Department of Computer Science
ETH Zurich

Department of Computer Science and Engineering
German University in Cairo

Bachelor Thesis

A Static Type Inference for Python 3

August, 2017

Author:	Mostafa Hassan
Supervisors:	Marco Eilers
	Dr. Caterina Urban
	Prof. Dr. Peter Muller

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

Zurich, August, 2017 Mostafa Hassan

Acknowledgments

Abstract

Contents

1 Introduction

"The cost to fix an error found after product release was four to five times as much as one uncovered during design, and up to 100 times more than one identified in the maintenance phase.", reported by the System Science Institute at IBM. This fact justifies the increasing investments in software analysis, software verification and the need to make programs more reliable and safe.

In Python, being a dynamically-typed language, the variables are bound to their types during the execution time. This may look appealing because programs have more type flexibility, and they do not need to contain the writing overhead for the type system, leading to shorter and quicker to write code. However, this comes at the cost of losing many static guarantees of program correctness. Dynamically-typed languages perform type checking at runtime, while statically typed languages perform type checking at compile time. Therefore, some type errors that can be detected at compile time in a statically-typed system, may lead the system to crash at runtime in a dynamically-typed one, incurring high costs and a harder debugging experience.

See the following example:

```
num = 1
num = num + "2"
```

The intention of the above program was to add the number 2 to the variable `num`, not the `string` representation of this number. This tiny mistake goes unnoticed at compile time, and leads the program to raise an exception during runtime.

In this thesis, we are presenting a tool for static type inference and static type checking for a subset of Python 3. The aim of the tool is to gain the benefits of static typing while maintaining some (yet not all) dynamic features of Python. We discuss later the details of the dynamic limitations imposed on the inferable Python programs.

The type inference is based on a nominal static type system that we define in the next chapter. The type inference is intended to be integrated into Lyra and VerifySCION, two ongoing projects at the Chair of Programming Methodology at ETH Zurich, which aim to develop a static analyzer and a program verifier for Python programs.

We present a new approach for tackling the type inference problem, solely based Satisfiability Modulo Theories (SMT) solving. We also augment the SMT solver with several enhancements to account for the dynamic nature for Python. We will go through the approach details and the SMT encoding in later chapters.

This thesis describes the design and the implementation of the static type inference for Python 3, using Z3 SMT solver. It is divided into seven chapters. The second chapter presents the background information that will help the reader comprehend the rest of the thesis. It reviews the already existing type inference algorithms and the past work done in this area, explains the syntax and the type system rules of the subset of Python 3 that our tool supports and explains the SMT concepts that we will be using throughout the thesis.

In the third chapter, we introduce the encoding of the type system that we support in the SMT solver. We also state and justify the limitations of this type system.

In the forth chapter, we describe the design and the implementation of the type inference algorithm in depth. We explain the components of the tool and all the SMT axioms for all the language constructs that we support.

The fifth chapter explains the experiments we have done to test the tool. We also highlight the current limitations of the type inference and problems it faces with certain types of programs.

Finally in the sixth chapter, we review our work and suggest more improvements in the future.

2 Background Information

2.1 Related Work

Many attempts have been made to infer types for Python, each of which had its own goals and limitations. We discuss here some work that we have studied, and we present some of their limitations and how similar and/or different they are from our tool.

2.1.1 Type Inference Algorithms

There are two type inference algorithms primarily used at the time of writing this thesis: Hindley-Milner algorithm and the Cartesian Product algorithm.

Hindley-Milner

bla

2.1.2 Mypy [2]

Mypy is a static type checker for Python. It depends on defining type annotations for almost all the constructs in the Python program to be checked. In addition, it performs local type inference. However, this type inference cannot be extended beyond local scopes. It requires that function definitions to be fully type-annotated and cannot infer function calls whose return type annotation is not specified. For example, mypy will fail to infer the type of variable x in the following program:

```
def f():
return "string"

x = f() # Infer type Any for x
```

What mypy intends to provide is closely related to the goal of our tool, that is to provide static type checking for the program. However, we aim to reduce (and sometimes eliminate) the writing overhead in defining the type annotations for the program constructs.

2.1.3 Inferência de tipos em Python [1]

The thesis [1] describes a static type system defined for a restricted version of RPython, and presents static type inference ideas based on this type system. The work presented in [1] also describes type inference implementation for Python expressions (like numbers, lists, dictionaries, binary and unary operations, etc.), assignment statements and conditional statements. It also gives an idea about inferring polymorphic and non-polymorphic function calls, class definitions and class instantiation. However, the approach they take has a handful of limitations and is not applicable to real Python code. It does not describe inferring function arguments, which is a critical step in the inference of function definitions and function calls. Accordingly, and similar to mypy, the inference they present is not extensible beyond local scopes inference.

3 Type System

4 Type Inference

5 Evaluation

6 Future Work

7 Conclusion

List of Figures

List of Tables

Bibliography

[1] Eva Catarina Gomes Maia. “Inferência de tipos em Python.” MA thesis. the Netherlands: Universidade do Porto, 2010.

[2] *mypy - Optional Static Typing for Python*. http://mypy-lang.org.