Program Synthesis in the Industrial World: Inductive, Incremental, Interactive

Alex Polozov
polozov@cs.washington.edu

Sumit Gulwani
sumitg@microsoft.com

And the rest of the PROSE team!
prose-contact@microsoft.com
PROgram Synthesis using Examples

Allen Cypher
Sumit Gulwani
Ranvijay Kumar
Vu Le
Daniel Perelman
Alex Polozov
Mohammad Raza
Danny Simmons
Adam Smith
Abhishek Udupa

R&D team, MSR → industrial Microsoft

We are hiring! Interns or full-time 😊
This talk

- Solutions
- Lessons
- Challenges
Outline

- Programming by Examples (PBE) & PROSE: Quick Background

- Mass-Market Deployment
  - Goals
  - Challenges
  - Solutions

- Discussion
PBE & PROSE

A 3-slide Background
Motivation

99% of spreadsheet users do not know programming

Data scientists spend 80% time wrangling raw data
PROSE Timeline

FlashFill (text transformations) 2010-2012 [POPL 11]

FlashExtract (text extraction) 2012-2014 [PLDI 14]

FlashRelate (table transformations) 2012-2015 [PLDI 15]

... FlashMeta (PBE framework) 2014-2015 [OOPSLA 15]

PROSE SDK 2015-present
PBE Architecture

Example-based intent spec $\varphi$ → Program Synthesizer → Ranked program set $\tilde{N}$ → Debugging → Intended program $P \in \mathcal{L}$

- Refined intent
- Example-based intent spec $\varphi$
- Program Synthesizer
- Ranking function $h$
- DSL $\mathcal{L}$
- Ranked program set $\tilde{N}$
- Debugging
- Test inputs $\tilde{\sigma}$
- Intended program in Python/C#/C++/…

Translator
Mass-Market Deployment

Goals & Challenges
User Experience

Inductive
(intent is easily specified)

Interactive
(facilitates the debugging cycle)

Ambiguity resolution

Predictive synthesis

Scalable
(snappy UI = responds in < 1 s)

Incremental synthesis

Engineering practices

Agile
(quick software development)

Engineering practices
User Experience

Inductive
(intent is easily specified)

Ambiguity resolution

Predictive synthesis

Interactive
(facilitates the debugging cycle)

Scalable
(snappy UI = responds in < 1 s)

Incremental synthesis

Agile
(quick software development)

Engineering practices
Engineering practices

• Production-quality library code
  • Prototyping still exists, but it’s not the final form

• Unit tests & TDD

• Integration tests: real-life scenarios
  • Close to 8K for all DSLs in total
  • Most are mined from public sources (e.g. help forums)
  • In preparation: benchmark suite release for the community
Performance-minded engineering

- Parallelization of learning matters
  - *E.g.*: multi-user log file processing in Azure Log Analytics

- Performance of program execution matters
  - *E.g.*: “Big Data” on an end-user’s machine
  - Smallest ≠ fastest!
  - *(1)* Synthesize many correct programs, then *(2)* optimize for the fast ones

Robustness-based ranking

Performance-based ranking
Development

• DSL design: \( \approx 10 \) months \( \rightarrow \approx 2 \) weeks
  • This is not a bottleneck!*

• Ranking: bulk of the effort
  • Designing a score for an operator \( F \) is 2-3x longer than designing \( F \) (incl. synthesis!)
  • *E.g.*: rock-paper-scissors among string processing operators

* Once you learn the skill...
Ambiguity resolution

Inductive
(intent is easily specified)

Predictive synthesis

Interactive
(facilitates the debugging cycle)

User Experience

Scalable
(snappy UI = responds in < 1 s)

Incremental synthesis

Engineering practices

Agile
(quick software development)
From:
all lines ending with “Number ◦ Dot”
“Space ◦ Number ◦ Dot”
starting with “Word ◦ Space ◦ CamelCase”

Extract:
the first “Number” before a “Dot”
the last “Number” before a “Dot”
the last “Number” before a “Dot ◦ LineBreak”
the last “Number”
text between the last “Space” and the last “Dot”
the first “Comma ◦ Space” and the last “Dot ◦ LineBreak”

…and up to $10^{20}$ more candidates
Anecdotes

- FlashFill was not accepted to Excel until it solved the most common scenarios from 1 example

<table>
<thead>
<tr>
<th>Adam Smith</th>
<th>Adam</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice Williams</td>
<td>Alic</td>
</tr>
</tbody>
</table>

- Some users still don’t know you can give 2!
Ambiguity resolution

Option 1: machine-learned robustness-based ranking

- Idioms/patterns from test data can influence search & ranking
- *E.g.:* bucketing

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>76-100</td>
</tr>
<tr>
<td>51</td>
<td>51-75</td>
</tr>
<tr>
<td>86</td>
<td></td>
</tr>
</tbody>
</table>
Ambiguity resolution

**Option 1: machine-learned robustness-based ranking**  
[CAV 15]

- Idioms/patterns from test data can influence search & ranking
- *E.g.:* bucketing

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>76-100</td>
</tr>
<tr>
<td>51</td>
<td>51-75</td>
</tr>
<tr>
<td>86</td>
<td></td>
</tr>
</tbody>
</table>

\[x \Rightarrow \text{Concat}(\text{Round}(x, \text{Down}, 25), \text{Const}("-"), \text{Round}(x, \text{Up}, 25))\]
Ambiguity resolution

Option 1: machine-learned robustness-based ranking

- Idioms/patterns from test data can influence search & ranking
- E.g.: bucketing

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>76-100</td>
</tr>
<tr>
<td>51</td>
<td>51-75</td>
</tr>
<tr>
<td>86</td>
<td></td>
</tr>
</tbody>
</table>

\[ x \Rightarrow \text{Concat} \left( \text{Round}(x, \text{Down}, 25), \, \text{Const}(\text{"-"}), \, \text{Round}(x, \text{Up}, 25) \right) \]
Ambiguity resolution

Option 1: machine-learned robustness-based ranking

- Idioms/patterns from test data can influence search & ranking
- *E.g.*: bucketing

\[
x \Rightarrow \text{Concat} \left( \text{Round}(x, \text{Down}, 25), \text{Const}(\text{"-"}), \text{Round}(x, \text{Up}, 25) \right)
\]

<table>
<thead>
<tr>
<th></th>
<th>100</th>
<th>76-100</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>51</td>
<td>51-75</td>
</tr>
<tr>
<td></td>
<td>86</td>
<td></td>
</tr>
</tbody>
</table>

Option 2: interactive clarification
Predictive synthesis

- Inductive (intent is easily specified)
- Scalable (snappy UI = responds in < 1 s)
- Ambiguity resolution
- Incremental synthesis
- Interactive (facilitates the debugging cycle)
- Engineering practices
- Agile (quick software development)
Example-based intent spec $\varphi$ → Program Synthesizer

- Ranking function $h$
- DSL $\mathcal{L}$

Ranked program set $\tilde{N}$ → Debugging

- Test inputs $\tilde{\sigma}$

Refined intent

→ Intended program $P \in \mathcal{L}$ → Translator

Intended program in Python/C#/C++/…
PBE Architecture

Example-based intent spec \( \varphi \)

Program Synthesizer

Refined intent

Ranked program set \( \tilde{N} \)

Hypothesizer

User

Questions

Test inputs \( \tilde{\sigma} \)

Intended program in Python/C#/C++/…

Translator

Intended program \( P \in \mathcal{L} \)

Ranking function \( h \)

DSL \( \mathcal{L} \)
PBE Architecture

Example-based intent spec $\varphi$ → Program Synthesizer → Ranked program set $\tilde{N}$ → Intended program $P \in \mathcal{L}$

Refined intent

- Ranking function $h$
- DSL $\mathcal{L}$

Hypothesizer

- Intended program in Python/C#/C++/…
- Test inputs $\hat{\sigma}$

User

Questions
Hypothesizer

Given a program set $\tilde{N}$, find program constraints ("hypotheses") $\varphi$ that best disambiguate among programs in $\tilde{N}$, and present them to the user as multiple-choice questions.

✓ Reduces the cognitive load on the user

✓ Reduces the number of iterations by choosing the most effective disambiguating questions

✓ Increases the user’s confidence in the system ("proactive = smart")
Example

<table>
<thead>
<tr>
<th>Missing page numbers, 1993</th>
<th>1993</th>
</tr>
</thead>
<tbody>
<tr>
<td>64-67, 1995</td>
<td>64</td>
</tr>
</tbody>
</table>

Which output is correct here?

a. 64  
b. 67  
c. 1995
Example

<table>
<thead>
<tr>
<th>Missing page numbers, 1993</th>
<th>1993</th>
</tr>
</thead>
<tbody>
<tr>
<td>64-67, 1995</td>
<td>64</td>
</tr>
</tbody>
</table>

Which output is correct here?

a. 64  
b. 67  
c. 1995

\[ \varphi_{i+1}: P(\sigma_2) = "1995" \]
Example

Which output is correct here?

a. 64  
b. 67  
c. 1995

\[ \varphi_{i+1}: P(\sigma_2) = "1995" \]
Example – alternative

<table>
<thead>
<tr>
<th>Missing page numbers, 1993</th>
<th>1993</th>
</tr>
</thead>
<tbody>
<tr>
<td>64-67, 1995</td>
<td>64</td>
</tr>
</tbody>
</table>

... ... ...

Is this part of the input relevant?
- a. Yes
- b. No
- c. Maybe
Picking the right question

“Distinguishability” = effectiveness for disambiguation

1. An input is distinguishing if many top-ranked candidate programs disagree on the intended output on it.
   • Any response will partition the program set well

2. A question is distinguishing if the alternative candidate programs corresponding to all potential responses have high ranks.
   • Any response will lead to a good alternative program

Preliminary results: good questions yield just 4-6 iterations until convergence
User Experience

- **Inductive** (intent is easily specified)
- **Ambiguity resolution**
- **Predictive synthesis**
- **Interactive** (facilitates the debugging cycle)
- **Scalable** (snappy UI = responds in < 1 s)
- **Engineering practices**
- **Agile** (quick software development)

**Incremental synthesis**
Big Data
Big Data + Program Synthesis
Problem definition

Given a program set $\tilde{N}_i \subset \mathcal{L}$ that satisfies the currently accumulated spec $\varphi_i$, and a new constraint $\psi_{i+1}$, learn a subset $\tilde{N}_{i+1} \subset \tilde{N}_i$ of programs that satisfy the new spec $\varphi_{i+1} = \varphi_i \land \psi_{i+1}$

- $\mathcal{L}$ is an industrial DSL (e.g., FlashFill)
- $|\tilde{N}_i| \approx 10^{20}$
- Time limit: $\approx 1$ sec
Background: Version Space Algebra

\[
\text{int positionIn[string } s] := \text{AbsPos}(s, k) \\
| \text{RegPos}(s, \text{std.Pair}(r, r), k);
\]
Background: Version Space Algebra

```c
int positionIn[string s] := AbsPos(s, k)
    | RegPos(s, std.Pair(r, r), k);
```

Sharing #1: cross-product representation
Background: Version Space Algebra

```c
int positionIn[string s] := AbsPos(s, k)
    | RegPos(s, std.Pair(r, r), k);
```

**Sharing #1:** cross-product representation

**Sharing #2:** equal sets are shared as the same DAG node
**Background: Version Space Algebra**

\[
\text{int} \quad \text{positionIn} \left[ \text{string} s \right] := \text{AbsPos} (s, k) \mid \text{RegPos} (s, \text{std.Pair} (r, r), k);
\]

Sharing #1: cross-product representation

For, e.g., FlashFill, volume \( (\tilde{N}) \approx 10^5 \) when \( |\tilde{N}| \approx 10^{20} \).

Runtime of VSA operations \( \propto \) VSA volume.

Sharing #2: equal sets are shared as the same DAG node

For, e.g., FlashFill, volume \( (\tilde{N}) \approx 10^5 \) when \( |\tilde{N}| \approx 10^{20} \).
VSAs and CFGs are two isomorphic representations for a language.

\[
\text{Filter}(\tilde{N}, \psi) \equiv \text{Learn}(\mathcal{L}(\tilde{N}), \psi)
\]
Incremental Inductive Synthesis

1. Implicitly represent $\tilde{N}_i$ (already a VSA!) as an isomorphic CFG $\mathcal{L}(\tilde{N}_i)$.

2. Analyze the descriptive power of $\psi_{i+1}$:
   - **Definitive** (e.g., examples, set membership, subsequence constraints):
     Apply regular top-down deductive synthesis on $\mathcal{L}(\tilde{N}_i)$
   - **Locally refining** (e.g., datatypes, input relevance):
     Re-run backpropagation only on relevant parts of $\mathcal{L}(\tilde{N}_i)$
   - **Globally refining** (e.g., negative examples):
     Filter $\tilde{N}_i$ at the top level
Preliminary results

- Big improvement when VSA fragmentation is limited
- Not the final results; work in progress has orders-of-magnitude improvements
Lessons & Conclusions

• Robust engineering is the key to PBE deployment
• Ranking $\gg$ learning (in industrial PBE)
• Interaction models should be first-class citizens in synthesis frameworks
  • Great theoretical results: e.g., OGIS [Jha & Seshia 2015]
  • Also need: HCI evaluations, comparison of query types, worst-case TD optimization
• Proactive ambiguity analysis of current candidate programs
• Incrementality: treat the previous program set as the new search space
  • Requires full program set computation (possibly in the background)

Thank you!

https://microsoft.github.io/prose
prose-contact@microsoft.com