软件测试第二次作业——结题报告

解温特 MF1833032

工具名称：

利用 CFG 生成与符号执行进行可执行路径与测试用例生成

主要功能描述：

本次实验主要利用了 soot 工具，将一段待测试代码进行分析处理，能够输出一个对应的控制流图，然后对控制流图进行分析计算得到其圈复杂度，接着分析输出其可执行路径，最后对可执行路径进行符号执行，旨在分析输出其测试用例。

输入：

一段待测试的代码

输出：

待测试代码的可执行路径，或者其测试用例

实验过程说明：

本次实验工程代码分为 Project2 与 Test 两个文件夹，其中第一个 Project2 里的 Generate 为测试 soot 工具安装是否成功的测试代码，也就是能否成功生成一段代码的控制流图，结果为成功。此处需要注意，path1 和 path2 分别要根据实际的测试代码所处的 bin 路径进行更改，因为要找到其 class 文件。

另一个 Test 里存放了待测试代码模块，以及用于转化.dot 文件为.png 文件的模块代码。

而成功输出的.dot 控制流图在 Test 文件夹路径下查看，上图的测试过程中利用的是 for 循环，旨在输出其控制流图。

而第一个 Project2 里的 Test.java 为整个流程的驱动代码，即读取一段待测试代码后，利用安装好的 soot 工具进行 java 程序分析，调用 CFGToDotGraph 类的 drawCFG()方法，便可以输出其对应的控制流图，可以选择以 Block 作为最小分析单位，也可以以 Unit 作为最小分析单位，本次实验主要选取 Unit 作为重点，即具体的每一条执行语句为一个最小执行单元。

在分析输出控制流图之后，便有了一个可以继续分析的中间结构。本次实验利用 V(G)=e-n+2p 公式计算此图的圈复杂度，其中 p 为图的连接组件数目，控制流图是流通的所以此处 p 为 1。圈复杂度是衡量一个模块判定结构复杂程度的重要通道，也是评判一段待测试代码的重要参数之一。

紧接着，对控制流图进行深度优先遍历，每遇到一个诸如 if 的判定结构，都能够将其加入到 paths 这个 List 中，最终很容易得到每段待测试代码的可执行路径，这个路径可以直接输出（若为 if 语句，则输出对应的 if 和 else 两条可执行路径）。

最后，对可执行路径进行符号执行，可以得到该段待测试代码的路径约束，即要走该段代码的某一条路径需要满足的情况，这个路径约束将决定着这段代码的测试用例结果，此结果便可以很清楚地判定出代码的具体约束情况，对测试很有帮助且很有意义。

实验步骤：

1、首先将待测试代码段放入 Test 文件夹下面的 symbol 包内，可以测试 for 循环、if 语句，while 循环等等；

2、执行 Project2 下的 Generate.java 以测试 soot 是否能正常输出代码块对应的控制流图，.dot 的结果在 Test 文件夹路径下查看，若正常显示且内容正确则继续，此处可以使用

graphviz 工具将.dot 文件转化成.png 增强控制流图的可读性，在终端的命令为：dot -Tpng -o symbol_out2.png symbol_out2.dot；

3、其次执行 Project2 下的 Test.java，将想要测试代码路径（包名+类名）添加进来，尤其要注意，这里的路径必须是编译后的.class 文件所在的路径，这里便是将测试代码与测试模块连接起来；

4、最后便可以输出控制流图对应的圈复杂度、可执行路径与路径约束测试用例等，输出可直接在 Console 内容块中查看。

实验中遇到的问题：

本次实验首先遇到最大的困难便是 soot 工具的安装使用，最开始我使用网上教程的通用方法，即在 eclipse 里面点击 help 里的 install new software，输入 soot 网址后显示安装失败，于是向实验室内的学长学姐寻求帮助，最后使用了学姐提供的 soot 包作为 jar 包添加到我工程中的依赖包中成功解决了此问题；

其次遇到的大问题便是 eclipse 里显示的错误情况：java.lang.reflect.AnnotatedElement，在网上查看便是 jdk 版本与 soot 版本不匹配等兼容性问题，我电脑装的是 jdk1.8，而学姐提供的 soot 工具包要求的是 jdk1.7 版本，于是重新下载 jdk1.7 并在 eclipse 里设置 java 环境路径等，最终能够正常运行 soot 并生成控制流图。