The MINOS running facility

Abstract

MINOS is a program that manages in a very primitive way a database that is use in a make-like fashion. Currently its main application is to run FORM programs for large numbers of Feynman graphs.

1 Running MINOS

The MINOS program is run like any regular UNIX utility. Its name is minos and the command to run it is 'minos filename' in which *filename* represents the name of a file which contains the commands for minos. The output consists of minos telling on the screen what it is doing. Minos may either do some things by itself, or decide to run external programs like FORM jobs. The output of these FORM jobs will of course not appear on the screen. On a system with more than one processor minos can run several jobs simultaneously. One can specify this with the -w option as in 'minos -w4 filename' to run 4 processes at the same time. Another way to achieve this is by defining the environment variable MINOSPROCESSORS as in

```
    setenv MINOSPROCESSORS 4
```

after which 4 will be the default number of simultaneous processes.

2 MINOS commands

The input file of MINOS consists of lines, each of which is either a command or commentary. A line is commentary if its first non-blank character is either %, or # or or an end-of-line. Hence an empty line is seen as commentary. Also lines after the 'end' command are considered as commentary. Currently the commands that are recognized are

- 'end' This statement ends the execution of the MINOS job.
- 'input' Tells which database is the input database. The name of the file is to be specified between double quotes as in 'input "gjgjdia.dat"'. It is strongly advised to use the extension '.dat' for the names of the databases.
- 'output' Tells which database is the output database. The name of the file is to be specified between double quotes as in 'output "gjgjdia.dat"'. It is strongly advised to use the extension '.dat' for the names of the databases.
- 'system' Send whatever follows to be executed as a system command. Example: 'system ls -l'.
- 'load' This command is meant to setup a database from a textfile. The proper syntax is

  'load "into" "from" "ITEMNAME" "itemseparation"'

  in which "into" refers to a database that has been declared as the output database, "from" is a text file that has not been declared in an input statement, "ITEMNAME" is the name that the subobjects will get in the database. Each object contains subobjects that each have a name. More subobjects can be built in later commands. The itemseparator is a string that identifies where an item is finished and the next item starts. Alternatively the itemseparator can be replaced by the word fold (without the double quotes around it). In that case it is assumed that the items are inside folds, which are defined in the same way as for FORM. In the case of folds, the program also assumes that lines of the type '#define VAR "value"' indicate a separate subobject with the name VAR and the contents 'value', in which 'value' can be any string. In the case of the fold option there is also another subobject. It has the name NAME and its value is the name of the fold.

- ‘make’ This is the command that is most useful. It treats the objects in the input database and converts them into objects in the output database, depending on a time stamp. Hence it acts like a UNIX make, except for that it is more primitive. The syntax is:

  ‘make option ”input” ”output” (OUTVAR) [min-max] NAME=value’

  The option can take several values, and the action depends on this value. ”input” is the name of the input database (which has been declared already. There is some doubling of information here), ”output” is the output database (same). OUTVAR is the name of the subject with the output. The range min-max is optional and should be enclosed between braces. It refers to the number of the objects that are considered for action. It is like an extra control. Other restrictions on the objects to be considered are the specification of a NAME and a value. NAME should be the name of a subobject and value its value. Only objects that contain a subobject with the proper name and value will be considered. If more than one of these names is specified the various ‘NAME=value’ fields should be separated by comma’s. The action specified by ‘option’ is

  0: The input objects should contain the subobjects named DIA, TOPO, COLOR, FLAVOR. If there is a subobject NAME it is used as a name in a FORM file diagram.h which will contain the statements ‘Local NAME=value-of-DIA;’ and ‘#define TOPO ”value-of-TOPO”’. Otherwise the name is derived from the number of the object in the database. The file diagram.h is created and filled, and then the command

  ‘form -t /tmp calcdia > calcdia.log’

  is run. This assumes that there is a file calcdia and that the program FORM is present in a way that it can be located. The output file calcdia.log is read and the output expression is placed in the output database as the value of the subobject with the name OUTVAR which is defined in the make command. All other subobjects are copied, except for the subobject DIA. In addition there are two new subobjects defined. One has the name EXTIME and its value is the execution time of the FORM program, and the other has the name DISKSPACE and its value is the maximum diskspace in bytes that the FORM program used for its expressions. If the call of form should be different, the user should consult the chapter on ‘If the user wants more’.

  1-4: This command expects the presence of the subobjects GRAF and SF in the input objects. In addition it will use the subobject with the name NAME. If this subobject is not present a name will be generated, based on the number of the object in the input database. The program will create a file convert.h and put in it the value of GRAF and the FORM statement ‘#define SF ”value-of-SF”’. It will then run FORM with the command

  ‘form -si -d PROCESS=value-of-option convdia > convdia.log’

  assuming that FORM can be located this way and that a proper program convdia exists for it. This program can pick up the file convert.h and the value of PROCESS. The output file convert.log is analyzed and the values of the following expressions are taken from it: ‘colour’ to be put in a subojbject named COLOR, ‘topology’ to be put in a subobject named TOPO, ‘expr1’ to be put in a subobject value-of-OUTVAR. If OUTVAR is not defined, its name will be DIA. It also looks for the expression ‘flavour’ to put it in FLAVOR. The absence of either of these expressions is a fatal error.

- 'sum' This statement can combine the results of the objects in either the input or the output database. The syntax is

  'sum option "database" [min-max] NAME=value'

  in which the treatment of the restriction fields [min-max] and NAME=value is as in the make statement, the number of the option indicates the specific action, and database is the name of the database with must have been declared with either an input or an output statement. The specific action is as a function of 'option':

  0: For each object the value of VALUE is multiplied by the value of COLOR and placed in the file totalinp.h. This files gets occasional .sort statements to prevent input overflows in FORM. Then, when all objects to be considered have been put in this file FORM is invoked with the statement:

  'form total > total.log'

  This assumes the presence of a file total that will pick up the file totalinp.h and does something with it. MINOS will not do anything with the output of this run.

  1: The values of the subobjects with the name EXTIME are added for the indicated objects. The sumvalue is printed.

  2: The values of the subobjects with the name DISKSPACE are compared for the indicated objects. The maximum value is printed.

- 'show' This command is used to dump the contents of a database in a user readable form. The syntax is:

  'show "database" [min-max] `@` (VAR1,...,VARn) NAME=value'

  in which [min-max] and NAME are treated as in the make statement to put a restriction on the objects that are shown. The database must have been declared before in either an input or an output statement. If the symbol `@` is present the timestamp is also printed. If the field with VAR1 till VARn is present only the indicated subobjects are printed.

- 'touch' Modifies the timestamp in the indicated database to the current time. This is used to force the program to redo objects that have been done before. The syntax is

  'touch "database" [min-max] NAME=value' in which [min-max] and NAME are treated as in the make statement to put a restriction on the objects that are touched. The database must have been declared before in either an input or an output statement.

- 'linsum' Currently not properly supported. Still being developed.

In old versions the executable of FORM had to be specified with an absolute pathname. In the new version this is different:

1. The executable of FORM is called form (no absolute pathname). This means that if this executable can be found (by means of the path variable) no further modification is needed.

2. If there is an environment variable FORM its contents will be used as the executable of FORM. Example:

```
setenv FORM /user/t68/formdir/ver3/form
```

would make minos look for that file for FORM.

3 If the user wants more

If the user is not pleased with the action of MINOS, and in particular if the calling statements for FORM are not exactly what is needed, the user can modify one of the files shell.c (to define completely new commands) or user.c (to modify some of the existing commands in the make or sum family. The calls of FORM are to be found near or in the system commands that do the execution. Hence searching for the string "system" and looking at these lines or the lines just before them will be sufficient. If more fundamental changes are necessary, the user should first study the source code.

4 Known bug

When the -w parameter is used in starting MINOS and several processes are started, each running FORM it could happen that these processes start so quickly that FORM gets a bit confused and creates two .str files with the same name. This is more a FORM problem, but because it isn't easy to solve the user better knows about this.