Project Location: /home/fse/PathComplexity
PAC => Path Complexity Analyzer
CFG => Control-Flow Graph

Quick start.

- Navigate to /home/fse/PathComplexity/scripts.
- Run ./simple_tests to get a quick idea of how things work.
- This processes the example control flow graphs in cfgs/simple_test_cfgs.
- Output is saved to results/simple_test_results.csv.
- You can view the results in LibreOffice. You may have to adjust column width. To view a CFG file you can run xdot. Try the following:

xdot ~/PathComplexity/cfgs/simple_test_cfgs/vlab_cs_ucsb_test_SimpleExample_test4_0_basic.dot

Overview.

There are two major steps:

1. Extract CFGs from binaries.
   This is done with a python script iterating over java CFG extraction program.
2. Compute cyclomatic, npath, and path complexities for each CFG.
   This is done with a handful of Mathematica scripts with paths.m as the main driver.

==========
Detailed instructions.

Directory Contents.

---
scripts/cfgexctractor

Contains scripts for extracting control-flow from java bytecodes. You can use that script to generate graphs for new inputs. Given a folder it searches for .jar files in any level and generates the CFGs as .dot files.

```
usage: python main.py -i path/to/a/folder/that/contains/jar/file(s)
ex   : python main.py -i
              home/fse/PathComplexity/lib_jars/apache_commons/bins/commons-cli-1.2/

output: generates .dot files (under scripts/cfgextractor/outputs/)
```

Once you have the new control-flow graphs, you use complexities* script to get path complexity results.

```
usage: ./scripts/complexities /path/to/cfg/directory
```

---
lib_jars

Contains Apache commons libraries and Oracle Java 7 runtime libraries used in the experiments. Subfolders under lib_jars contains .jar files for the libraries. cfgextractor/ program navigates through the folders and generated CFGs for each jar file in a separate folder. (We have included

generated CFG files under cfgs/ directory.)

---
scripts:

Contains 6 scripts for demonstrating computation of path complexity, cyclomatic complexity, and NPATH complexity. Also contains sub directory ‘cfgextractor’ to generated CFGs.

These scripts work on precomputed CFGs located in ~/PathComplexity/cfgs/

Output of all scripts contains

- testnumber
- filename
- cyclomatic complexity
- npath complexity
- path complexity classification: Constant, Polynomial, or Exponential
- path counting function
- asymptotic complexity

1. complexites*:

Computes results for a single cfg file or entire directory and outputs a line of comma separated results to sdtout for each file.

```
usage:  ./complexities /path/to/cfg/file
        ./complexities /path/to/cfg/directory
output: stdout
```

2. simple_tests*:

Computes results for small set of six artificial CFGs in cfgs/simple_tests.

```
usage :  ./simple_tests
output:  csv file results/simple_test_results.csv
         stdout
```

3. apache_small_sample*:

Computes results for a sample of ~800 apache CFGs in fgs/apache_cfgs/commons/beanutils-1.9.2/. Running time is approximately 5 minutes.

```
usage:   ./apache_small_sample
output:  csv file results/apache_small_sample_results.csv
         stdout
```

4. java_small_sample*

Computes results for a sample of ~800 java CFGs in cfgs/java_cfgs/alt-rt/. Running time approximately 5 minutes.

```
       usage:  ./java_small_sample
       output: csv file results/apache_small_sample_results.csv
               stdout
```

5. apache_all* (data used in the paper):

Computes results for all apache CFGs in the data set cfgs/apache_cfgs/. Running time approximately 3 hours.

```
      usage: ./apache_all
      output: csv file results/apache_all_results.csv
               stdout
```

6. java_all* (data used in the paper):

Computes results for all apache CFGs in the data set cfgs/java_cfgs. Running time approximately 10 hours.

```
       usage: ./java_all
       output: csv file results/java_all_results.csv
               stdout
```

---
results

Output folder for results from running scripts.

Contains directory 'expected', which has all of the results of running the scripts precomputed.

You can use scripts/cfgextractor/Classify.py to get the classification done on Table 4 column 1. The rest of the columns and the Figure 6 is computed based on that.

```
      usage: pythong Classify.py -i <input_to_file_or_folder>
      output: .csv files for each classification (under scripts/cfgextractor/outputs)
```

---
cfgs

Precomputed control flow graph files in graphviz dot format. These control-flow graphs were generated using cfgextractor with the inputs lib_jars/apache_commons and lib_jars/jre_lib. There are made available here for reference or to test the path complexity scripts.

To view a CFG use:
$xdot path/to/cfg/file

---

src

Contains Mathematica scripts for computing complexities.

- paths.m is the main driver.
    Loads filenames, loads and calls modules.
- pathComplexity.m
- cyclomaticComplexity.m
- nPathComplexity.m
- utils.m
- cfgextractor_source.jar used to extract CFGs.

To run on a directory of CFGs

ex:
/usr/local/Wolfram/Mathematica/10.0/Executables/math -script
/home/fse/PathComplexity/src/paths.m <path/to/cfgs> Infinity

ex to output to a file
/usr/local/Wolfram/Mathematica/10.0/Executables/math -script
/home/fse/PathComplexity/src/paths.m <path/to/cfgs> Infinity > output.csv

The parameter 'Infinity' is the nesting depth of the directories. This will recursively compute over all subdirectories.

Replicate experiments.

1 - run scripts/apache all* and scripts/java_all*.
2 - (Optional) run scripts/cfgextractor/ to generate all CFGs with the inputs under lib_jars/
3 - (Optional) run Classify.py to get classified results for the tables and figures in the paper.

Running on new input.

1 - use scripts/cfgextractor to extract CFGs from .jar files
2 - use scripts/complexities to run complexity analysis on CFGs
(Please refer to corresponding sections for example usages of scripts)