INTERPROCEDURAL PATH COMPLEXITY ANALYSIS

İ. Burak Kadron

March 22, 2017

University of California, Santa Barbara

OVERVIEW

Introduction & Motivation

Interprocedural Path Complexity Analysis

Future Work

Conclusion

Introduction & Motivation

- **Motivation**: Software verification and testing is focused on exploring all states of the program and traversing all paths. By finding an approximate for number of paths, we can find how difficult it is to cover all the paths.

INTRODUCTION

- **Motivation**: Software verification and testing is focused on exploring all states of the program and traversing all paths. By finding an approximate for number of paths, we can find how difficult it is to cover all the paths.
- **(Asymptotic) Path Complexity**: Introduced in Lucas et al.'s work on intra-procedural program analysis. Gives a measure of how complex a program is and how long the task of analysing a program will take.
- We improve upon Bang et al.'s work by doing inter-procedural program analysis and extract path complexity while taking recursion into account.

PREVIOUS WORK

- **Chomsky-Schutzenberger enumeration theorem**: If L is a context-free language and $a_k := |L \cap \Sigma^k|$ is the number of words in L with length k, $G(x) = \sum_{k=0}^{\infty} a_k x^k$ is a power series.

PREVIOUS WORK

- **Chomsky-Schutzenberger enumeration theorem**: If L is a context-free language and $a_k := |L \cap \Sigma^k|$ is the number of words in L with length k, $G(x) = \sum_{k=0}^{\infty} a_k x^k$ is a power series.
- If we have a grammar, we can replace each rule with a function and solve the function to obtain an approximation of power series.

PREVIOUS WORK

- **Chomsky-Schutzenberger enumeration theorem**: If L is a context-free language and $a_k := |L \cap \Sigma^k|$ is the number of words in L with length k, $G(x) = \sum_{k=0}^{\infty} a_k x^k$ is a power series.
- If we have a grammar, we can replace each rule with a function and solve the function to obtain an approximation of power series.
- Example:
- $S \to M \mid U$ $\quad S = M + U$
- $M \to 0M1M \mid \varepsilon$ $\quad M = M^2x^2 + 1$
- $U \to 0S \mid 0M1U$ $\quad U = Sx + MUx^2$

INTERPROCEDURAL PATH COMPLEXITY ANALYSIS

STEPS OF ANALYSIS

- **Control Flow Extraction**: We use Java bytecode analysis tools (Soot) to extract the Control flow graph.

STEPS OF ANALYSIS

- **Control Flow Extraction**: We use Java bytecode analysis tools (Soot) to extract the Control flow graph.
- **Conversion to Grammar**: We convert the graph to grammar using rules below.

STEPS OF ANALYSIS

- **Control Flow Extraction**: We use Java bytecode analysis tools (Soot) to extract the Control flow graph.
- **Conversion to Grammar**: We convert the graph to grammar using rules below.
- Our grammar contains a single character (k) in the alphabet.
- We convert the statements which don't invoke another function to k.
- We convert the statements which invoke functions we are analysing to their corresponding rule name.
- We convert the if statements to |(or) in the rules.

STEPS OF ANALYSIS PT. 2

- **Generating and Solving Equations**: According to Chomsky-Schutzenberger theorem, we convert our rules to functions by replacing | with +, characters with variable x, rules with function names.

STEPS OF ANALYSIS PT. 2

- **Generating and Solving Equations**: According to Chomsky-Schutzenberger theorem, we convert our rules to functions by replacing | with +, characters with variable x, rules with function names.
- **Generating Function**: We solve the generated function and expand the root to a power series with Taylor expansion.

STEPS OF ANALYSIS PT. 2

- **Generating and Solving Equations**: According to Chomsky-Schutzenberger theorem, we convert our rules to functions by replacing | with +, characters with variable x, rules with function names.
- **Generating Function**: We solve the generated function and expand the root to a power series with Taylor expansion.
- $g(x) = g(0) + \frac{x * g'(0)}{1!} + \frac{x^2 * g''(0)}{2!} + \cdots + \frac{x^n * g^{(n)}(0)}{n!} + \cdots$
- Coefficients of x^k denote number of paths in length k

EXAMPLE PROGRAM

```
public static int fibonacci(int n){
    if (n == 0 || n == 1){
        return 1;
    } else {
        return fibonacci(n-1) + fibonacci(n-2)
    }
}
```

EXTRACTED CFG

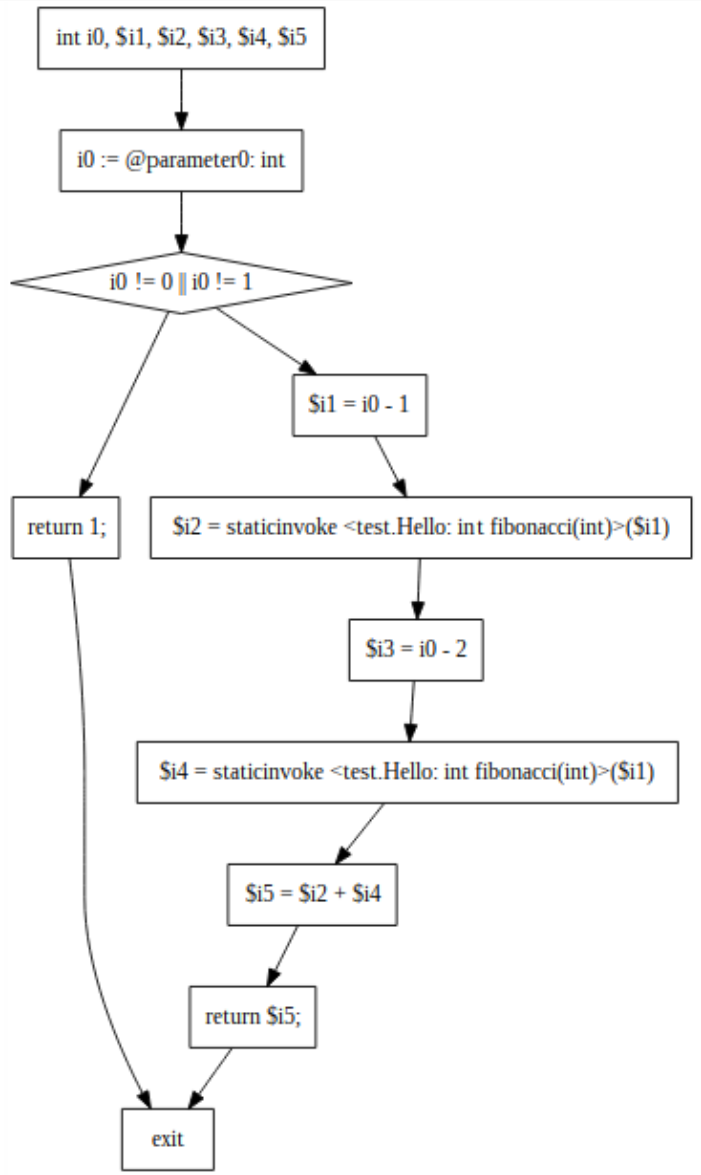

Figure: Fibonacci in CFG form

CONTEXT-FREE GRAMMAR FOR PROGRAM

- $F = x \mid x^4 * F^2 \rightarrow F(x) = x + x^4 * F^2(x)$ (Conversion to equations)
- $F(x) = \dfrac{1 + \sqrt{1 - 4x^5}}{2x^4}$ (Finding the root)

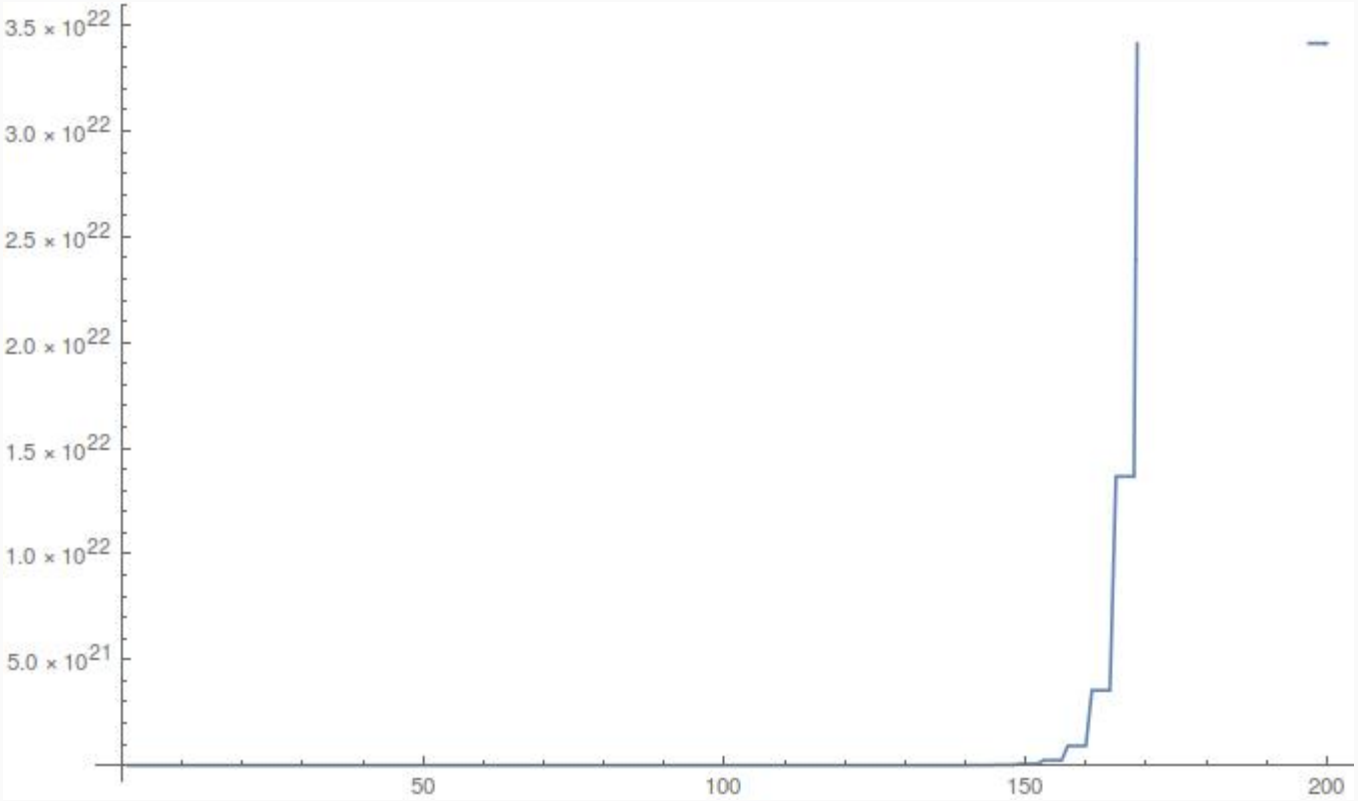

Figure: X axis denotes length of words, Y axis denotes number of words with length X or less

FUTURE WORK

FUTURE WORK

- **Improvement for Overapproximation**: For Quicksort, our analysis gives an exponential upper bound like k^n which is an overestimate given its complexity is n^2 in worst case.

FUTURE WORK

- **Improvement for Overapproximation**: For Quicksort, our analysis gives an exponential upper bound like k^n which is an overestimate given its complexity is n^2 in worst case.
- Unnecessary paths may be pruned out or variables can be simulated to reduce overapproximation.

FUTURE WORK

- **Improvement for Overapproximation**: For Quicksort, our analysis gives an exponential upper bound like k^n which is an overestimate given its complexity is n^2 in worst case.
- Unnecessary paths may be pruned out or variables can be simulated to reduce overapproximation.
- **Comparing our estimations**: We claim this metric is useful in estimating execution time of testing for full path coverage. We can compare our results to running times of testing tools and check the correlation.

CONCLUSION

SUMMARY

We have presented an analysis method built upon automata theory and word counting to estimate path complexity.

Although some improvements can be done, we believe it demonstrates some theoretical way to analyse the control flow graphs.

REFERENCES

- Bang, Lucas, Abdulbaki Aydin, and Tevfik Bultan. ”Automatically computing path complexity of programs.” Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering. ACM, 2015.
- Lam, Patrick, et al. ”The Soot framework for Java program analysis: a retrospective.” Cetus Users and Compiler Infastructure Workshop (CETUS 2011). Vol. 15. 2011.

QUESTIONS?