Proteum/IM - Version 1.1 C
User's Guide

Márcio Eduardo Delamaro [1]
José Carlos Maldonado [2]

January 11, 2002

[1]Doctoral level student at IFSC-USP, São Carlos - SP, Brazil. Financially supported by CAPES
[2]Titular professor at ICMSC - USP, São Carlos - SP, Brazil. Supported by CNPq.

1 Objective of this Report

Proteum/IM is a mutation based tool that consists of independent functional modules (programs) enabling experienced users to apply test trhough shell scripts, reducing the amount of interaction between the user and Proteum and saving time. *Proteum/IM*, unlike previous versions of Proteum, implements the *Interface Mutation* criterion [**?**]. Interface Mutation is an inter-procedural adequacy criterion, suitable to use at integration testing level.

This report describes the programs that make up *Proteum/IM* version 1.1 and the way users can use them to create testing sessions through scripts. It's assumed a previous knowledge on mutation testing and Proteum facilities. We suggest the user to read [**?**] before this document.

A test session in *Proteum/IM* is characterized by a database that basicaly described test cases and mutants. There are two groups of programs, described in this guide. The first, called "basis" programs act on this database to execute some specific operations. The second, called "utility" programs, use the first to perform more elaborated functions. For example, there is a basic program *muta* that creates an empty mutant database and another (*tcase*) to create an empty test case database. To create a test session (characterized by a mutant and a test case database), there exist a utility program, called *test-new* that invocates *tcase* and *muta* to create an empty test session. First we describe the basis programs and the parameters they accept. After, we present the utility programs and finally, we give a complete script example.

2 Basis Programs

In this section we describe the basis programs of *Proteum/IM*: *li*, *li2nli*, *ptest*, *instrum*, *tcase*, *muta*, *opmuta*, *exemuta*, *check-equiv* and *report*. Using these programs the tester can write down test scripts making the test activity more efficienty, less error prone and repeatable.

Some of these programs need to consult "system" files that should be in a know directory. Such directory is identified to these programs by setting the environment variable PROTEUMIMHOME to the directory where those files are. If, for example, Proteum is installed in directory /home/me-user/proteum/bin, the following command must be executed before using Proteum 1.2 programs:

```
setenv PROTEUM12HOME /home/me-user/proteum/bin
```

Next sections describe the programs:

2.1 li

NAME

li - transforms a C program into a intermediate representation called LI

SYNOPSIS

li [-l] [-P filename] [-D directory] source-filename LI-filename

DESCRIPTION

When generating mutants, Proteum uses a intermediate representation of source program called LI. A LI program identifies major elements like statements, declarations and expressions in the original source file. Program *li* is used to create a such file. **source-filename** is assumed to have suffix **.c**. and **LI-filename** is created with suffix **.li**. In addition, a file with **.fun** is created, containing one record for each function in the source file, given some characteristics of that function. Also creates a file **LI-filename.cgr** that represents the call-graph for the functions in the source file. The **.li** and **.fun** are required in next steps of *Proteum/IM*testing. The **.cgr** file is not used by *Proteum/IM* but may be usefull to other tools.

OPTIONS

-l

By default, li lists at stdout the names of all functions in the source file and the LOC's of each of them. Using -l, only the names are present, separated by a newline character.

- P filename

When -P is used, li calls a C preprocessor called proteumIMcpp before creating LI program. In this case, **source-filename** is preprocessed originating filename. Then, li reads this new file and creates corresponding LI file.

-D directory

Establish the directory where the source file is and where preprocessed file and LI file are created. If this option is not used, the current directory is assumed.

EXAMPLES

li -P _myprog myprog _myprog

Preprocess the source file myprog.c, creating preprocessed file _myprog.c (option -P). Reads this file and produces _myprog.li, _myprog.fun and _myprog.cgr.

li -P _myprog -D /usr/work/MYPROG myprog _myprog

The same as above but determines that the files are in the directory /usr/work/MYPROG.

2.2 li2nli

NAME

li2nli - creates the program graph and adds information about program graph nodes to the intermediate representation (LI program).

SYNOPSIS

li2nli [-D directory] LI-filename

DESCRIPTION

Program *li2nli* takes a LI program specified by **LI-filename** and creates, for each function on the program, a file that describes the corresponding program graph; these files have the name of the original functions with the suffix **.gfc** In adition, creates a file with the same **LI-filename** and the suffix **.nli** that includes information about program graph nodes in the LI program. The existence of **.nli** file is mandatory to the next step of the test; **.gfc** files are not used by *Proteum/IM*.

OPTIONS

-D directory

Establishes the directory where the LI file must be and where the .gfc and .nl i files are created. If this option is not used, the current directory is assum ed.

EXAMPLES

li2nli myprog

Takes file myprog.li and creates files myprog.nli, main.gfc and myfunc.gfc (supposing that main and myfunc are the only functions in myprog.li)

NOTES

This program uses POKE-TOOL's **chanomat** program. So, when calling li2nli, access to chanomat must be available.

2.3 ptest

NAME

ptest - creates and handles program test files

SYNOPSIS

ptest -create [-S filename] [-E filename] [-D directory] [-C command] [-test | -research] test-name

ptest -l [-D directory] test-name

DESCRIPTION

Program *ptest* is used to create files that describe a program test. In this files there is general data like: name of source and executable program file being tested, name of the functions being tested, etc. Parameter **test-name** determines the name of test file. It's created with suffix **.PTM**.

OPTIONS

-create

Determines that the operation to be performed is to create a new test file

-l

Lists the data in the existing program test file

-S filename

Determines the name of the source program file that will be tested. If this option is not used, the **test-name** is assumed as the source file. The suffix **.c** is appended to the filename, it means, the name provided by the tester should not contain the suffix.

-E filename

Name of executable program file. If this option is not used, the **test-name** is assumed as the name of the executable file.

-D directory

Determines the directory where the source file and the executable file are and where the program test file will be created.

-C command

Determines how to build the executable file from the source file. In general this command have more than one word, so it must be enclosed between double quotes ("). If this options is not provided, "gcc source-filename -o executable-program -w" is assumed.

-test

Determines that the program test type is "test". This determines the Proteum's behavior while executing mutants.

-research

Determines that the program test type is "research". Can't be used in conjunction with option -test

EXAMPLES

ptest myprog

Creates in current directory a program test file (myprog.PTM) with the following data:

Test name: myprog

Source file: myprog.c

Exec. file: myprog

Compilation command: gcc myprog.c -o myprog -w

ptest -S myprog -E myprog -C "cc myprog.c -o myprog -w -g" myprog-1

Creates in current directory a program test file (myprog-1.PTM) with the following data:

Test name: myprog-1

Source file: myprog.c

Exec. file: myprog

Compilation command: cc myprog.c -o myprog -w -g

2.4 instrum

NAME

instrum - instruments a C file to produce of log of path execution.

SYNOPSIS

instrum [-EE filename] [-D directory] source-filename

DESCRIPTION

Program *instrum* is used to create a C file that, when executed with a test case **t**, behaves exactly as the original file, given by **source-filename**.c but in addition, produces a file with the list of nodes travessed. This instrumented program is created with the name **source-filename** appended with the suffix **_inst.c** This program requires the existence of, besides the original source (.c) file, the correspondent **.nli** file.

OPTIONS

-D directory

Establishes the directory where the source file and the **.nli** file are and where the instrumented file will be created. If this option is not used, the current directory is assumed.

-EE filename

Determines the name of the log file to be produced by the instrumented program. If this option is not used, the name of the log file is the same of the original source file with the suffix **.LOG.TMP**.

EXAMPLES

instrum myprog

Creates in current directory a C program called myprog_inst.c that is similar to the original myprog.c but when compiled and executed produced a log of the execution path. This log will be store in the file myprog.LOG.TMP

instrum -EE myprog-1 myprog

The same as above, but the instrumented program, when executed, produces the file myprog-1.LOG.TMP.

2.5 tcase

NAME

tcase - creates and handle test case database.

SYNOPSIS

tcase -create [-D directory] TC-filename

tcase -l | -d | -e | -i [-t n] [-f n] [-x list] [-D directory] TC-filename

tcase -add [-DE directory] [-E filename] [-EE filename] [-p parameter | -P] [-trace] [-D directory] TC-filename

tcase -proteum [-DD directory] [-I filename] [-DE directory] [-E filename] [-EE filename] [-t n] [-v c] [-f n] [-trace] [-D directory] TC-filename

tcase -poke [-DD directory] [-DE directory] [-E filename] [-EE filename] [-t n] [-f n] [-trace] [-v c] [-D directory] TC-filename

tcase -ascii [-DD directory] [-I filename] [-DE directory] [-E filename] [-EE filename] [-t n] [-f n] [-v c] [-trace] [-D directory] TC-filename

tcase -z [-D directory] TC-filename

DESCRIPTION

With program *tcase* you can create a test case database and insert, delete, enable, disable and import test cases. Program *tcase* creates and uses two files: one with suffix **.TCS** and other with suffix **.IOL**. Parameter **TC-filename** identifies the name of test cases database being created/updated.

OPTIONS

-create

Creates a new test case database.

-add

Inserts a new test case in the database.

-l

Shows one or some test cases.

-d

Deletes one or some test cases from the database.

-e Enables one or some test cases.

-i

Disables one or some test cases.

-proteum
Imports one or some test cases from another proteum test case database.

-poke
Imports one or some test cases from a POKE-TOOL test case set.

-ascii
Imports one or some test cases from ASCII files.

-z
Each time a test case is delete from test case database, its content is not phisically removed from the files. Option -z do this operation, removing the deleted records from the test case database.

-D directory
Indicates the directory where the test case database is or will be created. If this option is not used, the current directory is assumed.

-DD directory
Indicates the directory where the file(s) that contains the test cases being imported is (are). If this option is not used, the same directory of test case database is assumed

-I filename
Determines the name of the file that contains the test cases being imported. If this option is not used, **TC-filename** is assumed for the name of the importing file.

-DE directory
Indicates the directory where the executable files (see option -Eand -EE) are. If this option is not used, the same directory of test case database is assumed

-E filename
Program *tcase* reads only input data from importing file. The test case output is determined by executing a program over those data. This option determines the name of such program (an executable file). If this options is not used, **TC-filename** is assumed.

-EE filename
This option is used in conjunction with option -trace. It determines the name of the executable file to be executed to produce the log of path execution for the test case being inserted/imported (see program *instrum*). If -trace is used and -EE option is not, the **TC-filename** is assumed as the name of the program.

-f n
n determines the number of the first test case on which the selected operation is going to be applied. Test cases are numbered from 1.

-t n
n determines the number of the last test case which the selected operation is going to be applied.

-x list
Determines a list of test cases which the operation is going to be applied. Since this list might have more than one number, it should be enclosed in double quotes (") in order to protect it from the shell.

-P

Initial parameters are also used to characterize a test case. With this option program *tcase* interactively asks the user for supplying those parameters.

-p parameter

With this option, the test case parameters is supplied in the command line. If neither -p nor -P options are used, no initial parameter is assumed.

-trace

Determines that the information about path execution should be included as part of the test case.

-v c

Verbose mode. Determines that a for each test case imported, a charactere 'c' should be displayed (writen to the stdout file), so the tester can follow the execution of the operation

EXAMPLES

tcase -create myprog-1

Creates an empty test case file called myprog-1.

tcase -add -DE ../exec -E myprog -p "-la -d" myprog-1

Starts inserting a test case in database myprog-1. Program ../exec/myprog is executed and the user interactively provides test case input. Initial parameters are "-la -d".

tcase -ascii -DE ../exec -E myprog -DD ../test-set -I case -p param -t 1 -f 200 myprog-1

Test cases are imported from ASCII files ../test-set/case1 to ../test-set/case200 that store the input (stdin) for the test cases and from ../test-set/param1 to ../test-set/param200 that store the initial parameters. Each par of file has one test case. Program ../exec/myprog is executed using input data from these files, generating output data.

tcase -ascii -DE ../exec -E myprog -EE myprog_instr -trace -DD ../test-set -I case -p param -t 1 -f 2

The same as above, but the program myprog.inst is executed to generate the log of execution path for the test case. This program has been created (see program *instrum*) such that it produces a log file called myprog-1.LOG.TMP, which is readen by program *tcase* and stored as part of the test case.

2.6 muta

NAME

muta - creates and handles mutant database.

SYNOPSIS

muta -create [-D directory] mutant-filename

muta -add [-D directory] mutant-filename

muta -l | -equiv | -nequiv | anomalous | nanomalous [-t n] [-f n] [-x list] [-D directory] mutant-filename

muta -ms [-D directory] mutant-filename

DESCRIPTION

Program *muta* permits users to make some operations in a mutant database. Such database stores mutants descriptors: the way Proteum describes changes done in the source program aiming to create mutant programs. Three files are used to store descriptors: an index file (suffix **.IND**), and two descriptor files (suffix **.MUT** and **.???**). Parameter **mutant-filename** identifies the target mutant database.

OPTIONS

-create

Creates a new mutant database.

-add

Inserts new descriptors. The data needed to characterize the mutation is read through stdin. See also program *opmuta* since its output is used as *muta -add* input.

-l

The operation to be performed is to show one or some mutant descriptors. The output is an unformated one, it means, just the data in descriptors are shown, not the mutated program.

-equiv

The operation to be performed is to set one or some mutants as equivalents.

-nequiv

The operation to be performed is to set one or some mutants as non-equivalents.

-anomalous

The operation to be performed is to set one or some mutants as anomalous.

-nanomalous

The operation to be performed is to set one or some mutants as non-anomalous.

-ms

The operation performed is only the display of the current Mutation Score, store in the mutant database

-D directory
Indicates the directory where the mutant database is or will be created. If this option is not used, the current directory is assumed.

-f n
n determines the number of the first mutant on which the selected operation is going to be applied. Mutants are numbered from 0.

-t n
n determines the number of the last mutant on which the selected operation is going to be applied.

-x list
Determines a list of mutants on which the selected operation is going to be applied.

EXAMPLES

muta -create -D myprog-test myprog-1
Creates a mutant database called myprog in directory ./myprog-test

muta -add -D myprog-test myprog-1
Starts the insertion of new descriptors in file **myprog-1**. The data for descriptors are read from stdin. A way to do that is creating a pipe from program *opmuta* output to program *muta* input. See program *opmuta* and *muta-gen* for more details.

muta -l -D myprog-test -f 10 myprog-1
Shows mutant descriptors from number 0 to number 10.

muta -equiv -D myprog-test -x "11 13 16 23" myprog-1
Sets mutants numbered 11, 13, 16 and 23 as equivalent mutants.

2.7 exemuta

NAME

exemuta - executes, selects or builds mutant source files.

SYNOPSIS

exemuta -exec [-T timeout] [-f n] [-t n] [-trace] [-v c] [-Q n] [-H directory] [-D directory] test-name

exemuta -select [-f n] [-t n] [-<operator> n] [-DD directory] [-O filename] [-c funcname1 funcname2] [-all n] [-k] [-D directory] test-name

exemuta -build [-x list] [-e] [-H directory] [-D directory] test-name

exemuta -update [-D directory] test-name

DESCRIPTION

This program has four main functions. The first one, using flag -exec is used to execute some or all mutants generate in a program test. Parameter **test-name** identifies the name of an already created test. So, before using program exemuta you must create a test file, a test case database and a database of mutation descriptors. Option -select permits you to choose mutants that will remain active. The others are marked as inactive and do no influence at computing mutation score. Option -build creates a file that stores the source code of a selected mutant or a set of selected mutants. Option -update re-calculate the mutation score stored in the mutant database, without re-executing the mutants. This is usefull when the mutants are executed in parts; at the end, *exemuta -update* computes the resulting score.

OPTIONS

-exec

Executes mutants.

-select

Selects mutants.

-build

Creates a mutant source file.

-update

Updates the mutation score without executing the mutants.

-D directory

Indicates the directory where the test file, test case file and mutation descriptor file are. If this option is not used, the current directory is assumed.

-f n

n determines the number of the first mutant on which the selected operation is going to be applied. Mutants are numbered from 0.

-t n

n determines the number of the last mutant on which the selected operation is going to be applied.

-x list

Determines a list of mutants that are going to be in the source file being created.

-Q n

When a mutant source file is built by *Proteum/IM* several mutants are 'joined' in a single file. Option -Q determines the maximum number of mutants that should be joined. The default is 100 but this number can produce a large source file. Tester can lower this number in order to avoid disk space consumption.

-T timeout

When a mutant is created, the mutation can make the program enters an infinite loop. When executing a mutant, program exemuta uses a value to calculate the maximum amount of time a mutant can run. The default amount is 5 times the time spent by original program with the same test case. Option -T change this default to **timeout** times the original program execution time.

-trace

Indicates that *Proteum/IM*should use the log of path execution stored for each test case to try to avoid the execution of mutants

-H directory

Program exemuta uses a header file "proteum.h" to build a mutant source file. This flag indicates where this file is. If this flag is not used, proteum uses directory set in environmental variable PROTEUM12HOME.

-<operator> n

Determines that n% of generated mutants from a specified mutation operator are going to be pined as active.

-all n

Determines that n% of generated mutants from all mutation operators are going to be pined as active.

-O filename

Determines that filename stores the list of operators and respective percentages to be used to select mutants.

-DD directory

Determines the directory where the file specified in option -O is. If This option is not used, the same directory of the test file is assumed.

-c funcname1 funcname2

Determines that only mutants generated for connection funcname1–funcname2 are going to be kept active

-k

This option indicates that the selection operation should consider only the active mutants. In other words, the inactive are kept inactive and the operation is applied on the active set

-e

Builds also executable mutant file.

-v c

Verbose mode. Determines that a for each mutant executed, a charactere 'c' should be displayed (writen to the stdout file), so the tester can follow the execution of the operation

EXAMPLES

exemuta -exec -f 900 -t 1000 -v + myprog-1

Executes mutants from 900 to 1000 of program test named myprog-1. For each mutant executed, a '+' is displayed.

exemuta -select -all 100 -strp 0 myprog-1

Selects 100% of all mutants, except those generated by STRP mutation operator. For that operator, none mutants are left active (0%).

exemuta -select -k -all 100 -ssdl 0 myprog-1

If used after the above example, result in a mutant set which have all mutants active, except those generated by operators STRP and SSDL.

exemuta -build -x "0 1 2 3 4 5 6 7 8 9" -e myprog-1

Creates a file, named muta0_myprog-1.c, that contains the source file relative to mutants 0 to 9.

2.8 opmuta

NAME

opmuta - applies mutation operators to source file

SYNOPSIS

opmuta [-<operator> n m] [-all n m] [-DD Directory] [-O filename] [-c function-name1 function-name2 [-D directory] source-filename LI-filename

DESCRIPTION

This program applies mutation operators to a C source file and produces as output the descriptions needed to build mutant programs. **source-filename** is the name of the file that is going to be mutated and **LI-filename** is the corresponding LI file. As output, opmuta produces descriptions of mutations in a format that program *muta* (option -add) is able to read and include in mutation descriptors database.

OPTIONS

-<operator> n m

Determines that mutants of **<operator>** mutation operator is going to be created. The name of the operator can be abreviated. For example, using the name "I-DirVar", the tester refers to all the mutation operators which name begins with this string, such as I-DirVarRep and I-DirVarInc. The parameter **m** determines that at most m mutants of this operator should be created on each mutation point. And **n** determines that n% of the mutants (already considering the selection determined by **m**) are generated. If **m** is 0, all possible mutants are created. If **n** is 0, none mutant is created.

-all n m

Determines that mutants from all mutation operators are going to be created.

-O filename

The mutation generation parameters are stored in file filename that is a normal ASCII format file. The example bellow is a valid file:

I-DirVarRep 100 2 I-IndVarRep 10 2

These three flags (-<operator>, -all and -O) can be used together and are interpreted in the sequence they appear in the command line.

-D directory

Indicates the directory where the source and the LI files are. If this option is not used, the current directory is assumed.

-DD directory

Indicates the directory where the file that contains generation parameters is. If this option is not used, the same directory of source and LI file is assumed.

-c calling-function called-function

Specifies the connection that is going to be mutated. More than one -c option can be used. If this option is not used, all connections are mutated. Either calling or called function names can be

replaced by "-all". If the calling (called) function name is -all it means that any connection that has the specified called (calling) function should be mutated.

-r

Determines that only connections to functions in the source file should be mutated. In this case, calls to extern or system functions (like *printf* and *scanf*) are not mutated.

EXAMPLES

opmuta -all 50 0 __myprog __myprog

Uses a source file called __myprog.c and a LI file called __myprog.nli to generate 50% of mutants from all operators.

opmuta -all 10 2 -II- 0 0 __myprog __myprog

The number of mutants for all the operators is restricted to 2 for each mutation point. From those mutants 10% are actually generated. For aperatos of group II (see the list of operators in the Appendix), none mutant is generated.

2.9 report

NAME

report - builds a file with a report about test cases effectiveness.

SYNOPSIS

report -tcase [-L n] [-S filename] [-D directory] test-filename

DESCRIPTION

This program is used to create a report with information about test cases. The user can select what information she wants to see. **test-filename** is a program test name.

OPTIONS

-tcase

Determines the kind of reports going to be created. At now, only test case report is available.

-D directory

Indicates the directory where program test files are. If this option is not used, the current directory is assumed.

-S filename

The report is stored in a file called **test-filename.lst** by default. Option -S determine another name to the report file.

-L n

Indicates the report level of details. The number n is a combination of some flags that indicate what will be present in the report. The flags are:

1 Only effective test cases are shown

2 Shows the number of not executed mutants for each test case

4 Shows the number of alive executed mutants for each test case

8 Shows the number of dead mutants for each test case, for each causa-mortis

16 Shows the total number of dead mutants for each test case

32 Indicates if mutant is enabled or disabled

64 Shows execution time, return code and initial parameters of each test case

128 Shows input for each test case

256 Shows output for each test case

In order to determine the items that are going to be in the report, you have to sum the chosen flags. If -L option is not used, default level 17 is used. It means, the report shows the number of dead mutants, only for affective test cases.

EXAMPLES

report -tcase myprog-1

Creates a file myprog-1.lst that stores a report containing the number of mutants that each test case has killed. Only those test cases that have killed at least one mutant are shown.

report -tcase -L 511 myprog-1

Creates a complete report, with all possible information but only for effective test cases.

report -tcase -L 510 myprog-1

Creates a complete report for effective and non-effective test cases.

2.10 check-equiv

NAME

check-equiv - mark mutants as equivalent, based on execution information.

SYNOPSIS

check-equiv [-r n] [-f n] [-t n] [-D directory] test-filename

DESCRIPTION

This program chooses and marks alive mutants as equivalents. The mutants chosen are those executed with highest number of test cases. It is important that mutants have been executed (program *exemuta*) using option -trace. In thi way, program *check-equiv* can take in consideration only the test cases really used to execute a mutant, not those which execution has been avoided.

OPTIONS

-r n

Parameter **n** determines the ceiling for the number of mutants the tester wants to mark as equivalent. For example, "-r 20" means that the tester wants to mark the mutants such that 20% are equivalent. Mutants already marked equivalent are considered to compute the number of mutants that should be marked.

-D directory

Indicates the directory where the test file is. If this option is not used, the current directory is assumed.

-f n

n determines the number of the first mutant on which the operation is going to be applied. Mutants are numbered from 0.

-t n

n determines the number of the last mutant on which the operation is going to be applied.

EXAMPLES

check-equiv -r 20 myprog-1

Mark mutants such that 20% of them are equivalents

3 Utility Programs

The basis programs described in the previous section are very flexible in the parameters and options they accept. But sometimes it's hard to create and run a test, due the number of programs invocation and parameters the tester must provide. For example, to create a program test called myprog-1 for executable program myprog, and corresponding source myprog.c, the following sequence is required:

```
ptest -create -S myprog -E myprog -C "cc myprog.c -o myprog" myprog-1
li -P __myprog myprog __myprog
li2nli __myprog
tcase -create myprog-1
muta -create myprog-1
```

The first line creates a test file called myprog-1.PTM that describes the program test. Second and third lines create the .li and .nli files; the name used for these files are __myprog. Fourth line creates the test case file and last line creates a file of mutation descriptors.

Proteum has some utility programs that help tester to execute a standard sequence of program calls like that above. These utility programs actually don't implement any functionality. They only take some parameters and pass them to the basis programs. The next sections present these utility programs: *test-new*, *tcase-add*, *muta-gen* and *muta-view*.

4 Creating a Program Test

This first utility program is used to create all files need to run program a test; it means: program test file (.PTM), intermediate language files (.li and .nli), test case files (.TCS and .IOL), mutants files (.IND and .MUT) and pre-processed source file.

This program, called *test-new*, accepts the same flags and parameters that program *pteste* accepts with option -create. It calls all basis programs needed at creating a program test. For example the command

```
test-new -S myprog -E myprog -C "cc myprog.c -o myprog" myprog-1
```

can be used to create a program test called myprog-1 for executable program myprog, and corresponding source myprog.c. It corresponds to the following sequence of commands:

```
ptest -create -S myprog -E myprog -C "cc myprog.c -o myprog" myprog-1
li -P __myprog myprog __myprog
li2nli __myprog
tcase -create myprog-1
muta -create myprog-1
```

In this way, tester has created and initialized all needed the files for a test.

5 Inserting Test Cases

Program *tcase* works on test case files. It does not need that tester has created other work files. Due to this, the tester must supply some parameters like the name of executable program when inserting or importing test cases. But if the tester has already created a program test file, these data are there. So, there are some utility programs that use these data, freeing the tester from supplying it.

Program *tcase-add* has the format:

tcase-add ¡options¿ test-filename

Options are the same options that program tcase -add accepts unless option -E. The executable file name is got from test file. For example, given the commands

test-new -S myprog -E myprog -C "cc myprog.c -o myprog" myprog-1
tcase-add -p "prog param" myprog-1

the second line causes the following program call:

tcase -add -E myprog -p "prog param" myprog-1

6 Generating mutants

Program *opmuta* is used to create mutant descriptors; it doesn't require the presence of all the work files, only source and intermediate files. Program *muta* is used to insert descriptors in the mutant database. The typical way to call these programs is:

opmuta ¡options¿ source-filename nli-filename | muta -add mutant-filename

There is a utility program that executes this sequence of commands. Program *muta-gen* has the format:

muta-gen <options> test-filename

where <options> are the same options that program *opmuta* accepts. For example, using the following commands,

test-new -S myprog -E myprog -C "cc myprog.c -o myprog" myprog-1
muta-gen -all 100 2 myprog-1

the tester has the second line changed by:

opmuta -all 100 2 _myprog _myprog | muta -add myprog-1

In this way its easier to use *muta-gen* than *opmuta* and *muta* separately.

7 Visualising mutants

The last utility program is used to visualize mutants. Program *muta-view* displays the main information of the mutants, one at each time. Figure **??** shows the basic operation of this program.

8 A complete example

References

[del95] M. E. Delamaro and J. C. Maldonado, Proteum User's Guide - Version 1.1 - C , Technical Report in preparation.