Toward Semantic Foundations for Program Editors

Cyrus Omar	Carnegie Mellon University
Ian Voysey	Carnegie Mellon University
Michael Hilton	Oregon State University
Joshua Sunshine	Carnegie Mellon University
Claire Le Goues	Carnegie Mellon University
Jonathan Aldrich	Carnegie Mellon University
Matthew Hammer	University of Colorado Boulder

SNAPL 2017

Semanticists usually study complete programs

```
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?

```
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

[Kats et al., OOPSLA 2009]

Syntactic structure editors

```
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]

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 = ▢
  }
```

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?

```
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?

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 inconsistencies?

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 inconsistencies?

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 = ▢
  }
```

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?

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

```
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.

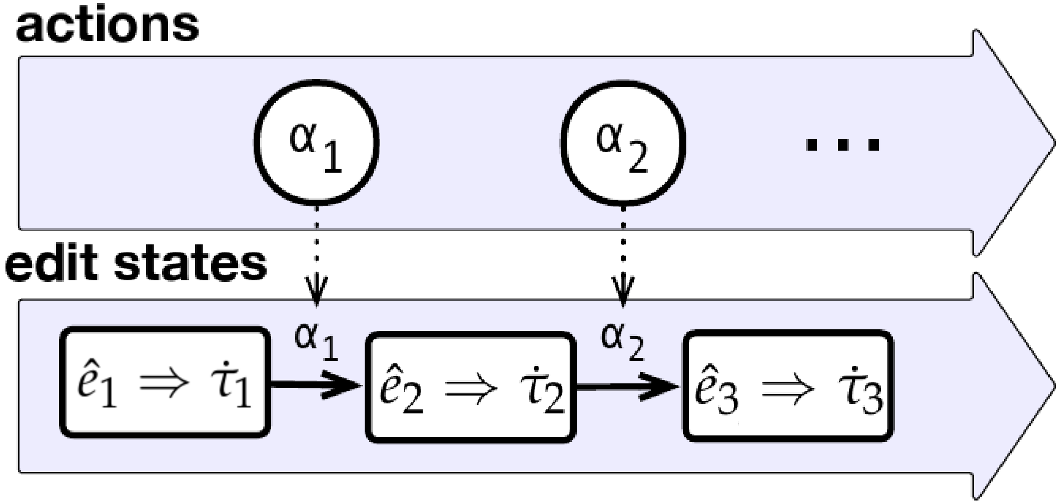
[Omar et al., POPL 2017]

A **static semantics** for lambda terms with holes

$$
\begin{array}{llll}
\mathsf{HTyp} & \dot{\tau} & ::= & (\dot{\tau} \to \dot{\tau}) \mid \mathtt{num} \mid \llparenthesis\rrparenthesis \\
\mathsf{HExp} & \dot{e} & ::= & x \mid (\lambda x.\dot{e}) \mid \dot{e}(\dot{e}) \mid \underline{n} \mid (\dot{e} + \dot{e}) \mid \dot{e} : \dot{\tau} \mid \llparenthesis\rrparenthesis \mid \llparenthesis\dot{e}\rrparenthesis
\end{array}
$$

[Omar et al., POPL 2017]

Hazelnut: A typed edit action semantics


actions
α_1
α_2
...
edit states
α_1
α_2
$\hat{e}_1 \Rightarrow \dot{\tau}_1$
$\hat{e}_2 \Rightarrow \dot{\tau}_2$
$\hat{e}_3 \Rightarrow \dot{\tau}_3$

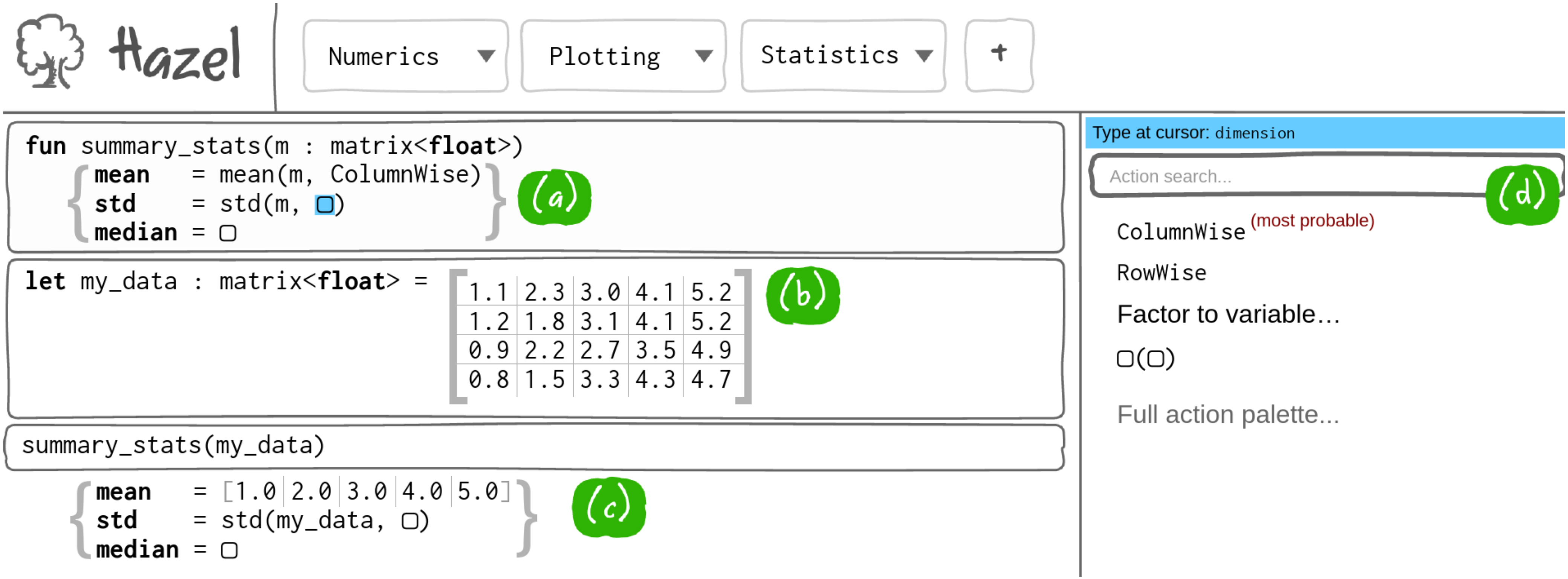
[Omar et al., POPL 2017]

Demo

See http://hazelgrove.org/hazel/hazel.html

[Omar et al., POPL 2017]

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 = ▢ }
```

Type at cursor: `dimension`

Action search...

ColumnWise (most probable)

RowWise

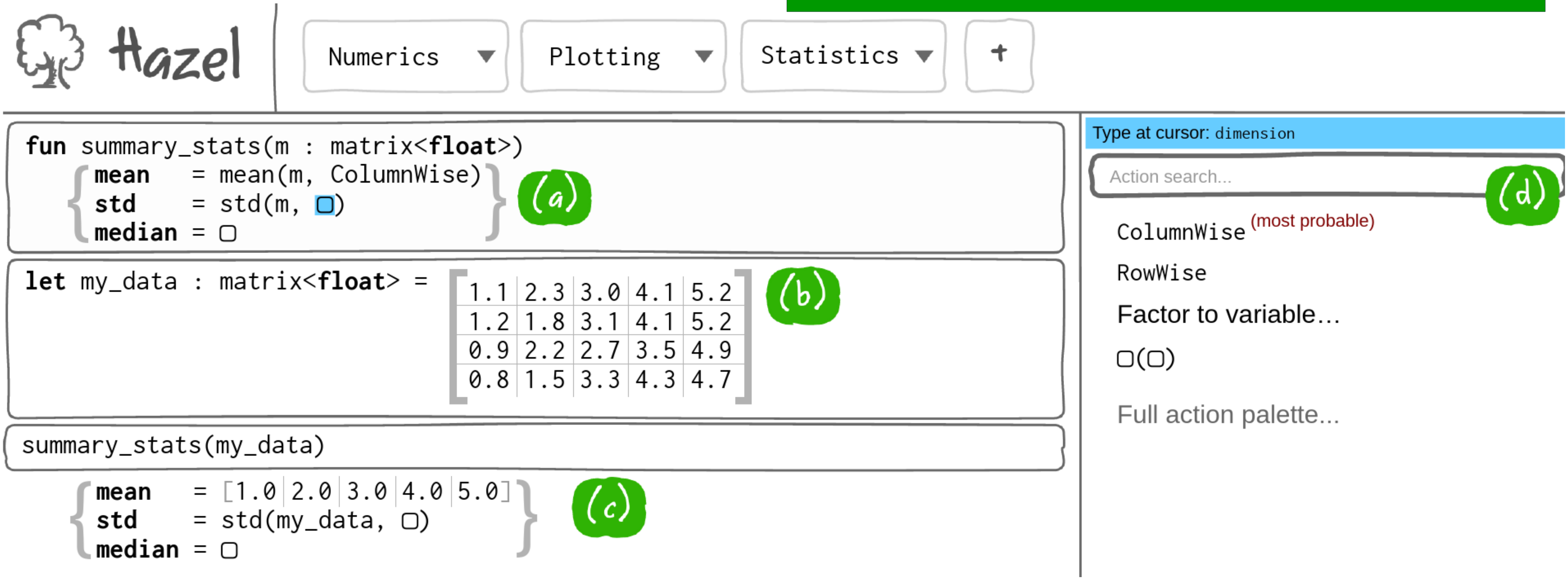
Factor to variable…

▢(▢)

Full action palette...

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 = □                             }
```

Type at cursor: dimension

Action search...

ColumnWise (most probable)

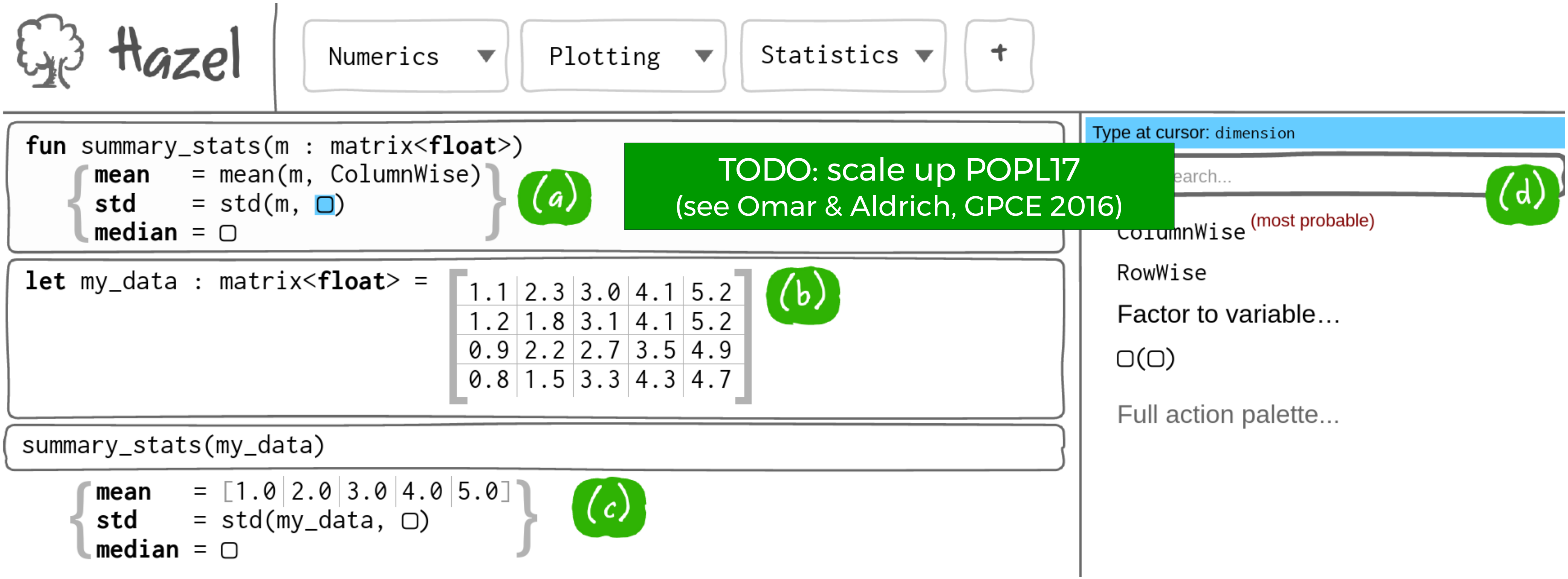
RowWise

Factor to variable…

□(□)

Full action palette...

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 = □                      }
```

Type at cursor: dimension

Search...

ColumnWise (most probable)

RowWise

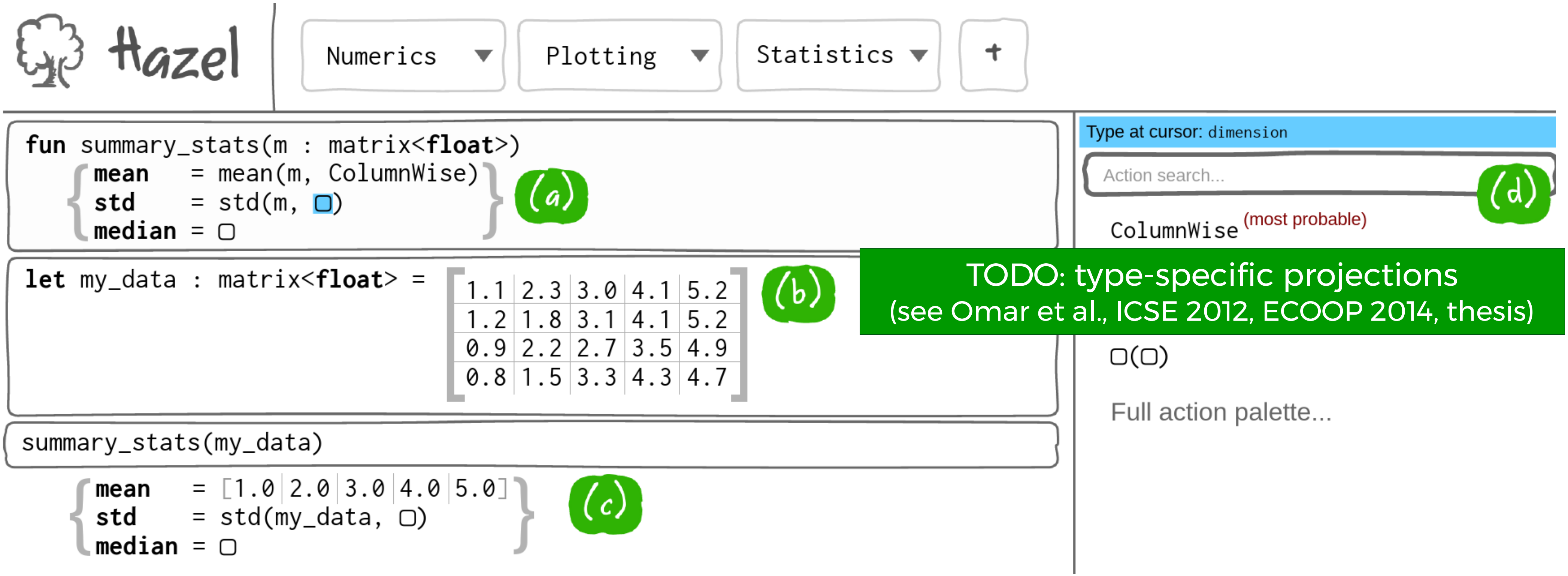
Factor to variable…

□(□)

Full action palette...

TODO: scale up POPL17
(see Omar & Aldrich, GPCE 2016)

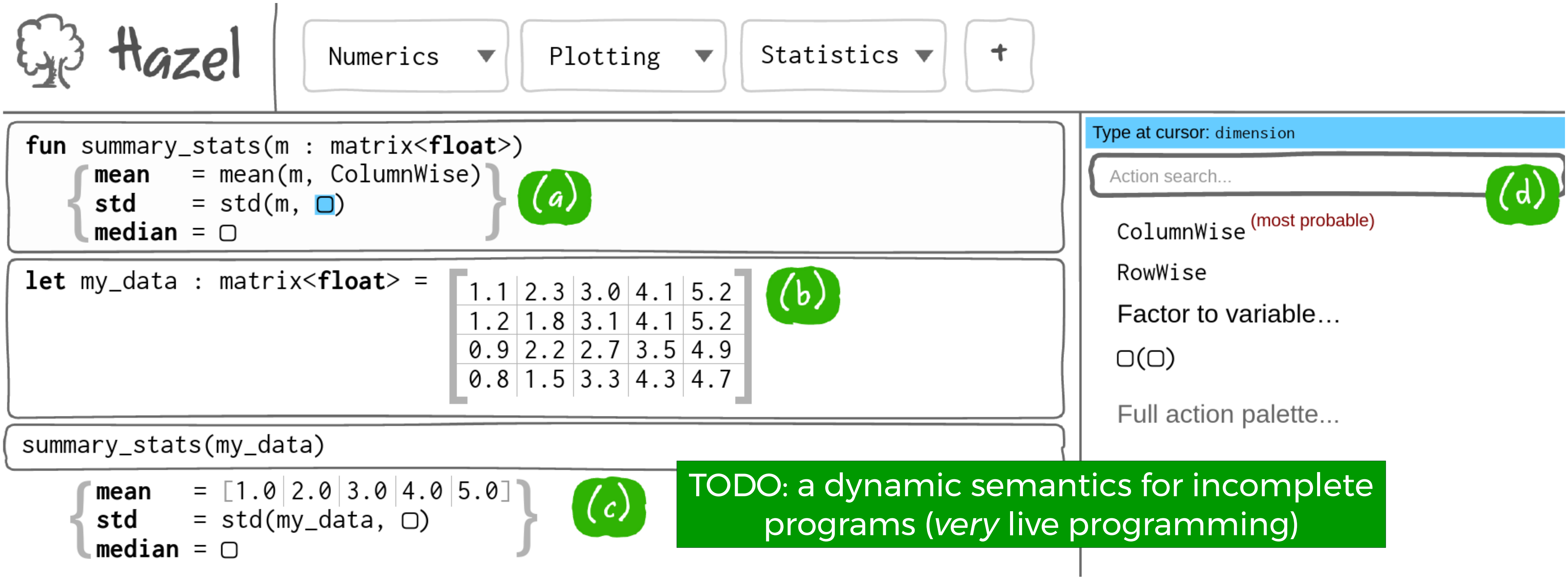
From Hazelnut to Hazel



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

From Hazelnut to **Hazel**



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

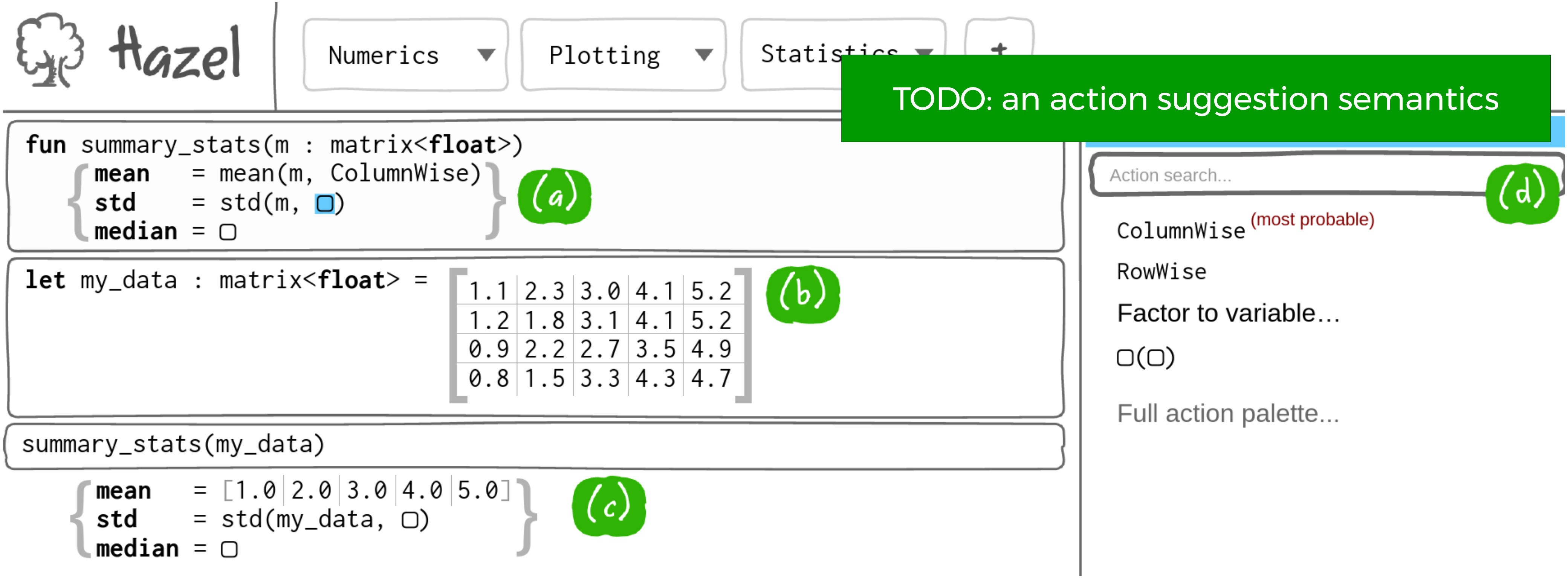
```
let my_data : matrix<float> =
    [ 1.1 | 2.3 | 3.0 | 4.1 | 5.2
      1.2 | 1.8 | 3.1 | 4.1 | 5.2
      0.9 | 2.2 | 2.7 | 3.5 | 4.9
      0.8 | 1.5 | 3.3 | 4.3 | 4.7 ]
```

```
summary_stats(my_data)
```

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

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

From Hazelnut to **Hazel**



TODO: an action suggestion semantics

```
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 = □                             }
```

Action search...

ColumnWise (most probable)

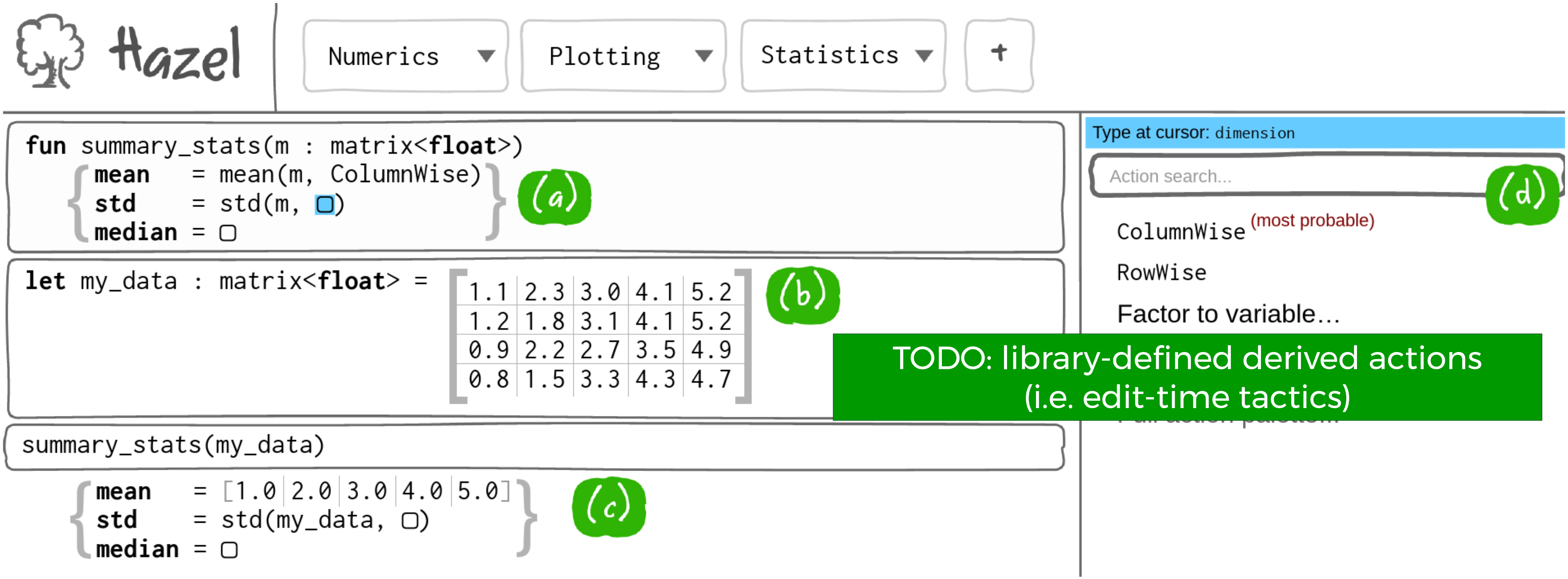
RowWise

Factor to variable…

□(□)

Full action palette...

From Hazelnut to **Hazel**



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

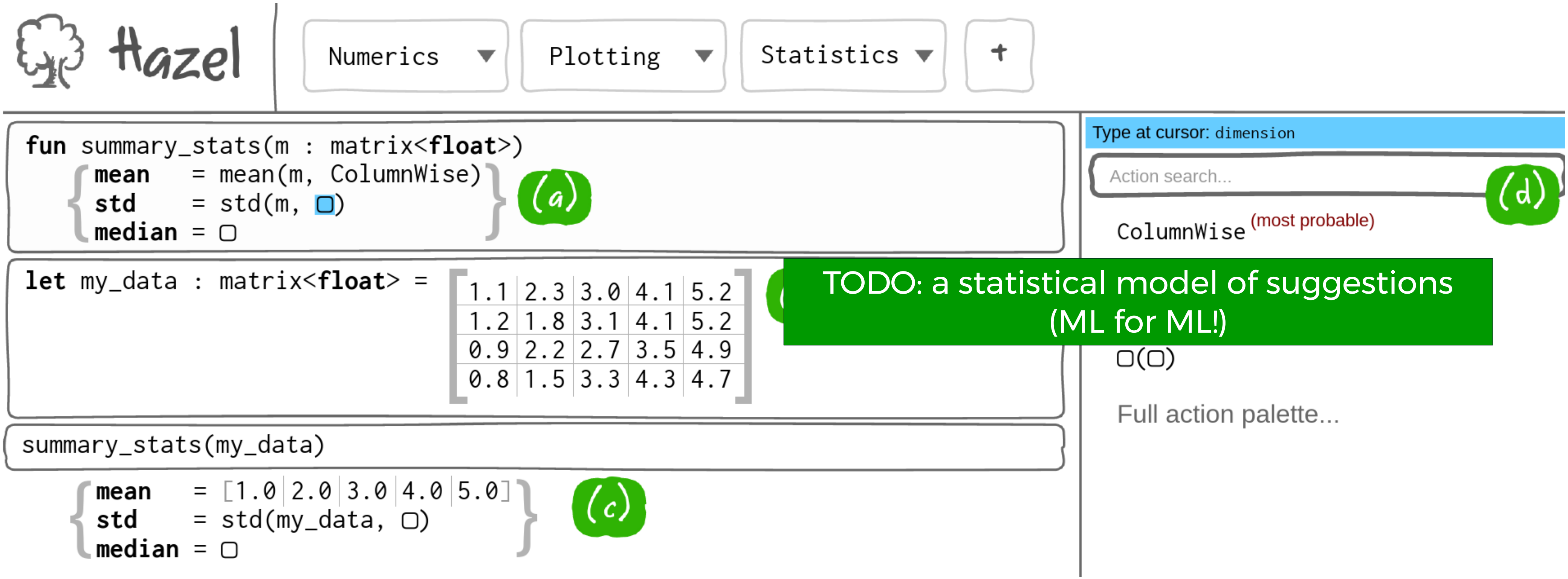
```
let my_data : matrix<float> = [ 1.1 | 2.3 | 3.0 | 4.1 | 5.2 ]
                              [ 1.2 | 1.8 | 3.1 | 4.1 | 5.2 ]
                              [ 0.9 | 2.2 | 2.7 | 3.5 | 4.9 ]
                              [ 0.8 | 1.5 | 3.3 | 4.3 | 4.7 ]
```

```
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
(i.e. edit-time tactics)

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 = □                             }
```

Type at cursor: `dimension`

Action search...

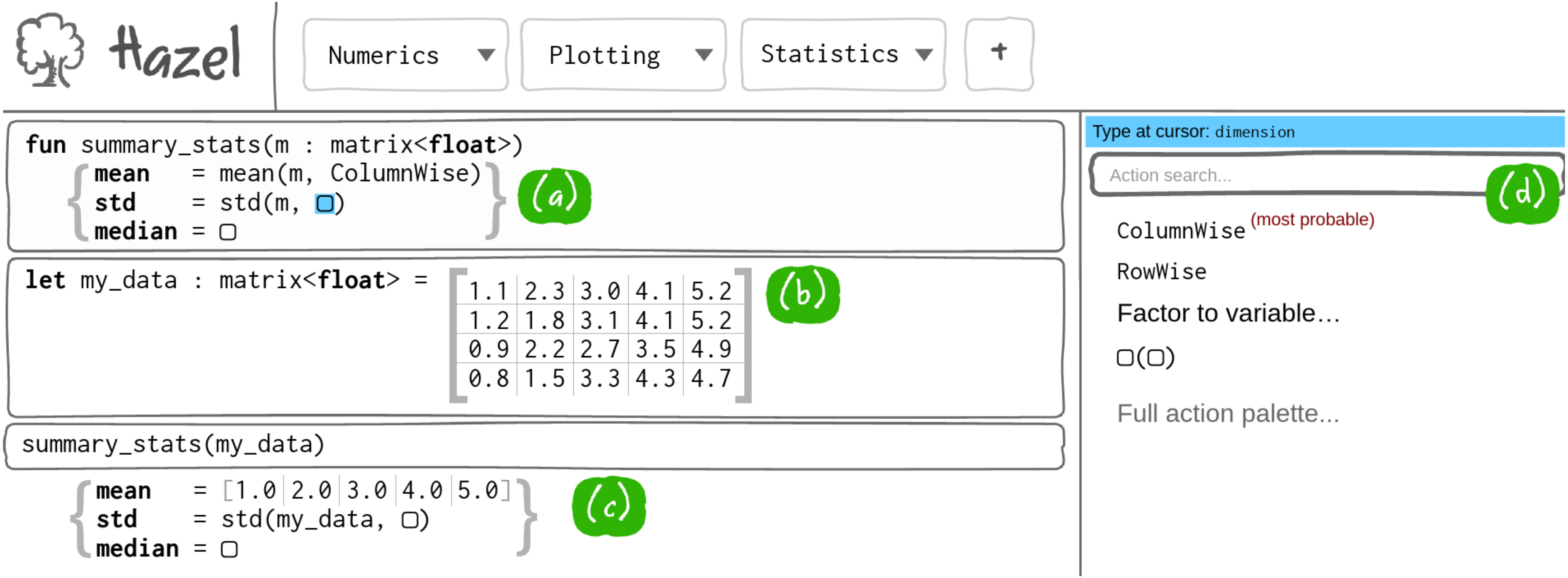
`ColumnWise` (most probable)

□(□)

Full action palette...

TODO: a statistical model of suggestions (ML for ML!)

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 = □                     }
```

Type at cursor: dimension

Action search...

ColumnWise (most probable)

RowWise

Factor to variable…

□(□)

Full action palette...

A real opportunity to apply foundational type theory and modern PL techniques to deliver a **best-in-class programming experience**.