Interactive Program Synthesis

The big picture

What if your excel was smart ?

	A	B
1	Email	Column 2
2	Nancy.FreeHafer@fourthcoffee.com	nancy freehafer
3	Andrew.Cencici@northwindtraders.com	andrew cencici
4	Jan.Kotas@litwareinc.com	jan kotas
5	Mariya.Sergienko@gradicdesigninstitute.com	mariya sergienko
6	Steven.Thorpe@northwindtraders.com	steven thorpe
7	Michael.Neipper@northwindtraders.com	michael neipper
8	Robert.Zare@northwindtraders.com	robert zare
9	Laura.Giussani@adventure-works.com	laura giussani
10	Anne.HL@northwindtraders.com	anne hl
11	Alexander.David@contoso.com	alexander david
12	Kim.Shane@northwindtraders.com	kim shane
13	Manish.Chopra@northwindtraders.com	manish chopra
14	Gerwald.Oberleitner@northwindtraders.com	gerwald oberleitner
15	Amr.Zaki@northwindtraders.com	amr zaki
16	Yvonne.McKay@northwindtraders.com	yvonne mckay
17	Amanda.Pinto@northwindtraders.com	amanda pinto

Input	Output
123.4567	123.46
123.4	123.40
78.234	78.23

(a)

Language	Format descriptor for rounding to two decimal places
Excel, C#	#.00
Python, C	.2f
Java	#.##

(b)

Input	Output
0d 5h 26m	5:00
0d 4h 57m	4:30
0d 4h 27m	4:00
0d 3h 57m	3:30

(c)

Input	Output
08/21/2010	08/21/2010
07/24/2010	07/24/2010
20.08.2010	08/20/2010
23.08.2010	08/23/2010
2010-06-07	07/06/2010
2010-24-08	08/24/2010

(d)

Input v_1	**Input** v_2	**Output**
Stroller	10/12/2010	$145.67+0.30*145.67
Bib	23/12/2010	$3.56+0.45*3.56
Diapers	21/1/2011	$21.45+0.35*21.45
Wipes	2/4/2009	$5.12+0.40*5.12
Aspirator	23/2/2010	$2.56+0.30*2.56

MarkupRec

Id	**Name**	**Markup**
S30	Stroller	30%
B56	Bib	45%
D32	Diapers	35%
W98	Wipes	40%
A46	Aspirator	30%
...	...	...

CostRec

Id	**Date**	**Price**
S30	12/2010	$145.67
S30	11/2010	$142.38
B56	12/2010	$3.56
D32	1/2011	$21.45
W98	4/2009	$5.12
A46	2/2010	$2.56
...	...	...

```
191.128.19.55 - - [09/Jun/2016:18:05:33 -0800] "GET /checks.txt
174.13.04.3 - - [09/Jun/2016:19:43:23 -0800] "GET /images/pic.png
192.16.201.109 - - [10/Jun/2016:06:10:03 -0800] "GET /pdf/document.pdf
11.0.4.50 - - [10/Jun/2016:16:10:02 -0800] "GET /index.html
191.169.12.13 - - [11/Jun/2016:11:10:02 -0800] "GET /index.html
172.18.0.102 - - [11/Jun/2016:16:12:34 -0800] "GET /logs/access.log
192.19.2.100 - - [11/Jun/2016:17:32:36 -0800] "GET /index.html
10.129.2.78 - - [11/Jun/2016:17:45:38 -0800] "GET /data/2/4
171.19.3.12 - - [11/Jun/2016:22:12:01 -0800] "GET /data/
191.168.125.112 - - [11/Jun/2016:23:12:52 -0800] "GET /pictures/pic2.png
174.26.0.223 - - [12/Jun/2016:16:13:04 -0800] "GET /images/pic4.gif
175.16.0.24 - - [12/Jun/2016:17:29:33 -0800] "GET /index.html
196.168.29.105 - - [12/Jun/2016:18:33:11 -0800] "GET /styles.css
101.22.54.38 - - [13/Jun/2016:20:32:43 -0800] "GET /js/scripts.js
```

(a)

191.128.19.55	- - [	09/Jun/2016	:	18:05:33	-	0800	] "	GET	/	checks.txt
174.13.04.3	- - [	09/Jun/2016	:	19:43:23	-	0800	] "	GET	/	images/pic.png
192.16.201.109	- - [	10/Jun/2016	:	06:10:03	-	0800	] "	GET	/	pdf/document.pdf
11.0.4.50	- - [	10/Jun/2016	:	16:10:02	-	0800	] "	GET	/	index.html
191.169.12.13	- - [	11/Jun/2016	:	11:10:02	-	0800	] "	GET	/	index.html
172.18.0.102	- - [	11/Jun/2016	:	16:12:34	-	0800	] "	GET	/	logs/access.log
192.19.2.100	- - [	11/Jun/2016	:	17:32:36	-	0800	] "	GET	/	index.html
10.129.2.78	- - [	11/Jun/2016	:	17:45:38	-	0800	] "	GET	/	data/2/4
171.19.3.12	- - [	11/Jun/2016	:	22:12:01	-	0800	] "	GET	/	data/
191.168.125.112	- - [	11/Jun/2016	:	23:12:52	-	0800	] "	GET	/	pictures/pic2.png
174.26.0.223	- - [	12/Jun/2016	:	16:13:04	-	0800	] "	GET	/	images/pic4.gif
175.16.0.24	- - [	12/Jun/2016	:	17:29:33	-	0800	] "	GET	/	index.html
196.168.29.105	- - [	12/Jun/2016	:	18:33:11	-	0800	] "	GET	/	styles.css
101.22.54.38	- - [	13/Jun/2016	:	20:32:43	-	0800	] "	GET	/	js/scripts.js

(b)

Ana Trujillo
357 21th Place SE
Redmond, WA
(757) 555-1634

Antonio Moreno
515 93th Lane
Renton, WA
(411) 555-2786

Thomas Hardy
742 17th Street NE
Seattle, WA
(412) 555-5719

Christina Berglund
475 22th Lane
Redmond, WA
(443) 555-6774

Hanna Moos
785 45th Street NE
Puyallup, WA
(376) 555-2462

Frederique Citeaux
308 66th Place
Redmond, WA
(689) 555-2770

(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)

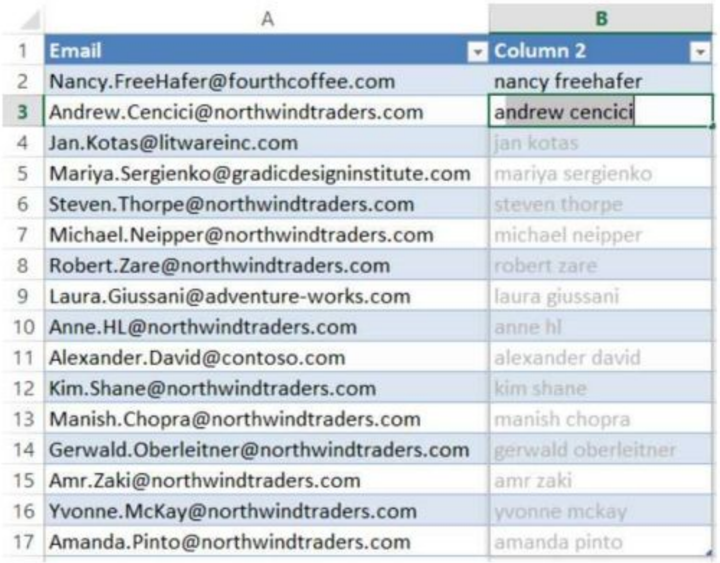
	A	B	C	D	E
1		value	year	value	year
2	Albania	1000	1950	930	1981
3	Austria	3139	1951	3177	1955
4	Belgium	541	1947	601	1950
5	Bulgaria	2964	1947	1959	1958
6	Czech ...	2416	1950	2503	1960

...

(a)

Country	Harvest	Date
Albania	1000	1950
Albania	930	1981
...		
Austria	3139	1951
Austria	3177	1955
...		
Belgium	541	1947
Belgium	601	1950
...		

(b)



	A	B
1	Email	Column 2
2	Nancy.FreeHafer@fourthcoffee.com	nancy freehafer
3	Andrew.Cencici@northwindtraders.com	andrew cencici
4	Jan.Kotas@litwareinc.com	jan kotas
5	Mariya.Sergienko@gradicdesigninstitute.com	mariya sergienko
6	Steven.Thorpe@northwindtraders.com	steven thorpe
7	Michael.Neipper@northwindtraders.com	michael neipper
8	Robert.Zare@northwindtraders.com	robert zare
9	Laura.Giussani@adventure-works.com	laura giussani
10	Anne.HL@northwindtraders.com	anne hl
11	Alexander.David@contoso.com	alexander david
12	Kim.Shane@northwindtraders.com	kim shane
13	Manish.Chopra@northwindtraders.com	manish chopra
14	Gerwald.Oberleitner@northwindtraders.com	gerwald oberleitner
15	Amr.Zaki@northwindtraders.com	amr zaki
16	Yvonne.McKay@northwindtraders.com	yvonne mckay
17	Amanda.Pinto@northwindtraders.com	amanda pinto

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.

Input	Output
123.4567	123.46
123.4	123.40
78.234	78.23

(a)

Language	Format descriptor for rounding to two decimal places
Excel, C#	#.00
Python, C	.2f
Java	#.##

(b)

Input	Output
0d 5h 26m	5:00
0d 4h 57m	4:30
0d 4h 27m	4:00
0d 3h 57m	3:30

(c)

Input	Output
08/21/2010	08/21/2010
07/24/2010	07/24/2010
20.08.2010	08/20/2010
23.08.2010	08/23/2010
2010-06-07	07/06/2010
2010-24-08	08/24/2010

(d)

Figure 2.3: 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).

Input v_1	Input v_2	Output
Stroller	10/12/2010	$145.67+0.30*145.67
Bib	23/12/2010	$3.56+0.45*3.56
Diapers	21/1/2011	$21.45+0.35*21.45
Wipes	2/4/2009	$5.12+0.40*5.12
Aspirator	23/2/2010	$2.56+0.30*2.56

MarkupRec

Id	Name	Markup
S30	Stroller	30%
B56	Bib	45%
D32	Diapers	35%
W98	Wipes	40%
A46	Aspirator	30%
...	...	...

CostRec

Id	Date	Price
S30	12/2010	$145.67
S30	11/2010	$142.38
B56	12/2010	$3.56
D32	1/2011	$21.45
W98	4/2009	$5.12
A46	2/2010	$2.56
...	...	...

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].

```
191.128.19.55 - - [09/Jun/2016:18:05:33 -0800] "GET /checks.txt
174.13.04.3 - - [09/Jun/2016:19:43:23 -0800] "GET /images/pic.png
192.16.201.109 - - [10/Jun/2016:06:10:03 -0800] "GET /pdf/document.pdf
11.0.4.50 - - [10/Jun/2016:16:10:02 -0800] "GET /index.html
191.169.12.13 - - [11/Jun/2016:11:10:02 -0800] "GET /index.html
172.18.0.102 - - [11/Jun/2016:16:12:34 -0800] "GET /logs/access.log
192.19.2.100 - - [11/Jun/2016:17:32:36 -0800] "GET /index.html
10.129.2.78 - - [11/Jun/2016:17:45:38 -0800] "GET /data/2/4
171.19.3.12 - - [11/Jun/2016:22:12:01 -0800] "GET /data/
191.168.125.112 - - [11/Jun/2016:23:12:52 -0800] "GET /pictures/pic2.png
174.26.0.223 - - [12/Jun/2016:16:13:04 -0800] "GET /images/pic4.gif
175.16.0.24 - - [12/Jun/2016:17:29:33 -0800] "GET /index.html
196.168.29.105 - - [12/Jun/2016:18:33:11 -0800] "GET /styles.css
101.22.54.38 - - [13/Jun/2016:20:32:43 -0800] "GET /js/scripts.js
```

(a)

191.128.19.55	--[	09/Jun/2016	:	18:05:33	-	0800	]	"	GET	/	checks.txt
174.13.04.3	--[	09/Jun/2016	:	19:43:23	-	0800	]	"	GET	/	images/pic.png
192.16.201.109	--[	10/Jun/2016	:	06:10:03	-	0800	]	"	GET	/	pdf/document.pdf
11.0.4.50	--[	10/Jun/2016	:	16:10:02	-	0800	]	"	GET	/	index.html
191.169.12.13	--[	11/Jun/2016	:	11:10:02	-	0800	]	"	GET	/	index.html
172.18.0.102	--[	11/Jun/2016	:	16:12:34	-	0800	]	"	GET	/	logs/access.log
192.19.2.100	--[	11/Jun/2016	:	17:32:36	-	0800	]	"	GET	/	index.html
10.129.2.78	--[	11/Jun/2016	:	17:45:38	-	0800	]	"	GET	/	data/2/4
171.19.3.12	--[	11/Jun/2016	:	22:12:01	-	0800	]	"	GET	/	data/
191.168.125.112	--[	11/Jun/2016	:	23:12:52	-	0800	]	"	GET	/	pictures/pic2.png
174.26.0.223	--[	12/Jun/2016	:	16:13:04	-	0800	]	"	GET	/	images/pic4.gif
175.16.0.24	--[	12/Jun/2016	:	17:29:33	-	0800	]	"	GET	/	index.html
196.168.29.105	--[	12/Jun/2016	:	18:33:11	-	0800	]	"	GET	/	styles.css
101.22.54.38	--[	13/Jun/2016	:	20:32:43	-	0800	]	"	GET	/	js/scripts.js

(b)

Figure 2.4: PBE can be used for column splitting: (a) input data column, (b) output columns.

Ana Trujillo
357 21th Place SE
Redmond, WA
(757) 555-1634

Antonio Moreno
515 93th Lane
Renton, WA
(411) 555-2786

Thomas Hardy
742 17th Street NE
Seattle, WA
(412) 555-5719

Christina Berglund
475 22th Lane
Redmond, WA
(443) 555-6774

Hanna Moos
785 45th Street NE
Puyallup, WA
(376) 555-2462

Frederique Citeaux
308 66th Place
Redmond, WA
(689) 555-2770

(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)

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).

	A	B	C	D	E
1		value	year	value	year
2	Albania	1000	1950	930	1981
3	Austria	3139	1951	3177	1955
4	Belgium	541	1947	601	1950
5	Bulgaria	2964	1947	1959	1958
6	Czech ...	2416	1950	2503	1960

...

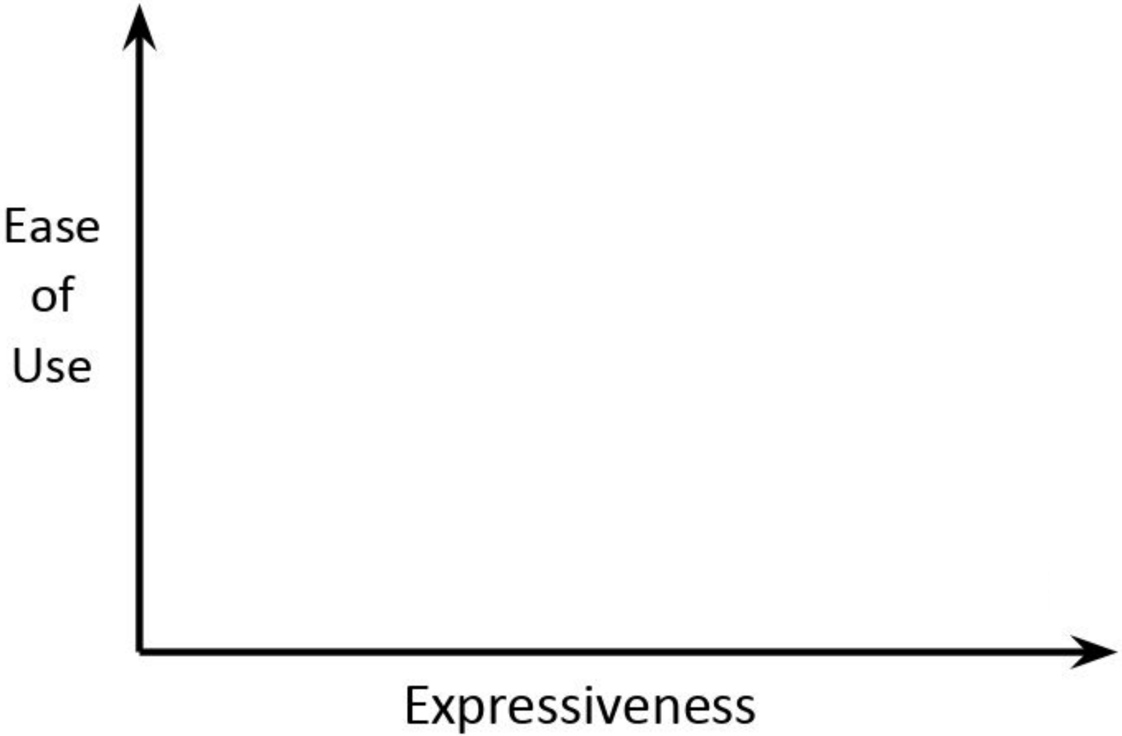
(a)

Country	Harvest	Date
Albania	1000	1950
Albania	930	1981
...		
Austria	3139	1951
Austria	3177	1955
...		
Belgium	541	1947
Belgium	601	1950
...		

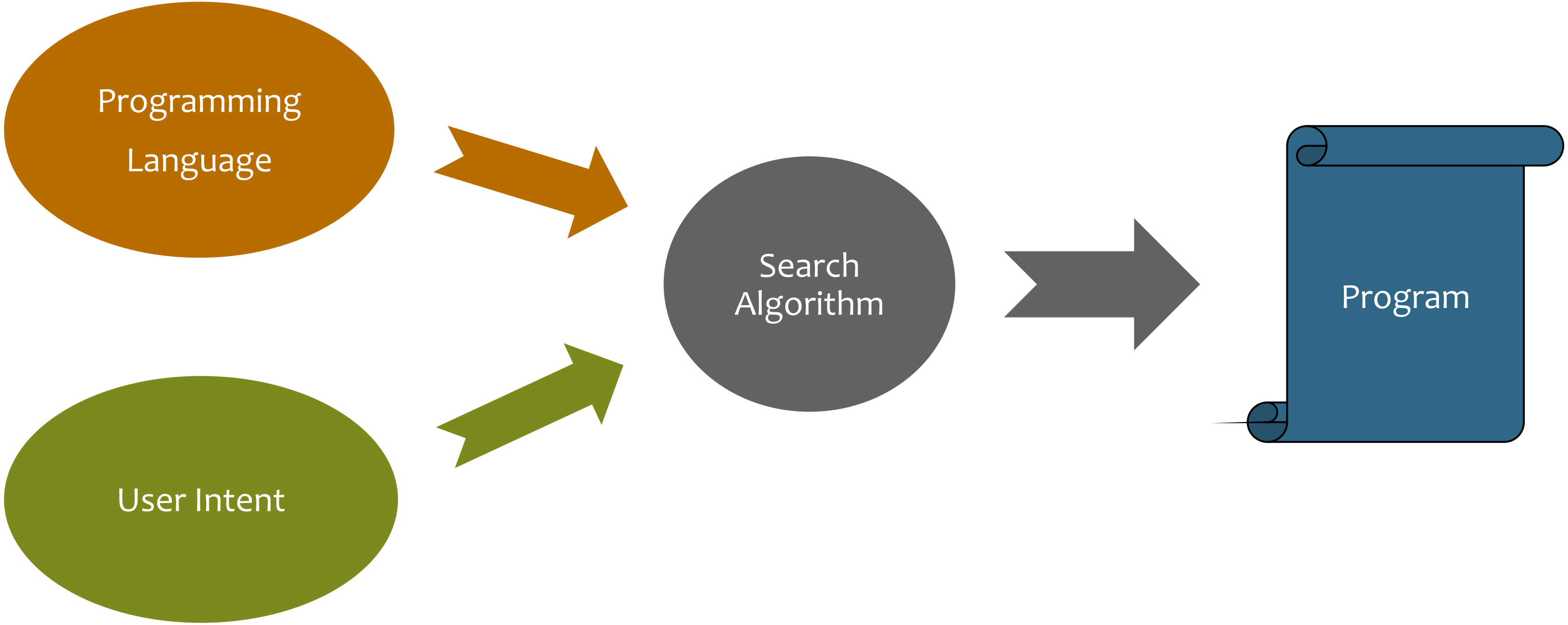
(b)

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



Program Synthesis: “The Ultimate Dream” of CS



Applications

2.1 Data Wrangling . . . . .

2.2 Graphics . . . .

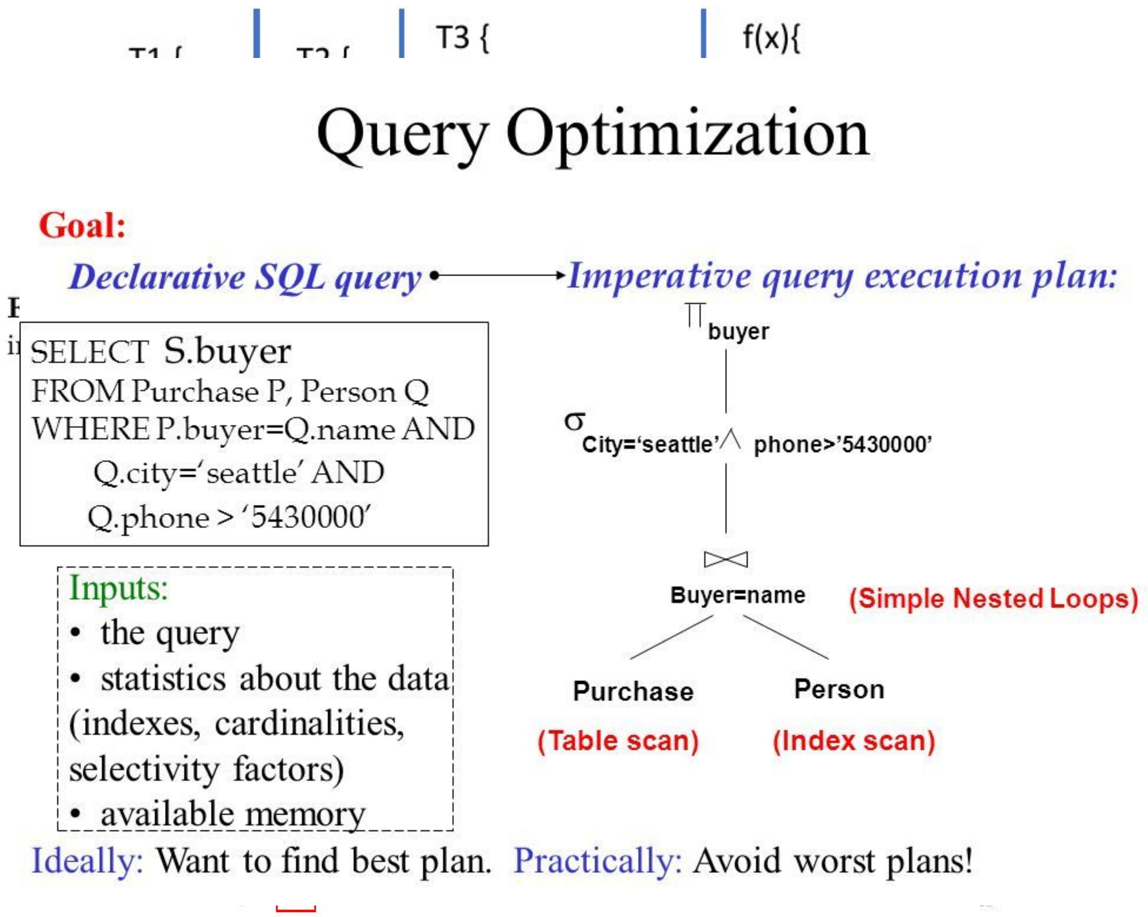
2.3 Code Repair . . . . . .

2.4 Code Suggestions . . . . .

2.5 Modeling . . . .

2.6 Superoptimization . . . .

2.7 Concurrent Programming



Case for Data Wrangling.

99% of spreadsheet
do not know programming

Data scientists spend 80% time extracting & cleaning data

Case for Data Wrangling.

Input-Output Examples

- *input state* σ $\Longrightarrow$ *output value out*

"206-279-6261"	$\Longrightarrow$	"(206) 279-6261"
"415.413.0703"	$\Longrightarrow$	"(415) 413-0703"
"(646) 408 6649"	$\Longrightarrow$	"(646) 408-6649"

Milestones

A Case for Programming by Examples

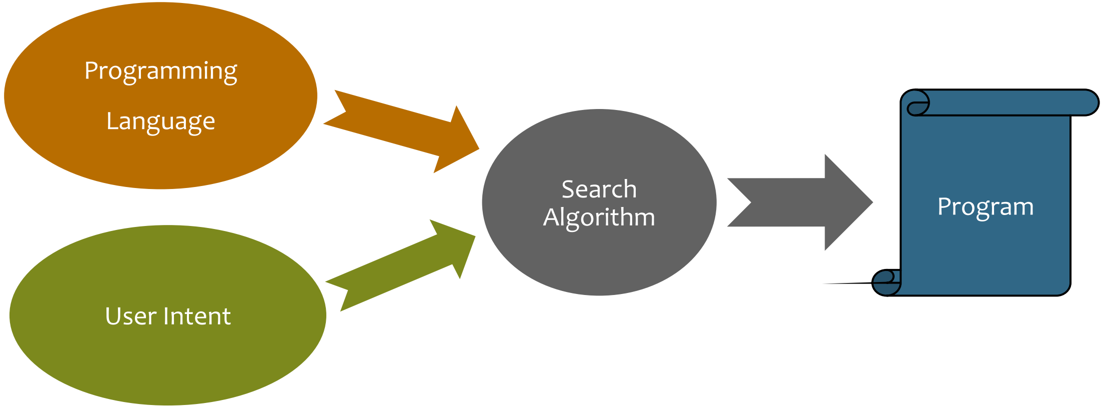
- Focus on Data Wrangling.

Key Challenges

Intractable program space

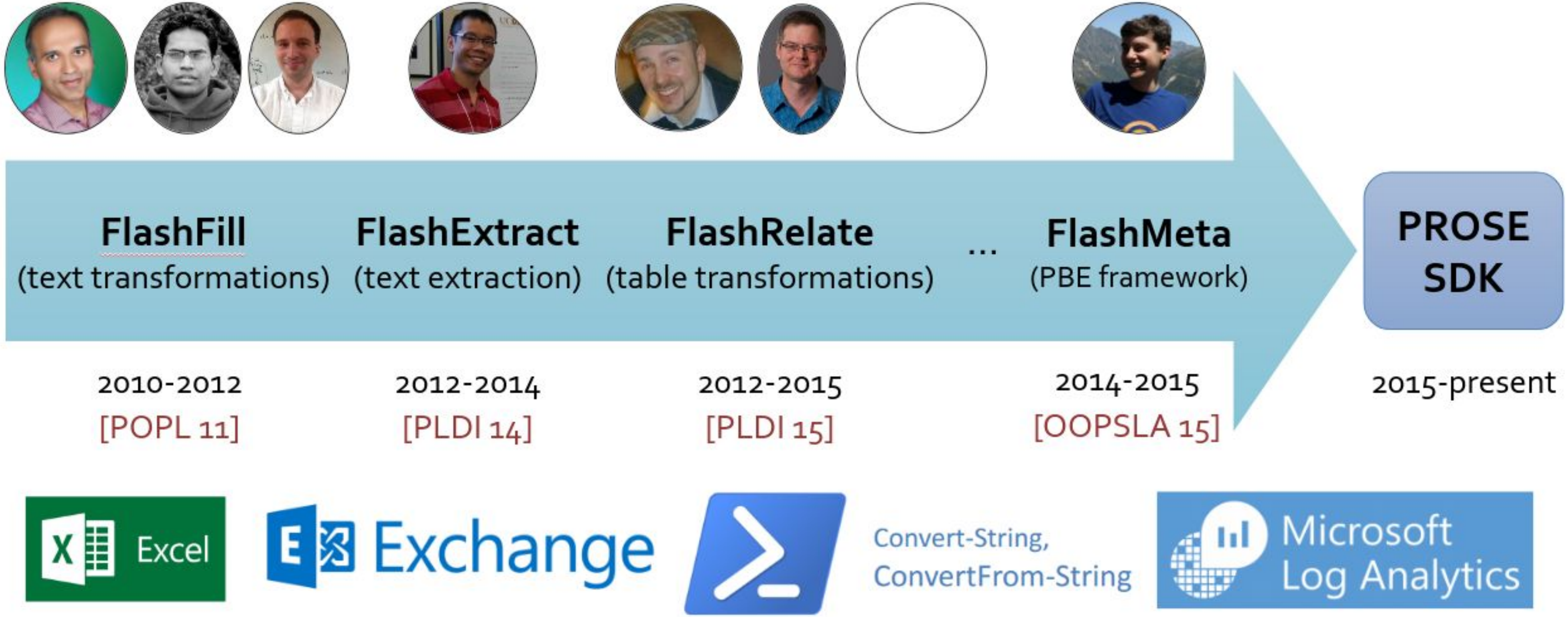
Domain Specific Language (DSL)
+
Version Space Algebra (VSA)

Programming By Examples
Framework



Diversity of users intent

PBE Timeline



Domain Specific Language (DSL)

- 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)

```
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;
```

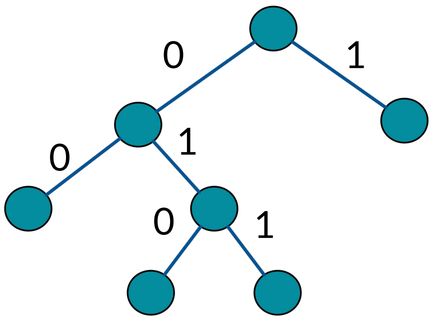
Figure 1: FlashFill DSL $\mathcal{L}_{\mathsf{FF}}$ for string transformations in spreadsheets [4]. Each program rooted at *start* takes an input a spreadsheet row *vs* and performs a chain of `if-elseif` matches on some cells of *vs*. The expression in the chosen `ITE` branch returns a concatenation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

- Start, End
- concatenate(stringA, stringB)
- condition_on_char(condition, stringA)
- substring(stringA, start, end)
- kth_elem(stringA, k)

Version Space Algebra (VSA)

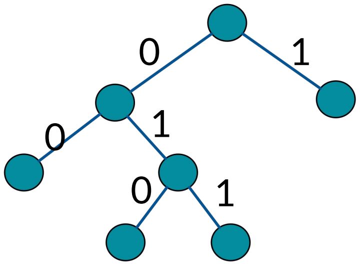
- **P** = Program
- **Ñ** = Set of Programs {P1, P2, ... Pn}
- $\mathcal{L}$ = Domain Specific Language (DSL)

- A version space algebra (VSA) is a data structure for efficient storage of candidate programs in deductive synthesis.
- Operations like union , intersection, Top_h (**Ñ**, k), and projection(filtering) is permitted



1	0
2	1
3	01
4	00
5	011
6	010

Version Space Algebra (VSA)



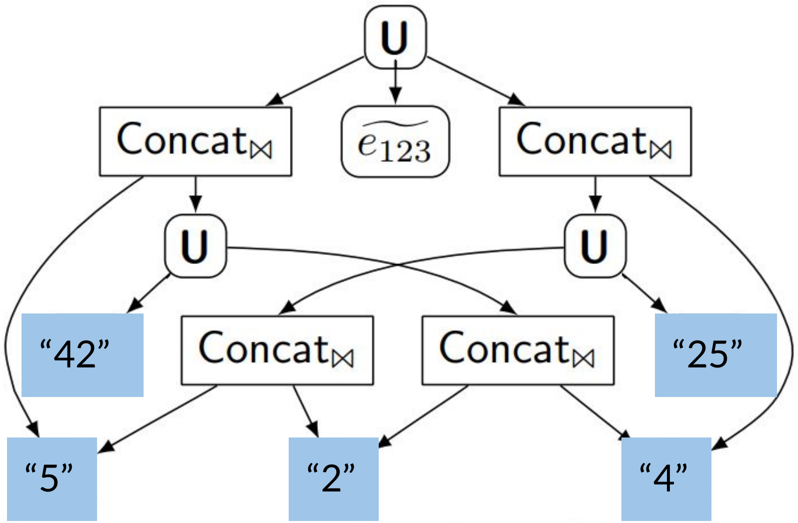


Figure 4: A VSA representing all possible programs output the string “425” in $\mathcal{L}_{\mathsf{FF}}$. Leaf nodes $\widetilde{e_1}, \widetilde{e_2}, \widetilde{e_3}, \widetilde{e_{12}}, \widetilde{e_{23}}$ are VSAs representing all ways to output substrings “4”, “2”, “5”, “42”, and “25” respectively (not expanded in the figure).

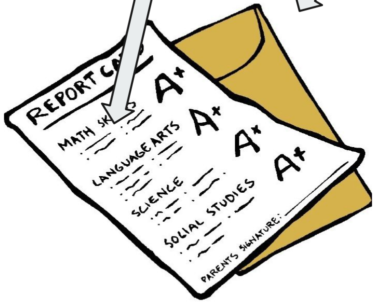
Inductive Synthesis Problem

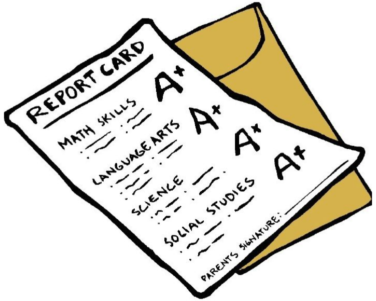
- **P** = Program
- **Ñ** = Set of Programs {P1, P2, …. Pn}
- $\mathcal{L}$ = Domain Specific Language (DSL)
- **$\sigma \rightsquigarrow \psi$** = Input output constraint (i.e. o/p of the program for i/p **σ** follows constraint **ψ**)
- **φ** = Collection of input output examples / constraints. (Specification)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

An inductive synthesis problem refers to synthesis of a program set **Ñ** $\subset \mathcal{L}$ that is consistent with a given inductive **specification φ**.

Inductive Synthesis Problem

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





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**, $\boldsymbol{\varphi}$)
- Backpropagate constraints on a program F (N1, … Nk) to deduced constraints on its Subexpressions N1, . . ., Nk.
- Witness function W transforms the Spec $\boldsymbol{\varphi}$ to corresponding spec $\boldsymbol{\varphi}$‘ for N‘

Learn (**N**, $\boldsymbol{\varphi}$)

Learn ($\mathbf{N}_1$, $\boldsymbol{\varphi}_1$)   Learn ($\mathbf{N}_2$, $\boldsymbol{\varphi}_2$)

```
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;
```

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 a 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

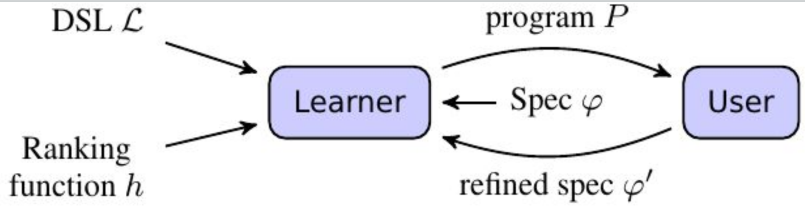
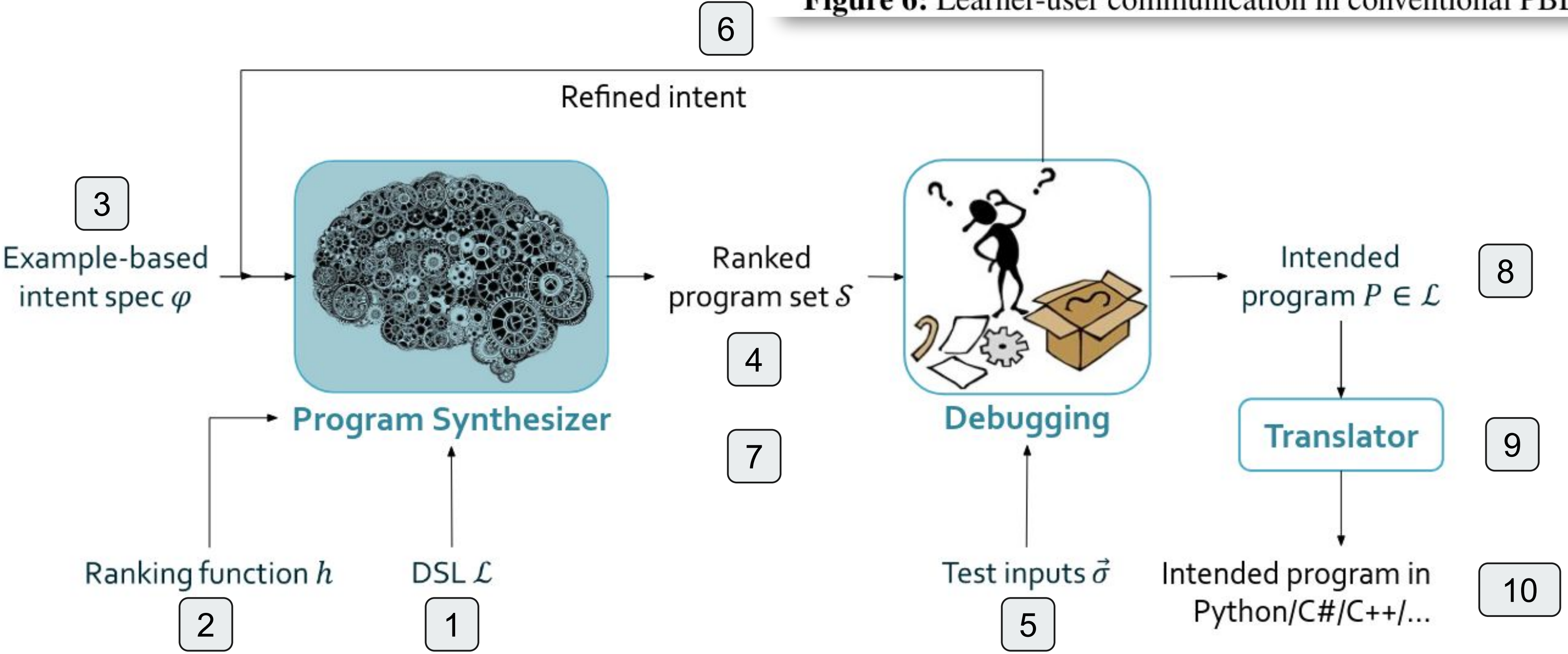


Figure 6: Learner-user communication in conventional PBE.



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
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** ∈ $\mathcal{L}$.
- Growing number of constraints **φ**
- Decreasing number of output programs **Ñ**

What if I could update the DSL $\mathcal{L}$ after each iteration ?

- May be a sub-DSL ?

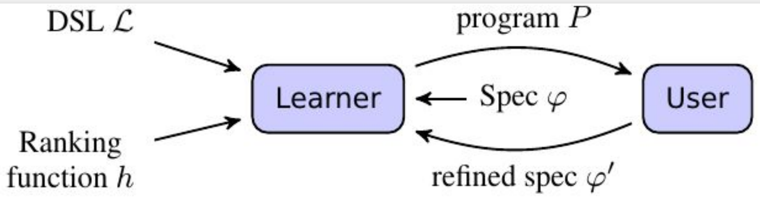
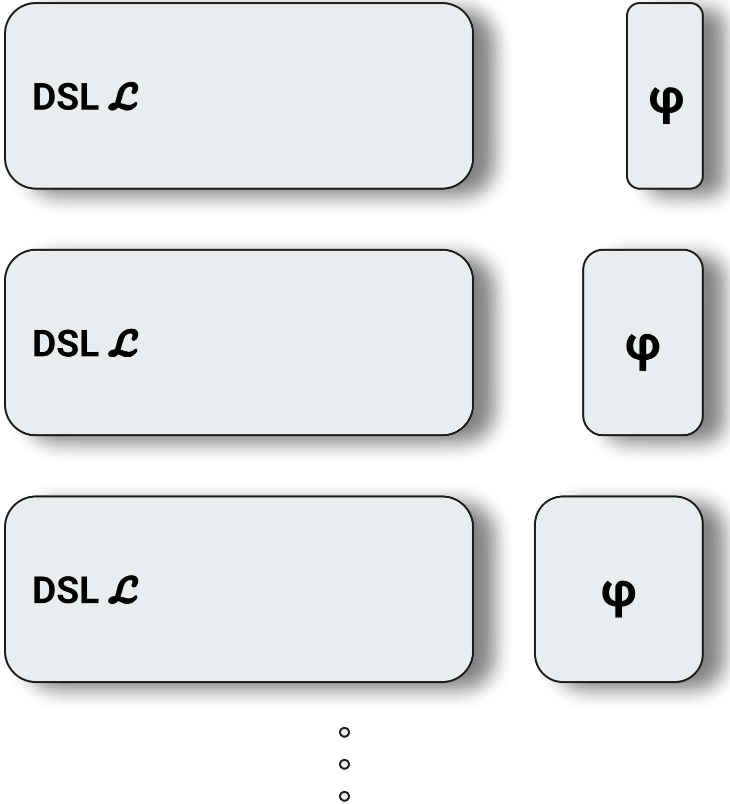


Figure 6: Learner-user communication in conventional PBE.



Inspirations: Incrementality

Incremental : (one logic at a time)

Observation on each iteration:

- Search over the same program space $\mathbf{N} \in \mathcal{L}$.
- Growing number of constraints $\boldsymbol{\varphi}$
- Decreasing number of output programs $\tilde{\mathbf{N}}$

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

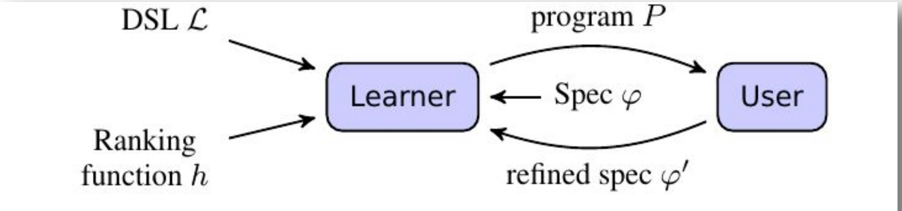
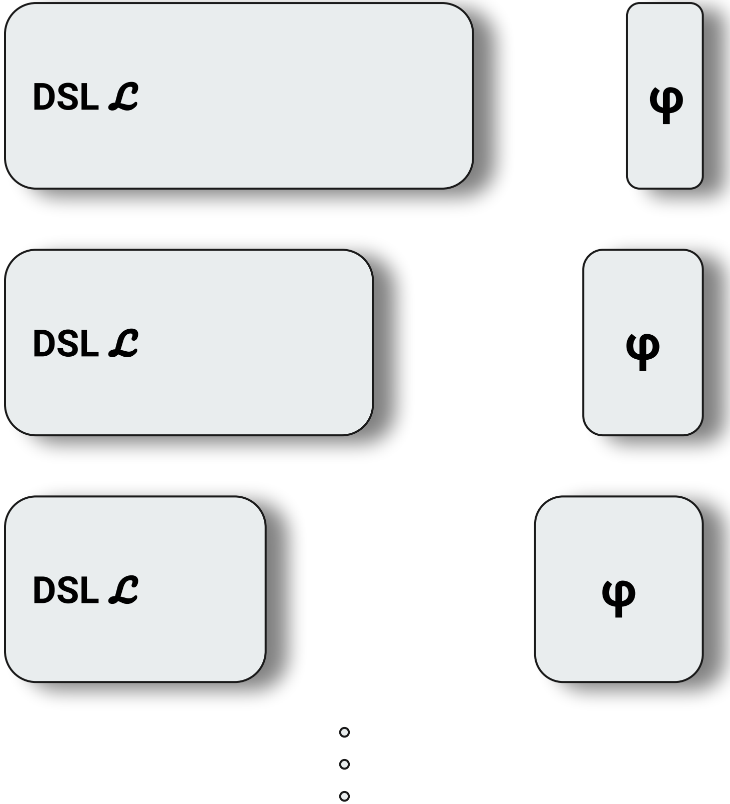


Figure 6: Learner-user communication in conventional PBE.



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 ?

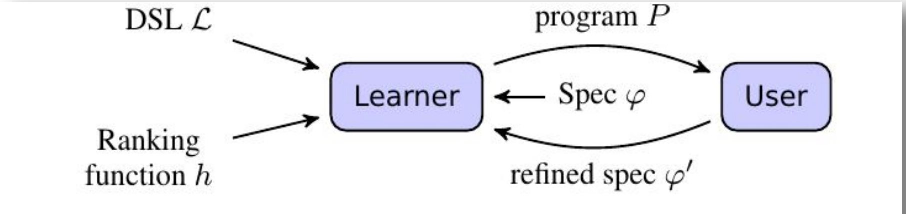
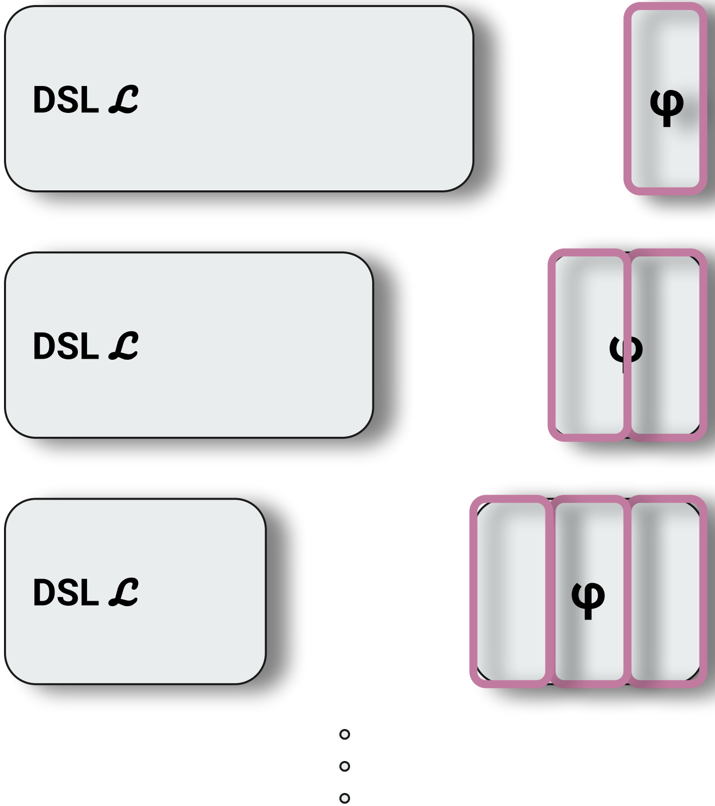


Figure 6: Learner-user communication in conventional PBE.



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 ?

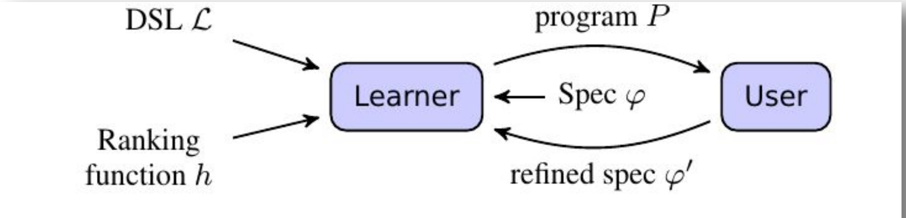
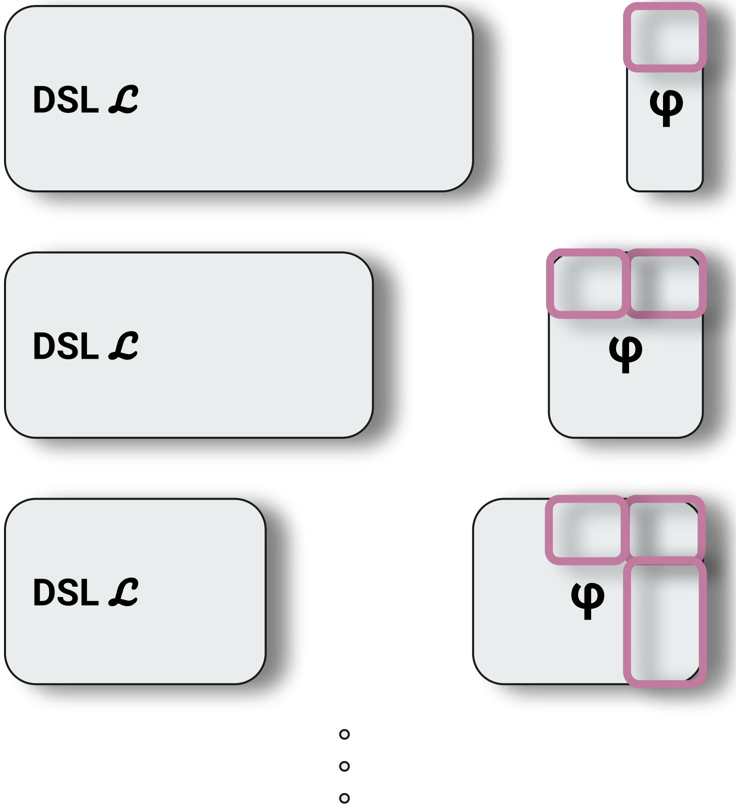


Figure 6: Learner-user communication in conventional PBE.



Inspirations: Feedback based interaction

Feedback-based : (Evolving codebase)

Observation on each iteration:

- No clear understanding of full spec **φ**.
- Lots of ambiguous programs **Ñ**
- 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)

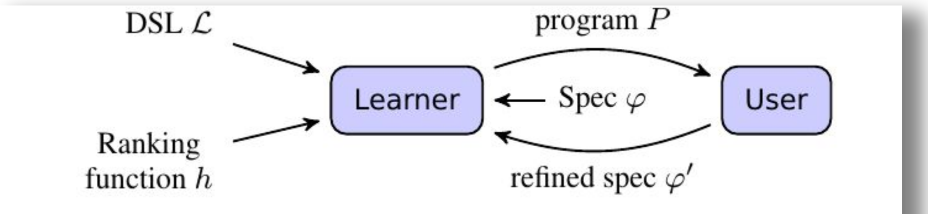
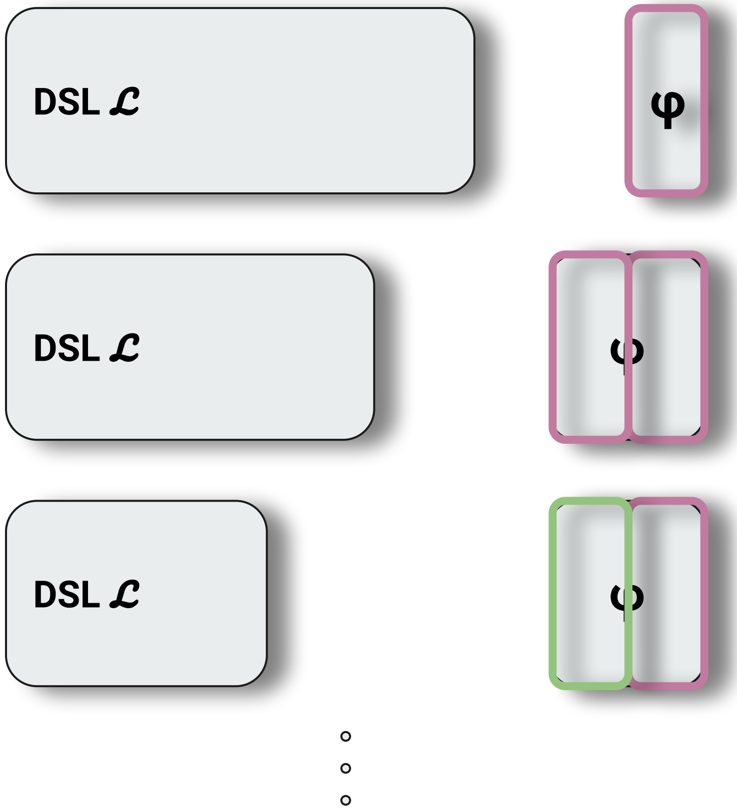


Figure 6: Learner-user communication in conventional PBE.



Inspirations: Feedback based interaction

Feedback-based : (Evolving codebase)

Observation on each iteration:

- No clear understanding of full spec **φ**.
- Lots of ambiguous programs **Ñ**
- 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)

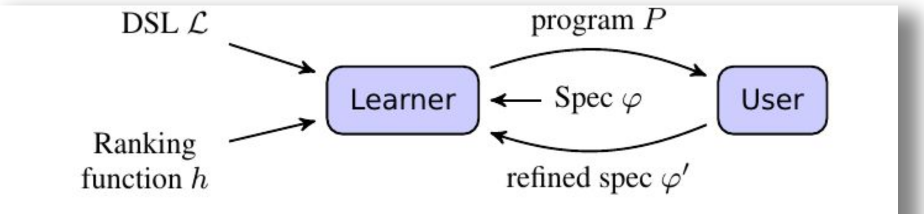
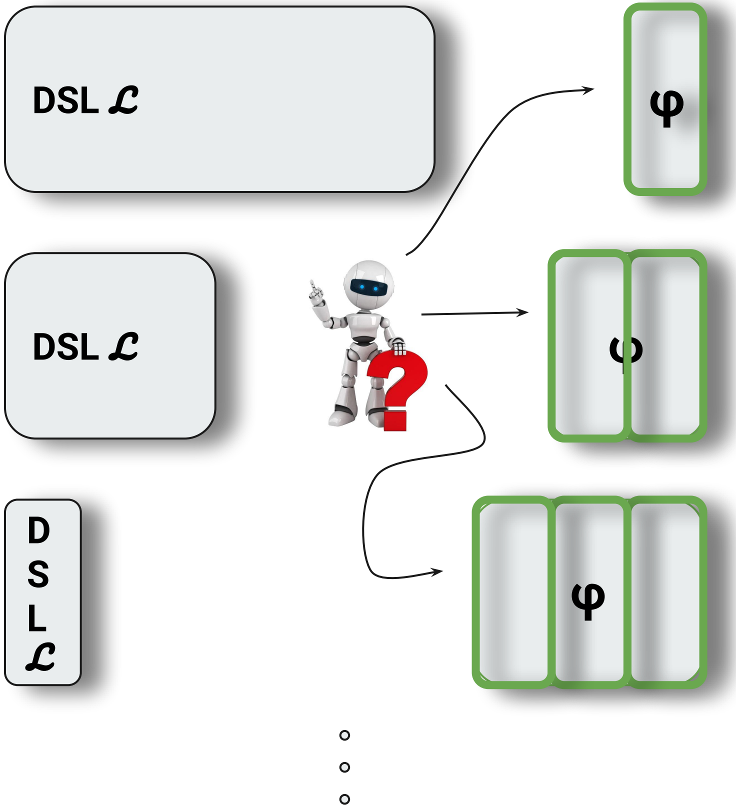


Figure 6: Learner-user communication in conventional PBE.



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

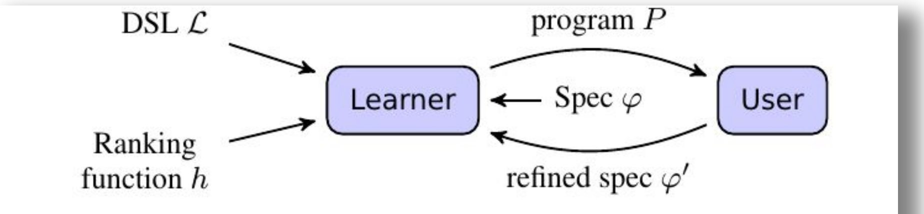
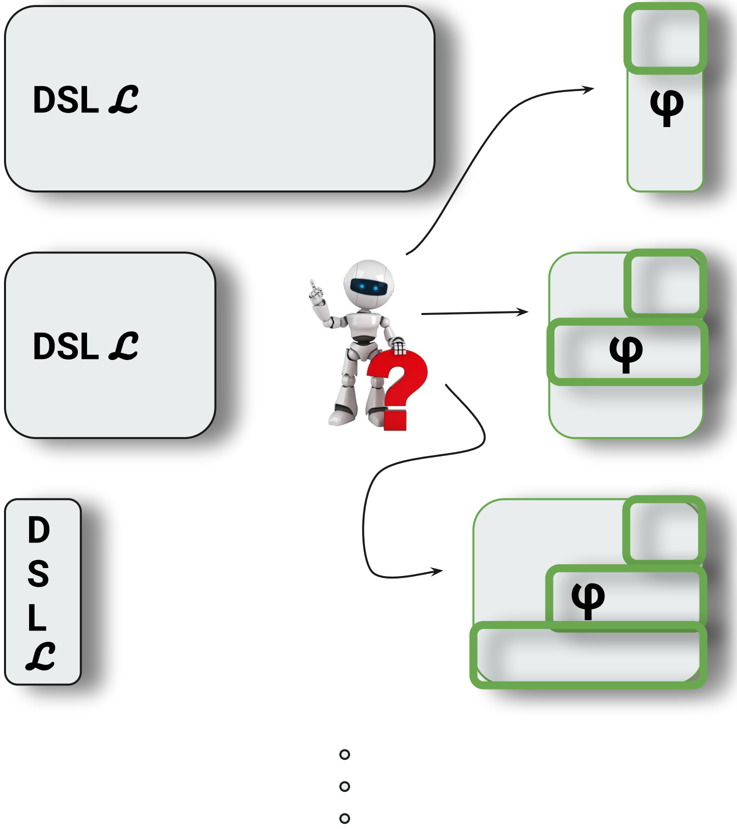


Figure 6: Learner-user communication in conventional PBE.



Interactive PBE Architecture

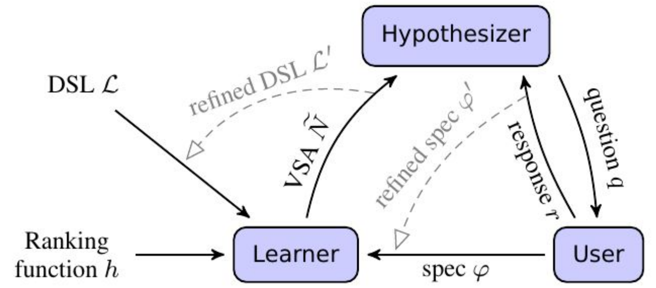
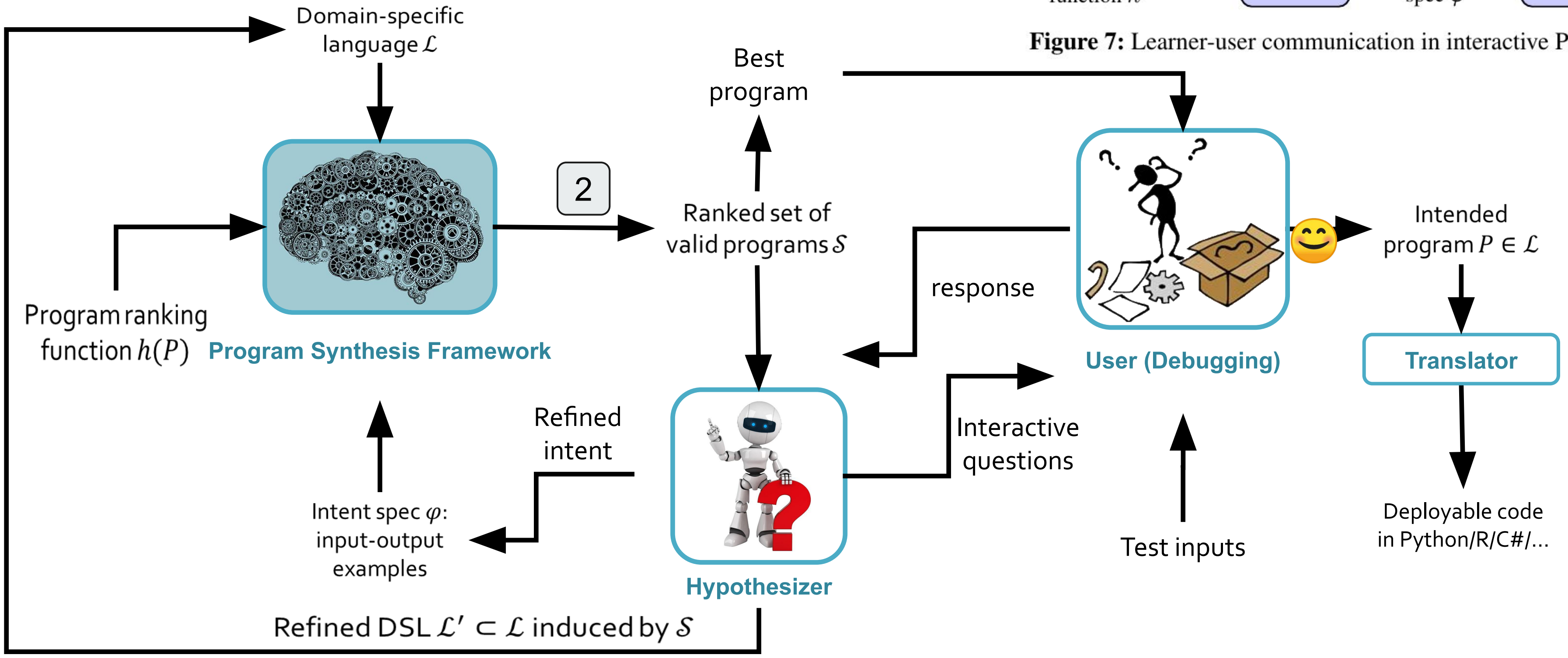


Figure 7: Learner-user communication in interactive PBE.



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

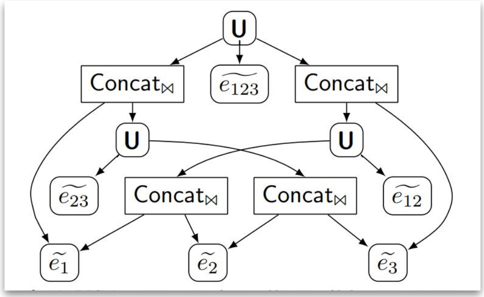
3 Key Inspirations towards Interactivity (Partial)

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




```
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);
```

function VsaToDsl(VSA $\widetilde{N}$)

Let V be a set of fresh nonterminals, one per each non-leaf node in $\widetilde{N}$
Let Σ be a set of fresh terminals, one per each leaf node in $\widetilde{N}$
// We write $\mathsf{sym}(\widetilde{N}') \in V \cup \Sigma$ *to denote the corresponding fresh*
// symbol for a node $\widetilde{N}'$ *from* $\widetilde{N}$
Productions $R \leftarrow \emptyset$
// Create "symbol := symbol" productions for all union nodes
for all union nodes $\widetilde{N}' = \mathbf{U}(\widetilde{N}_1, \ldots, \widetilde{N}_k)$ in $\widetilde{N}$ **do**
  $R \leftarrow R \cup \{\mathsf{sym}(\widetilde{N}') := \mathsf{sym}(\widetilde{N}_i) \mid i = 1 \ldots k\}$
// Create operator productions for all join nodes
for all join nodes $\widetilde{N}' = F_{\bowtie}(\widetilde{N}_1, \ldots, \widetilde{N}_k)$ in $\widetilde{N}$ **do**
  $R \leftarrow R \cup \{\mathsf{sym}(\widetilde{N}') := F(\mathsf{sym}(\widetilde{N}_1), \ldots, \mathsf{sym}(\widetilde{N}_k))\}$
// Annotate terminal symbols with values extracted from leaf nodes
for all leaf nodes $\widetilde{N}' = \{P_1, \ldots, P_k\}$ in $\widetilde{N}$ **do**
  Annotate in Σ that $\mathsf{sym}(\widetilde{N}') \in \{P_1, \ldots, P_k\}$
return the context-free grammar $G = \langle V, \Sigma, R, \mathsf{sym}(N) \rangle$

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 $\in \{v_1, v_2, v_3\}$",
- *Prefix constraint*: "output $= [v_1, v_2, \ldots]$",
- *Subset/subsequence constraint*: "output $\sqsupseteq [v_1, v_2, v_3]$".

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 : τ". 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 $\neq v$",
- *Negative membership constraint:* "output $\not\ni v$".

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 ψ."*

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)

$\mathcal{L}$ = Domain Specific Language (DSL)

$\mathcal{CT}$ = all top-level constraint types supported by synthesizer .

$\mathcal{C} \in \mathcal{CT}$, $\mathcal{C}$ is a top-level constraint type.

Ψ = instance_of($\mathcal{C}$) is a constraint.

For each $\mathcal{C} \in \mathcal{CT}$ We associate a descriptive " question **q** " such that a " response **r** " for this question directly

constitutes Ψ = instance_of($\mathcal{C}$)

$\mathcal{C} \in \mathcal{CT}$

Ψ = instance_of($\mathcal{C}$)

Response $\leftarrow$ Question

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 " get_disambiguation_score(**q**, **Ñ**, **φ**). "
- 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.

$$\mathsf{ds}_{\mathsf{R}}(q, \tilde{N}, \varphi) \stackrel{\text{def}}{=} \min_{r \in R(q)} \max_{P \in \tilde{N}_r} h(P)$$

Higher if every response for the question q leads to a higher-ranked alternative program.

3. Feedback-based Synthesis

procedure DISAMBIGUATE(Candidate programs $\widetilde{N}$, current spec φ)

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

$q^* \leftarrow \underset{q \in Q}{\operatorname{argmax}}\, \mathsf{ds}(q, N, \varphi)$ *// Compare the disa of all questions*

if $\mathsf{ds}(q^*, \widetilde{N}, \varphi) <$ threshold T **then**

    break

else

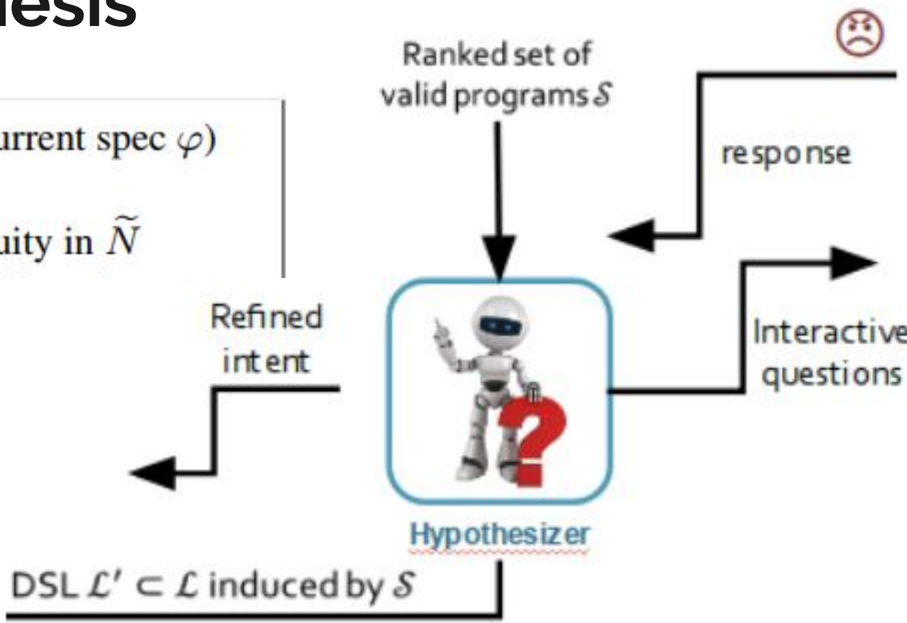
    Present the question q^* to the user

    Let r be the user's response to q

    Let ψ be the response r converted into a constraint

    return ψ to the learner and invoke a new round of synthesis

Figure 9: The hypothesizer's proactive disambiguation algorithm.



PBE Architecture

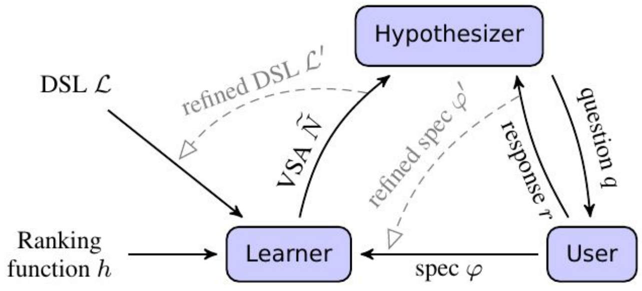
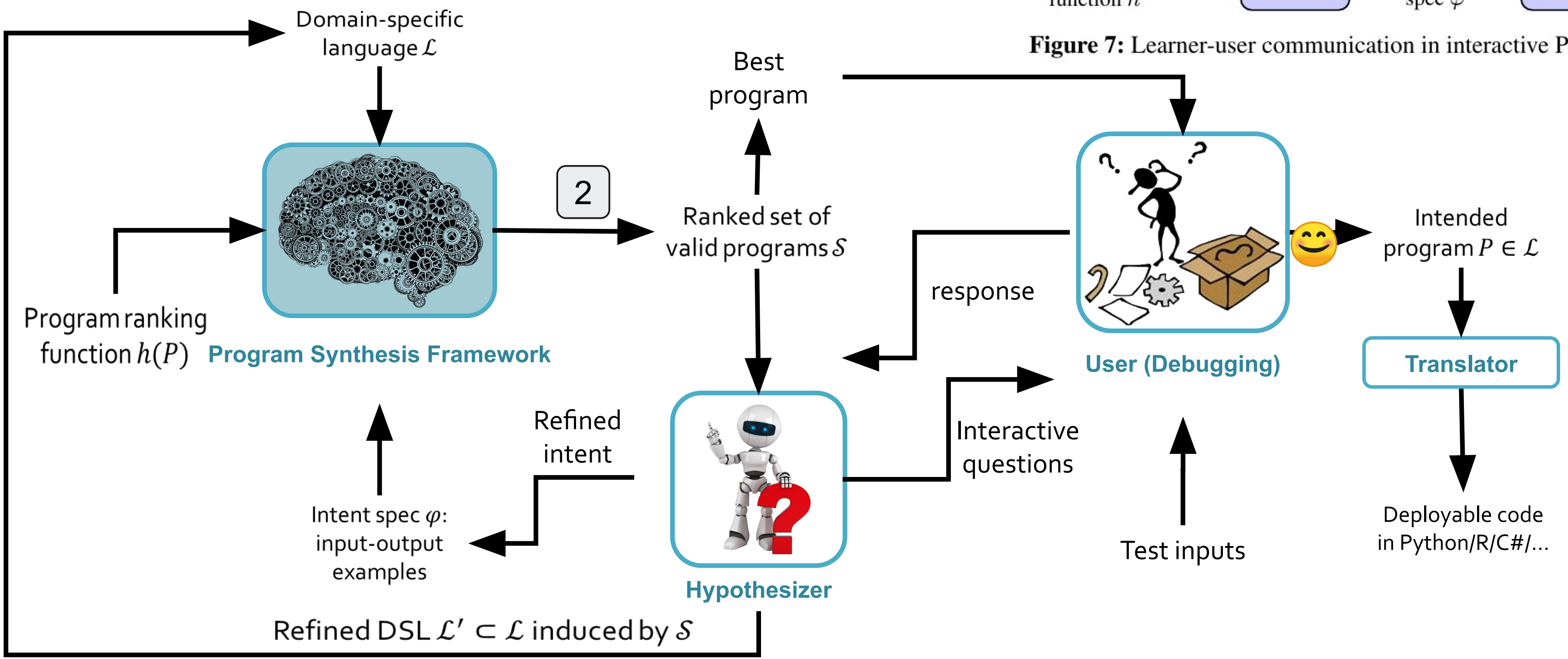


Figure 7: Learner-user communication in interactive PBE.



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)

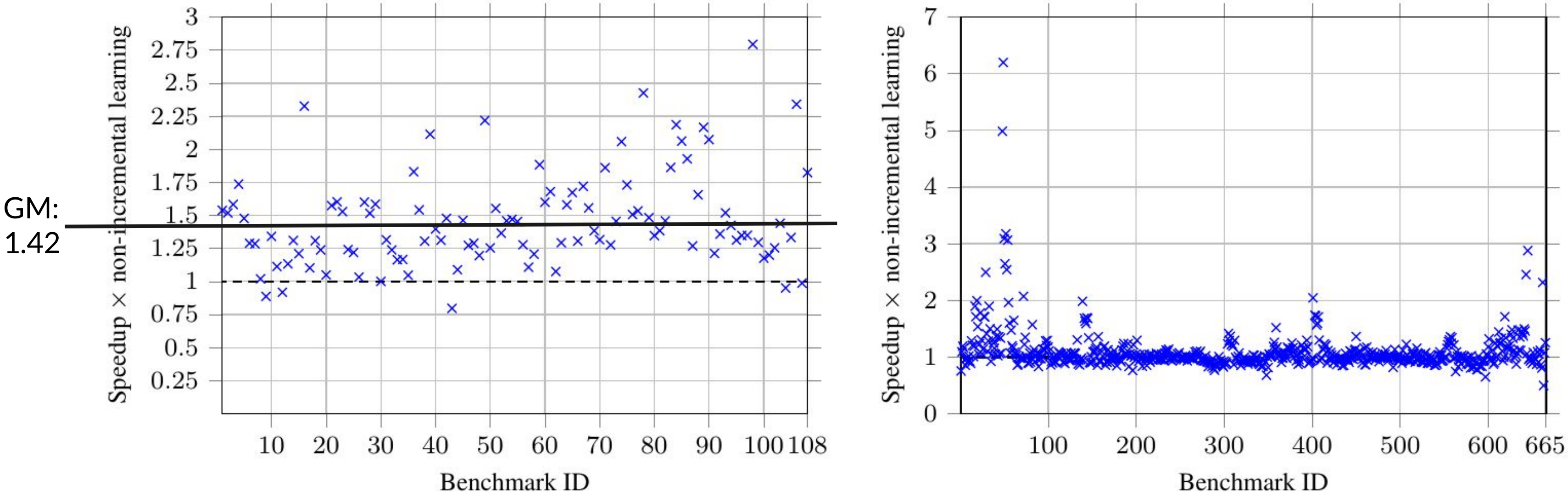
Hurdles to Mass Market

- 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)

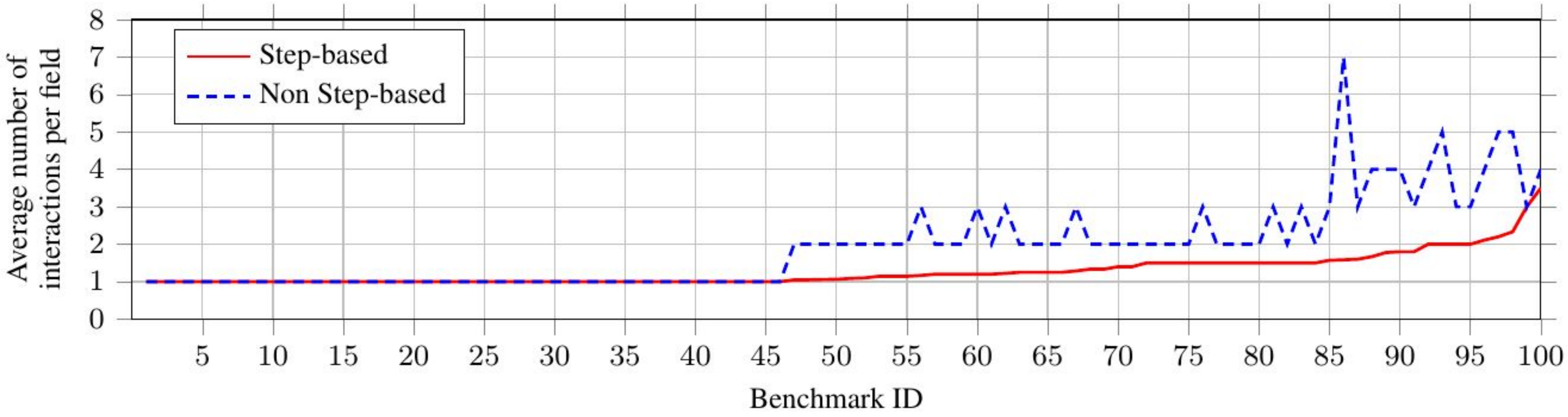


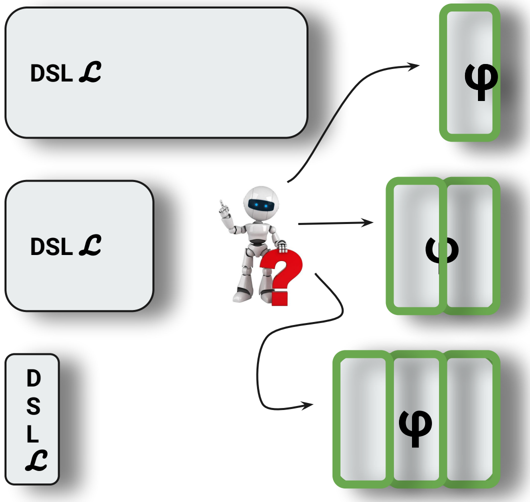
Figure 11: The average number of interactions per field across all benchmarks. Lower is better.

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 ∈ Qb presents a single position in the input row and asks if it is a desired splitting point.
- **Confirmation questions.** A confirmation question q ∈ Qc presents a set of positions to the user and asks whether all of these posi-tions 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 confir-mation questions, using the disambiguation score dsFS

Strategy
Baseline
BinaryQ
ConfirmationQ
CombinedQ



Evaluation - Feedback-based Synthesis

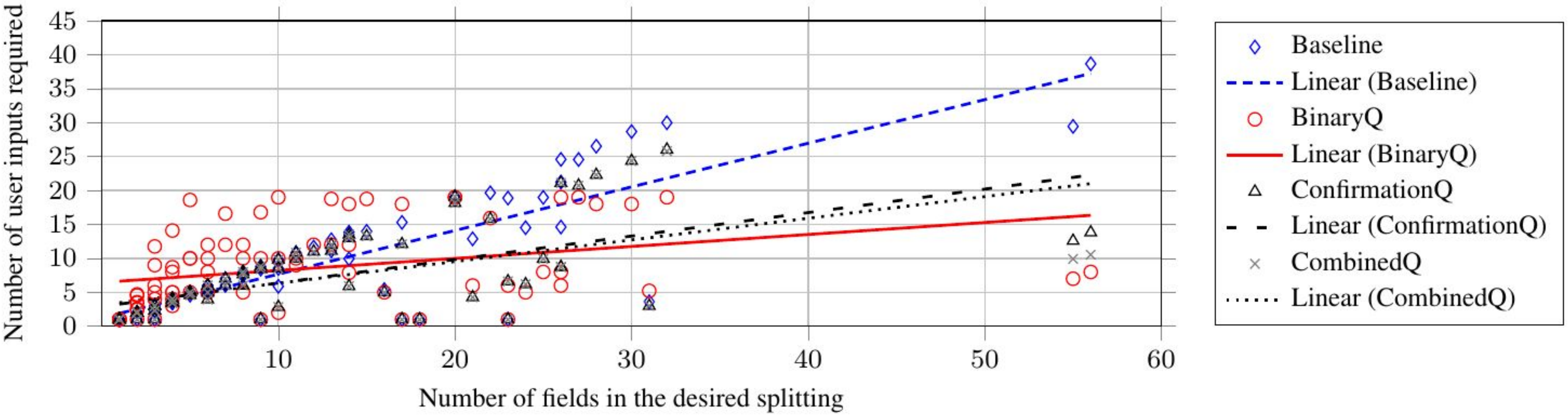


Figure 12: Comparison of effectiveness of different example provision strategies for field splitting tasks from log files. Lower is better.

Strategy	Avg. number of inputs
Baseline	8.81
BinaryQ	8.54
ConfirmationQ	6.98
CombinedQ	6.90

	Feedback						
Baseline	**1**	**2**	**3**	**4**	**5**	**6**	**7**
1	241	50	16	0	1	0	0
2	75	30	4	0	0	0	0
3	20	7	2	0	0	0	0
4	0	5	3	0	0	0	0
5	0	1	0	0	0	0	0
6	1	1	0	0	0	0	0

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)

Thank you !

Questions . . .

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.