Control-Flow Instability

Abhiruchi Karwa

March 25, 2019

1 Introduction

Instability in the control-flow of a program for the same test case renders a program unreliable which is not desirable and disastrous. This is because the program exhibits different control-flows for the same inputs due to differences in external factors.

This project proposes to identify such programs and test cases across which control-flow instability can be noticed, identifying factors that cause control-flow instability and possible approaches to generate these cases.

2 Motivation

The reliability and portability of a program is affected if the same program, run on same inputs behaves differently across different platforms[1] or after changing from single to double precision for a floating-point value.

This problem is interesting because we observe different executions that lead to disparity in evaluation of conditional statements such as *if statements* and loops in a program. When any branching decision in a program involves some floating point arithmetic, the path taken by the program thereafter can vary. This can happen due to various factors like the changes in floating-point representation, compiler optimization or different execution platforms [1].

3 Plan

For identifying the effects and occurrences of control-flow instabilities, it may be necessary to formulate programs or find snippets that display this behaviour. Furthermore, to identify the specific input test cases that affect these conditional statements, this procedure could be followed:

1. Take the comparison of values in the condition check for the control-flow, convert it to an equality condition such that we get an assert statement.
2. Solve the assert statement using a floating-point constraint solver to identify the interesting test cases

These test cases are interesting because they will demonstrate how a slight change in the factors mentioned earlier affect the floating-point arithmetic and in turn reveal the control-flow instability in the program. Additionally, using a floating-point constraint solver is necessary as obtaining floating-point values for test cases are important to observe changes in floating point arithmetic.

This is a potential methodology to obtain promising test cases to observe effects of changes in floating-point representation, compiler optimization or different execution platforms on the reliability of a program.

4 Summary

This project aims to achieve the following goals:

1. Identify programs or snippets of programs consisting unstable control-flows due to floating-point arithmetic and enumerate test cases which recognize these instabilities.
2. Present some strategies used to generate and identify the test-cases for recognizing the branching decision instabilities.
3. Identify the reasons behind these inconsistent branching decisions.

5 Conclusion

The project aims to identify the factors, test-cases and approaches to generate the said test cases that cause control-flow instabilities in a program and make it unreliable.

6 References

1. Gu, Y., Wahl, T., Bayati, M., and Leeser, M.: Behavioral non-portability in scientific numeric computing.