Direct Manipulation Programming Systems

Ravi Chugh
Brian Hempel
Jacob Albers
Grace Lu
Justin Lubin
Mitchell Spradlin
Idea
Idea
Prototype… Repair… Refactor…

Idea
<table>
<thead>
<tr>
<th>Programming Language</th>
<th>Direct Manipulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>2D Graphics</td>
<td><img src="image" alt="Ai" /></td>
</tr>
<tr>
<td>Presentations</td>
<td><img src="image" alt="P" /></td>
</tr>
<tr>
<td>Documents</td>
<td><img src="image" alt="W" /></td>
</tr>
<tr>
<td>Spreadsheets</td>
<td><img src="image" alt="X" /></td>
</tr>
<tr>
<td>Web Apps</td>
<td><img src="image" alt="d" /></td>
</tr>
</tbody>
</table>
Prototype... Repair... Refactor...
Programming with:

Less Keyboard.

More Mouse.
Programming with:
Less Keyboard. More Mouse.

Idea
Prototype... Repair... Refactor...
\(\lambda\)-Calculus + Direct Manipulation
λ-Calculus + Direct Manipulation

0 : Text-Edit Code
λ-Calculus + Direct Manipulation of Code

0: Text-Edit Code

1: Mouse-Edit Code
\(\lambda\text{-Calculus} + \text{Direct Manipulation of Output}\)
Part 1

1: Mouse-Edit Code
Lightweight Structured Editing via Direct Manipulation of (Plain Text) Code

Plain Text Editing

Structured Editing

Automated Refactoring
Lightweight Structured Editing

[In submission]
Sketch-n-Sketch Demo

Live Synchronization
[PLDI 2016]

Lightweight Structured Editing
[In submission]
Part 1

Lightweight Structured Editing
= Plain Text
+ AST Node Selection
+ AST Transformations

1: Mouse-Edit Code
Part 2

2: Mouse-Edit Output
Semi-Automated Programming by Manipulating Output
Sketch-n-Sketch Demo

Live Synchronization  
[PLDI 2016]

Lightweight Structured Editing  
[In submission]

Draw, Relate, Group and Abstract  
[UIST 2016]
Live Synchronization
[PLDI 2016]

Lightweight Structured Editing
[In submission]

Draw, Relate, Group and Abstract
[UIST 2016]
Draw  Relate  Group  Tweak

e ⇓ v
Constraint on Original $e$:

$e \downarrow_t v$

$\Rightarrow$

$v = t$

Constraint on Updated $e'$:

$v' = t$
Constraint on Original $e$: 

$$v = t$$

Constraint on Updated $e'$: 

$$v' = t$$
Dataflow-Only Traces

\[ t ::= x \]
\[ t_1 + t_2 \]
\[ t_1 \times t_2 \]
\[ \sin t \]
\[ \text{pow } t_1 t_2 \]
\[ \ldots \]
let a = 1

b = 2

in ...

a+b

a+b*b

(\x.x) a

true ? a : b
110 = x0 + 2 * sep

155 = x0 + 2 * sep

Change x0 or sep? Or both? Or more?

Change exactly one constant.
Heuristics to automatically choose.
Annotations to freeze (n!) or thaw (n?).
Can solve for $x$ if $x$ only occurs once.

\[
5*\sin(x) = n \quad ✓
\]
\[
x*(y+y^2) = n \quad ✓
\]
\[
x^2 = n \quad ✓
\]
Can solve for $x$ if $x$ only occurs once.

- $5\sin(x) = n$  ✓
- $x(y+y^2) = n$  ✓
- $x(x+y^2) = n$  ✗
- $x^2 = n$  ✓
- $xx = n$  ✗

Good enough for 80% of equations.
"Raw" Shape

(x1, y1)

(x3, y3)

(x2, y2)
Draw

Relate

Group

Tweak

"Anchored" Shape

(x1, y1)

(x1+2, y1+8)

(x1+10, y1+10)
"Bounded" Shape

\[(x_1, y_1)\]

\[(x_1 + 0.20 \times (x_2 - x_1), y_1 + 0.80 \times (y_2 - y_1))\]

\[(x_2, y_2)\]
Draw | Relate | Group | Tweak

Representation affects grouping operations.

```
(def rawPoly
  (let [x1 y1] ...
    (let [x2 y2] ...
      (let [x3 y3] ...
        ... ))))
```

```
(def anchoredPoly
  (let [x1 y1] ...
    (let [dx2 dy2] ...
      (let [dx3 dy3] ...
        ... ))))
```

```
(def boundedPoly
  (let [x1 y1] ...
    (let [x2 y2] ...
      (let [px3 py3] ...
        ... ))))
```
"Anchored" Group

"Bounded" Group
"Anchored" Group

"Bounded" Group
"Anchored" Group

"Bounded" Group
"Anchored" Group

"Bounded" Group
(def rect1
  (let x 52 ... ))

(def line2
  (let x1 50 ... ))

(def line3
 ... )
def rect1:
  let x = 52
  ...
  )

def line2:
  let x1 = 50
  ...
  )

def line3:
  ...
  )

SELECT Features in Canvas
DIG New Hole in Code
FILL Hole with Relationship
CLEAN New Code
(def rect1
  (let x 52 ... ))

(def line2
  (let x1 50 ... ))

(def line3
  ... )
(def rect1
  (let [x 52 ... ]))

(def line2
  (let [x1 50 ... ]))

(def line3
  ... )
(def rect1_x 52)
(def line2_x1 50)
(def [rect1_x' line2_x1'] [rect1_x line2_x1])

(def rect1
  (let x rect1_x' ... ))
(def line2
  (let x1 line2_x1' ... ))
(def line3 ...

LIFT  Constants and
DIG  New Hole
FILL  Hole w/
      Desired Relationship
(def rect1_x 52)
(def line2_x1 50)
(def [rect1_x' line2_x1'] [rect1_x rect1_x])

(def rect1
  (let x rect1_x' ... ))

(def line2
  (let x1 line2_x1' ... ))

(def line3
  ... )
(def rect1_x 52)

(def [rect1_x' line2_x1'] [rect1_x rect1_x])

(def rect1
  (let x rect1_x'
    ... ))

(def line2
  (let x1 line2_x1'
    ... ))

(def line3
  ... )
(def rect1_x 52)

(def rect1
  (let x rect1_x ... ))

(def line2
  (let x1 rect1_x ... ))

(def line3
  ... )
(def rect1_x 52)

(def rect1
  (let x rect1_x ... ))

(def line2
  (let x1 rect1_x ... ))

(def line3
  ... )

Automatic solver for Make Equal.
Which Constants to Abstract?

**Heuristic: Unfrozen and Named**

```
(def foo
  (let [a b] [1 2!]
    [a b 3]))
foo
```
Which Constants to Abstract?

Heuristic: Unfrozen and Named

(def foo
  (let [a b] [1 2!]
    [a b 3]))

foo

(def foo (
  \a
  (let [b] [2!]
    [a b 3]))

(foo 1)
Programming in 2015

Sketch-n-Sketch

Less Keyboard.
More Mouse.
Draw  Relate  Group  Tweak

Sketch-n-Sketch
Less Reliance on Syntactic Structure
; Top-Level Defs
(def rect1 ...)
(def line2 ...)
(def line3 ...)

; Main Expression
[ rect1 ... ]
(def group1
  (def rect1 ...)
  (def line2 ...)
  (def line3 ...)
  [ rect1 ... ] )

group1

Less Reliance on Syntactic Structure
(def polygon7_bot (+ (+ (* 0.5!...)
(def k3105 // (- (+ (- polygon6...)
(def polygon7_top (- (* 0.5! (+...)
(def [polygon5_right k3038] [(-....
(def k3061 // (- (+ polygon5_r...)
(def polygon6_bot (- (+ (- poly...)
(def k3063 // (- (+ polygon6_bo...)
(def polygon5_top (- polygon6_t...)
(def k3103 // (- (- polygon5...)
(def [k3041 polygon5_bot] [(- p...)
(def k3134 // (- (+ k3041 helpe...)
(def k3141 // (- (+ k3038 helpe...
Draw

Relate

Group

Tweak

Sketch-n-Sketch

Sketch-n-Sketch ++

Additional Interaction to Resolve User Intent
**Draw**  Sketch-n-Sketch

**Relate**  Sketch-n-Sketch ++

**Group**  Limited, Syntactic Program Updates

**Tweak**  Expressive, General Program Synthesis
Idea

Prototype; Repair; Refactor; Repeat
Prototype; Repair; Refactor; Repeat

<table>
<thead>
<tr>
<th>2D Graphics</th>
<th>PL</th>
<th>DM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Presentations</td>
<td>General Purpose PL</td>
<td>Domain Specific UI</td>
</tr>
<tr>
<td>Documents</td>
<td>+ General Program Synthesis</td>
<td>+ Domain Specific Program Synthesis</td>
</tr>
<tr>
<td>Spreadsheets</td>
<td>Sketch-n-Sketch</td>
<td></td>
</tr>
<tr>
<td>Web Apps</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Prototype; Repair; Refactor; Repeat

<table>
<thead>
<tr>
<th>PL</th>
<th>DM</th>
</tr>
</thead>
<tbody>
<tr>
<td>2D Graphics</td>
<td>Domain Specific UI</td>
</tr>
<tr>
<td>Presentations</td>
<td>+</td>
</tr>
<tr>
<td>Documents</td>
<td>Domain Specific Program</td>
</tr>
<tr>
<td>Spreadsheets</td>
<td>Synthesis</td>
</tr>
<tr>
<td>Web Apps</td>
<td>Sketch-n-Sketch</td>
</tr>
</tbody>
</table>
## Prototype; Repair; Refactor; Repeat

<table>
<thead>
<tr>
<th></th>
<th>PL</th>
<th>DM</th>
</tr>
</thead>
<tbody>
<tr>
<td>2D Graphics</td>
<td>General Purpose PL + General Program Synthesis</td>
<td>Domain Specific UI + Domain Specific Program Synthesis</td>
</tr>
<tr>
<td>Presentations</td>
<td></td>
<td>DM Code / Traces</td>
</tr>
<tr>
<td>Documents</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spreadsheets</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Web Apps</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Programs</td>
<td></td>
<td>Sketch-n-Sketch</td>
</tr>
</tbody>
</table>
Programming with:

Less Keyboard.

More Mouse.
Related Work

Plain Text Editing

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

Structured Editing

Automated Refactoring

Alternative UIs for Refactoring
- Refactoring w/ Synthesis
- Drag-n-Drop Refactoring
- Sketch-n-Sketch
Related Work

**DM + Code**
Dynamic Drawing Apparatus

**Code + DM**
Wang et al. [FSE 2012]
McDirmid's Demos
Sketch-n-Sketch

**Constraints**
Sketchpad
ThingLab
Juno-2

**Program Synthesis**
(Sketching, PBE)

**Program Repair**
http://ravichugh.github.io/sketch-n-sketch/

DEMO      VIDEOS
CODE      PAPERS

20,000+ LOC Elm

PLDI 2016

UIST 2016
\(\lambda\text{-Calculus + Direct Manipulation of Code and Output}\)
\textbf{\textit{\lambda}}-\textit{Calculus} + \textit{Direct Manipulation}

Reuse effort across domains.

Enable experts and library writers to extend built-ins.

Smooth spectrum between "Experts" and "End Users".