Generating correct code for your programmers

PLISS 2025 – Part II

Hila Peleg - Technion

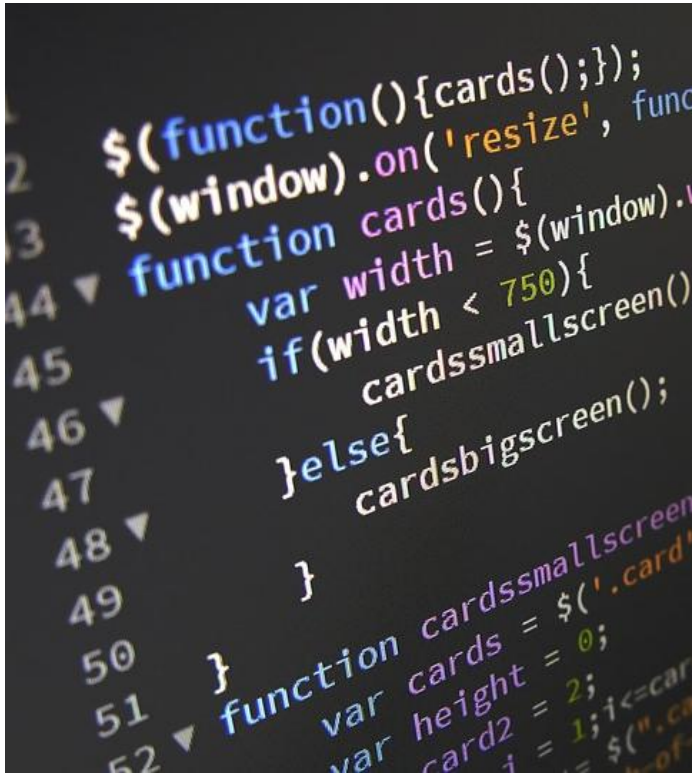
Funded by the European Union (ERC, EXPLOSYN, 101117232). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.

Last time: synthesis algorithms

$$
\begin{array}{l}
\wedge\ \forall p \in Proc, d \in Disk\ : \\
\quad \wedge\ (d \in disksWritten[p]) \Rightarrow \wedge\ phase[p] \in \{1,2\} \\
\qquad\qquad\qquad\qquad \wedge\ disk[d][p] = dblock[p] \\
\quad \wedge\ (phase[p] \in 1,2) \Rightarrow \wedge\ (blocksRead[p][d] \neq \{\}) \Rightarrow \\
\qquad\qquad\qquad\qquad\qquad\qquad (d \in disksWritten[p]) \\
\qquad\qquad\qquad\qquad\quad \wedge\ \neg hasRead(p, d, p) \\
\wedge\ \forall p \in Proc\ : \\
\quad \wedge\ (phase[p] = 0) \Rightarrow \wedge\ dblock[p] = InitDB \\
\qquad\qquad\qquad\qquad\quad \wedge\ disksWritten[p] = \{\} \\
\qquad\qquad\qquad\qquad\quad \wedge\ \forall d \in Disk\ :\ \forall br \in blocksReac \\
\qquad\qquad\qquad\qquad\qquad\quad \wedge\ br.proc = p \\
\qquad\qquad\qquad\qquad\qquad\quad \wedge\ br.block = disk[d][p] \\
\quad \wedge\ (phase[p] \neq 0) \Rightarrow \wedge\ dblock[p].mbal \in Ballot(p) \\
\qquad\qquad\qquad\qquad\quad \wedge\ dblock[p].bal \in Ballot(p) \cup \{0\} \\
\qquad\qquad\qquad\qquad\quad \wedge\ \forall d \in Disk\ :\ \forall br \in blocksReac \\
\qquad\qquad\qquad\qquad\qquad\quad br.block.mbal < dblock[p]. \\
\quad \wedge\ (phase[p] \in \{2,3\}) \Rightarrow (dblock[p].bal = dblock[p].mb \\
\quad \wedge\ output[p] = \text{IF } phase[p] = 3 \text{ THEN } dblock[p].inp \text{ E} \\
\wedge\ chosen \in allInput \cup \{NotAnInput\} \\
\wedge\ \forall p \in Proc\ :\ \wedge\ input[p] \in allInput \\
\qquad\qquad\qquad\quad \wedge\ (chosen = NotAnInput) \Rightarrow (output[p]
\end{array}
$$

$\Rightarrow$

$\Rightarrow$



Generic synthesis recipe

1. Generate a candidate program

- Enumerate trees
  - Top-down
  - Bottom-up
- Traverse automata
- Graph reachability
- Enumerate deduction rules
- Cheat* by looking at spec/domain

2. Test against specification

- Run tests
  - Examples
  - Unit-
- Encode for SMT solver
- Apply typing rules

Design Spaces

Input Medium	Text	Audio	Image	
Output Medium	Text	Audio	Image	
Operation	Completion	Creation	Information extraction	Transformation
Refinement	None	Code	Puppies	

Design Spaces

Input Medium	Text	Audio	Image	
Output Medium	Text	Audio	Image	
Operation	Completion	Creation	Information extraction	Transformation
Refinement	None	Code	Puppies	

Design Spaces

Input Medium	Text	Audio	Image	
Output Medium	Text	Audio	Image	
Operation	Completion	Creation	Information extraction	Transformation
Refinement	None	Code	Puppies	

Design Spaces

Input Medium	Text	Audio	Image	
Output Medium	Text	Audio	Image	
Operation	Completion	Creation	Information extraction	Transformation
Refinement	None	Code	Puppies	

The space of synthesis algorithms

- Specifications
  (e.g., PBE, types, sketches) φ
- Search algorithm
  (e.g., inductive, deductive, VSA)

The space of synthesis algorithms

- Specifications
  (e.g., PBE, types, sketches)
- Search algorithm
  (e.g., inductive, deductive, VSA)

$= (\varphi ,$  $)$

Last time: the search is highly customized

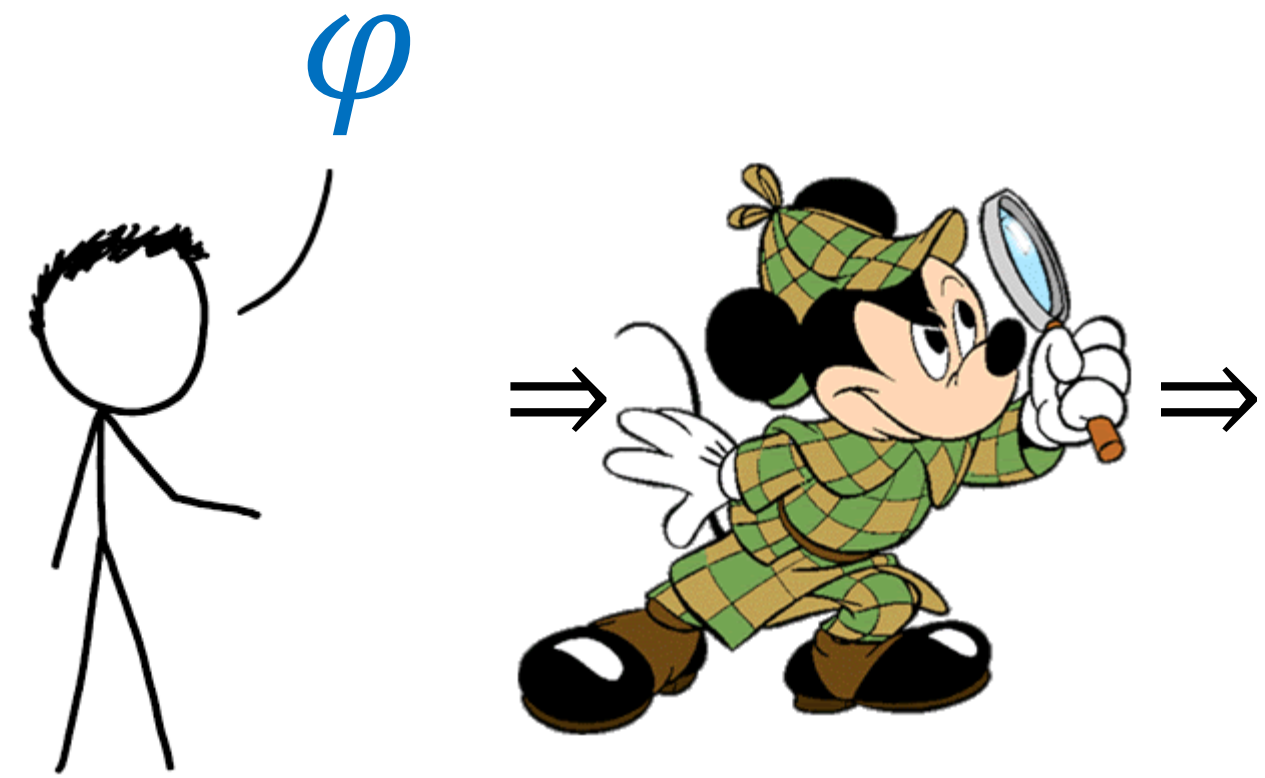
Search by example

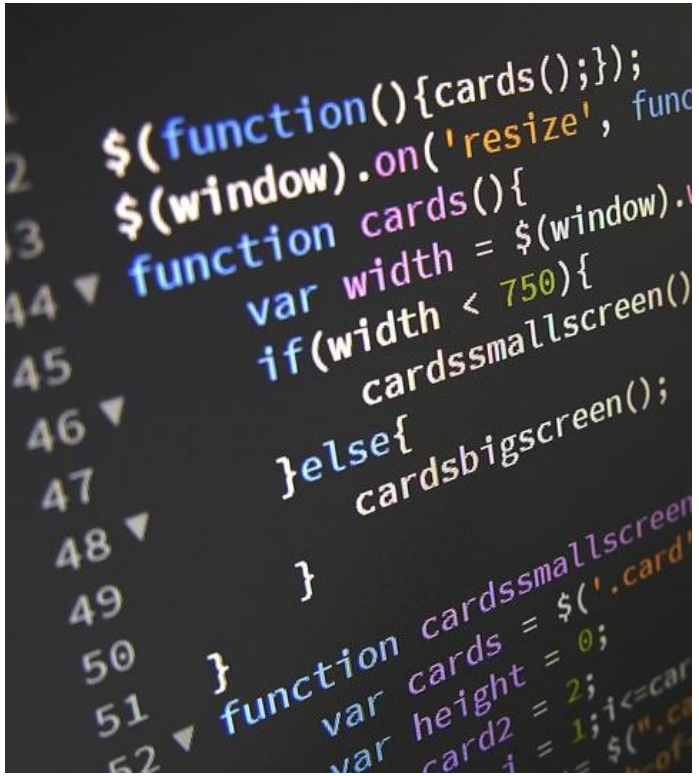
Search by type

Search by logical formulae

Also last time

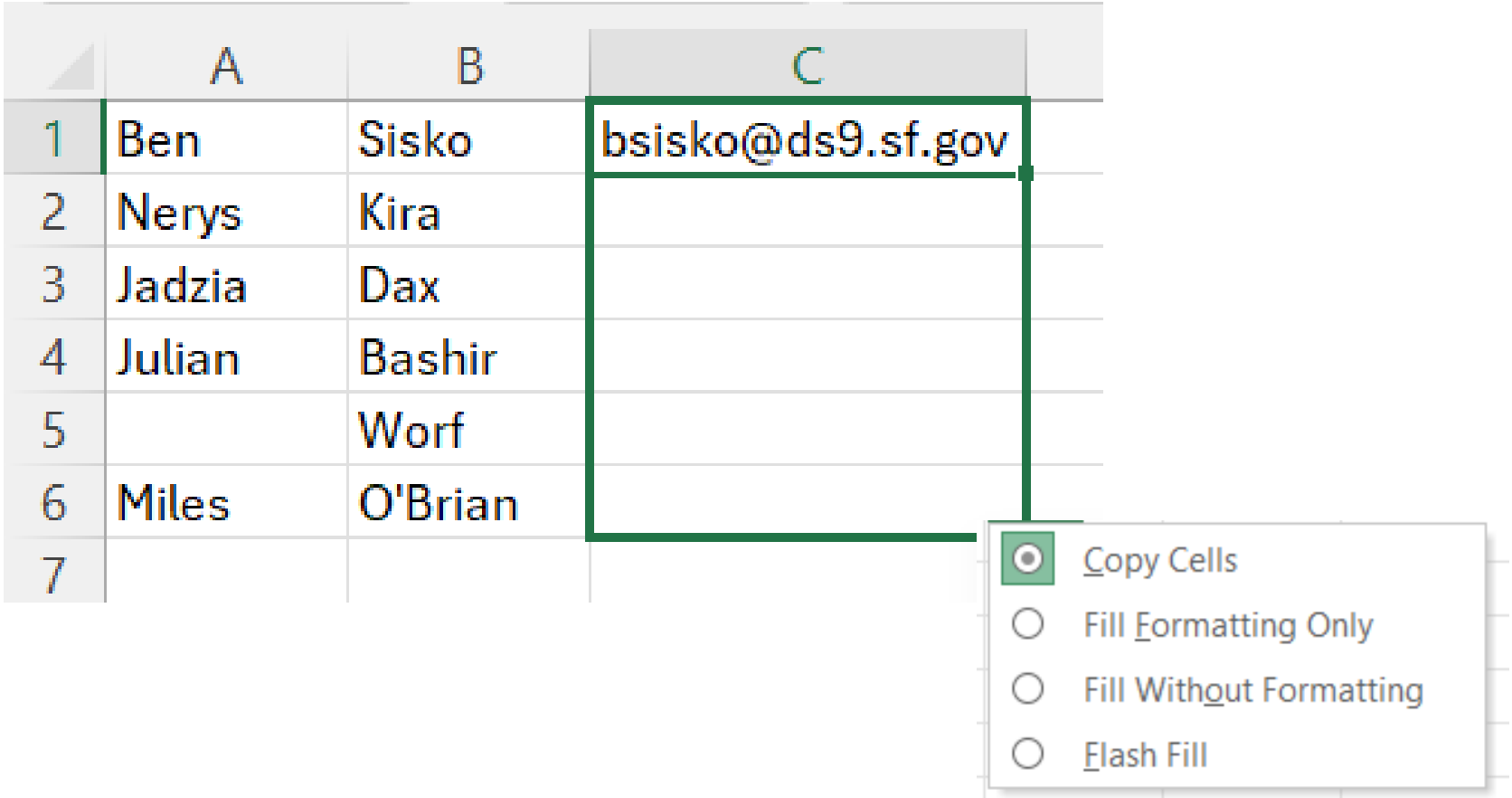
Program Synthesis





FlashFill

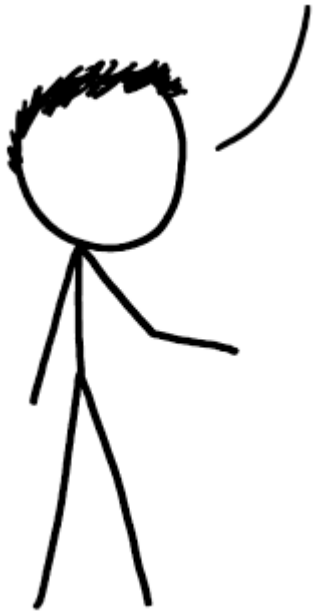
POPL'11, Excel since 2013




North Carolina
thank you flash fill that is exactly what I wanted
City
No. o
Austin TX Texas 2
alt Lake City UT Utah 1
Durham NC North Carolina 3
Columbus OH Oorth Carolina 35
Baton Rouge LA Lorth Carolina 11
Omaha NE North Carolina 27
New Orleans LA Lorth Carolina 39
Des Moines IA lorth Carolina 16
Seattle WA Worth Carolina 42
klahoma City OK Oorth Carolina 14
Houston TX Torth Carolina 62
Charleston SC Sorth Carolina
Washington DC Dorth Carolina 225
Milwaukee WI Worth Carolina 528
Columbia SC Sorth Carolina 56
San, Diego CA Corth Carolina 329
Orlando FL Forth Carolina 190
Boston MA Morth Carolina 489
Dallas TX Torth Carolina 489
Minneapolis MN Morth Carolina 396

Program Synthesis: closer to real

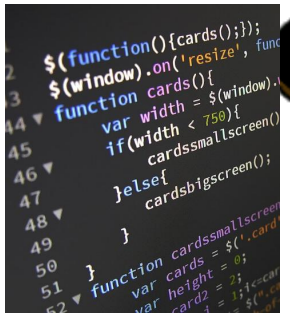
This is what I want



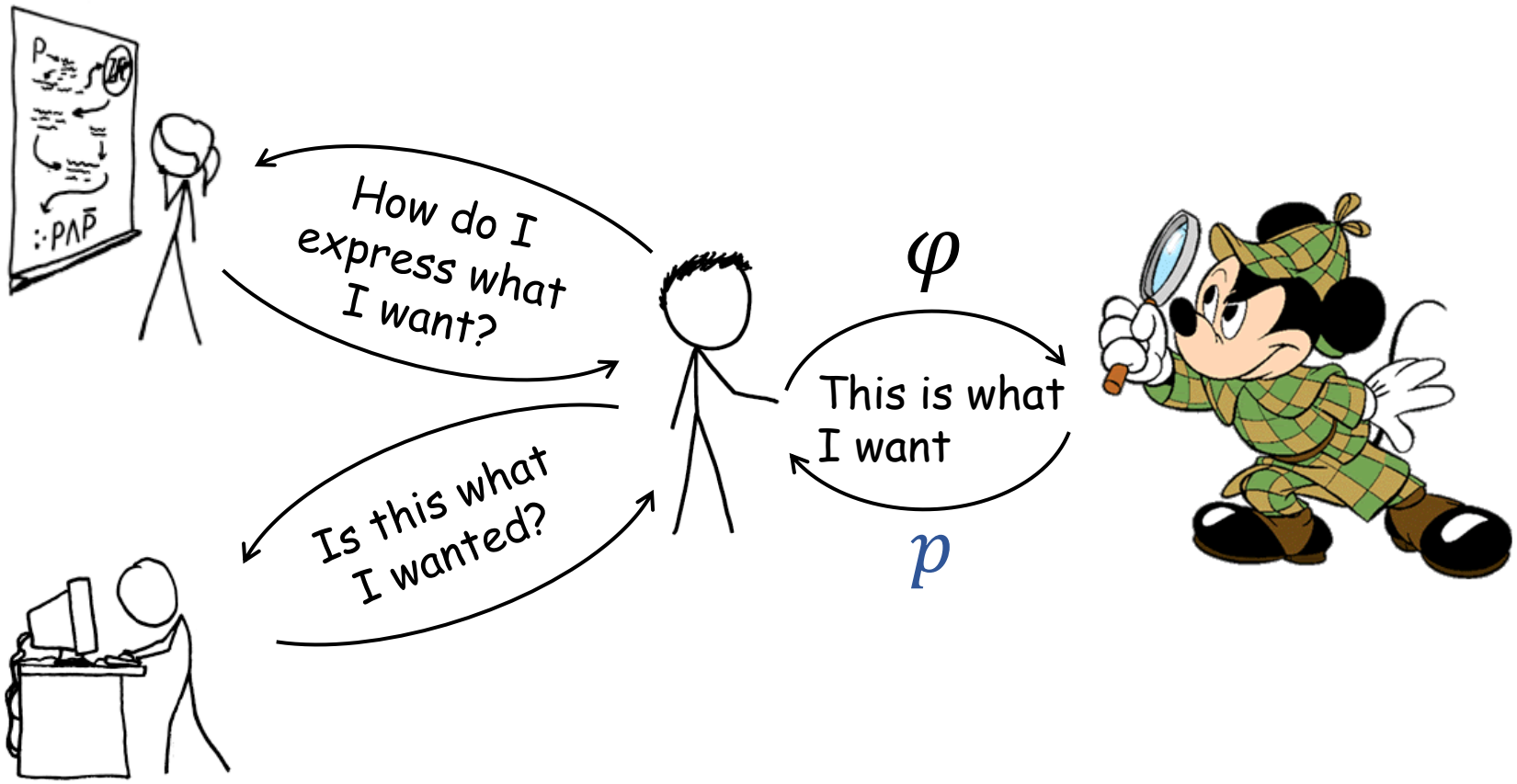
φ

$\Rightarrow$

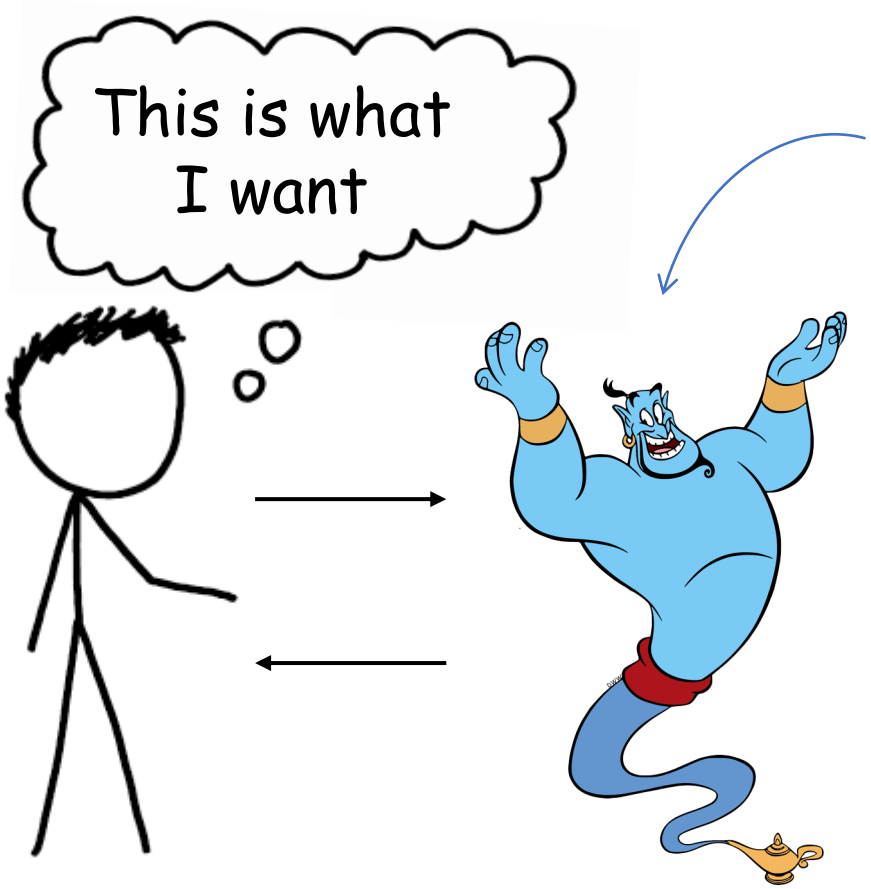
$\Leftarrow$



Interaction models



Another way to think about it

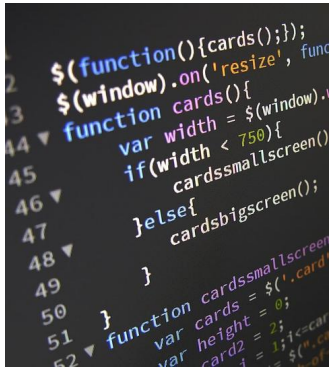


Interaction Model

φ

$\Rightarrow$

$\Leftarrow$



The design space of interaction models

Exploring the Learnability of Program Synthesizers by Novice Programmers

Dhanya Jayagopal*
dhanyajayagopal@berkeley.edu
University of California, Berkeley
Berkeley, USA

Justin Lubin*
justinlubin@berkeley.edu
University of California, Berkeley
Berkeley, USA

Sarah E. Chasins
schasins@cs.berkeley.edu
University of California, Berkeley
Berkeley, USA

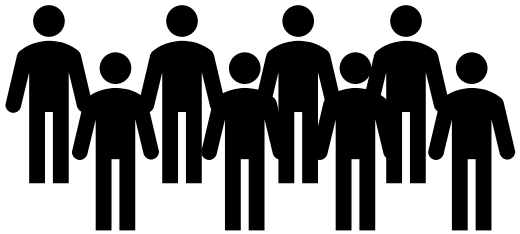
ABSTRACT

Modern program synthesizers are increasingly delivering on their promise of lightening the burden of programming by automatically generating code, but little research has addressed how we can make

1 INTRODUCTION

The promise of *program synthesis* is to lighten the burden of programming by automatically generating code that satisfies a user-provided specification. However, little work has studied how novice

Learnability of Synthesizers



22 users

- Students in 2nd semester data structures class
- Screened to have no additional programming experience

3 tasks

BluePencil

FlashFill

SnipPy*

Copilot

Regae

Semi-structured interview

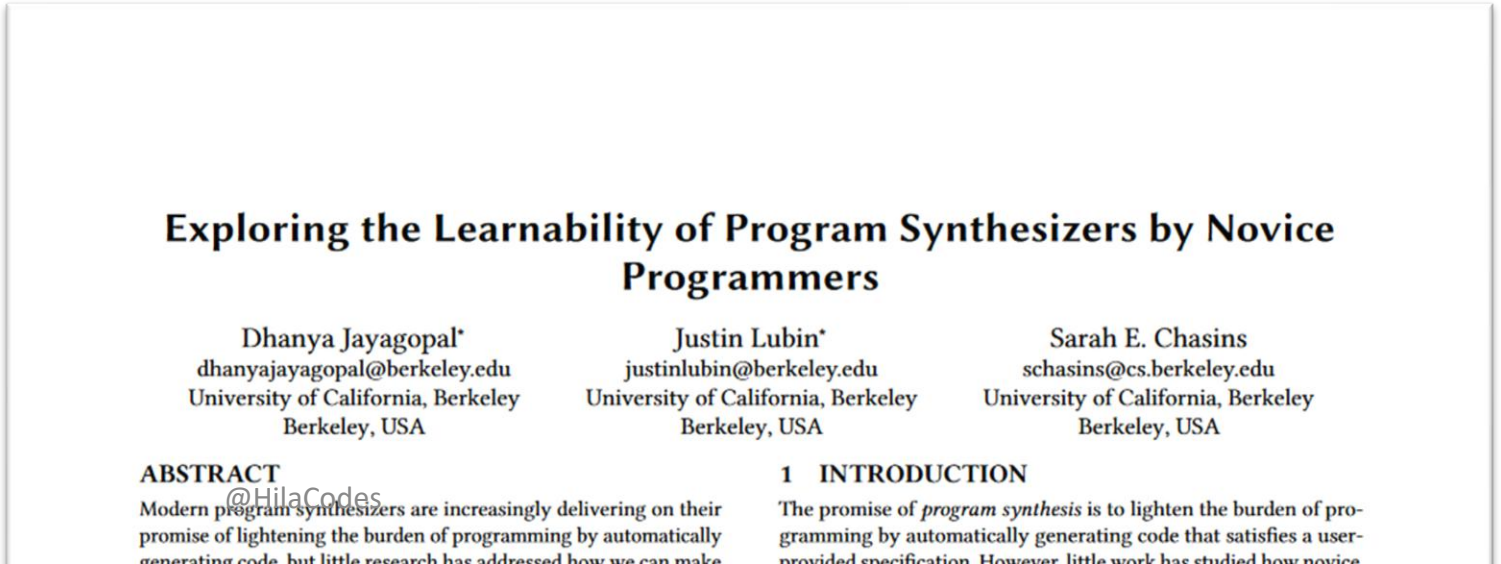
Sentiments about tool used

Synthesizer	Name	#	Description
Blue-Pencil	Point	1	Change the program to use `Point` objects to represent position rather than a pair of integers `(x, y)`.
Blue-Pencil	Rename	2	Change the name of variable `X` to `latitude`. Change the name of variable `Y` to `longitude`.
Blue-Pencil	LinkedList	3	Change a list to be a `LinkedList` (built-in Java class) instead of a primitive array.
Copilot	Abbreviate	1	Write a program to return the abbreviation of a given name.
Copilot	Occurrences	2	Write a program to turn a list into a dictionary that counts the number of occurrences of each character.
Copilot	Subsequence	3	Write a program to find the length of the longest subsequence of a given sequence such that all elements of the subsequence are sorted in increasing order. For example, the output for `[10, 22, 9, 33, 21, 50, 41, 60, 80]` should be 6 because the longest sorted subsequence is `[10, 22, 33, 50, 60, 80]`.
Flash Fill	Names	1	Using the data from Column A, populate Column B with `<Last Name>, <First Name>` and populate Column C with `<First Initial><Last Name>` in lower case.
Flash Fill	Emails	2	Populate Column B with the prefixes of the email addresses in Column A.
Flash Fill	Characters	3	Populate Column B with all of the upper case letters from Column A. Populate Column C with all of the lower case letters from Column A. Populate Column D with all of the numbers from Column A.
Regae	Plus	1	Write a regular expression that accepts strings that contain + or digits but no ++.
Regae	ABC	2	Write a regular expression that accepts strings that only have `A`, `B`, `C`, or any combinations of them.
Regae	Phone	3	Write a regular expression that accepts phone numbers that start with one optional + symbol and follow with a sequence of digits. For example, `+91` and `91`, but not `91+`.
SnipPy	Abbreviate	1	Write a program to return the abbreviation of a given name.
SnipPy	Reverse	2	Write a program to reverse a given string.
SnipPy	Filter	3	Write a program to return a given string without a specified letter.

The space of interaction models

Specifications:	Voluntary	⇔	Incidental
Initiation:	Triggerless	⇔	User-Triggered
Result communication:	Triggerless	⇔	User-triggered

More learnable
(by novices)



Exploring the Learnability of Program Synthesizers by Novice Programmers

Dhanya Jayagopal*
dhanyajayagopal@berkeley.edu
University of California, Berkeley
Berkeley, USA

Justin Lubin*
justinlubin@berkeley.edu
University of California, Berkeley
Berkeley, USA

Sarah E. Chasins
schasins@cs.berkeley.edu
University of California, Berkeley
Berkeley, USA

ABSTRACT

Modern program synthesizers are increasingly delivering on their promise of lightening the burden of programming by automatically

1 INTRODUCTION

The promise of *program synthesis* is to lighten the burden of programming by automatically generating code that satisfies a user-

The interactive synthesis space

The interactive synthesis space

	A	B	C
1	First Name	Last Name	Email
2	Benjamin	Sisko	benjamins@ds9.sf
3	Nerys	Kira	
4	Miles	O'Brien	
5	Jadzia	Dax	
6	Julian	Bashir	

$\in$

$\times$

$\ni$

$$\{x \mapsto a * y \mapsto b\}$$
$$p$$
$$\{y \mapsto a * emp\}$$

The interactive synthesis space

Algorithm first: an example

“users can provide specifications using a user interface”

—imaginary quote based on dozens and dozens of papers

Interaction first: example

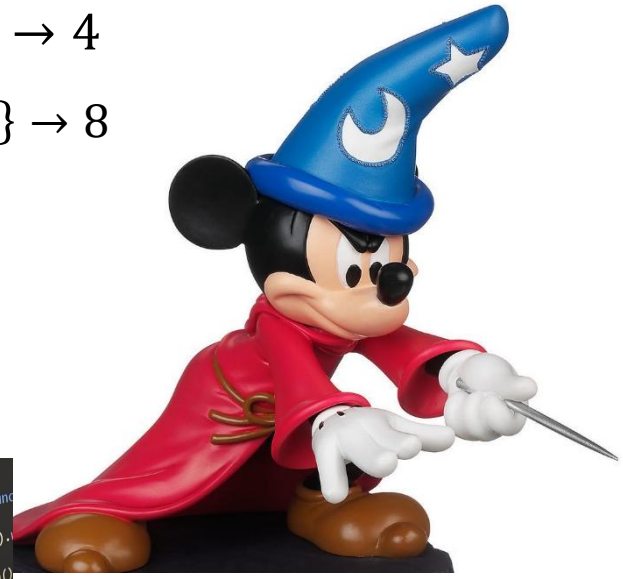
Programming by Example be like:

$$\{x \mapsto 2, y \mapsto 8\} \to 4$$

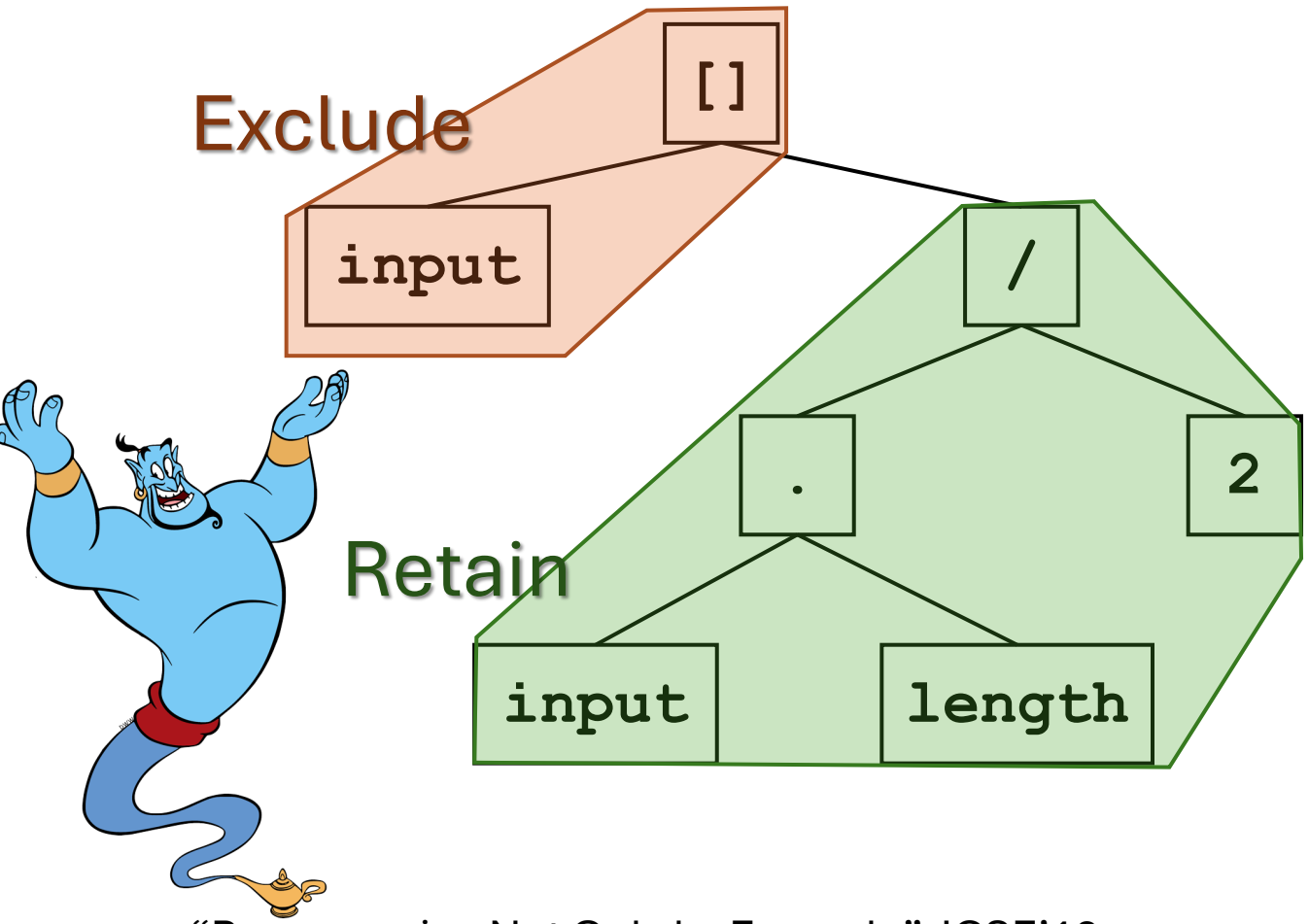
$$\{x \mapsto 16, y \mapsto 128\} \to 8$$

$$\Rightarrow$$

$$\Leftarrow$$



Interaction first: an example



$\{input \mapsto \texttt{[1,9,5]}\} \rightarrow 5$

Retain(input.length / 2)

No program

"Programming Not Only by Example", ICSE'18

Synthesis Co-Design

Why did synthesis break?

$\{input \mapsto \texttt{[1,9,5]}\} \rightarrow 5$

Retain(`input.length / 2`)

height 1 ... **`input.length`** <3> ...

$i_0 = \{input \mapsto \texttt{[1,9,5]}\}$

height 2 ... **`input.length / 2`** <1.5> ...

Why did synthesis break?

$\{input \mapsto$ `[1,9,5]`$\} \rightarrow 5$

Retain(`input.length / 2`)

height 1 `2+1` <3> ... `input.length` <3> ...

$i_0 = \{input \mapsto$ `[1,9,5]`$\}$

height 2 ... `input.length / 2` <1.5> ...

Why did synthesis break?

$\{input \mapsto$ `[1,9,5]`$\} \rightarrow 5$

Retain(`input.length / 2`)

height 1 **`2+1`** <3> ... $i_0 = \{input \mapsto$ `[1,9,5]`$\}$

height 2 ... **`input.length / 2`** <1.5> ...

Why did synthesis break?

$\{input \mapsto \texttt{[1,9,5]}\} \rightarrow 5$

Retain(`input.length / 2`)

height 1 **`2+1`** <3> ... $i_0 = \{input \mapsto \texttt{[1,9,5]}\}$

height 2 ... `input.length / 2` <1.5> ...

Solution: Generalize synthesis algorithm

Before (examples $\mathcal{E}$):

Observational equivalence:

$$m_1 \equiv_{\mathcal{E}} m_2 \Leftrightarrow$$
$$\forall (i, o) \in \mathcal{E}. [\![m_1]\!](i) = [\![m_2]\!](i)$$

Problem: only considers execution results for equivalence

After (general specification S):

Observational equivalence:

$$m_1 \equiv_S m_2 \Leftrightarrow$$
$$\forall \pi \in S. \pi(m_1) = \pi(m_2)$$

Solution: each spec element has an *observer* that expresses what equivalence means for it

…for an example, it's still execution results

Synthesis fixed

$\{input \mapsto \texttt{[1,9,5]}\} \rightarrow 5$

Retain(`input.length / 2`)

height 1

`2+1`
<3,not part of retain>

...

`input.length`
<3, part of retain>

...

height 2

...

`input.length / 2`
<1.5,has retained expr>

...

Synthesis can do more now

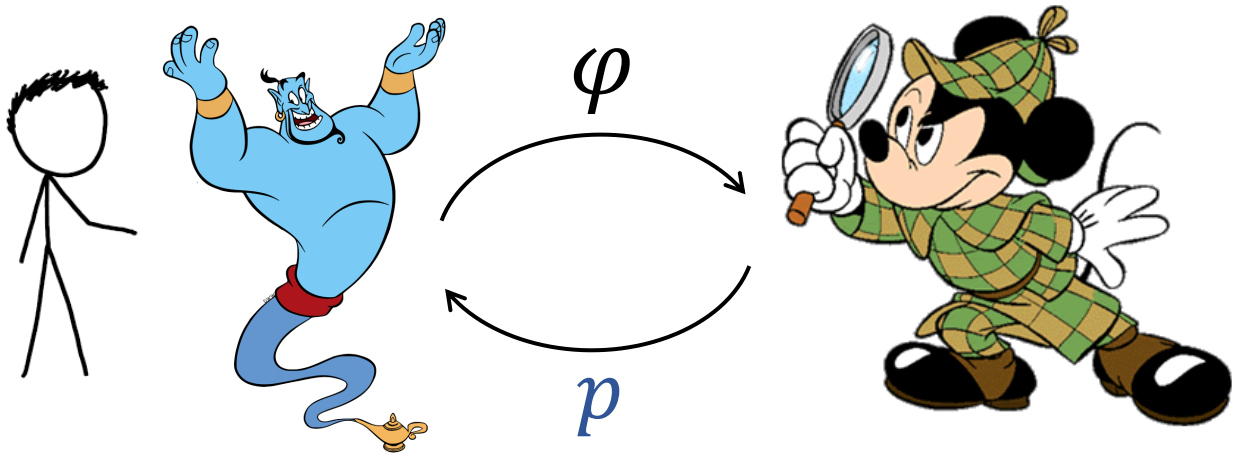
Other things we can create an observer (π) for:

- Specifying exceptions
- Negative examples
- Types of subexpressions
- Anything, really, as long as π has two properties we care about

Type Constraints:

	require	prohibit
Number:	require	prohibit
Boolean:	require	prohibit
String:	require	prohibit
Object:	require	prohibit
Array:	require	prohibit

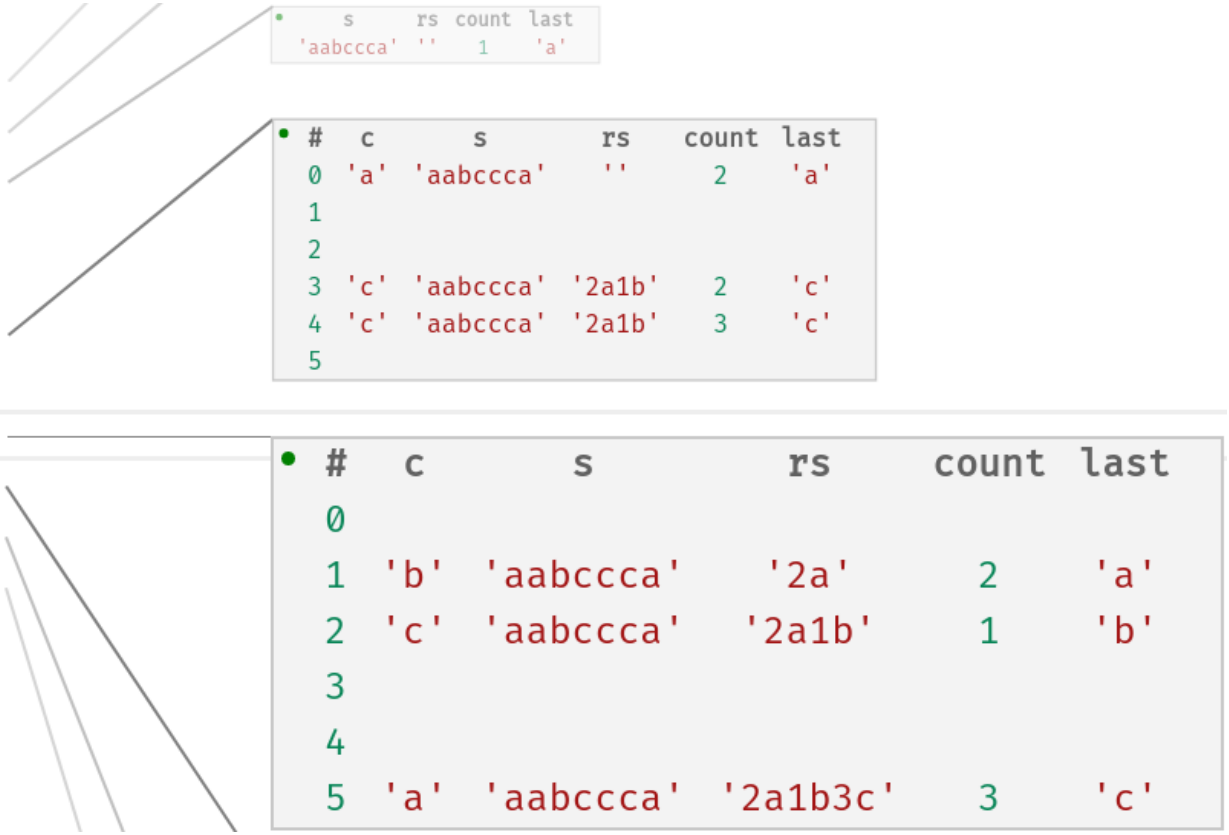
Interaction models can referee



Help both sides out!

Live Programming

```
1   def task(s):
2       rs = ''
3       count = 1
4       last = s[0]
5       for c in s[1:]:
6           if c == last:
7               count += 1
8           else:
9               rs += str(count) + last
10              count = 1
11              last = c
12      return rs
13
14  task('aabccca')
```



Live Programming by Example

```
1  def initials(s):
2      letters = ''
3      return ''
4
5  initials('Augusta Ada King')
6
```

#	s	letters
	'Augusta Ada King'	''

#	s	letters	rv
	'Augusta Ada King'	''	''

```
1  def initials(s):
2      letters = ''
3      return ''
4
5  initials('Augusta Ada King')
6
```

#	s	letters
	'Augusta Ada King'	['A', 'A', 'K']

#	s	letters	rv
	'Augusta Ada King'	''	''

```
1  def initials(s):
2      # Synthesizing. Please wait...
3      return ''
4
5  initials('Augusta Ada King')
6
```

#	s
	'Augusta Ada King'

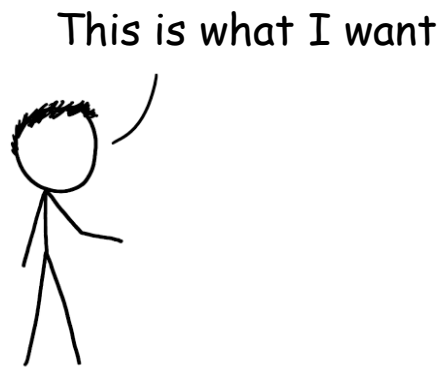
#	s	rv
	'Augusta Ada King'	''

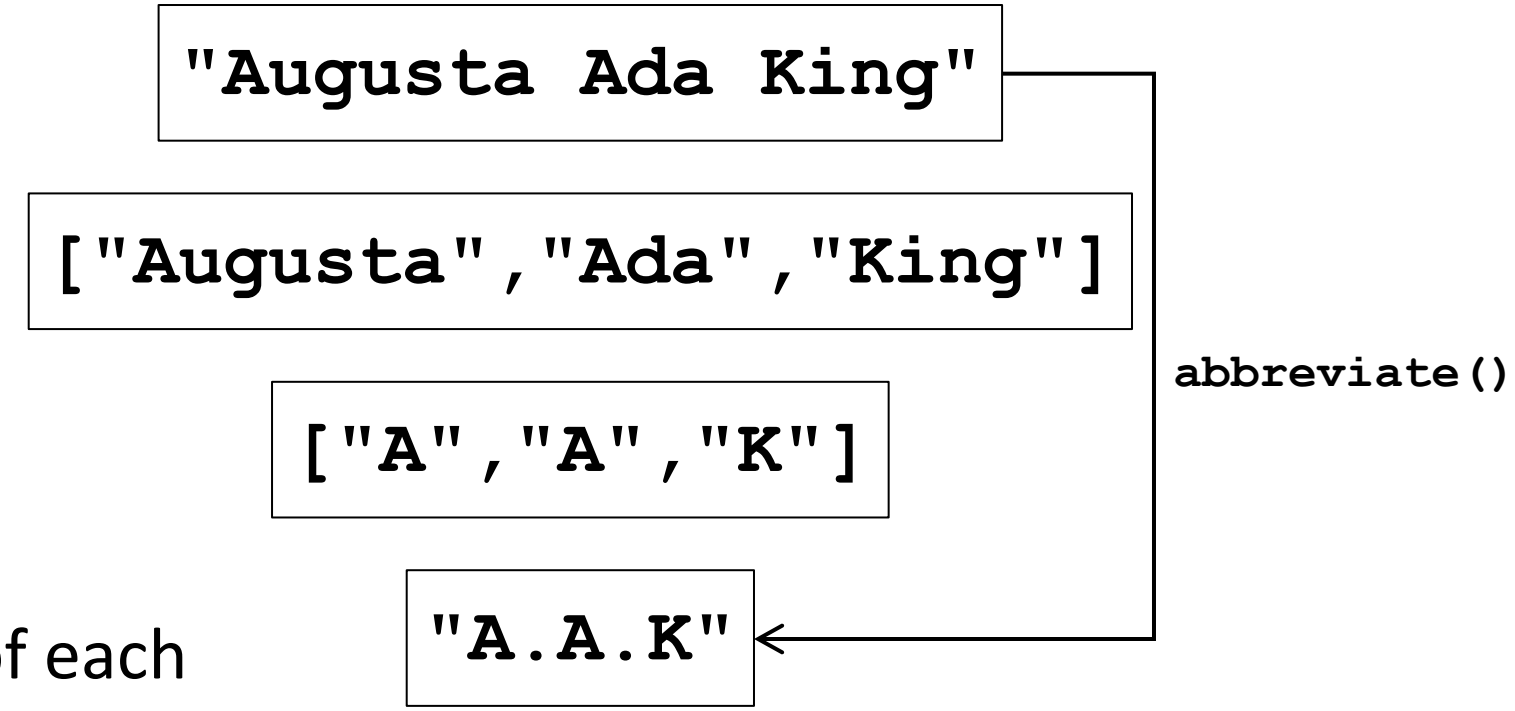
```
1  def initials(s):
2      letters = [var[0] for var in s.split(" ")]
3      return ''
4
5  initials('Augusta Ada King')
6
```

#	s	letters
	'Augusta Ada King'	['A', 'A', 'K']

#	s	letters	rv
	'Augusta Ada King'	['A', 'A', 'K']	''

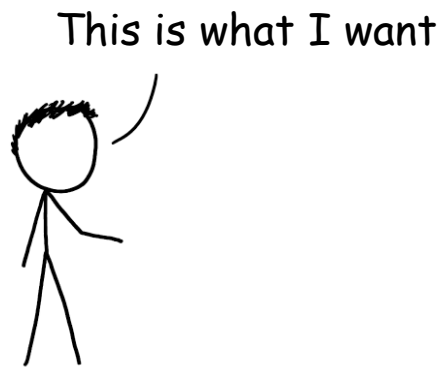
Task: Abbreviate



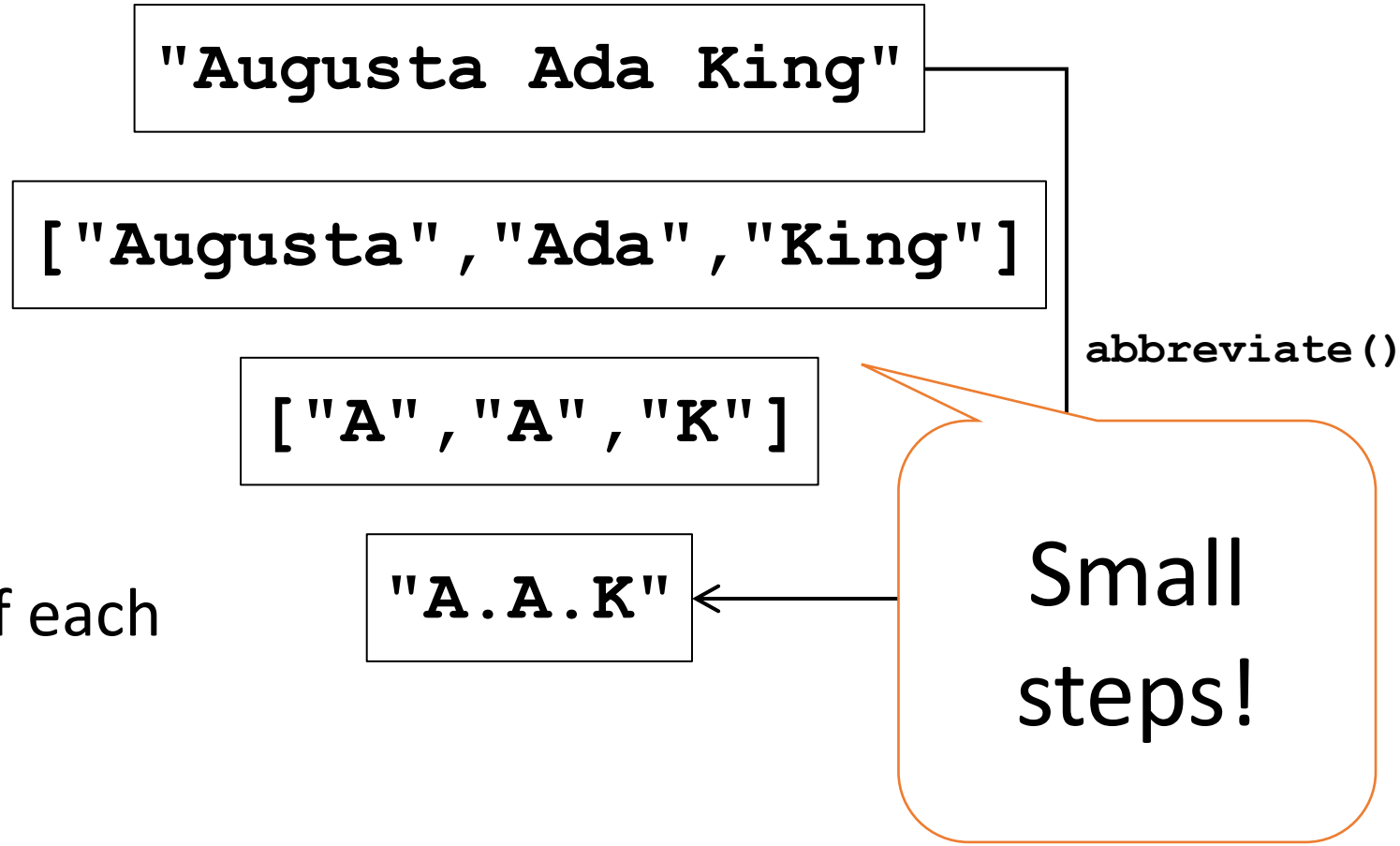


1. Split into words
2. Get the first letter of each
3. Put dots in between

Task: Abbreviate

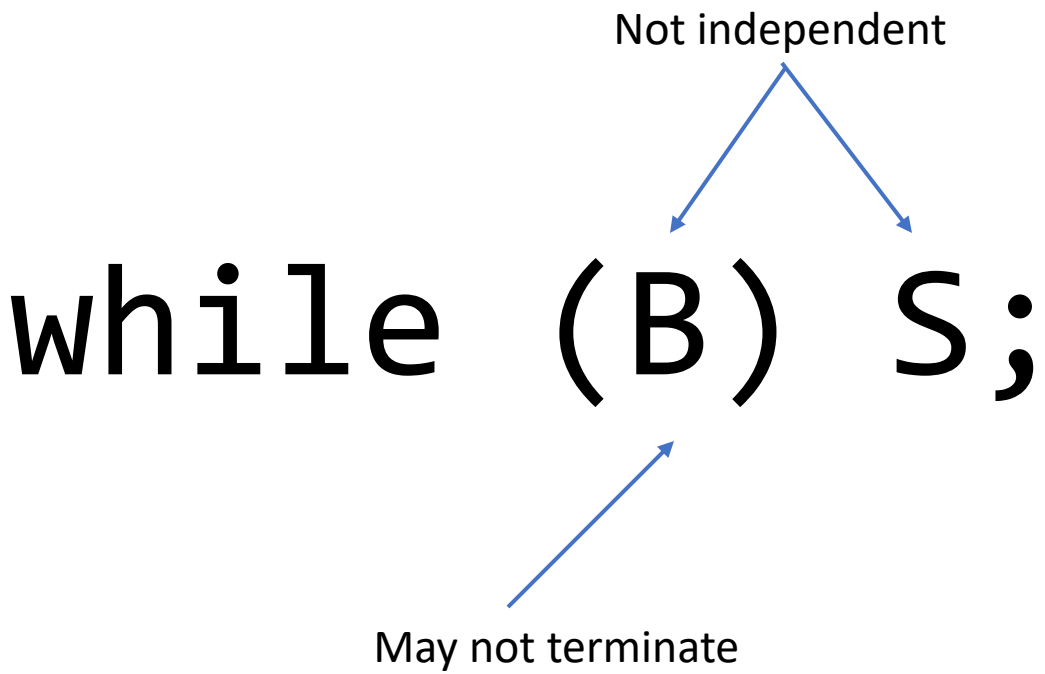


1. Split into words
2. Get the first letter of each
3. Put dots in between



Let’s make it harder: loops

Synthesizing loops



Reducing the class of programs

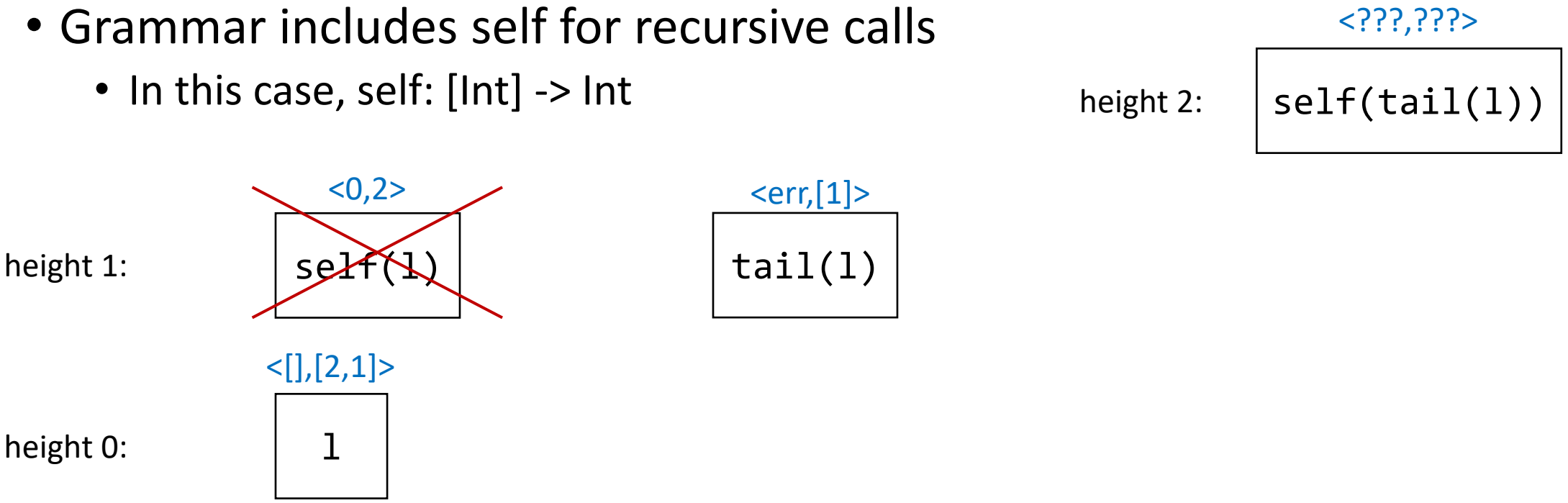
```
function f(args) {
  if (B) {
    return E;
  } else {
    self(args);
  }
}
```

```
input.map(e => E)
input.filter(e => B)
input.reduce((acc,e) => E)
```

```
for x in xs:
    v1 = E1
    v2 = E2
    ⋮
    vn = En
```

Recursion by Example

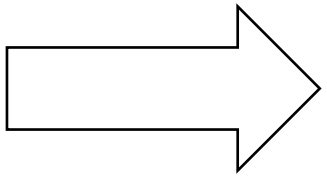
- Specification: $\{l \mapsto []\} \rightarrow 0$
  $\{l \mapsto [2,1]\} \rightarrow 2$
- Grammar includes self for recursive calls
  - In this case, self: [Int] -> Int



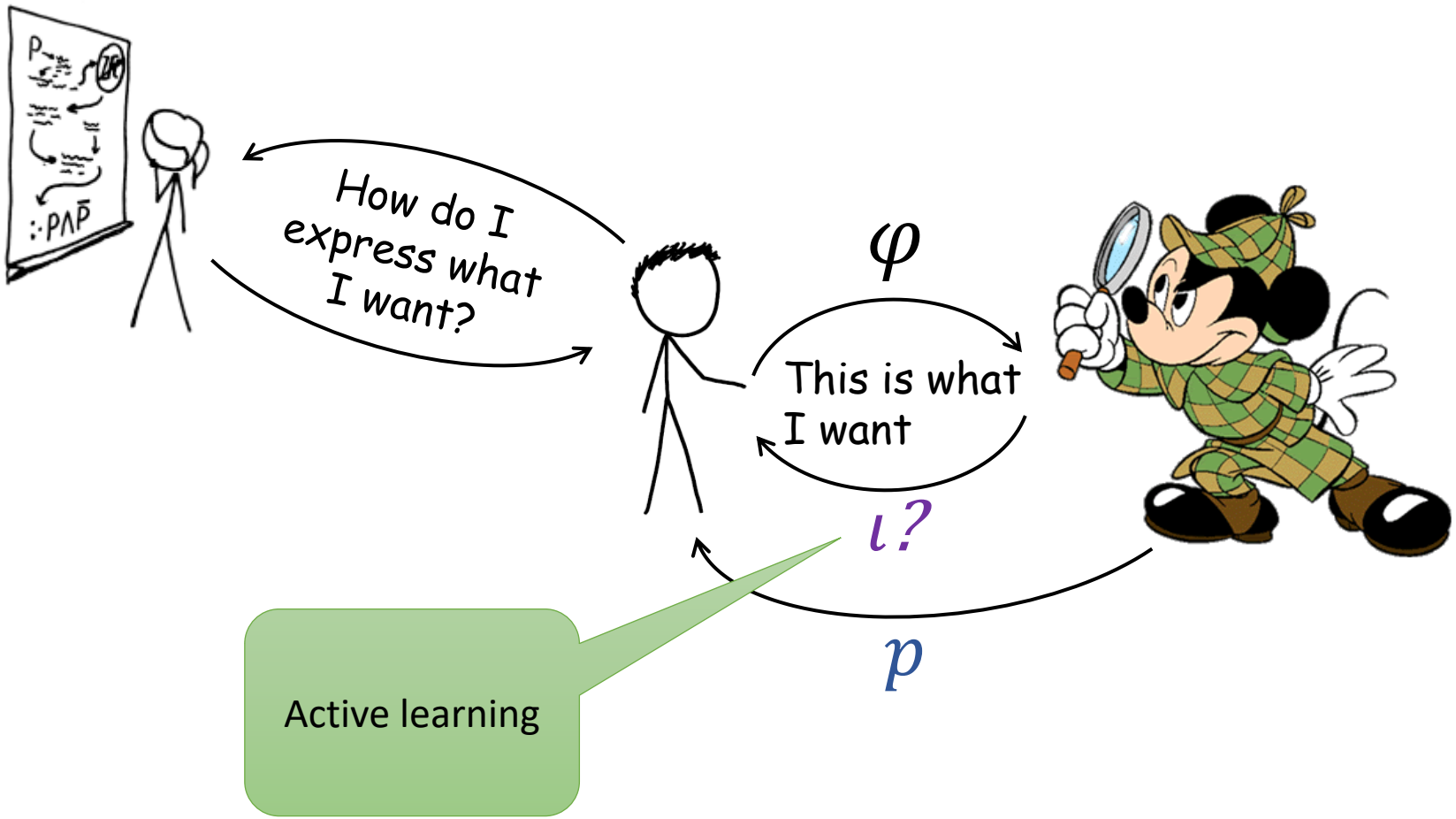
“Recursive Program Synthesis”, Albarghouthi et al. 2013

New values for self?

- You mean, new examples
- Where do these come from?



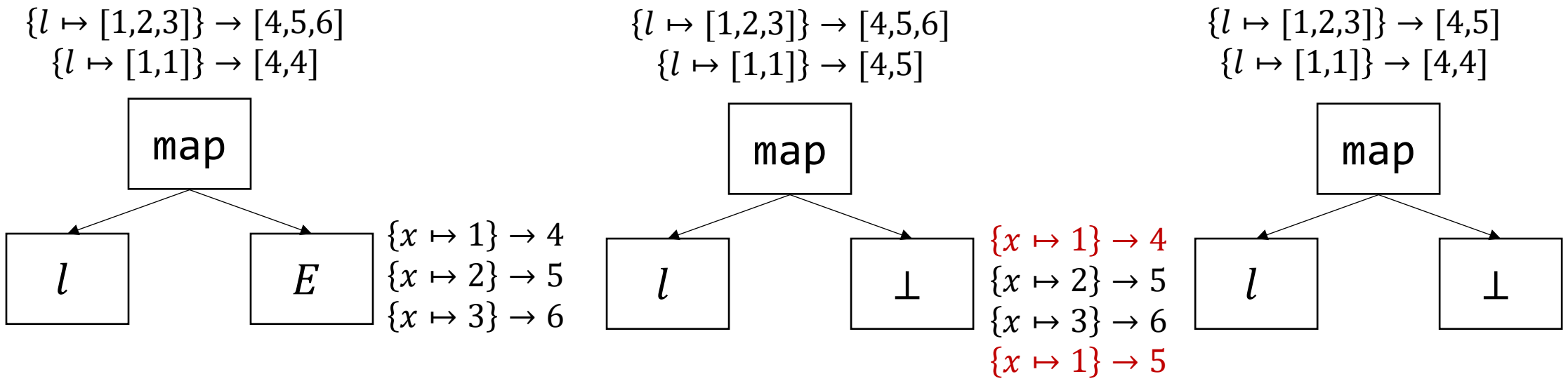
An interaction model!



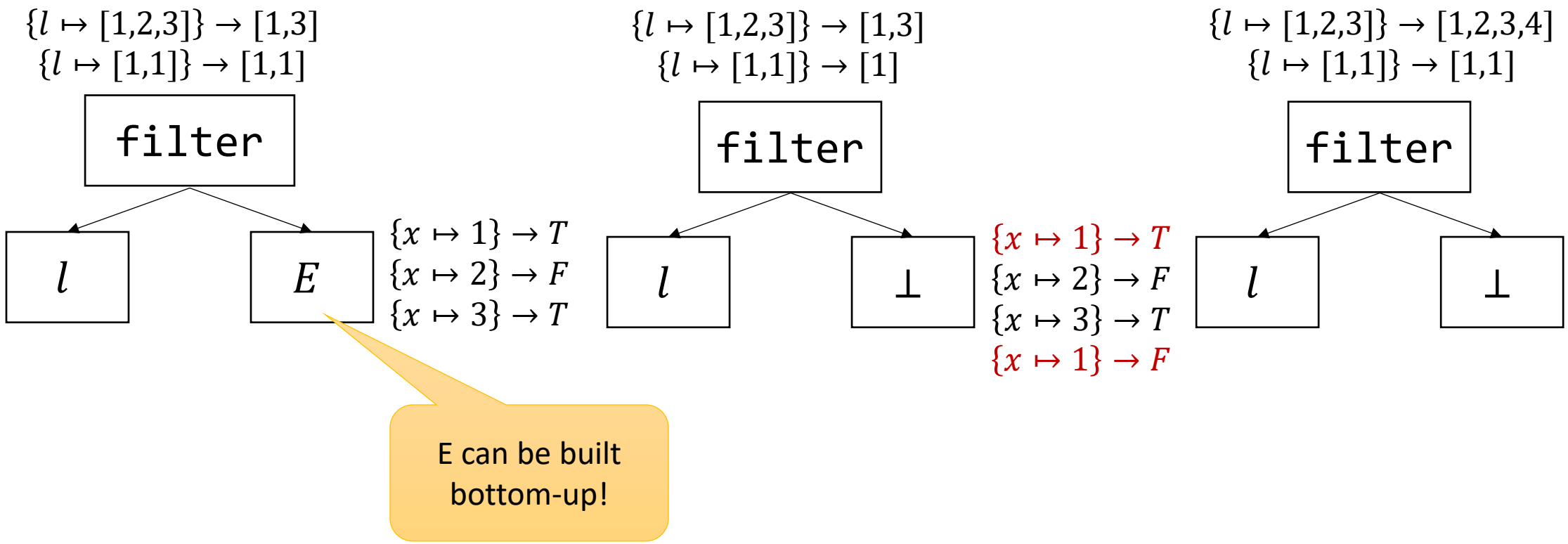
What about higher-order functions?

```
input.map(e => E)
input.filter(e => B)
input.reduce((acc,e) => E)
```

Reminder: Example propagation



Reminder: Example propagation



What about reduce?

Independent iterations are fine:

$\{arr \mapsto [1,2,3]\}$

```
[?? for x in arr]
```

$[2,3,4]$

??

$\{x \mapsto 1\} \to 2$

$\{x \mapsto 2\} \to 3$

$\{x \mapsto 3\} \to 4$

What about reduce?

Independent iterations are fine:

$\{arr \mapsto [1,2,3]\}$

```
[?? for x in arr]
```

$[2,3,4]$

?? $\{x \mapsto 1\} \to 2$

?? $\{x \mapsto 2\} \to 3$

?? $\{x \mapsto 3\} \to 4$

Data dependencies make everything hard:

$\{arr \mapsto [1,2,3]\}$

```
arr.reduce(0)((acc,x) => ??)
```

6

$\{x \mapsto 1, acc \mapsto 0\} \to ?_1$

$\{x \mapsto 2, acc \mapsto ?_1\} \to ?_2$

$\{x \mapsto 3, acc \mapsto ?_2\} \to 6$

What about reduce?

Independent iterations are fine:

$\{arr \mapsto [1,2,3]\}$

```
[?? for x in arr]
```

$[2,3,4]$

?? $\{x \mapsto 1\} \to 2$

$\{x \mapsto 2\} \to 3$

$\{x \mapsto 3\} \to 4$

Data dependencies make everything hard:

$\{arr \mapsto [1,2,3]\}$

```
arr.reduce(0)((acc,x) => ??)
```

6

$\{x \mapsto 1, acc \mapsto 0\} \to ?_1$

$\{x \mapsto 2, acc \mapsto ?_1\} \to ?_2$

$\{x \mapsto 3, acc \mapsto ?_2\} \to 6$

How many possible values do $?_1$ and $?_2$ have?

λ^2: Trace-Complete examples

$$\{arr \mapsto [1,2,3]\} \rightarrow 6$$
$$\{arr \mapsto [1,2]\} \rightarrow 3$$
$$\{arr \mapsto [1]\} \rightarrow 1$$

```
arr.reduce(0)((acc,x) => ??)
```

$$\Downarrow$$

$$\{x \mapsto 1, acc \mapsto 0\} \rightarrow ?_1$$
$$\{x \mapsto 2, acc \mapsto ?_1\} \rightarrow ?_2$$
$$\{x \mapsto 3, acc \mapsto ?_2\} \rightarrow 6$$

Synthesizing data structure transformations from input-output examples, Feser et al. 2015

λ^2: Trace-Complete examples

$$\{arr \mapsto [1,2,3]\} \rightarrow 6$$
$$\{arr \mapsto [1,2]\} \rightarrow 3$$
$$\{arr \mapsto [1]\} \rightarrow 1$$

```
arr.reduce(0)((acc,x) => ??)
```

$$\Downarrow$$

$$\{x \mapsto 1, acc \mapsto 0\} \rightarrow 1$$
$$\{x \mapsto 2, acc \mapsto 1\} \rightarrow ?_2$$
$$\{x \mapsto 3, acc \mapsto ?_2\} \rightarrow 6$$

Synthesizing data structure transformations from input-output examples, Feser et al. 2015

λ^2: Trace-Complete examples

$$\{arr \mapsto [1,2,3]\} \rightarrow 6$$
$$\{arr \mapsto [1,2]\} \rightarrow 3$$
$$\{arr \mapsto [1]\} \rightarrow 1$$

```
arr.reduce(0)((acc,x) => ??)
```

$$\Downarrow$$

$$\{x \mapsto 1, acc \mapsto 0\} \rightarrow 1$$
$$\{x \mapsto 2, acc \mapsto 1\} \rightarrow 3$$
$$\{x \mapsto 3, acc \mapsto 3\} \rightarrow 6$$

Synthesizing data structure transformations from input-output examples, Feser et al. 2015

λ^2: Trace-Complete examples

$$\{arr \mapsto [1,2,3]\} \rightarrow 6$$
$$\{arr \mapsto [1,2]\} \rightarrow 3$$
$$\{arr \mapsto [1]\} \rightarrow 1$$

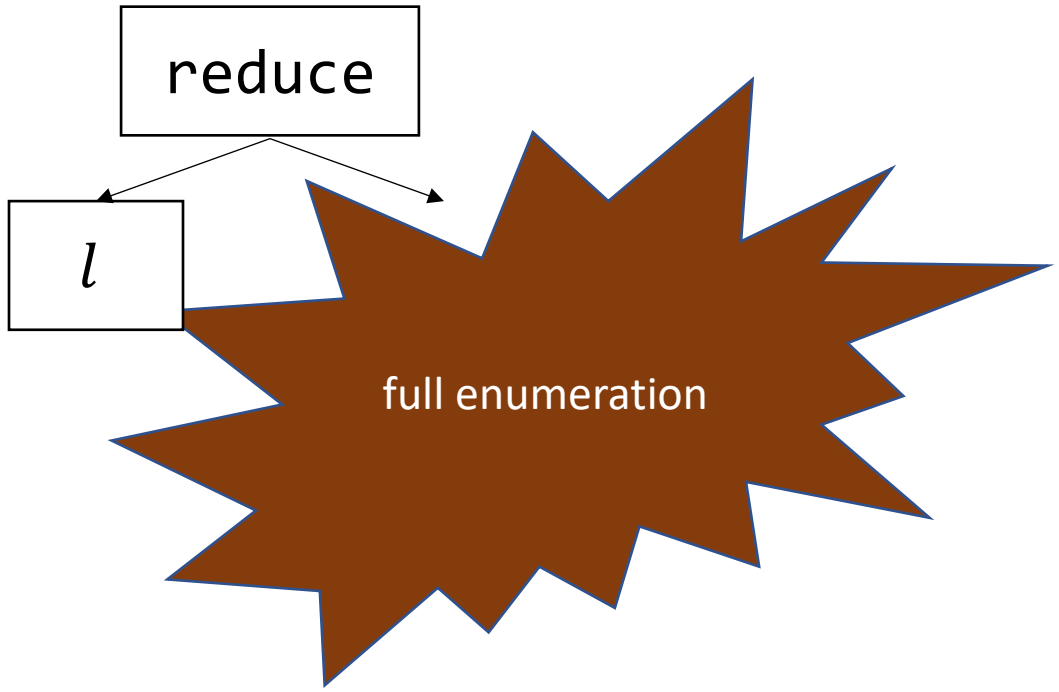
```
arr.reduce(0)((acc,x) => ??)
```

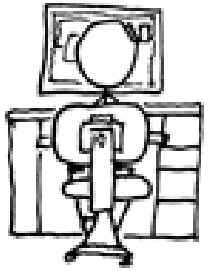

$$\{x \mapsto 1, acc \mapsto 0\} \rightarrow 1$$
$$\{x \mapsto 2, acc \mapsto 1\} \rightarrow 3$$
$$\{x \mapsto 3, acc \mapsto 3\} \rightarrow 6$$


```
arr.reduce(0)((acc,x) => x + acc)
```

Synthesizing data structure transformations from input-output examples, Feser et al. 2015

And if examples are not trace complete?





Live Programming by Example

```
1  def initials(s):
2      letters = ''
3      return ''
4
5  initials('Augusta Ada King')
6
```

#	s	letters
	'Augusta Ada King'	''

#	s	letters	rv
	'Augusta Ada King'	''	''

```
1  def initials(s):
2      letters = ''
3      return ''
4
5  initials('Augusta Ada King')
6
```

#	s	letters
	'Augusta Ada King'	['A', 'A', 'K']

#	s	letters	rv
	'Augusta Ada King'	''	''

```
1  def initials(s):
2      # Synthesizing. Please wait...
3      return ''
4
5  initials('Augusta Ada King')
6
```

#	s
	'Augusta Ada King'

#	s	rv
	'Augusta Ada King'	''

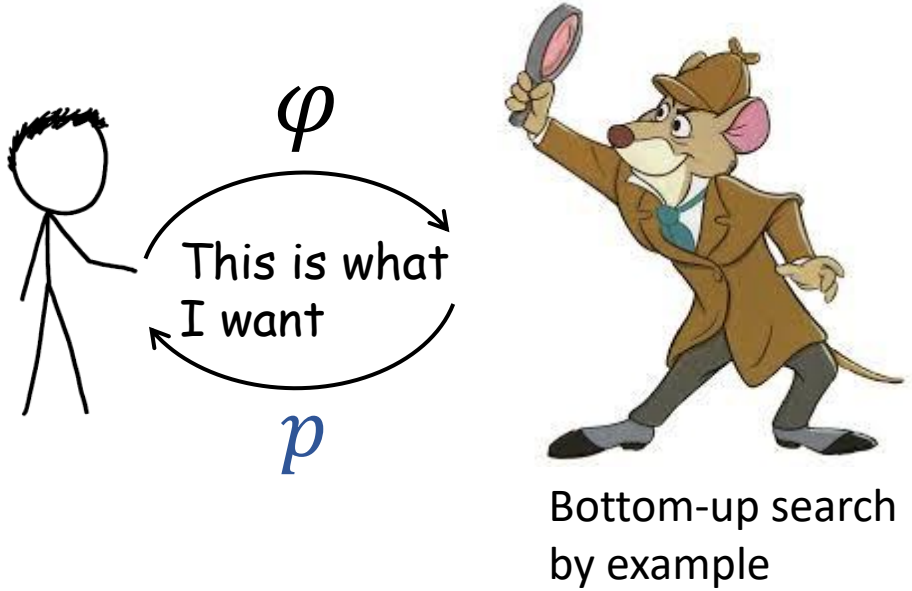
```
1  def initials(s):
2      letters = [var[0] for var in s.split(" ")]
3      return ''
4
5  initials('Augusta Ada King')
6
```

#	s	letters
	'Augusta Ada King'	['A', 'A', 'K']

#	s	letters	rv
	'Augusta Ada King'	['A', 'A', 'K']	''

What about loops?

- Trace-Complete examples will solve this
- How can we be sure the user gives us trace-complete examples?



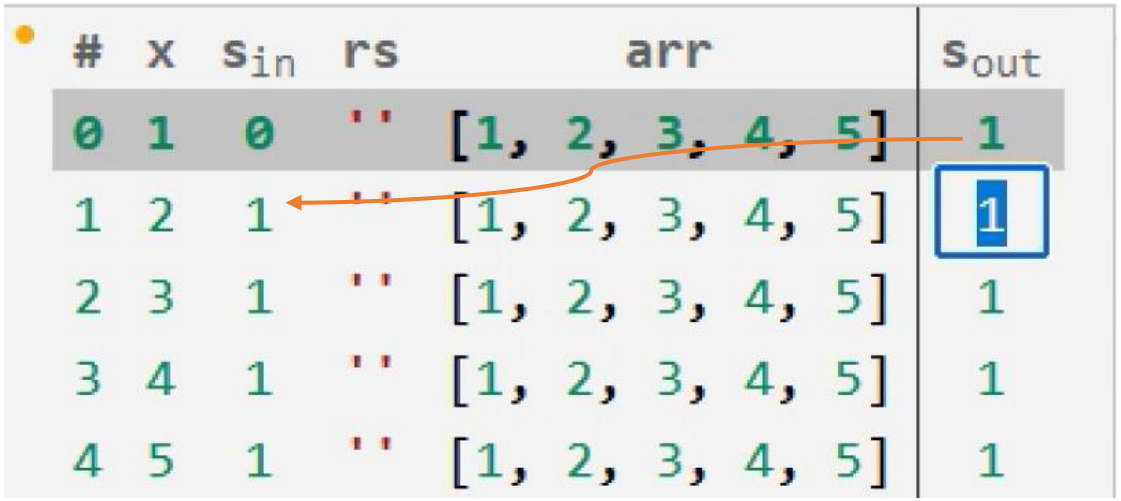
Bottom-up search by example

Interaction model to the rescue!

#	x	s	rs	arr
0	1	0	''	[1, 2, 3, 4, 5]
1	2	0	''	[1, 2, 3, 4, 5]
2	3	0	''	[1, 2, 3, 4, 5]
3	4	0	''	[1, 2, 3, 4, 5]
4	5	0	''	[1, 2, 3, 4, 5]

What happens here?

- Use the after-state to compute the next before-state
- Help the programmer enter examples



#	x	s_{in}	rs	arr	s_{out}
0	1	0	''	[1, 2, 3, 4, 5]	1
1	2	1	''	[1, 2, 3, 4, 5]	1
2	3	1	''	[1, 2, 3, 4, 5]	1
3	4	1	''	[1, 2, 3, 4, 5]	1
4	5	1	''	[1, 2, 3, 4, 5]	1

But more than one stmt happens in a block

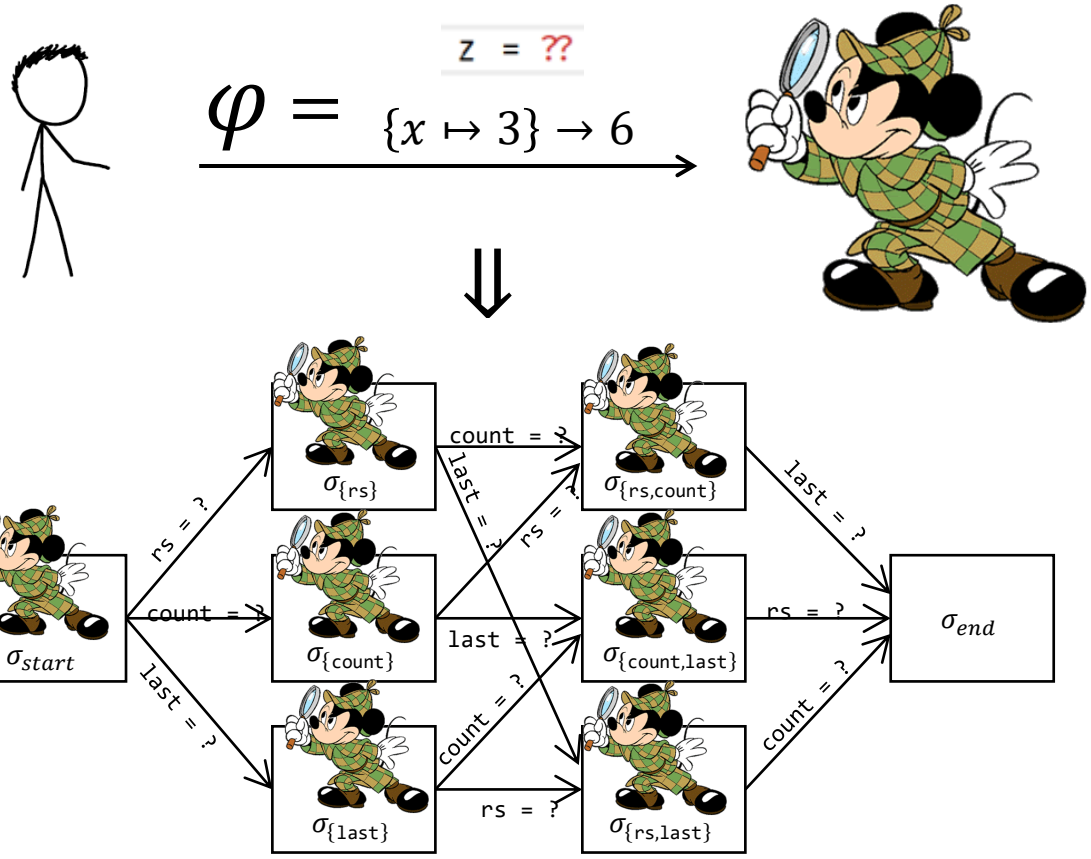
You have to specify all at once

```
5       for c in s[1:]:
6           if c == last:
7               count += 1
8           else:
9               rs += str(count) + last
10              count = 1
11              last = c
12      return rs
```

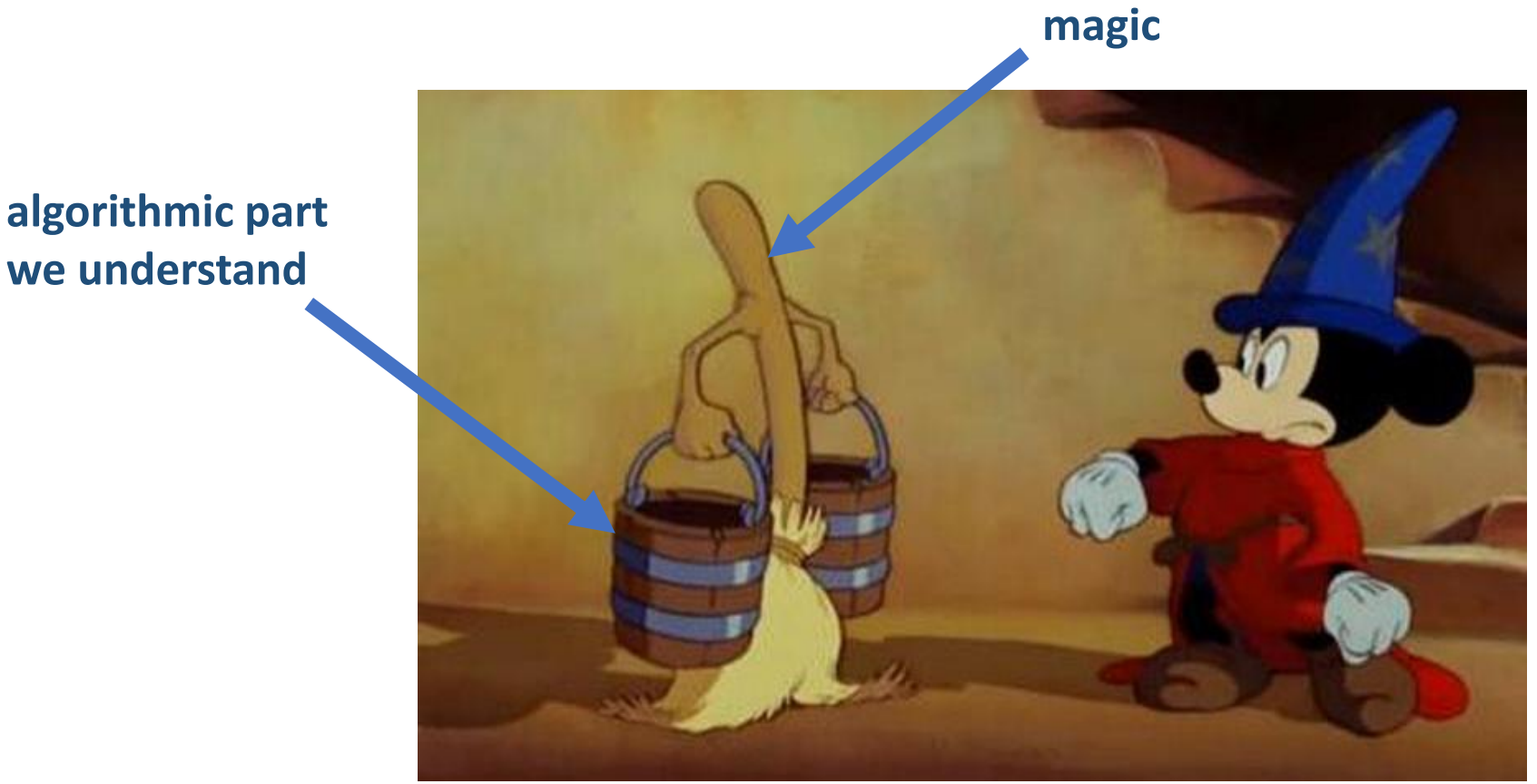
⇓

```
6           if last == c:
7               count += 1
8           else:
9               last, count, rs = ??
10      return rs
```

Now to synthesize it:



Formal methods: a bird's eye view



Magic, an incomplete definition

Something that solves really hard problems…
… some of the time.

“incomplete assistants”

Available magics

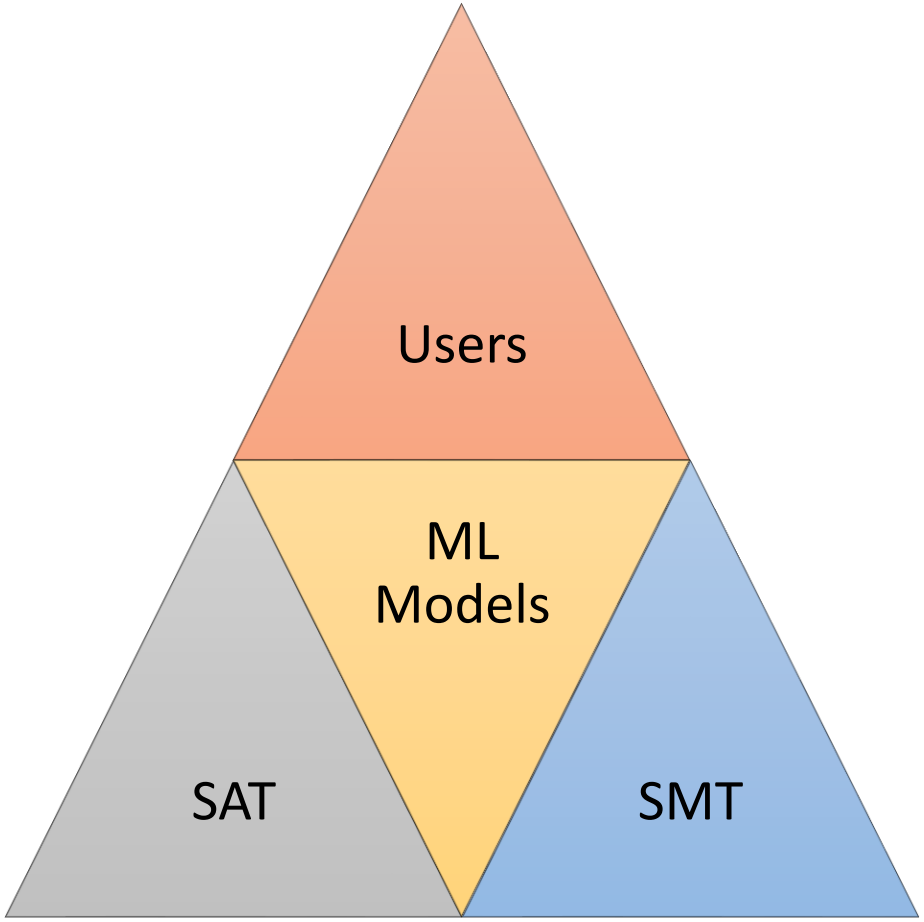
Hard subproblem

Model?

Available magics

Hard subproblem

Model?



Summary

- Think about interactions, not just algorithms
- Both will change once you start doing that
  - For the better, if you do it right!
- You can help both sides of the interaction (human, algorithm) out