A reflective language for the analysis of dataflow semantics

Isaac Hiram Lopez Diaz

November 13, 2025

Abstract

This paper presents the design and implementation of a reflective programming language able to reason about its dataflow semantics.

1 Introduction

A language (or any system, for that matter) is reflective when it is able to reason about itself.[2] The result from this is that one would have a tower of interpreters, each interpreting the one "above it", meaning, an interpreter interpreting an interpreter interpreting an interpreter, and so on. This would give the power of some interpreter to "reach down" the tower and change the semantics of the interpreter interpreting it and extend the language. Friedman and Wand simplify this notion by introducing two processes: (1) *reification*, the ability to transform code into data, and (2) *reflection*, the ability to transform data into code. [1] Now instead of a tower of interpreters one can have two interpreters

References

[1] 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.

[2] 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.