Studying link-time optimizations in programming language development to facilitate the continuum of static and dynamic modules

David Callanan

21444104

Final Year Project – 2026

B.Sc. Single Honours in Computational Thinking

Department of Computer Science

Maynooth University

Maynooth, Co. Kildare

Ireland

A thesis submitted in partial fulfilment of the requirements for the

B.Sc. Single Honours in Computational Thinking.

Supervisor: Dr. Phil Maguire

Contents

Declaration

I hereby certify that this material, which I now submit for assessment on the program of study as part of the B.Sc. Computational Thinking qualification, is entirely my own work and has not been taken from the work of others - save and to the extent that such work has been cited and acknowledged within the text of my work.

I hereby acknowledge and accept that this thesis may be distributed to future final year students, as an example of the standard expected of final year projects.

Signed: ______________________ Date: ____________

Acknowledgements

I would like to thank my parents/guardians/siblings/classmate/friends for their . . .

Abstract

The logic of computer software is primarily expressed using programming languages, but the balance between versatile and efficient programming languages remains a major challenge. Changes to the execution environment of a program can impact performance, or worse, cause incompatibilities that require major refactoring. Developers often maintain multiple branches of a codebase to keep performance to acceptable levels. One particular difference between execution environments is whether individual modules are statically linked at compile-time or dynamically resolved at runtime. In this research project, we propose a prototype programming language called “Essence C” that better supports the continuum of static and dynamic modules, by allowing the programmer to write code once that is agnostic to the linking strategy. By taking advantage of LLVM’s intermediate representation, it is possible to advance the state-of-the-art of link-time optimizations across module boundaries.

Chapter 1 – Introduction

1.1 Overview

1.2 Motivation

1.3 Methodology

Thesis structure, and a resesarch strategy.

To address the objectives of this research, a design-science methodology is employed with a focus on continued interation and evaluation of “Essence C”. This approach was selected because the project is concerned with the implementation of a novel programming language that advances the flexibility and performance of modules in software development, and thus requires a design and implementation process that is informed by ongoing evaluation.

The structure of this thesis is organized to reflect this methodology. Chapter 2 reviews existing literature.

1.4 Success Criteria

Success will be evidenced by the following deliverables:

(1) A functional compiler that implements the designed module system;

(2) Documentation of the module system and other core language features;

(3) A simple IDE extension that provides syntax highlighting to the programmer;

(4) A body of research into the continuum of static and dynamic modules in software development, with kernel development as a case study;

(5) An implementation of link-time optimization techniques that leverage the designed module system;

(6) An evaluation of the effectiveness of these optimizations through the benchmarking of sample programs;

(7) A monorepo containing extensive git history, demonstrating ongoing development progress;

(8) A final written report detailing the research, design, implementation and evaluation of the project.

Chapter 2 – Technical Background

2.1 Literature Review

2.1.1 Continuum of static and dynamic modules

Static vs dynamic dispatch. Everything in between.

2.1.2 Module loading in operating system kernels

2.1.3 Link-time optimizations

2.1.4 Inlining and IR-level linking

2.2 Technical Material

2.2.1 LLVM intermediatea representation

Chapter 3 – The Problem

TODO

Chapter 4 – The Solution

4.1 Individual elements

4.2 Picking of overall solution

4.3 Design and implementation

Chapter 5 – Evaluation

TODO

Chapter 6 – Conclusion

TODO

Chapter 7 – Parsing Library

Prior to this project I had already developed a simple parsing library in JavaScript, which allows for:

(1) Parsing terminal rules using regular expressions.

(2) Forming non-terminal rules by combining other rules in the following ways, sufficient to develop any complex grammar:

   (1) “or” making rule optional

   (2) “join” sequencing rules

   (3) “multi” repeating rules zero or more times

   (4) “opt” making rule optional

(3) Map parsed data to custom structures using the “mapData” function.

In addition it was necessary to implement a trace system to debug issues with parse rules.