Placeholder

Isaac H. Lopez Diaz

November 6, 2025

1 Introduction

This paper presents the design of a reflective programming language (PL) that reasons about its dataflow semantics. A PL is said to be reflective when is able to reason about itself.[5] It can be thought of as the process of converting data into a program. The inverse of this process, reification, can be thought of turning a program into data.[3] These two processes allow a programmer to see the contents of the current execution, much like debugging. However, unlike debugging, one can change the semantics of the language on-the-fly.[2]

The goal of the language is to better understand dataflow semantics. One such application would be on machine learning (ML) programs, since ML programs rely on dataflow graph execution models.[1] The conversion from imperative to graph execution has proven to be challenging for programmers, primarily looking to optimize their code, leading to bugs or perfomance issues (the opposite of what the programmer intended to do).[6]

The argument is that by having a language to have programmable semantics that allows programmers to specify the semantics (dataflow, procedural, imperative, etc.) leads to less bugs.[4]

2 Reflective languages

Reflective languages started off with the notion of an infinite tower of interpreters. This means that you'd have an interpreter interpreting an interpreter, and so on. In order for a language to be reflective, it must have two properties: (1) the ability to reify its own interpreter, and (2) the ability to reflect on the reified interpreter.[3] This gives the ability to extend the language syntax and semantics. Think of macros at runtime.

3 Graph Execution and ML

4 Future Work

The idea is to alter the store of the interpreter (the graph that is building)

References

[1] Martín Abadi, Michael Isard, and Derek G Murray. A computational model for tensorflow: an introduction. In *Proceedings of the 1st ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pages 1–7, 2017.

[2] Kenichi Asai, Satoshi Matsuoka, and Akinori Yonezawa. Duplication and partial evaluation: For a better understanding of reflective languages. *Lisp and Symbolic Computation*, 9(2):203–241, 1996.

[3] Daniel P Friedman and Mitchell Wand. Reification: Reflection without metaphysics. In *Proceedings of the 1984 ACM Symposium on LISP and functional programming*, pages 348–355, 1984.

[4] Mike Innes, Stefan Karpinski, Viral Shah, David Barber, PLEPS Saito Stenetorp, Tim Besard, James Bradbury, Valentin Churavy, Simon Danisch, Alan Edelman, et al. On machine learning and programming languages. Association for Computing Machinery (ACM), 2018.

[5] Brian Cantwell Smith. Reflection and semantics in lisp. In *Proceedings of the 11th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 23–35, 1984.

[6] Tatiana Castro Vélez, Raffi Khatchadourian, Mehdi Bagherzadeh, and Anita Raja. Challenges in migrating imperative deep learning programs to graph execution: an empirical study. In *Proceedings of the 19th International Conference on Mining Software Repositories*, MSR '22, page 469–481, New York, NY, USA, 2022. Association for Computing Machinery.