CodeHint: Dynamic and Interactive Synthesis for Modern IDEs

Joel Galenson, Philip Reames, Rastislav Bodik, Koushik Sen

University of California, Berkeley
{joel, reames, bodik, ksen}@cs.berkeley.edu

June 10, 2013
Problem

Autocomplete is broken:

- static
- inexpressive
Problem Solution

Autocomplete is broken can be fixed:

- static dynamic
- inexpressive easy-to-use specifications
- interactive
Demo: First task
Demo: First task

Summary:

- generate and run expressions
- inspect result
Demo: Second task
Demo: Second task

Summary:
- want queries...
- ...over dynamic values
Demo: Third task
Demo: Third task

Summary:
- general queries
- multiple demonstrations
Demo: Summary

Bringing autocomplete beyond the 1980’s:

- Query for types, values, and more
- Support runtime reasoning
- Interactivity
- Use information from tests
Why am I at the mobile session?

Storytime
Specifications

- Context-dependent
  - Value demonstrations
    - Programming by Demonstration
  - Classic correctness conditions
    - Strong

- Context-independent
  - API Exploration
    - Static type demonstrations
    - Weak

Strength
Generate and evaluate expressions at runtime.
- Lets us handle I/O, native calls, reflection, etc.

Depth 0
- x
- y

Depth 1
- +
  - x
  - y
  - foo
    - x
    - y

Depth 2
- bar
  - foo
    - x
    - y
  +
    - x
    - y
- Generate and evaluate expressions at runtime.
  - Lets us handle I/O, native calls, reflection, etc.
- Group results into equivalence classes.
Algorithms

- Generate and evaluate expressions at runtime.
  - Lets us handle I/O, native calls, reflection, etc.
- Group results into equivalence classes.
- Prefer more likely expressions with probabilistic model.
What about side effects?

- Detect with watchpoints
  - Requires no special sandbox or VM.
  - Lots of tricky special cases.
What about side effects?

- Detect with watchpoints
  - Requires no special sandbox or VM.
  - Lots of tricky special cases.
- Use security manager to block bad native calls.
What about side effects?

- Detect with watchpoints
  - Requires no special sandbox or VM.
  - Lots of tricky special cases.
- Use security manager to block bad native calls.
- Define equivalence-relation per-state.

\[
x = 0
\]

\[
\text{xPlus42()} \sim \text{fortyTwo()}
\]

but

\[
\begin{align*}
\text{incX()} & \rightarrow \text{xPlus42()} \\
\text{incX()} & \rightarrow \text{fortyTwo()}
\end{align*}
\]
### Empirical Results

**Old numbers!**

<table>
<thead>
<tr>
<th></th>
<th>Depth 1</th>
<th>Normal algorithm</th>
<th>Depth 2</th>
<th>Depth 3</th>
<th>No equivalences</th>
<th>No heuristics</th>
<th>Brute force</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># gen</td>
<td>Time (s)</td>
<td># gen</td>
<td>Time (s)</td>
<td># gen</td>
<td>Time (s)</td>
<td># gen</td>
</tr>
<tr>
<td>P1</td>
<td>34</td>
<td>0.2</td>
<td>1189</td>
<td>1.7</td>
<td>54850</td>
<td>13.6</td>
<td>12050</td>
</tr>
<tr>
<td>P2</td>
<td>52</td>
<td>0.3</td>
<td>939</td>
<td>1.8</td>
<td>45895</td>
<td>9.2</td>
<td>8454</td>
</tr>
<tr>
<td>P3</td>
<td>118</td>
<td>0.3</td>
<td>1222</td>
<td>0.9</td>
<td>226654</td>
<td>26.6</td>
<td>25677</td>
</tr>
<tr>
<td>P4</td>
<td>7</td>
<td>0.2</td>
<td>46</td>
<td>0.2</td>
<td>686</td>
<td>0.7</td>
<td>118</td>
</tr>
<tr>
<td>P5</td>
<td>25</td>
<td>0.2</td>
<td>264</td>
<td>0.2</td>
<td>6046</td>
<td>1.3</td>
<td>1590</td>
</tr>
<tr>
<td>S1</td>
<td>7</td>
<td>0.6</td>
<td>222</td>
<td>1.4</td>
<td>15107</td>
<td>6.2</td>
<td>470</td>
</tr>
<tr>
<td>S2</td>
<td>11</td>
<td>0.6</td>
<td>257</td>
<td>1.5</td>
<td>21214</td>
<td>5.3</td>
<td>652</td>
</tr>
<tr>
<td>S3</td>
<td>16</td>
<td>0.6</td>
<td>458</td>
<td>2.1</td>
<td>11430</td>
<td>6.2</td>
<td>1081</td>
</tr>
<tr>
<td>S4</td>
<td>106</td>
<td>0.9</td>
<td>1788</td>
<td>1.9</td>
<td>1345324</td>
<td>-</td>
<td>16339</td>
</tr>
<tr>
<td>S5</td>
<td>326</td>
<td>0.7</td>
<td>629</td>
<td>2.1</td>
<td>23678</td>
<td>61.9</td>
<td>2088</td>
</tr>
<tr>
<td>R 1</td>
<td>20</td>
<td>0.1</td>
<td>91</td>
<td>0.2</td>
<td>3447</td>
<td>0.9</td>
<td>1511</td>
</tr>
<tr>
<td>R 2</td>
<td>12</td>
<td>0.1</td>
<td>136</td>
<td>0.2</td>
<td>1160</td>
<td>0.6</td>
<td>2688</td>
</tr>
<tr>
<td>R 3</td>
<td>7</td>
<td>0.1</td>
<td>12</td>
<td>0.1</td>
<td>35</td>
<td>0.3</td>
<td>42</td>
</tr>
<tr>
<td>R 4</td>
<td>6</td>
<td>0.1</td>
<td>21</td>
<td>0.1</td>
<td>78</td>
<td>0.2</td>
<td>68</td>
</tr>
<tr>
<td>R 5</td>
<td>24</td>
<td>0.2</td>
<td>249</td>
<td>0.2</td>
<td>14337</td>
<td>2</td>
<td>3926</td>
</tr>
<tr>
<td>Average</td>
<td>31.4</td>
<td>0.3</td>
<td>501.5</td>
<td>1.0</td>
<td>132209.4</td>
<td>9.6</td>
<td>5116.9</td>
</tr>
<tr>
<td>Median</td>
<td>20</td>
<td>0.2</td>
<td>257</td>
<td>0.9</td>
<td>14337</td>
<td>3.7</td>
<td>1590</td>
</tr>
</tbody>
</table>

Note: The numbers represent different algorithmic depths and execution times. The table compares the performance of the normal algorithm with variations in depth and heuristics. The "Brute force" column indicates the number of generations for each depth.
Productivity Results

Fewer bugs ($p < 0.2$)
Productivity Results

Productivity (p < 0.01)

% completed with CodeHint

% completed without CodeHint
Conclusion

CodeHint is:

▶ dynamic
▶ easy to use
▶ interactive

It improves programmer productivity.

Try it out!

http://www.cs.berkeley.edu/~joel/codehint/
https://github.com/jgalenson/codehint