

# Proposal: Gradual Typing for Octave Language\*

University of British Columbia CPSC 311 Course Project

Ada Li

University of British Columbia  
adali@alumni.ubc.ca

Kathy Wang

University of British Columbia  
kathy.wang@alumni.ubc.ca

Yuchong Pan

University of British Columbia  
yuchong.pan@alumni.ubc.ca

Paul Wang

University of British Columbia  
paul.wang@alumni.ubc.ca

## Abstract

**Keywords** gradual typing, Octave

## 1 Introduction

Static and dynamic type systems have their respective advantages. Static typing allows early error detection and enforces code style in a collaborative setting. It is, however, acknowledged that dynamic languages are better for fast prototyping. Over the past several decades, researchers in the programming language community have been working on integrating static typing and dynamic typing in a single language, having programmers control the level of type annotations. Gradual typing is a solution to combine the two type systems, proposed by [2]. It is of increasing interest in the programming language community and has been adopted by many programming languages, both in the industry and in the academia, such as Typed Racket [3], TypeScript [1] and Reticulated Python [4].

Octave is a scientific programming language with a dynamic type system. Because it has powerful matrix operations and is compatible with MATLAB scripts, it is widely used in statistics, mathematics and computer science communities for idea validation and fast prototyping. In this project, we propose a gradual type system for the Octave language to allow Octave programmers to annotate source code with optional type annotations, making Octave programs more robust and more suitable for production environments.

## Acknowledgments

The authors would like to thank the course staff of CPSC 311 at University of British Columbia for their feedback.

## References

- [1] Gavin Bierman, Martin Abadi, and Mads Torgersen. 2014. Understanding typescript. In *European Conference on Object-Oriented Programming*. Springer, 257–281.
- [2] Jeremy G Siek and Walid Taha. 2006. Gradual typing for functional languages. In *Scheme and Functional Programming Workshop*, Vol. 6. 81–92.
- [3] Sam Tobin-Hochstadt and Matthias Felleisen. 2008. The design and implementation of typed scheme. *ACM SIGPLAN Notices* 43, 1 (2008), 395–406.
- [4] Michael M Vitousek, Andrew M Kent, Jeremy G Siek, and Jim Baker. 2014. Design and evaluation of gradual typing for Python. In *ACM SIGPLAN Notices*, Vol. 50. ACM, 45–56.

---

\*Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).