Proteum -Version 1.2 C Running Program Tests Trough Shell Scripts

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

March 20, 1999

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

1 Objective of this Report

Proteum version 1.2 C is a mutation based tool that consists of independent functional modules (programs) characterized from Proteum version 1.1 C, aiming at enabling experienced users to apply test trhough shell scripts, reducing the amount of interaction between the user and Proteum and saving time.

This report describes the programs that make up the Proteum tool version 1.2 C and the way users can use them to create testing sessions through scripts. It's assumed a previous knowledge on mutation testing and Proteum fecilities. We suggest the user to read [**?**] before this document.

First we describe the programs and the parameters they accept. After, we present some programs built over this basis programs to facilitate creating and runing testing sessions, and finally, we give some script examples.

2 Basis Programs

In this section we describe the basis programs of Proteum version 1.2 C: li, li2nli, ptest, tcase, muta, exemuta, opmuta 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 consult files that should be in a know directory. Such directory is identified to these programs by setting the environment variable PROTEUM12HOME 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
```

2.1 li

NAME

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

SYNOPSIS

li [-l] [-P filename] [-D directory] [-H 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 having suffix **.c**. and **LI-filename** is created with suffix **.li**.

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, just the names are present, separated by a newline character.

- P filename

When -P is used, li calls C preprocessor cpp before creating LI program. In this case, **source-filename** is preprocessed originating filename. Then, li reads t this new file and creates corresponding LI file. When used in a Proteum test, the preprocessed filename and li filename must be **__source-filename**, where **source-filename** is the name of the source file the user is testing. See examples below.

-D directory

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

-H directory

Program li uses some tables that describe C language. This tables are read fro m files in the directory set in environment variable **PROTEUM12HOME**. Option -H de termines other directory to search for those files.

EXAMPLES

li -P __myprog myprog __myprog

Preprocess the source file myprog.c, creating preprocessed file __myprog.c. Reads this file and produces __myprog.li

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

The same as above but determines that the work 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** Also creates a file with the same **LI-filename** and the suffix **.nli** that includes information about program graph nodes in the LI program.

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 and main.gfc (supposing that main is the single function 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] [-all | -u function-name, ...] [-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 are 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 program test file

-S filename

Determines the name of the program source 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.

-E filename

Name of executable program file. If this option is not used, the **test-name** is assumed as 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 t his command have more than one word, so it must be enclosed between double quotes ("). If this options is not provided, "cc source-filename -o executable-program -w" is assumed.

-all

Determines that all the functions in the source file are going to be tested.

-u function

Specifies the names of functions that are going to be tested. This options can not be used with option -all. More than one -u option can be used.

-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 with option -test

EXAMPLES

ptest myprog

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

Name: myprog Source File: myprog.c Exec. File: myprog
Compilation Command: cc myprog.c -o myprog Functions: all

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

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

Name: myprog-1 Source File: myprog.c Exec. File: myprog
Compilation Command: cc myprog.c -o myprog -w -g Functions: main

2.4 tcase

NAME

tcase - creates and handle test cases file.

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] [-p parameter | -P] [-D directory] TC-filename

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

tcase -poke [-DD directory] [-DE directory] [-E filename] [-t n] [-f n] [-D directory] TC-filename

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

DESCRIPTION

With program tcase you can create a test case file 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 file being updated.

OPTIONS

-create

Creates a new test case file.

-add

Inserts a new test case.

-l

Shows one or some test cases.

-d

Deletes one or some test cases.

-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 file.

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

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

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

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

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

-DE directory
Indicates the directory where the executable file (see option -E) is. If this option is not used, the same directory of test case file is assumed

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

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

-f 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 shell.

-P
Initial parameter is 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.

EXAMPLES

tcase -create myprog-1

Creates an empty test case file called myprog-1.

tcase -add -DE ../exec -E myprog -p 'init parameter' myprog-1

Starts inserting a test case in file myprog-1. Program ../exec/myprog is executed and the user interactively provides test case input. Initial parameters is "init parameter".

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

Test cases are imported from ASCII files ../test-set/case1 to ../test-set/case200. Each file has one test case. Program ../exec/myprog is executed using input data from these files, generating output data.

2.5 muta

NAME

muta - creates and handles files of mutant descriptors.

SYNOPSIS

muta -create [-D directory] mutant-filename

muta -add [-D directory] mutant-filename

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

DESCRIPTION

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

OPTIONS

-create

Creates a new mutants file.

-add

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

-l

The operation to be performed is to show one or some mutant descriptors. The o utput is an unformated one, it means, just the data in descriptors are shown, n ot the mutated program.

-equiv

The operation to be performed is to set an equivalent mutant.

-nequiv

The operation to be performed is to set an non equivalent mutant.

-D directory

Indicates the directory where the file of descriptors is or will be created. If this option is not used, the current directory is assumed.

-t n

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

-f n

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

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

EXAMPLES

muta -create -D myprog-test myprog-1
Creates a mutant file 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** 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.

2.6 exemuta

NAME

exemuta - executes, selects or builds mutants

SYNOPSIS

exemuta -exec [-T timeout] [-f n] [-H directory] [-D directory] test-name

exemuta -select [-f n] [-<operator> n] [-all n] [-D directory] test-name

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

DESCRIPTION

This program has three 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 file and a file 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.

OPTIONS

-exec

Executes mutants.

-select

Selects mutants.

-build

Creates a mutant source file.

-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.

-t n

This flag is used with option -build. Indicates that the source file being created contains only one mutant numbered n.

-x list

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

-f n

Indicates that mutants 0 to n are going to be executed or selected. The other mutants are not considered on the operation.

-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.

-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.

-e

Builds also executable mutant file.

EXAMPLES

exemuta -exec -f 1000 myprog-1

Executes mutants from 0 to 1000 from program test named myprog-1

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 -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.

2.7 opmuta

NAME

opmuta - applies mutation operators to source file

SYNOPSIS

opmuta [-<operator> n] [-all n] [-DD Directory] [-O filename] [-H directory] [-u function-name, ...] [-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 file.

OPTIONS

-<operator> n

Determines that n% of mutants from <**operator**> mutation operator is going to be created.

-all n

Determines that n% of mutants from all mutation operators is going to be created.

-O filename

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

STRP 100 Varr 50 SSDL 0

These three flags can be used together and are interpreted in the sequence they appear in the command line. At least one of them must be 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 generating percentages is. If this option is not used, the same directory of source and LI file is assumed .

- H directory

Program opmuta uses some tables that describe C language. This tables are read from files in the directory which name is set in environment variable PROTEUM12HOME. Option -H determines other directory to search those files.

-u function

Specifies the names of functions that are going to be mutated. More than one -u option can be used. If this option is not used, all functions are mutated.

EXAMPLES

opmuta -all 50 __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 100 -strp 0 -stri 0 __myprog __myprog

Generates 100% of mutants from all operators except STRP and STRI that generate no mutants.

2.8 report

NAME

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

SYNOPSIS

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

DESCRIPTION

This program is used to create a report having information about test cases. U ser can select what information he wants to see. **test-filename** is a program test name. The report is stored in a file called **test-filename.lst**

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.

-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 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.

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 passing the tester must provide. For example, to create a program test called myprog-1 for executable program myprog, and c orresponding 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 iprogram 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 programs calls like that above. These utility programs actually don't implement any functionality; they just take some parameters and pass them to basis programs. The next sections present these utility programs: **test-new**, **tcase-add** and **muta-gen**.

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 ipre-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 don't 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 to supply 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

Following this same idea, there are three other programs to import test cases. These programs are called **tcase-proteum**, **tcase-poke** and **tcase-ascii**. They are called as program **tcase** but do not require executable file name.

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 mutants files. The typical way for calling 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 -strp 0 myprog-1

the tester has the second line changed by:

opmuta -all 100 -strp 0 __myprog __myprog | muta -add myprog-1

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

7 Scripts

This section has a script example of how to use Proteum 1.2. Figure 1 shows this script and comments about it follow.

```
#############################################################################
# Script to test program FIND
set NOME=FIND
set COMP="cc find.c prog31.o -o a.out"
$COMP
foreach SETOP (A B C D E F )
   @ dno = 1
   while ($dno <= 3)
echo rm $NOME-$SETOP$dno'*'
    rm $NOME-$SETOP$dno*

echo --------- test-new --------------
    test-new -S find -E a.out -C "cc find.c prog31.o -o a.out" \
                                          -u find $NOME-$SETOP$dno

echo --------- tcase  - import --------------
    tcase -ascii -E a.out -DD con$SETOP-$dno -I case -t 1 -f 200 $NOME-$SETOP$dno

echo --------- muta-gen --------------
       muta-gen -DD .. -O con$SETOP.op $NOME-$SETOP$dno

echo -------------- exemuta ----------------------------
  exemuta -exec -T 5 $NOME-$SETOP$dno

echo --------- report --------------
    report -tcase $NOME-$SETOP$dno
@ dno = $dno + 1

   end
end
```

Figure 1: An Example of Test Script

In this script, program FIND is tested with six distinct sets of mutation operators and for each one, with three test case sets. Variable SETOP controls the operator set and variable dno controls the number of test case set. Let's follow the initial sequence of commands produced by this script:

```
set NOME= FIND
set COMP="cc find.c prog31.o -o a.out"
cc find.c prog31.o -o a.out          # compiles and links the program
@ dno = 1
echo rm FIND-A1'*'
rm FIND-A1*                           # deletes all files name FIND-A1...

echo --------- test-new -----------   # creates program test named FIND-A1
test-new -S find -E a.out -C "cc find.c prog31.o -o a.out"  -u find FIND-A1

echo ----- tcase  - import -----    # import test cases from subdirectory
                                    # test cases are stored in files case1, ... case200
tcase-ascii  -DD conA-1 -I case -t 1 -f 200 FIND-A1

echo ----- muta-gen ------------- # generates mutants
                                  # operators being used are described in file ../conA.op
muta-gen -DD .. -O conA.op FIND-A1

echo -------------- exemuta ---------- # executes mutants against imported test cases
exemuta -exec -T 5 FIND-A1

echo --------- report --------------   # creates a report about test cases
report -tcase FIND-A1
@ dno = $dno + 1                    # go to next test cases set, from subdirectory conA-2
                                    # continues with conA-3, conB-1, ..., conF-3
```

Internal loop (command while) uses variable dno to control the number of test case set being used. Its value goes from 1 to 3. External loop (command foreach) sets the value of variable SETOP which value can be 'A', 'B', 'C', 'D', 'E' or 'F'; this variable determines the mutation operator set being used. With this scripts it's easy to test FIND in 18 different ways (6 mutation operators sets times 3 test cases sets).

Aiming a comparison, Figure 2 shows a similar script that does not use utility programs, only basis programs.

References

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

```
##########################################################################
# Script to test program FIND
set NOME=FIND
set COMP="cc find.c prog31.o -o a.out"
$COMP
foreach SETOP (A B C D E F )
   @ dno = 1
   while ($dno <= 3)
     echo rm $NOME-$SETOP$dno'*'
     rm $NOME-$SETOP$dno*

     echo ------- li -------------------
     li -P __$NOME-$SETOP$dno find __$NOME-$SETOP$dno

     echo --------- pteste --------------
     pteste -create -S find.c -E a.out -C "cc find.c prog31.o -o a.out" \
                                              -u find $NOME-$SETOP$dno

     echo --------- tcase --------------
     tcase -create $NOME-$SETOP$dno

     echo --------- muta --------------
     muta -create $NOME-$SETOP$dno

     echo --------- tcase  - import --------------
     tcase -ascii -E a.out -DD con$SETOP-$dno -I case -t 1 -f 200 $NOME-$SETOP-$dno

     rm x
     echo --------- opmuta --------------
     opmuta -DD .. -O con$SETOP.op __$NOME-$SETOP$dno __$NOME-$SETOP$dno

     echo --------- muta -add --------------
     muta -add $NOME-$SETOP$dno <x
     rm x

     echo -------------- exemuta ---------------------------
     exemuta -exec -T 5 $NOME-$SETOP$dno

     echo --------- report --------------
     report -tcase $NOME-$SETOP$dno
     @ dno = $dno + 1

   end
end
```

Figure 2: Script Using Only Basis Programs