THE COOPER UNION FOR THE ADVANCEMENT OF SCIENCE AND ART
ALBERT NERKEN SCHOOL OF ENGINEERING

Implementation of fill-and-resume in the Hazel live programming environment

by

Jonathan Lam

A thesis submitted in partial fulfillment of the requirements for the degree of
Master of Engineering

Professor Fred L. Fontaine, Advisor
Professor Robert Marano, Co-advisor

THE COOPER UNION FOR THE ADVANCEMENT OF SCIENCE AND ART

ALBERT NERKEN SCHOOL OF ENGINEERING

This thesis was prepared under the direction of the Candidate's Thesis Advisor and has received approval. It was submitted to the Dean of the School of Engineering and the full Faculty, and was approved as partial fulfillment of the requirements for the degree of Master of Engineering.

Barry L. Shoop, Ph.D., P.E. Date

Fred L. Fontaine, Ph.D. Date

ACKNOWLEDGEMENTS

TODO

ABSTRACT

TODO

TABLE OF CONTENTS

LIST OF FIGURES

TABLE OF NOMENCLATURE

TODO

1 Introduction

1.1 Functional programming

1.1.1 Recursion and mathematical induction

1.1.2 The λ-calculus

1.1.3 Purity and statefulness

1.1.4 Comparison to other programming paradigms

1.2 Classifications of type systems

1.2.1 Primer on programming language semantics and reasoning

1.2.2 Static vs. dynamic typing

1.2.3 Strong vs. weak typing

1.2.4 Inferred vs. manifold typing

1.2.5 Gradual typing

1.2.6 Other classifications

1.3 Approaches to programming interfaces

1.3.1 Structure editors

1.3.2 Graphical editors

1.3.3 Intentional, generative, and meta-programming

1.3.4 Applications to programming education

1.3.5 Drawbacks of non-textual editors

2 An overview of the Hazel programming environment

2.1 Hazelnut static semantics

2.1.1 Expression and type holes

2.1.2 Bidirectional typing

2.1.3 Example of bidirectional type derivation

2.2 Hazelnut Live dynamic semantics

2.2.1 Example of elaboration

2.2.2 Example of evaluation

2.2.3 Example of hole instance numbering

2.3 Hazel programming environment

2.3.1 Explanation of interface

2.3.2 Implications of Hazel

3 Problem statement and related works

3.1 Problem statement

3.2 Issues with the current implementation

3.2.1 Hole numbering inefficiencies

3.2.2 Hole instance tracking inefficiencies

3.3 CMTT interpretation of fill-and-resume

3.4 Notebook-style live programming environments

4 Implementation and optimization of FAR

4.1 Evaluation with environments

4.2 Restructuring hole numbering

4.3 Implementing FAR

4.4 Memoization of recent actions

4.5 UI changes for notebook-like editing

5 Evaluation of FAR

6 Conclusions and recommendations

REFERENCES

APPENDICES

A Formalization of evaluation with environments

B Contributions to Hazel

B.1 Equivalent evaluation with environments

B.2 Identifying performance issue

B.3 Simplifying hole renumbering process

B.4 Implementation of FAR functionality

B.5 Memoization for FAR

B.6 Additional performance improvements

C Selected code samples