Toward Semantic Foundations for Program Editors

Cyrus Omar
Ian Voysey
Michael Hilton
Joshua Sunshine
Claire Le Goues
Jonathan Aldrich
Matthew Hammer

Carnegie Mellon University
Carnegie Mellon University
Oregon State University
Carnegie Mellon University
Carnegie Mellon University
Carnegie Mellon University
University of Colorado Boulder
Semanticists usually study **complete** programs

```c
fun summary_stats(m : matrix<float>) =
{
    mean   = stats.mean(m, ColumnWise),
    std    = stats.std(m, ColumnWise),
    median = stats.median(m, ColumnWise)
}
```
Q: What about **incomplete** programs?

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

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

syntactically malformed program text → term with holes

[Kats et al., OOPSLA 2009]
fun summary_stats(m : matrix<float>) =
    { mean   = stats.mean(m, ColumnWise),
      std    = stats.std(m, □),
      median = □
    }

Syntactically malformed program text → term with holes

[Teitelbaum and Reps, Comm. ACM 1981; many others]
fun summary_stats(m : matrix<float>) =
{
    mean = stats.mean(m, ColumnWise),
    std = stats.std(m, □),
    median = □
}
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?

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

What type of expression is expected here?
fun summary_stats(m : matrix<float>) =
{
  mean   = stats.mean(m, ColumnWise),
  std    = stats.std(m, □),
  median = □
}

Q: How to reason statically about terms with holes?

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

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

What type is synthesized for the function as a whole?

matrix<float> →
{ mean   : vec<float>,
  std    : vec<float>,
  median : □ }
Q: How to reason statically about terms with holes?

What type is synthesized for the function as a whole?

```plaintext
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 inconsistencies?

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

```ocaml
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 inconsistencies?

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

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

Reify type inconsistencies as **non-empty** holes!

A: A **static semantics** for terms with holes.

[Omar et al., POPL 2017]
Q: How to **reason statically** about terms with type inconsistencies?

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

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

Reify type inconsistencies as **non-empty** holes!

**A:** A **static semantics** for terms with holes.

[Omar et al., POPL 2017]
A static semantics for lambda terms with holes

\[
\begin{align*}
\text{HTyp } \bar{\tau} &::= (\bar{\tau} \to \bar{\tau}) \mid \text{num} \mid () \\
\text{HExp } \bar{e} &::= x \mid (\lambda x.\bar{e}) \mid \bar{e}(\bar{e}) \mid n \mid (\bar{e} + \bar{e}) \mid \bar{e} : \bar{\tau} \mid () \mid (\bar{e})
\end{align*}
\]

[Omar et al., POPL 2017]
Hazelnut: A typed edit action semantics

[Omar et al., POPL 2017]
See http://hazelgrove.org/hazel/hazel.html
From Hazelnut to Hazel

fun summary_stats(m : matrix<float>):
    mean = mean(m, ColumnWise)
    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]

summary_stats(my_data)

    mean = [1.0 2.0 3.0 4.0 5.0]
    std  = std(my_data, □)
    median = □
From Hazelnut to Hazel

Web-based UI. Libraries are Git repos w/URLs.

```
fun summary_stats(m : matrix<float>)
{
    mean = mean(m, ColumnWise)
    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]
]

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

fun summary_stats(m : matrix<float>)
  { mean = mean(m, ColumnWise), 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] ]

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

TODO: scale up POPL17 (see Omar & Aldrich, GPCE 2016)
**From Hazelnut to Hazel**

```hazel
todo: type-specific projections
(see Omar et al., ICSE 2012, ECOOP 2014, thesis)
```

```haskell
fun summary_stats(m : matrix<float>)
{
  mean = mean(m, ColumnWise)
  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
summary_stats(my_data)
{
  mean = [1.0 2.0 3.0 4.0 5.0]
  std = std(my_data, □)
  median = □
}
```
TODO: a dynamic semantics for incomplete programs (very live programming)
From Hazelnut to **Hazel**

**TODO: an action suggestion semantics**

```hazel
fun summary_stats(m : matrix<float>)
    { mean = mean(m, ColumnWise) } 
    { 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]

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

- **ColumnWise** (most probable)
- **RowWise**
- Factor to variable...
- □(□)
- Full action palette...
### From Hazelnut to Hazel

#### Hazel

```hazel
fun summary_stats(m : matrix<float>)
    {mean = mean(m, ColumnWise)}
    {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]

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

**TODO:** library-defined derived actions (i.e. edit-time tactics)
TODO: a statistical model of suggestions (ML for ML!)
A real opportunity to apply foundational type theory and modern PL techniques to deliver a best-in-class programming experience.