

Hazel: Semantic Foundations for Interactive Programming Tools



Cyrus Omar

Carnegie Mellon University

Q: What do programmers interact with?

Q: What do programmers interact with?

```
fun summary_stats(m : matrix<float>) =  
  { mean    = stats.mean(m, ColumnWise),  
    std     = stats.std(m,  
    median =
```

syntactically malformed program text

Syntactic error recovery heuristics

```
fun summary_stats(m : matrix<float>) =  
  { mean    = stats.mean(m, ColumnWise),  
    std     = stats.std(m, □),  
    median  = □  
  }
```

syntactically malformed program text → term with holes

Syntactic structure editors

```
fun summary_stats(m : matrix<float>) =  
  { mean    = stats.mean(m, ColumnWise),  
    std     = stats.std(m, 0),  
    median = 0  
  }
```

~~syntactically malformed program text~~ → term with holes

[Teitelbaum and Reps, Comm. ACM 1981; many others]

Q: How to reason statically about terms with holes?

```
fun summary_stats(m : matrix<float>) =  
  { mean    = stats.mean(m, ColumnWise),  
    std     = stats.std(m, 0),  
    median  = 0  
  }
```

Q: How to reason statically about terms with holes?

```
fun summary_stats(m : matrix<float>) =  
  { mean    = stats.mean(m, ColumnWise),  
    std     = stats.std(m, □),  
    median  = □  
  }
```

What **type** of expression is expected here?

Q: How to reason statically about terms with holes?

What **type** is synthesized for the function as a whole?

```
fun summary_stats(m : matrix<float>) =  
  { mean    = stats.mean(m, ColumnWise),  
    std     = stats.std(m, □),  
    median  = □  
  }
```

What **type** of expression is expected here?

Q: How to reason statically about terms with holes?

What **type** is synthesized for the function as a whole?

```
fun summary_stats(m : matrix<float>) =  
  { mean    = stats.mean(m, ColumnWise),  
    std     = stats.std(m, □),  
    median  = □  
  }
```

What **type** of expression is expected here?

A: A static semantics for terms with holes.

Q: How to reason statically about terms with holes?

What **type** is synthesized for the function as a whole?

```
fun summary_stats(m : matrix<float>) =  
  { mean    = stats.mean(m, ColumnWise),  
    std     = stats.std(m, □),  
    median  = □  
  }
```

What **type** of expression is expected here?

A: A static semantics for terms with holes.

[Omar et al., POPL 2017]

Q: How to reason statically about terms with holes?

What **type** is synthesized for the function as a whole?

```
fun summary_stats(m : matrix<float>) =  
  { mean    = stats.mean(m, ColumnWise),  
    std     = stats.std(m, □),  
    median  = □  
  }
```

What **type** of expression is expected here?

```
matrix<float> →  
{ mean    : vec<float>,  
  std     : vec<float>,  
  median  : □ }
```

A: A static semantics for terms with holes.

[Omar et al., POPL 2017]

Q: How to reason statically about terms with holes?

What **type** is synthesized for the function as a whole?

```
matrix<float> →  
{ mean   : vec<float>,  
  std    : vec<float>,  
  median : □ }
```

```
fun summary_stats(m : matrix<float>) =  
  { mean   = stats.mean(m, ColumnWise),  
    std    = stats.std(m, □),  
    median = □  
  }
```

What **type** of expression is expected here? (RowWise + ColumnWise)

A: A static semantics for terms with holes.

[Omar et al., POPL 2017]

Q: How to reason statically about terms with type errors?

What **type** is synthesized for the function as a whole?



```
fun summary_stats(m : matrix<float>) =  
  { mean    = stats.mean(m, ColumnWise),  
    std     = stats.std(m, "oops"),  
    median  = □  
  }
```

A: A static semantics for terms with holes.

[Omar et al., POPL 2017]

Q: How to reason statically about terms with type errors?

What **type** is synthesized for the function as a whole?



```
fun summary_stats(m : matrix<float>) =  
  { mean    = stats.mean(m, ColumnWise),  
    std     = stats.std(m, "oops"),  
    median  = □  
  }
```

```
matrix<float> →  
{ mean    : vec<float>,  
  std     : vec<float>,  
  median  : □ }
```

A: A static semantics for terms with holes.

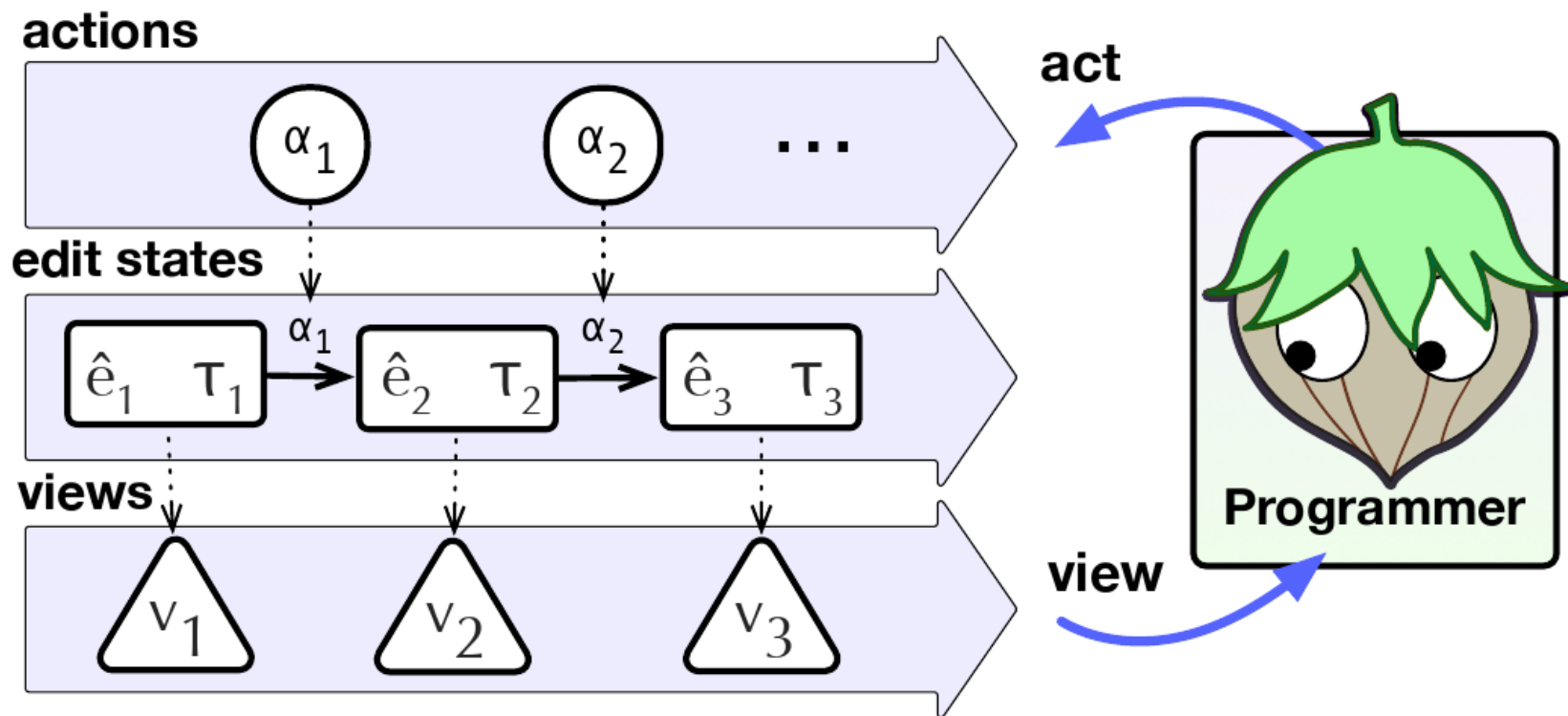
[Omar et al., POPL 2017]

A static semantics for lambda terms with holes

HTyp $\dot{\tau} ::= (\dot{\tau} \rightarrow \dot{\tau}) \mid \mathbf{num} \mid \langle \rangle$

HExp $\dot{e} ::= x \mid (\lambda x. \dot{e}) \mid \dot{e}(\dot{e}) \mid \underline{n} \mid (\dot{e} + \dot{e}) \mid \dot{e} : \dot{\tau} \mid \langle \rangle \mid \langle \dot{e} \rangle$

A typed edit action semantics



[Omar et al., POPL 2017]

See <http://hazeltrove.github.io/>

From Hazelnut to Hazel



Hazel

- Numerics ▾
- Plotting ▾
- Statistics ▾
- +

```
fun summary_stats(m : matrix<float>)  
  { mean = mean(m, ColumnWise)  
    std  = std(m, □)  
    median = □ } (a)
```

```
let my_data : matrix<float> =  
  [ [ 1.1 | 2.3 | 3.0 | 4.1 | 5.2 ]  
    [ 1.2 | 1.8 | 3.1 | 4.1 | 5.2 ]  
    [ 0.9 | 2.2 | 2.7 | 3.5 | 4.9 ]  
    [ 0.8 | 1.5 | 3.3 | 4.3 | 4.7 ] ] (b)
```

```
summary_stats(my_data)  
  
{ mean = [ 1.0 | 2.0 | 3.0 | 4.0 | 5.0 ]  
  std  = std(my_data, □)  
  median = □ } (c)
```

Type at cursor: dimension

Action search... (d)

- ColumnWise (most probable)
- RowWise
- Factor to variable...
- (□)
- Full action palette...

From Hazelnut to Hazel



Numerics ▾

Plotting ▾

Statistics ▾

+

```
fun summary_stats(m : matrix<float>)
```

```
{ mean = mean(m, ColumnWise)
  std  = std(m, □)
  median = □ }
```

(a)

TODO: scale up POPL17

```
let my_data : matrix<float> =
```

1.1	2.3	3.0	4.1	5.2
1.2	1.8	3.1	4.1	5.2
0.9	2.2	2.7	3.5	4.9
0.8	1.5	3.3	4.3	4.7

(b)

```
summary_stats(my_data)
```

```
{ mean = [1.0 | 2.0 | 3.0 | 4.0 | 5.0]
  std  = std(my_data, □)
  median = □ }
```

(c)

Type at cursor: dimension

Action search...

(d)

ColumnWise (most probable)

RowWise

Factor to variable...

□(□)

Full action palette...

From Hazelnut to Hazel



Hazel

Numerics ▾ Plotting ▾ Statistics ▾ +

```
fun summary_stats(m : matrix<float>)  
  { mean = mean(m, ColumnWise)  
    std  = std(m, □)  
    median = □ } (a)
```

```
let my_data : matrix<float> =  
  [ 1.1 | 2.3 | 3.0 | 4.1 | 5.2  
    1.2 | 1.8 | 3.1 | 4.1 | 5.2  
    0.9 | 2.2 | 2.7 | 3.5 | 4.9  
    0.8 | 1.5 | 3.3 | 4.3 | 4.7 ] (b)
```

```
summary_stats(my_data)  
  
{ mean = [1.0 | 2.0 | 3.0 | 4.0 | 5.0]  
  std  = std(my_data, □)  
  median = □ } (c)
```

TODO: type-specific projections
(based on my work at ICSE 2012, ECOOP 2014)

Type at cursor: dimension

Action search... (d)

ColumnWise (most probable)

□(□)

Full action palette...

From Hazelnut to Hazel



Hazel

- Numerics ▾
- Plotting ▾
- Statistics ▾
- +

```
fun summary_stats(m : matrix<float>)  
  { mean = mean(m, ColumnWise) }  
  { std = std(m, □) }  
  { median = □ }
```

(a)

```
let my_data : matrix<float> =  
  [ [ 1.1 | 2.3 | 3.0 | 4.1 | 5.2 ]  
    [ 1.2 | 1.8 | 3.1 | 4.1 | 5.2 ]  
    [ 0.9 | 2.2 | 2.7 | 3.5 | 4.9 ]  
    [ 0.8 | 1.5 | 3.3 | 4.3 | 4.7 ] ]
```

(b)

```
summary_stats(my_data)  
  
{ mean = [ 1.0 | 2.0 | 3.0 | 4.0 | 5.0 ] }  
{ std = std(my_data, □) }  
{ median = □ }
```

(c)

TODO: a dynamic semantics for incomplete programs (very live programming)

Type at cursor: dimension

Action search... (d)

- ColumnWise (most probable)
- RowWise
- Factor to variable...
- (□)
- Full action palette...

From Hazelnut to Hazel



Hazel

- Numerics ▾
- Plotting ▾
- Statistics ▾
- + ▾

TODO: an action suggestion semantics

```
fun summary_stats(m : matrix<float>)  
  { mean = mean(m, ColumnWise) }  
  { std = std(m, □) }  
  { median = □ }
```

(a)

```
let my_data : matrix<float> =  
  [ [ 1.1 | 2.3 | 3.0 | 4.1 | 5.2 ]  
    [ 1.2 | 1.8 | 3.1 | 4.1 | 5.2 ]  
    [ 0.9 | 2.2 | 2.7 | 3.5 | 4.9 ]  
    [ 0.8 | 1.5 | 3.3 | 4.3 | 4.7 ] ]
```

(b)

```
summary_stats(my_data)
```

```
{ mean = [ 1.0 | 2.0 | 3.0 | 4.0 | 5.0 ] }  
{ std = std(my_data, □) }  
{ median = □ }
```

(c)

Action search... (d)

- ColumnWise (most probable)
- RowWise
- Factor to variable...
- (□)
- Full action palette...

From Hazelnut to Hazel



Hazel

- Numerics ▾
- Plotting ▾
- Statistics ▾
- +

```
fun summary_stats(m : matrix<float>)  
  { mean = mean(m, ColumnWise) }  
  { std = std(m, □) }  
  { median = □ }
```

(a)

```
let my_data : matrix<float> =  
  [ [ 1.1 | 2.3 | 3.0 | 4.1 | 5.2 ]  
    [ 1.2 | 1.8 | 3.1 | 4.1 | 5.2 ]  
    [ 0.9 | 2.2 | 2.7 | 3.5 | 4.9 ]  
    [ 0.8 | 1.5 | 3.3 | 4.3 | 4.7 ] ]
```

TODO: a statistical model of edit actions

```
summary_stats(my_data)  
  
{ mean = [ 1.0 | 2.0 | 3.0 | 4.0 | 5.0 ] }  
{ std = std(my_data, □) }  
{ median = □ }
```

(c)

Type at cursor: dimension

Action search... (d)

ColumnWise (most probable)

□(□)

Full action palette...

From Hazelnut to Hazel



Hazel

Numerics ▾ Plotting ▾ Statistics ▾ +

```
fun summary_stats(m : matrix<float>)  
  { mean = mean(m, ColumnWise)  
    std  = std(m, □)  
    median = □ } (a)
```

```
let my_data : matrix<float> =  
  [ 1.1 | 2.3 | 3.0 | 4.1 | 5.2 ] (b)  
  [ 1.2 | 1.8 | 3.1 | 4.1 | 5.2 ]  
  [ 0.9 | 2.2 | 2.7 | 3.5 | 4.9 ]  
  [ 0.8 | 1.5 | 3.3 | 4.3 | 4.7 ]
```

```
summary_stats(my_data)  
  
{ mean = [1.0 | 2.0 | 3.0 | 4.0 | 5.0]  
  std  = std(my_data, □)  
  median = □ } (c)
```

Type at cursor: dimension

Action search... (d)

- ColumnWise (most probable)
- RowWise
- Factor to variable...

TODO: library-defined derived actions

From Hazelnut to Hazel

 Hazel

Numerics ▾ Plotting ▾ Statistics ▾ +

```
fun summary_stats(m : matrix<float>)  
  { mean = mean(m, ColumnWise) } (a)  
  { std = std(m, □) }  
  { median = □ }
```

```
let my_data : matrix<float> = 

|     |     |     |     |     |
|-----|-----|-----|-----|-----|
| 1.1 | 2.3 | 3.0 | 4.1 | 5.2 |
| 1.2 | 1.8 | 3.1 | 4.1 | 5.2 |
| 0.9 | 2.2 | 2.7 | 3.5 | 4.9 |
| 0.8 | 1.5 | 3.3 | 4.3 | 4.7 |

 (b)
```

```
summary_stats(my_data)  
  
{ mean = [1.0 | 2.0 | 3.0 | 4.0 | 5.0] } (c)  
{ std = std(my_data, □) }  
{ median = □ }
```

Type at cursor: dimension

Action search... (d)

- ColumnWise (most probable)
- RowWise
- Factor to variable...
- (□)
- Full action palette...

Joint work with Ian Voysey (CMU), Matt Hammer (CU Boulder), Michael Hilton (Oregon State), Claire Le Goues (CMU), Jonathan Aldrich (CMU), Josh Sunshine (CMU).
Interested? Contact me! <http://hazलगrove.github.io/>