Structured Editing for Elm* in Elm

Ravi Chugh, Nick Collins, Brian Hempel, Grace Lu, Justin Lubin, Mikaël Mayer
I ♥ FP

OCaml

Scala
type LogoParams
  = TopLeft {x: Number, y: Number, size: Number}
  | Center {cx: Number, cy: Number, rad: Number}

logo : String -> LogoParams -> Svg
logo fill logoParams =
  let
  {x, y, size} = 
    case logoParams of 
      TopLeft data ->
        data
      Center {cx, cy, rad} ->
        {x=x-cx-rad, y=cy-rad, size=2*rad}
  in
  [ rect fill x y size size
    , line "white" 10 x y (x + size) (y + size)
    , line "white" 10 x (y + size) cx cy
  ]

main =
  svg <| logo "gray" <| TopLeft {x=100, y=130, size=200}
Sketch-n-Sketch

Direct Code Manipulation

Direct Output Manipulation

Strange Loop 2016

[ICSE 2018]

[work-in-progress]

[PLDI 2016]

[UIST 2016]

[OOPSLA 2018]
Sketch-n-Sketch

Direct Code Manipulation

- [ICSE 2018]
- [work-in-progress]

Direct Output Manipulation

- Strange Loop 2016
- [PLDI 2016]
- [UIST 2016]
- [OOPSLA 2018]
Sketch-n-Sketch

Direct Code Manipulation

Show AST on Code

[ICSE 2018]
[work-in-progress]

Demo: "Deuce"
Code Editor
(func arg)
Show AST on Code

(func arg)
Show AST on Code

(func arg)
Show AST on Code

```
func arg
```
n = 2018
m =
  if n < 0 then
  "negative"
  else
  n

-- TYPE MISMATCH -------------- tmp.elm
The branches of this `if` produce different types of values.

The `then` branch has type:

  String

But the `else` branch is:

  number

Hint: These need to match so that no matter which branch we take, we always get back the same type of value.
"Live and Direct" Type Inspector
Make This State Impossible! ...
Holes for Incomplete Code

HAZEL: A LIVE FUNCTIONAL PROGRAMMING ENVIRONMENT WITH TYPED HOLES

When programming, we spend a substantial amount of our time working on programs not yet complete. For example, there can be blank spots, type errors or merge conflicts at various locations in the program. Conventional programming language definitions of formal meaning to these incomplete programs, so development tools can at best only use ad hoc heuristics to provide various useful language services without interruption, including completion, type inspection, code navigation, and live debugging.

We are working to develop a more principled approach to working with incomplete programs, rooted in the first principles of type theory. We model incomplete programs as programs with holes, which (1) stand for parts of the program that are missing and (2) serve as membranes around parts of the program that are erroneous or conflicted.

We are incorporating this approach into Hazel (hazel.org), a web-based live programming environment for an Elm-like functional language designed from the ground up to support typed hole-driven development. Uniquely, Hazel inserts holes automatically to ensure that every editor state has both static meaning (so we can provide type-based services, like type inspection, without interruption) and dynamic meaning (so we can provide run-time services, like Hazel's novel live debugger, without interruption). We are currently using Hazel to develop interactive course material that introduces functional programming.

Cyrus Omar
UNIVERSITY OF CHICAGO

Cyrus Omar got his PhD in programming languages from Carnegie Mellon University in May 2017. He is now a post-doctoral researcher at The University of Chicago, where he leads the Hazel collaboration. In a past life, Cyrus was a computational neurobiologist. That experience motivated him to switch fields and focus on designing a better front-end programming experience for scientists, engineers, artists and other skilled end-users, rooted in the principles in type theory and interaction design.

http://hazel.org/

Cyrus Omar
Fri 1:30pm @ Strange Loop

Hazel
AST on Code
Live, Direct Types
Code Tools
Holes
Thanks!

Nick Collins
Brian Hempel
Grace Lu
Justin Lubin
Mikaël Mayer

Swiss National Science Foundation
Structured Editing for Elm* in Elm

http://ravichugh.github.io/sketch-n-sketch/

AST on Code
Live, Direct Types
Code Tools
Holes
EXTRA SLIDES
Related Work

Plain Text Editing

Hybrid Editors
- Barista
- Active Code Completion
- Greenfoot
- Code Bubbles
  Sketch-n-Sketch: Deuce

Structured Editing

Automated Refactoring

Alternative UIs for Refactoring
- Refactoring w/ Synthesis
- Drag-n-Drop Refactoring
  Sketch-n-Sketch: Deuce
Lightweight Structured Editing via Direct Manipulation

- Structural (Multi-)Selection;
- Context-Sensitive Tool Menu;
- Hover Preview and Confirm

Traditional UI

- Text Selection;
- Tool Menu;
- Configuration Wizard

Automated Refactoring
1. Learning Curve for New UI + Tools
2. Faster Tool Invocations using New UI
3. Overwhelming Preference for New UI

Structural (Multi-)Selection; Context-Sensitive Tool Menu; Hover Preview and Confirm

Traditional UI
Text Selection; Tool Menu; Configuration Wizard

21 users
The branches of this `if` produce different types of values.

31> if n < 0 then
32>    "negative"
33> else
34>    n

The `then` branch has type:

    String

But the `else` branch is:

    number

Hint: These need to match so that no matter which branch we take, we always get back the same type of value.