实验报告

功能描述

将 Python 代码的过程间调用图展现出来

需求分析

输入为 Python 抽象语法树或者 Python 源代码输出为过程间调用关系图

PyCallGraph 通过记录程序运行 trace 来构建调用关系。然而现实中许多 Python 项目因为缺少外部依赖或者存在运行时错误等原因无法运行程序，这将限制 PyCallGraph 的使用。

本实验使用静态分析的方法构建

概要设计

1）静态分析：抽象解释

2）动态分析：求解器求解程序输入

详细设计

分为四个步骤

1. 生成抽象语法树
2. 简化 AST，去除控制流信息，如 for 循环、while 循环、if 语句等。
3. 解析 AST 获取全局定义，类似 Python 的 import 语句。
4. 解析 AST 获取过程间调用关系

实现代码请见InterPy

实验 1

输入程序为以下源代码

```
class Apple:
    def __init__(self, name):
        self.name = name
    def color(self):
        lname = self.name.lower()
        if lname == 'gala' or lname == 'fuji':
            self.red(self.name)
            return 'red'
```

```
        else:
            self.green(self.name)
            return 'green'

    def red(self, rname):
        print('Are u want to eat red apples? so '+ rname + ' is red. Plz')
    def green(self, gname):
         print('Are u want to eat green apples? so '+ gname + ' is red. Plz')

def main():
    apple = Apple('fuji')
    color = apple.color()
    print('The color of this apple is: '+ color)

if __name__ == '__main__':
    main()
```

输出为关系图

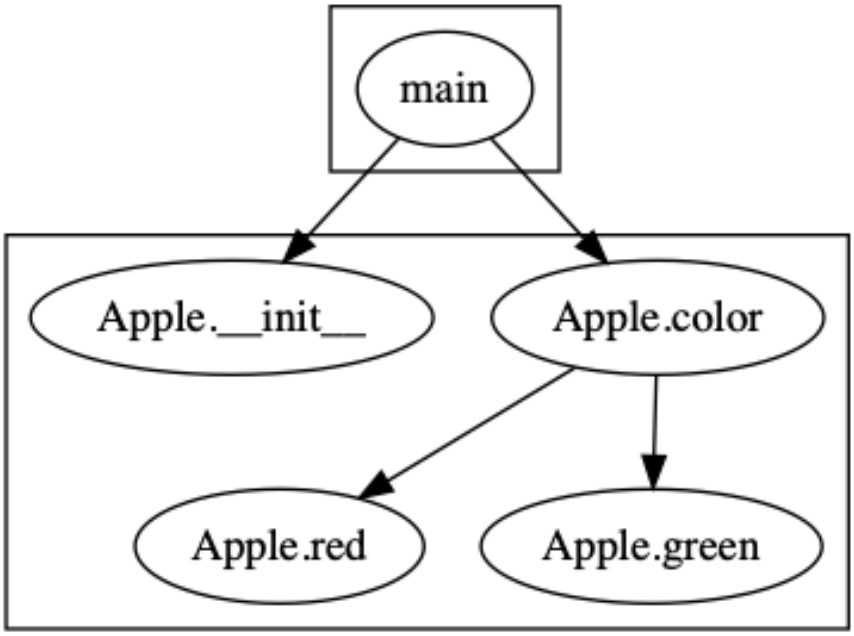

Figure 1:

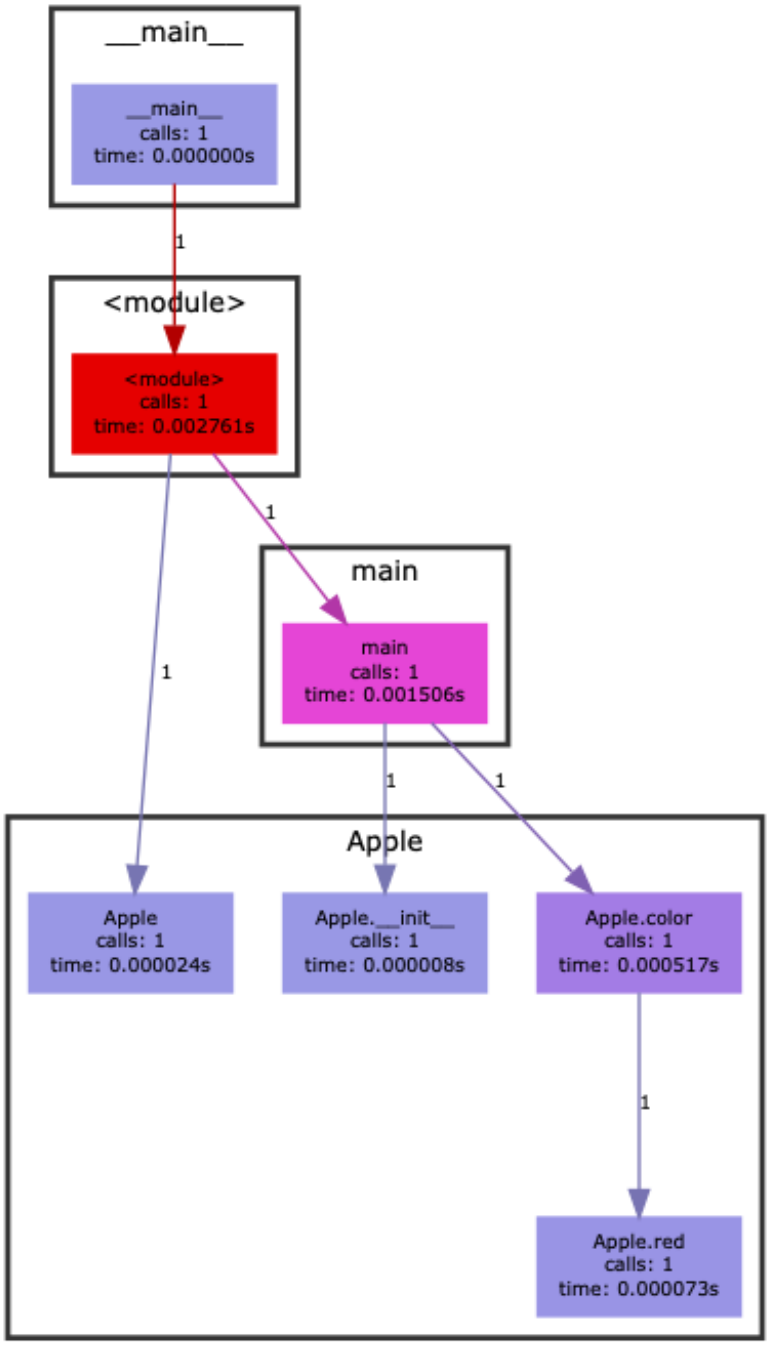
__main__
__main__
calls: 1
time: 0.000000s
1
<module>
<module>
calls: 1
time: 0.002761s
1
1
main
main
calls: 1
time: 0.001506s
1
1
1
Apple
Apple
calls: 1
time: 0.000024s
Apple.__init__
calls: 1
time: 0.000008s
Apple.color
calls: 1
time: 0.000517s
1
Apple.red
calls: 1
time: 0.000073s
Generated by Python Call Graph v1.0.1
http://pycallgraph.slowchop.com

Figure 2:

与 **pycallgraph** 对比

可见静态分析（interpy）在一般情况下的代码覆盖率比动态分析（pycallgraph）高。

实验 **2**

分析本项目内的源代码文件 `./interpy/visitor1.py`

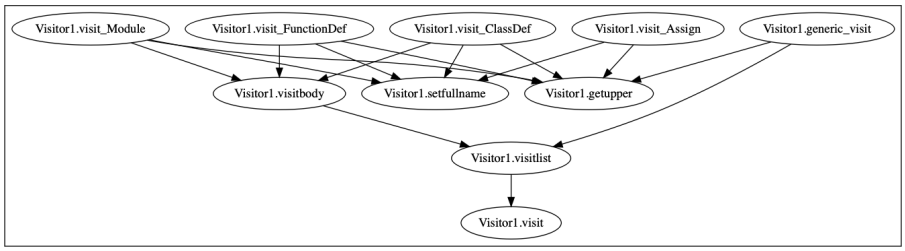

Figure 3:

实验 **3**

分析 requests 项目下源代码文件的 'models.py'

总结

难点

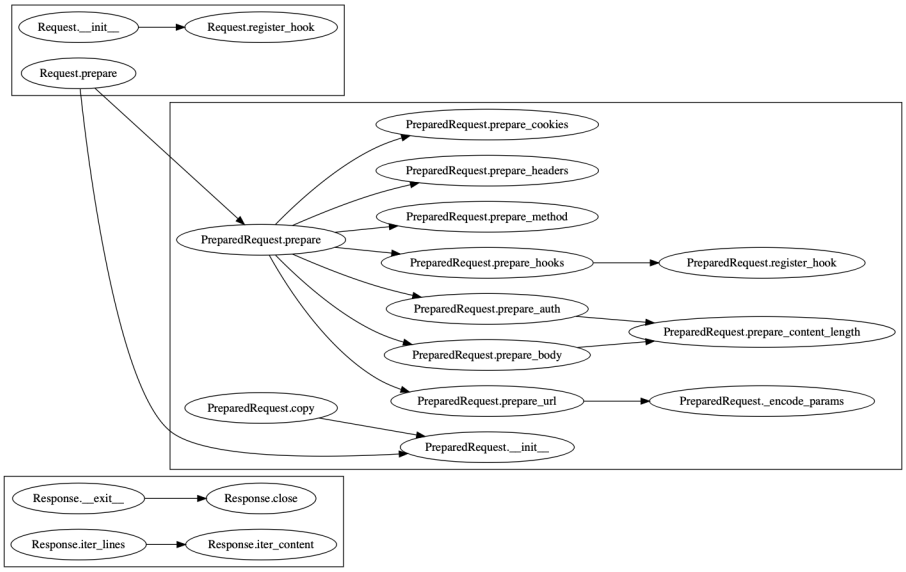
Request.__init__
Request.register_hook
Request.prepare
PreparedRequest.prepare_cookies
PreparedRequest.prepare_headers
PreparedRequest.prepare_method
PreparedRequest.prepare
PreparedRequest.prepare_hooks
PreparedRequest.register_hook
PreparedRequest.prepare_auth
PreparedRequest.prepare_content_length
PreparedRequest.prepare_body
PreparedRequest.prepare_url
PreparedRequest._encode_params
PreparedRequest.copy
PreparedRequest.__init__
Response.__exit__
Response.close
Response.iter_lines
Response.iter_content

Figure 4: