Practical performance enhancements to the evaluation model of the Hazel programming environment

Jonathan Lam[1] Prof. Fred Fontaine, Advisor[1]
Prof. Robert Marano, Co-advisor[1] Prof. Cyrus Omar[2]

[1]Electrical Engineering
The Cooper Union for the Advancement of Science and Art

[2]Electrical Engineering and Computer Science
Future of Programming Lab (FPLab), University of Michigan

2022/04/29

Overview I

Project context

Implementation-based
: Mostly practically-driven

Functional programming
: Context for PL theory

Hazel live programming environment
: An experimental editor with typed holes aimed at solving the “gap problem,” developed at UM

Overview II

Project scope

Evaluation with environments Lazy variable lookup for performance

Hole instances to hole closures Redefining hole instances for performance

Implementing fill-and-resume (FAR) Efficiently resume evaluation

Project evaluation

Empirical evaluation Measure performance gain of motivating cases

Informal metatheory State metatheorems and provide proof sketches

Table of Contents

A programming language is a specification

Syntax is the grammar of a valid program

Semantics describes the behavior of a syntactically valid program

$$\tau ::= \tau \to \tau \mid b \mid (\!|\,|\!)$$

$$e ::= c \mid x \mid \lambda x : \tau . e \mid e\ e \mid e : \tau \mid (\!|\,|\!) \mid (\!|e|\!)$$

Figure: Hazelnut grammar

Static and dynamic semantics

Statics Edit actions, type-checking, elaboration ("compile-time")

Dynamics Evaluation ("run-time")

$$\frac{e_1 \Downarrow \lambda x.e_1' \qquad e_2 \Downarrow e_2' \qquad [e_2'/x]e_1' = e}{e_1\ e_2 \Downarrow e}\ \text{EAp}$$

Figure: Evaluation rule for function application using a big-step semantics

A brief primer on the λ-calculus

Untyped λ-calculus Simple universal model of computation by Church

Simply-typed λ-calculus Extension of the ULC with static type-checking

Gradually-typed λ-calculus Optionally-typed, with "pay-as-you-go" benefits of static typing

$$\begin{aligned} e ::= {} & x \\ \mid {} & \lambda x.e \\ \mid {} & e\ e \end{aligned}$$

(a) Grammar

$$\frac{}{\lambda x.e \Downarrow \lambda x.e}\ \Lambda\text{-ELam}$$

$$\frac{e_1 \Downarrow \lambda x.e_1' \qquad [e_2/x]e_1' \Downarrow e}{e_1\ e_2 \Downarrow e}\ \Lambda\text{-EAp}$$

(b) Dynamic semantics

Figure: The untyped λ-calculus

Table of Contents

The Hazel programming language and environment

Live programming Rapid static and dynamic feedback ("gap problem")

Structured editor Elimination of syntax errors

Bidirectionally typed Simple type inference

Gradually typed Hole type and cast-calculus based on Siek et al. [1, 2]

Purely functional Avoids side-effects and promotes commutativity

(a) The Hazelgrove organization

(b) Implemented in ReasonML and JSOO

Figure: Hazel implementation

The Hazel programming interface

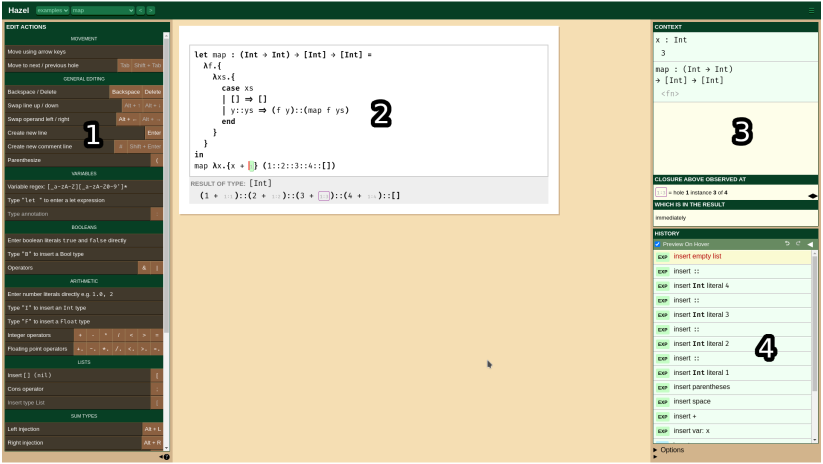

Figure: The Hazel interface

Hazelnut: A bidirectionally-typed static semantics

Hazelnut Live: A bidirectionally-typed dynamic semantics

Table of Contents

Evaluation using environments vs. substitution

Updated evaluation rules

Handling recursion

Matching the result from evaluation using substitution

Memoizing by environments for substitution and equality checking

Generalized closures

Table of Contents

Motivating example

Hole instances vs. hole closures/instantiations

Hole instance parent vs. hole closure parents

The hole numbering algorithm

A unified postprocessing algorithm

Table of Contents

Motivating example

The FAR process

1-step vs. n-step FAR

Detecting a valid fill operation

The fill operation

The resume operation

The postprocessing operation

Table of Contents

Evaluation with environments

Hole numbering motivating example

FAR motivating example

Table of Contents

Generalized closures

Unique hole closures

FAR as a generalization of evaluation

Table of Contents

Future work

n-step FAR: Integrate edit history into FAR

Generalized memoization: Unify notation and metatheory of memoization

Formal evaluation of metatheory: Check coverage and correctness of metatheorems using Agda

Conclusions

References I

Jeremy G. Siek and Walid Taha.
Gradual typing for functional languages.
In *IN SCHEME AND FUNCTIONAL PROGRAMMING WORKSHOP*, pages 81–92, 2006.

Jeremy G Siek, Michael M Vitousek, Matteo Cimini, and John Tang Boyland.
Refined criteria for gradual typing.
In *1st Summit on Advances in Programming Languages (SNAPL 2015)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

Cyrus Omar, Ian Voysey, Michael Hilton, Jonathan Aldrich, and Matthew A. Hammer.
Hazelnut: A Bidirectionally Typed Structure Editor Calculus.
In *44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL 2017)*, 2017.

References II

Cyrus Omar, Ian Voysey, Ravi Chugh, and Matthew A. Hammer.
Live functional programming with typed holes.
PACMPL, 3(POPL), 2019.