Type Inference For Python Programs

Andrei Nacu
”Alexandru Ioan Cuza” University of Iasi, Faculty of Computer Science
andreinaku@gmail.com

Abstract

My PhD research theme consists in translating Python code to C++ code. One of the first important obstacles encountered is the type of variables/expressions, because Python is dynamically typed, while C++ is not. Therefore, a basis for successful translation consists of correctly computing the possible types that appear in a Python program provided as input. We intend to use static program analysis techniques to infer types and provide useful information to be used in translating variable declaration and various statements. This paper describes the current status of our work, alongside examples that highlight some of the challenges encountered and the solutions proposed to address them.

Keywords: Python; C++; Inference; Cross-compilation
Domain: computer science
Section: Elaboration of the doctoral thesis

Motivation

Traditional cross-compilers seek to improve overall performance for a specific programming language. Facebook made significant efforts to compile PHP to C++ in order to improve their website (which has its fair share of visitors). Bitdefender also had a project at work that needed to be deployed on a NETGEAR router, where it had a maximum of 15MB of storage reserved. Even with enough optimizations under Python, it was still not small enough. After translating functionality from Python dll modules fully into C++, its size was reduced with about 87\% (down to 2MB). This required a Python developer to learn C++ enough to translate the code's functionality and check it afterwards.
Projects like Nuitka try to outdo themselves when it comes to speed. In this project, our focus is on FAITHFUL compilation. Faithful, in this case, means as close as possible to the original source code, syntactically and semantically equivalent.
While not expecting to get the same performance as a compiler that uses every trick of the destination language to improve speed, compiling Python to C++ should yield a faster-working code. Apart from functionality, efforts will be made to keep the output code to a form that is as similar as possible to the original source. Why? First and foremost, for maintenance purposes. A high-level Python programmer will not be able to understand every nook and cranny of an overly-optimized C++ code. However, reading and understanding a code that is similar in form to something familiar will make less of a chore out of maintaining.

Methodology of Research

We are currently focusing on type inference. We started with working with inference for simple, non-recursive Python code and try to formalise a type inference framework.
Program analysis offers static compile-time techniques for predicting behaviours that arise dynamically at run-time when executing a program [1]. We intend to use the well-known form of program analysis that is dataflow analysis. At the end of the analysis we should be provided with information about the possible types of program variables in every point of the program.

A complete formalization of the framework is still underway.

Results and Comparison with State-of-the-art

For now, based on the Python Library Reference [2], we managed to extract information about possible variable types without running the code. We used an external project, staticfg, to extract the control flow graph from the code. We tested our results with similar projects, such as:

- Nuitka: a very complex Python compiler with intermediary C code. It does not have the goal of making the intermediary C code similar and type inference is future work.
- Py14: project that is no longer being developed that uses the C++ auto keyword for type inference. The output code is similar but there is no real type inference other than the one auto has.
- Shedskin: an experimental compiler written for Python 2.4-2.6 designed to speed up resource-hungry Python programs. It is no longer being developed and variables are required to have a single static type throughout the program.
- Pytype: a mature and actively maintained project that is also used by Google for some of their scripts. It generates type annotations that follow the PEP484 guidelines. It does not infer types for variables in every program point.

Conclusions

We found that refined type inference is necessary for a correct translation of Python programs to C++ programs and that a refined dataflow analysis framework will be well-suited for our case. The current status of the work is that we have intra-procedural type inference for a kernel of types and we are working on refining type inference for heterogenous containers and inter-procedural type inference. Among other future challenges, we will have to take care of correctly inferring the types within recursive bits of code and taking type annotation into account, if it is already available within the Python code.

Acknowledgements

I would like to thank my PhD coordinator, Mr. Dorel Lucanu, for all his hard work both in teaching and putting up with me and my dynamic schedule.
I would also like to thank my wife and daughter for supporting my endeavour and my parents for making my education a top priority.

References

[1] F. Nielson, H. R. Nielson and C. Hankin, Principles of Program Analysis, Springer, 2005.

[2] "The Python Standard Library," [Online]. Available: https://docs.python.org/3/library/. [Accessed 14 10 2020].