Interactive Program Synthesis
The big picture

What if your excel was smart?
<table>
<thead>
<tr>
<th>Email</th>
<th>Column 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Image 18x28 to 350x352]</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>123.4567</td>
<td>123.46</td>
</tr>
<tr>
<td>123.4</td>
<td>123.40</td>
</tr>
<tr>
<td>78.234</td>
<td>78.23</td>
</tr>
</tbody>
</table>

(a)

<table>
<thead>
<tr>
<th>Language</th>
<th>Format descriptor for rounding to two decimal places</th>
</tr>
</thead>
<tbody>
<tr>
<td>Excel, C#</td>
<td>.00</td>
</tr>
<tr>
<td>Python, C</td>
<td>.2f</td>
</tr>
<tr>
<td>Java</td>
<td>#.###</td>
</tr>
</tbody>
</table>

(b)

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0d 5h 26m</td>
<td>5:00</td>
</tr>
<tr>
<td>0d 4h 57m</td>
<td>4:30</td>
</tr>
<tr>
<td>0d 4h 27m</td>
<td>4:00</td>
</tr>
<tr>
<td>0d 3h 57m</td>
<td>3:30</td>
</tr>
</tbody>
</table>

(c)

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>08/21/2010</td>
<td>08/21/2010</td>
</tr>
<tr>
<td>07/24/2010</td>
<td>07/24/2010</td>
</tr>
<tr>
<td>20.08.2010</td>
<td>08/20/2010</td>
</tr>
<tr>
<td>23.08.2010</td>
<td>08/23/2010</td>
</tr>
<tr>
<td>2010-06-07</td>
<td>07/06/2010</td>
</tr>
<tr>
<td>2010-24-08</td>
<td>08/24/2010</td>
</tr>
</tbody>
</table>

(d)
<table>
<thead>
<tr>
<th>Input $v_1$</th>
<th>Input $v_2$</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stroller</td>
<td>10/12/2010</td>
<td>$145.67 + 0.30 \times 145.67$</td>
</tr>
<tr>
<td>Bib</td>
<td>23/12/2010</td>
<td>$3.56 + 0.45 \times 3.56$</td>
</tr>
<tr>
<td>Diapers</td>
<td>21/1/2011</td>
<td>$21.45 + 0.35 \times 21.45$</td>
</tr>
<tr>
<td>Wipes</td>
<td>2/4/2009</td>
<td>$5.12 + 0.40 \times 5.12$</td>
</tr>
<tr>
<td>Aspirator</td>
<td>23/2/2010</td>
<td>$2.56 + 0.30 \times 2.56$</td>
</tr>
</tbody>
</table>

(a) MarkUpReCod

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Markup</th>
</tr>
</thead>
<tbody>
<tr>
<td>S30</td>
<td>Stroller</td>
<td>30%</td>
</tr>
<tr>
<td>B56</td>
<td>Bib</td>
<td>45%</td>
</tr>
<tr>
<td>D32</td>
<td>Diapers</td>
<td>35%</td>
</tr>
<tr>
<td>W98</td>
<td>Wipes</td>
<td>40%</td>
</tr>
<tr>
<td>A46</td>
<td>Aspirator</td>
<td>30%</td>
</tr>
</tbody>
</table>

(b) CostRec

<table>
<thead>
<tr>
<th>Id</th>
<th>Date</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>S30</td>
<td>12/2010</td>
<td>$145.67</td>
</tr>
<tr>
<td>B56</td>
<td>12/2010</td>
<td>$142.38</td>
</tr>
<tr>
<td>D32</td>
<td>1/2011</td>
<td>$21.45</td>
</tr>
<tr>
<td>W98</td>
<td>4/2009</td>
<td>$5.12</td>
</tr>
<tr>
<td>A46</td>
<td>2/2010</td>
<td>$2.56</td>
</tr>
</tbody>
</table>
(a) 

BlueLabel  GreenLabel  YellowLabel
Ana Trujillo  Redmond  (757) 555-1634
Antonio Moreno  Renton  (411) 555-2786
Thomas Hardy  Seattle  (412) 555-5719
Christina Berglund  Redmond  (443) 555-6774
Hanna Moos  Puyallup  (376) 555-2462
Frederique Citeaux  Redmond  (689) 555-2770

(b) 

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Albania</td>
<td>1000</td>
<td>1950</td>
<td>930</td>
</tr>
<tr>
<td>3</td>
<td>Austria</td>
<td>3139</td>
<td>1951</td>
<td>3177</td>
</tr>
<tr>
<td>4</td>
<td>Bulgaria</td>
<td>541</td>
<td>1947</td>
<td>601</td>
</tr>
<tr>
<td>5</td>
<td>Bulgaria</td>
<td>2864</td>
<td>1947</td>
<td>1950</td>
</tr>
<tr>
<td>6</td>
<td>Czech</td>
<td>2416</td>
<td>1950</td>
<td>2503</td>
</tr>
</tbody>
</table>

(a) 

<table>
<thead>
<tr>
<th>Country</th>
<th>Harvest</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Albania</td>
<td>1000</td>
<td>1950</td>
</tr>
<tr>
<td>Albania</td>
<td>930</td>
<td>1981</td>
</tr>
<tr>
<td>Austria</td>
<td>3139</td>
<td>1951</td>
</tr>
<tr>
<td>Austria</td>
<td>3177</td>
<td>1955</td>
</tr>
<tr>
<td>Belgium</td>
<td>541</td>
<td>1947</td>
</tr>
<tr>
<td>Belgium</td>
<td>601</td>
<td>1950</td>
</tr>
</tbody>
</table>

(b)
### 2.1. Data Wrangling

**Figure 2.1:** The FlashFill PBE technology, released in Excel 2013, can automate syntactic string transformations. Once the user provides one instance of the transformation (row 2, col. B) and proceeds to transforming another instance (row 3, col. B), FlashFill synthesizes an intended program and applies it to the remaining rows to populate col. B.

<table>
<thead>
<tr>
<th>Email</th>
<th>Column 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="mailto:Nancy.FreHafer@fourthcoffee.com">Nancy.FreHafer@fourthcoffee.com</a></td>
<td>nancy freehafer</td>
</tr>
<tr>
<td><a href="mailto:Andrew.Cencio@northwindtraders.com">Andrew.Cencio@northwindtraders.com</a></td>
<td>andrew cencio</td>
</tr>
<tr>
<td><a href="mailto:Jan.Kotas@litwareinc.com">Jan.Kotas@litwareinc.com</a></td>
<td>jan kotas</td>
</tr>
<tr>
<td><a href="mailto:Mariya.Sergenko@gradiodesignstitute.com">Mariya.Sergenko@gradiodesignstitute.com</a></td>
<td>mariya sergenko</td>
</tr>
<tr>
<td><a href="mailto:Steven.Thorpe@northwindtraders.com">Steven.Thorpe@northwindtraders.com</a></td>
<td>steven thorpe</td>
</tr>
<tr>
<td><a href="mailto:Michael.Neiper@northwindtraders.com">Michael.Neiper@northwindtraders.com</a></td>
<td>michael neiper</td>
</tr>
<tr>
<td><a href="mailto:Robert.Zare@northwindtraders.com">Robert.Zare@northwindtraders.com</a></td>
<td>robert zare</td>
</tr>
<tr>
<td><a href="mailto:Laura.Giussani@adventure-works.com">Laura.Giussani@adventure-works.com</a></td>
<td>laura giussani</td>
</tr>
<tr>
<td><a href="mailto:Anne.HL@northwindtraders.com">Anne.HL@northwindtraders.com</a></td>
<td>anne hl</td>
</tr>
<tr>
<td><a href="mailto:Alexander.David@contoso.com">Alexander.David@contoso.com</a></td>
<td>alexander david</td>
</tr>
<tr>
<td><a href="mailto:Kim.Shane@northwindtraders.com">Kim.Shane@northwindtraders.com</a></td>
<td>kim shane</td>
</tr>
<tr>
<td><a href="mailto:Manish.Chopra@northwindtraders.com">Manish.Chopra@northwindtraders.com</a></td>
<td>manish chopra</td>
</tr>
<tr>
<td><a href="mailto:Gerwald.Oberleitner@northwindtraders.com">Gerwald.Oberleitner@northwindtraders.com</a></td>
<td>gerwald oberleitner</td>
</tr>
<tr>
<td><a href="mailto:Amr.Zaki@northwindtraders.com">Amr.Zaki@northwindtraders.com</a></td>
<td>amr zaki</td>
</tr>
<tr>
<td><a href="mailto:Yvonne.McKay@northwindtraders.com">Yvonne.McKay@northwindtraders.com</a></td>
<td>yvonne mckay</td>
</tr>
<tr>
<td><a href="mailto:Amanda.Pinto@northwindtraders.com">Amanda.Pinto@northwindtraders.com</a></td>
<td>amanda pinto</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>123.4567</td>
<td>123.46</td>
</tr>
<tr>
<td>123.4</td>
<td>123.40</td>
</tr>
<tr>
<td>78.234</td>
<td>78.23</td>
</tr>
</tbody>
</table>

**Figure 2.2:** Sample number and date transformations that can be automated using PBE: (a) Rounding to two decimal places, (c) Nearest lower half hour, (d) Formatting dates to a consistent format. (b) shows the format descriptors in different programming languages required to perform the rounding transformation in (a).

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0d 5h 26m</td>
<td>5:00</td>
</tr>
<tr>
<td>0d 4h 57m</td>
<td>4:30</td>
</tr>
<tr>
<td>0d 4h 27m</td>
<td>4:00</td>
</tr>
<tr>
<td>0d 3h 57m</td>
<td>3:30</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>08/21/2010</td>
<td>08/21/2010</td>
</tr>
<tr>
<td>07/24/2010</td>
<td>07/24/2010</td>
</tr>
<tr>
<td>20/08/2010</td>
<td>08/20/2010</td>
</tr>
<tr>
<td>23/08/2010</td>
<td>08/23/2010</td>
</tr>
<tr>
<td>2010-06-07</td>
<td>07/06/2010</td>
</tr>
<tr>
<td>2010-24-08</td>
<td>08/24/2010</td>
</tr>
</tbody>
</table>
**Applications**

<table>
<thead>
<tr>
<th>Input $v_1$</th>
<th>Input $v_2$</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stroller</td>
<td>10/12/2010</td>
<td>$145.67 + 0.30 \times 145.67$</td>
</tr>
<tr>
<td>Bib</td>
<td>23/12/2010</td>
<td>$3.56 + 0.45 \times 3.56$</td>
</tr>
<tr>
<td>Diapers</td>
<td>21/1/2011</td>
<td>$21.45 + 0.35 \times 21.45$</td>
</tr>
<tr>
<td>Wipes</td>
<td>2/4/2009</td>
<td>$5.12 + 0.40 \times 5.12$</td>
</tr>
<tr>
<td>Aspirator</td>
<td>23/2/2010</td>
<td>$2.56 + 0.30 \times 2.56$</td>
</tr>
</tbody>
</table>

**Figure 2.2**: A semantic string transformation that requires performing syntactic manipulations on multiple lookup results. The goal is to compute the selling price of an item (Output) from its name (Input $v_1$) and selling date (Input $v_2$) using the MarkupRec and CostRec tables. The selling price of an item is computed by adding its purchase price (for the corresponding month) to its markup charges, which in turn is calculated by multiplying the markup percentage by the purchase price. Such transformations can be inferred by examples [127].

**Figure 2.4**: PBE can be used for column splitting: (a) input data column, (b) output columns.
Figure 2.6: FlashExtract enables tabular data extraction from text/log files and web pages using examples. Once the user highlights one or two examples of each field in a different color (in the text file on the left side), FlashExtract extracts more such instances and arranges them in a structured data format (table on the right side).

Figure 2.5: FlashRelate can transform the semi-structured table (a) into the output structured table (b) once the user provides a couple of examples of tuples in the output table, for instance, the ones highlighted in orange and green, respectively.
Solution Space

Ease of Use

Expressiveness
Program Synthesis: “The Ultimate Dream” of CS

Programming Language

User Intent

Search Algorithm

Program
## Query Optimization

**Goal:**

Declarative SQL query → Imperative query execution plan:

```
SELECT S.buyer
FROM Purchase P, Person Q
WHERE P.buyer = Q.name AND
  Q.city = 'seattle' AND
  Q.phone > '5430000'
```

**Inputs:**
- the query
- statistics about the data (indexes, cardinalities, selectivity factors)
- available memory

**Ideally:** Want to find best plan. **Practically:** Avoid worst plans!
Case for Data Wrangling.

99% of spreadsheet users do not know programming.

Data scientists spend 80% time extracting & cleaning data.
Case for Data Wrangling.

Input-Output Examples

- *input state* $\sigma$  $\implies$  *output value* $out$

  "206-279-6261"  $\implies$  "(206) 279-6261"

  "415.413.0703"  $\implies$  "(415) 413-0703"

  "(646) 408 6649"  $\implies$  "(646) 408-6649"
Milestones

A Case for Programming by Examples
- Focus on Data Wrangling.
Key Challenges

 мир

Domain Specific Language (DSL) + Version Space Algebra (VSA)

Programming By Examples Framework

Programming Language

Search Algorithm

User Intent

Program

Intractable program space

Diversity of users intent
PBE 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

Excel, Exchange, Convert-String, ConvertFrom-String, Microsoft Log Analytics
A domain-specific language (DSL) is a computer language specialized to a particular application domain. Eg. spreadsheet formulas and macros.

A synthesis problem is defined for a given domain specific language (DSL) L.

A DSL is specified as a context free grammar (CFG)

A CFG describing strings of letters with the word "main" somewhere in the string:

```
<program>  -->  <letter*> m a i n <letter*>  
<letter*>  -->  <letter> <letter*> | epsilon  
<letter>   -->  A | B | ... | Z | a | b ... | z
```
Domain Specific Language (DSL)

```plaintext
language FlashFill;

@output string start := e | std.ITE(cond, e, start);
string e := f | Concat(f, e);
string f := ConstStr(w)
    | let string x = std.Kth(vs, k) in sub;

string sub := SubStr(x, pp);
Tuple<int, int> pp := std.Pair(pos, pos);
int pos := AbsPos(x, k) | RegPos(x, rr, k);
Tuple<Regex, Regex> rr := std.Pair(r, r);

bool cond := let string s = std.Kth(vs, k) in b;
@extern[std.text.match] bool b;  // FV(b) = {s: string}
@input string[] vs;  string w;  int k;  Regex r;
```

**Figure 1:** FlashFill DSL $L_{FF}$ for string transformations in spreadsheets [4]. Each program rooted at `start` takes an input a spreadsheet row `vs` and performs a chain of `if-elseif` matches on some cells of `vs`. The expression in the chosen `ITE` branch returns a concatenation of constants and input substrings.

- Start, End
- `concatenate(stringA, stringB)
- `condition_on_char(condition, stringA)
- `substring(stringA, start, end)
- `kth Elem(stringA, k)
A version space algebra (VSA) is a data structure for efficient storage of candidate programs in deductive synthesis. Operations like union, intersection, Top_h ($\tilde{\mathcal{N}}$, k), and projection (filtering) is permitted.
Figure 4: A VSA representing all possible programs output the string “425” in $\mathcal{L}_{\text{FF}}$. Leaf nodes $\tilde{e}_1, \tilde{e}_2, \tilde{e}_3, \tilde{e}_{12}, \tilde{e}_{23}$ are VSAs representing all ways to output substrings “4”, “2”, “5”, “42”, and “25” respectively (not expanded in the figure).
Inductive Synthesis Problem

- \( P \) = Program
- \( \tilde{N} \) = Set of Programs \{P1, P2, ..., Pn\}
- \( \mathcal{L} \) = Domain Specific Language (DSL)
- \( \sigma \rightarrow \psi \) = Input output constraint (i.e. o/p of the program for i/p \( \sigma \) follows constraint \( \psi \))
- \( \varphi \) = Collection of input output examples / constraints. (Specification)
- PBE and inductive synthesis used interchangeably.

An inductive synthesis problem refers to synthesis of a program set \( \tilde{N} \subset \mathcal{L} \) that is consistent with a given inductive specification \( \varphi \).
Inductive Synthesis Problem

A program $P$ satisfies a spec $\varphi$ (written $P \models \varphi$) iff it satisfies all constraints $\sigma_i \rightsquigarrow \psi_i$ in $\varphi$. A program $P$ satisfies an input-output constraint $\sigma \rightsquigarrow \psi$ iff its output $[P]_\sigma$ on the given input state $\sigma$ satisfies the corresponding constraint predicate $\psi$, i.e. if $\psi([P]_\sigma)$ is true. A program set $\tilde{N}$ is said to be valid w.r.t. a spec $\varphi$ (written $\tilde{N} \models \varphi$) iff all programs $P \in \tilde{N}$ satisfy $\varphi$. 
BackPropagation / Deductive Synthesis

- The main synthesis algorithm in FlashMeta formalism employed for PBE.
- Follows the grammar top-down, applying the principle of divide and conquer.
- At each step, it reduces the synthesis problem to Learn \( (N, \varphi) \)

- Backpropagate constraints on a program \( F(N_1, \ldots, N_k) \) to deduced constraints on its Subexpressions \( N_1, \ldots, N_k \).

- Witness function \( W \) transforms the Spec \( \varphi \) to corresponding spec \( \varphi' \) for \( N' \)

\[
\text{Learn} \ (N, \varphi) \quad \text{Learn} \ (N_1, \varphi_1) \quad \text{Learn} \ (N_2, \varphi_2)
\]

```plaintext
language FlashFill;

@output string start := e | std.IFE(cond, e, start);
string e := f | Concat(f, e);
string f := ConstStr(w)
    | let string x = std.Kth(vs, k) in sub;

string sub := SubStr(x, pp);
Tuple<int, int> pp := std.Pair(pos, pos);
int pos := AbsPos(x, k) | RegPos(x, rr, k);
Tuple<Reg, Reg> rr := std.Pair(r, r);

bool cond := let string s = std.Kth(vs, k) in b;
@extern[std.text.match] bool b; // FV(b) = \{s: string\}
@input string[] vs;
string w; int k; Reg r;
```
Ranking Function

- The main idea of ranking is to assign a likelihood score to each program in the set of programs, induced from a small set of input-output examples, such that the programs with the highest scores correspond to the desired user-intended programs.

- In simple words (converts a set of programs to an ordered list, the top of the list is most likely to be the desired program)

- Can be designed manually [51, 84, 90, 106]
  Or
- Generated using Machine learning [128, 30]
Milestones

A Case for Programming by Examples
- Focus on Data Wrangling.

Defining Program Synthesis
- Key Challenges
- Background
  - Domain Specific Language (DSL)
  - Inductive Synthesis Problem
  - Version Space Algebra (VSA)
  - Ranking Function & Backprop
PBE Architecture

Figure 6: Learner-user communication in conventional PBE.
Hurdles to Mass Market

- The performance of the Synthesizer.
  - Cannot wait more than 1-2 seconds per round.

- The correctness of Synthesized program.
  - Must capture a large class of tasks.
Inspirations Towards Interactivity.

- Incremental: (one logic at a time)
- Step-based: (One component at a time)
- Feedback-based: (Evolving codebase)

```python
class Oval:
    def __init__(self, p1, p2):
        self.p1 = p1
        self.p2 = p2

    def __repr__(self):
        return "Oval({}, {})".format(str(self.p1), str(self.p2))

    def clone(self):
        other = Oval(self.p1, self.p2)
        other.config = self.config.copy()
        return other

    def _draw(self, canvas, options):
        p1 = self.p1
        p2 = self.p2
        x1, y1 = canvas.toScreen(p1.x, p1.y)
        x2, y2 = canvas.toScreen(p2.x, p2.y)
        return canvas.create_oval(x1, y1, x2, y2, options)
```
Inspirations: Incrementality

Incremental: (one logic at a time)

Observation on each iteration:
- Search over the same program space \( N \in \mathcal{L} \).
- Growing number of constraints \( \varphi \).
- Decreasing number of output programs \( \tilde{N} \).

What if I could update the DSL \( \mathcal{L} \) after each iteration?
- May be a sub-DSL?

**Figure 6:** Learner-user communication in conventional PBE.
Inspirations: Incrementality

Incremental: (one logic at a time)

Observation on each iteration:

- Search over the same program space $N \in \mathcal{L}$.
- Growing number of constraints $\Phi$
- Decreasing number of output programs $\bar{N}$

What if I could update the DSL $\mathcal{L}$ after each iteration?
- May be a sub-DSL?

Figure 6: Learner-user communication in conventional PBE.
Inspirations: Step-based formulation

Step-based: (One component at a time)

Observation on each iteration:
- Black box model (DSL and ranking function)
- Dependence on counter example
- Whole program to be analyzed for generating Counter-example.

What if I could focus on a part of the program for sub expressions and fine tune that. Then move to next?

**Figure 6:** Learner-user communication in conventional PBE.
Inspirations: Step-based formulation

Step-based: (One component at a time)

Observation on each iteration:
- Black box model (DSL and ranking function)
- Dependence on counter example
- Whole program to be analyzed for generating Counter-example.

What if I could focus on a part of the program for sub expressions and fine tune that. Then move to next?

**Figure 6:** Learner-user communication in conventional PBE.
Inspirations: Feedback based interaction

Feedback-based: (Evolving codebase)

Observation on each iteration:
- No clear understanding of full spec $\varphi$.
- Lots of ambiguous programs $\tilde{N}$.
- Certain constraint disambiguates the program set more than the other.

What if I could find the right set of next constraints that shrinks the DSL themost. (say a hypothesizer)

---

**Figure 6:** Learner-user communication in conventional PBE.
Inspirations: Feedback based interaction

Feedback-based: (Evolving codebase)

Observation on each iteration:
- No clear understanding of full spec $\varphi$.
- Lots of ambiguous programs $\mathcal{N}$.
- Certain constraint disambiguates the program set more than the other.

What if I could find the right set of next constraints that shrinks the DSL themost. (say a hypothesizer)

---

**Figure 6:** Learner-user communication in conventional PBE.
Inspirations Towards Interactivity.

- Incremental: (one logic at a time)
  - Sub-DSL search
- Step-based: (One component at a time)
  - Sub Expression Constraints
- Feedback-based: (Evolving codebase)
  - Best resolve ambiguity using Hypothesizer to generate questions and answering them

**Figure 6:** Learner-user communication in conventional PBE.
Interactive PBE Architecture

Program ranking function $h(P)$  Program Synthesis Framework

Domain-specific language $\mathcal{L}$

Best program

Ranked set of valid programs $\mathcal{S}$

Hypothesizer

Refined intent

Intent spec $\varphi$: input-output examples

Refined DSL $\mathcal{L}' \subset \mathcal{L}$ induced by $\mathcal{S}$

User (Debugging)

Test inputs

Deployable code in Python/R/C#/

Translator

Learner

Figure 7: Learner-user communication in interactive PBE.
Milestones

A Case for Programming by Examples
- Focus on Data Wrangling.

Defining Program Synthesis
- Key Challenges
- Background
  - Domain Specific Language (DSL)
  - Inductive Synthesis Problem
  - Version Space Algebra (VSA)

Hurdles to Mass Market
- Performance & Correctness

3 Key Inspirations towards Interactivity (Partial)
- Incremental : (one logic at a time)
- Step-based : (One component at a time)
- Feedback-based : (Evolving codebase)
1. Incremental Synthesis

VSA as a DSL

- VSA $\tilde{N}$ is a DAG-like program set representation.
- It is simply an AST-based representation of a sub-DSL $\mathcal{L}' \subset \mathcal{L}$.

```
function VSAtoDSL(VSA $\tilde{N}$)
1: Let $V$ be a set of fresh nonterminals, one per each non-leaf node in $\tilde{N}$
2: Let $\Sigma$ be a set of fresh terminals, one per each leaf node in $\tilde{N}$
3: // We write $\text{sym}(\tilde{N}') \in V \cup \Sigma$ to denote the corresponding fresh symbol for a node $\tilde{N}'$ from $\tilde{N}$
4: Productions $R \leftarrow \emptyset$
5: // Create “symbol := symbol” productions for all union nodes
6: for all union nodes $\tilde{N}' = U(\tilde{N}_1, \ldots, \tilde{N}_k)$ in $\tilde{N}$ do
7:     $R \leftarrow R \cup \{\text{sym}(\tilde{N}') := \text{sym}(\tilde{N}_i) \mid i = 1 \ldots k\}$
8: // Create operator productions for all join nodes
9: for all join nodes $\tilde{N}' = F_{\otimes}(\tilde{N}_1, \ldots, \tilde{N}_k)$ in $\tilde{N}$ do
10:    $R \leftarrow R \cup \{\text{sym}(\tilde{N}') := F(\text{sym}(\tilde{N}_1), \ldots, \text{sym}(\tilde{N}_k))\}$
11: // Annotate terminal symbols with values extracted from leaf nodes
12: for all leaf nodes $\tilde{N}' = \{P_1, \ldots, P_k\}$ in $\tilde{N}$ do
13:     Annotate in $\Sigma$ that $\text{sym}(\tilde{N}') \in \{P_1, \ldots, P_k\}$
14: return the context-free grammar $G = (V, \Sigma, R, \text{sym}(\tilde{N}))$
```

```
string sub := SubStr(x, pp);
Tuple<int, int> pp := std.Pair(pos, pos);
int pos := AbsPos(x, k) | RegPos(x, rr, k);
Tuple<RegEx, RegEx> rr := std.Pair(r, r);
```
1. Incremental Synthesis

Constraint Resolution

- Definitive constraints
  
  Constructively define a subset of DSL by their own. (irrespective of witness function)

- Example constraint: “output = v”,
- Membership constraint: “output ∈ \{v_1, v_2, v_3\}”,
- Prefix constraint: “output = [v_1, v_2, ...]”,
- Subset/subsequence constraint: “output ⊇ [v_1, v_2, v_3]”.
1. Incremental Synthesis

Constraint Resolution

- Definitive constraints
  Constructively define a subset of DSL by their own. (irrespective of witness function)

- Locally refining constraints
  Do not define a subset of DSL on their own. But can trim/refine the DSL. (only relevant to select witness functions.)

- Datatype constraint: “output: $\tau$”. Eliminates all top-level programs rooted at any type-incompatible DSL operators.
1. **Incremental Synthesis**

**Constraint Resolution**

- **Definitive constraints**
  Constructively define a subset of DSL by their own. (irrespective of witness function)

- **Locally refining constraints**
  Do not define a subset of DSL on their own. But can trim/refine the DSL. (only relevant to select witness functions.)

- **Globally refining constraints**
  Do not define a subset of DSL on their own and do not permit any local refining logic.

- *Negative example constraint*: “output ≠ v”,
- *Negative membership constraint*: “output ∉ v”.
2. Step-based Synthesis

**Definition 3** (Compound DSL). A DSL $\mathcal{L}$ is called a **compound DSL** if it includes any *extern* nonterminals $N_1, \ldots, N_m$, which resolve to output symbols of some *sub-DSLs* $\mathcal{L}_1, \ldots, \mathcal{L}_m$ respectively.

**Definition 4** (Constraints on named subexpressions). Given a compound DSL $\mathcal{L}$, a *named constraint* $\psi^e : N$ is specified for an extern nonterminal $N$ in $\mathcal{L}$. Its meaning is as follows:

“There exists a subexpression rooted at $N$ in the desired compound program. We mark it with ID $e$. This subexpression must satisfy the constraint $\psi$.”

When used in a context of iterative learner-user interaction on a larger task, all named constraints with the same ID $e$ apply to the same subexpression in the desired compound program in $\mathcal{L}$.

**Named specs** $\varphi^e : N$ are defined similarly as conjunctions of named constraints on $e$. 
3. Feedback-based Synthesis: Q -> A

- Proposed learner-user interaction model that leverages proactive feedback in the form of queries to the user. (Ask questions -> get Answers -> convert to constraints)

\[ \mathcal{L} = \text{Domain Specific Language (DSL)} \]

\[ \mathcal{C} = \text{all top-level constraint types supported by synthesizer} \]

\[ C \in \mathcal{C}, C \text{ is a top-level constraint type.} \]

\[ \psi = \text{instance}_\text{of}(C) \text{ is a constraint.} \]

For each \( C \in \mathcal{C} \) We associate a descriptive "question q" such that a "response r" for this question directly constitutes \( \psi = \text{instance}_\text{of}(C) \)
3. **Feedback-based Synthesis : Disamb Score**

- Which questions to ask?
- To evaluate a question’s effectiveness, the hypothesizer is parameterized with a disambiguation score function \( \text{get\_disambiguation\_score}(q, \tilde{N}, \varphi) \).
- Higher the score, greater the number of ambiguity resolved.
- Since response \( r \) is unknown, function represents the potential effectiveness.
- May be domain-specific or general.

**Case Study:**
General purpose disambiguation score function.

\[
\text{ds}_R(q, \tilde{N}, \varphi) \overset{\text{def}}{=} \min_{r \in R(q)} \max_{P \in \tilde{N}_r} h(P)
\]

Higher if every response for the question \( q \) leads to a higher-ranked alternative program.
3. Feedback-based Synthesis

**procedure** DISAMBIGUATE(Candidate programs $\tilde{N}$, current spec $\varphi$)

1: Analyze the ambiguities in $\tilde{N}$ w.r.t. $\varphi$.
   Let $Q$ be a set of questions that may resolve ambiguity in $\tilde{N}$

2: $q^* \leftarrow \operatorname{argmax}_{q \in Q} ds(q, N, \varphi)$ \hspace{1em} // Compare the disa of all questions

3: if $ds(q^*, \tilde{N}, \varphi) < \text{threshold } T$ then
   break

4: else
   Present the question $q^*$ to the user

5: Let $r$ be the user’s response to $q$
6: Let $\psi$ be the response $r$ converted into a constraint
7: return $\psi$ to the learner and invoke a new round of synthesis

**Figure 9:** The hypothesizer’s proactive disambiguation algorithm.
PBE Architecture

Program ranking function $h(P)$

Program Synthesis Framework

Domain-specific language $\mathcal{L}$

Best program

Ranked set of valid programs $\mathcal{S}$

Refined intent

Hypothesizer

User (Debugging)

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

Translator

Deployable code in Python/R/C#/

Figure 7: Learner-user communication in interactive PBE.
Milestones

A Case for Programming by Examples
- Focus on Data Wrangling.

Defining Program Synthesis
- Key Challenges
- Background
  - Domain Specific Language (DSL)
  - Inductive Synthesis Problem
  - Version Space Algebra (VSA)

Hurdles to Mass Market
- Performance & Correctness

3 Key Inspirations towards Interactivity
- Incremental: (one logic at a time)
- Step-based: (One component at a time)
- Feedback-based: (Evolving codebase)
Evaluation - Incremental Synthesis

(a) Speedup on the FlashFill DSL.
(b) Speedup on the FlashExtract DSL.

Figure 10: Speedups obtained by the incremental synthesis algorithm vs. the non-incremental algorithm. Values higher above the $y = 1$ line (where the runtimes are equal) are better.
Evaluation - Step-based Synthesis

- **Non Step-based**: The baseline of this evaluation is a non-interactive FlashExtract where the user has to provide examples for all the fields at once.
- **Step-based**: The user of this system extracts fields in topological order (i.e., from top-level fields to leaf fields)

![Graph showing average number of interactions per field across all benchmarks](image)

**Figure 11**: The average number of interactions per field across all benchmarks. Lower is better.
Evaluation - Feedback-based Synthesis

- **FlashSplit** We evaluate the feedback-driven synthesis for Flash-Split on a set of 77 splitting tasks on different log files.

- **Binary position questions.** A binary position question $q \in Q_b$ presents a single position in the input row and asks if it is a desired splitting point.

- **Confirmation questions.** A confirmation question $q \in Q_c$ presents a set of positions to the user and asks whether all of these positions are valid splitting points.

- **Baseline setting.** Split position examples are provided randomly until the splitting is correct.

- **BinaryQ** One random example is provided, after which the system keeps asking binary position questions until the correct program is achieved.

- **ConfirmationQ** One random example is provided by the user, after which the system poses a confirmation question if one exists.

- **CombinedQ** The system uses a combination of binary and confirmation questions, using the disambiguation score $dsFS$.
Evaluation - Feedback-based Synthesis

Figure 12: Comparison of effectiveness of different example provision strategies for field splitting tasks from log files. Lower is better.

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Avg. number of inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>8.81</td>
</tr>
<tr>
<td>BinaryQ</td>
<td>8.54</td>
</tr>
<tr>
<td>ConfirmationQ</td>
<td>6.98</td>
</tr>
<tr>
<td>CombinedQ</td>
<td>6.90</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Feedback</th>
<th>Baseline</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>241</td>
<td>50</td>
<td>16</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>75</td>
<td>30</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>7</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 1: Number of rows inspected in the baseline and the feedback driven settings for FlashFill evaluation.
Milestones

A Case for Programming by Examples
- Focus on Data Wrangling.

Defining Program Synthesis
- Key Challenges
- Background
  - Domain Specific Language (DSL)
  - Inductive Synthesis Problem
  - Version Space Algebra (VSA)

Hurdles to Mass Market
- Performance & Correctness

3 Key Inspirations towards Interactivity
- *Incremental*: (one logic at a time)
- *Step-based*: (One component at a time)
- *Feedback-based*: (Evolving codebase)

Results / Ablation Study.

Thank you!
Questions . . .