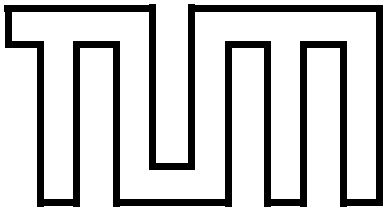


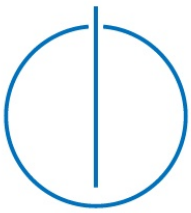
DEPARTMENT OF INFORMATICS

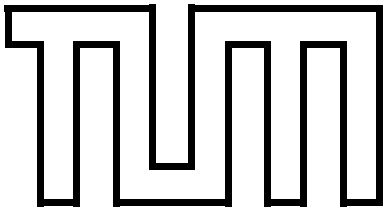
TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Inferring String Properties from Code Property Graphs

Severin Schmidmeier





DEPARTMENT OF INFORMATICS

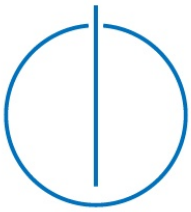
TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Inferring String Properties from Code Property Graphs

Herleitung von Eigenschaften von Strings aus Code Property Graphen

Author:	Severin Schmidmeier
Supervisor:	Prof. Dr. Claudia Eckert
Advisor(s):	Alexander Küchler, Florian Wendland
Submission:	15.03.2023



I confirm that this bachelor's thesis is my own work and I have documented all sources and material used.

Ich versichere, dass ich diese Bachelorarbeit selbstständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

München, 15.03.2023

(Severin Schmidmeier)

Acknowledgments

Thanks everyone!

Abstract

In the last couple of years, I have supervized numerous bachelor's and master's thesis and various seminars. This led to a broad observation of typical questions and issues the students faced when writing their thesis or papers. Surprisingly, they are always quite similar. This template aims to give advise to future sstudents in order to answer the most frequent questions and avoid the most common mistakes. It provides the TUM template which has already been accepted many times, shows the most basic outline and some tips on the contents of each chapter. It further contains some tips on the style of scientific works. An evaluation on a small set of students showed that this guideline can assist in making progress faster. However, we found that we have to keep improving the tips to achieve better results.

Your abstract goes here. The typical structure is:

- Broad description of the current state
- Gap in the current state
- Your contribution

Contents

1 Introduction

Your introduction goes here

- Generic description of the broad field of research
- Current state of research
- What's the gap that you're trying to fill?
- Short motivation
- Summary of the most important results
- Your contribution
- Structure of the thesis

1-2.5 pages

This text is not too detailed. Start quite high-level, then narrow down until you reach your topic. After the introduction, the reader must want to read the rest of your thesis and understand the relevance. However, it doesn't have to be super technical.

2 Problem Description

The introduction is a bit like a teaser. Here, you dig more into details, also technical ones. After this chapter, the reader must understand why you do this work, why it's important, what makes it difficult and what you want to achieve.

- What's the problem that you're trying to solve?
- What is your goal?
- What is/are the research question(s)?
- What are special problems?

Probably 1-3 pages

3 Background

The library[1] we extend in this thesis extracts a Code Property Graph (CPG) out of source code of a set of different programming languages.

The CPG is a directed multi graph, where the nodes represent syntactic elements like simple expressions or function declarations and the edges represent the relations between those elements. The nodes and edges have a list of key - value pairs called properties which contain general information for the element. For example, a Node representing a statement in a source file contains the location of the underlying code and an edge representing evaluation order may contain whether the target statement is unreachable. The graph is initially created by language frontends, which create partially connected abstract syntax trees (ASTs), which are then enriched by additional information like the mentioned evaluation order by multiple passes [4].

Users of the library can extend this functionality by adding additional passes, which is how we implement the hotspot collection in this thesis.

While the CPG contains many different types of edges, the most relevant edge type for this thesis are data flow edges, which represent the data flow between different expressions.

```
String s = "xyz";
System.out.println(s);
```

Consider the short code example in listing 3. Here, amongst others, the following nodes are part of the CPG:

- `Literal`, representing the string literal `"xyz"`
- `VariableDeclaration`, representing the declaration and initialization of the variable `s`
- `DeclaredReferenceExpression`, representing the reference to the variable `s` in line 2.

In this example, the data flows from the `Literal` node to the `VariableDeclaration` and from there to the `DeclaredReferenceExpression`.

[1] https://github.com/Fraunhofer-AISEC/cpg

The nodes connected by those egdes effectively form a subgraph of the CPG, the data flow graph (DFG), from which we then extract the information on string values.

4 Approach and Implementation

- What's your general approach to answer your questions?
- What were different design decisions/options? Which one did you chose? What are the pros and cons?
- What did you implement? (including a description of your PoC)

This is most likely your main part! Probably 10-25 pages. Can also be split up into two or more chapters depending on your work. Probably, it makes sense to break down the title of your thesis into multiple components and make a section for each of them.

It often makes sense to start with a description of the steps of your program with a nice (yet often similar) visual representation as shown in Figure 4.1. If you're not familiar with `tikz` and so on, I recommend using draw.io[1] and export the figure as pdf. (Hint: you can also track the .drawio file in git — it's a simple xml format).

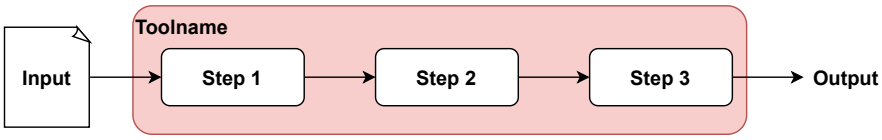


Figure 4.1: Workflow of my amazing tool

Similarly, create tables and please, always explain and reference the tables, figures, listings in the text with a reference. Examples are Table 4.1 summarizing the medals won during the Olympics 2021 for the top 3 nations or Listing 4.1 calculating the factorial of the input.

Table 4.1: Olympics 2021: Medals of top 3 nations.

	Gold	**Silver**	**Bronze**
USA	39	41	33
China	38	32	18
Japan	27	14	17

[1] `https://app.diagrams.net/`

Listing 4.1: Function computing the factorial

```
def frac(n):
    if n == 1:
        return 1
    return n * frac(n-1)
```

5 Evaluation and Discussion

- How did you test/evaluate your PoC?
  - E.g. case studies, large-scale studies, test bench, etc.
  - What did you do to verify results (if applicable)
- What did you learn from these tests? Depends on your work. E.g.
  - TP/TN/FP/FN rates
  - Performance
  - Results of your studies
  - Interpretation of the results, lessons learned
- Limitations of the approach and your implementation. Any ideas on how to fix them?

Probably 5-15 pages

6 Related Work

The challenge of statically obtaining information about the values of strings is not new and over the years there have been different approaches to it.

The approach we generally follow in our implementation is described in [1] . The authors also construct a context free grammar from a flow graph, but instead of creating it on-demand, starting at the chosen hotspot node like we do, they consider the total flow graph for grammar creation. They use the same approximation methods for obtaining regular languages from the generated context free grammars, but instead of making the regular languages available as a regular expression they generate automata. Furthermore they introduce a novel formalism, the MLFA (multi-level automaton) which allows easy extraction of these automata for different hotspots. Due to the aforementioned on-demand generation of the grammar, we don't need this extraction for single hotspots the MLFA provides in our implementation. The authors provide a feature rich implementation[1] of their approach and show that it efficiently produces useful results.

In [2] the authors describe a type system for a minimal functional calculus, where strings have a regular expression as their type. They show that their proposed type system can produce good results when applied to their minimal calculus. While we considered implementing this approach for the analysis, there are some problems, especially due to our different requirements and prerequisites.

To use the presented approach in practice an (efficient) algorithm for type checking and type reconstruction is needed. The given paper does not include those, but rather indicates several problems in constructing such algorithms for the given situation without losing some of the desired preciseness. The authors mention that using standard type reconstruction by constraint solving for the proposed type system even is impossible due to limitations of regular languages.

Additionally this approach is tailored to the mentioned calculus and utilizes specific features like pattern matching, which would make adapting it to our use case more difficult.

The additional layer of abstraction introduced by the DFG used in the approach we chose eliminates this problem and makes adaption easier.

[1] https://www.brics.dk/JSA/

In [3] Wassermann and Su present an approach comparable to ours, where they also characterize values of string variables using context free grammars. They specifically target SQL injection vulnerabilities by using the generated CFGs to check whether user input can change the syntactic structure of a query. While this approach is successful in detecting those vulnerabilities, our approach is more general and not focused on detecting one specific type of problem but rather on providing general information for unspecified further use.

7 Conclusion

Summarize your main contributions and observations. Further research directions?
≤ 1 page

Bibliography

[1] A. S. Christensen, A. Møller, and M. I. Schwartzbach. "Precise Analysis of String Expressions." In: *Proc. 10th International Static Analysis Symposium (SAS)*. Vol. 2694. LNCS. Available from `http://www.brics.dk/JSA/`. Springer-Verlag, June 2003, pp. 1–18.

[2] N. Tabuchi, E. Sumii, and A. Yonezawa. "Regular Expression Types for Strings in a Text Processing Language." In: *Electronic Notes in Theoretical Computer Science* 75 (2003). TIP'02, International Workshop in Types in Programming, pp. 95–113. ISSN: 1571-0661. DOI: `https://doi.org/10.1016/S1571-0661(04)80781-3`.

[3] G. Wassermann and Z. Su. "Sound and precise analysis of web applications for injection vulnerabilities." In: *ACM-SIGPLAN Symposium on Programming Language Design and Implementation*. 2007.

[4] K. Weiss and C. Banse. *A Language-Independent Analysis Platform for Source Code*. 2022. DOI: `10.48550/ARXIV.2203.08424`.