

# 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 [7]. 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 [9], TypeScript [2] and Reticulated Python [10].

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.

### 2 Future Work

Because of the time limitation of the course, this project will be mainly focusing on the static semantics of the proposed gradual type system for Octave. In the future, however, more work can be done to extend the proposed gradual type system. For instance, dynamic type checking could be added during the interpretation, properly tracking errors from source

code. This can be done by translation into an internal cast calculus with the blame tracking [8]. In addition to that, type inference could be added to automatically deduce types at compile time. [5] introduces an approach to gradual type inference.

### 3 Related Work

There have been many research publications and projects on adding gradual type systems to an existing dynamic language. Diamondback Ruby (DRuby) is an extension to the Ruby language that combines static and dynamic typing in Ruby with constraint-based type inference [4]. Typed Racket [9] recently began to support refinement and dependent function types as experimental features [3]. Other similar projects include Typed Lua [6] and Gradualtalk [1] that brings an optional type system to Lua and to Smalltalk, respectively. Our project would largely benefit from these projects.

### Acknowledgments

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

### References

- [1] Esteban Allende, Oscar Callaú, Johan Fabry, Éric Tanter, and Marcus Denker. 2014. Gradual typing for Smalltalk. *Science of Computer Programming* 96 (2014), 52–69.
- [2] Gavin Bierman, Martín Abadi, and Mads Torgersen. 2014. Understanding typescript. In *European Conference on Object-Oriented Programming*. Springer, 257–281.
- [3] Racket Blog. 2017. Refinement Types in Typed Racket. <http://blog.racket-lang.org/2017/11/adding-refinement-types.html>
- [4] Michael Furr. 2009. *Combining static and dynamic typing in Ruby*. Ph.D. Dissertation.
- [5] Ronald Garcia and Matteo Cimini. 2015. Principal type schemes for gradual programs. In *ACM SIGPLAN Notices*, Vol. 50. ACM, 303–315.
- [6] André Murbach Maidl, Fabio Mascarenhas, and Roberto Ierusalimsky. 2014. Typed Lua: An optional type system for Lua. In *Proceedings of the Workshop on Dynamic Languages and Applications*. ACM, 1–10.
- [7] Jeremy G Siek and Walid Taha. 2006. Gradual typing for functional languages. In *Scheme and Functional Programming Workshop*, Vol. 6. 81–92.
- [8] Jeremy G Siek, Michael M Vitousek, Matteo Cimini, and John Tang Boyland. 2015. Refined criteria for gradual typing. In *LIPICs-Leibniz*

\*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).

*International Proceedings in Informatics*, Vol. 32. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

- [9] Sam Tobin-Hochstadt and Matthias Felleisen. 2006. Interlanguage migration: from scripts to programs. In *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*. ACM, 964–974.
- [10] 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.