

实验报告--基于变异测试的模糊器评估 (Fuzz-Mut)

1. 小组成员

2. 过程报告

2.1 选题描述

2.1.1 AFL过程

2.1.2 mull过程

2.1.3 衡量指标

2.2 主要工作:

2.3 环境、实验设置

2.4 Fuzzing 配置

2.5 实验整体过程

2.5.1AFL工作流程

2.5.2AFL结果

2.5.3对AFL用例筛选

2.5.4 变异测试理解

2.5.5 变异测试运行示例

2.5.6 差分测试过程

2.5.7 差分测试结果

readelf

optdump

size

nm

cxxfilt

2.5.8 差分测试结果说明

2.6 实现、配置和流程上的坑

2.6.1 模糊测试更改编译器环境

2.6.2 模糊测试插桩失败

2.6.3 模糊测试运行错误

[2.6.4 模糊测试并无make 之后并无可执行文件](#)

[2.6.5 变异测试检测并未插桩](#)

[2.6.6 变异测试运行select脚本出错](#)

[2.6.7 变异测试显示没有变异体](#)

3. 结果分析

3.1 模糊测试过程及变异测试结果的绘图结果

[cxxfilter](#)

[nm](#)

[objdump](#)

[readelf](#)

[size](#)

[strip](#)

[xpdf](#)

3.2 结论

4. 目录结构

1. 小组成员

姓名	学号
苏致成（组长）	201250104
陈凯文	201250198
谢学成	201250208
孙宇鹏	201850100

2. 过程报告

2.1 选题描述

2.1.1 AFL过程

AFL对原程序进行模糊测试产生的用例分为两类：

1. 一类会让原程序报错，这类测试用例存于crashes文件夹中。
2. 另一类探明了原程序新的执行路径，这类测试用例存于queue文件夹中。

2.1.2 mull过程

按照定义好的变异算子，mull 先对源程序变异，得到变异体，利用变异体去衡量AFL产生的测试用例质量，分为三类情况：

1. AFL产生的测试用例会使变异体崩溃（变异体被杀死）
2. AFL产生的测试用例会使变异体输出和原程序不一样的结果（差分测试思路，同样导致变异体被杀死）
3. AFL产生的测试用例会使变异体输出和原程序一样的结果(变异体存活)

2.1.3 衡量指标

变异得分，即为所有被杀死的变异体除以变异体总数。变异得分越高则表明 AFL 效果越好。

注意：由于mull对每个变异体的执行过程高度封装，难以进入到子进程内部去修改运行逻辑，因此我们将情况1和情况2分别进行实验。

2.2 主要工作：

projects名称	afl过程	mull过程	differential comparing	绘图过程
readelf	完成	完成	完成	完成
objdump	完成	完成 (含有final)	完成	完成
cxxfilt	完成	完成 (含有final)		完成
nm	完成	完成 (含有final)	完成	完成
size	完成	完成 (含有final)	完成	完成
strip	完成	完成 (含有final)		完成
libxml2	卡住，未找到可执行文件			
matio	卡住，未找到可执行文件			
openssl	卡住，程序过大 (12G)			
xpdf	完成			

2.3 环境、实验设置

软件环境：

Ubuntu 20.04

clang 12

python 3.8

测试原程序：

binutils-2.39

2.4 Fuzzing 配置

fuzzer：

afl-2.52b

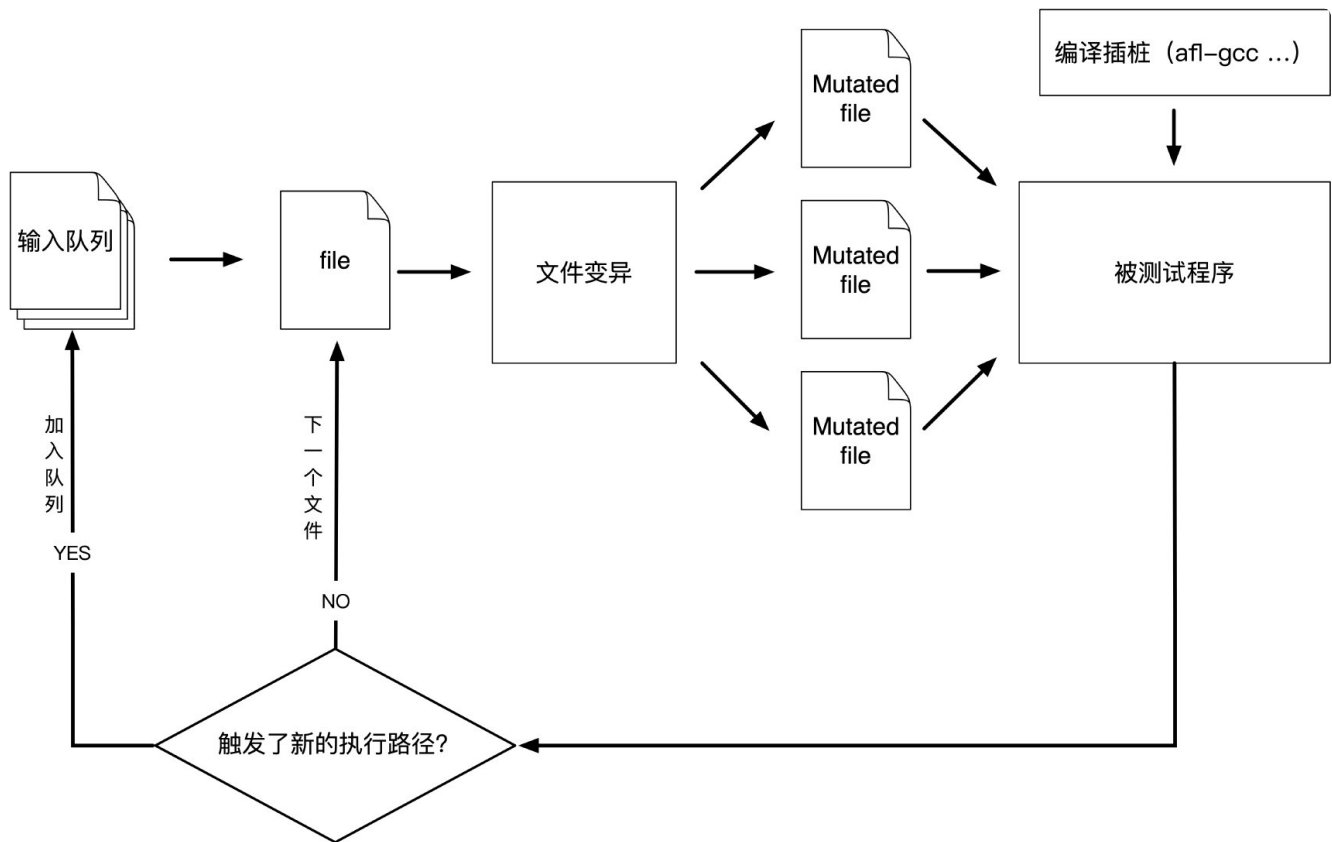
补充库：

bison 3.5.1

texinfo 6.7.0

2.5 实验整体过程

面对众多的 Fuzzer，如何准确、有效地评估 Fuzzer 的性能成了一项值得关注的难题。本选题从变异杀死的角度对 Fuzzer 进行评估。AFL的运行流程如下图：



2.5.1AFL工作流程

- ①从源码编译程序时进行插桩，以记录代码覆盖率（Code Coverage）；
- ②选择一些输入文件，作为初始测试集加入输入队列（queue）；
- ③将队列中的文件按一定的策略进行“突变”；
- ④如果经过变异文件更新了覆盖范围，则将其保留添加到队列中；
- ⑤上述过程会一直循环进行，期间触发了crash的文件会被记录下来。

我们在实验过程中大致遵循上述流程，步骤如下：

1. 下载

从[AFL官网](#)下载AFL。

2. 安装

将压缩包解压完成之后，进入到其目录下打开终端。

▼ install afl

Shell | 复制代码

```
1 make
2 sudo make install
```

如果在该目录下，出现了afl-fuzz的可执行文件，说明安装成功。

3. 模糊测试实验--以readelf为例

下载binutils，完成后解压进入其目录。

必须修改编译器环境为afl-gcc和afl-g++，在大部分的源码编译插桩中基本都需要这一步。

▼ 更改编译器环境

Shell | 复制代码

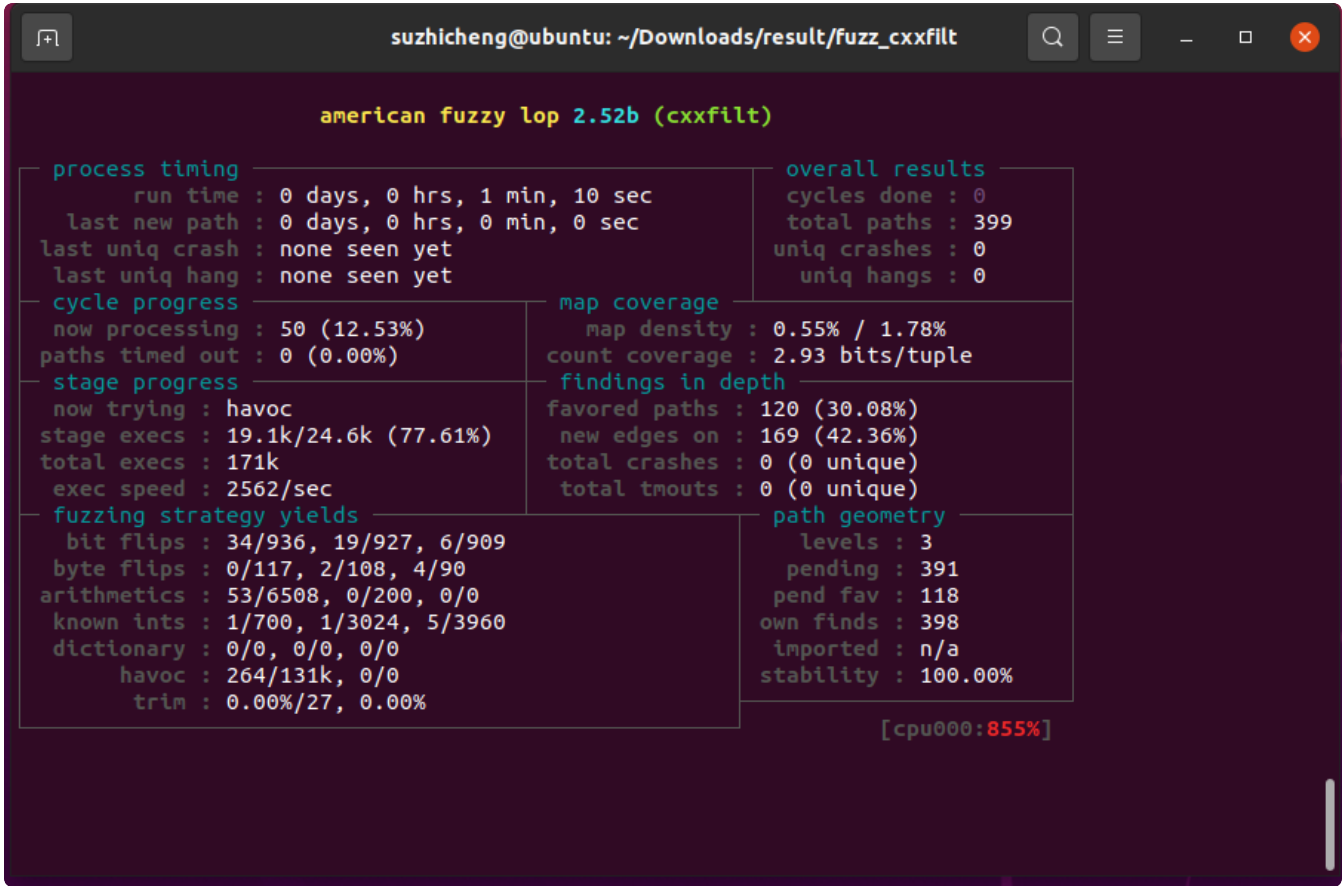
```
1 CC=~/.afl-gcc CXX=~/.afl-g++ ./configure //替换为自己afl的目录
2 make
```

▼ 进行模糊测试

Shell | 复制代码

```
1 >mkdir fuzz_in
2 >mkdir fuzz_out
3 >~/.afl-fuzz -i fuzz_in -o fuzz_out binutils/readelf -a @@
```

执行截图（由于当时未截图，以cxxfilt程序代替，其内容几乎相同）



2.5.2AFL结果

输出文件夹fuzz_out内部结构如下：

 crashes	27/11/2022 下午8:30	文件夹	
 hangs	27/11/2022 下午8:30	文件夹	
 queue	27/11/2022 下午8:55	文件夹	
 .cur_input	27/11/2022 下午8:30	CUR_INPUT 文件	1 KB
 fuzz_bitmap	27/11/2022 下午8:30	文件	64 KB
 fuzzer_stats	27/11/2022 下午8:30	文件	1 KB
 plot_data	27/11/2022 下午8:30	文件	2 KB

- 1. crashes：引起原程序崩溃的测试用例；
- 2. queue：能探明原程序新的执行路径的测试用例；
- 3. plot_data：按时间顺序存储了afl执行过程，用于绘图；
- 4. hangs：导致目标超时的独特测试用例；
- 5. fuzzer_stats：afl-fuzz的运行状态。

选择说明：

本实验后续基本使用 queue 中的用例作为变异测试的用例输入。这是有缺陷的，因为queue文件夹中存放的是所有具有独特执行路径的测试用例，一般并不会导致变异杀死，因此变异得分总体较低。但是crashes文件夹中的测试用例过少。权衡之后，我们更多衡量queue文件夹中的测试用例质量。

2.5.3对AFL用例筛选

由AFL得到对readelf分支全覆盖的用例，使用脚本筛选出可正确执行的部分用例并记录。

运行select脚本，尝试运行所有模糊输入，找出使其不崩溃的模糊输入。

▼ Shell | 复制代码

```
1 python3 select.py ./readelf
```

```
1  import sys
2  import subprocess
3  import os
4
5  test_executable = sys.argv[1]
6
7
8  path = "./fuzz_out/queue" #文件夹目录
9  files= os.listdir(path) #得到文件夹下的所有文件名称
10 s = []
11 for file in files:
12     if not file.endswith(".elf"):
13         os.rename(path+"/"+file,path+"/"+file+".elf")
14 record = open("./fuzz_out/correct","w+")
15 recordE = open("./fuzz_out/error","w+")
16 for file in files: #遍历文件夹
17     if not os.path.isdir(file) and file!=".state": #判断是否是文件夹，不是文件
        夹才打开
18         try:
19             subprocess.run([test_executable, "-a",path+"/"+file], check=True)
20             record.write(file+'\n')
21         except:
22             recordE.write(file+'\n')
23
24 record.close()
25
```

2.5.4 变异测试理解

变异测试是一种基于故障的软件测试技术。它通过计算变异分数来评估测试套件的质量。它通过创建原始程序、和其稍微修改的版本（称为变异体），并针对每个版本运行测试套件来实现评估测试套件。如果测试套件检测到变更，则认为变异体已被杀死，否则该变异体得到存活。如果至少有一项测试使得运行失败，那么变异体就会被杀死。

mutll作为变异测试工具，先对源程序变异，得到变异体。利用AFL的输出作为测试套件，衡量其质量。变异得分，即为所有变异体在这个测试用例组合的通过率越高，则AFL产生的测试套件的质量越好。

2.5.5 变异测试运行示例

以下流程均以readelf为例。

1. 首先需要 配置&编译&插桩， 命令如下：


```
1 export CC=clang-12
2 export CXX=clang++-12 //更改编译器环境
3 ./configure CFLAGS="-O0 -fexperimental-new-pass-manager -fpass-plugin=/usr/lib/mull-ir-frontend-12 -g -grecord-command-line -fprofile-instr-generate -fcoverage-mapping"
4
5 make
```

3. 通过编写脚本 test.py 使得测试能够自动化进行,命令如下:

```
1 mull-runner-12 ./readelf -ide-reporter-show-killed --test-program=python3
-- test.py ./readelf
```

```
1 import sys
2 import subprocess
3 import os
4
5 test_executable = sys.argv[1]
6
7
8 path = "./fuzz_out/queue" #文件夹目录
9 file = open("./fuzz_out/correct","r")
10 records = file.read().splitlines()
11 records = records[200:210]
12 for record in records:
13     subprocess.run([test_executable, "-a",path+"/"+record], check=True)
14 file.close()
15
```

4. 运行结果截图:

```

1:13: warning: Survived: Replaced == with != [cxx_eq_to_ne]
    if (nelem == 0 || elsize == 0)
        ^
/home/compiler/Documents/auto-testing/lib/binutils-2.39/libiberty/./xmalloc.c:16
1:28: warning: Survived: Replaced == with != [cxx_eq_to_ne]
    if (nelem == 0 || elsize == 0)
        ^
/home/compiler/Documents/auto-testing/lib/binutils-2.39/libiberty/./xmalloc.c:16
6:27: warning: Survived: Replaced * with / [cxx_mul_to_div]
    xmalloc_failed (nelem * elsize);
                        ^
/home/compiler/Documents/auto-testing/lib/binutils-2.39/libiberty/./xmalloc.c:17
6:12: warning: Survived: Replaced == with != [cxx_eq_to_ne]
    if (size == 0)
        ^
/home/compiler/Documents/auto-testing/lib/binutils-2.39/libiberty/./xstrdup.c:33
:36: warning: Survived: Replaced + with - [cxx_add_to_sub]
    register size_t len = strlen (s) + 1;
                                   ^
[info] Mutation score: 1%
[info] Total execution time: 255513ms

```

2.5.6 差分测试过程

思路：将AFL产生的未导致原程序崩溃的测试用例在原程序中的运行结果存储下来（利用select.py脚本），并且将同一测试用例在变异体中的运行结果存储下来（利用test.py脚本）。比较同一用例在原程序和变异体上运行的结果，若不同，则该变异体被杀死，否则该变异体存活（利用differential_comparing.py脚本）。

脚本如下：

```
1  import sys
2  import subprocess
3  import os
4
5  test_executable = sys.argv[1]
6
7
8  path = "./fuzz_out/queue" #文件夹目录
9  files= os.listdir(path) #得到文件夹下的所有文件名
10 s = []
11 for file in files:
12     if not file.endswith(".elf"):
13         os.rename(path+"/"+file,path+"/"+file+".elf")
14 record = open("./fuzz_out/correct","w+")
15 recordE = open("./fuzz_out/error","w+")
16 i = 0
17 for file in files: #遍历文件夹
18     if not os.path.isdir(file) and file!=".state": #判断是否是文件夹，不是文件
        夹才打开
19         file1 = open("./fuzz_out/correct_result/"+str(i),"w+")
20         try:
21             subprocess.run([test_executable, "-SD",path+"/"+file], check=True,stdout=file1)
22             record.write(file+'\n')
23         except:
24             recordE.write(file+'\n')
25         file1.close()
26         size = os.path.getsize("./fuzz_out/correct_result/"+str(i))
27         if size==0:
28             os.remove("./fuzz_out/correct_result/"+str(i))
29             i-=1
30         i = i+1
31 record.close()
32
33
```

```
1 import sys
2 import subprocess
3 import os
4
5 test_executable = sys.argv[1]
6
7
8 path = "./fuzz_out/queue" #文件夹目录
9 files= os.listdir(path) #得到文件夹下的所有文件名
10 s = []
11 for file in files:
12     if not file.endswith(".elf"):
13         os.rename(path+"/"+file,path+"/"+file+".elf")
14 record = open("./fuzz_out/correct","w+")
15 recordE = open("./fuzz_out/error","w+")
16 i = 0
17 for file in files: #遍历文件夹
18     if not os.path.isdir(file) and file!=".state": #判断是否是文件夹，不是文件
        夹才打开
19         file1 = open("./fuzz_out/correct_result/"+str(i),"w+")
20         try:
21             subprocess.run([test_executable, "-SD",path+"/"+file], check=True,stdout=file1)
22             record.write(file+'\n')
23         except:
24             recordE.write(file+'\n')
25         file1.close()
26         size = os.path.getsize("./fuzz_out/correct_result/"+str(i))
27         if size==0:
28             os.remove("./fuzz_out/correct_result/"+str(i))
29             i-=1
30         i = i+1
31 record.close()
32
33
```

```
1 import sys
2 import subprocess
3 import os
4
5 path1 = "./fuzz_out/correct_result/"
6 file2 = open("./fuzz_out/final", "rb")
7 corrects = []
8 for i in range(20, 50):
9     with open(path1 + str(i), "rb") as file1:
10         try:
11             corrects.append(file1.read().split(str.encode('\n' + "./fuzz_out/queue/"))[1])
12         except:
13             corrects.append("")
14 check = file2.read()
15 checks = check.split(str.encode('\n' + "./fuzz_out/queue/"))
16 length = len(checks) - 1
17 count = 0
18 for i in checks:
19     if i in corrects:
20         count += 1
21
22 print(count)
23 print(length)
24 print(count / length)
25 file1.close()
26 file2.close()
27
28
```

```
1 import sys
2 import subprocess
3 import os
4
5
6 path1 = "./fuzz_out/correct_result/"
7 file2 = open("./fuzz_out/final", "rb")
8 corrects = []
9 for i in range(0, 3680):
10     with open(path1 + str(i), "rb") as file1:
11         corrects.append(file1.read())
12
13 check = file2.read()
14 count = 0
15 for i in corrects:
16     if i in check:
17         count += 1
18 length = 4517 * 30
19 print(count)
20 print(length)
21 print(count / length)
22
23 file2.close()
24
```

2.5.7 差分测试结果

readelf

输出结果相同的测试用例数量：8

所有测试用例数量：197408

输出结果不同的比例：99.995947479332144593937429080888%

optdump

输出结果相同的测试用例数量：350188

所有测试用例数量：756270

输出结果不同的比例：53.695373345498300871381913866741%

size

输出结果相同的测试用例数量：64,804

所有测试用例数量：122,384

输出结果不同的比例：47.04863380834096%

nm

输出结果相同的测试用例数量：12

所有测试用例数量：144510

输出结果不同的比例：99.991696076396097155906165663276%

cxxfilt

输出结果相同的测试用例数量：30

所有测试用例数量：135510

输出结果不同的比例：99.97786141245%

2.5.8 差分测试结果说明

如上文所述，由于mull对每个变异体的执行过程高度封装，因此难以进入到子进程内部去修改运行逻辑，我们难以得到测试用例在每个变异体上的表现情况，即无法获知对于特定变异体，AFL产生的测试用例运行的结果是否与原程序有区别。而考虑到，mull工具对每个变异体都会执行相同的测试用例，因此我们转变视角：以测试用例为基准，计量有多少测试用例的在不同程序上的输出有区别。以此虽不能得出变异得分，但也能反映出AFL产生的测试用例的质量。即差分测试结果中的**输出结果不同的比例**。

2.6 实现、配置和流程上的坑

2.6.1 模糊测试更改编译器环境

描述：在运行如下命令时出现错误，提示找不到makeinfo。



Shell | 复制代码

```
1 >CC=~/.afl-gcc CXX=~/.afl-g++ ./configure //替换为自己afl的目录
2 >make
```

```

/home/arthur/Downloads/binutils-2.39/missing: 81: Makeinfo: not found
WARNING: 'makeinfo' is missing on your system.
You should only need it if you modified a '.texi' file, or
any other file indirectly affecting the aspect of the manual.
You might want to install the Texinfo package:
<http://www.gnu.org/software/texinfo/>
The spurious makeinfo call might also be the consequence of
using a buggy 'make' (AIX, DU, IRIX), in which case you might
want to install GNU make:
<http://www.gnu.org/software/make/>
make[4]: *** [Makefile:1461: doc/as.info] 错误 127
make[4]: 离开目录"/home/arthur/Downloads/binutils-2.39/gas"
make[3]: *** [Makefile:1643: all-recursive] 错误 1
make[3]: 离开目录"/home/arthur/Downloads/binutils-2.39/gas"
make[2]: *** [Makefile:992: all] 错误 2
make[2]: 离开目录"/home/arthur/Downloads/binutils-2.39/gas"
make[1]: *** [Makefile:5453: all-gas] 错误 2
make[1]: 离开目录"/home/arthur/Downloads/binutils-2.39"
make: *** [Makefile:1004: all] 错误 2
arthur@arthur-virtual-machine:~/Downloads/binutils-2.39$

```

解决方案: `sudo apt-get install texinfo`

继续报错: `configure: error: Building gprofng requires bison 3.0.4 or later.`

解决方案: `sudo apt-get install bison`

2.6.2 模糊测试插桩失败

```

make[2]: Leaving directory '/root/afl_soft/libxml2-2.10.3/xstc'
Making all in python
make[2]: Entering directory '/root/afl_soft/libxml2-2.10.3/python'
make all-recursive
make[3]: Entering directory '/root/afl_soft/libxml2-2.10.3/python'
Making all in .
make[4]: Entering directory '/root/afl_soft/libxml2-2.10.3/python'
CC      libxml.lo
../libtool: line 1748: ../../afl-2.52b/afl-gcc: No such file or directory
make[4]: *** [Makefile:624: libxml.lo] Error 1
make[4]: Leaving directory '/root/afl_soft/libxml2-2.10.3/python'
make[3]: *** [Makefile:748: all-recursive] Error 1
make[3]: Leaving directory '/root/afl_soft/libxml2-2.10.3/python'
make[2]: *** [Makefile:517: all] Error 2
make[2]: Leaving directory '/root/afl_soft/libxml2-2.10.3/python'
make[1]: *** [Makefile:1608: all-recursive] Error 1
make[1]: Leaving directory '/root/afl_soft/libxml2-2.10.3'
make: *** [Makefile:804: all] Error 2
root@auto-testing:~/afl_soft/libxml2-2.10.3#

```

描述: 尚未解决, 已检查路径及权限, 并且确认该文件存在。

2.6.3 模糊测试运行错误


```
american fuzzy lop 2.52b (cxxfilt)

process timing
  run time : 0 days, 0 hrs, 0 min, 2 sec
  last new path : none yet (odd, check syntax!)
  last uniq crash : none seen yet
  last uniq hang : none seen yet
cycle progress
  now processing : 0 (0.00%)
  paths timed out : 0 (0.00%)
stage progress
  now trying : havoc
  stage execs : 255/256 (99.61%)
  total execs : 16.1k
  exec speed : 4571/sec
fuzzing strategy yields
  bit flips : 0/32, 0/31, 0/29
  byte flips : 0/4, 0/3, 0/1
  arithmetics : 0/222, 0/0, 0/0
  known ints : 0/23, 0/84, 0/44
  dictionary : 0/0, 0/0, 0/0
  havoc : 0/15.4k, 0/0
  trim : 69.23%/3, 0.00%

overall results
  cycles done : 57
  total paths : 1
  uniq crashes : 0
  uniq hangs : 0
map coverage
  map density : 0.07% / 0.07%
  count coverage : 1.00 bits/tuple
findings in depth
  favored paths : 1 (100.00%)
  new edges on : 1 (100.00%)
  total crashes : 0 (0 unique)
  total tmouts : 0 (0 unique)
path geometry
  levels : 1
  pending : 0
  pend fav : 0
  own finds : 0
  imported : n/a
  stability : 100.00%

^C [cpu000:111%]
```

描述：在运行如下指令时出现错误，发现实际上并没有实际执行afl。

▼

Bash | 复制代码

```
1 ~/../afl-fuzz -i fuzz_in -o fuzz_out binutils/readelf -d @@
```

解决方案：详细查坎指令参数，如将-d替换为-p，并删除@@等等。

2.6.4 模糊测试并无make 之后并无可执行文件

描述：按照流程，需要对可执行文件进行afl过程，但是如 libxml2 的工具make之后产物为 shell 脚本（如xmllint）

解决方案：尝试使用shc将 shell 脚本打包成可执行文件。

2.6.5 变异测试检测并未插桩

描述：在模糊测试中进行了插桩工作，但是在变异测试过程中显示并未进行插桩。

解决方案：使用两套插桩的程序，一套负责模糊测试过程，一套负责变异测试过程。分别如下配置 CC 和 CXX 指令：

▼ afl插桩

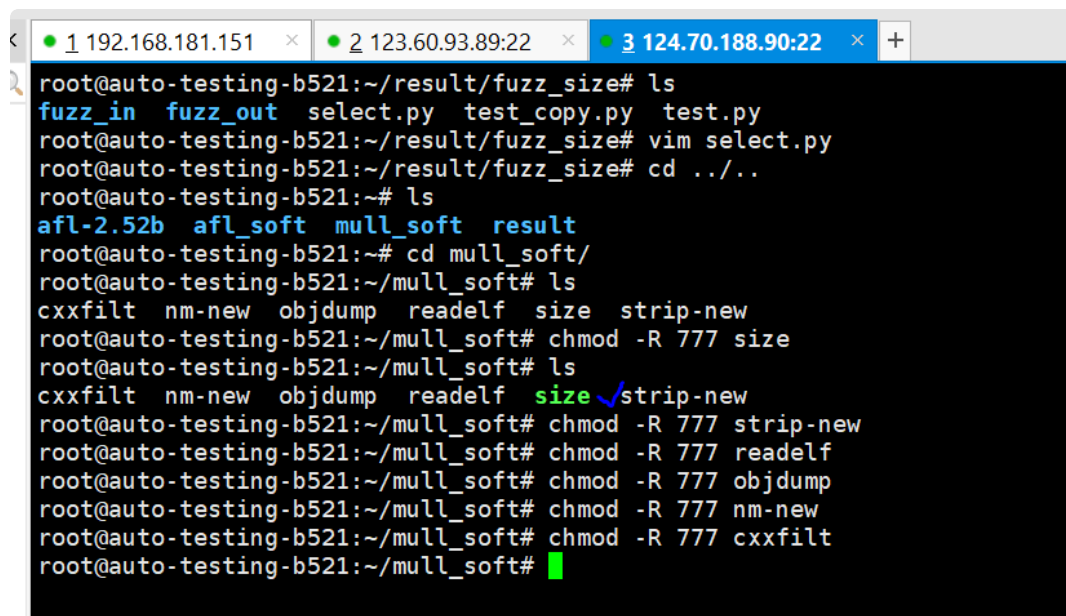
Bash | 复制代码

```
1 CC=~/.../afl-gcc CXX=~/.../afl-g++ ./configure
```

```
1 export CC=clang-12
2 export CXX=clang++-12
3 ./configure CFLAGS="-O0 -fexperimental-new-pass-manager -fpass-plugin=/usr/lib/mull-ir-frontend-12 -g -grecord-command-line -fprofile-instr-generate -fcoverage-mapping"
```

2.6.6 变异测试运行select脚本出错

解决方案：需要修改权限，权限修改可见其颜色发生变化。



```
root@auto-testing-b521:~/result/fuzz_size# ls
fuzz_in fuzz_out select.py test_copy.py test.py
root@auto-testing-b521:~/result/fuzz_size# vim select.py
root@auto-testing-b521:~/result/fuzz_size# cd ../../
root@auto-testing-b521:~# ls
afl-2.52b afl_soft mull_soft result
root@auto-testing-b521:~# cd mull_soft/
root@auto-testing-b521:~/mull_soft# ls
cxxfilt nm-new objdump readelf size strip-new
root@auto-testing-b521:~/mull_soft# chmod -R 777 size
root@auto-testing-b521:~/mull_soft# ls
cxxfilt nm-new objdump readelf size ✓ strip-new
root@auto-testing-b521:~/mull_soft# chmod -R 777 strip-new
root@auto-testing-b521:~/mull_soft# chmod -R 777 readelf
root@auto-testing-b521:~/mull_soft# chmod -R 777 objdump
root@auto-testing-b521:~/mull_soft# chmod -R 777 nm-new
root@auto-testing-b521:~/mull_soft# chmod -R 777 cxxfilt
root@auto-testing-b521:~/mull_soft#
```

2.6.7 变异测试显示没有变异体

```
1 192.168.181.151 x 2 123.60.93.89:22 x 3 124.70.188.90:22 x +
root@auto-testing-b521:~/result/fuzz_xpdf# cd fuzz_out/
root@auto-testing-b521:~/result/fuzz_xpdf/fuzz_out# ls
correct crashes error final fuzz_bitmap fuzzer_stats hangs plot_data queue
root@auto-testing-b521:~/result/fuzz_xpdf/fuzz_out# vim correct
root@auto-testing-b521:~/result/fuzz_xpdf/fuzz_out# cd ..
root@auto-testing-b521:~/result/fuzz_xpdf# mull-runner-12 ../../mull_soft/xpdf/xpdf-3
.02/xpdf/pdftotext -ide-reporter-show-killed -test-program=python3 -- test.py ../../
/mull_soft/xpdf/xpdf-3.02/xpdf/pdftotext > result
root@auto-testing-b521:~/result/fuzz_xpdf# mull-runner-12 ../../mull_soft/xpdf/xpdf-3
.02/xpdf/pdftotext -ide-reporter-show-killed -test-program=python3 -- test.py ../../
/mull_soft/xpdf/xpdf-3.02/xpdf/pdftotext
[info] Using config /root/result/fuzz_xpdf/mull.yml
[warning] Could not find dynamic library: libm.so.6
[warning] Could not find dynamic library: libstdc++.so.6
[warning] Could not find dynamic library: libgcc_s.so.1
[warning] Could not find dynamic library: libc.so.6
[info] Warm up run (threads: 1)
[#####] 1/1. Finished in 301ms
[info] Extracting coverage information (threads: 1)
[warning] cannot read raw profile data: No such file or directory
[#####] 1/1. Finished in 0ms
[warning] Cannot read coverage info: No such file or directory

[info] Filter mutants (threads: 1)
[#####] 1/1. Finished in 0ms
[info] Baseline run (threads: 1)
[#####] 1/1. Finished in 255ms
[info] No mutants found. Mutation score: infinitely high
[info] Total execution time: 558ms
root@auto-testing-b521:~/result/fuzz_xpdf#
```

描述：显示 No mutants found。

解决方案：插桩过程出现问题，需要检查插桩的情况。

3. 结果分析

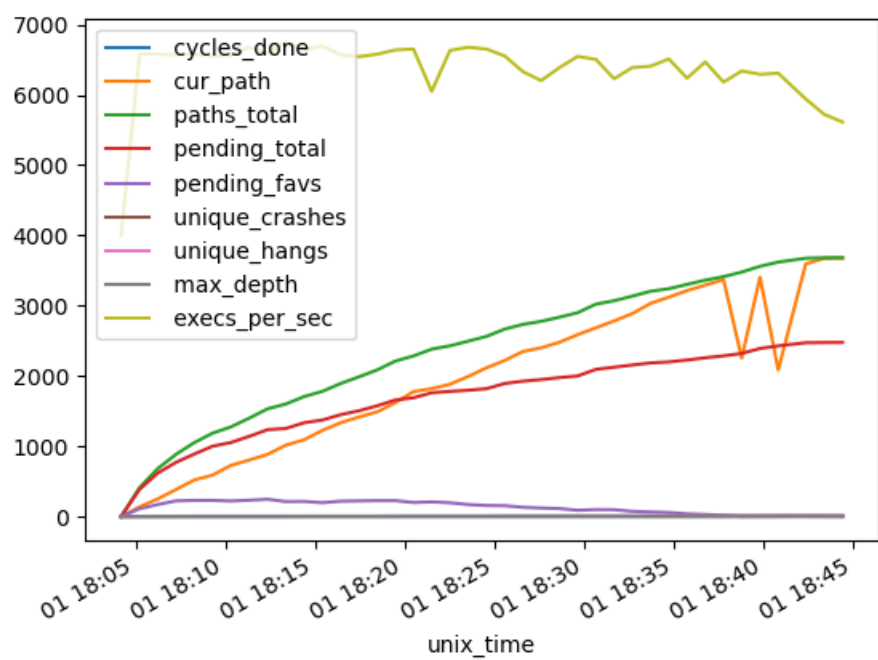
3.1 模糊测试过程及变异测试结果的绘图结果

对变异测试结果中，各变异算子的解释可查看 mull 官网：

<https://mull.readthedocs.io/en/latest/SupportedMutations.html>

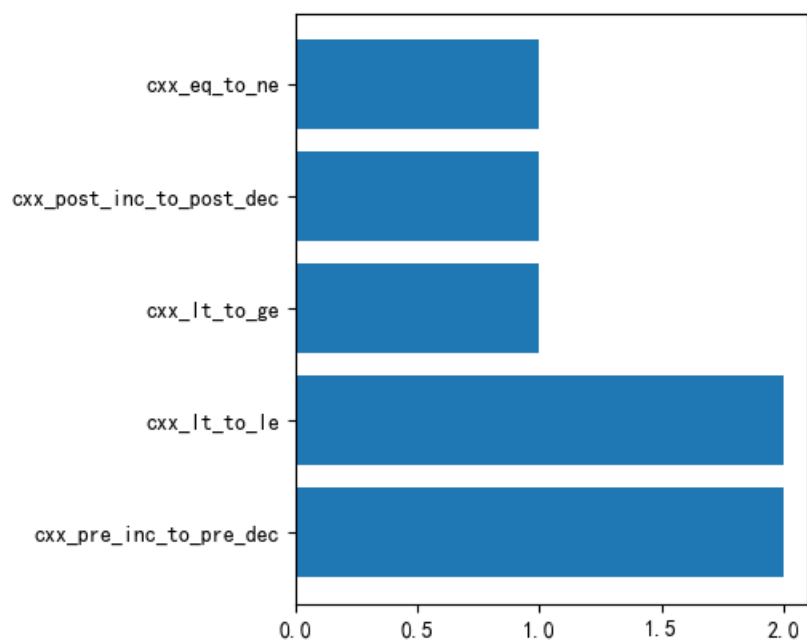
cxxfilter

模糊测试过程：

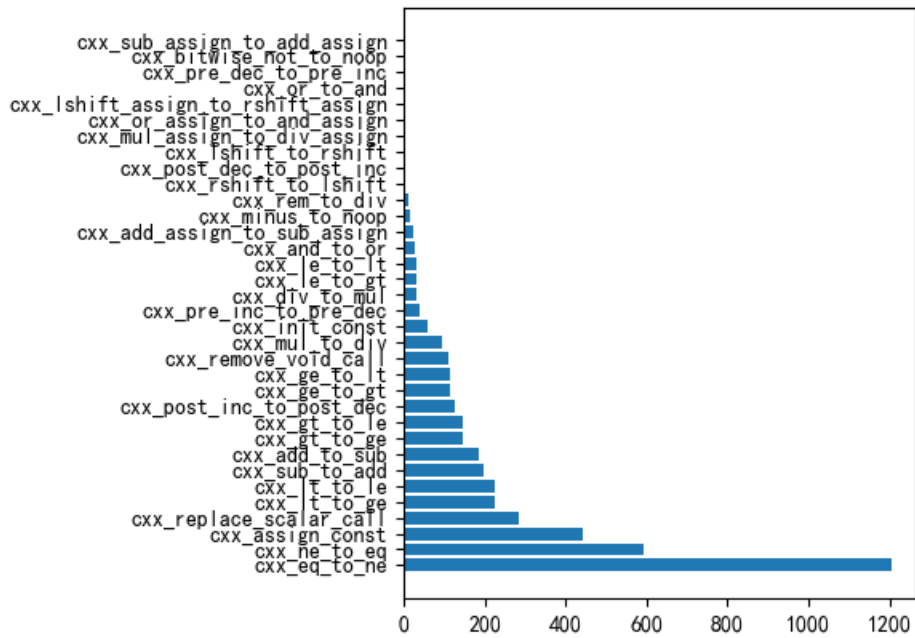


变异测试结果：

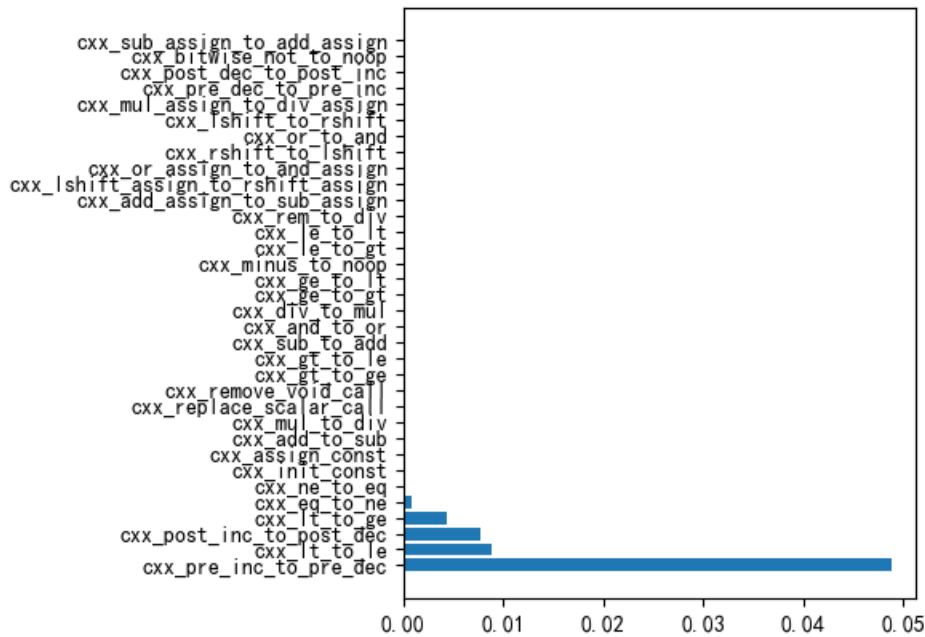
cxxfilter中导致变异杀死的变异算子



cxxfilter中未导致变异杀死的变异算子

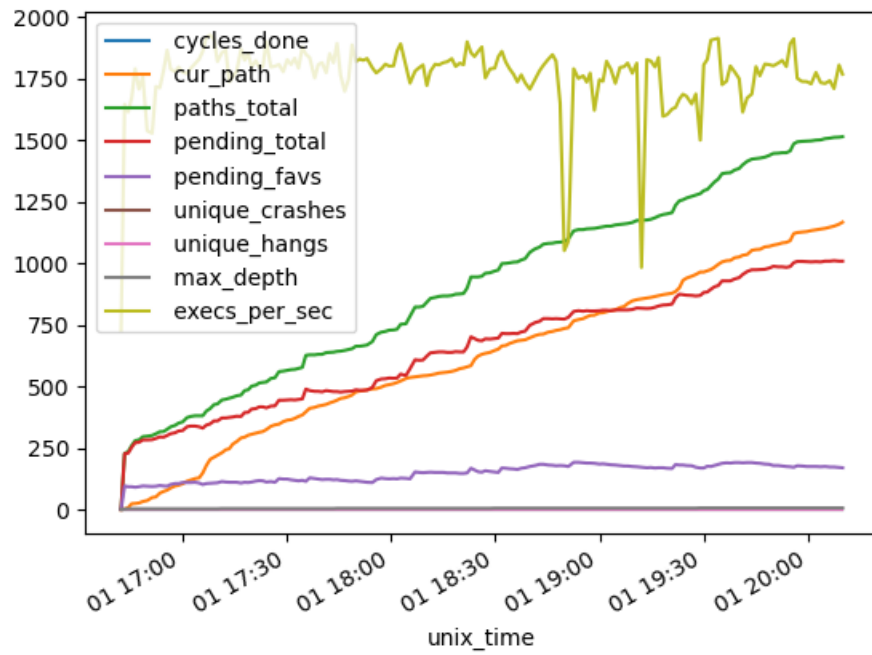


cxxfilter中变异算子被杀死的比例



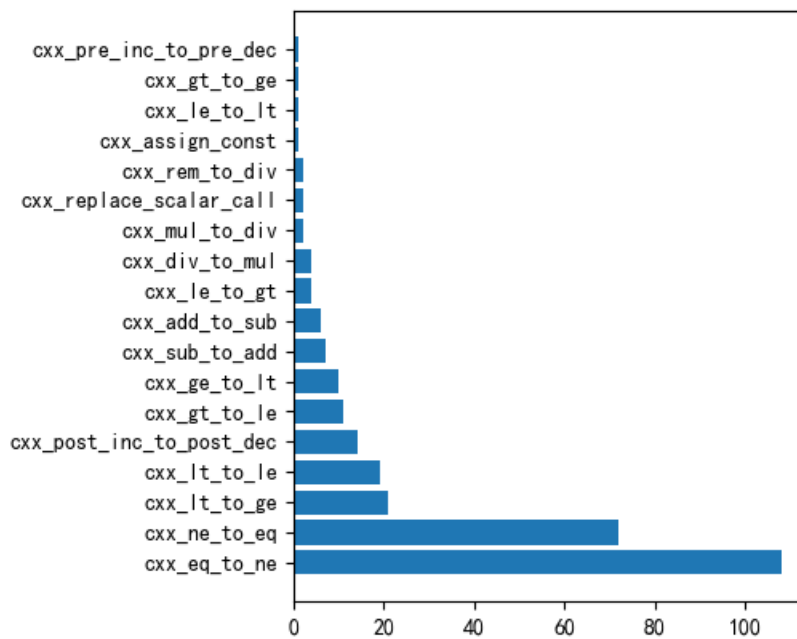
nm

模糊测试过程：

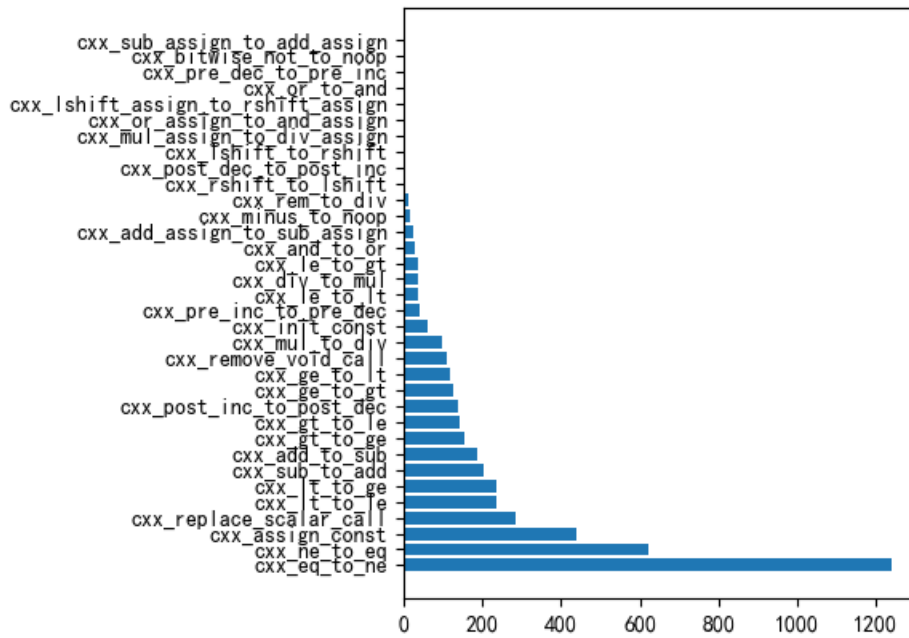


变异测试结果：

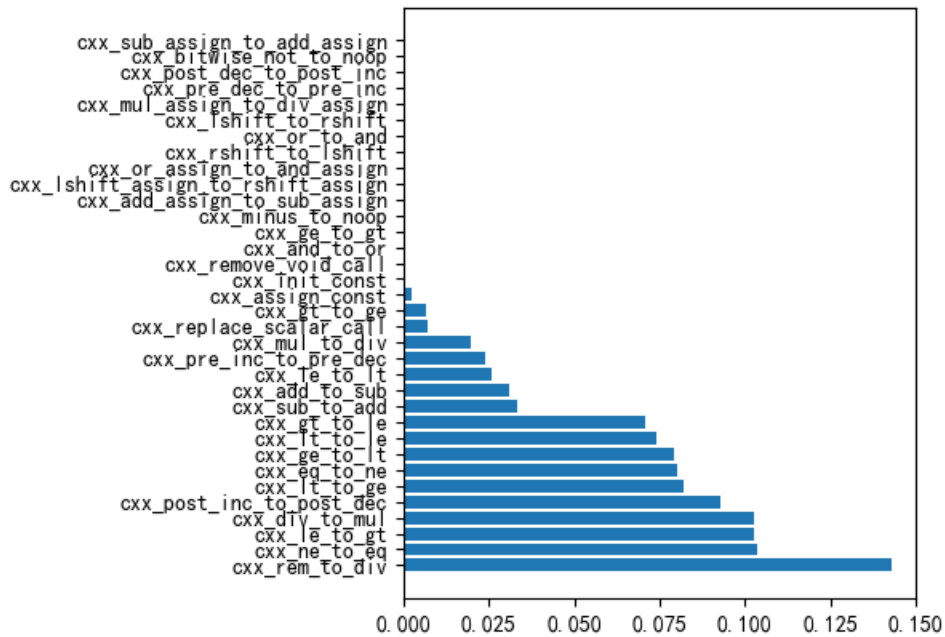
nm中导致变异杀死的变异算子



nm中未导致变异杀死的变异算子

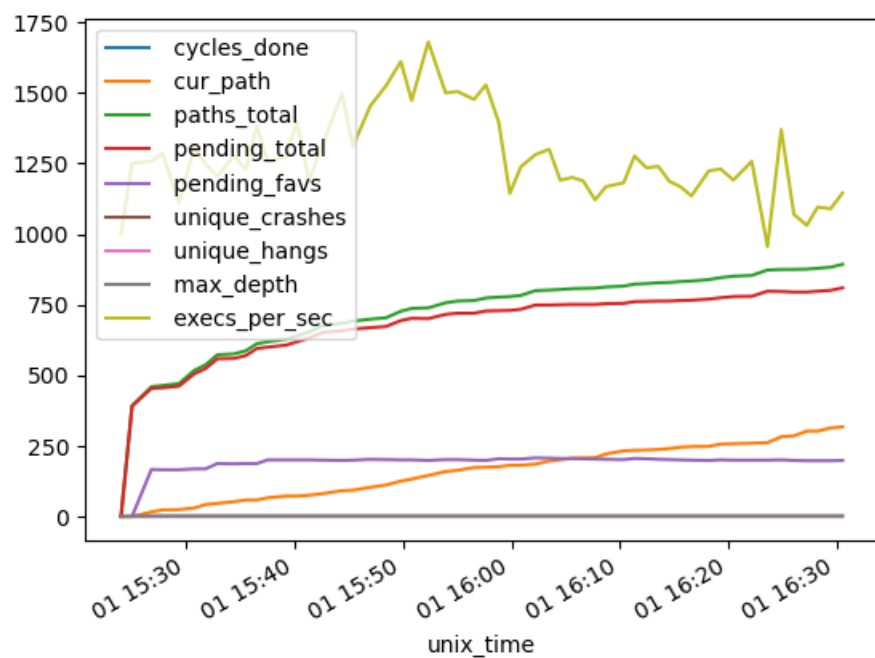


nm中变异算子被杀死的比例



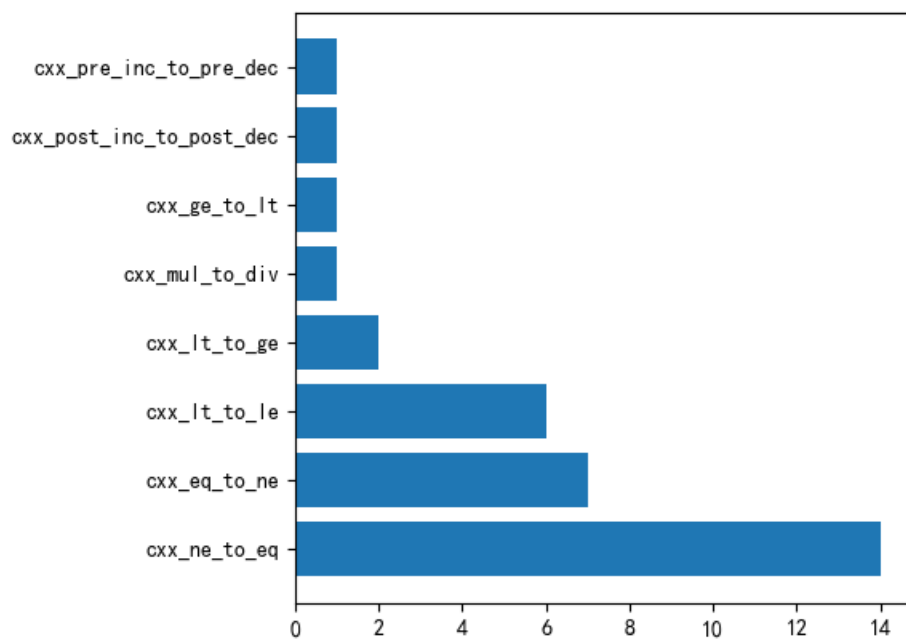
objdump

模糊测试结果:

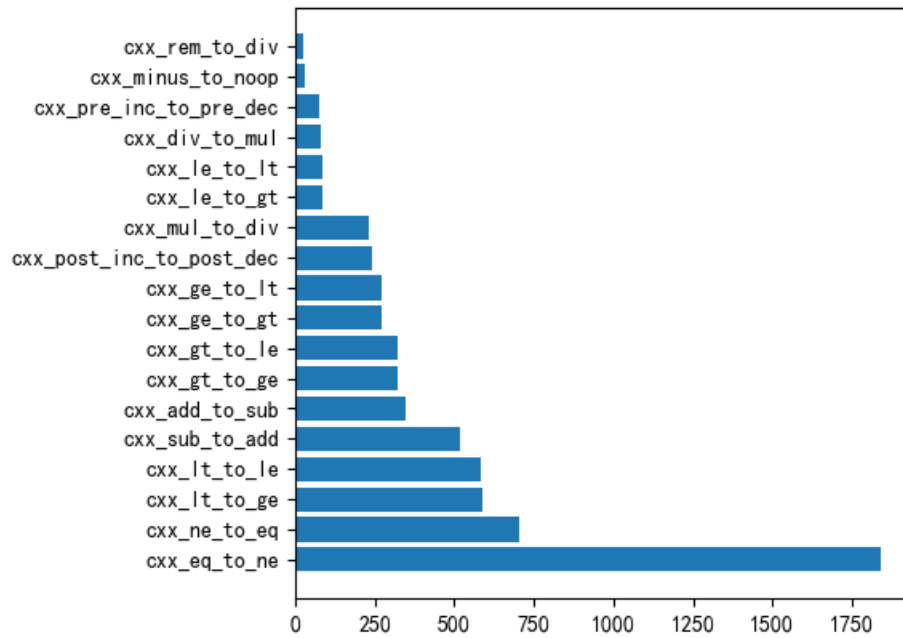


变异测试结果：

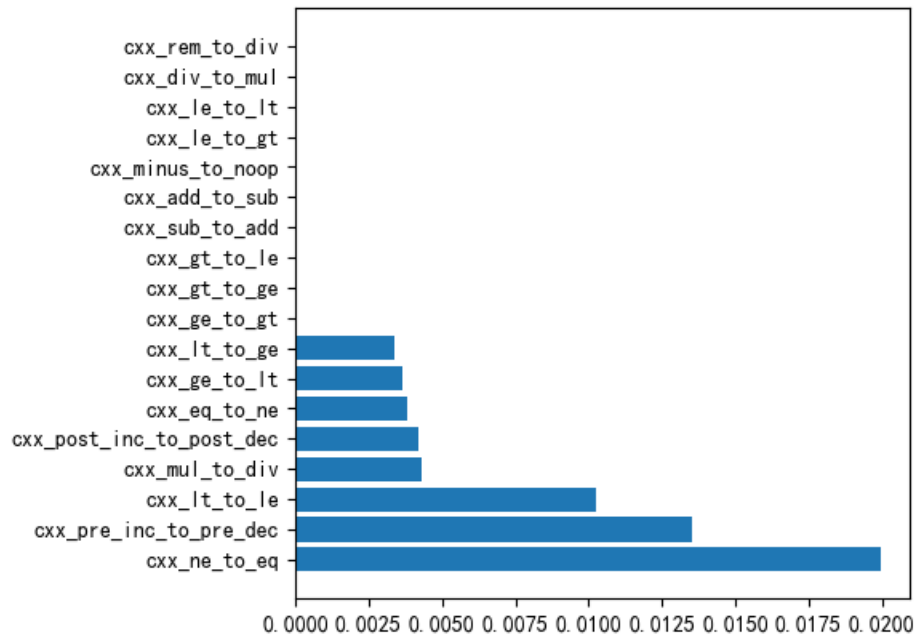
objdump中导致变异杀死的变异算子



objdump中未导致变异杀死的变异算子

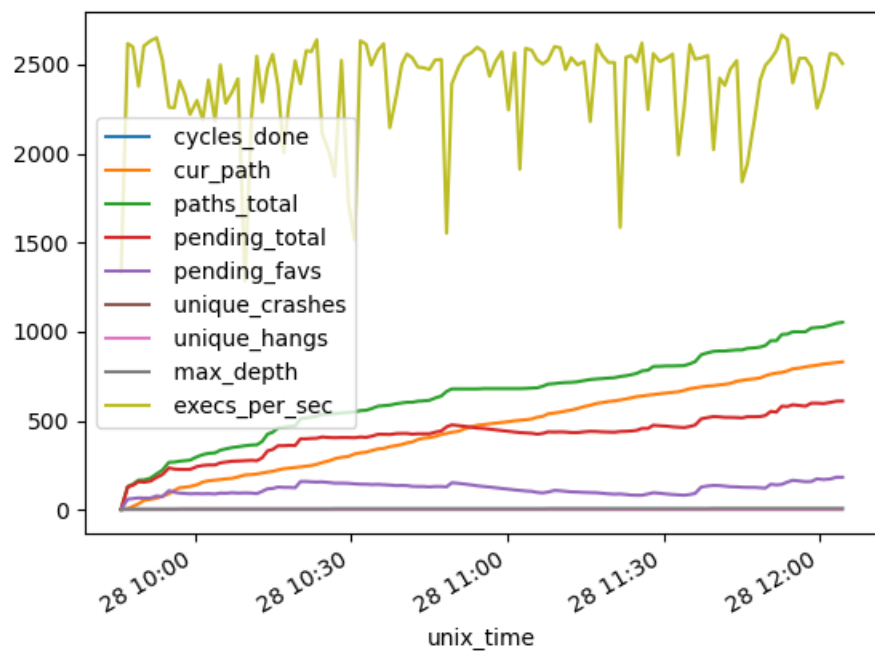


objdump中变异算子被杀死的比例



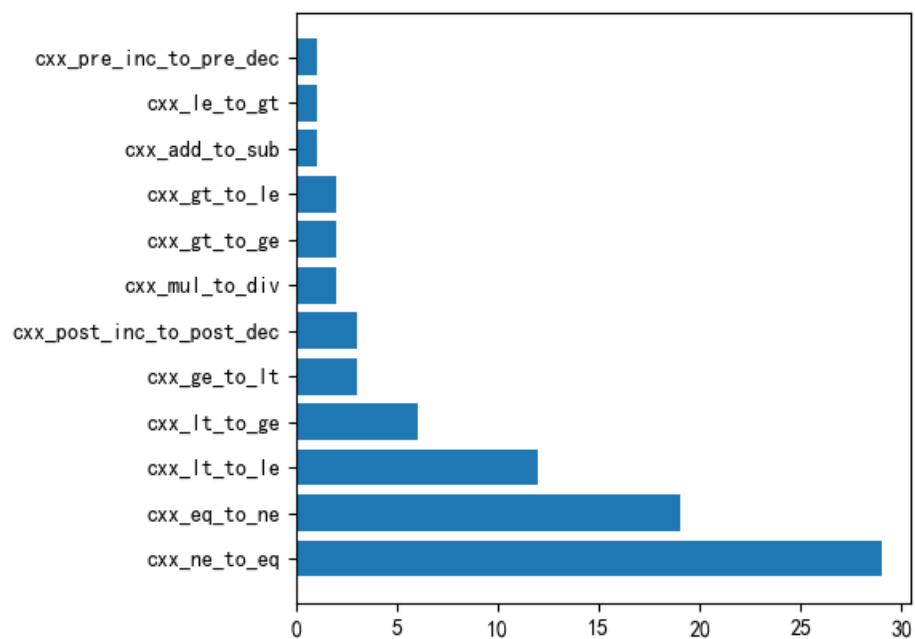
readelf

模糊测试过程：

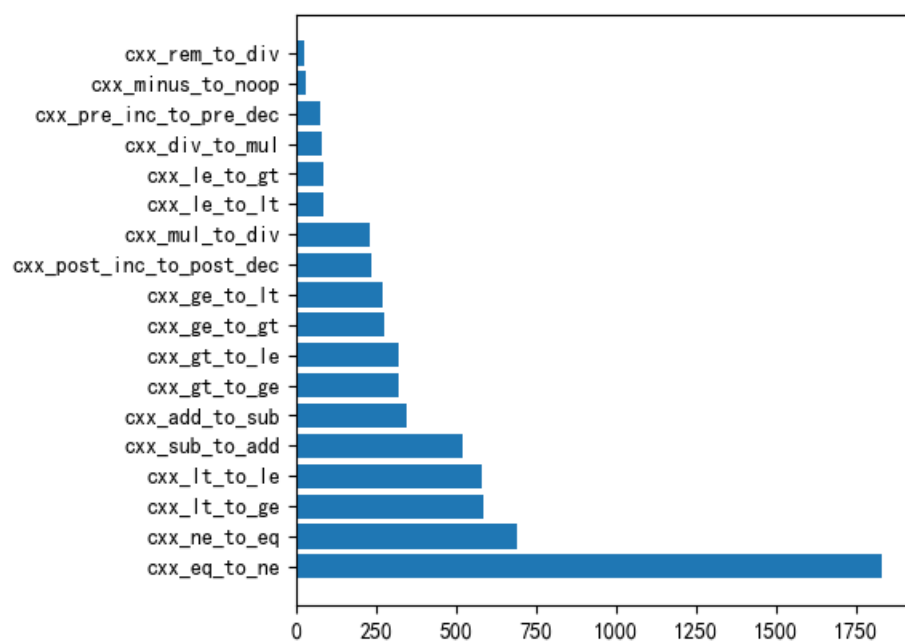


变异测试结果：

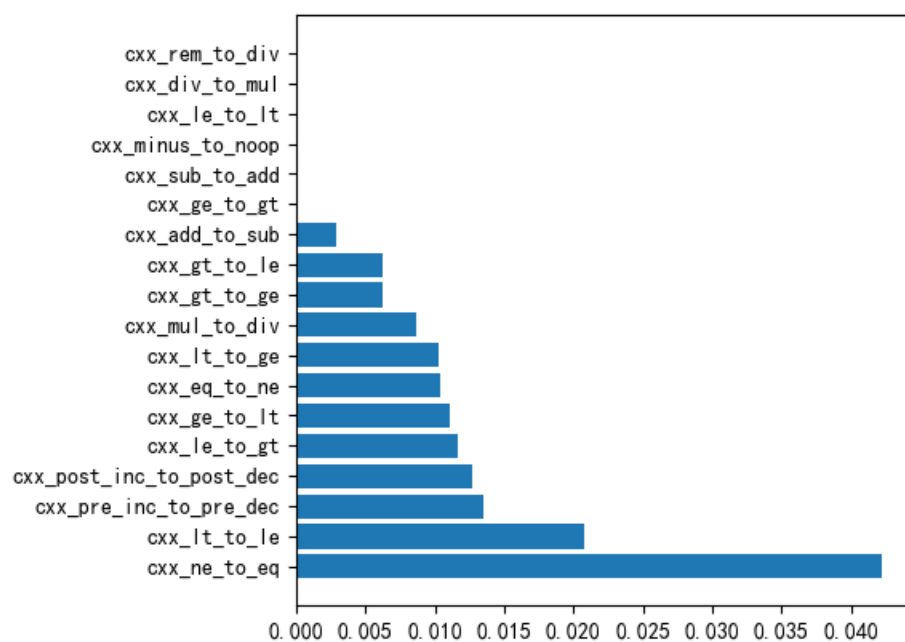
readelf中导致变异杀死的变异算子



readelf中未导致变异杀死的变异算子

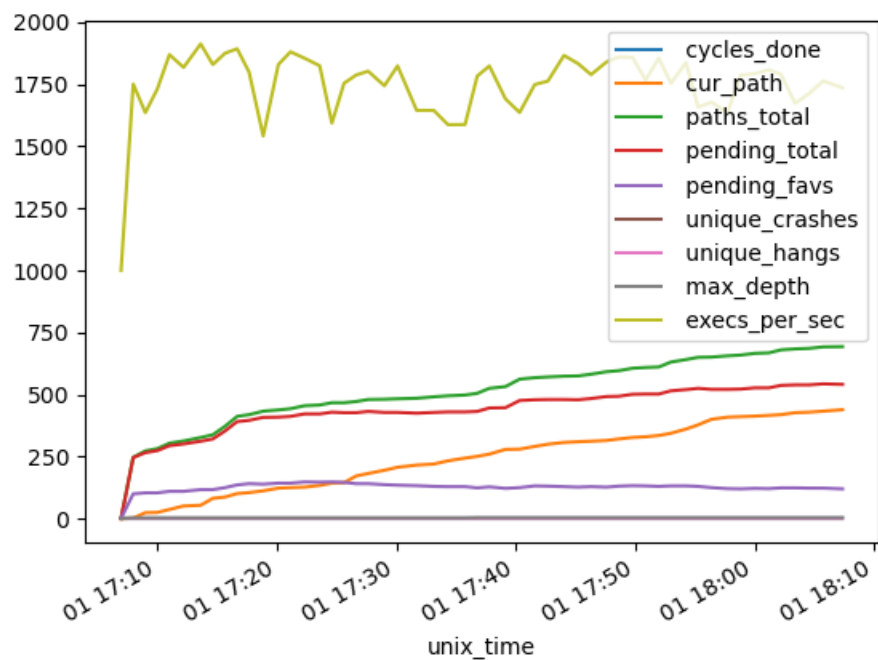


readelf中变异算子被杀死的比例



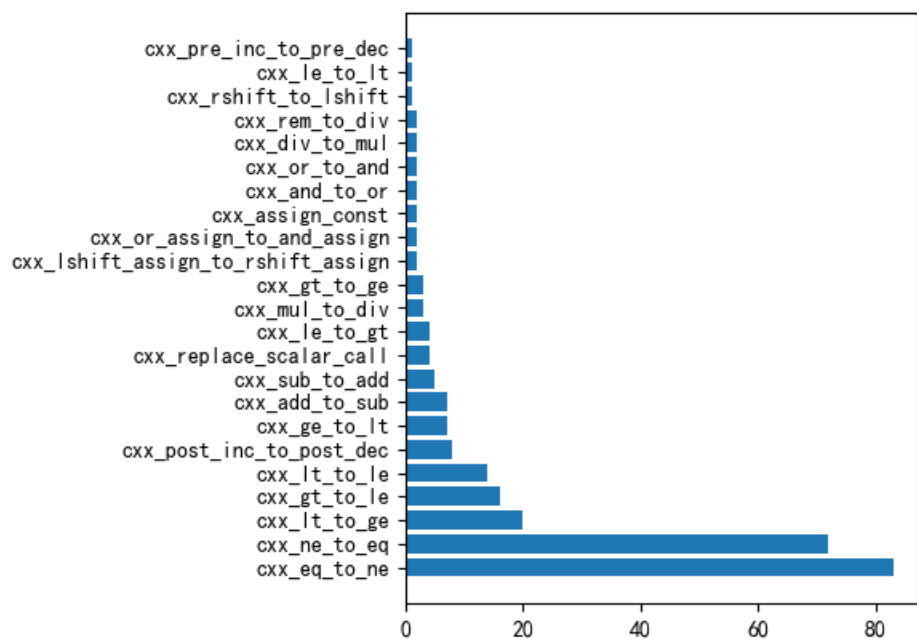
size

模糊测试过程：

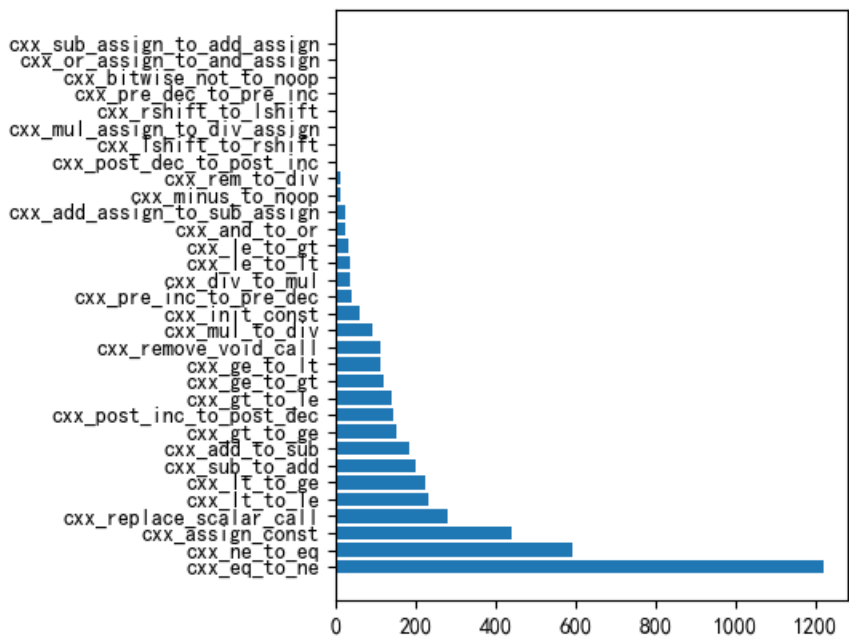


变异测试结果：

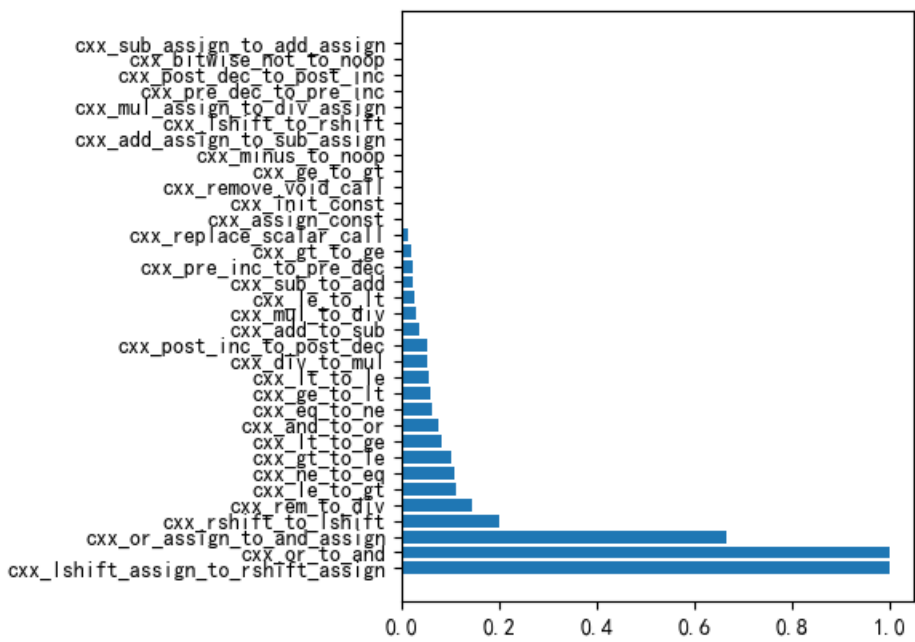
size中导致变异杀死的变异算子



size中未导致变异杀死的变异算子

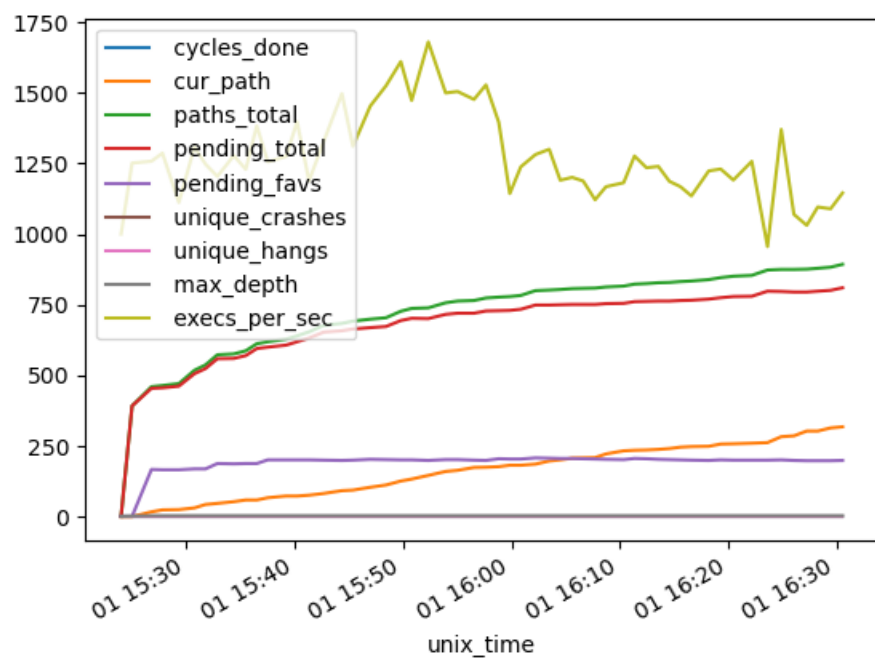


size中变异算子被杀死的比例



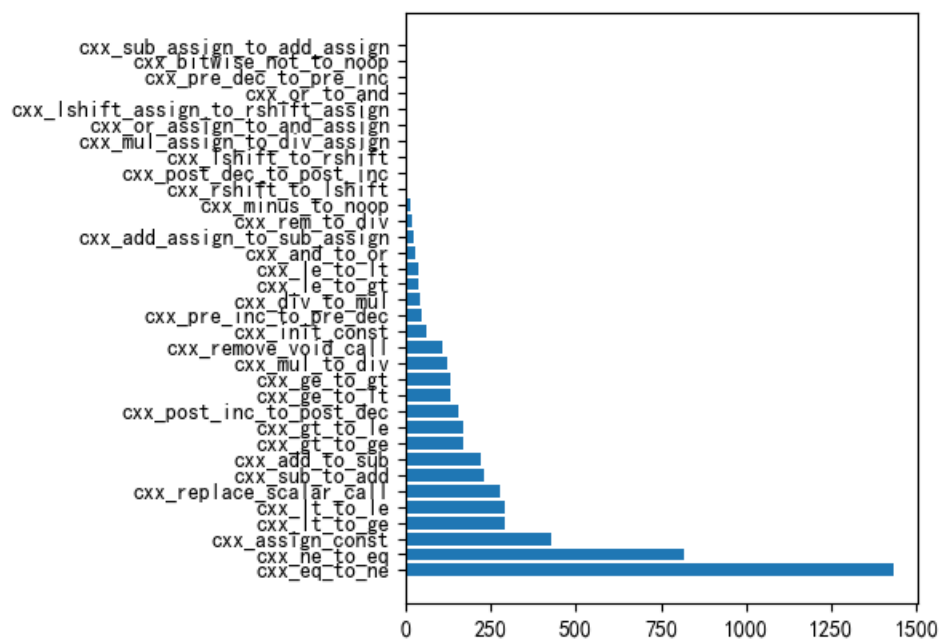
strip

模糊测试过程

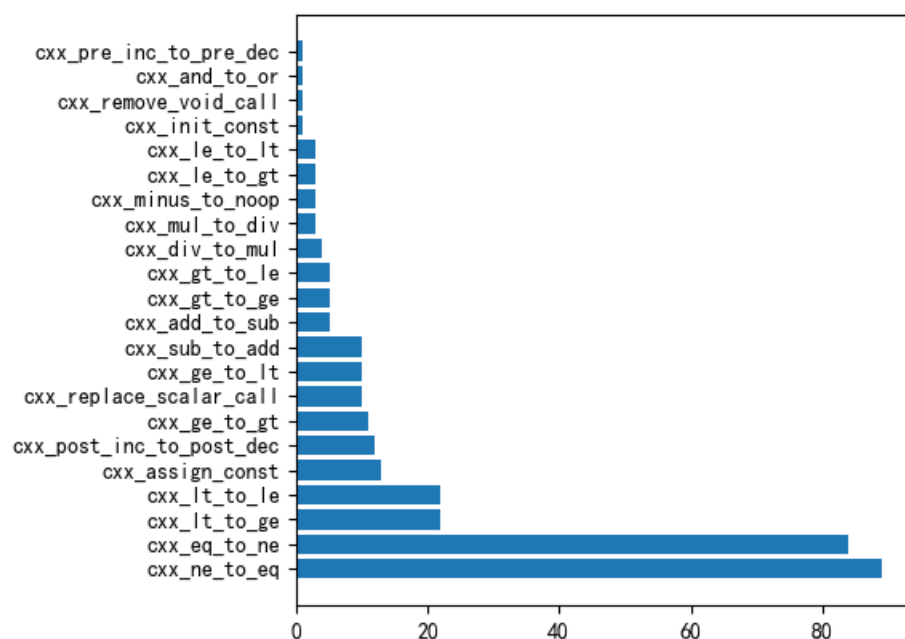


变异测试结果：

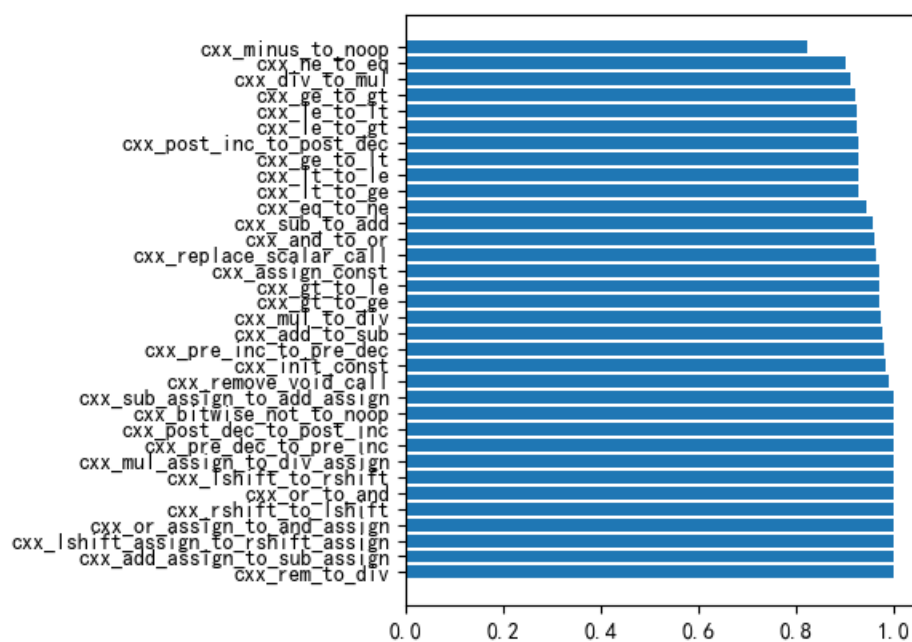
strip中导致变异杀死的变异算子



strip中未导致变异杀死的变异算子

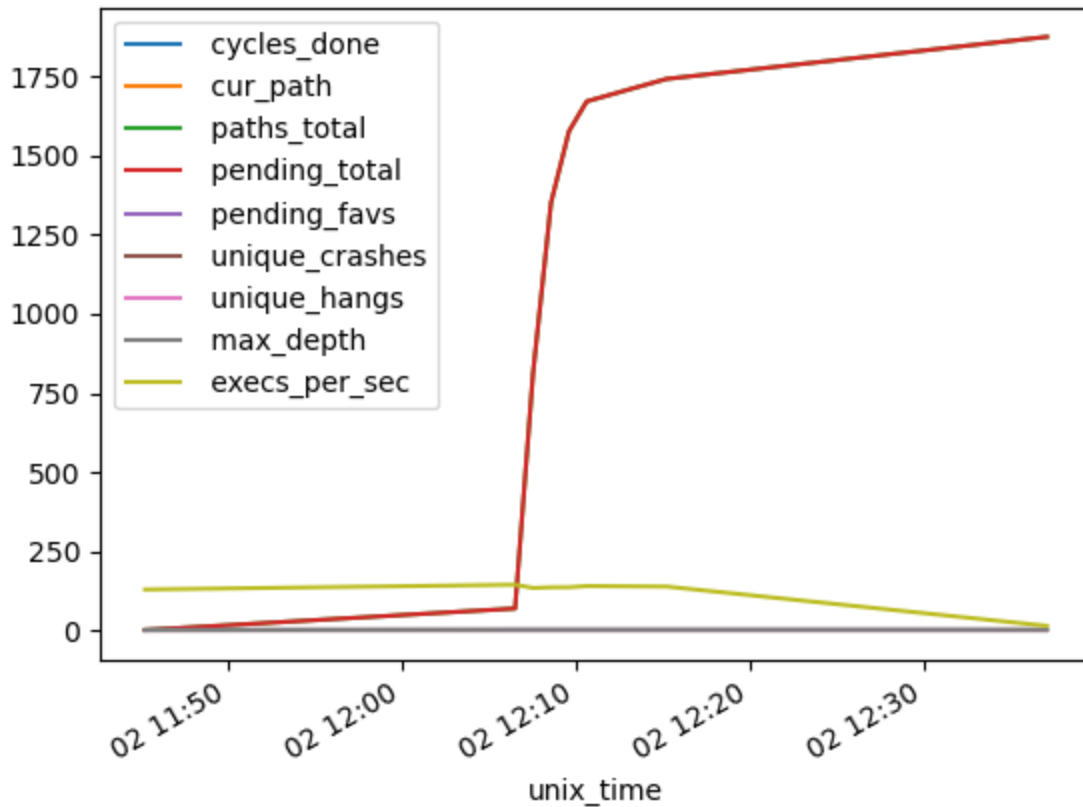


strip中变异算子被杀死的比例



xpdf

模糊测试过程：



3.2 结论

1.通过上述项目afl模糊测试过程结构可知，afl工具的执行效率较高，每个程序都能在每秒有大于750次执行，此外，由于待测程序本身质量较高，因此模糊测试过程中，并未得出能让待测程序崩溃的测试用例。而当AFL模糊测试过程进行充足时间后，可以探测出大量的独特的原程序执行路径，生成的测试用例保存在queue文件中。

2.在本次实验，变异体被杀死条件是测试用例使得变异体崩溃。而AFL在进行模糊测试过程中，因为测试的对象基本都是广泛在真实世界使用的程序，因此其bug较少，导致存储导致原程序崩溃的测试用例的crashes文件几乎没有结果。因此将queue文件夹中存有的测试用例用于mull的输入，其导致变异体崩溃的概率较小是正常的，因此变异得分低并不能说明AFL的效果不佳。

3.在本次实验中，我们进行了差分测试，以测试用例为基准，计量有多少测试用例的在不同程序上的输出有区别。以此虽不能得出变异得分，但也能反映出AFL产生的测试用例的质量，通过差分测试，我们发现，AFL的效果较好。

4. 目录结构

```
D:\.  
├──.idea  
│   └──inspectionProfiles  
├──afl_fuzz_out 存储afl模糊测试后的过程描述结果  
├──afl_images 存储afl进行模糊测试后的绘图结果，以及绘图脚本  
│   ├──exxfilt  
│   ├──nm  
│   ├──objdump  
│   ├──readelf  
│   ├──size  
│   └──strip  
├──mull_img 存储mull进行变异测试后的绘图结果，以及绘图脚本  
│   ├──exxfilter  
│   ├──nm  
│   ├──objdump  
│   ├──readelf  
│   ├──size  
│   └──strip  
├──mull_result 存储变异测试过程中，控制台重定向后的结果  
├──original_result 存储模糊测试以及变异测试后的所有原始结果（不止是用于绘图  
差分测试 过程的数据）
```

