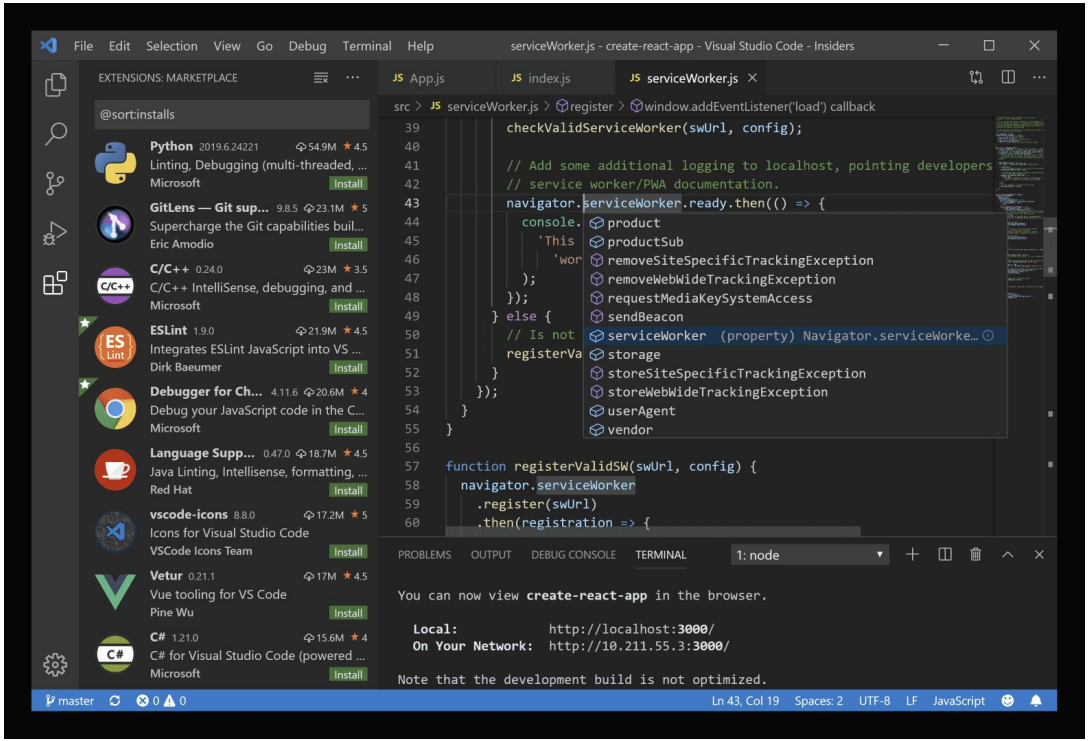
Totally Live Programming (and Proving?) with Hazel

Cyrus Omar , Andrew Blinn, and David Moon (*+ many Hazel contributors*)

Future of Programming Lab (FP Lab)
University of Michigan

Topos Institute
August 29, 2024

Live Language Services

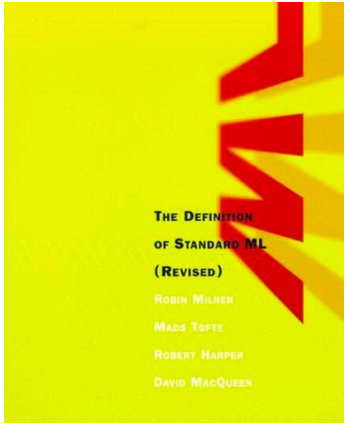


The Gap Problem

When the program is incomplete, there is a gap in service.

- Syntax errors `2 +`
- Type errors `2 + true`
- Run-time errors `2 / 0`
- Merge conflicts `<<<<<<< HEAD 2 + 2 = 4 ====== ...`

This is rooted in a definitional gap: language definitions don't assign structure and meaning to incomplete programs.



Solving the Gap Problem...

The Hazel structure editor ensures that there are no syntax errors by inserting holes.

Solving the Gap Problem...

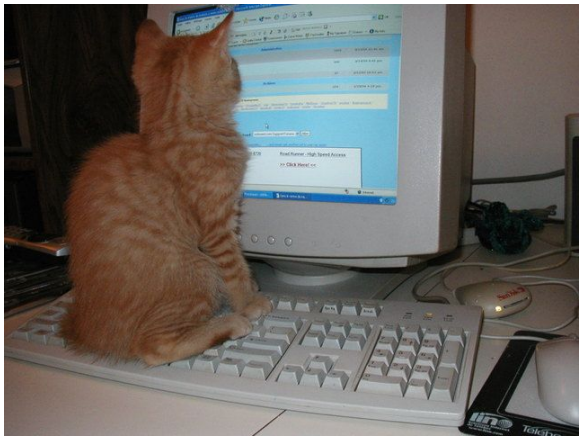
The Hazel **structure editor** ensures that there are no syntax errors by inserting holes. The **semantics understands holes**, so every editor state is therefore syntactically, statically, and dynamically well-defined (i.e. Hazel is totally live!)

[POPL 2024, OOPSLA 2023, POPL 2019, POPL 2017]

Solving the Gap Problem...

The Hazel structure editor ensures that there are no syntax errors by inserting holes. The semantics understands holes , so every editor state is therefore syntactically, statically, and dynamically well-defined (i.e. Hazel is totally live !)

[POPL 2024, OOPSLA 2023, POPL 2019, POPL 2017]



Solving the Gap Problem...

The Hazel **structure editor** ensures that there are no syntax errors by inserting holes. The **semantics understands holes**, so **every editor state** is therefore syntactically, statically, and dynamically well-defined (i.e. Hazel is **totally live**!)

[POPL 2024, OOPSLA 2023, POPL 2019, **POPL 2017**]

Theorem 1 (Action Sensibility).

1. *If* $\dot{\Gamma} \vdash \hat{e}^{\diamond} \Rightarrow \dot{\tau}$ *and* $\dot{\Gamma} \vdash \hat{e} \Rightarrow \dot{\tau} \xrightarrow{\alpha} \hat{e}' \Rightarrow \dot{\tau}'$ *then* $\dot{\Gamma} \vdash \hat{e}'^{\diamond} \Rightarrow \dot{\tau}'$.
2. *If* $\dot{\Gamma} \vdash \hat{e}^{\diamond} \Leftarrow \dot{\tau}$ *and* $\dot{\Gamma} \vdash \hat{e} \xrightarrow{\alpha} \hat{e}' \Leftarrow \dot{\tau}$ *then* $\dot{\Gamma} \vdash \hat{e}'^{\diamond} \Leftarrow \dot{\tau}$.

Solving the Gap Problem...

The Hazel structure editor ensures that there are no syntax errors by inserting holes. The semantics understands holes , so every editor state is therefore syntactically, statically, and dynamically well-defined (i.e. Hazel is totally live !)

[POPL 2024 , OOPSLA 2023, POPL 2019, POPL 2017]

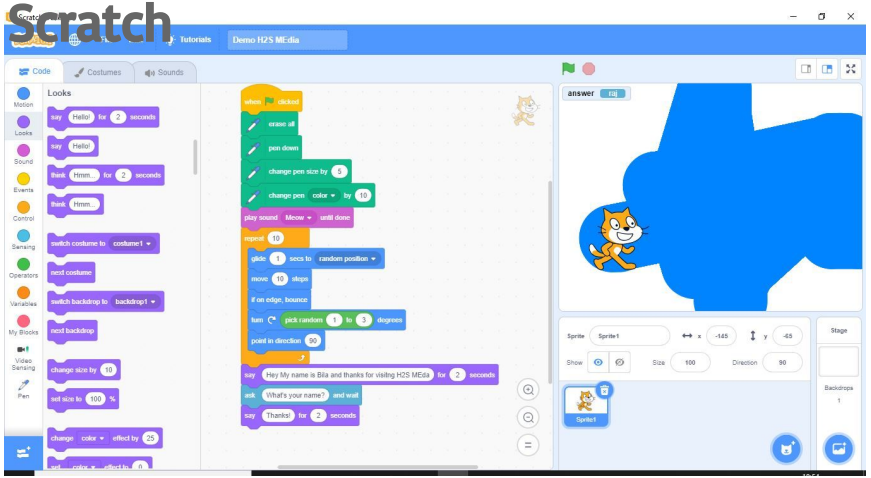
Theorem 2.1 (Marking Totality).

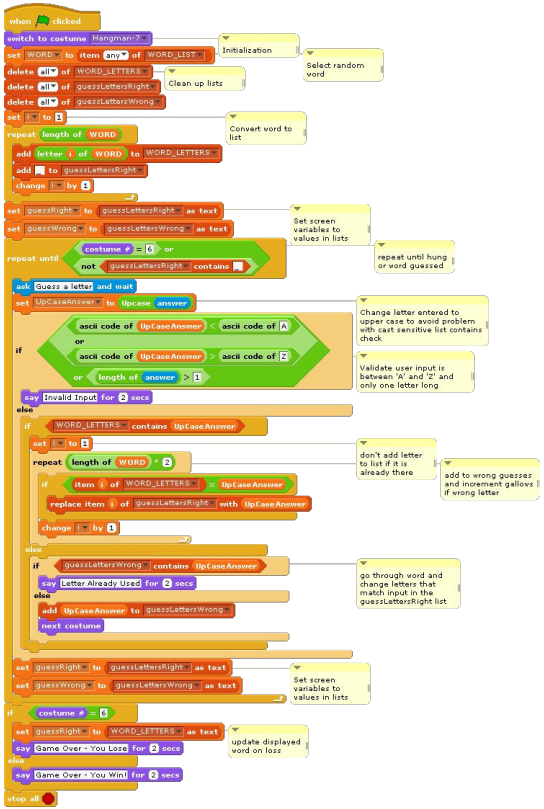
(1) *For all* Γ *and* e, *there exist* $\check{e}$ *and* τ *such that* $\Gamma \vdash e \hookrightarrow \check{e} \Rightarrow \tau$ *and* $\Gamma \vdash_M \check{e} \Rightarrow \tau$.

(2) *For all* Γ, e, *and* τ, *there exists* $\check{e}$ *such that* $\Gamma \vdash e \hookrightarrow \check{e} \Leftarrow \tau$ *and* $\Gamma \vdash_M \check{e} \Leftarrow \tau$.

Structure Editing has a Viscosity Problem

Block Editors like Scratch





Structure Editing has a Viscosity Problem

Keyboard-Driven Structure Editors like Jetbrains MPS

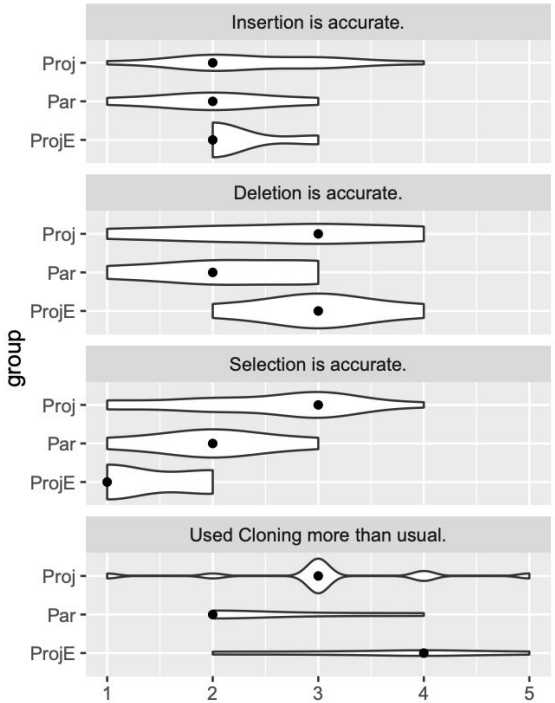
```
public class Folder {

  @Override
  public String toString() {
    StringBuilder content = new StringBuilder();
    $VAR$       [$LOOP$[content.append("$[filePath]"); ]]
      folderNode
    return content.toString();
  }
}
```

Inspector

jetbrains.mps.lang.generator.structure.PropertyMacro — Open Concept Declaration

```
property value

comment : <none>
value : (templateValue, genContext, node)->string {
          genContext.fileIndex + " - " + genContext.folderNode.name + "/" + node.name;
        }
```

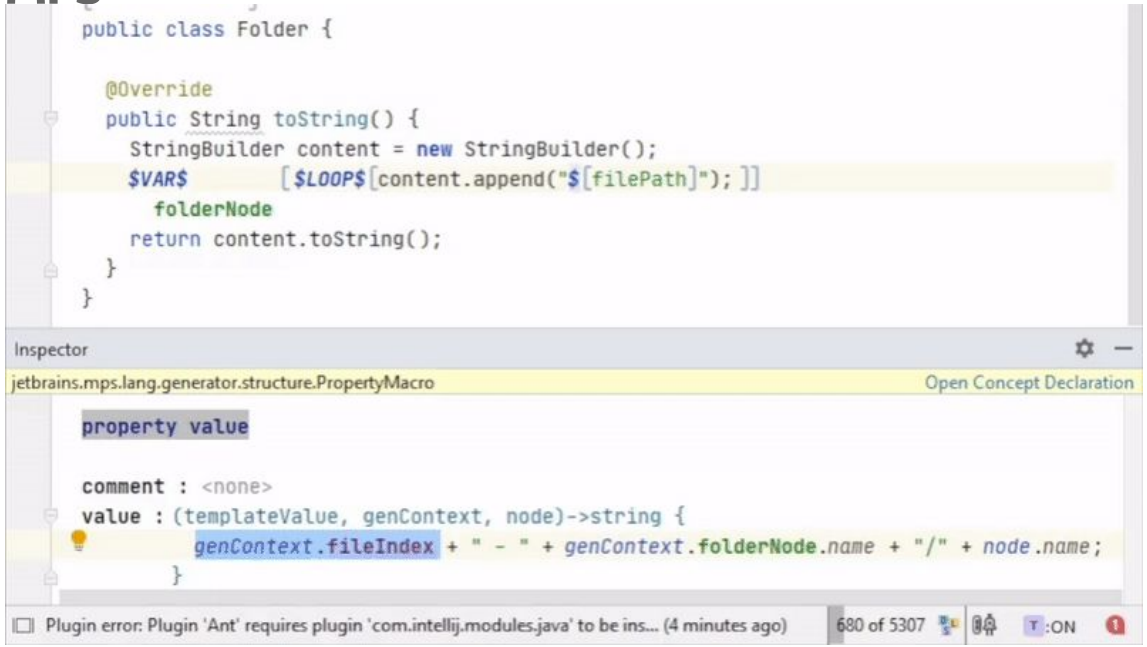
Plugin error: Plugin 'Ant' requires plugin 'com.intellij.modules.java' to be ins... (4 minutes ago) 680 of 5307 :ON



[Berger et al., FSE 2016]

Structure Editing has a Viscosity Problem

Keyboard-Driven Structure Editors like Jetbrains MPS



"MPS was EXTREMELY cognitively demanding to me; it felt like I was solving tree-manipulating puzzles the entire time I used it… the worst part by far is the lack of 'scratch' workspace" (P2)

"Towers of Hanoi" (P1, P4, P5)

[Moon et al., VL/HCC 2022]

Solving the Gap Problem...

The Hazel structure editor ensures that there are no syntax errors by inserting holes. The semantics understands holes , so every editor state is therefore syntactically, statically, and dynamically well-defined (i.e. Hazel is totally live !)

[POPL 2024, OOPSLA 2023, POPL 2019, POPL 2017]

Solving the Gap Problem...

The Hazel structure editor ensures that there are no syntax errors by inserting holes. The semantics understands holes , so every editor state is therefore syntactically, statically, and dynamically well-defined (i.e. Hazel is totally live !)

[POPL 2024, OOPSLA 2023, POPL 2019, POPL 2017]

...and the Viscosity Problem

Solving the Gap Problem...

The Hazel structure editor ensures that there are no syntax errors by inserting holes. The semantics understands holes, so every editor state is therefore syntactically, statically, and dynamically well-defined (i.e. Hazel is totally live!)

[POPL 2024, OOPSLA 2023, POPL 2019, POPL 2017]

...and the Viscosity Problem

The Hazel structure editor introduces gradual structure editing, which allows direct manipulation of the textual projection, rather than requiring the user to work only with tree transformations.

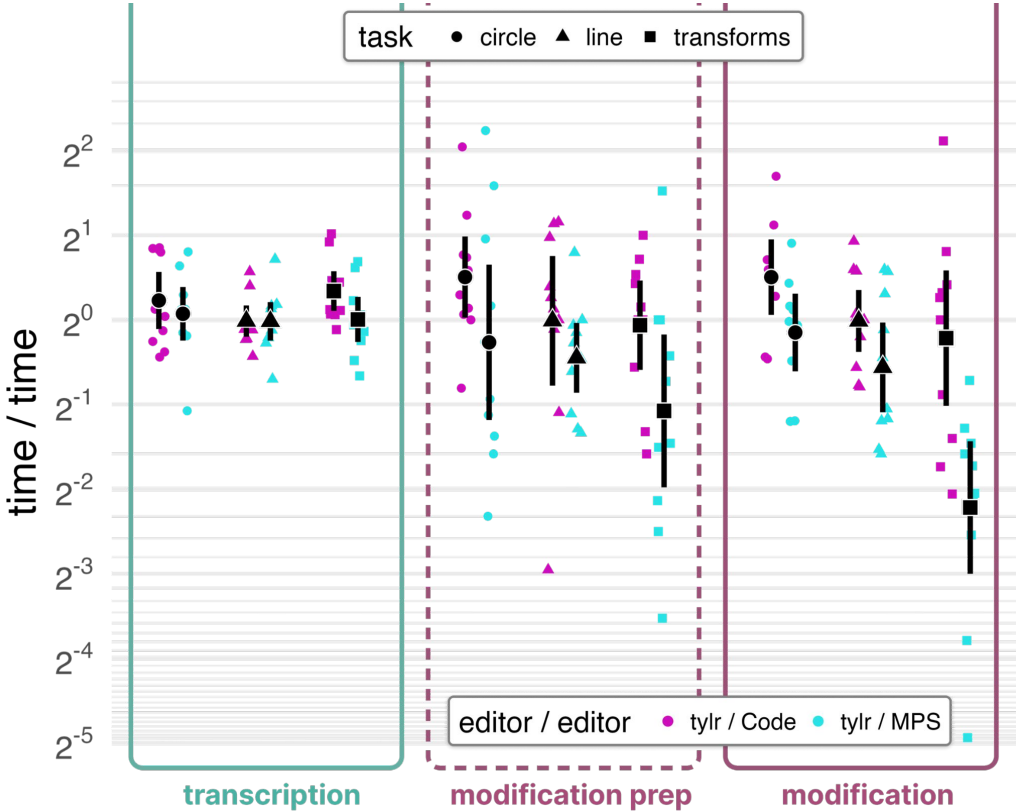
[VL/HCC 2023, TyDe 2022, ongoing work]

Hazel: A Totally Live Demo

Try it yourself:

hazel.org/build/dev

Usability of Gradual Structure Editing



Gradual structure editing is about as productive as text editing, more so than MPS for code modification. [VL/HCC 2023]

Usability of Gradual Structure Editing

inserted via typing by | 0 | 1 - 2 | 3 - 4 | 5 - 6 | 7 - 8 | 9 - 10 | participants

Code

```
let dist =
  fun p1, p2 ->
    let x1, y1 = p1 in
    let x2, y2 = p2 in
    sqrt(pow(x1 - x2, 2) + pow(y1 - y2, 2))
in
fun center, p ->
  let r = dist(center, p) in
  circle(center, r)


fun square, p1, p2 ->
  let mark =
   fun center ->
      if square then
        let x, y = center in
        rect(x - 2, y - 2, 4, 4)
      else
        let r = 4 in
        circle(center, r)
  in
  [mark(p1); line(p1, p2); mark(p2)]


shapes
|> filter(fun shape -> area(shape) < 50)
|> map(dilate(5))
|> map(rotate(pi / 4))
|> map(translate(6, 7))
```

MPS

```
let dist =
  fun p1, p2 ->
    let x1, y1 = p1 in
    let x2, y2 = p2 in
    sqrt(pow(x1 - x2, 2) + pow(y1 - y2, 2))
in
fun center, p ->
  let r = dist(center, p) in
  circle(center, r)


fun square, p1, p2 ->
  let mark =
   fun center ->
      if square then
        let x, y = center in
        rect(x - 2, y - 2, 4, 4)
      else
        let r = 4 in
        circle(center, r)
  in
  [mark(p1); line(p1, p2); mark(p2)]


shapes
|> filter(fun shape -> area(shape) < 50)
|> map(dilate(5))
|> map(rotate(pi / 4))
|> map(translate(6, 7))
```

tylr

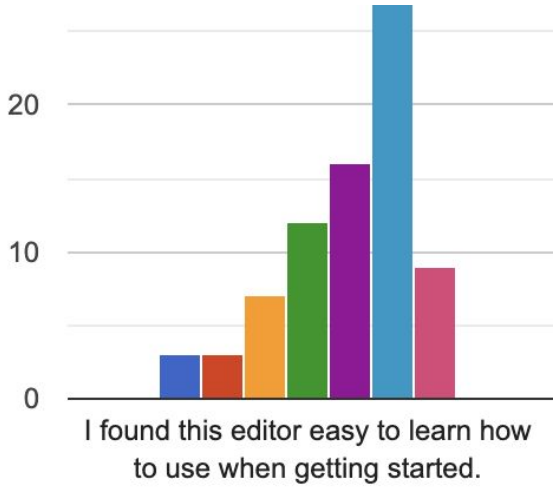
```
let dist =
  fun p1, p2 ->
    let x1, y1 = p1 in
    let x2, y2 = p2 in
    sqrt(pow(x1 - x2, 2) + pow(y1 - y2, 2))
in
fun center, p ->
  let r = dist(center, p) in
  circle(center, r)


fun square, p1, p2 ->
  let mark =
   fun center ->
      if square then
        let x, y = center in
        rect(x - 2, y - 2, 4, 4)
      else
        let r = 4 in
        circle(center, r)
  in
  [mark(p1); line(p1, p2); mark(p2)]


shapes
|> filter(fun shape -> area(shape) < 50)
|> map(dilate(5))
|> map(rotate(pi / 4))
|> map(translate(6, 7))
```

Gradual structure editing allows code modification tasks to be patterned as they are in text editors. [VL/HCC 2023]

Teaching with Hazel: Initial Feedback

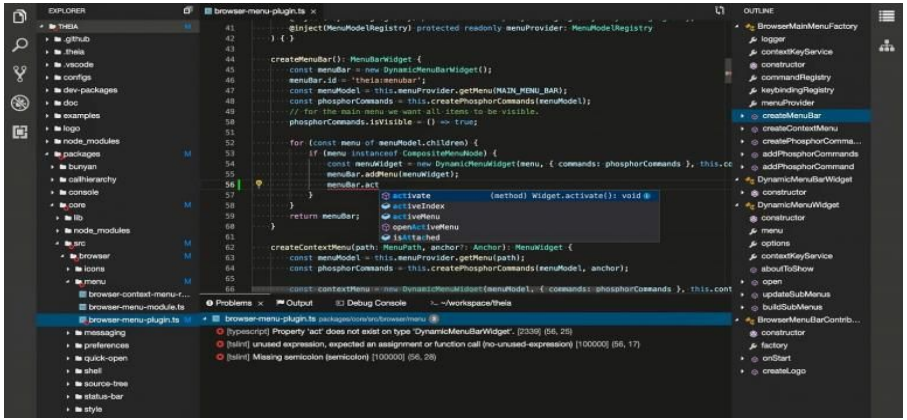
We use Hazel in an undergraduate PL course (which introduces FP).

- **200+ students** have used it successfully to **complete ~1-3 hour coding tasks** after only a **~5 minute tutorial** on how to edit in Hazel.



"The Hazel editor is by far the most user friendly IDE/editor I have ever used!!! The colors/design are awesome and made me feel welcomed (sounds cheesy but true) as a newcomer to functional programming, and the colorful highlighting and explanations were lifesavers!! I have never used a better editor, and I was sad we had to switch to Learn OCaml! I also loved the Swift Playgrounds-style "insta-run" and the instant feedback for the code questions!"

Abstract & Symbolic

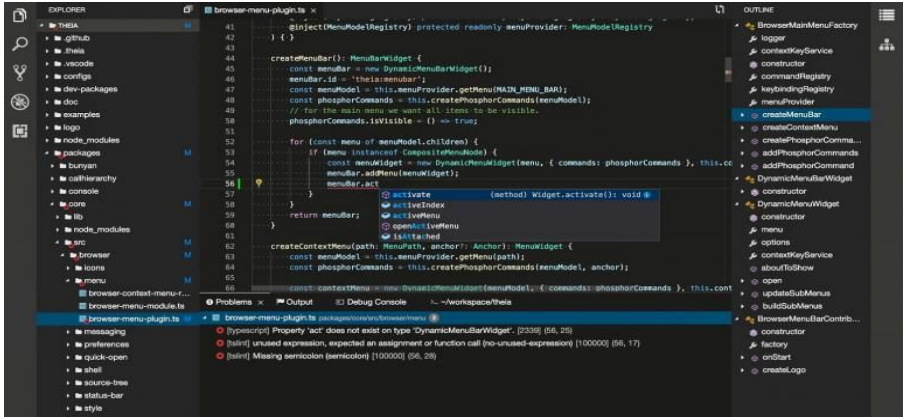


+ Generic symbolic representations
+ Symbol manipulation affordances
+ Abstraction and composition and calculation and automation

Live & Direct

+ Domain-specific representations
+ Direct manipulation affordances
+ Live (immediate + uninterrupted) feedback

Abstract & Symbolic

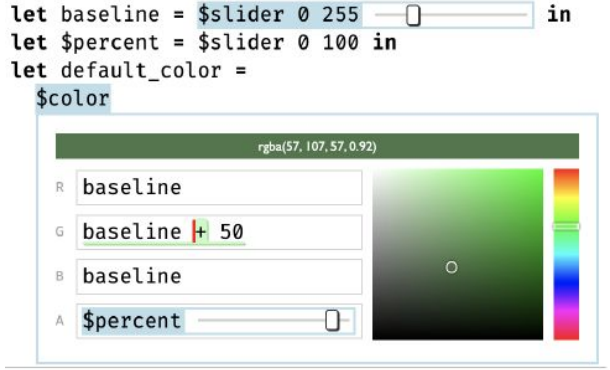


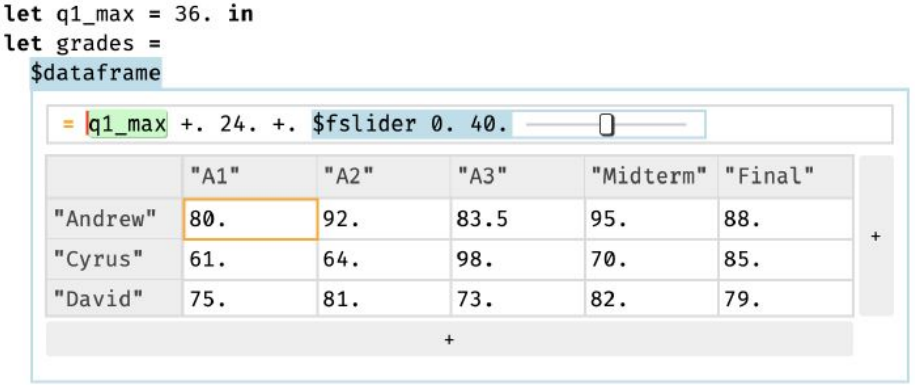
+ Generic symbolic representations
+ Symbol manipulation affordances
+ Abstraction and composition and calculation and automation

Live & Direct → Abstract & Symbolic

Hazel

Livelits: Filling Typed Holes with Live GUIs (PLDI 2021)





+ Domain-specific representations
+ Direct manipulation affordances
+ Live (immediate + uninterrupted) feedback

+ Generic symbolic representations
+ Symbol manipulation affordances
+ Abstraction and composition and calculation and automation

Big Picture

Combining foundational PL + HCI research enables ambitious applications:

- Building state-of-the-art educational technology
- Building usable formal reasoning assistants
- Building collaborative computational science environments
- Sensibly integrating semantics + large language models + humans
- Together, we could build a live computational commons

Hazel as a Classroom Proof Assistant

- **Specification and proof is an increasingly important component of a computer science education, but we still largely teach it on-paper. Can we make a classroom proof assistant ?**
  - Lots of prior work in this area informing our efforts! We wrote a survey that will be presented at HATRA 2024 this fall.
  - Key distinction: **scaffolding** vs. **support**

Demo: Hazel Stepper

https://hazel.org/build/dev/

Reasoning with Equations

```
let square(x) = fun x -> x * x in

theorem t:forall x:Nat -> square(x+1) = square(x) + 2*x + 1 in
```

proof:

```
forall x:Nat -> square(x+1) = square(x) + 2*x + 1
```

Perform an induction on `x:Nat`

case ⬡ :

```
square(⬡+1) = square(⬡) + 2*⬡ + 1
```

Missing proof. Select part of the expression to rewrite it or choose a proof pattern below.

Assume ? | Introduce ? | Nested Induction | Step

Add Case

(!) Proof is missing some cases

Demo: Logical Derivations

https://hazel.org/build/derivation/

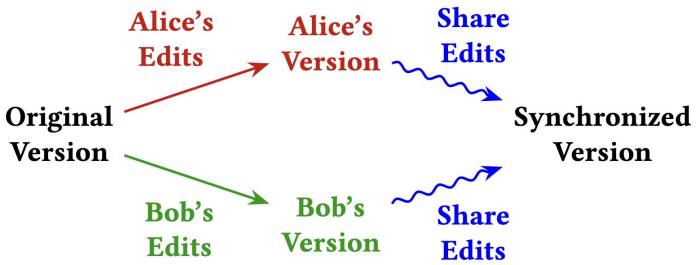
Big Picture

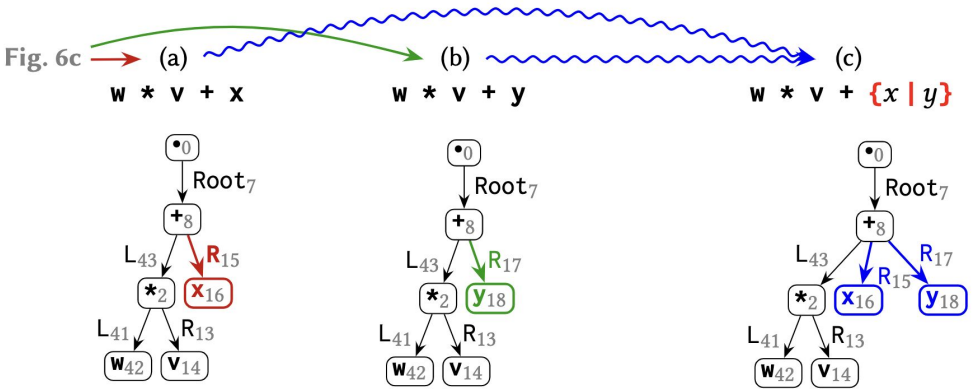
Combining foundational PL + HCI research enables ambitious applications:

- Building state-of-the-art educational technology
- Building usable formal reasoning assistants
- Building collaborative computational science environments
- Sensibly integrating semantics + large language models + humans
- Together, we could build a live computational commons

Ongoing Work

Collaborative Structure Editing with CmRDTs [in submission]





Ongoing Work

Collaborative Structure Editing with CmRDTs [in submission]