ECE491 Advanced compilers final report
Implementing a lazy functional language

Jonathan Lam

2022/05/12

Contents

	Purely functional	Non-purely functional
Untyped	Core	Scheme[1]
Typed	Hazel[2]	C[3]

Table 1: Summary of programming language implementation projects

1 Project overview

1.1 Motivation and overview

This project was the semester-long project for the ECE491 independent study on advanced compilers. It is the implementation of a simple lazy (normal-order) evaluation functional language similar to Haskell. The work follows the tutorial, “Implementing functional languages: a tutorial” by Simon Peyton Jones at Microsoft Research [2]. This was published two years after Haskell 1.0 was defined in 1990 [1], to which SPJ was a major contributor.

This project is the culmination of my studies in programming languages and functional programming over the past two years. These studies include writing a C compiler in C (Spring 2021), a LISP interpreter in LISP (Summer 2021), and this project, a Core interpreter and compiler in Haskell (Spring 2022). Additionally, my Master’s thesis (2021-2022 school year) was about updating the evaluation model of Hazel, a programming language implemented in OCaml. We can summarize these languages in Table 1.

1.2 Commentary on the tutorial

1.3 Implementation setup

TODO: github repo (all three, including this one)

2 Background

2.1 Definition of a programming language

TODO: syntax, semantics

TODO: notation used for dynamic semantics here: state machine?

2.2 Implementations of programming language

TODO: compilers, interpreters

2.3 The untyped λ-calculus

2.4 Functional programming

TODO: haskell and miranda

3 The Core language

3.1 Terminology

TODO: lazy evaluation (different terms), supercombinators, etc.

3.2 Syntax

3.3 Dynamic semantics

3.4 Sample programs

3.5 Data structures

3.6 Lexer

3.7 Parser

3.8 Pretty-print utility

4 Template instantiation (TI) evaluator

4.1 The TI abstract machine

4.2 Sample evaluations

5 G-Machine (GM) compiler

5.1 The GM abstract machine

5.1.1 List of opcodes

5.2 Compilation schemes

5.3 Evaluator

5.4 Sample compilations and evaluations

6 Future work

TODO: garbage collection

TODO: last marks of TI and GM

TODO: 3-address machine and parallel g-machine

TODO: lambda lifting

TODO: study implementation of STG

7 Conclusions

8 References

[1] Paul Hudak et al. “A history of Haskell: being lazy with class”. In: *Proceedings of the third ACM SIGPLAN conference on History of programming languages*. 2007, pp. 12–1.

[2] Simon L Peyton Jones and David R Lester. “Implementing functional languages: a tutorial”. In: *Department of Computer Science, University of Glasgow* (2000).