Bidirectional Programming with Direct Manipulation

Ravi Chugh
I ❤️ GUIs
I ❤️ GUIs
I love GUIs
I love PLs

I love GUIs

Scala

OCaml
I love PLs

I love GUIs
<table>
<thead>
<tr>
<th>PLs</th>
<th>GUIs</th>
</tr>
</thead>
<tbody>
<tr>
<td>2D Graphics</td>
<td></td>
</tr>
<tr>
<td>Documents</td>
<td></td>
</tr>
<tr>
<td>Presentations</td>
<td></td>
</tr>
<tr>
<td>Spreadsheets</td>
<td></td>
</tr>
<tr>
<td>Web Apps</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
I love PLs

I love GUIs
New PL Features?

New UI Features?
PLUIs

SE

HCl
Bidirectional Evaluation for Program Repair
[PLDI 2016, OOPSLA 2018]

Live Evaluation with Program Synthesis
[POPL 2019, work-in-progress]

Lightweight Structured Text Editing
[ICSE 2018, work-in-progress]

Output-Directed Program Synthesis
[UIST 2016, work-in-progress]
Bidirectional Evaluation for Program Repair
[PLDI 2016, OOPSLA 2018]

Live Evaluation with Program Synthesis
[POPL 2019, work-in-progress]

Lightweight Structured Text Editing
[ICSE 2018, work-in-progress]

Output-Directed Program Synthesis
[UIST 2016, work-in-progress]
Bidirectional Evaluation for Program Repair
[PLDI 2016, OOPSLA 2018]

Live Evaluation with Program Synthesis
[POPL 2019, work-in-progress]

PLUIs

Lightweight Structured Text Editing
[ICSE 2018, work-in-progress]

Output-Directed Program Synthesis
[UIST 2016, work-in-progress]
Edit Program
Edit Program; Run
Edit Program; Run; View Output
(Edit Program; Run; View Output; Edit Output) +

Synthesize Program Repair
Our Goal

let f x = x + " 2018" in
Our Goal

let f x = x + " 2018" in
f "OOPS"
+ "\n@ " + f "SPLASH"
Our Goal

\[
\text{let } f \ x = x + \" 2018\" \text{ in }
\]

\[
f \ "OOPS"
+ \"\n@ \" + f \ "SPLASH"
\]

⇒

"OOPS 2018
@ SPLASH 2018"
Our Goal

\[
\text{let } f \ x = x + " \ 2018" \text{ in }
\]
\[
f "OOPS"
\]
\[
+ "\n@ " + f "SPLASH"
\]\n
⇒

"OOPS 2018
@ SPLASH 2018"

⇒

"OOPSLA 2019
@ SPLASH 2018"
Our Goal

```oca
let f x = x + " 2018" in
  f "OOPS"
+ "\n@ " + f "SPLASH"
⇒ "OOPS 2018 @ SPLASH 2018"
```

```oca
let f x = x + " 2019" in
  f "OOPSLA"
+ "\n@ " + f "SPLASH"
⇐ "OOPSLA 2019 @ SPLASH 2018"
```
Our Goal

```plaintext
let f x = x + " 2018" in
f "OOPS"
+ "\n@ " + f "SPLASH"
```

⇒

```plaintext
"OOPS 2018 @ SPLASH 2018"
```

```plaintext
let f x = x + " 2019" in
f "OOPSLA"
+ "\n@ " + f "SPLASH"
```

⇐

```plaintext
"OOPSLA 2019 @ SPLASH 2018"
```
Our Goal

let f x = x + " 2018" in f "OOPS" + "\n@ " + f "SPLASH"

⇒
"OOPS 2018 @ SPLASH 2018"

let f x = x + " 2019" in f "OOPSLA" + "\n@ " + f "SPLASH"

⇒
"OOPSLA 2019 @ SPLASH 2019"
Lenses

\[ f(x) \Rightarrow y \]

\[ f(x') \Leftarrow y' \]
1. Function $f$ in DSL or restricted style (i.e. point-free)
1. Function $f$ in DSL or restricted style (i.e. point-free)

2. Update "data" (x) but not "code" (f)
1. Function \( f \) in DSL or restricted style (i.e. point-free)

2. Update "data" \((x)\) but not "code" \((f)\)

3. Restricted structural changes between \( y \) and \( y' \)

Lenses
\[ f(x) \Rightarrow y \]
\[ f(x') \Leftrightarrow y' \]
Bidirectional Evaluation

\[ f \ x \quad \Rightarrow \quad y \]
\[ f \ x^I \quad \Leftrightarrow \quad y^I \]
Bidirectional Evaluation

All expressions (code + data) can be run backwards and updated
Bidirectional Evaluation

All expressions (code + data) can be run backwards and updated

"Small" structural changes to values + user-defined lenses for customization
Bidirectional Evaluation

$e \Rightarrow v$

$e^I \Leftrightarrow v^I$
Bidirectional Evaluation

(\textit{E} \vdash e) \Rightarrow \textit{v}

(\textit{E}' \vdash e') \leftrightarrow \textit{v}'

\text{Bidirectional Evaluation}
let f x = x + " 2018" in
f "OOPS"
+ "\n@ " + f "SPLASH"
⇒
"OOPS 2018
@ SPLASH 2018"

let f x = x + " 2019" in
f "OOPSLA"
+ "\n@ " + f "SPLASH"
⇐
"OOPSLA 2019
@ SPLASH 2018"
let f x = x + " 2018" in
f "OOPS"
 + "\n@ " + f "SPLASH"

"OOPS 2018
@ SPLASH 2018"

let f x = x + " 2019" in
f "OOPSLA"
 + "\n@ " + f "SPLASH"

"OOPSLA 2019
@ SPLASH 2018"
\[(E ⊢ e) \Rightarrow v \leftrightsquigarrow (E' ⊢ e') \Leftarrow v'\]

\[(E ⊢ e_1 e_2) \Rightarrow v \leftrightsquigarrow v'\]
$$(E \vdash e) \Rightarrow v$$

$$(E' \vdash e') \Leftarrow v'$$

$$(E... \vdash f \ x) \Rightarrow v$$

$$\Leftrightarrow v'$$
\[ ((E, f \mapsto [E_f] \lambda y. e_f, x \mapsto v_x) \vdash f \ x) \Rightarrow v \]

\[ (E \vdash e) \Rightarrow v \]

\[ (E' \vdash e') \Leftarrow v' \]
\[(E \vdash e) \Rightarrow v \]
\[(E' \vdash e') \Leftarrow v' \]

\[
\begin{array}{c}
((E, f \mapsto [E_f] \lambda y.e_f, x \mapsto v_x) \vdash f x) \Rightarrow v \\
\end{array}
\]
\[(E' \vdash e') \iff v'\]

\[
((E_f, y \mapsto v_x) \vdash e_f) \Rightarrow v
\]

\[
((E, f \mapsto [E_f] \lambda y. e_f, x \mapsto v_x) \vdash f x) \Rightarrow v
\]

\[
(E \vdash e) \Rightarrow v
\]
\[(E \vdash e) \Rightarrow v \quad \Leftrightarrow \quad (E' \vdash e') \Leftarrow v'\]

\[
((E_f, y \mapsto v_x) \vdash e_f) \Rightarrow v
\]

\[
((E, f \mapsto [E_f]) \lambda y. e_f, x \mapsto v_x) \vdash f x \Rightarrow v
\]

\[
\Leftrightarrow v'
\]
\[(E \vdash e) \Rightarrow v \]
\[(E' \vdash e') \Leftarrow v' \]

\[ ((E_f, y \mapsto v_x) \vdash e_f) \Rightarrow v \]
\[ ((E, f \mapsto [E_f] \lambda y.e_f, x \mapsto v_x) \vdash f x) \Rightarrow v \]

\[ v' \]
\[
\begin{align*}
((E_f, \ y \mapsto v_x) \vdash e_f) & \Rightarrow v \\
((E'_f, \ y \mapsto v'_x) \vdash e'_f) & \iff v' \\
((E, \ f \mapsto [E_f] \ \lambda y.e_f, \ x \mapsto v_x) \vdash f \ x) & \Rightarrow v \\
& \iff v'
\end{align*}
\]
\[(\mathcal{E} \vdash e) \Rightarrow \nu \]
\[(\mathcal{E}' \vdash e') \Leftarrow \nu' \]

\[((E_f, y \mapsto v_x) \vdash e_f) \Rightarrow \nu\]
\[((E_f', y \mapsto v_x') \vdash e_f') \Leftarrow \nu'\]

\[((E, f \mapsto [E_f] \lambda y. e_f, x \mapsto v_x) \vdash f x) \Rightarrow \nu\]
\[((E, f \mapsto , x \mapsto ) \vdash f x) \Leftarrow \nu'\]
\[(E \vdash e) \Rightarrow v \quad \overset{\top}{\longleftrightarrow} \quad (E' \vdash e') \Leftarrow v'\]

\[\begin{align*}
((E_f, y \mapsto v_x) \vdash e_f) & \Rightarrow v \\
((E'_f, y \mapsto v'_x) \vdash e'_f) & \Leftarrow v'
\end{align*}\]

\[\begin{align*}
((E, f \mapsto [E_f] \lambda y.e_f, x \mapsto v_x) \vdash f x) & \Rightarrow v \\
((E, f \mapsto [E'_f] \lambda y.e'_f, x \mapsto \quad ) \vdash f x) & \Leftarrow v'
\end{align*}\]

New Code
\[(E ⊢ e) \Rightarrow v \quad \Leftrightarrow \quad (E' ⊢ e') \Leftarrow v'\]

\[
((E_f, y \mapsto v_x) ⊢ e_f) \Rightarrow v
\]

\[
((E'_f, y \mapsto v'_x) ⊢ e'_f) \Leftarrow v'
\]

\[
((E, f \mapsto [E_f] \lambda y.e_f, x \mapsto v_x) ⊢ f x) \Rightarrow v
\]

\[
((E, f \mapsto [E'_f] \lambda y.e'_f, x \mapsto v'_x) ⊢ f x) \Leftarrow v'
\]

New Code  New Data
let f x = x + " 2018" in
f "OOPS"
+ "\n@ " + f "SPLASH"
⇒
"OOPS 2018
@ SPLASH 2018"

let f x = x + " 2019" in
f "OOPSLA"
+ "\n@ " + f "SPLASH"
⇐
"OOPSLA 2019
@ SPLASH 2018"
Merge: $E_1 \oplus_{\text{conservative}} E_2$

(year $\mapsto$ 2018) $\oplus_{\text{conservative}}$ (year $\mapsto$ 2019) undefined

Enables a "PutGet" theorem
Merge: $E_1 \oplus_{\text{conservative}} E_2$

$(\text{year} \mapsto 2018) \oplus_{\text{conservative}} (\text{year} \mapsto 2019)$ undefined

Enables a "PutGet" theorem

Merge: $E_1 \oplus_{\text{optimistic}} E_2$

$(\text{year} \mapsto 2018) \oplus_{\text{optimistic}} (\text{year} \mapsto 2019) = 2019$

Enables many desirable interactions
let f x = x + " 2018" in
f "OOPS"
+ "\n@ " + f "SPLASH"

⇒

"OOPS 2018
@ SPLASH 2018"

let f x = x + " 2019" in
f "OOPSLA"
+ "\n@ " + f "SPLASH"

⇐

"OOPSLA 2019
@ SPLASH 2018"
let \( f \ x = x + " \ 2018" \) in
\[ f "OOPS" \\
+ "\\n@ " + f "SPLASH" \]

\[
\Rightarrow\\n\Rightarrow\\n\left( E \vdash e \right) \Rightarrow \nu\\n\left( E' \vdash e' \right) \Leftarrow \nu'\\n\Rightarrow\\n\Rightarrow\\n\Rightarrow
\]

let \( f \ x = x + " \ 2019" \) in
\[ f "OOPSLA" \\
+ "\\n@ " + f "SPLASH" \]

\[
\Leftarrow\\n\Leftarrow\\n"OOPSLA 2019 @ SPLASH 2018"\\n\Rightarrow
\]

\[
"OOPS 2018 @ SPLASH 2018"\\n\Rightarrow
\]

56
let \( f \) \( x \) = \( x + "\ 2018" \) in
\( f "\text{OOPS}" + "\n@ " + f "\text{SPLASH}" \)

\[
\begin{align*}
\text{let } f \ x &= \ x + "\ 2018" \ 	ext{in} \\
& f "\text{OOPS}" + "\n@ " + f "\text{SPLASH}" \\
\end{align*}
\]

\[
\begin{align*}
\text{let } f \ x &= \ x + "\ 2019" \ 	ext{in} \\
& f "\text{OOPSLA}" + "\n@ " + f "\text{SPLASH}" \\
\end{align*}
\]
Propagate updates \textit{through} function calls

\textit{Allow} merge conflicts
"Bidirectional Evaluation..."

\[(E \vdash e) \Rightarrow v\]

\[(E' \vdash e') \Leftarrow v'\]
"Bidirectional Evaluation... with Direct Manipulation"

\[(E \vdash e) \Rightarrow v \rightleftharpoons v^i\]

\[(E' \vdash e^i) \Leftarrow v^i\]
let headerRow =
let styles = [padding] in
Html.tr  ()  ()  (List.map (Html.th styles [])) headers

let stateRows =
let colors = ["yellow", "white"] in
drawRow i row =
let color = List.nth colors (mod i colors)
let columns =
List.map
(Html.td [padding, ["background-color", color]])
\[(E \vdash e) \Rightarrow (E' \vdash e')\]
$\text{Sketch-n-Sketch}$

$\text{Elm-like PL +}$

$(E + e) \Rightarrow v \Rightarrow (E' + e') \Leftarrow v'$
(E' + e') \Rightarrow v' \Rightarrow \text{Elm-like PL + HTML GUI}
Sketch-n-Sketch

Demo

Elm-like PL + HTML GUI

\[(E + e) \Rightarrow v \Rightarrow (E' + e') \Rightarrow \]

(Alabama Montgomery, AL)
(Arizona Phoenix, AZ)
(Colorado Denver, CO)
(Connecticut Hartford, CT)
(South Dakota Pierre, SD)
(Alaska Juneau, AK)
(Arkansas Little Rock, AR)
(California Sacramento, CA)
(Florida Tallahassee, FL)
(Georgia Atlanta, GA)
(Hawaii Honolulu, HI)
(Indiana Indianapolis, IN)
(Illinois Springfield, IL)
(Alabama Montgomery, AL)
(Alaska Juneau, AK)
(Arkansas Little Rock, AR)
(California Sacramento, CA)
(Florida Tallahassee, FL)
(Georgia Atlanta, GA)
(Hawaii Honolulu, HI)
(Indiana Indianapolis, IN)
(Illinois Springfield, IL)
(Connecticut Hartford, CT)
(South Dakota Pierre, SD)
(Alaska Juneau, AK)
(Arkansas Little Rock, AR)
(California Sacramento, CA)
(Florida Tallahassee, FL)
(Georgia Atlanta, GA)
(Hawaii Honolulu, HI)
(Indiana Indianapolis, IN)
(Illinois Springfield, IL)
(Connecticut Hartford, CT)
(South Dakota Pierre, SD)
# Bidirectional Evaluation

## Future Work

<table>
<thead>
<tr>
<th>Repair Algorithm</th>
<th>User Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tracing plus bidirectional evaluation</td>
<td>Interaction to resolve ambiguities</td>
</tr>
<tr>
<td>Version space algebra of diffs</td>
<td>Tools for distributed web development</td>
</tr>
<tr>
<td>Synthesis of larger structural repairs</td>
<td>Other source languages and target domains</td>
</tr>
</tbody>
</table>
Bidirectional Evaluation for Program Repair
[PLDI 2016, OOPSLA 2018]

Live Evaluation with Program Synthesis
[POPL 2019, work-in-progress]

Lightweight Structured Text Editing
[ICSE 2018, work-in-progress]

Output-Directed Program Synthesis
[UIST 2016, work-in-progress]
Bidirectional Evaluation for Program Repair
[PLDI 2016, OOPSLA 2018]

Live Evaluation with Program Synthesis
[POPL 2019, work-in-progress]

PLUIs

Lightweight Structured Text Editing
[ICSE 2018, work-in-progress]

Output-Directed Program Synthesis
[UIST 2016, work-in-progress]
Semantic Foundations for Hole-Driven Development

[Omar et al. POPL 2017, POPL 2019, wip]
Semantic Foundations for Hole-Driven Development

[Omar et al. POPL 2017, POPL 2019, wip]
Bidirectional Evaluation for Program Repair
[PLDI 2016, OOPSLA 2018]

Live Evaluation with Program Synthesis
[POPL 2019, work-in-progress]

Lightweight Structured Text Editing
[ICSE 2018, work-in-progress]

Output-Directed Program Synthesis
[UIST 2016, work-in-progress]
Bidirectional Evaluation for Program Repair
[PLDI 2016, OOPSLA 2018]

Output-Directed Program Synthesis
[POPL 2019, work-in-progress]

Live Evaluation with Program Synthesis

Lightweight Structured Text Editing
[ICSE 2018, work-in-progress]

Output-Directed Program Synthesis
[UIST 2016, work-in-progress]
Prototype... Repair... Refactor...
Programming with:
Less Keyboard.
More Mouse.
Programming with:
Less Keyboard.
More Mouse.

Idea
Prototype... Repair... Refactor...
Output-Directed Programming

Demo
Output-Directed Programming

Future Work

More control over generated code

More relationship / abstraction patterns

Visualization / manipulation of intermediates

DSLs for defining new transformations
PLs + GUIs

General Purpose PL + Domain Specific UI
General Purpose Program Synthesis + Domain Specific Program Synthesis
Bidirectional Evaluation for Program Repair
[PLDI 2016, OOPSLA 2018]

Live Evaluation with Program Synthesis
[POPL 2019, work-in-progress]

Lightweight Structured Text Editing
[ICSE 2018, work-in-progress]

Output-Directed Program Synthesis
[UIST 2016, work-in-progress]

PLUIs
Bidirectional Evaluation for Program Repair
[PLDI 2016, OOPSLA 2018]

Live Evaluation with Program Synthesis
[POPL 2019, work-in-progress]

Lightweight Structured Text Editing
[ICSE 2018, work-in-progress]

Output-Directed Program Synthesis
[UIST 2016, work-in-progress]
Show AST on Code

- Structure Editors
- Refactoring Tools
- Text Editors
- BEST OF BOTH?
Show AST on Code
Show AST on Code
(func arg)
Show AST on Code

Structural (Multi-)Select
Context-Sensitive Transforms

\((\text{func}\ arg)\)
Structured Text Editing

Demo
Structured Text Editing versus "Traditional" Refactoring

21 users
Traditional ("Text-Select") Mode

Text Select

(def image1
  (let [width height]
    (let [x y] [100 100]
      (image "lightgrey")
    )
  )

Right-Click Menu

Select Arguments

Defaults

Deuce ("Box-Select") Mode

Structure Select

(def image1
  (let [width height]
    (let [x y] [100 100]
      (image "lightgrey")
    )
  )

Short Menu

Code Tools

- Move Definition
- Duplicate Definition

Defaults

Move width and height

Move Definition

Requirements
- Select one or more variable definitions and one target position (i.e. whitespace) (Satisfied)

Code Updates

Move width and height
1. Tutorial

2. **Head-to-Head** Tasks (2x each; once per mode)

3. **Mix-and-Match** Tasks (free to use both modes)

4. Exit Survey
Deuce more effective than Traditional?

Deuce doesn't help discoverability
Deuce more effective than Traditional?

Deuce may be faster once learned
Deuce preferred to Traditional?

Survey

Modest subjective preference for Deuce

Observed

Almost everyone used Deuce more
**Hypothesis:**
Deuce better for multi-arg transforms
Deuce versus Traditional

Traditional may be better for discovery

Deuce may be faster once learned

Deuce strongly preferred
Structured Text Editing

Future Work

UI concerns for larger programs

How to encourage structured transforms?

DSLs for defining new transformations

DSLs for defining custom, type-specific UIs

Real languages in real editors
Bidirectional Evaluation for Program Repair
[PLDI 2016, OOPSLA 2018]

Live Evaluation with Program Synthesis
[POPL 2019, work-in-progress]

Output-Directed Program Synthesis
[UIST 2016, work-in-progress]

Lightweight Structured Text Editing
[ICSE 2018, work-in-progress]
Bidirectional Evaluation for Program Repair
[PLDI 2016, OOPSLA 2018]

Live Evaluation with Program Synthesis
[POPL 2019, work-in-progress]

Lightweight Structured Text Editing
[ICSE 2018, work-in-progress]

Output-Directed Program Synthesis
[UIST 2016, work-in-progress]
PLs + GUIs

General Purpose PL
+ General Purpose Program Synthesis

Domain Specific UI
+ Domain Specific Program Synthesis

General Purpose Programming Environments
Bidirectional Evaluation for Program Repair
[PLDI 2016, OOPSLA 2018]

Live Evaluation with Program Synthesis
[POPL 2019, work-in-progress]

Sketch-n-Sketch
http://ravichugh.github.io/sketch-n-sketch/

Lightweight Structured Text Editing
[ICSE 2018, work-in-progress]

Output-Directed Program Synthesis
[UIST 2016, work-in-progress]
\[
\begin{align*}
e & ::= c \mid \lambda x.e \mid x \mid (e_1, e_2) \mid e_1 e_2 \\
v & ::= c \mid [E] \lambda x.e \mid (v_1, v_2) \\
E & ::= \_ \mid (E, x \mapsto v) \\
\end{align*}
\]

\[
\begin{align*}
(E \vdash e) & \Rightarrow v \\
(E^I \vdash e^I) & \iff v^I
\end{align*}
\]
e ::= c | \( \lambda x.e \) | x | (e_1, e_2) | e_1 e_2

v ::= c | [E] \( \lambda x.e \) | (v_1, v_2)

E ::= – | (E, x \mapsto v)

(\( E \vdash e \) \Rightarrow v)

(\( E' \vdash e' \) \Leftarrow v')
\[
e ::= c \mid \lambda x. e \mid x \mid (e_1, e_2) \mid e_1 e_2 \\
v ::= c \mid [E] \lambda x. e \mid (v_1, v_2) \\
E ::= \bot \mid (E, x \mapsto v)
\]

\[
(E \vdash e) \Rightarrow v \\
(E' \vdash e') \iff v'
\]
\[ e ::= c \mid \lambda x.e \mid x \mid (e_1, e_2) \mid e_1 \, e_2 \]

\[ v ::= c \mid [E] \lambda x.e \mid (v_1, v_2) \]

\[ E ::= \_ \mid (E, x \mapsto v) \]

\[ (E \vdash e) \Rightarrow v \quad \Downarrow \]

\[ (E' \vdash e') \Leftarrow v' \]
\[
\begin{align*}
e & ::= \mathbf{c} \mid \lambda x. e \mid x \mid (e_1, e_2) \mid e_1 e_2 \\
v & ::= \mathbf{c} \mid [E] \lambda x. e \mid (v_1, v_2) \\
E & ::= \_ \mid (E, x \mapsto v)
\end{align*}
\]

\[
\begin{align*}
(E \vdash e) & \Rightarrow v \\
(E' \vdash e') & \Leftrightarrow v'
\end{align*}
\]
\[ e ::= c \mid \lambda x.e \mid x \mid (e_1, e_2) \mid e_1 e_2 \]

\[ v ::= c \mid [E] \lambda x.e \mid (v_1, v_2) \]

\[ E ::= - \mid (E, x \mapsto v) \]

\[(E \vdash e) \Rightarrow v \]

\[(E' \vdash e') \Leftarrow v' \]
\[
\begin{align*}
e & ::= c \mid \lambda x.e \mid x \mid (e_1, e_2) \mid e_1 e_2 \\
v & ::= c \mid [E] \lambda x.e \mid (v_1, v_2) \\
E & ::= \epsilon \mid (E, x \mapsto v)
\end{align*}
\]
\[ e ::= c \mid \lambda x.e \mid x \mid (e_1, e_2) \mid e_1 e_2 \]

\[ v ::= c \mid [E] \lambda x.e \mid (v_1, v_2) \]

\[ E ::= \_ \mid (E, x \mapsto v) \]

\[ ([E] \lambda x.e) \vdash x \Rightarrow v \]

\[ (E \vdash e) \Rightarrow v' \]

\[ (E' \vdash e') \iff v' \]
\[ \begin{align*}
e & ::= c \mid \lambda x.e \mid x \mid (e_1, e_2) \mid e_1 e_2 \\
v & ::= c \mid [E] \lambda x.e \mid (v_1, v_2) \\
E & ::= \bot \mid (E, x \mapsto v) \\
\end{align*} \]

\[ (E \vdash e) \Rightarrow v \]

\[ (E' \vdash e') \Leftrightarrow v' \]

\[ (E \vdash (e_1, e_2)) \Rightarrow (v_1, v_2) \]

\[ (v_1', v_2') \]

\[ (E', e') \Leftrightarrow (v_1', v_2') \]
\[ e ::= c \mid \lambda x.e \mid x \mid (e_1, e_2) \mid e_1 e_2 \]

\[ v ::= c \mid [E] \lambda x.e \mid (v_1, v_2) \]

\[ E ::= \_ \mid (E, x \mapsto v) \]

\[
\begin{align*}
(E \vdash e_1) & \Rightarrow v_1 \\
(E'_1 \vdash e'_1) & \iff v'_1
\end{align*}
\]

\[
\begin{align*}
(E \vdash (e_1, e_2)) & \Rightarrow (v_1, v_2) \\
(v_1', v_2') & \Rightarrow (v_1', v_2')
\end{align*}
\]
\[ e ::= c \mid \lambda x.e \mid x \mid (e_1, e_2) \mid e_1 e_2 \]
\[ v ::= c \mid [E] \lambda x.e \mid (v_1, v_2) \]
\[ E ::= \_ \mid (E, x \mapsto v) \]

- \((E \vdash e) \Rightarrow v\)
- \((E' \vdash e') \Leftarrow v'\)

\[(E \vdash e_1) \Rightarrow v_1 \]
\[(E_1' \vdash e_1') \Leftarrow v_1' \]
\[(E \vdash e_2) \Rightarrow v_2 \]
\[(E_2' \vdash e_2') \Leftarrow v_2' \]

\[(E \vdash (e_1, e_2)) \Rightarrow (v_1, v_2) \]
\[(v_1', v_2') \Rightarrow (v_1', v_2')\]
\[e ::= c \mid \lambda x.e \mid x \mid (e_1, e_2) \mid e_1 e_2\]

\[v ::= c \mid [E] \lambda x.e \mid (v_1, v_2)\]

\[E ::= \_ \mid (E, x \mapsto v)\]

\[(E \vdash e) \Rightarrow v\]

\[(E' \vdash e') \Leftarrow v'\]

\[
\begin{align*}
(E \vdash e_1) & \Rightarrow v_1 \quad (E \vdash e_2) \Rightarrow v_2 \\
(E_1' \vdash e_1') & \Leftarrow v_1' \\
(E_2' \vdash e_2') & \Leftarrow v_2'
\end{align*}
\]

\[
\begin{align*}
(E \vdash (e_1, e_2)) & \Rightarrow (v_1, v_2) \\
(E_1' \oplus E_2' \vdash (e_1', e_2')) & \Leftarrow (v_1', v_2')
\end{align*}
\]
\[ e ::= c \mid \lambda x.e \mid x \mid (e_1, e_2) \mid e_1 e_2 \]

\[ v ::= c \mid [E] \lambda x.e \mid (v_1, v_2) \]

\[ E ::= \_ \mid (E, x \mapsto v) \]
\[ e ::= c \mid \lambda x. e \mid x \mid (e_1, e_2) \mid e_1 e_2 \]

\[ v ::= c \mid [E] \lambda x. e \mid (v_1, v_2) \]

\[ E ::= \_ \mid (E, x \mapsto v) \]

\[ (E \triangleright e) \Rightarrow \triangleright v \]

\[ (E' \triangleright e') \Leftarrow \triangleright v' \]

\[ ((x \mapsto 1) \triangleright (x, x)) \Rightarrow (1, 1) \]

\[ (2, 3) \]
\[ e ::= c \mid \lambda x.e \mid x \mid (e_1, e_2) \mid e_1 e_2 \]
\[ v ::= c \mid [E] \lambda x.e \mid (v_1, v_2) \]
\[ E ::= \_ \mid (E, x \mapsto v) \]

\[
\begin{array}{c}
((x \mapsto 1) \vdash x) \Rightarrow 1 \\
((x \mapsto 2) \vdash x) \Leftarrow 2 \\
((x \mapsto 3) \vdash x) \Leftarrow 3 \\
((x \mapsto 1) \vdash (x, x)) \Rightarrow (1, 1) \\
((x \mapsto 1) \vdash (x, x)) \Rightarrow (2, 3)
\end{array}
\]
\[
e ::= c \mid \lambda x.e \mid x \mid (e_1, e_2) \mid e_1 e_2
\]

\[
v ::= c \mid [E] \lambda x.e \mid (v_1, v_2)
\]

\[
E ::= \bot \mid (E, x \mapsto v)
\]
\[ \begin{align*}
e & ::= c \mid \lambda x.e \mid x \mid (e_1, e_2) \mid e_1 e_2 \\
v & ::= c \mid [E] \lambda x.e \mid (v_1, v_2) \\
E & ::= \_ \mid (E, x \mapsto v)
\end{align*} \]

**Merge:** \[ E_1 \oplus_{\text{conservative}} E_2 \]

\[ (x \mapsto 2) \oplus_{\text{conservative}} (x \mapsto 3) \text{ undefined} \]

Enables a "PutGet" theorem
\[ e ::= c \mid \lambda x. e \mid x \mid (e_1, e_2) \mid e_1 e_2 \]
\[ v ::= c \mid [E] \lambda x. e \mid (v_1, v_2) \]
\[ E ::= \_ \mid (E, x \mapsto v) \]

**Merge:** \( E_1 \oplus_{\text{conservative}} E_2 \)

\((x \mapsto 2) \oplus_{\text{conservative}} (x \mapsto 3)\) undefined

Enables a "PutGet" theorem

**Merge:** \( E_1 \oplus_{\text{optimistic}} E_2 \)

\((x \mapsto 2) \oplus_{\text{optimistic}} (x \mapsto 3) = 2 \text{ or } 3 \) (not 1)

Enables many desirable interactions
\[ e ::= \mathit{c} \mid \lambda x.e \mid x \mid (e_1, e_2) \mid e_1 \ e_2 \]
\[ v ::= \mathit{c} \mid \lfloor E \rfloor \lambda x.e \mid (v_1, v_2) \]
\[ E ::= \varepsilon \mid (E, x \mapsto v) \]

\[ (E \vdash e) \Rightarrow v \]
\[ \varepsilon \]
\[ (E' \vdash e') \Leftrightarrow v' \]
\[
e ::= c \mid \lambda x.e \mid x \mid (e_1, e_2) \mid e_1 e_2
\]

\[
v ::= c \mid \text{[E]} \lambda x.e \mid (v_1, v_2)
\]

\[
E ::= \_ \mid (E, x \mapsto v)
\]

\[
(E \vdash e) \Rightarrow v
\]

\[
(E' \vdash e') \leftrightarrow v'
\]

\[
(E \ldots \vdash f x) \Rightarrow v
\]

\[
(E \ldots \vdash f x) \Rightarrow v
\]

\[
(E \ldots \vdash f x) \Rightarrow v
\]

\[
(E \ldots \vdash f x) \Rightarrow v
\]
\[
\begin{align*}
e & ::= c \mid \lambda x. e \mid x \mid (e_1, e_2) \mid e_1 e_2 \\
v & ::= c \mid [E] \lambda x. e \mid (v_1, v_2) \\
E & ::= - \mid (E, x \mapsto v)
\end{align*}
\]

\[
((E, f \mapsto [E_f] \lambda y. e_f, x \mapsto v_x) \vdash f x) \Rightarrow v \quad \Downarrow \quad v'
\]
\[ e ::= c \mid \lambda x.e \mid x \mid (e_1, e_2) \mid e_1 e_2 \]

\[ v ::= c \mid \left[ E \right] \lambda x.e \mid (v_1, v_2) \]

\[ E ::= - \mid (E, x \mapsto v) \]

\[ (E \vdash e) \Rightarrow v \]

\[ (E' \vdash e') \Leftarrow v' \]
\[ e ::= c \mid \lambda x.e \mid x \mid (e_1, e_2) \mid e_1 e_2 \]

\[ v ::= c \mid [E] \lambda x.e \mid (v_1, v_2) \]

\[ E ::= \_ \mid (E, x \mapsto v) \]

\[ (E \vdash e) \Rightarrow v \]

\[ (E' \vdash e') \iff v' \]
\[
\begin{aligned}
e &::= c \mid \lambda x. e \mid x \mid (e_1, e_2) \mid e_1 e_2 \\
v &::= c \mid \llbracket E \rrbracket \lambda x. e \mid (v_1, v_2) \\
E &::= \epsilon \mid (E, x \mapsto v)
\end{aligned}
\]

\[
\begin{aligned}
((E_f, y \mapsto v_x) \vdash e_f) \Rightarrow v
\end{aligned}
\]

\[
\begin{aligned}
((E, f \mapsto \llbracket E_f \rrbracket) \lambda y. e_f, x \mapsto v_x) \vdash f x) \Rightarrow v
\end{aligned}
\]
\[
\begin{align*}
e &::= c \mid \lambda x.e \mid x \mid (e_1, e_2) \mid e_1 e_2 \\
v &::= c \mid [E] \lambda x.e \mid (v_1, v_2) \\
E &::= - \mid (E, x \mapsto v)
\end{align*}
\]
\( e ::= c \mid \lambda x. e \mid x \mid (e_1, e_2) \mid e_1 e_2 \)

\( v ::= c \mid [E] \lambda x. e \mid (v_1, v_2) \)

\( E ::= \_ \mid (E, x \mapsto v) \)

\[ ((E_f, y \mapsto v_x) \vdash e_f) \Rightarrow v \]

\[ ((E'_f, y \mapsto v'_x) \vdash e'_f) \Leftarrow v' \]

\[ ((E, f \mapsto [E_f] \lambda y. e_f, x \mapsto v_x) \vdash f x) \Rightarrow v \]

\[ ((E, f \mapsto \_ , x \mapsto \_) \vdash f x) \Leftarrow v' \]
\[ e ::= c \mid \lambda x. e \mid x \mid (e_1, e_2) \mid e_1 e_2 \]

\[ v ::= c \mid [E] \lambda x. e \mid (v_1, v_2) \]

\[ E ::= \_ \mid (E, x \mapsto v) \]

\[
\begin{align*}
((E_f, y \mapsto v_x) \vdash e_f) & \Rightarrow v \\
((E'_f, y \mapsto v'_x) \vdash e'_f) & \Leftarrow v'
\end{align*}
\]

\[
\begin{align*}
((E, f \mapsto [E_f] \lambda y. e_f, x \mapsto v_x) \vdash f x) & \Rightarrow v \\
((E, f \mapsto [E'_f] \lambda y. e'_f, x \mapsto v'_x) \vdash f x) & \Leftarrow v'
\end{align*}
\]

New Code
\[ e ::= c \mid \lambda x. e \mid x \mid (e_1, e_2) \mid e_1 \cdot e_2 \]
\[ v ::= c \mid [E] \lambda x. e \mid (v_1, v_2) \]
\[ E ::= \_ \mid (E, x \mapsto v) \]

\[(E \vdash e) \Rightarrow v \]
\[ (E' \vdash e') \Leftarrow v' \]
Key Idea

((E_f, y \mapsto v_x) \vdash e_f) \Rightarrow v

((E'_f, y \mapsto v'_x) \vdash e'_f) \Leftarrow v'

New Code

New Data
\[
\begin{align*}
  e & ::= \texttt{c} \mid \lambda x. e \mid x \mid (e_1, e_2) \mid e_1 \ e_2 \\
(\mathcal{E} \vdash e) & \Rightarrow \nu \\
(\mathcal{E'} \vdash e') & \Leftrightarrow \nu'
\end{align*}
\]
e ::= c | λx.e | x | (e₁, e₂) | e₁ e₂
| e₁ ::= e₂ | [e₁, … , eₙ] | {f₁ = e₁, … , fₙ = eₙ}
| let x = e₁ in e₂ |
| if e₁ then e₂ else e₃ |
| e₁ + e₂ | ¬e | … |
| dict_get e₁ e₂ | … |
| re_match e₁ e₂ | … |
| eval e |
\( e ::= c \mid \lambda x.e \mid x \mid (e_1, e_2) \mid e_1 e_2 \mid \ldots \)
\[ e ::= c \mid \lambda x. e \mid x \mid (e_1, e_2) \mid e_1 \ e_2 \]

- \[ e_1 :: e_2 \mid [e_1, \ldots, e_n] \]
- \( \{f_1 = e_1, \ldots, f_n = e_n\} \)
- let \( x = e_1 \) in \( e_2 \)
- if \( e_1 \) then \( e_2 \) else \( e_3 \)
- \( e_1 + e_2 \mid \neg e \mid \ldots \)
- dict_get \( e_1, e_2 \)
- re_match \( e_1, e_2 \)
- eval \( e \)

\[
(E \vdash e) \Rightarrow v \\
\Downarrow \\
(E' \vdash e') \leftarrow v'
\]

Structural updates through, e.g.,

List.map \( f' \) \( xs' \) \(\leftarrow vs' \)?
\[ e ::= \quad c \quad | \quad \lambda x.e \quad | \quad x \quad | \quad (e_1, e_2) \quad | \quad e_1 e_2 \]

\[ e_1 :: e_2 \quad | \quad [ e_1, \ldots, e_n ] \]

\[ \{ f_1 = e_1, \ldots, f_n = e_n \} \]

let \( x = e_1 \) in \( e_2 \)

if \( e_1 \) then \( e_2 \) else \( e_3 \)

e_1 + e_2 \quad | \quad \neg e \]

dict_get e_1 e_2

re_match e_1 e_2

eval e

\( (E \vdash e) \Rightarrow v \)

\( \Downarrow \)

\( (E' \vdash e') \Leftarrow v' \)

Structural updates through, e.g.,

\textbf{List.map \( f' \) \( xs' \) \Leftarrow vs' ?}

Domain-specific primitives?
\( e ::= c \mid \lambda x.e \mid x \mid (e_1, e_2) \mid e_1 e_2 \)

\( e_1 ::= e_2 \mid [e_1, \ldots, e_n] \)

\{ \( f_1 = e_1, \ldots, f_n = e_n \) \}

let \( x = e_1 \) in \( e_2 \)

if \( e_1 \) then \( e_2 \) else \( e_3 \)

\( e_1 + e_2 \mid \neg e \mid \ldots \)

dict_get \( e_1 e_2 \mid \ldots \)

dict_match \( e_1 e_2 \mid \ldots \)

eval e

\( \text{appLens} (e_{\text{get}}, e_{\text{put}}) \) \( e_x \)

(\( E \vdash e \) \( \Rightarrow v \) \( \Downarrow \)

(\( E' \vdash e' \) \( \Leftarrow v' \)

Structural updates through, e.g.,

List.map \( f' \) \( xs' \) \( \Leftarrow vs' \) ?

Domain-specific primitives?

User-defined lenses for customization
\[
\begin{align*}
e & ::= \ldots \mid \text{appLens} \ (e_{\text{get}}, e_{\text{put}}) \ e_x \\
(E \vdash e) & \Rightarrow v \\
(E' \vdash e') & \iff v'
\end{align*}
\]
\[ e ::= \ldots \mid \text{appLens} \left( e_{\text{get}}, e_{\text{put}} \right) e_x \]

\[
\begin{align*}
\vdash e & \Rightarrow v \\
E' \vdash e' & \iff v'
\end{align*}
\]

\[
\begin{align*}
((E, x \mapsto v_x) \vdash \text{appLens} \left( f_{\text{get}}, f_{\text{put}} \right) x) & \Rightarrow 
\end{align*}
\]
\[ e ::= \ldots \mid \text{appLens } (e_{\text{get}}, e_{\text{put}}) \, e_x \]

\[
\begin{align*}
(E \vdash e) & \Rightarrow \nu \\
(E' \vdash e') & \Leftrightarrow \nu'
\end{align*}
\]

\[
\begin{align*}
(E \vdash f_{\text{get}} \, v_x) & \Rightarrow \\
((E, x \mapsto v_x) \vdash \text{appLens } (f_{\text{get}}, f_{\text{put}}) \, x) & \Rightarrow
\end{align*}
\]
\[ e ::= \ldots \mid \text{appLens}\ (e_{\text{get}}, e_{\text{put}})\ e_x \]

\[(E \vdash e) \Rightarrow v \]
\[(E' \vdash e') \Leftarrow v' \]

\[(E \vdash f_{\text{get}}\ v_x) \Rightarrow v_y \]

\[(E, x \mapsto v_x) \vdash \text{appLens}\ (f_{\text{get}}, f_{\text{put}})\ x \Rightarrow v_y \]
\[ e ::= \ldots \mid \text{appLens} (e_{\text{get}}, e_{\text{put}}) \ e_x \]

\[ (E \vdash e) \Rightarrow v \]

\[ (E' \vdash e') \Leftarrow v' \]

\[ (E \vdash f_{\text{get}} \ v_x) \Rightarrow v_y \]

\[ ((E, x \mapsto v_x) \vdash \text{appLens} (f_{\text{get}}, f_{\text{put}}) \ x) \Rightarrow v_y \]

\[ v_y \]

\[ v'_y \]
\[ \text{e ::= ... | appLens (e_{\text{get}}, e_{\text{put}}) e_x} \]

\[ (E \vdash e) \Rightarrow v \]
\[ (E' \vdash e') \Leftarrow v' \]

\[ (E \vdash f_{\text{get}} v_x) \Rightarrow v_y \]
\[ (E \vdash f_{\text{put}} v_x v_y v_y') \Rightarrow \]

\[ ((E, x \mapsto v_x) \vdash \text{appLens} (f_{\text{get}}, f_{\text{put}}) x) \Rightarrow v_y \]
\[ v_y \]
\[ v_y' \]
\( e ::= \ldots \mid \text{appLens} (e_{\text{get}}, e_{\text{put}}) e_x \)
\[ e ::= \ldots \mid \text{appLens} \ (e_{\text{get}}, e_{\text{put}}) \ e_x \]

\[(E \vdash e) \Rightarrow v \]
\[(E' \vdash e') \leftarrow v' \]

\[(E \vdash f_{\text{get}} \ v_x) \Rightarrow v_y \]

\[(E \vdash f_{\text{put}} \ v_x \ v_y \ v_y') \Rightarrow v_x' \]

\[(E, x \to v_x) \vdash \text{appLens} \ (f_{\text{get}}, f_{\text{put}}) \ x \Rightarrow v_y \]

\[(E, x \to v_x') \vdash \text{appLens} \ (f_{\text{get}}, f_{\text{put}}) \ x \leftarrow v_y' \]
\( e ::= \ldots \mid \text{appLens} \ (e_{\text{get}}, e_{\text{put}} \ e_x) \) 

\[
\begin{align*}
(E \vdash f_{\text{get}} \ v_x) & \Rightarrow v_y \\
(E \vdash f_{\text{put}} \ v_x \ v_y \ v_y') & \Rightarrow v_x'
\end{align*}
\]

\[
\begin{align*}
((E, x \mapsto v_x) \vdash \text{appLens} \ (f_{\text{get}}, f_{\text{put}}) \ x) & \Rightarrow v_y \\
((E, x \mapsto v_x') \vdash \text{appLens} \ (f_{\text{get}}, f_{\text{put}}) \ x) & \leftarrow v_y'
\end{align*}
\]
Bidirectional Programming

**Lenses** [Foster et al. 2007, many others]

- DSLs for relational data, trees, strings, graphs
- Point-free programming style

**HOBiT** [Matsuda and Wang 2018]

- Instead of lens combinators, staged evaluator for $\lambda$-calculus
- First evaluation stage eliminates function calls $(e_1, e_2)$
- Bidirectional evaluation for "residual" first-order programs
- More direct support for "branch switching" than our approach
### Related Work

<table>
<thead>
<tr>
<th></th>
<th>[Chugh et al. PLDI 2016]</th>
<th>[OOPSLA 2018]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Repair</strong></td>
<td>Evaluation with tracing + constraint solving</td>
<td>Re-run evaluation &quot;in reverse&quot; only <strong>as needed</strong></td>
</tr>
<tr>
<td><strong>Algorithm</strong></td>
<td>[Chugh et al. PLDI 2016]</td>
<td>[OOPSLA 2018]</td>
</tr>
<tr>
<td><strong>User Interface</strong></td>
<td>Little (\lambda)-calculus (\Rightarrow) SVG</td>
<td><strong>Elm</strong>-like language (\Rightarrow) <strong>HTML</strong></td>
</tr>
<tr>
<td></td>
<td>Auto disambiguation via simple heuristics</td>
<td><strong>User disambiguation</strong> via dropdown menu</td>
</tr>
</tbody>
</table>

[Chugh et al. PLDI 2016]

[OOPSLA 2018]

**Lens API for customization**

**User disambiguation** via dropdown menu
Structured Text Editing

Sketch-n-Sketch : Deuce Code Editor

Elm* ≈ Elm ≈ ML

Demo

Structured Text Editing
Structured Text Editing
[ICSE 2018, work-in-progress]
Related Work

Plain Text Editing

Hybrid Editors
- Barista
- Active Code Completion
- Greenfoot
- Code Bubbles
  Sketch-n-Sketch: Deuce

Structured Editing

Automated Refactoring

Alternative UIs for Refactoring
- Refactoring w/ Synthesis
- Drag-n-Drop Refactoring
  Sketch-n-Sketch: Deuce
Lightweight Structured Editing via Direct Manipulation

Structural (Multi-)Selection; Context-Sensitive Tool Menu; Hover Preview and Confirm

Traditional UI
Text Selection; Tool Menu; Configuration Wizard

Automated Refactoring
1. Learning Curve for New UI + Tools
2. Faster Tool Invocations using New UI
3. Overwhelming Preference for New UI

Structural (Multi-)Selection; Context-Sensitive Tool Menu; Hover Preview and Confirm

Traditional UI
Text Selection; Tool Menu; Configuration Wizard

21 users
END