUTP-98/28, hep-ph/9812357.

[2] J. Küblbeck, M. Böhm and A. Denner, Comp. Phys. Comm. 60 (1990) 165; R. Mertig, M. Böhm and A. Denner, Comp. Phys. Comm. 64 (1991) 345.

[3] S.A. Larin, F.V. Tkachov, J.A.M. Vermaseren, NIKHEF-H/91-18.

[4] L.V. Avdeev, Comp. Phys. Comm. **98** (1996) 15.

[5] T. Ishikawa *et al*, Minami-Taeya group "GRACE manual", KEK-92-19, 1993.

[6] E.E. Boos *et al*, SNUTP 94-116 (1994); (hep-ph/9503280).

[7] L. Brücher *et al*, Nucl.Instrum. Meth. **A389** (1997) 323; A. Frink, J.G. Körner and J.B. Tausk, (hep-ph/9709490).

[8] T. van Ritbergen et al., Int. J. Mod. Phys. **C6** (1995) 513.

[9] M. Steinhauser, BUTP-98/26, hep-ph/9811342.

[10] L.V. Avdeev, J. Fleischer, M. Yu. Kalmykov, M. Tentyukov, Nucl.Instrum. Meth. A 389 (1997) 343; *Towards Automatic Analytic Evaluation of Diagrams with Masses*, accepted for publication in Comp.Phys.Comm., (hep-ph/9710222); L.V. Avdeev, M.Yu. Kalmykov, Nucl. Phys. **B502** (1997) 419.

[11] P. Nogueira, J. Comput. Phys. **105** (1993), 279.

Received by Publishing Department
on December 25, 1998.