UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Faculty for Computer Science, Electrical Engineering and Mathematics
Department of Computer Science
Research Group Secure Software Engineering

Master's Thesis Proposal
Submitted to the Secure Software Engineering Research Group
in Partial Fulfilment of the Requirements for the Degree of
Master of Science in Computer Science

TypeEvalPy — Common Evaluation Framework with Benchmarks for Type Inference

by
Samkutty Sabu

Thesis Supervisor:
Ashwin Prasad Shivarpatna Venkatesh

Paderborn, March 1, 2023

Erklärung [redacted]

Abstract. Type inference is a key feature in static analysis that helps developers detect errors early, improve code quality, and enhance maintainability. Python is a dynamically-typed language, which means that types are inferred at runtime, making it challenging for developers to ensure type safety. Several type inference tools have been developed to help overcome this issue. However, the effectiveness and efficiency of these tools depend on various factors, such as the complexity of the code and the nature of the problem being solved. Therefore, it is crucial to have a benchmark framework to evaluate and compare the performance of these tools. Existing research has primarily focused on evaluating type inference tools using real-world benchmarks or custom benchmarks created by the researchers. While these methods are useful, they do not consider the importance of micro-benchmarks in evaluating tool performance. Moreover it is needed to establish a common result format to facilitate the comparison and interpretation of the benchmark results for all tools. To address these issues, the Benchmark for Type Evaluation (TypeEvalPy) proposes the establishment of a common framework for evaluating type inference tools. The proposed framework includes both micro-benchmarks and real-world benchmarks, which can provide a comprehensive evaluation of the performance of the tools. Additionally, the framework will also provide a common framework for tools analysis and results, which can facilitate the comparison of results from different tools.

Contents

1 Introduction

Type inference is a crucial part of static analysis, and it plays a vital role in improving the quality and reliability of software. Python is a dynamically typed programming language, which means that the data type of a variable is determined at runtime rather than at compile-time. This dynamic nature of Python poses a challenge for static analysis tools as they cannot determine the data type of variables without executing the code. As a result, type inference in Python is a complex and challenging problem.

There has been significant research in developing type inference tools for Python. These tools aim to improve the efficiency and accuracy of type inference in Python. However, evaluating the performance of these tools is a challenging task due to the dynamic nature of Python. Real-world programs are often complex, and evaluating the performance of type inference tools on these programs can be time-consuming and resource-intensive.

To address this challenge, micro-benchmarks can be a good method to evaluate the performance of type inference tools. Micro-benchmarks are small code snippets that can be used to test specific language features and constructs. These code snippets are designed to be easy to execute and evaluate, and they can be used to quickly identify the strengths and weaknesses of type inference tools.These benchmarks are essential because they can help identify performance issues in specific language constructs that may not be apparent in larger programs. Moreover, micro-benchmarks can be used to evaluate the performance of different type inference tools against specific language constructs, making it easier to identify which tool performs better

Despite the benefits of micro-benchmarks, most developers and researchers still rely on real-world benchmarks to evaluate the performance of type inference tools. Real-world benchmarks are often considered a more accurate representation of the performance of type inference tools as they evaluate the tools on real-world programs. However, using real-world benchmarks for evaluation can be challenging due to the complexity of the programs and the time and resources required to evaluate the tools.

Another issue that needs to be addressed is the lack of a common result format for type inference tool evaluations. Existing research often reports tool performance using different metrics, making it challenging to compare and interpret the results of different studies. Therefore, it is crucial to establish a common result format that can be used by all researchers to facilitate comparison and interpretation of the benchmark results for all tools.

This research aims to provide a benchmark framework for evaluating type inference tools for Python that incorporates both micro-benchmarks and real-world benchmarks. The primary objective of this research is to develop a comprehensive and standardized benchmark framework that can be used by developers to evaluate the performance of type inference tools. The framework also provides a common result format for tools analysis and results, making it easier to compare and interpret the results of different studies.

To achieve this objective, this research has three main goals:

1. Developing a micro-benchmark: The first goal is to develop a micro-benchmark for evaluating the performance of specific language features and constructs in Python. This will involve identifying a set of commonly used language features and constructs and developing code snippets to test them.

2. Aggregating real-world benchmarks: The second goal is to aggregate existing real-world benchmarks and extend them to cover a wide range of Python code. This will involve identifying a set of real-world Python programs and developing test cases to evaluate the performance of type inference tools on these programs.

3. Developing a common framework: The third goal is to develop a common framework for evaluating type inference tools that incorporates the micro-benchmark and real-world benchmarks. This framework will provide a standardized way of evaluating the performance of type inference tools and will also include a common result format for reporting the results of the evaluations.

The research questions that will be addressed as part of this research are:

1. How do existing type inference tools for Python perform on the developed micro-benchmark?

2. How consistent are the results of type inference tools when tested on a common micro-benchmark compared to real-world benchmarks, and how can a common benchmark improve the accuracy of evaluating the performance of type inference tools?

3. How does the developed common evaluation framework improve the consistency and comparability of type inference tool performance evaluations?

In summary, this paper proposes a benchmark framework for evaluating type inference tools that includes both micro-benchmarks and real-world benchmarks. The framework will provide a common result format for tools analysis and results and evaluate the performance of type inference tools. The proposed benchmark framework will be built on top of Scalpel, a static analysis tool for Python code. Scalpel has been used in various research studies to evaluate the performance of type inference tools. Therefore, using Scalpel as the basis for the proposed benchmark framework will provide a solid foundation for the research.

2 Problem Description

Static analysis is a well-established technique for detecting programming errors and improving software quality. One of the key components of static analysis is type inference, which involves determining the types of variables and expressions in a program without running the program. This information can be used to detect type-related errors, improve program comprehension, and enable other program transformations. However, the performance of type inference tools varies significantly based on the code's structure, the type of variables used, and other factors. This makes it challenging for developers to evaluate the performance of different type inference tools effectively.

Several type inference tools are available for Python, including PYInfer, HiTyper, Typilus, and Typewriter. These tools vary in their approach and effectiveness, with some focusing on specific types of programs or language features.

To evaluate the performance of these tools, a benchmark framework is needed that incorporates both micro-benchmarks and real-world benchmarks. The micro-benchmarks focus on specific language features and constructs, while the real-world benchmarks cover a wide range of Python code.

Some popular real-world benchmarks include the ManyTypes4Py and Type4Py datasets, the typePY dataset from PYInfer, the Typilus benchmark suite, and the OSS dataset of Typewriter.

Developers need to be able to evaluate the performance of type inference tools accurately to choose the most appropriate tool for their specific use case. However, there is currently no standard benchmark framework available for evaluating the performance of type inference tools in Python. The lack of a standard benchmark framework makes it challenging for developers to compare the performance of different type inference tools effectively. This issue can result in developers using suboptimal tools, which can lead to poor program performance and errors.

3 Thesis Goals

The primary objective of this thesis is to provide a benchmark framework that can be used by developers to evaluate type inference tools for Python. To achieve this objective, the following four main goals will be pursued:

3.1 Developing a Micro-benchmark

The first goal is to develop a micro-benchmark for evaluating the performance of specific language features and constructs in Python. This will involve identifying a set of commonly used language features and constructs and developing code snippets to test them. The micro-benchmark will also need to be designed in a way that can be easily extended to incorporate new language features and constructs as needed.

3.1.1 Diverse benchmark

The first goal of this research is to create a more comprehensive and diverse benchmark suite that covers a broad range of Python language features and constructs. This is essential to test the performance of type inference tools in different scenarios and to ensure that the tools can handle a wide variety of code snippets.

To achieve this goal, two existing benchmark suites will be combined: PyCG's benchmark suite and Scalpel's benchmark suite. PyCG's benchmark suite contains a comprehensive set of code snippets categorized into subdirectories based on the type of language feature, while Scalpel's benchmark suite includes a smaller set of test cases with ground truths for expected output. By integrating these two benchmark suites, a larger and more diverse benchmark suite can be created that covers a wide range of language features.

To combine the benchmark suites, the Scalpel test cases will first be organized into appropriate subdirectories within the PyCG suite based on their language feature. Next, the PyCG suite will be reviewed to identify any gaps in the coverage of language features, and additional test cases will be constructed to fill those gaps. The result will be a comprehensive benchmark suite that covers a broad range of Python language features and constructs.

3.1.2 Develop ground truths

Once the benchmark code snippets have been categorized, the next step is to develop the ground truth data for each benchmark test case. The ground truth data is essential to evaluate the performance of each type inference tool being tested. When a tool's inferred types match the expected output types, it will be deemed successful in correctly inferring the types for that code snippet.

Scalpel benchmark suite provides ground truths for each test case with their expected output types. To extend this to the combined benchmark suite, ground truths must be generated for each code snippet. The process involves using a type inference tool to generate a set of inferred types for each code snippet and manually converting the results into the required format.The resulting ground truth data will enable a standardized evaluation of the performance of different type inference tools, making it easier to compare their results and identify areas for improvement.

3.1.3 Type annotations

In addition to the development of ground truths, the micro-benchmark will include type annotations to aid in proper type inference by various tools. As Python is a dynamically typed language, the use of type annotations is a relatively new feature, introduced in recent versions of the language. Type annotations will be utilized to specify expected variable and function argument types in the code.

Type annotation tools will be used to automatically generate type annotations for each variable and function argument in the code snippets. The annotations will be formatted in a manner that can be easily parsed and interpreted by the type inference tools being evaluated, following the new convention of Python coding, specifically PEP 484. Maintaining a copy of each code snippet with type annotations added will provide an additional option for tools that require type annotations to infer types. This will ensure that each tool is evaluated fairly and accurately, regardless of its ability to infer types without annotations.

3.2 Real-world benchmarks

The second goal is to aggregate existing real-world benchmarks and extend them to cover a wide range of Python code. This will involve identifying a set of real-world Python programs and developing test cases to evaluate the performance of type inference tools on these programs.

One way to aggregate these benchmarks is by utilizing existing datasets, such as ManyTypes4Py, Type4Py, PYInfer, Typilus, and TypeWriter. These datasets have been specifically curated to include a large number of Python programs with type annotations and are therefore ideal for evaluating the performance of type inference tools.For example, the ManyTypes4Py dataset contains over 5,000 Python projects with more than 869K type annotations, all of which have mypy as a dependency. The Type4Py dataset includes over 5,000 Python projects with 869K type annotations and used Pyre for type inference, resulting in 3.3 million type annotations.

3.3 Common Evaluation Framework

The third goal of this research is to develop a common evaluation framework that integrates both the micro-benchmark and real-world benchmarks to provide a standardized way of evaluating the performance of type inference tools. This framework will include a common input and result format for reporting the results of the evaluations.

3.3.1 Integration of Benchmarks

The initial step in creating a framework for evaluating type inference tools is to merge the micro-benchmark and real-world benchmarks. This framework will assess the performance of these tools across various Python language features and structures.

The goal is to construct a user-friendly interface that allows for the selection of either benchmark for evaluating type inference tools. The framework should smoothly integrate both benchmarks, ensuring an efficient and transparent testing process. This will enable users to choose the most fitting benchmark for their particular requirements and assess their tools' performance accordingly. The interface should also provide detailed information about the benchmark and evaluation results.

3.3.2 Common Input and Result Format

The second sub-goal is to develop a common input and result format for reporting the results of the evaluations. The input format will define how the code snippets are presented to the type inference tools, while the result format will define how the output of the tools is reported. Developing a common format will ensure that the evaluation results are consistent and can be easily compared across different tools.

Designing a common input format for type inference tools can be difficult since each tool may have different input requirements. Thus, creating a flexible input format that can adapt to the different requirements of various tools is a crucial sub-goal of the common framework development. However, it may be possible that some tools may not fit into the common input format, and the framework may need to be limited to the requirements of existing tools. In such cases, future tools may need to adapt to the established input format of the framework.

The common result format for reporting the evaluation results of the type inference tools will include various information such as the success rate, inferred types, any issues or errors with the tool, and the time taken for inference. By providing this information, the framework will enable users to compare the performance of different type inference tools and make informed decisions about which tool is best suited to their particular use case. Furthermore, the standardized result format will make it easier to reproduce and share the results of evaluations across different research groups and projects.

Overall, the development of a common evaluation framework will provide a standardized way of evaluating the performance of type inference tools and enable a comprehensive comparison of their results. The use of a common input and result format will ensure that the evaluation results are consistent and can be easily compared across different tools.

3.4 Evaluation

3.4.1 Evaluation of tools

Evaluate the performance of existing type inference tools using the common framework Based on the goals of the research, some potential research questions for the evaluation are :

1. How do existing type inference tools for Python perform on the developed micro-benchmark?

2. How consistent are the results of type inference tools when tested on a common micro-benchmark compared to real-world benchmarks, and how can a common benchmark improve the accuracy of evaluating the performance of type inference tools?

3. How does the developed common evaluation framework improve the consistency and comparability of type inference tool performance evaluations?

3.4.2 Guidelines

Create a set of guidelines to help users utilize the common evaluation framework to assess the performance of type inference tools. These guidelines will be developed based on the insights gained from evaluating the performance of existing tools using the common framework. They will aim to provide clear instructions on how to use the framework effectively and interpret the evaluation results accurately.

Overall, the thesis work will provide a comprehensive and standardized benchmark framework for evaluating type inference tools for Python. The micro-benchmark and real-world benchmarks will cover a wide range of language features and constructs, and the common framework will provide a standardized way of evaluating the performance of type inference tools. The guidelines developed as part of the thesis work will also help developers effectively use the benchmark framework to evaluate the performance of type inference tools.

4 Thesis Structure

1. Introduction
   (a) Motivation
   (b) Problem Statement
   (c) Goals
2. Related Work
3. Problem Analysis
4. Methodology
   (a) Micro Benchmark
      i. Diverse benchmark
      ii. Ground truths
      iii. Type annotations
   (b) Real World Benchmark
   (c) Common Evaluation Framework
      i. Integration of Benchmarks
      ii. Common Input and Result Format
5. Implementation
6. Evaluation
7. Conclusion and Future Work
   (a) Limitations
   (b) Future work
8. Abbreviations
9. Bibliography

5 Time Plan

Task	Weeks
Micro Benchmark	
Diverse benchmark	1
Develop Ground Truths	3
Type Annotations	2
Real World Benchmark	
Integration	2
Common Framework	
Integration	1
Common input/output format	3
Evaluation	
Evaluating tools	2
Thesis Report	
Writing Thesis Report	6
Timeline Overview	
Total	20

Bibliography

[redacted]