Proposal: Gradual Typing for Octave Language*†

University of British Columbia CPSC 311 Course Project

Ada Li
University of British Columbia
adali@alumni.ubc.ca

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

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

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

Abstract

In this proposal, we outline the general milestones and approaches that will be taken for our project of introducing gradual typing to Octave and provide brief motivation for the subject area. Further, we argue that the project is low-risk and interesting as a term project by showing relevant pieces of supportive literature and open-sourced projects and contend that there is much potential in future extensions.

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 [11]. 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 [13], TypeScript [2] and Reticulated Python [15].

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.

†Revised on November 9, 2018.

2 Overview and Plans

In this project, we will add a gradual type system to the Octave language. Because of the time limitation of the course, we will be focusing on the static semantics. Future work is discussed in Section 3. This means that our proposed gradual type system will mainly be used for static analysis and IDE tooling (e.g., code completion and refactoring, etc.), but not for runtime type checking. Therefore, this project is similar to type hints in Python [1] and the type system of TypeScript.

2.1 How to Fulfill Background Research Report Milestone

We will be using the resources listed in Section 5 as starting points and guidelines for us to work from as we explore the articles that focuses on gradual typing. We will cover its value in everyday programming practice as well as cover the technical elements.

2.2 How to Fulfill Proof-of-Concept Milestone

For the proof-of-concept milestone, we will be focusing on implementing a static type checker for the gradually-typed lambda calculus (GTLC) by following [10]. The gradually-typed lambda calculus can be regarded a core of gradually typed programming languages, and implementing a static type checker for it will help us gain deeper understanding of gradual type systems. The following grammar defines the syntax of the gradually-typed lambda calculus [10].

variables x integers n blame labels l

basic types	B	::=	`int` | `bool`
types	T	::=	$B \mid (\rightarrow T\ T) \mid ?$
constants	k	::=	n | `#t` | `#f`
operators	op	::=	`inc` | `dec` | `zero?`
expressions	e	::=	$k \mid (op\ e\ l) \mid (\texttt{if}\ e\ e\ e\ l) \mid x \mid (e\ e\ l) \mid (\lambda(x)\ e) \mid (\lambda(x : T)\ e) \mid (e : T\ l)$

Here, the dynamic type is denoted by `?`. The lambda expression $(\lambda(x)\ e)$ is a syntactic sugar of $(\lambda(x : ?)\ e)$. The expression $(e : T\ l)$ is an explicit cast.

[1] New in version 3.5.

In addition, we are going to build a parser that converts Octave source code to abstract syntax trees in Racket, using parsing tools available in Racket. For practical reasons, we plan to implement the most significant subset of the Octave language to demonstrate Octave-specific features.

2.3 How to Fulfill Final Project Milestone

For the final project milestone, we would like to apply what we will have learned in the gradually-typed lambda calculus to the Octave language, implementing a static type checker for the proposed gradual counterpart of the Octave language. Additionally, we intend to enrich our proposed gradual type system with more domain-specific features, such as static dimension checking for matrix operations, as matrices are one of the most significant component of the Octave language. We plan to survey and experiment various approaches to enforce type checking for these language-specific features, for which we have seen several design choices.

2.4 How to Fulfill Poster Milestone

Our poster will emphasize the usefulness it has for users when dealing with matrices, which will entice our peers to learn more about the project as well as possibly integrating it. This is particularly useful for students involved in mathematics and statistic courses as they will be using software like Octave frequently. We will be including diagrams on how this project works so students can easily understand without having to read a large amount of text. We will try to include an easy way for students to learn more about this project topic. Some ideas include compiling a list of topics on a single webpage and including that on our poster. Another idea would be creating a QR code for students to scan and receive more information.

2.5 How This Can Be a Low-Risk Approach

This project has a low risk because researchers in the gradual typing community have been exploring designs of gradual type systems for many years. There have been many publications in this field and other related fields, among which we have compiled a list of resources that may be of particular importance for our project, listed in Section 5. For the language choice, Octave is an open source language compatible with MATLAB. This ensures the audience of our project. In addition, there are many open source projects related to MATLAB/Octave parsing, which will help us parse Octave source code.

3 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 [12]. In addition to that, type inference could be added to automatically deduce types at compile time. [5] introduces an approach to gradual type inference.

4 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 [13] recently began to support refinement and dependent function types as experimental features [3]. Other similar projects include Typed Lua [8] and Gradualtalk [1] which bring optional type systems to Lua and to Smalltalk, respectively. Our project would largely benefit from these projects.

5 Supporting Resources

We have compiled a list of resources that may be of particular importance for our project, listed as follows.

- Siek and Garcia [10]: This will be one of our main supporting documents. This paper serves as a tutorial on how to type-check and interpret the gradually-typed lambda calculus. We will use this as a guide when we implement our proposed gradual typing system.
- Siek et al. [12]: This paper provides a formal characterization of behaviors of gradual type systems. It also surveys popular programming languages with gradual type systems. There are also very useful examples in the paper which we can reference.
- Furr [4]: This project adds type annotations, type inference and dynamic type checking to Ruby. We can use this project as a reference when we design the proposed gradual type system for Octave.
- Tobin-Hochstadt et al. [14]: This link contains a list of publications in the field of gradual typing. In case the links above do not provide enough information for our project, we will still have a large pool of resources to look through.
- Yakimovic [16] and Harley [6]: These two projects are both MATLAB parsers that we can reference. Since the syntax of Octave is very similar to MATLAB, we can get valuable insight from these parsers.
- Hendren [7]: This paper explains how they added typing aspects to MATLAB. They did this by adding the "atype" statement, which is a form of type annotation used to specify runtime types of variables, especially for function inputs and outputs. Since MATLAB and

Octave are mostly compatible, we conclude that Octave can also benefit from type annotations, which is a part of what we will be introducing in this project.

- Olmos and Visser [9]: This paper illustrates how Octave can benefit from introducing static type checking and shape analysis into a dynamically typed language. Specifically, static type and shape inferencing can improve the efficiency of the generated code. Functions and operators are highly overloaded in Octave, which can lead to bottlenecks; thus, the consequence/tradeoff for this flexibility is computational performance. The reason why is because the run-time system is responsible for type checking, array shape determination, function call dispatching, and handling possible run-time errors. In order to improve efficiency, these issues should be addressed at compile-time.

6 Summary

The objective for this project is to develop a gradually-typed variant for Octave that sufficiently expresses the basic data types of the language. We believe that the flexible nature of gradual typing offers great value to data scientists and various users of Octave in that it remains an effective prototyping tool while optionally providing compile-time type and invariant assertions. To achieve this, we intend to base our initial prototypes for static semantics on the existing literature described above. Additionally – and as time permits – we intend to enrich and further introduce nuances into our gradual typing scheme by delving into features such as static dimension checking (as matrices are a core component of the language domain) and other more domain-specific aspects. We hope to create a meaningful and useful tool for Octave developers to use and ideally contribute towards in the future.

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] Eric Harley. 2013. https://github.com/ericharley/matlab-parser.

[7] Laurie Hendren. 2011. Typing aspects for MATLAB. In *Proceedings of the sixth annual workshop on Domain-specific aspect languages*. ACM, 13–18.

[8] André Murbach Maidl, Fabio Mascarenhas, and Roberto Ierusalimschy. 2014. Typed Lua: An optional type system for Lua. In *Proceedings of the Workshop on Dynamic Languages and Applications*. ACM, 1–10.

[9] Karina Olmos and Eelco Visser. 2003. Turning dynamic typing into static typing by program specialization in a compiler front-end for Octave. In *Source Code Analysis and Manipulation, 2003. Proceedings. Third IEEE International Workshop on*. IEEE, 141–150.

[10] Jeremy G Siek and Ronald Garcia. 2012. Interpretations of the gradually-typed lambda calculus. In *Proceedings of the 2012 Annual Workshop on Scheme and Functional Programming*. ACM, 68–80.

[11] Jeremy G Siek and Walid Taha. 2006. Gradual typing for functional languages. In *Scheme and Functional Programming Workshop*, Vol. 6. 81–92.

[12] Jeremy G Siek, Michael M Vitousek, Matteo Cimini, and John Tang Boyland. 2015. Refined criteria for gradual typing. In *LIPIcs-Leibniz International Proceedings in Informatics*, Vol. 32. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[13] 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.

[14] Sam Tobin-Hochstadt, Jeremy G. Siek, Asumu Takikawa, Ben Greenman, Jason Yeo, Deyaaeldeen Almahallawi, Jack W, Éric Tanter, Ambrose Bonnaire-Sergeant, Andrew Kent, Srinivas Thatiparthy, and Matthias Felleisen. 2013. A bibliography on Gradual Typing. https://github.com/samth/gradual-typing-bib.

[15] 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.

[16] Yauhen Yakimovic. 2013. https://github.com/ewiger/decade.