Programming By Demonstration

But what if the user is fallible?

Thibaud Hottelier, Rastislav Bodik
How Fancy Layout Are Created Today

Pick a “canned” layout from ProtoViz/D3 (DSL)
  • Limited to the library.
  • Non-programmers cannot define their own.

If programmer, write your own layout engine.
  • Takes days in our experience (cs164).
  • Cannot quickly try out ideas. Slow turnaround.

Best of both world with PBD?
  • Touch friendly programming?
A Treemap the Easy Way

Buttons and Knobs

and many more...
Intent Through Mockups

1. Prune layouts disagreeing with mockup.
   - Criteria: containment, overlap
2. Rank layouts using heuristics
3. Check results on other documents
   - We compile to layout engines.

<table>
<thead>
<tr>
<th></th>
<th>Treemap</th>
<th>Bar chart</th>
<th>Icicle chart</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plain</td>
<td>964</td>
<td>736</td>
<td>84</td>
</tr>
<tr>
<td>With Mockup</td>
<td>6</td>
<td>44</td>
<td>27</td>
</tr>
<tr>
<td>With Ranking</td>
<td>2nd</td>
<td>12th</td>
<td>3rd</td>
</tr>
</tbody>
</table>
Exploration Through Corrections

User can edit layout by dragging/resizing elements
  • Direct interaction. Concrete, speaks to designers.
  • Under the hood, remove and put back constraints.

Demonstration on Barchart.

Conclusion
  • Only three layout explored (in a space of ~800).
  • Compiles down to a reusable layout engine.
Science!
Vanilla PBD

Each demonstration narrows the space of candidates until only program is left: The target program.
What if demonstrations have noise?

User means but demos

With classical SAT approaches target program will be missed!

A single tiny imprecision in an early demonstration can jeopardize the entire process.
Abstract Demonstrations

Extract high level properties that are hopefully noise resistant.

But some candidates cannot be ruled out with demonstrations.

Problem: unquantifiable loss of distinguishing power!
Relaxed Demonstrations

Start with a plausible candidate (guess). Execute it.

Let the user edit/fix directly the program output.

Editing is an instance of bi-directional programming. Each changes on the output affects the program in either by

- Introducing non-determinism: Punch a hole.
- Restricting non-determinism: Fill a hole.
Relaxed Demo Details

With each new “correction” provided by the user, we move through the space of programs, closing in upon the goal.

The space of programs form a subset lattice.
\[ p_1 \subseteq p_2 \iff \text{outputs}(p_1) \subseteq \text{outputs}(p_2) \]
- Most non-deterministic program at the top
- Fully deterministic programs at the bottom

Navigation is non-monotonic.
Goal is always reachable, imprecisions cause detours.
Questions

Can we define “progress” in presence of noise?

How can I compare two program spaces?
What happen when I grow program space with new construct?
  • Can I measure ease of demonstration?

How many steps? / How big is each step?
How much information is
  • Contained in user demonstration
  • Gained by the PBD system