Fatmah Assiri
fyassiri@gmail.com
March 1, 2015

This is a documentation to run MUT-APR. I included instructions to configure the machine to run MUT-APR, solutions to some issues I faced developing and studying MUT-APR, and steps to fix faults.

*- MUT-APR folder contains:
 - MUT-APR code which includes: cdiff.ml, coverage.ml, faultLoc.ml, modify.ml, mutator.ml, normalization.ml, stats2.ml, README.

 - replace folder (example of a faulty program) which contain: faulty programs (e.g.,repalce.c), test inputs and test output files of faulty program (if applicable), passing tests script (test-good.sh), failing tests script (test-bad.sh), script to run MUT-APR 100 times for the faulty program (test-replace.sh),
scripts to create text files contains statement IDs when running passing tests (CreateGpaths.sh), scripts to create text files contains statement IDs when running failing tests (createNpaths.sh), script to run fault localization techniques and create LPFS when different faulty localization are used (runallfault.sh)

*- MUT-APR folder after finishing all steps below will also contain: instrumented version of faulty program (replace-coverage.c), compiled version (replace-coverage), files are needed by MUT-APR (replace.i.ht, replace.i.idfun, replace.i.ast), text files that are created to collect coverage information by running CreateGpaths.sh and CreateNpaths.sh (Gpath files, Npath files, replace.i.goopath, replace.i.path), at the end MUT-APR generates: replace.i-.debug which is a text file contains summary information about the tool execution, replace.i-baseline.c which is an instrumented version of the faulty program that contain the fault, (if faulty is fixed) MUT-APR generate a new version of the faulty program (replace.i-best.c) where fault is fixed and this version must pass the passing and failing tests that used in test-good.sh and test-bad.sh.

A. Machine configuration to get MUT-APR to work:

1- MUT-APR runs successfully on Linux.

2- MUT-APR written in Ocaml. It works successful with OCaml 3.12.1 and 4.00.1.
  http://caml.inria.fr/ocaml/release.en.html

3- MUT-APR required a CIL library that used to parse the source code and produce the AST. I worked with Cil-1.4.0.

http://www.cs.berkeley.edu/~necula/cil/
To get CIL working should do

```
cd cil
./configure
make
make cillib   //this step is more important than "make"
```

4- Changes to the makefile in MUT-APR folder
at the top of the folder (after OCAML_OPTIONS) add:

```
CIL = /s/chopin/a/grad/fatmahya/GenTech/cil-1.4.0cil  (add the path to CIL folder)
```

*** If running make file shows an error related to one of OCaml libraries, you should check the command to run modify, coverage, cdiff, and mutator that contains the following four libraries: nums.cmxa unix.cmxa str.cmxa cil.cmxa

5- To compile and run MUT-APR code, navigate to folder where MUT-APR codes are (where Makefile locates), and type "make".

B. Issues might occur:

1- When you run MUT-APR code and an error message rapper related to the incompatibility between Ocaml and CIL
Error message: “the files /usr/lib64/ocaml/arg.cm1 and obj/x86_LINUX/ciloptions.cm1   male inconstant assumptions over interface Arg"
You should do the followings:

```
cd cil
make clean
./configure
make
make cillib
```

C. To fix faults in C source code using MUT-APR you should have:

1- The Source file

2- Test Scripts contain a set of passing and failing tests and write test scripts: test-good.sh and test-bad.sh.
test-good.sh: contains all passing tests
test-bad.sh: contains all failing tests
Examples of test scripts are exist in the folder contains the faulty program inside MUT-APR folder (E.g., inside replace folder)

3- Create instrumented version of faulty programs by:

```
  cd faultyProgramFolder
  ../coverage --fixfun --mt-cov faultyProggram.c > faultyProgram-
coverage.c
  ex: ./coverage --fixfun --mt-cov replace.c > replace-coverage.c

 *** If this is does not work, preprocess source file of faulty
program by doing
  gcc -E replace.c > replace.i
  Then replace.i is used in all the steps (starting from step 3)
  ex: ./coverage --fixfun --mt-cov replace.i > replace-coverage.c

4- Compile the instrumented version by
   gcc -o replace-coverage replace-coverage.c

5- run test inputs on the compiled instrumented version
  ./repalce-coverage <testinput>
   ex: replace-coverage "fffff"< inpfile
 - This step will create a text file contains the statement IDs that
are executed by the test input.
 - Each text file should be named Gpath# (# is a number, for example
Gpath1) when passing test input is used and Npath# when failing test
is used. Then Gpath files are combined to     create one text file
called faultyProgram.i.goodpath, and Npaths are combined to create one
text file called faultyProgram.i.path.
 - I created test scripts to run all passing and failing tests on the
coverage code and returns the Gpath file, Npath files, and created the
*.i.path and *.i.goodpath files: CreateNpaths.sh and CreateGpaths.sh.
 - Example of the test scripts are inside the faulty program folder
inside MUT-APR folder

6- run a fault localization technique to create the list of
potentially faulty statements:
 ../faultLoc --pass <number of passing tests> --fail <number of
failing tests> --fl <fault localization technique name> Gpath1
Gpath2 .... Gpathn Npath1 Npath2 .. Npathm > LPFS1

7- To fix faults, run modify as follow:

 cd faulty program
 ../modify --faultLoc --gen <number of generation> --pop <number of
population> --mut 0.01 --max <Maximum fitness Value> faultyProgram.c

        *** number of generation and number of population can be any
integer number. For example, when you want to run the search algorithm
for one time, gen = 1
        *** mut is the mutation rate. This is always equal to 0.01
because we did not study different values of mutation rates.
        *** max is the maximum fitness value that determines if the
generated variant is a repair or not. max is an integer value = Number
of passing tests *1 + Number of failing tests *10
```

*** if you prepossessed the source file if the faulty program in step 3 to create faultyProgram.i, then use the processes faultyProgram in "modify" step (E.g., instead of using faultyProgram.c use faultyProgram.i)

*** For more details on how to run MUT-APR and command lines, you can check GenProg documentation in the following link http://dijkstra.cs.virginia.edu/genprog/
MUT-APR built by adapt GenProg. Thus, MUT-APR can run using GenProg instructions. However, MUT-APR adapt the interface to support different fault Localization technique, and this is different from GenProg. So running "modify" and coverage is slightly different with MUT-APR.

D. Fault Localization code:

1- If you need to add/remove/edit a fault localization technique, you should modify the code in faultLoc.ml.

2- To compile faultLoc.ml: ocaml str.cma -o faultLoc faultLoc.ml

---