Master Thesis

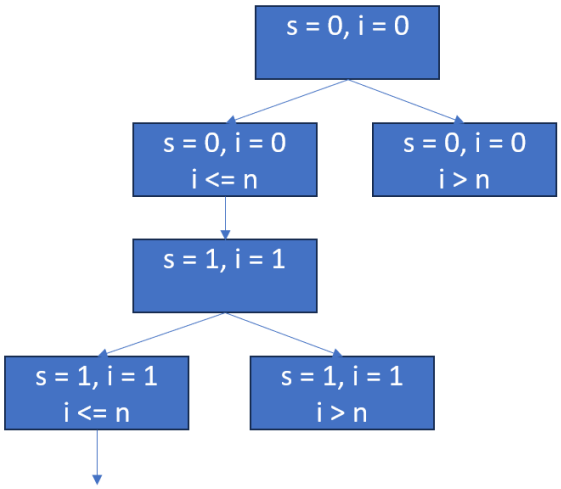
Symbolic Execution of Python Bytecode

Problem Statement

In software analysis, it is often helpful to know whether a certain program instruction is reachable (e.g., an error location) and which conditions need to be fulfilled (e.g., an input variable must be in specific range). One possible technique to analyze reachability is symbolic execution. The figure below shows a short function and simplified depiction of the infinite execution tree. In the second line of the node labels information about the branching conditions are stored as logical formulas.

In our research project, we develop a runtime verification framework with look-ahead capabilities for Python programs. We analyze Python programs not directly, but the generated Python Bytecode which is an assembly-like intermediate representation for Python programs. For the prediction of the program execution a symbolic execution-based analysis is needed.

```
def sum(n: int):
    s = 0
    i = 0
    while i <= n:
        s = i + s
        i = i + 1
    return s
```



SIMPLE PYTHON PROGRAM AND INFINITE EXECUTION TREE

Your Tasks

In this thesis a symbolic execution method for Python Bytecode should be developed and is to be evaluated using an industrial use case.

- Literature research on symbolic execution techniques for assembly-like languages
- Adaption and implementation of a promising approach which can handle pointer aliasing
- Evaluation on an industrial use case

Your Profile

The analysis should be implemented in Python and therefore intermediate language skills are preferred. Optional is knowledge about formal methods and symbolic execution, but general understanding of program analysis is helpful.

Contact

Thomas Henn, M. Sc. RWTH
henn@embedded.rwth-aachen.de