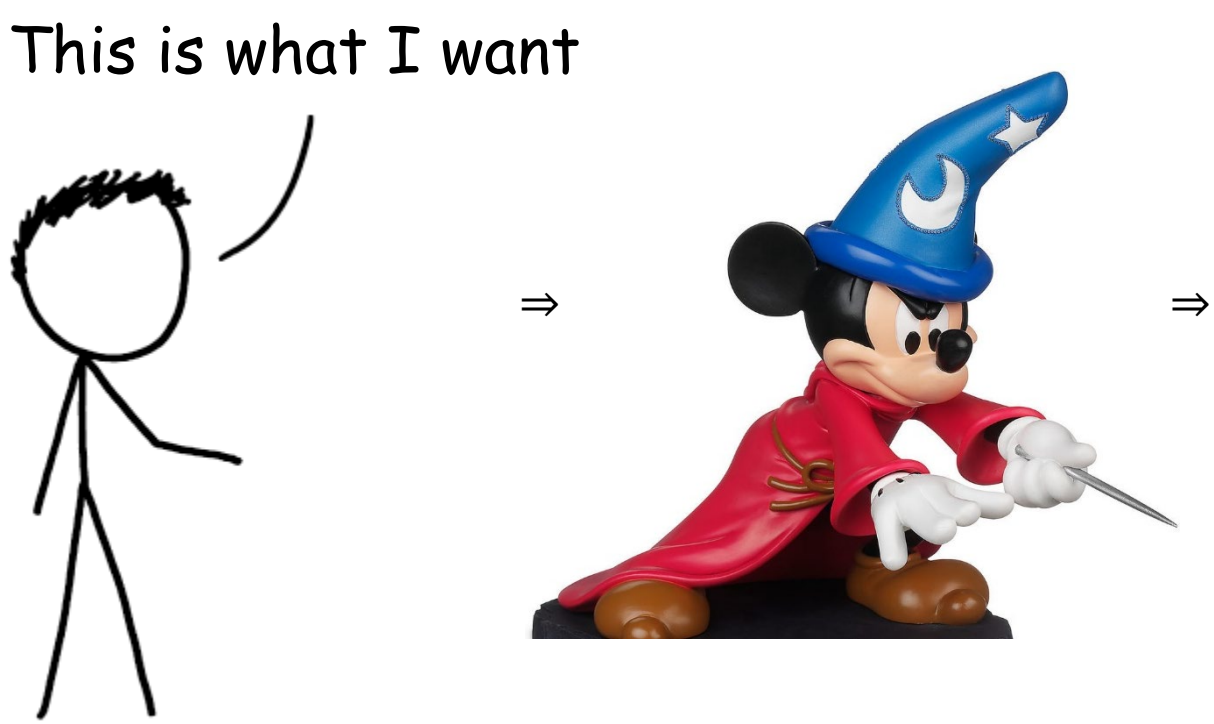
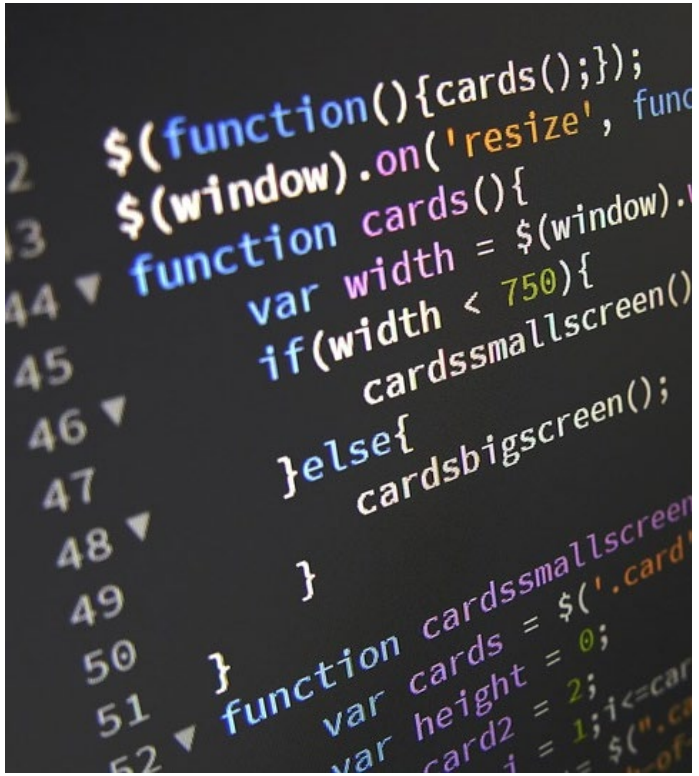
Program Synthesis Co-Design


Hila Peleg – Technion


@HilaCodes

Program Synthesis



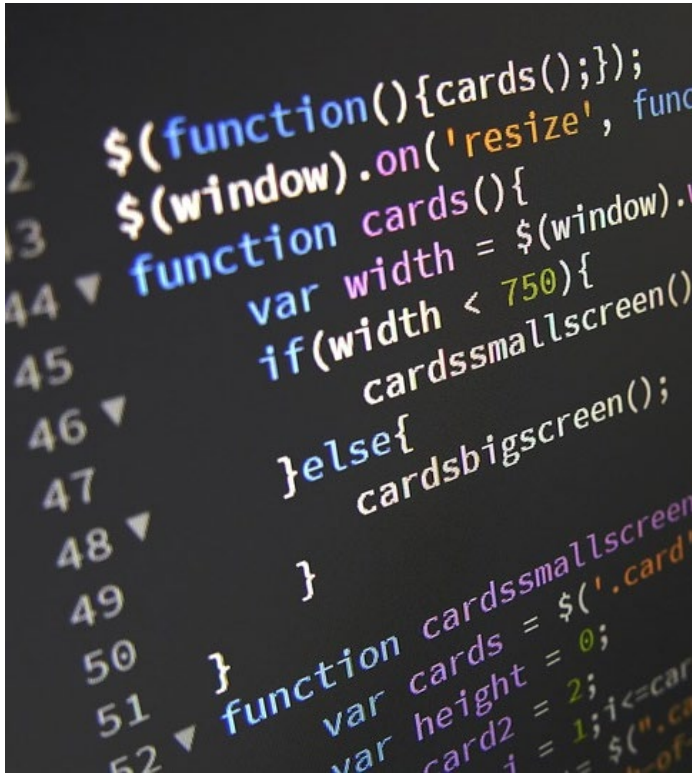


I’ll tell you what I want

```
Int -> [Maybe a] -> a
```

⇒

⇒

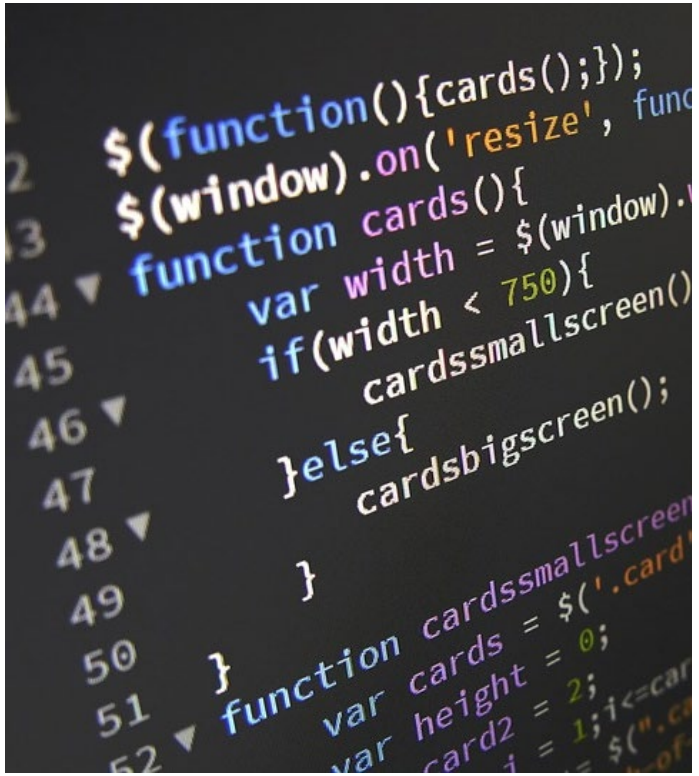


I’ll tell you what I want

Looks like `x.f(a,b,???)`

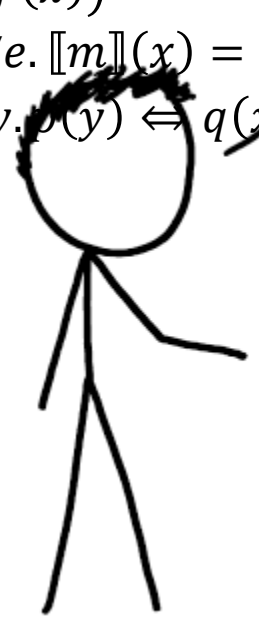
⇒

⇒



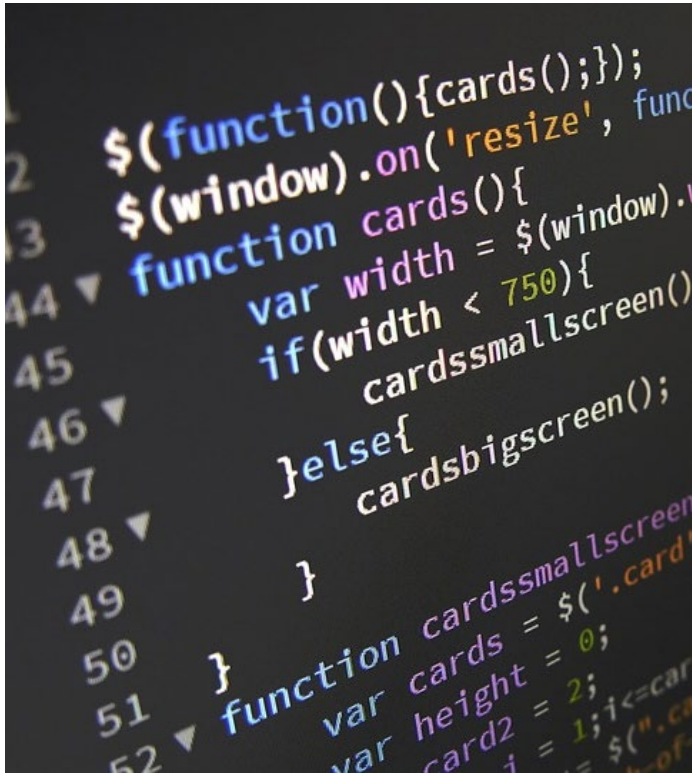
I'll tell you what I want

$$\exists m.\, \forall x.\, \phi(x) \wedge \neg\psi(x)$$
$$\vee\, \psi\big(f(x)\big)$$
$$\Rightarrow \big(\forall e.\, [\![m]\!](x) = e \Rightarrow \mathcal{V}(e)\big)$$
$$\wedge \big(\exists y.\, p(y) \Leftrightarrow q(x)\big) \vee \tau(m)$$
$$= \tau_1$$



⇒

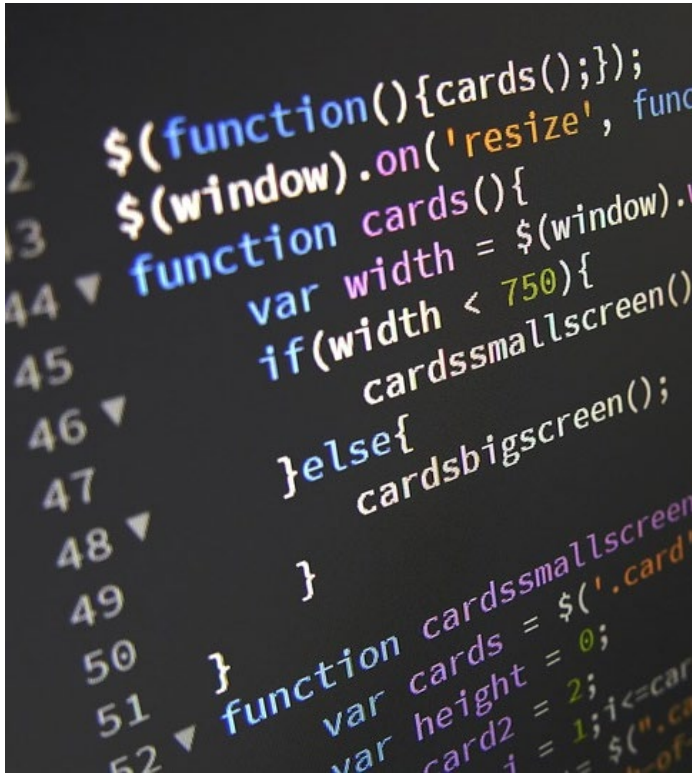
⇒



I’ll tell you what I want

⇒

⇒

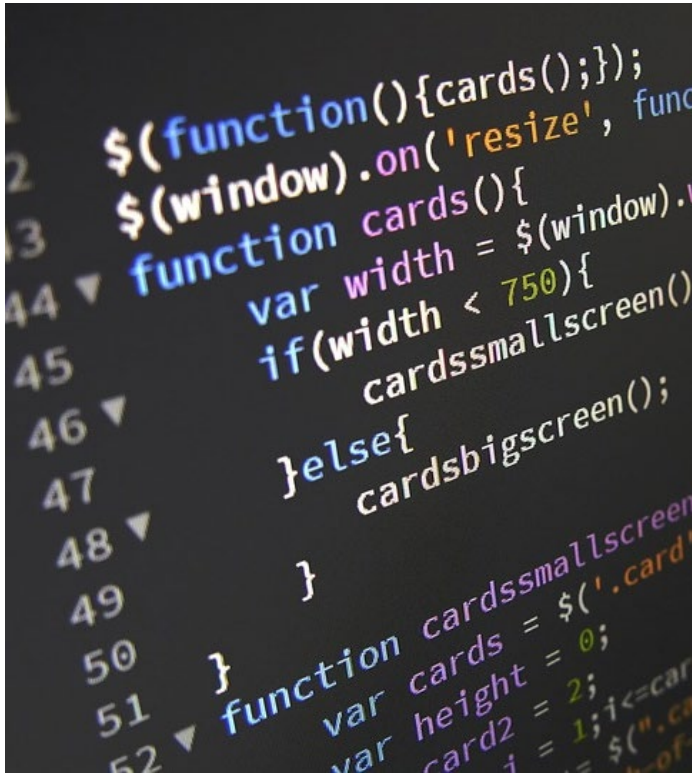


Program Synthesis

φ

$\Rightarrow$

$\Rightarrow$



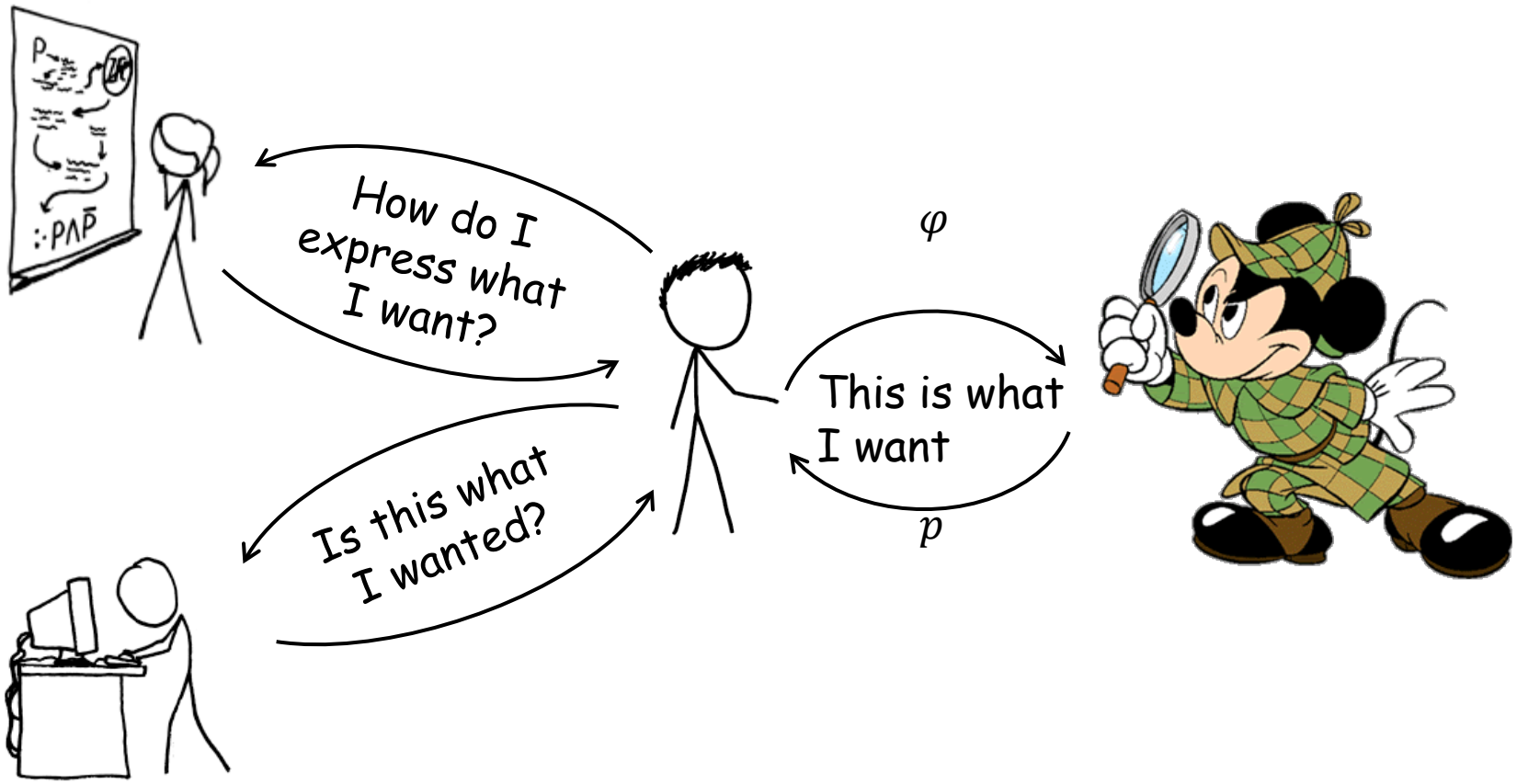
The search is highly customized

Search by example

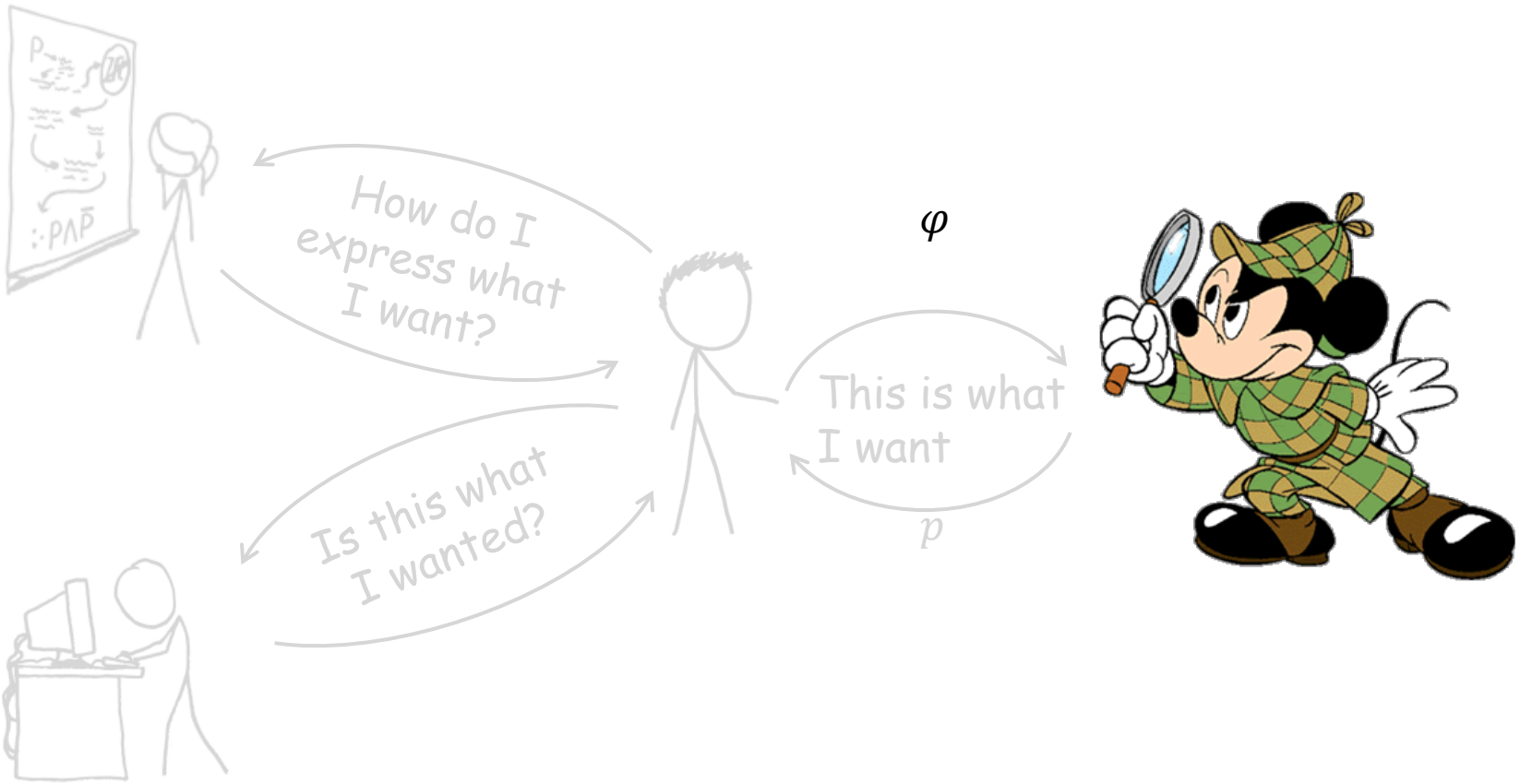
Search by type

Search by natural language

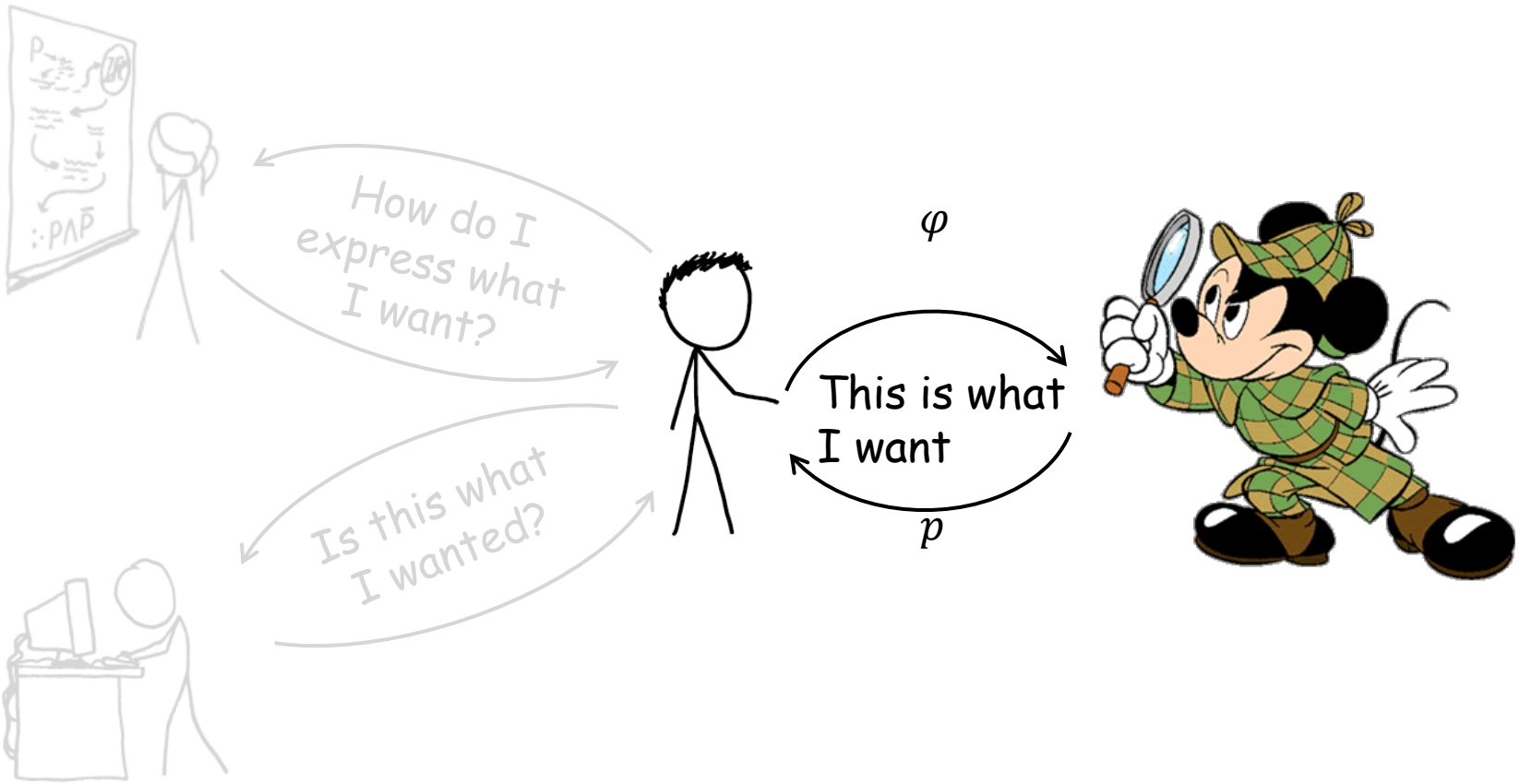
Interaction models



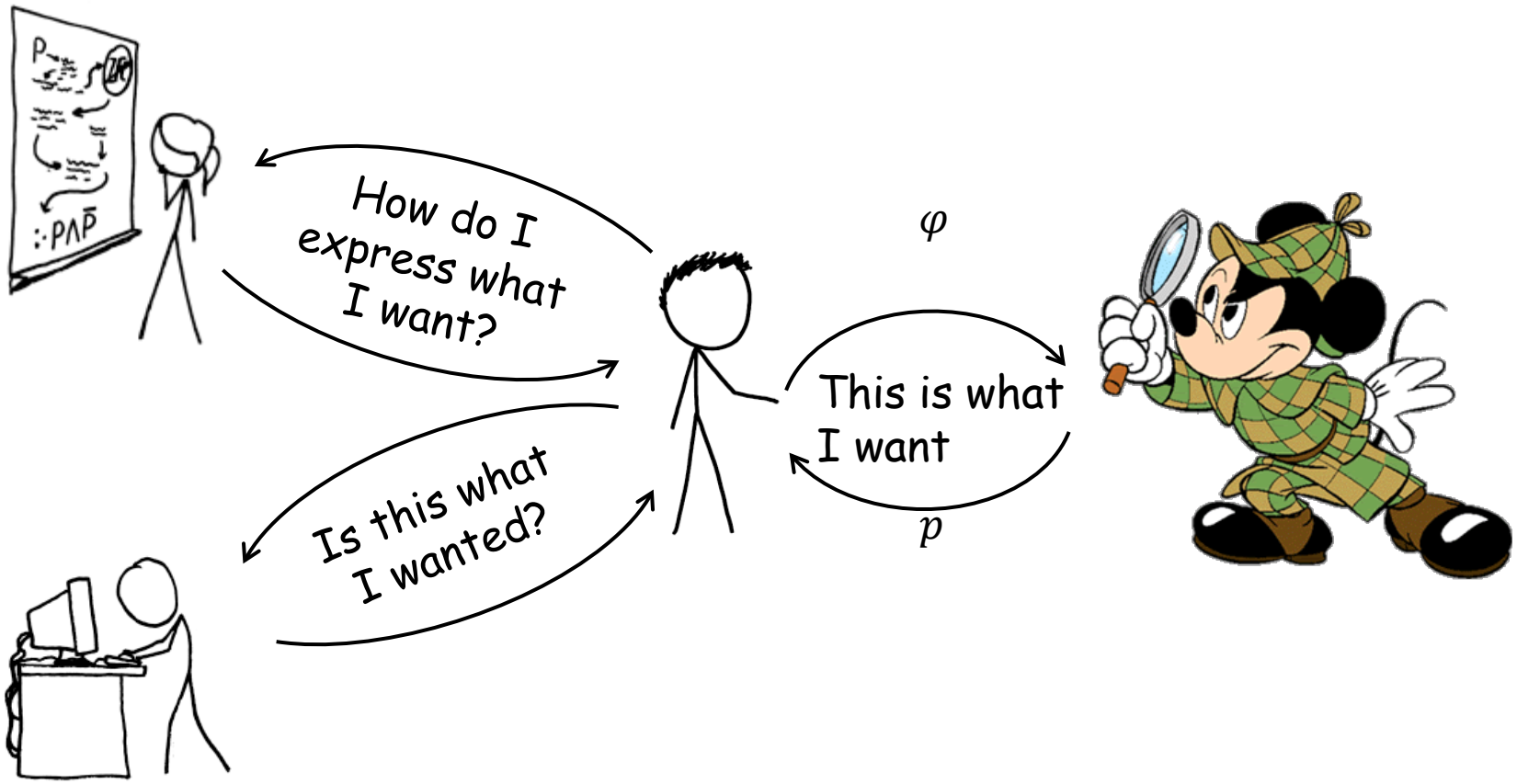
Interaction models are part of the package



Interaction models are part of the package



Interaction models are part of the package



Example I: interaction ideas

Interaction idea: assignment order is hard for users

```
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
```

LooPy: interactive program synthesis with control structures [OOPSLA'21]

Example I: interaction ideas

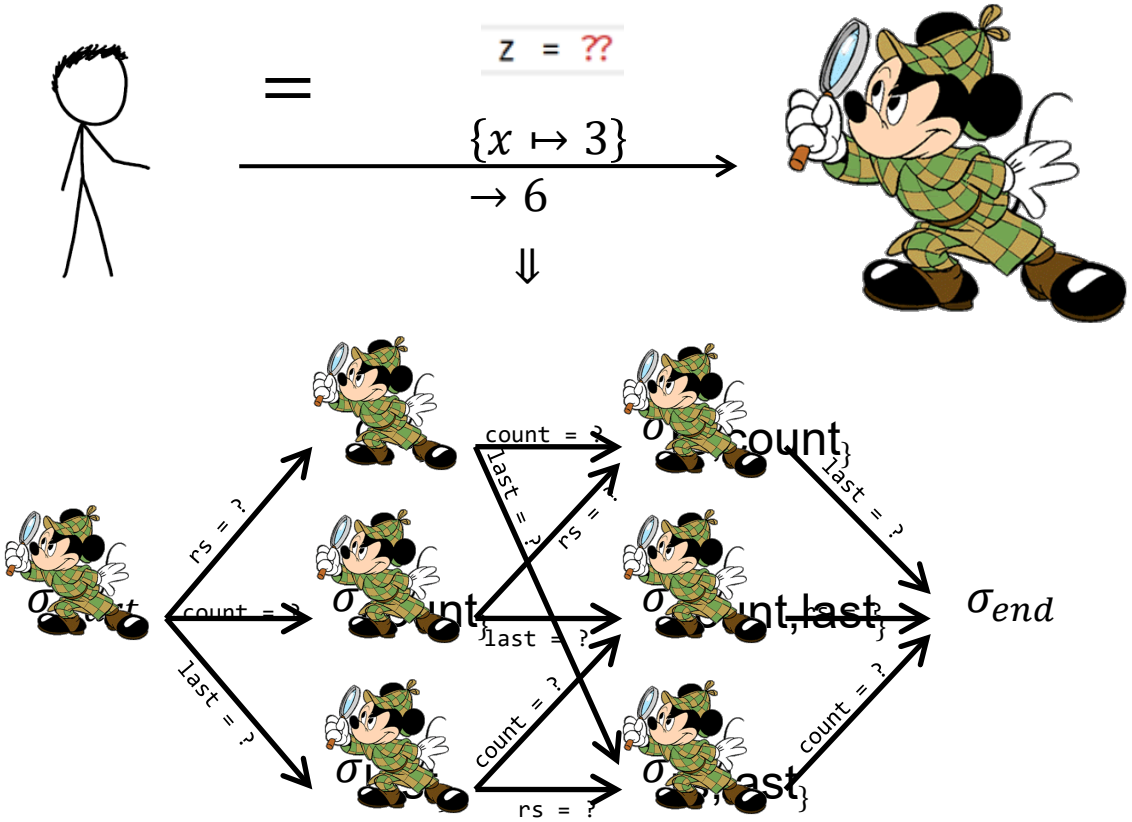
Interaction idea: assignment order is hard for users

```
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:



LooPy: interactive program synthesis with control structures [OOPSLA'21]

Example II: synthesizer constraints

Independent iterations are fine:

```
arr2 = [?? for x in arr]
```

⇓

```
??
```

Example II: synthesizer constraints

Independent iterations are fine:

```
arr2 = [?? for x in arr]
```

⇓

```
??
```

Data dependencies make everything hard:

```
s = 0
for x in arr:
    s = ??
```

⇒

Example II: synthesizer constraints

Independent iterations are fine:

```
arr2 = [?? for x in arr]
```

⇓

```
??
```

Data dependencies make everything hard:

```
s = 0
for x in arr:
    s = ??
```

⇒

Common solution: trace-complete specifications

$$\{arr \mapsto [], s \mapsto 0\} \to 0$$
$$\{arr \mapsto [1], s \mapsto 0\} \to 1$$
$$\{arr \mapsto [1,2], s \mapsto 1\} \to 3$$

LooPy: interactive program synthesis with control structures [OOPSLA'21]

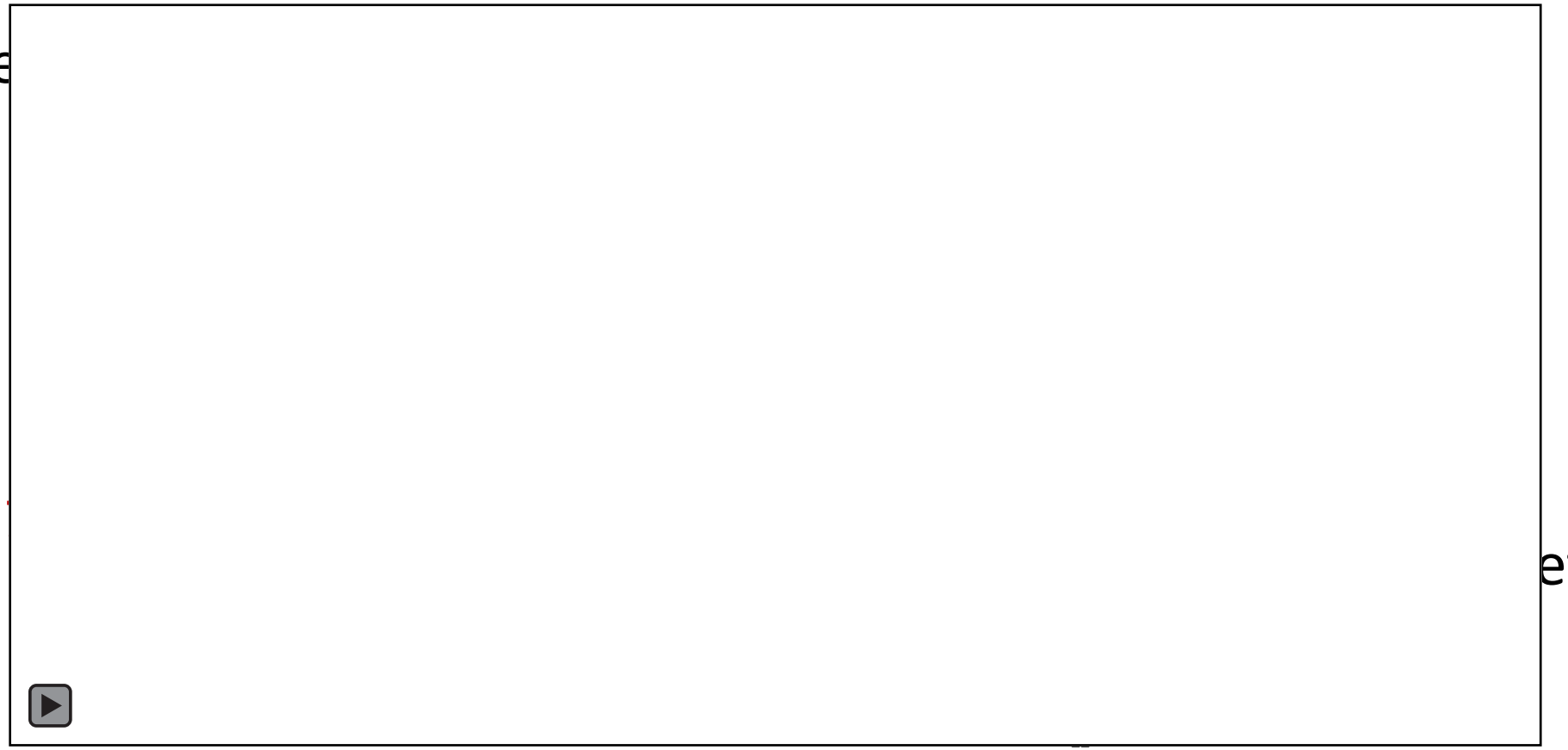
Example II: synthesizer constraints

Independe

```
arr2 = [??
```

?

ete

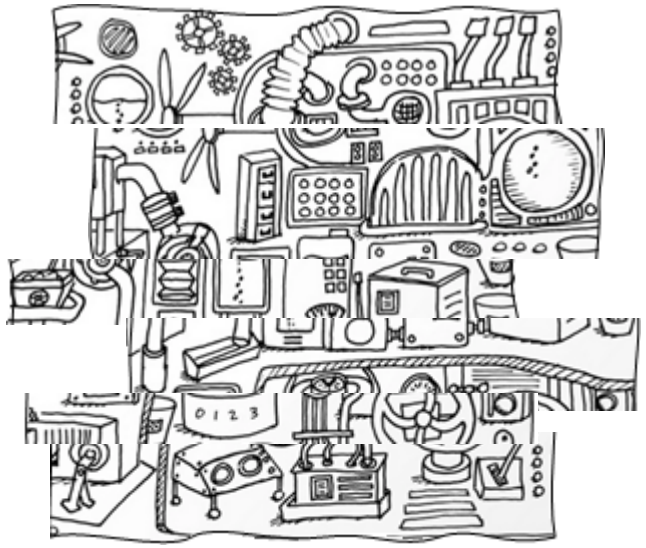


$\{arr \mapsto [1], s \mapsto 0\} \to 1$

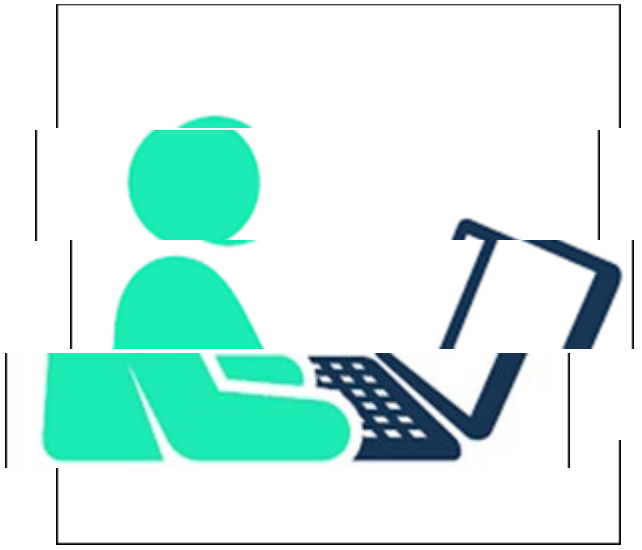
$\{arr \mapsto [1,2], s \mapsto 1\} \to 3$

LooPy: interactive program synthesis with control structures [OOPSLA'21]

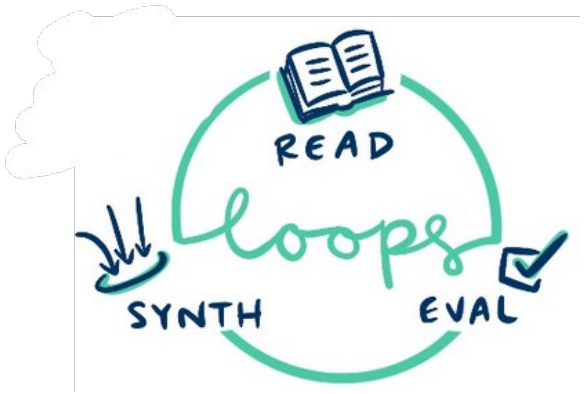
Synthesis Co-Design



synthesizer



user interaction

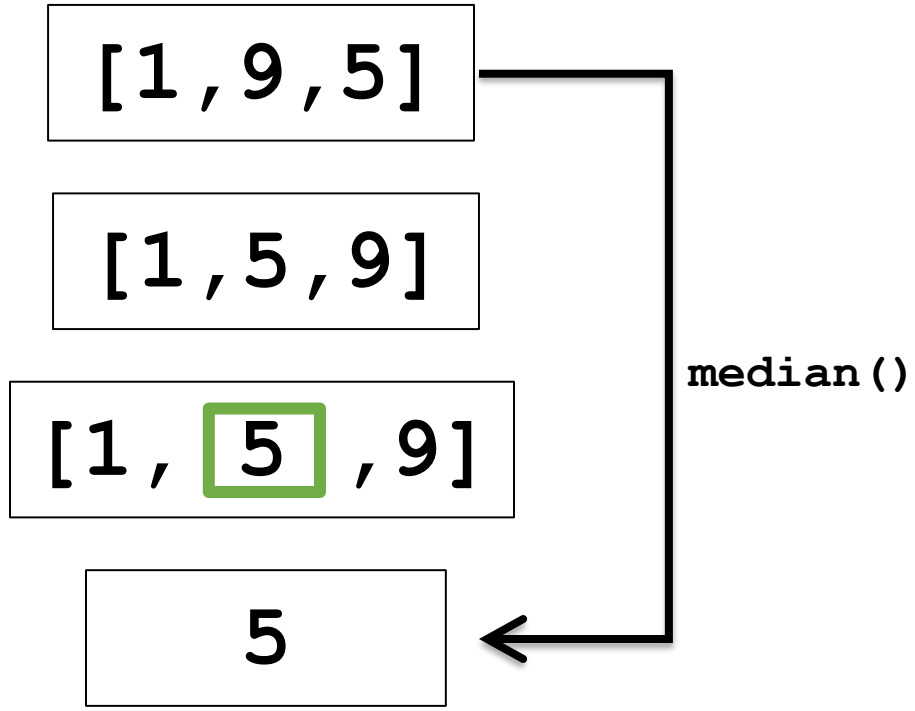


Example 1: Read-Eval-Synth Loop

Programming with a Read-Eval-Synth Loop [OOPSLA'20]

Problem: Median (simple variant)

1. Order the elements
2. Find the middle
3. Get the element




```
input.sort()[Math.floor(input.length / 2)]
```

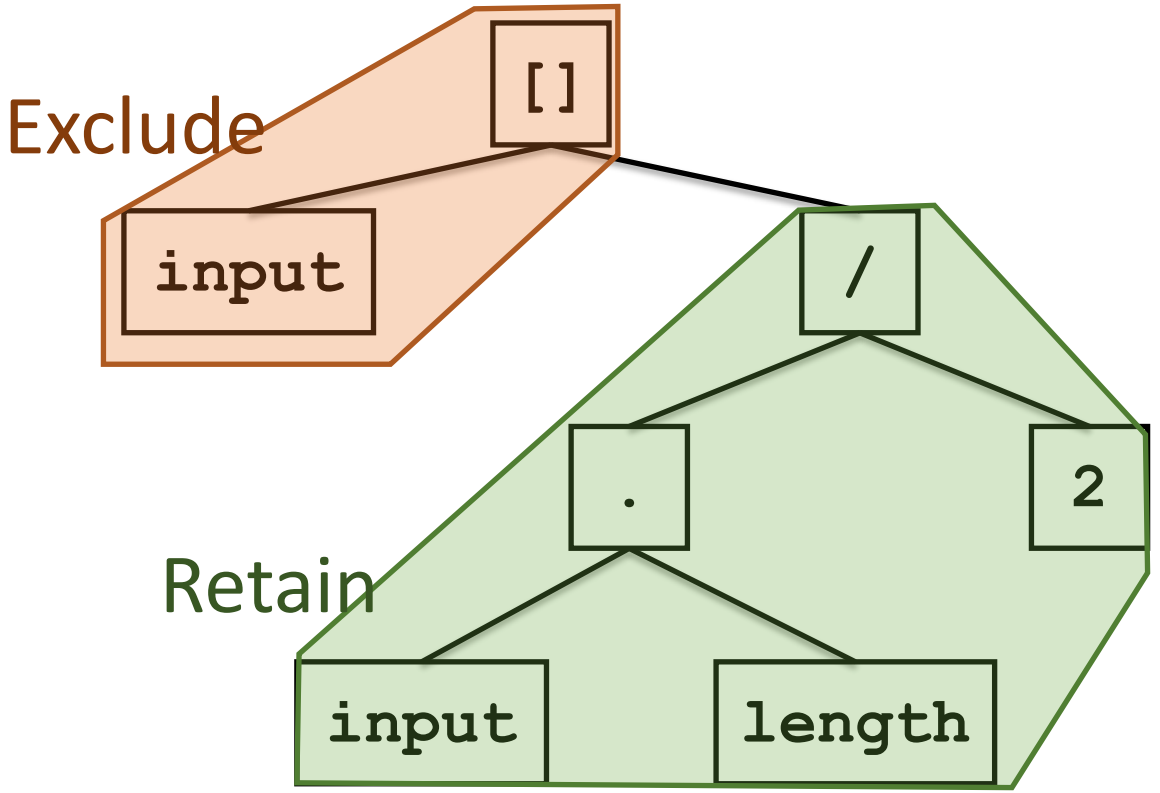
RESL: REPLs meet local specifications

Programming with a Read-Eval-Synth Loop: Peleg, Gabai, Itzhaki, Yahav [OOPSLA'20]

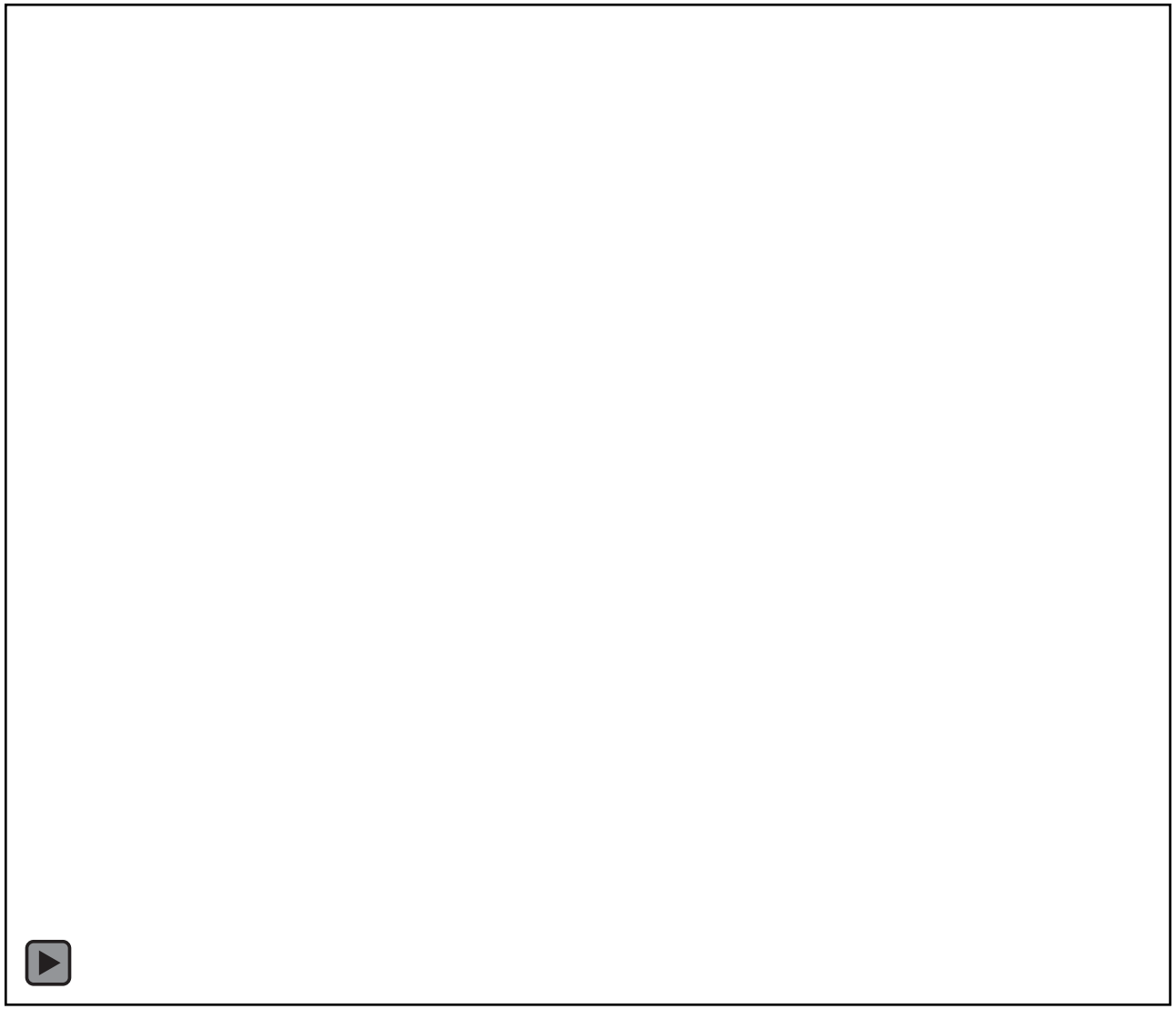
Read-Eval-Print Loops

```
> input = [7,8,7,2]
[ 7, 8, 7, 2 ]
> input[input.length - 1]
2
> input[input.length / 2]
7
> input = [1,3,2]
[ 1, 3, 2 ]
> input[input.length / 2]
undefined
> input.length
3
> input.length / 2
1.5
>
```

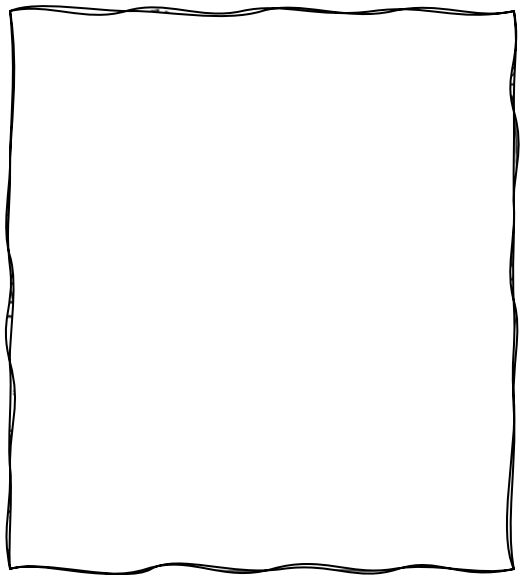
Syntactic code-review-like specs



Read-Eval-Synth Loop (RESL)



RESL challenges



synthesizer

Challenge 1: Local specifications interfere with synthesis algorithm

Challenge 2: Loops are hard for the synthesizer

user interaction

Challenge 1: Synthesis breaks

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

Retain(`input.length / 2`)

No program

`input.sort()[Math.floor(input.length / 2)]`
is in the synthesizer's space…

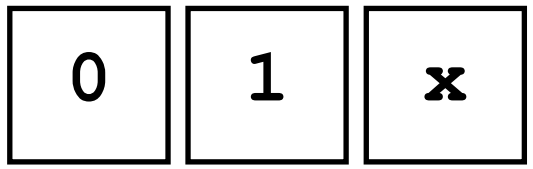
Searching the space by enumeration

height 0

$$S \rightarrow \quad E$$
$$E \rightarrow \quad 0|1|x$$

Searching the space by enumeration

height 0



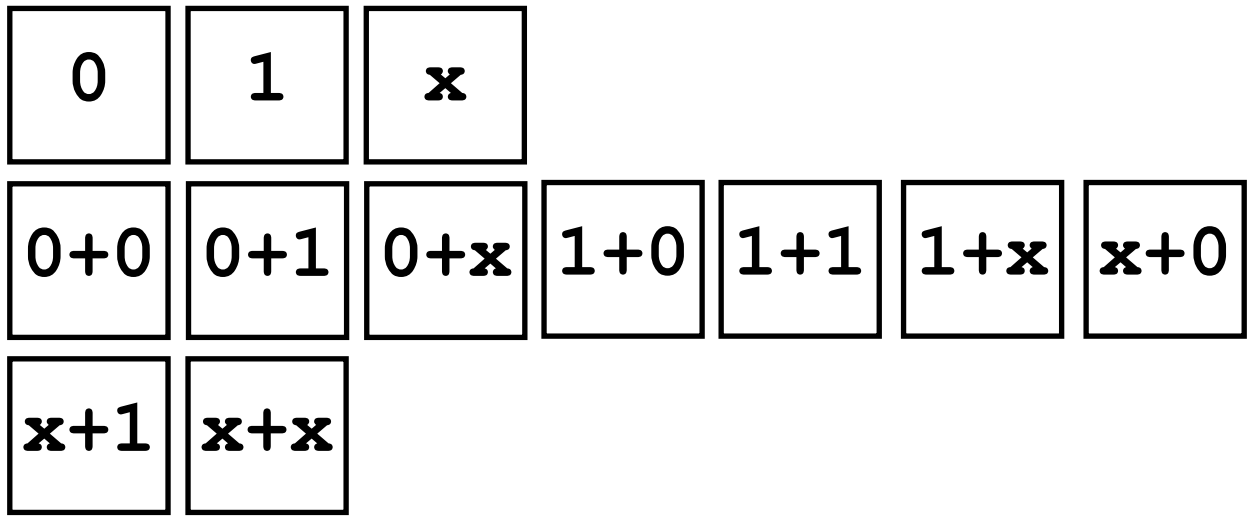
$S \rightarrow \quad E$

$E \rightarrow \quad 0|1|x$

Searching the space by enumeration

height 0

height 1



$S \rightarrow \quad E$

$E \rightarrow \quad 0|1|x$

Searching the space by enumeration

height 0

`0`
`1`
`x`

height 1

`0+0`	`0+1`	`0+x`	`1+0`	`1+1`	`1+x`	`x+0`
`x+1`	`x+x`	`0*0`	`0*1`	`0*x`	`1*0`	`1*1`
`1*x`	`x*0`	`x*1`	`x*x`			

$S \rightarrow \quad E$

$E \rightarrow \quad 0|1|x$

Searching the space by enumeration

height 0

0	1	x

height 1

0+0	0+1	0+x	1+0	1+1	1+x	x+0
x+1	x+x	0*0	0*1	0*x	1*0	1*1
1*x	x*0	x*1	x*x			

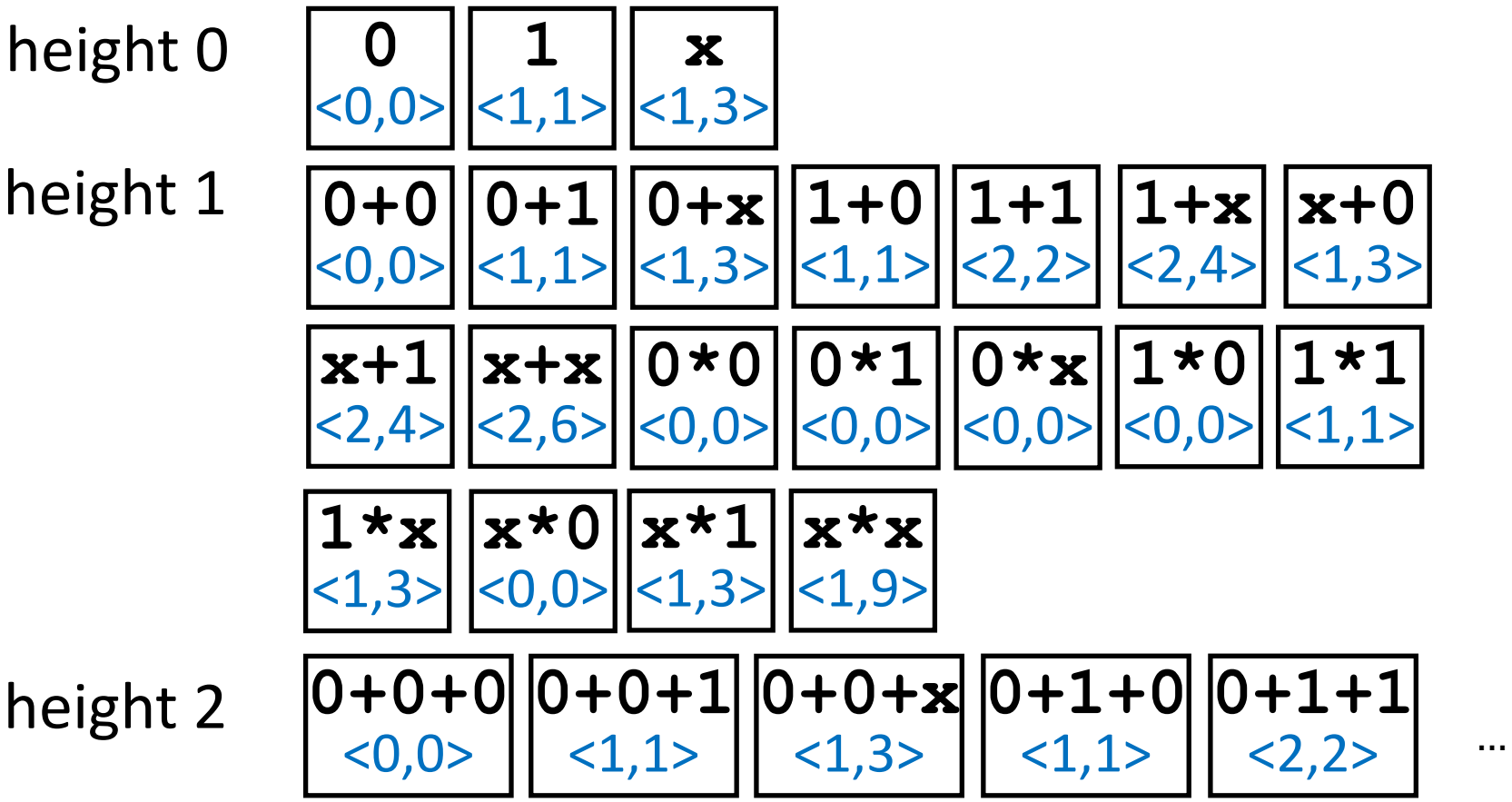
height 2

0+0+0	0+0+1	0+0+x	0+1+0	0+1+1	…

$S \rightarrow \quad E$

$E \rightarrow \quad 0|1|x$

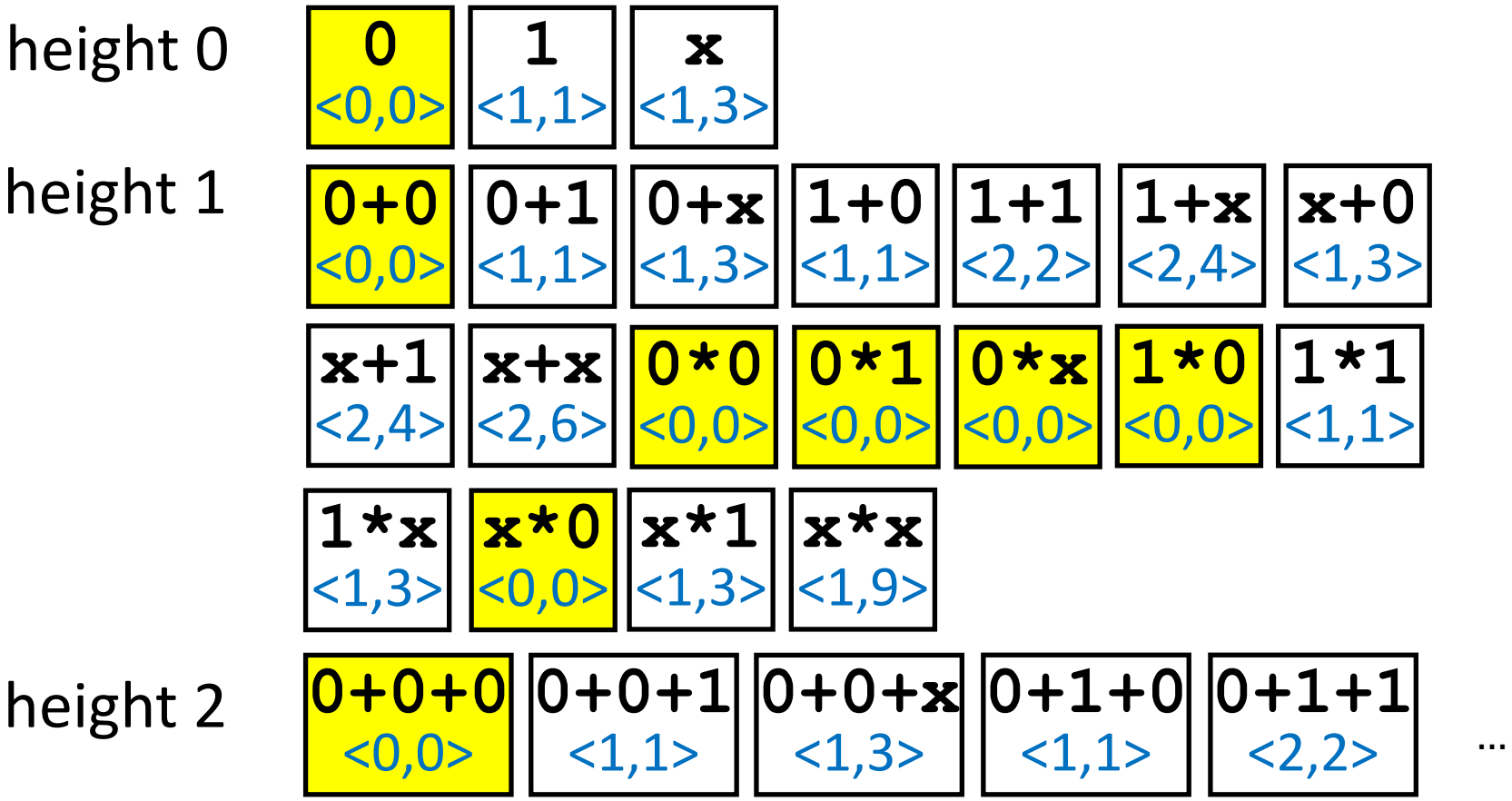
Searching the space by enumeration



$$S \rightarrow \quad E$$
$$E \rightarrow \quad 0|1|x$$

$$i_0 = \{x \mapsto 1\}$$
$$i_1 = \{x \mapsto 3 \backslash\}$$

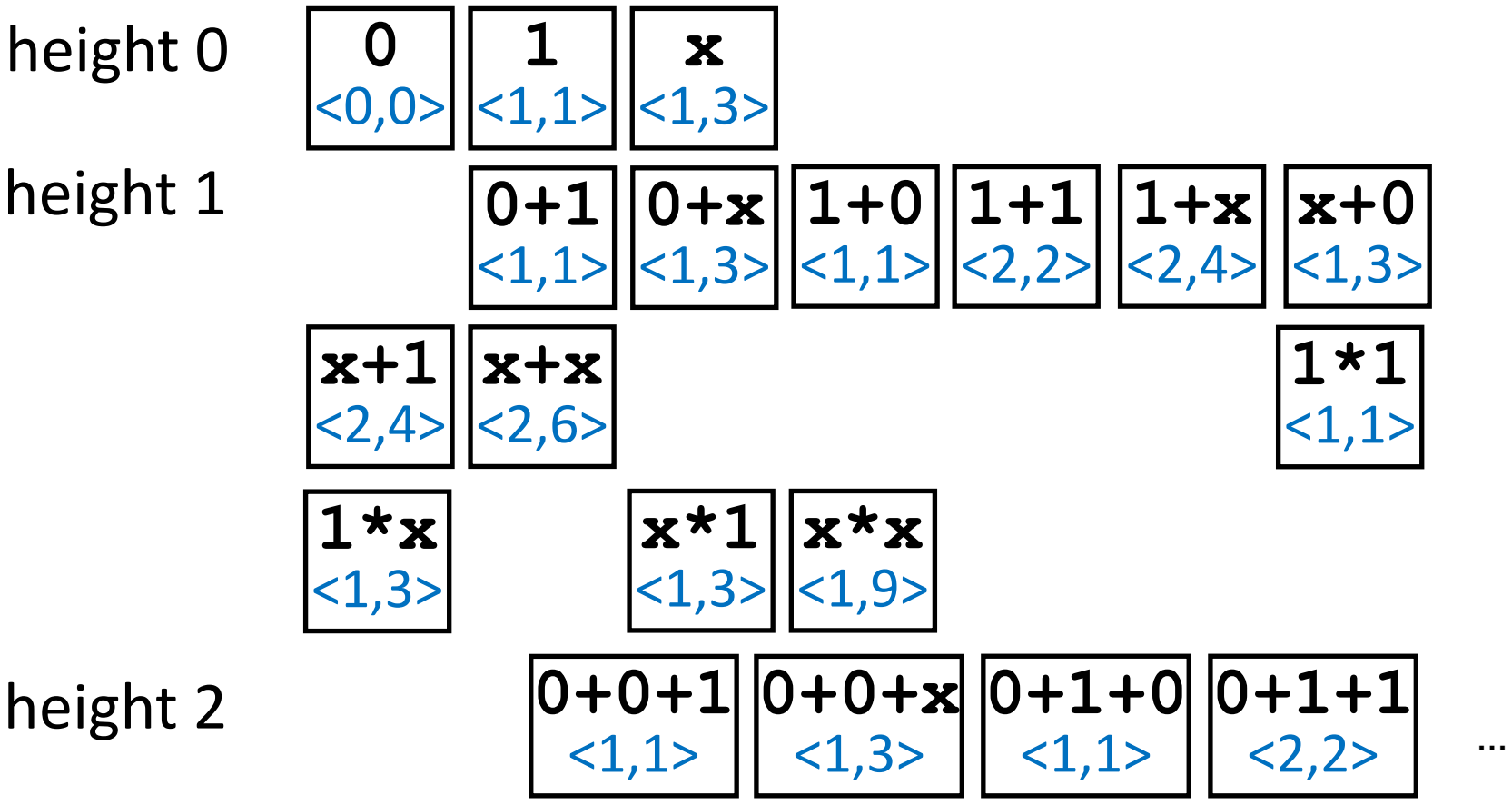
Searching the space by enumeration



$S \rightarrow \quad E$
$E \rightarrow \quad 0|1|x$

$i_0 = \{x \mapsto 1\}$
$i_1 = \{x \mapsto 3 \backslash\}$

Searching the space by enumeration



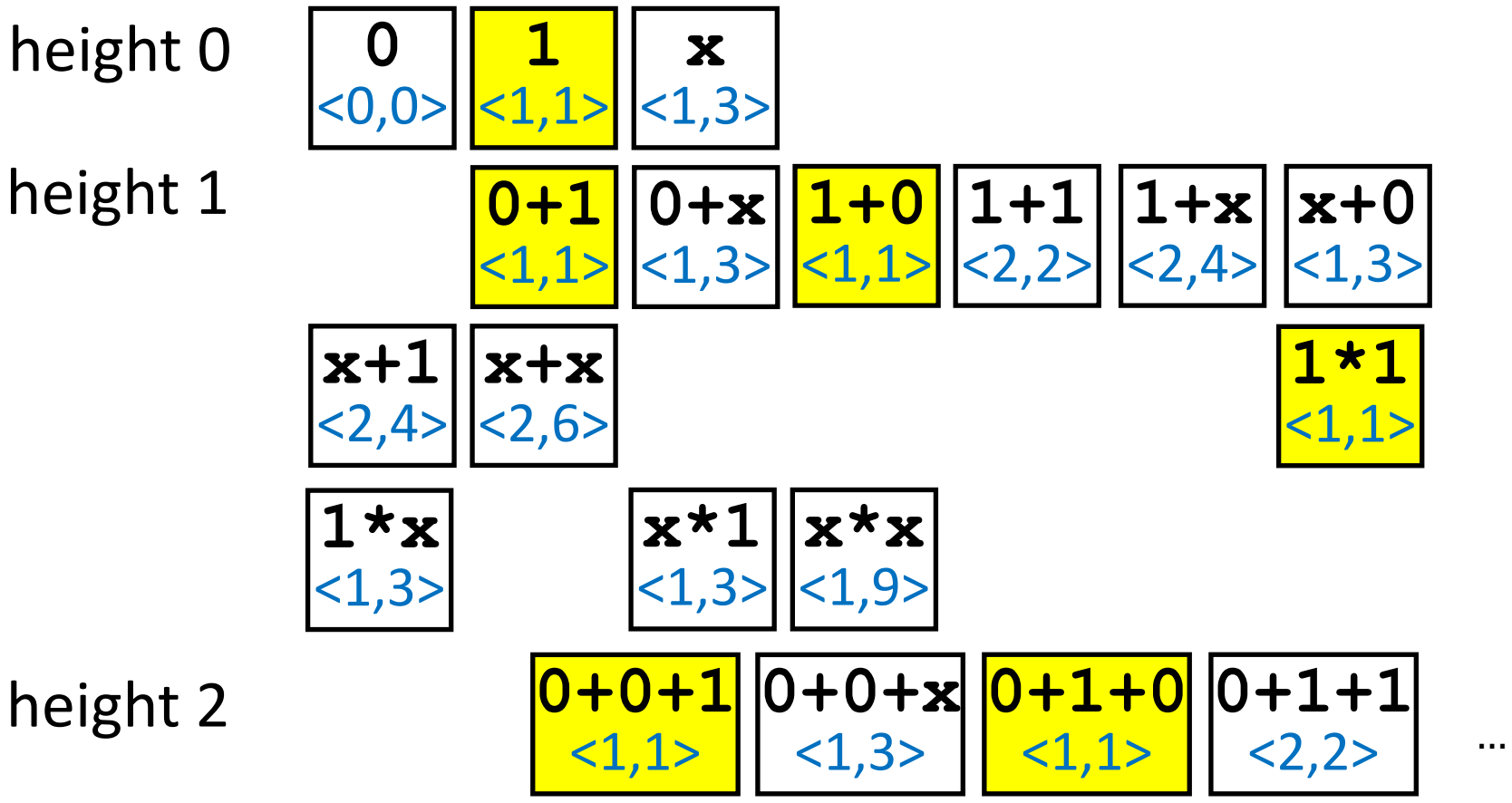
$S \rightarrow \quad E$

$E \rightarrow \quad 0|1|x$

$i_0 = \{x \mapsto 1\}$

$i_1 = \{x \mapsto 3 \backslash\}$

Searching the space by enumeration



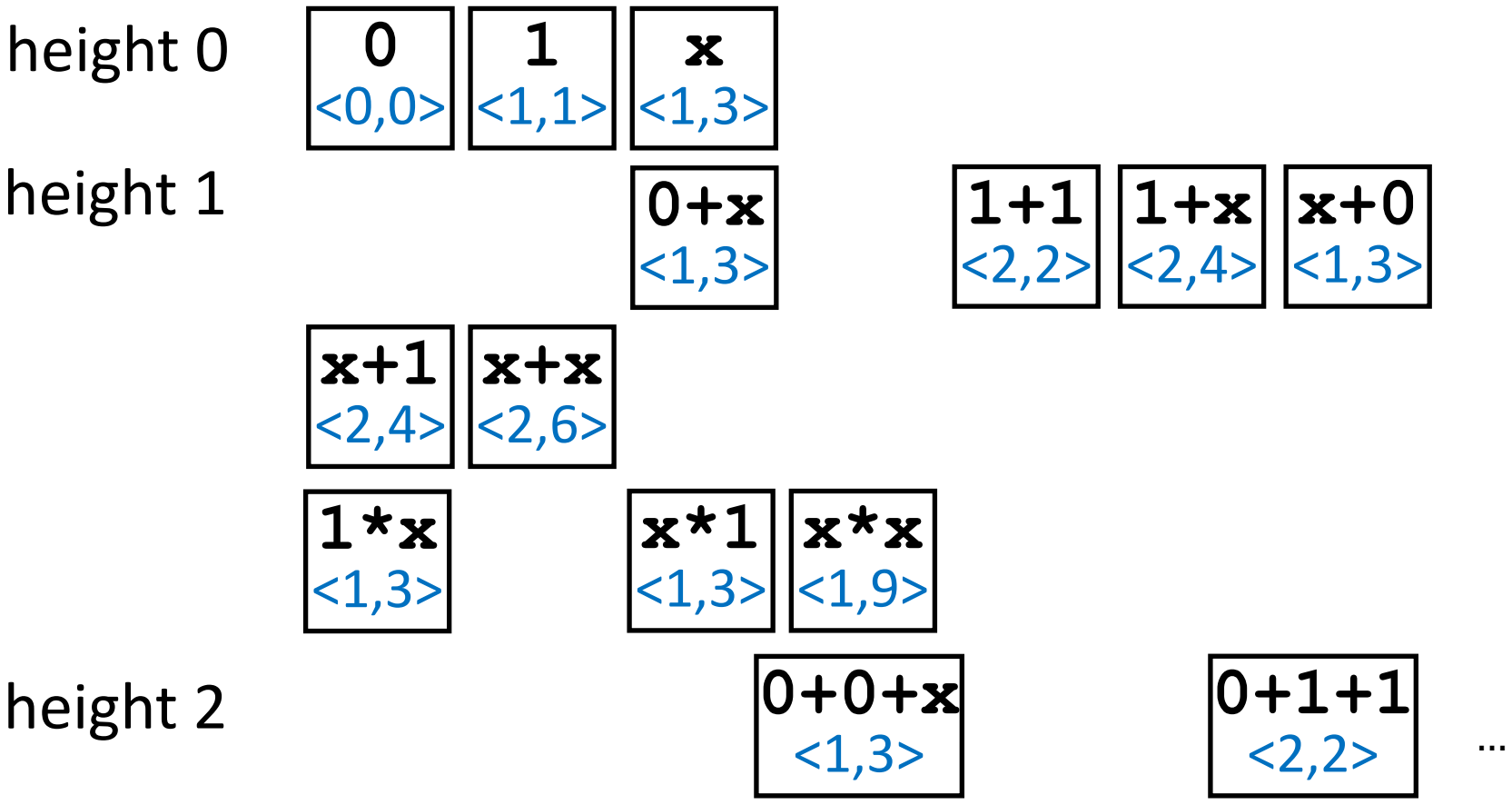
$S \rightarrow \quad E$

$E \rightarrow \quad 0|1|x$

$i_0 = \{x \mapsto 1\}$

$i_1 = \{x \mapsto 3 \backslash\}$

Searching the space by enumeration



$S \rightarrow \quad E$
$E \rightarrow \quad 0|1|x$

$i_0 = \{x \mapsto 1\}$
$i_1 = \{x \mapsto 3 \backslash\}$

Searching the space by enumeration

height 0

0 <0,0>	**1** <1,1>	**x** <1,3>

height 1

1+1 <2,2>	**1+x** <2,4>

x+x <2,6>

x*x <1,9>

height 2

$$S \rightarrow \quad E$$
$$E \rightarrow \quad 0|1|x$$

$$i_0 = \{x \mapsto 1\}$$
$$i_1 = \{x \mapsto 3 \backslash\}$$

…

Challenge 1 (ctd): why does synthesis break?

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

Retain(`input.length / 2`)

height 1 … **`input.length`** <3> …

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

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

Challenge 1 (ctd): why does 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> …

Challenge 1 (ctd): why does 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> …

Challenge 1 (ctd): why does 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> …

Solution 1: Generalize synthesis algorithm

Before (examples):

Observational equivalence:

Problem: only considers execution results for equivalence

After (general specification S):

Observational equivalence:

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

…for an example, it's still execution results

Challenge 1 (ctd): why does synthesis break?

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

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>

…

Byproduct 1: More specs supported now

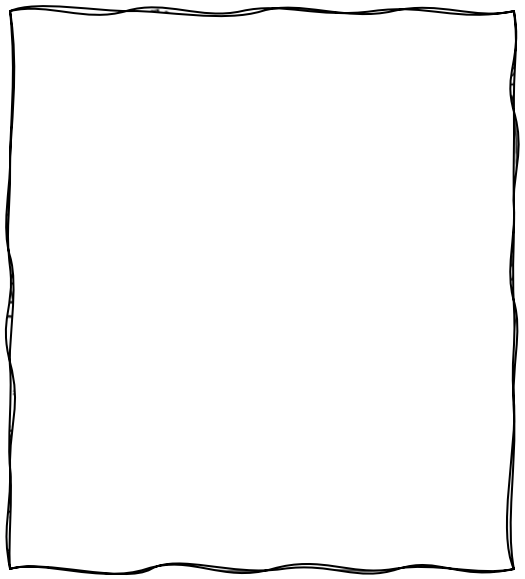
Synthesizer can support more things to specify.

We picked one that looks like it would work well in the interaction:

Type Constraints:

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

RESL challenges



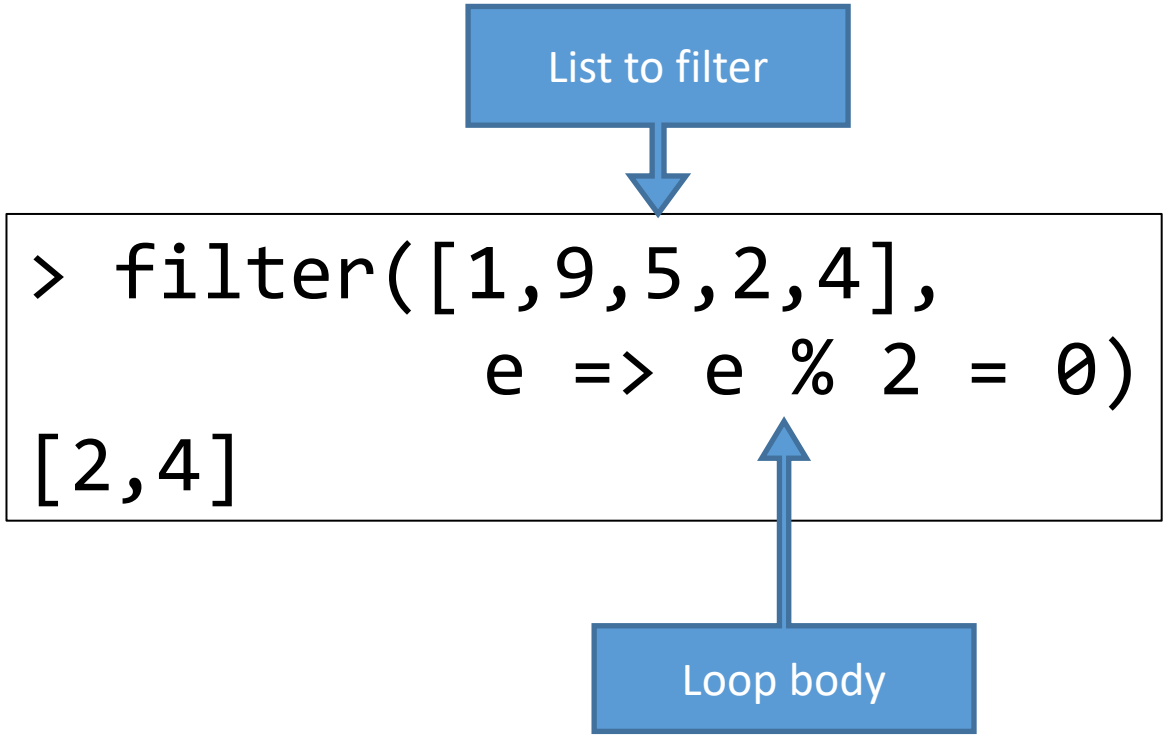
synthesizer

Challenge 1: Local specifications interfere with synthesis algorithm

Challenge 2: Loops are hard for the synthesizer

user interaction

Challenge 2: loops are hard for the synthesizer



Challenge 2: loops are hard for the synthesizer

$\{input \mapsto [1,9,5,2,4]\} \rightarrow [2,4]$

$i_0 = \{input \mapsto [1,9,5,2,4]\}$

height 3 … `input.filter(e => e % 2 = 0)` <[2,4]> …

Challenge 2: loops are hard for the synthesizer

$\{input \mapsto [1,9,5,2,4]\} \rightarrow [2,4]$

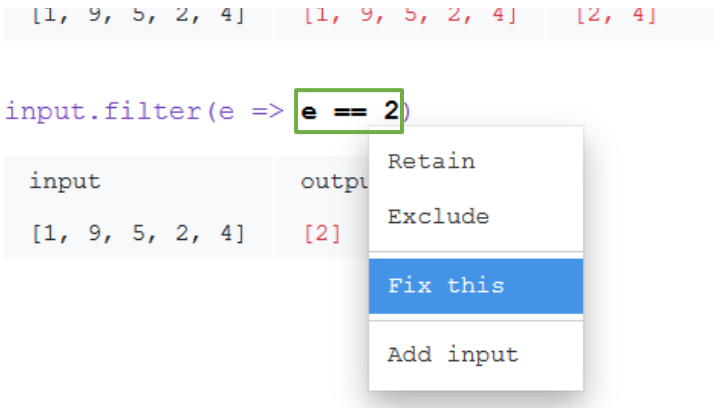
$i_0 = \{input \mapsto [1,9,5,2,4]\}$

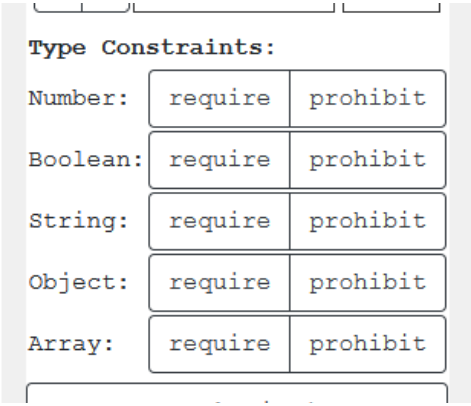
height 3 … `input.filter(e => e % 2 = 0)` … <[2,4]>

Solution 2: User-introduced loops

Synthesizer will no longer add loops: `map`, `filter`, etc.

User adds the loop manually, then asks the synthesizer to tackle its inside





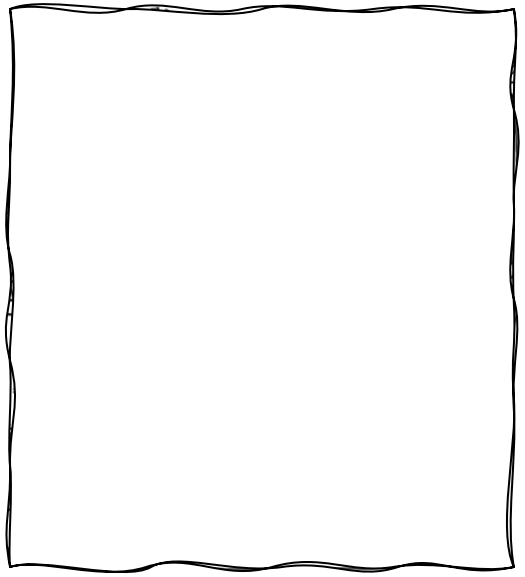
Solution 2: User-introduced loops

Program is `input.filter(e => ?)`

$i_0 = \{e \mapsto 1\}$
$i_1 = \{e \mapsto 9\}$
$i_2 = \{e \mapsto 5\}$
$i_3 = \{e \mapsto 2\}$
$i_4 = \{e \mapsto 4\}$

height 2 … `input.filter(e => e % 2 = 0)` …
<[false,false,false,true,true]>

RESL co-design result



synthesizer

add local specifications

more forms of spec available

require users to introduce loops

need to find inner inputs for loops

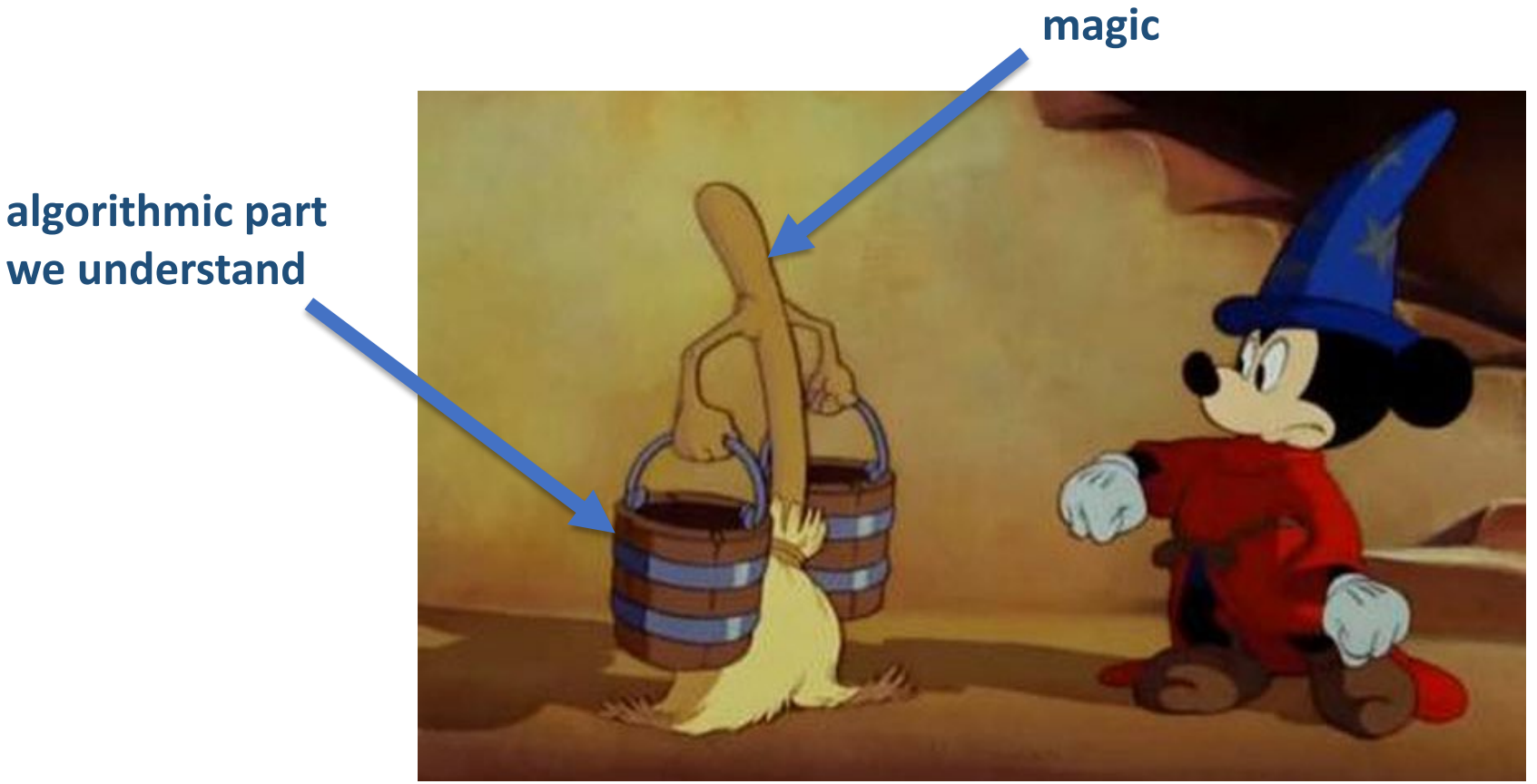
user interaction

This is a lot of work.

We know what's in it for the user (and maybe that's worth it)

But what's in it for us?

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

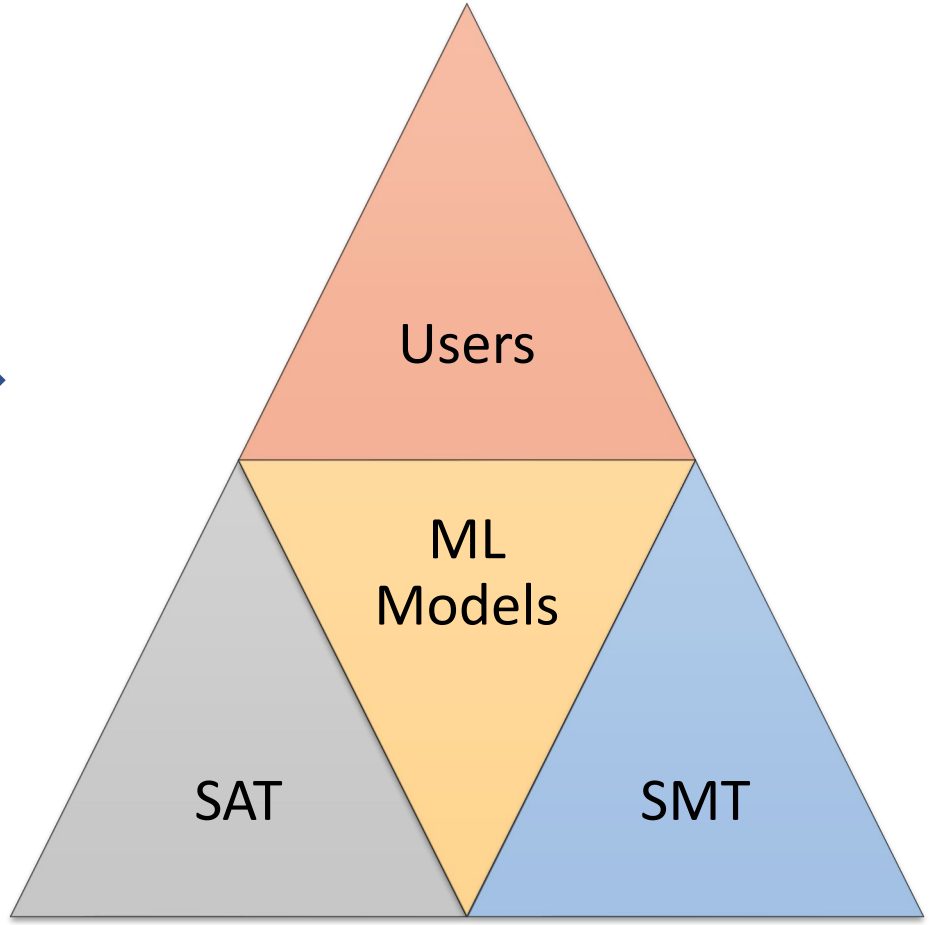
Hard subproblem

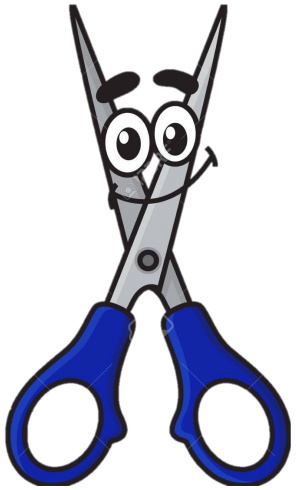
Model?

Available magics

Hard subproblem

Model?



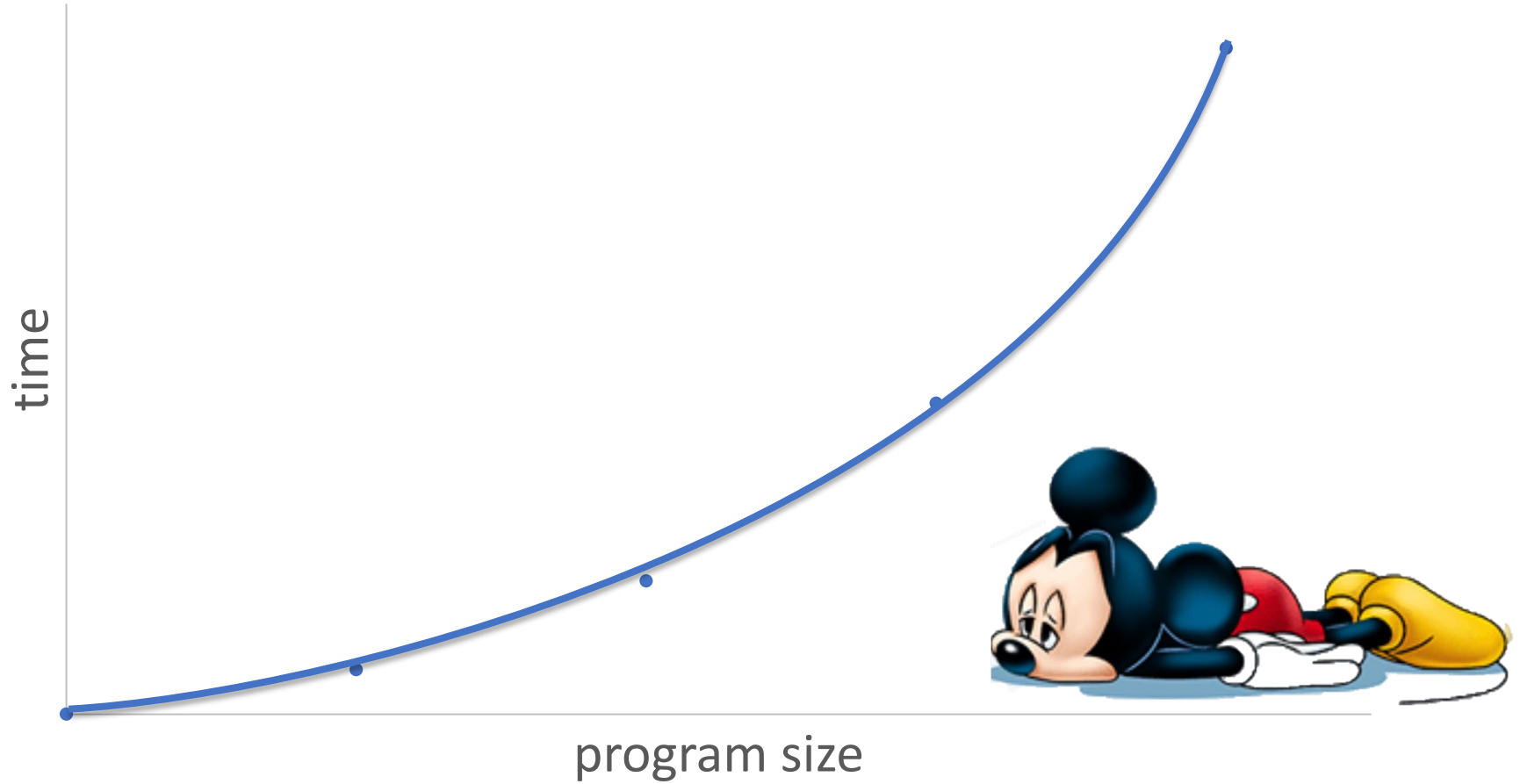


Example 2: Small-Step Synthesis

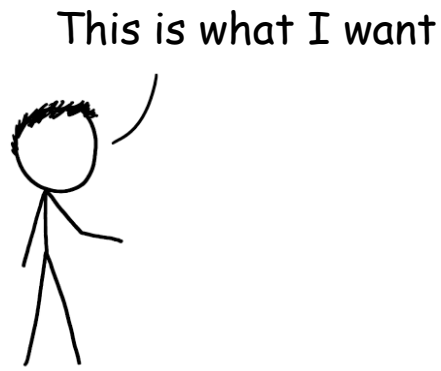
Small-Step Live Programming by Example [UIST'20]

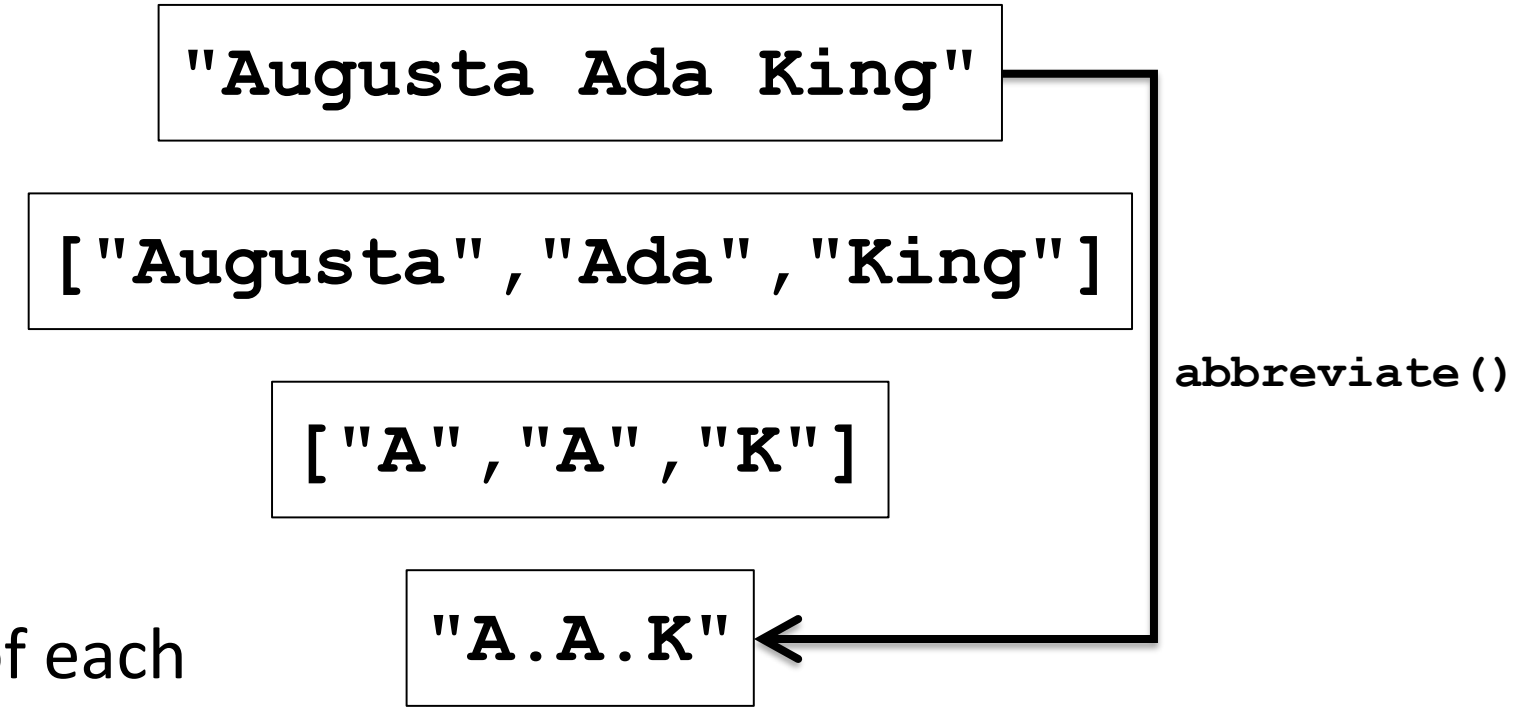
LooPy: Interactive Program Synthesis with Control Structures [OOPSLA'21]

Program synthesis is hard



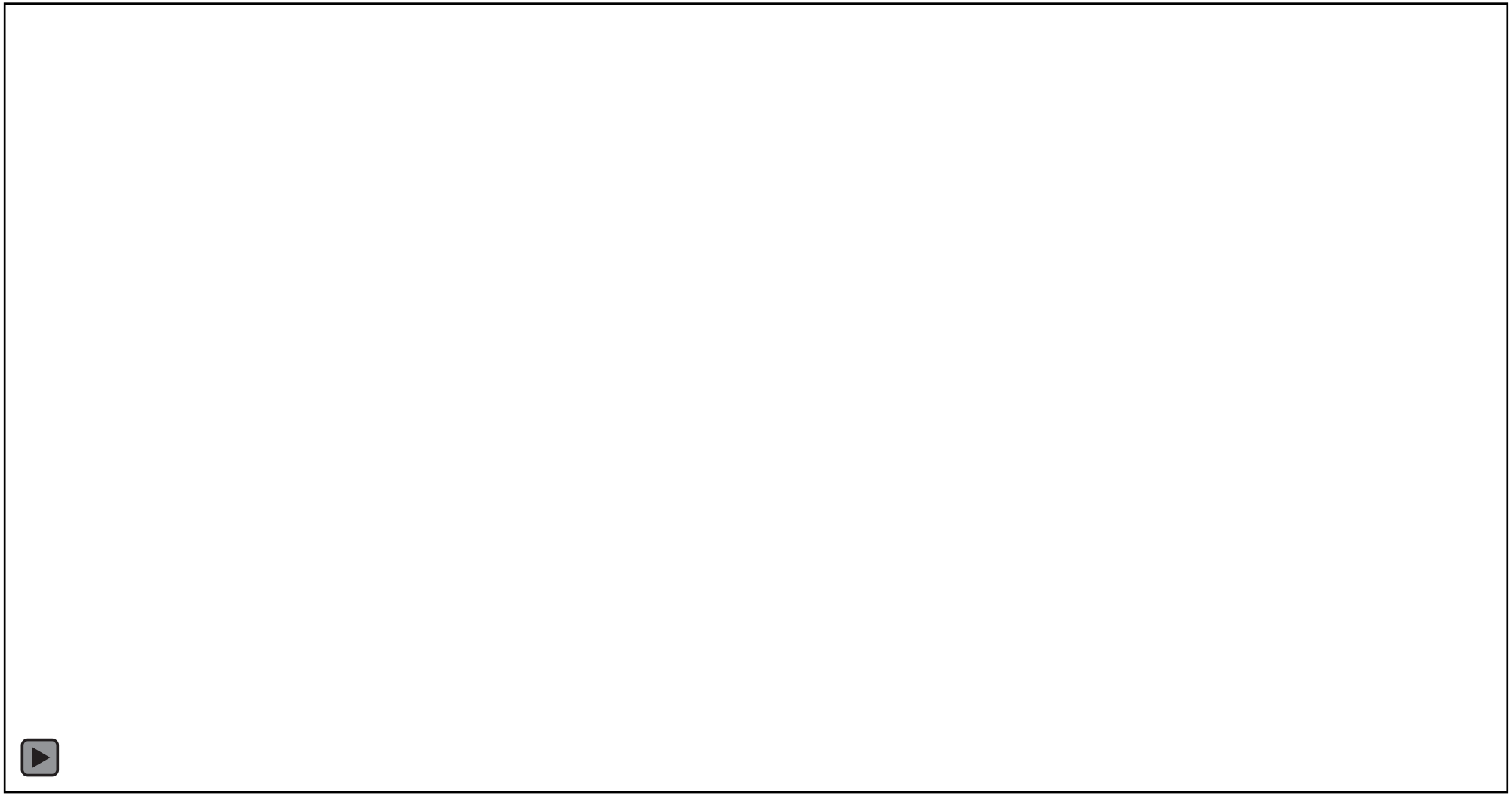
Problem: Abbreviate



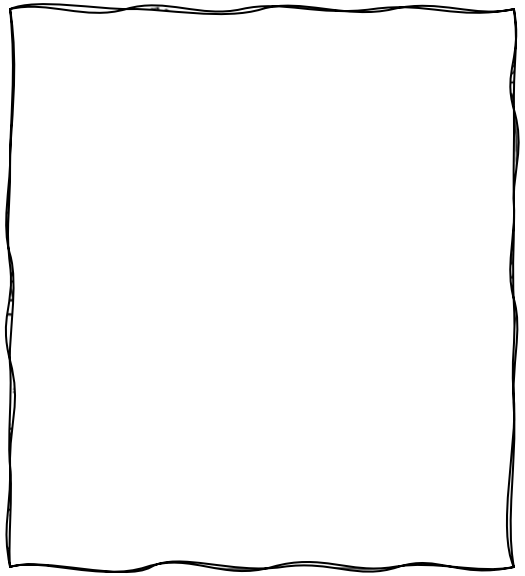


1. Split into words
2. Get the first letter of each
3. Put dots in between

SnipPy



SnipPy Co-Design



synthesizer

small-step specifications

interactive-time synthesis

reinforce already-correct outputs

result with fewer examples

force trace-complete examples

data-dependent loop synthesis

user interaction

With thanks to my co-authors


Roi Gabai


Shachar Itzhaky


Eran Yahav


Kasra Ferdowsifard


Shraddha Barke


Sorin Lerner


Nadia Polikarpova

Summary

1. For humans to use our tools, human interaction has to impact algorithm design
2. Interaction design can (secretly) offload part of the problem to the user
3. Program synthesis is cool