Control Flow Graph Generator in Python

Steven Qin
20819760
s45qin@uwaterloo.ca

Huijie Chu
20862266
h25chu@uwaterloo.ca

Tiankai Jiang
20834939
t57jiang@uwaterloo.ca

May 28, 2020

Abstract

In 1970, Frances E. Allen proposed the idea of control flow graph(CFG). It is a graphical representation of the flow of the computation during the program execution. Since it represents the flow of logic, it can be used for static analysis or compiler optimization, etc. In this project, my team aims to build a static control flow graph generator from scratch using the concept of an abstract syntax tree. The essential part of the program will be written in Python with the assistance of Graphviz.

1 Problem Statement

The basic block is a series of program instructions that has only one entry point and one exit point. And the control flow graph is a directed graph in which the vertices represent the basic blocks and the edges represent paths between adjacent vertices. Using CFG, the control flow of the program can be shown easily, making it easier for developers to reveal logic error or expose paths that are not covered by the current tests. Also, researches show that CFG can be used for mulware detection and software plagiarism detection. Thus, a neat and efficient control flow graph generator is required for software development purposes. In this project, a control flow graph generator for high-level representation code(code before being sent to interpreter or compiler) will be written in Python, with initial support for source code in Python, and may retrieve other languages support later. The generator is static, which means it will generate the graph without executing the input source code, and the generator should be highly efficient(the basic blocks in CFG are as few as possible and as large as possible).

2 Approach

There are mainly two steps for CFG generation. First, find the basic blocks of the source code and then transform them into a graph representation. To find the basic blocks of the code, simply find the beginning and the ending of a block. The beginning of a basic block includes the first instruction of the code, the next line of the conditional instruction and the function call. And the end of the block could be the conditional instruction, the return, throw, or similar exit keywords that contain the last instruction of the code. However, this method does not always generate maximal basic blocks, but usually sufficient.

Since a module called Abstract Syntax Trees(AST) is provided in Python, we can directly transform the parsed result of AST to CFG. However, the technique mentioned above is still useful if we want to generate CFG from other coding languages like C++.

An abstract syntax tree is a tree representation of the abstract syntactic structure of the source code, with each tree node represents a keyword, a variable, or an operator of the code. It contains basic block information. Thus, to generate CFG from Python code, we can start by converting the original code into an AST directly. In the fuzzing book, detailed algorithm is proposed to generate CFGs using AST.

The output format of the graph could be a dot file or an image generated by that dot file using a build-in package Graphviz in Python.

3 Framework and Toolchain

Since my team plans to build this CFG generator from scratch, there is no framework or toolchain required. We plan to use the AST package in Python for syntax parsing and Graphviz package for graphing.

4 Benchmarks

A semi-random selected LeetCode solution set (about 200 independent pieces, each contains about 10-40 LOC) in Python will be used to test out the program. The solution set has a moderate length and contains various conditional instructions, which are nearly impeccable for the testing purposes in this case. To test the runtime performance of the program, a few completed Python projects, larger in size, will also be used as a representation of benchmarks for larger source codes and it infers to approximately simulate the performance of daily application scenarios.

5 Plan of Action and Feasibility Analysis

Week	Action
3-4	Read papers & tutorials about CFG and Python AST and algorithm design.
5-8	Implementation of the control flow graph generator.
9	Testing
10-12	Add support to another language(C or Java).
13	Demo

6 Demo

The program is expected to recognize the correct control flow of the input source code and generate the correct CFG. The control flow is not restricted to basic keywords such as if, else, while under normal syntax, but also more complex syntax like list comprehension or lambda expression. And it should detect syntax errors in the source code.