Hazel: Semantic Foundations for Interactive Programming Tools

Cyrus Omar
Carnegie Mellon University
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
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 = □
}

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?
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 = □
}

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?

```plaintext
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?

```plaintext
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?

```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 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 : □ } [Omar et al., POPL 2017]

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

\[
\text{HTyp } \tilde{\tau} ::= (\tilde{\tau} \to \tilde{\tau}) \mid \text{num} \mid \emptyset
\]

\[
\text{HExp } \hat{e} ::= x \mid (\lambda x.\hat{e}) \mid \hat{e}(\hat{e}) \mid n \mid (\hat{e} + \hat{e}) \mid \hat{e} : \tilde{\tau} \mid \emptyset \mid (\hat{e})
\]

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

[Omar et al., POPL 2017]
See http://hazelgrove.github.io/
fun summary_stats(m : matrix<float>) {
    mean = mean(m, ColumnWise)
    std = std(m, □)
    median = □
}

let my_data : matrix<float> =

<table>
<thead>
<tr>
<th>1.1</th>
<th>2.3</th>
<th>3.0</th>
<th>4.1</th>
<th>5.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.2</td>
<td>1.8</td>
<td>3.1</td>
<td>4.1</td>
<td>5.2</td>
</tr>
<tr>
<td>0.9</td>
<td>2.2</td>
<td>2.7</td>
<td>3.5</td>
<td>4.9</td>
</tr>
<tr>
<td>0.8</td>
<td>1.5</td>
<td>3.3</td>
<td>4.3</td>
<td>4.7</td>
</tr>
</tbody>
</table>

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
From Hazelnut to Hazel

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

Type at cursor: dimension

Action search...

ColumnWise (most probable)
From Hazelnut to Hazel

There is a code snippet that calculates summary statistics for a matrix of floating-point numbers. It defines a function `summary_stats` that computes the mean, standard deviation, and median of the matrix. Here is the code:

```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 = □
    }
```

The TODO note mentions the development of a dynamic semantics for incomplete programs, which is a very live programming approach.
From Hazelnut to Hazel

```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: an action suggestion semantics

Action search...

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

22
From Hazelnut to 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: a statistical model of edit actions
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: library-defined derived actions
From Hazelnut to Hazel

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://hazelgrove.github.io/