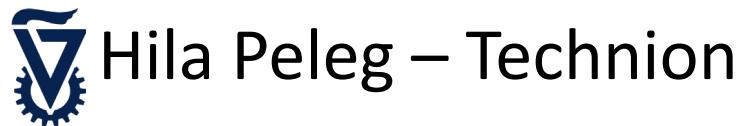


Program Synthesis Co-Design



Program Synthesis

This is what I want



⇒



⇒

```
1 $(function(){cards();});  
2 $(window).on('resize', function  
3     cards(){  
4         var width = $(window).width();  
5         if(width < 750){  
6             cardssmallscreen();  
7         } else {  
8             cardsbigscreen();  
9         }  
10    }  
11    function cardssmallscreen(){  
12        var cards = $('.card');  
13        var height = 0;  
14        var card2 = 2;  
15        var card1 = 1;  
16        var of = 1;  
17        var i = 1;  
18        var j = 1;  
19        var k = 1;  
20        var l = 1;  
21        var m = 1;  
22        var n = 1;  
23        var o = 1;  
24        var p = 1;  
25        var q = 1;  
26        var r = 1;  
27        var s = 1;  
28        var t = 1;  
29        var u = 1;  
30        var v = 1;  
31        var w = 1;  
32        var x = 1;  
33        var y = 1;  
34        var z = 1;  
35        var a = 1;  
36        var b = 1;  
37        var c = 1;  
38        var d = 1;  
39        var e = 1;  
40        var f = 1;  
41        var g = 1;  
42        var h = 1;  
43        var i = 1;  
44        var j = 1;  
45        var k = 1;  
46        var l = 1;  
47        var m = 1;  
48        var n = 1;  
49        var o = 1;  
50        var p = 1;  
51        var q = 1;  
52        var r = 1;
```

I'll tell you what I want

Int -> [Maybe a] -> a



⇒



⇒

```
1   $(function(){cards();});  
2   $(window).on('resize', function  
3     cards(){  
4       var width = $(window).width();  
5       if(width < 750){  
6         cardssmallscreen();  
7       }else{  
8         cardsbigscreen();  
9       }  
10      }  
11      function cardssmallscreen()  
12      {  
13        var cards = $('.card');  
14        var height = 0;  
15        var card2 = 2;  
16        var i = 1;  
17        cards.each(function(index, value){  
18          if(index < card2){  
19            cards.eq(index).css({  
20              height: height + 100  
21            });  
22          }  
23        });  
24      }  
25      function cardsbigscreen()  
26      {  
27        var cards = $('.card');  
28        var height = 0;  
29        var card2 = 2;  
30        var i = 1;  
31        cards.each(function(index, value){  
32          if(index < card2){  
33            cards.eq(index).css({  
34              height: height + 100  
35            });  
36          }  
37        });  
38      }  
39    }  
40  }  
41  cards();  
42  window.cards();  
43  cards();  
44  cards();  
45  cards();  
46  cards();  
47  cards();  
48  cards();  
49  cards();  
50  cards();  
51  cards();  
52  cards();
```

I'll tell you what I want

Looks like $x.f(a,b,\textcolor{orange}{??})$



⇒



⇒

```
$(function(){cards();});  
$(window).on('resize', function  
{  
    var width = $(window).width();  
    if(width < 750){  
        cardssmallscreen();  
    }else{  
        cardsbigscreen();  
    }  
});  
function cardssmallscreen()  
{  
    var cards = $('.card');  
    var height = 0;  
    var card2 = 2;  
    var i = 1;  
    cards.each(function(index){  
        if(index == 0){  
            $(this).css({  
                'height': height + 'px'  
            });  
        }else{  
            $(this).css({  
                'height': height + 'px'  
            });  
            card2++;  
        }  
    });  
}  
function cardsbigscreen()  
{  
    var cards = $('.card');  
    var height = 0;  
    var card2 = 2;  
    var i = 1;  
    cards.each(function(index){  
        if(index == 0){  
            $(this).css({  
                'height': height + 'px'  
            });  
        }else{  
            $(this).css({  
                'height': height + 'px'  
            });  
            card2++;  
        }  
    });  
}
```

I'll tell you what I want

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


\Rightarrow



\Rightarrow

```
1      $(function(){cards();});  
2      $(window).on('resize', function  
3          cards(){  
4              var width = $(window).  
5                  if(width < 750){  
6                      cardssmallscreen()  
7                  }else{  
8                      cardsbigscreen();  
9                  }  
10             }  
11         }  
12         function cardssmallscreen()  
13             var cards = $('.card');  
14             var height = 0;  
15             var card2 = 2;  
16             var card1 = 1;  
17             cards.height(height);  
18             height += 300;
```

A snippet of JavaScript code. Lines are numbered from 1 to 52. The code uses jQuery to handle window resize events, calling a function named 'cards'. It checks the window width and calls either 'cardssmallscreen' or 'cardsbigscreen' functions based on whether the width is less than 750 pixels. It also handles card heights and indices.

I'll tell you what I want



⇒



⇒

```
1 $(function(){cards();});  
2 $(window).on('resize', function  
3     cards(){  
4         var width = $(window).width();  
5         if(width < 750){  
6             cardssmallscreen();  
7         } else{  
8             cardsbigscreen();  
9         }  
10    }  
11    function cardssmallscreen(){  
12        var cards = $('.card');  
13        var height = 0;  
14        var card2 = 2;  
15        var i = 1;  
16        cards.each(function(index){  
17            if(index == 0){  
18                height = 100;  
19            } else{  
20                height = 100 * (index + 1);  
21            }  
22            $(this).css('height', height);  
23        })  
24    }  
25    function cardsbigscreen(){  
26        var cards = $('.card');  
27        var height = 100;  
28        cards.css('height', height);  
29    }  
30    cards();  
31    window.cards = cards;  
32    window.height = height;
```

Program Synthesis

 φ  \Rightarrow

```
41
42 $(function(){cards();});
43 $(window).on('resize', function()
44   var width = $(window).width();
45   if(width < 750){
46     cardssmallscreen();
47   }else{
48     cardsbigscreen();
49   }
50   }
51   function cardssmallscreen(){
52     var cards = $('.card');
53     var height = 0;
54     var card2 = 2;
55     card1 = 1;
56     cards.css('height', height + 'px');
```

The search is highly customized



Search by example

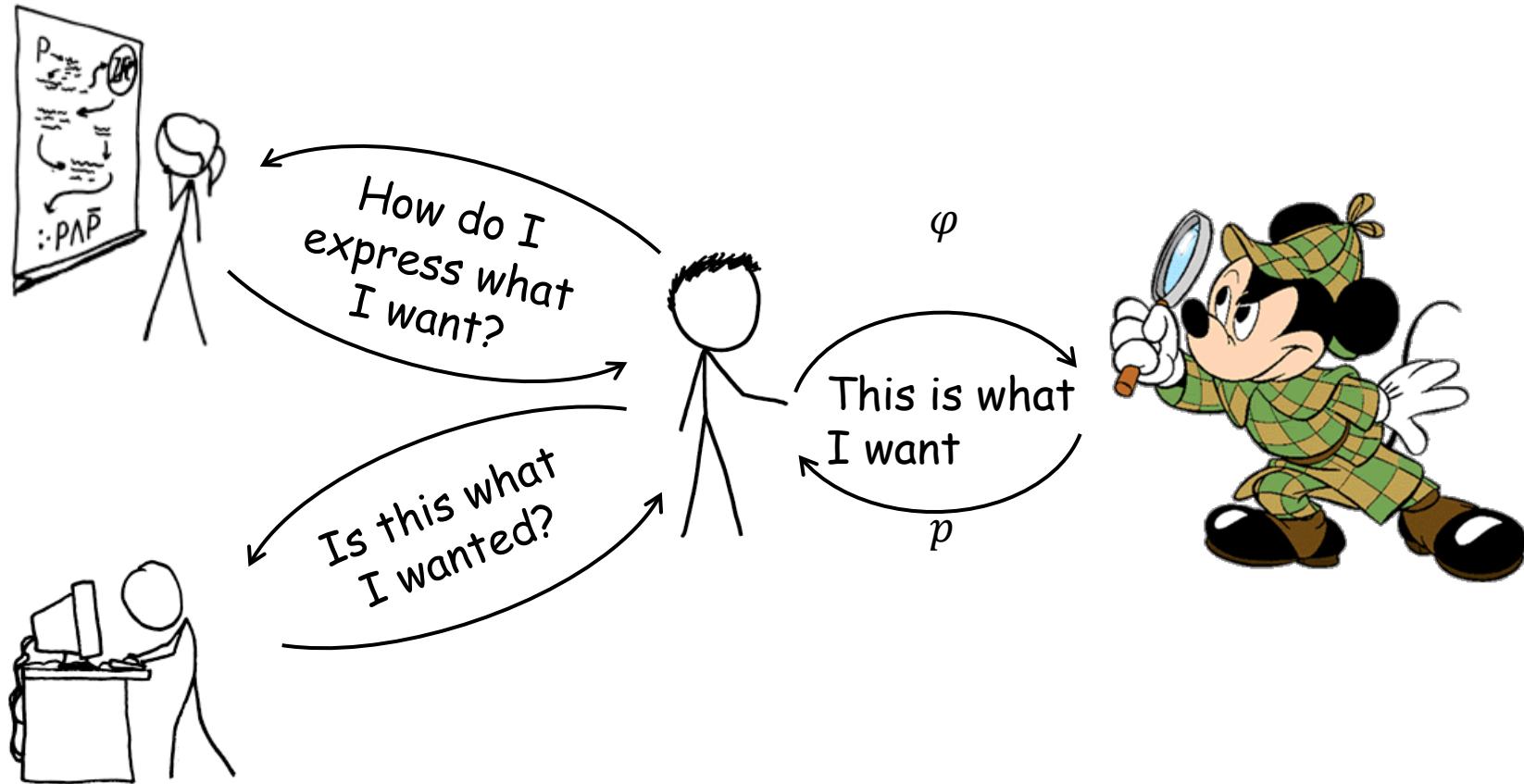


Search by type

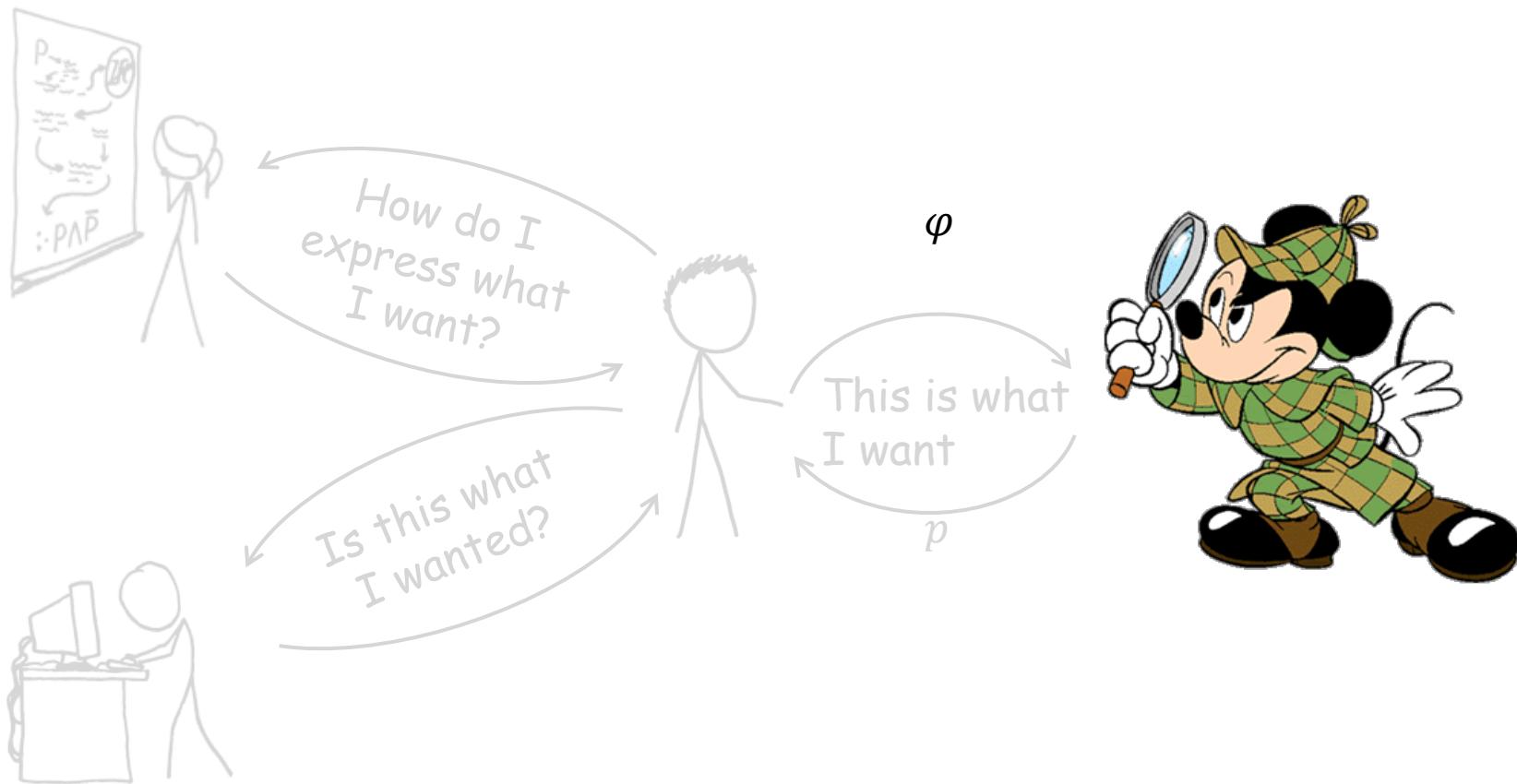


Search by natural language

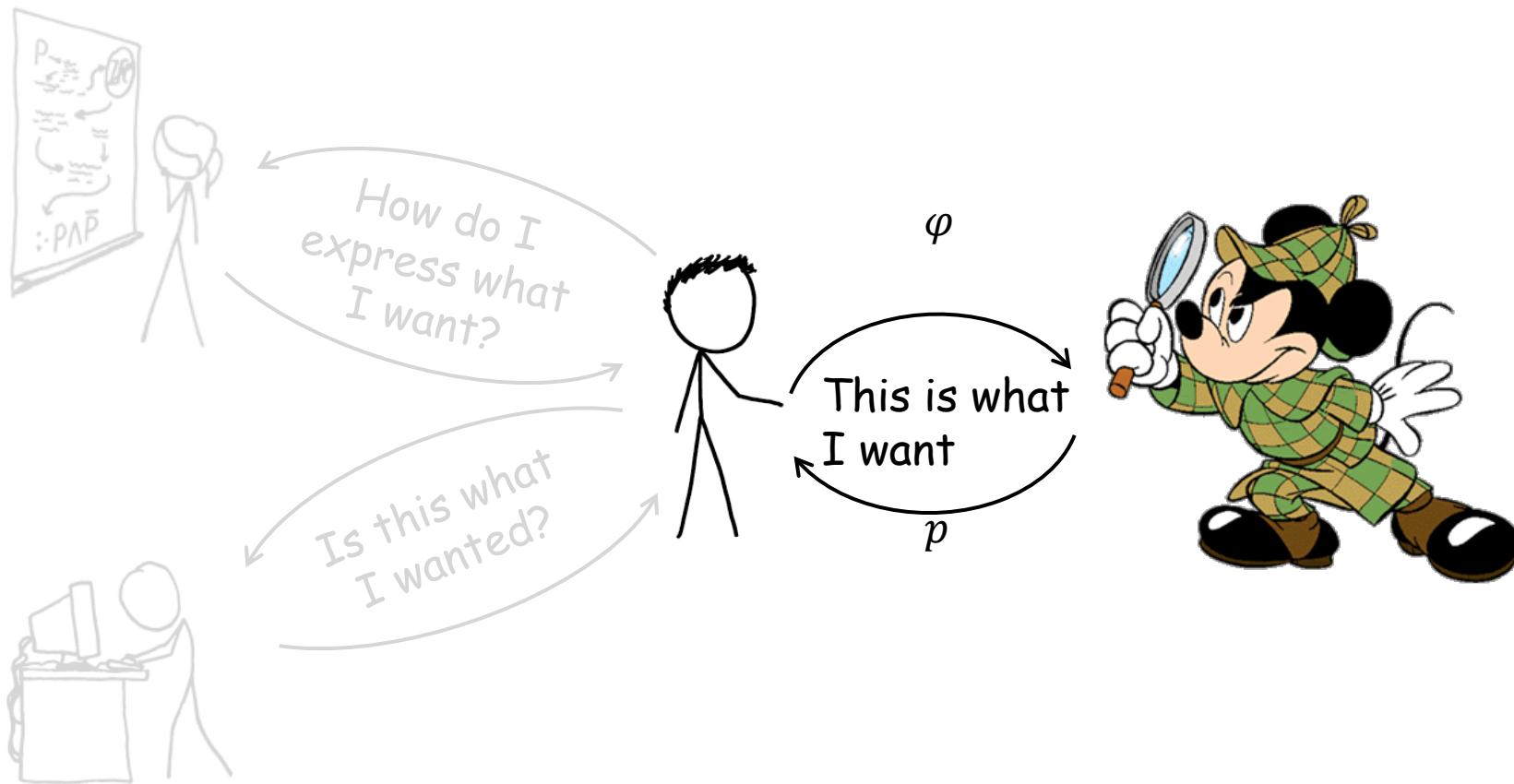
Interaction models



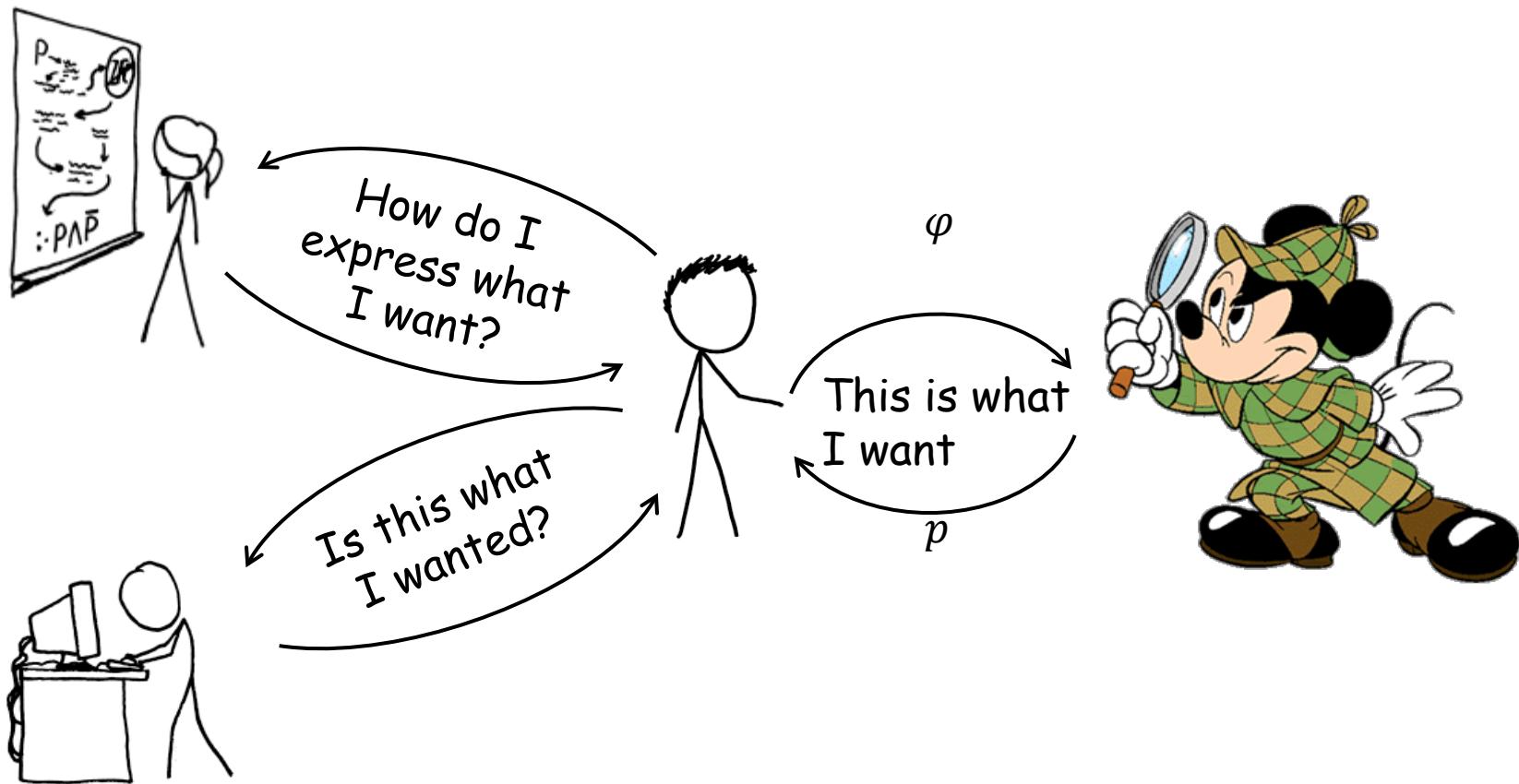
Interaction models are part of the package



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
```



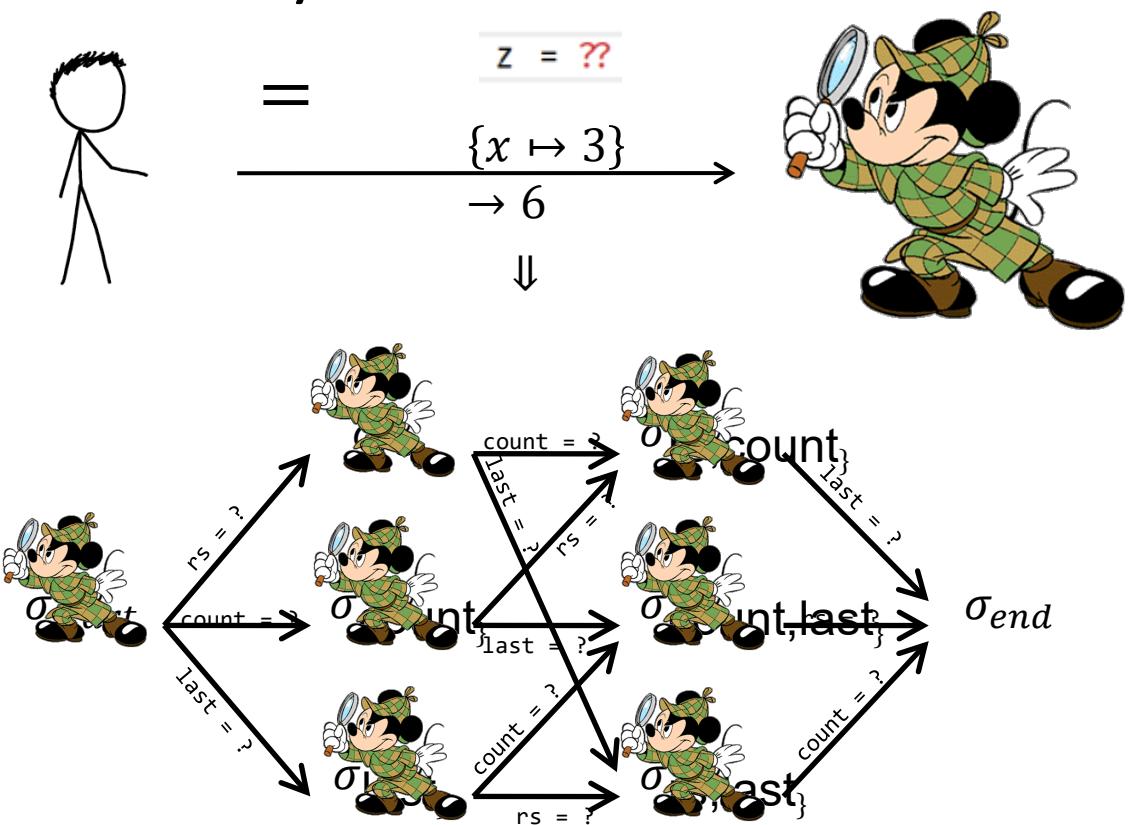
```
6     if last == c:
7         count += 1
8     else:
9         last, count, rs = ???
10    return rs
```

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
13
14      last = c:
15          count += 1
16      else:
17          last, count, rs = ???
18
19      return rs
```

Now to synthesize it:



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

$$\begin{aligned}\{arr \mapsto [], s \mapsto 0\} &\rightarrow 0 \\ \{arr \mapsto [1], s \mapsto 0\} &\rightarrow 1 \\ \{arr \mapsto [1,2], s \mapsto 1\} &\rightarrow 3\end{aligned}$$

Example II: synthesizer constraints

Independent

arr2 = [??]

?



ete

$$\begin{aligned}\{arr \mapsto [1], s \mapsto 0\} &\rightarrow 1 \\ \{arr \mapsto [1,2], s \mapsto 1\} &\rightarrow 3\end{aligned}$$

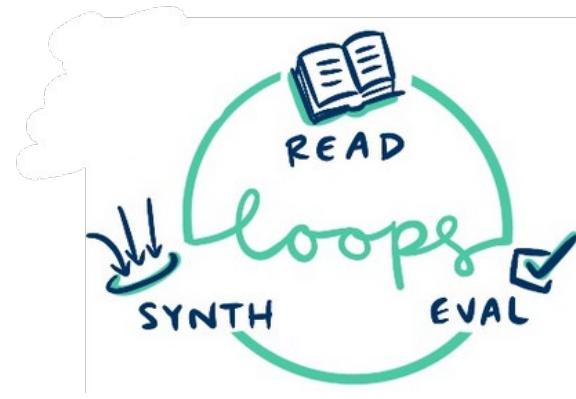
Synthesis Co-Design



synthesizer



user interaction

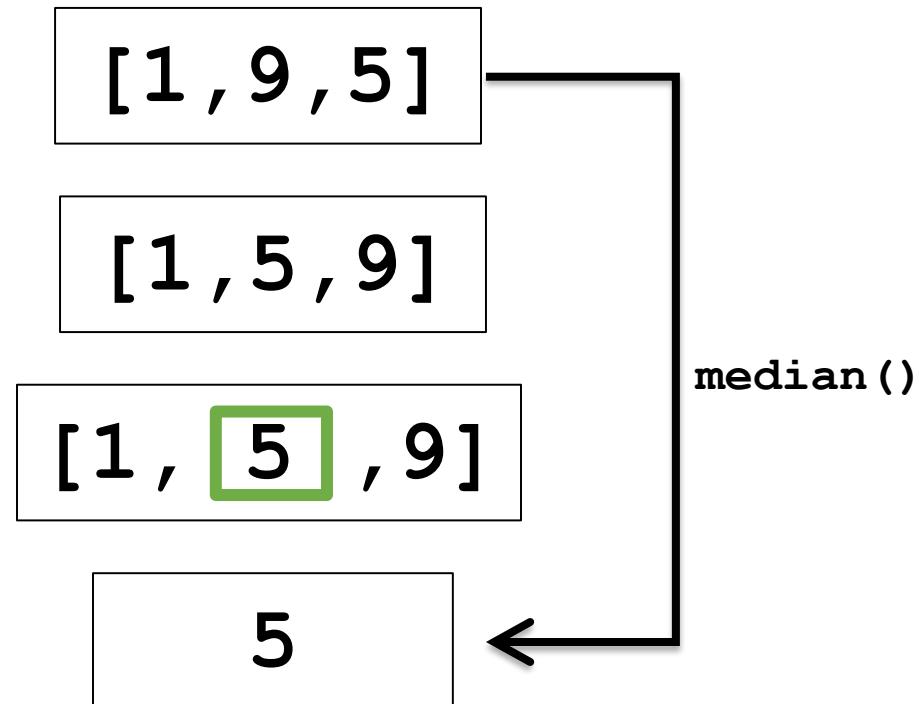


Example 1: Read-Eval-Synth Loop

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)]
```

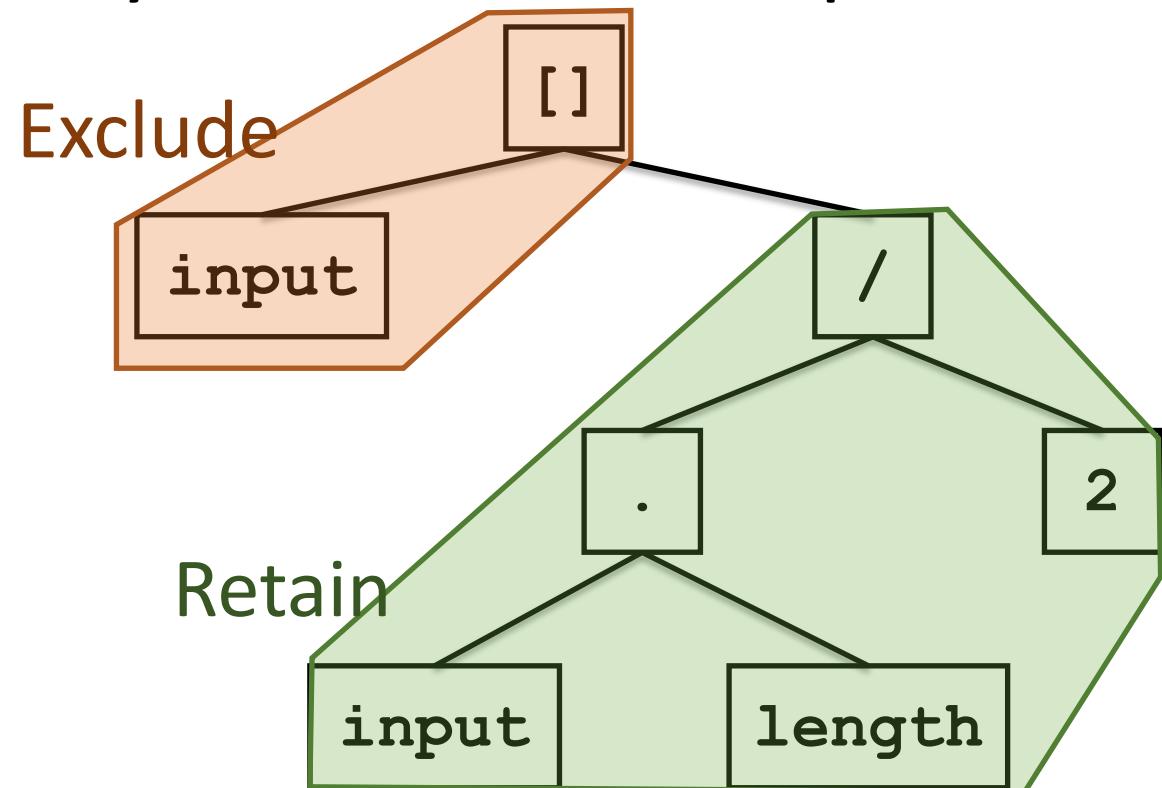
RESL: REPLs meet local specifications

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
>
```

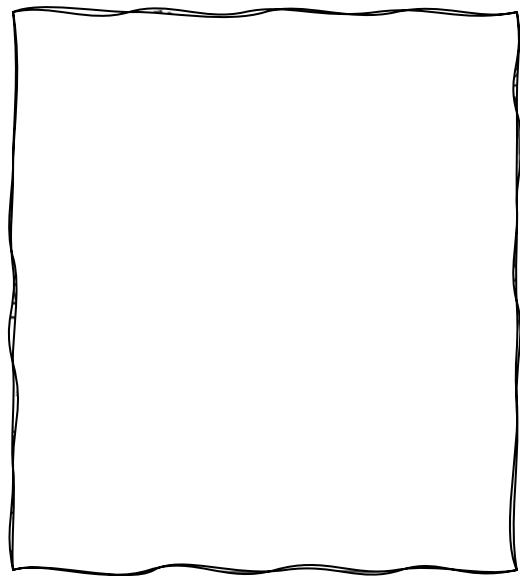
Syntactic code-review-like specs



Read-Eval-Synth Loop (RESL)



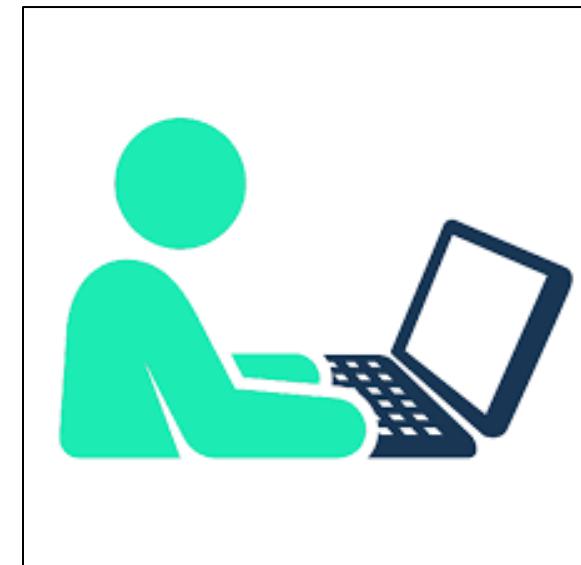
RESL challenges



synthesizer

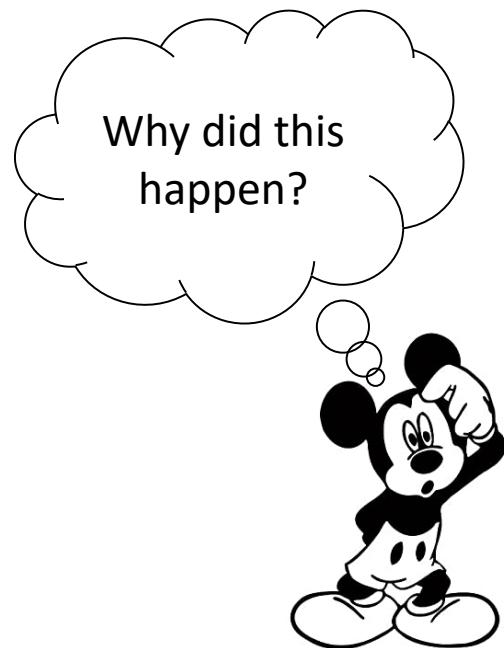
Challenge 1: Local specifications
interfere with synthesis algorithm

Challenge 2: Loops are hard
for the synthesizer

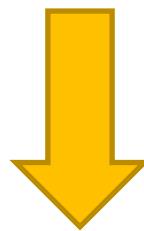


user interaction

Challenge 1: Synthesis breaks



```
{input ↦ [1,9,5]} → 5  
Retain(input.length / 2)
```



```
No program
```

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

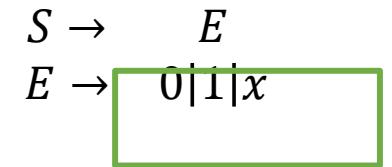
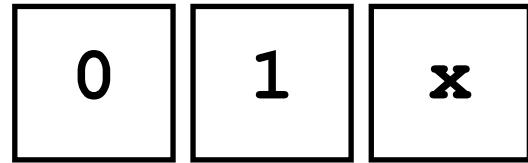
Searching the space by enumeration

height 0

$$\begin{array}{rcl} S & \rightarrow & E \\ E & \rightarrow & 0|1|x \end{array}$$

Searching the space by enumeration

height 0



Searching the space by enumeration

height 0	0	1	x		$S \rightarrow E$	$E \rightarrow 0 1 x$		
height 1	0+0	0+1	0+x	1+0	1+1	1+x	x+0	
	x+1	x+x						

Searching the space by enumeration

height 0

0	1	x
---	---	---

$$\begin{array}{l} S \rightarrow E \\ E \rightarrow 0|1|x \end{array}$$

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			



Searching the space by enumeration

height 0	0	1	x		$S \rightarrow E$	$E \rightarrow 0 1 x$		
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		...	

Searching the space by enumeration

height 0	0 $\langle 0,0 \rangle$	1 $\langle 1,1 \rangle$	x $\langle 1,3 \rangle$				
height 1	$0+0$ $\langle 0,0 \rangle$	$0+1$ $\langle 1,1 \rangle$	$0+x$ $\langle 1,3 \rangle$	$1+0$ $\langle 1,1 \rangle$	$1+1$ $\langle 2,2 \rangle$	$1+x$ $\langle 2,4 \rangle$	$x+0$ $\langle 1,3 \rangle$
	$x+1$ $\langle 2,4 \rangle$	$x+x$ $\langle 2,6 \rangle$	$0*0$ $\langle 0,0 \rangle$	$0*1$ $\langle 0,0 \rangle$	$0*x$ $\langle 0,0 \rangle$	$1*0$ $\langle 0,0 \rangle$	$1*1$ $\langle 1,1 \rangle$
	$1*x$ $\langle 1,3 \rangle$	$x*0$ $\langle 0,0 \rangle$	$x*1$ $\langle 1,3 \rangle$	$x*x$ $\langle 1,9 \rangle$			
height 2	$0+0+0$ $\langle 0,0 \rangle$	$0+0+1$ $\langle 1,1 \rangle$	$0+0+x$ $\langle 1,3 \rangle$	$0+1+0$ $\langle 1,1 \rangle$	$0+1+1$ $\langle 2,2 \rangle$...

$$\begin{array}{l} S \rightarrow E \\ E \rightarrow 0|1|x \end{array}$$

$$\begin{array}{l} i_0 = \{x \mapsto 1\} \\ i_1 = \{x \mapsto 3\} \end{array}$$

Searching the space by enumeration

height 0	0 $<0,0>$	1 $<1,1>$	x $<1,3>$				
height 1	0+0 $<0,0>$	0+1 $<1,1>$	0+x $<1,3>$	1+0 $<1,1>$	1+1 $<2,2>$	1+x $<2,4>$	x+0 $<1,3>$
	x+1 $<2,4>$	x+x $<2,6>$	0*0 $<0,0>$	0*1 $<0,0>$	0*x $<0,0>$	1*0 $<0,0>$	1*1 $<1,1>$
	1*x $<1,3>$	x*0 $<0,0>$	x*1 $<1,3>$	x*x $<1,9>$			
height 2	0+0+0 $<0,0>$	0+0+1 $<1,1>$	0+0+x $<1,3>$	0+1+0 $<1,1>$	0+1+1 $<2,2>$...

$$\begin{array}{l} S \rightarrow E \\ E \rightarrow 0|1|x \end{array}$$

$$\begin{array}{l} i_0 = \{x \mapsto 1\} \\ i_1 = \{x \mapsto 3\} \end{array}$$

Searching the space by enumeration

height 0

0	1	x
$<0,0>$	$<1,1>$	$<1,3>$

$$\begin{matrix} S \rightarrow & E \\ E \rightarrow & 0|1|x \end{matrix}$$

height 1

0+1	0+x	1+0	1+1	1+x	x+0
$<1,1>$	$<1,3>$	$<1,1>$	$<2,2>$	$<2,4>$	$<1,3>$
x+1	x+x				1*x
$<2,4>$	$<2,6>$				$<1,1>$
1*x		x*x			
$<1,3>$		$<1,3>$	$<1,9>$		

$$\begin{matrix} i_0 = \{x \mapsto 1\} \\ i_1 = \{x \mapsto 3\} \end{matrix}$$

height 2

0+0+1	0+0+x	0+1+0	0+1+1
$<1,1>$	$<1,3>$	$<1,1>$	$<2,2>$

...

Searching the space by enumeration

height 0

0 $<0,0>$	1 $<1,1>$	x $<1,3>$
--------------	--------------	----------------

$$\begin{matrix} S \rightarrow & E \\ E \rightarrow & 0|1|x \end{matrix}$$

height 1

$0+1$ $<1,1>$	$0+x$ $<1,3>$	$1+0$ $<1,1>$	$1+1$ $<2,2>$	$1+x$ $<2,4>$	$x+0$ $<1,3>$
------------------	------------------	------------------	------------------	------------------	------------------

$x+1$ $<2,4>$	$x+x$ $<2,6>$	$1*x$ $<1,3>$	$x*1$ $<1,3>$	$x*x$ $<1,9>$	$1*x$ $<1,1>$
------------------	------------------	------------------	------------------	------------------	------------------

height 2

$0+0+1$ $<1,1>$	$0+0+x$ $<1,3>$	$0+1+0$ $<1,1>$	$0+1+1$ $<2,2>$
--------------------	--------------------	--------------------	--------------------

$$\begin{matrix} i_0 = \{x \mapsto 1\} \\ i_1 = \{x \mapsto 3\} \end{matrix}$$

...

Searching the space by enumeration

height 0

0 $\langle 0,0 \rangle$	1 $\langle 1,1 \rangle$	x $\langle 1,3 \rangle$
----------------------------	----------------------------	----------------------------

$$\begin{matrix} S \rightarrow & E \\ E \rightarrow & 0|1|x \end{matrix}$$

height 1

$0+x$ $\langle 1,3 \rangle$	$1+1$ $\langle 2,2 \rangle$	$1+x$ $\langle 2,4 \rangle$	$x+0$ $\langle 1,3 \rangle$
--------------------------------	--------------------------------	--------------------------------	--------------------------------

$x+1$ $\langle 2,4 \rangle$	$x+x$ $\langle 2,6 \rangle$
--------------------------------	--------------------------------

$1*x$ $\langle 1,3 \rangle$	$x*1$ $\langle 1,3 \rangle$	$x*x$ $\langle 1,9 \rangle$
--------------------------------	--------------------------------	--------------------------------

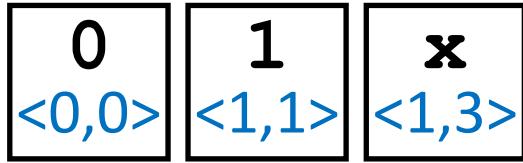
height 2

$0+0+x$ $\langle 1,3 \rangle$	$0+1+1$ $\langle 2,2 \rangle$...
----------------------------------	----------------------------------	-----

$$\begin{matrix} i_0 = \{x \mapsto 1\} \\ i_1 = \{x \mapsto 3\} \end{matrix}$$

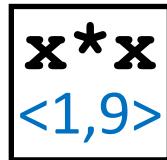
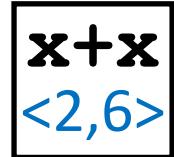
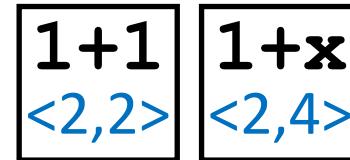
Searching the space by enumeration

height 0



$$\begin{array}{l} S \rightarrow E \\ E \rightarrow 0|1|x \end{array}$$

height 1



height 2

$$\begin{array}{l} i_0 = \{x \mapsto 1\} \\ i_1 = \{x \mapsto 3\} \end{array}$$

...

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

<3>

<3>

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

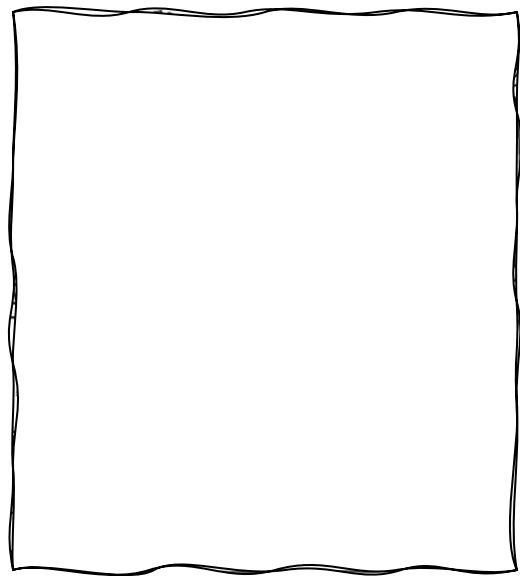
Synthesizer can support more things to specify.

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

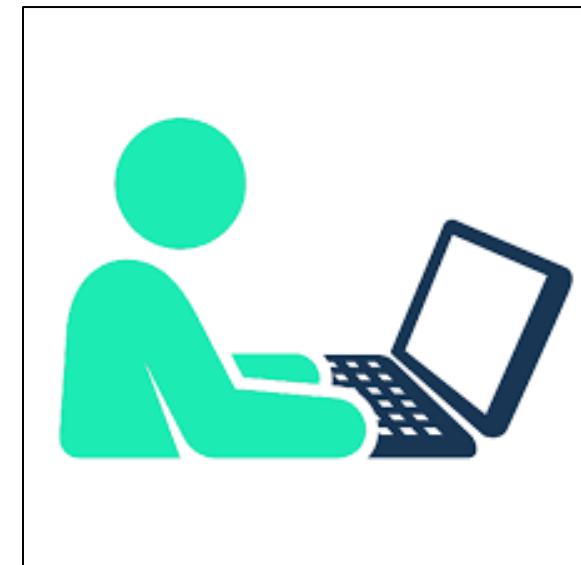
RESL challenges



synthesizer

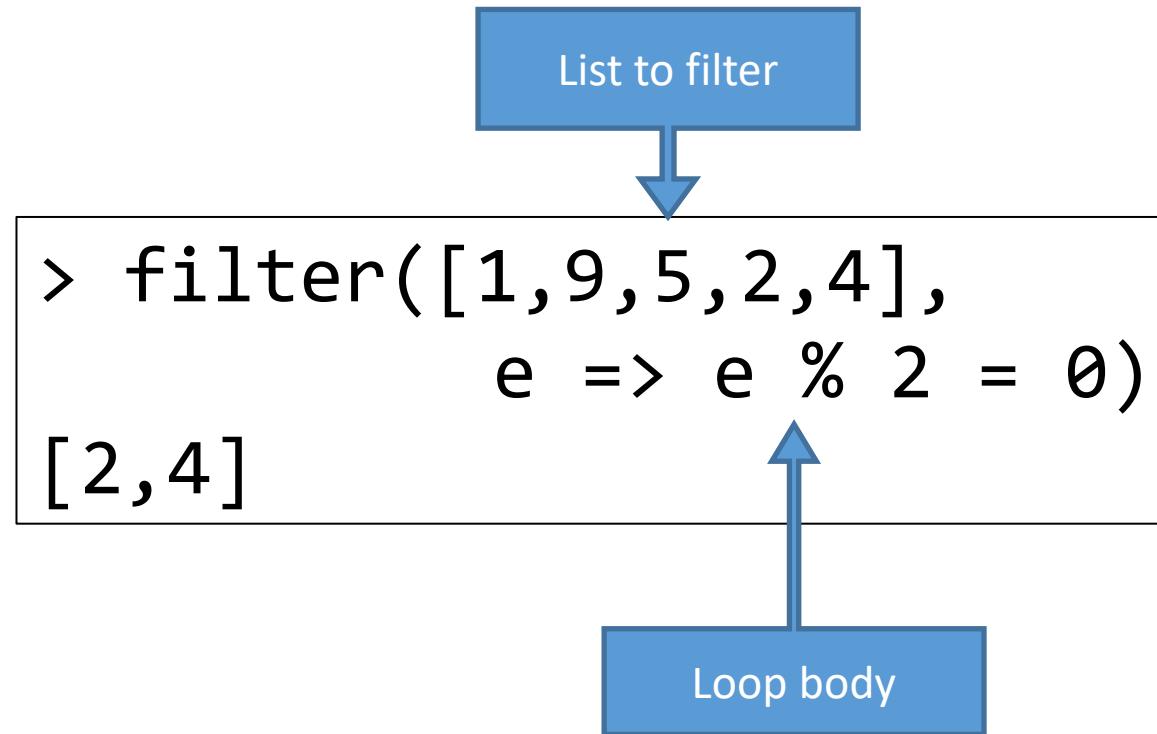
Challenge 1: Local specifications
interfere with synthesis algorithm

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 ↦ [1,9,5,2,4]} → [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 ↦ [1,9,5,2,4]} → [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

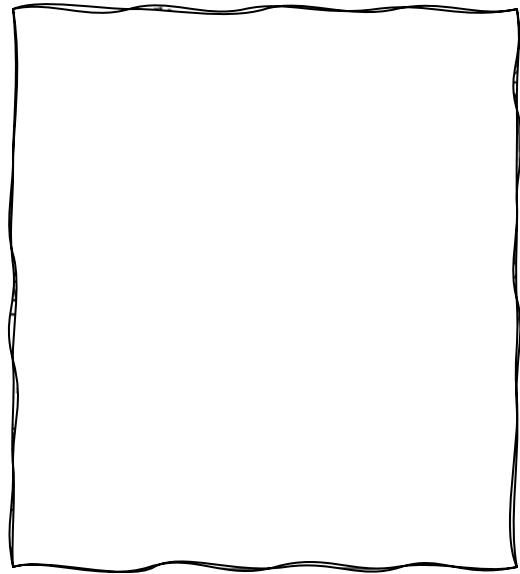
Program is `input.filter(e => ?)`

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

height 2

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

RESL co-design result



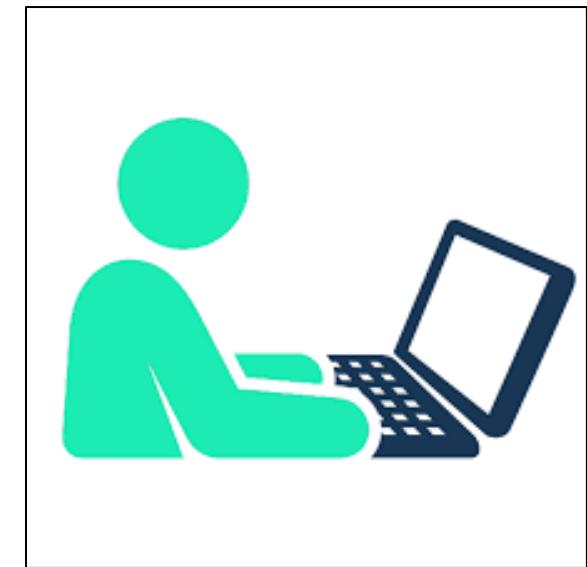
synthesizer

add local specifications

more forms of spec available

require users to introduce loops

need to find inner inputs for loops



user interaction

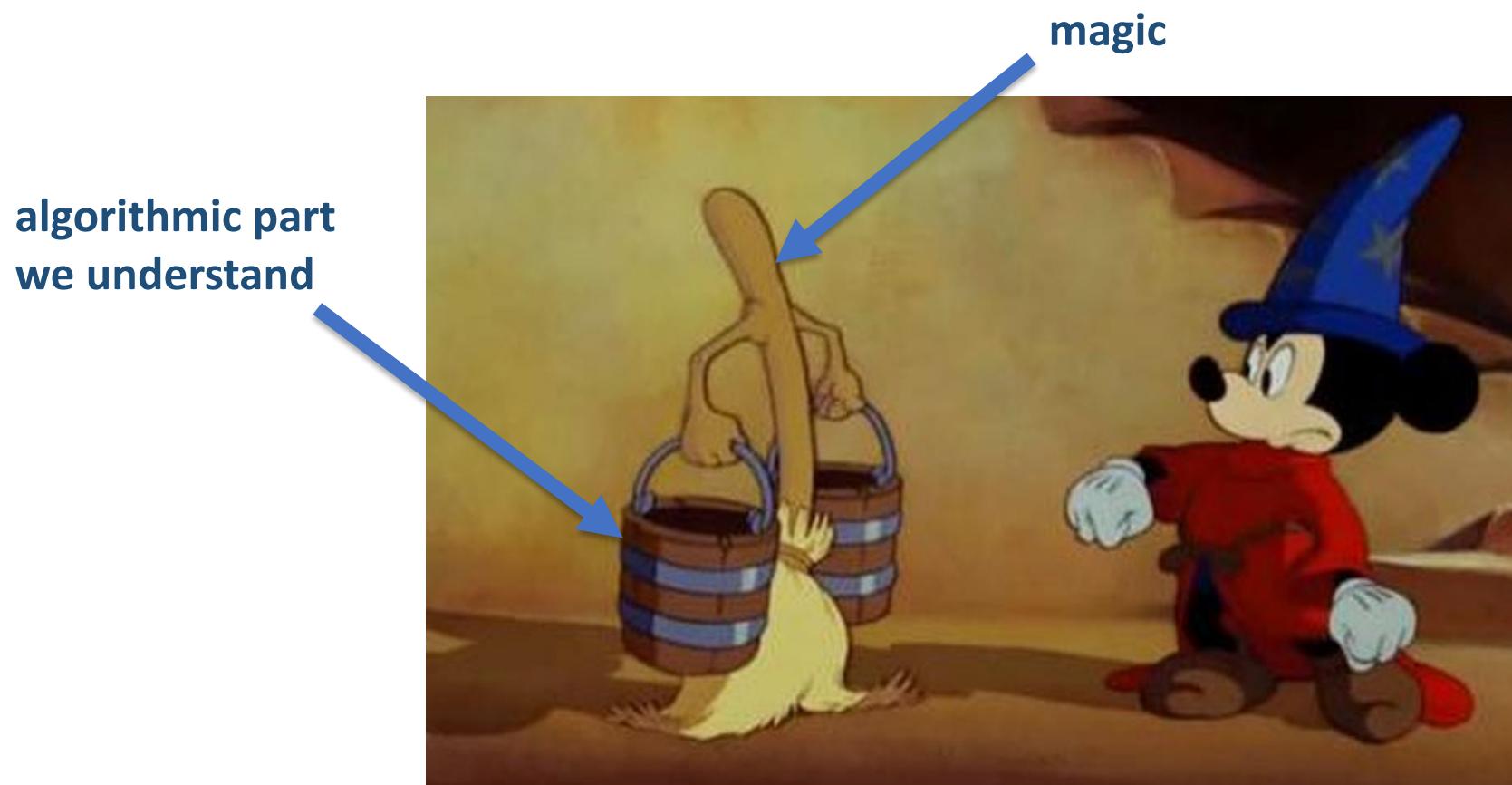
This is a lot of work.

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



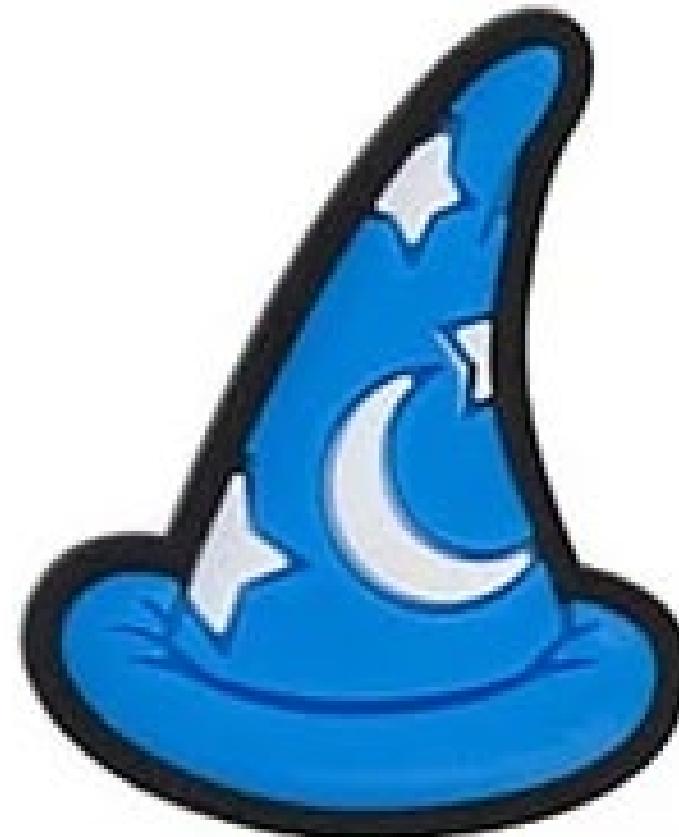
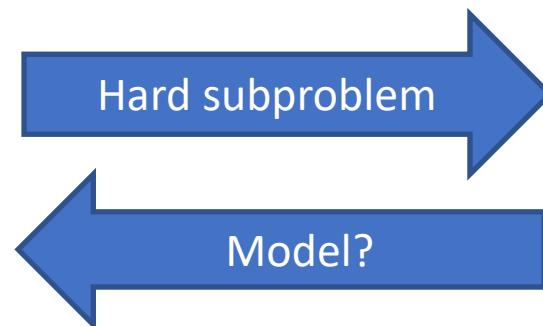
Magic, an incomplete definition

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

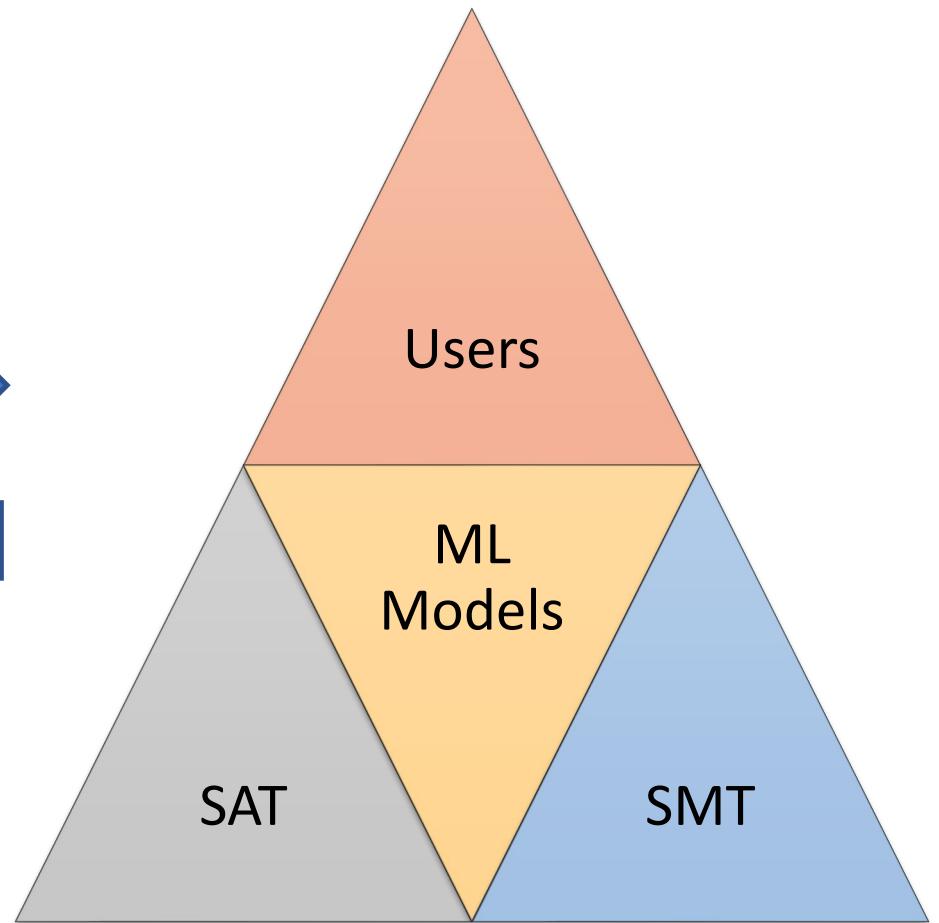
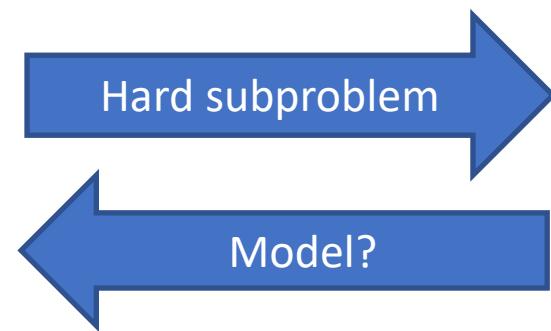


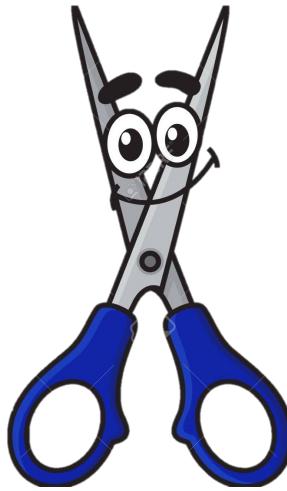
“incomplete assistants”

Available magics



Available magics



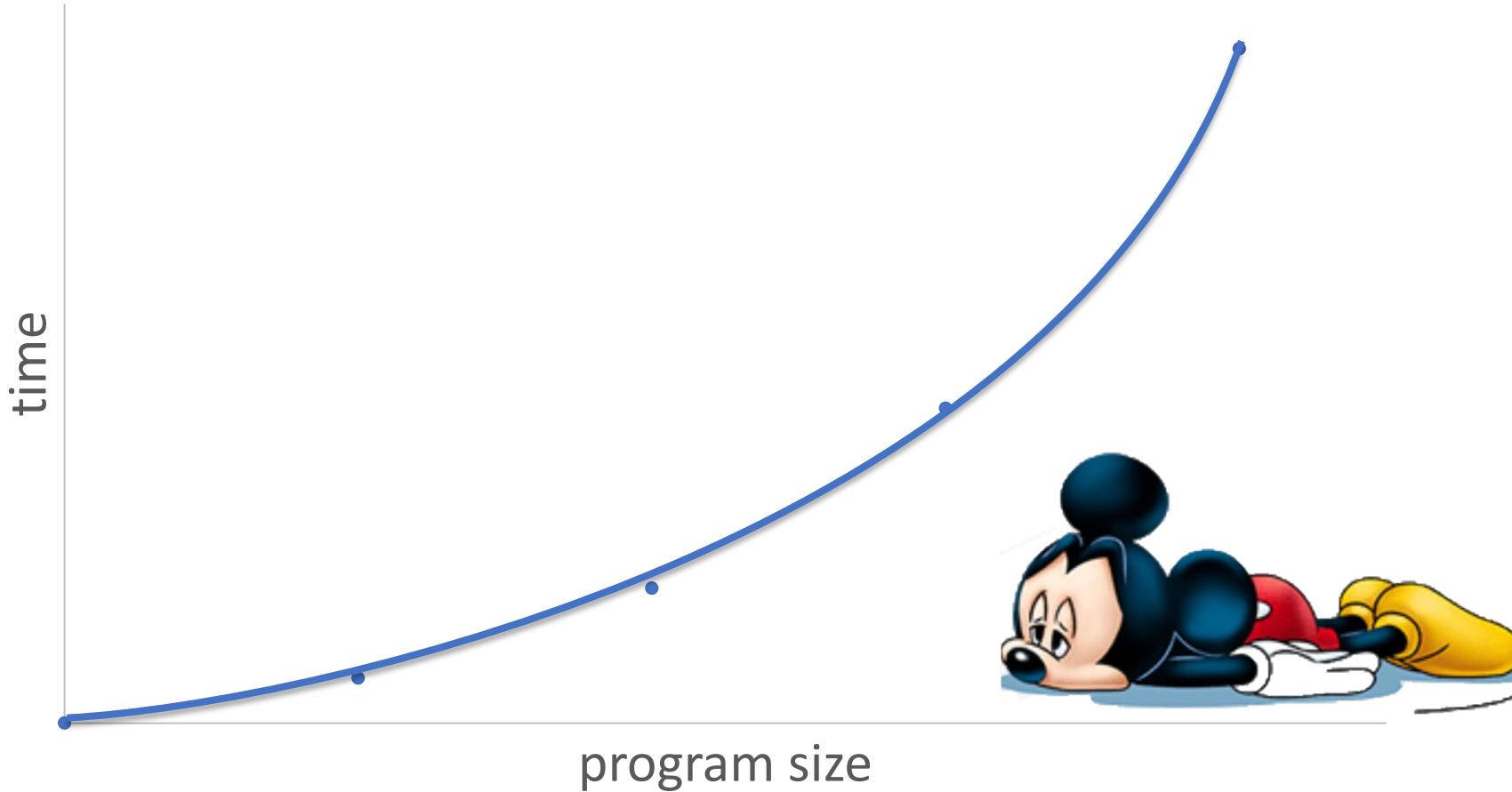


Example 2: Small-Step Synthesis

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

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

Program synthesis is hard



Problem: Abbreviate



This is what I want

"Augusta Ada King"

["Augusta" , "Ada" , "King"]

["A" , "A" , "K"]

"A . A . K"

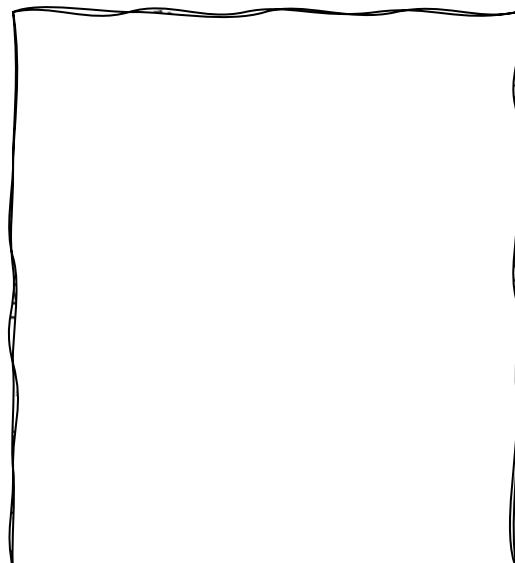
abbreviate()

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

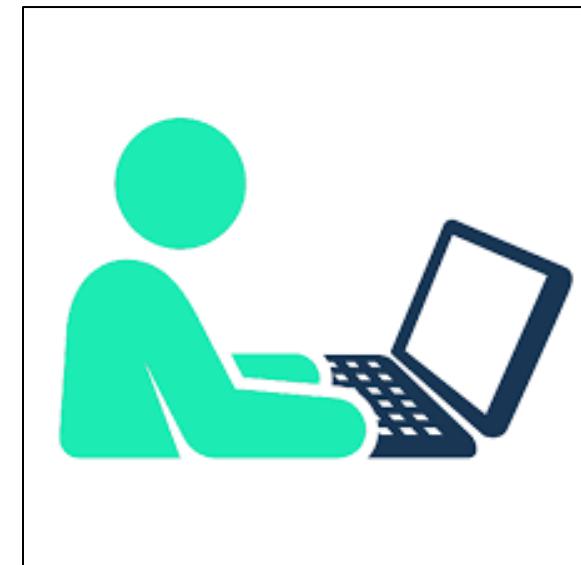
SnipPy



SnipPy Co-Design



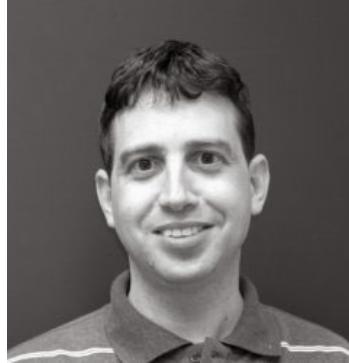
small-step specifications
interactive-time synthesis
reinforce already-correct outputs
result with fewer examples
force trace-complete examples
data-dependent loop synthesis



With thanks to my co-authors



Roi Gabai



Shachar Itzhaky



Eran Yahav



Kasra Ferdowsifard



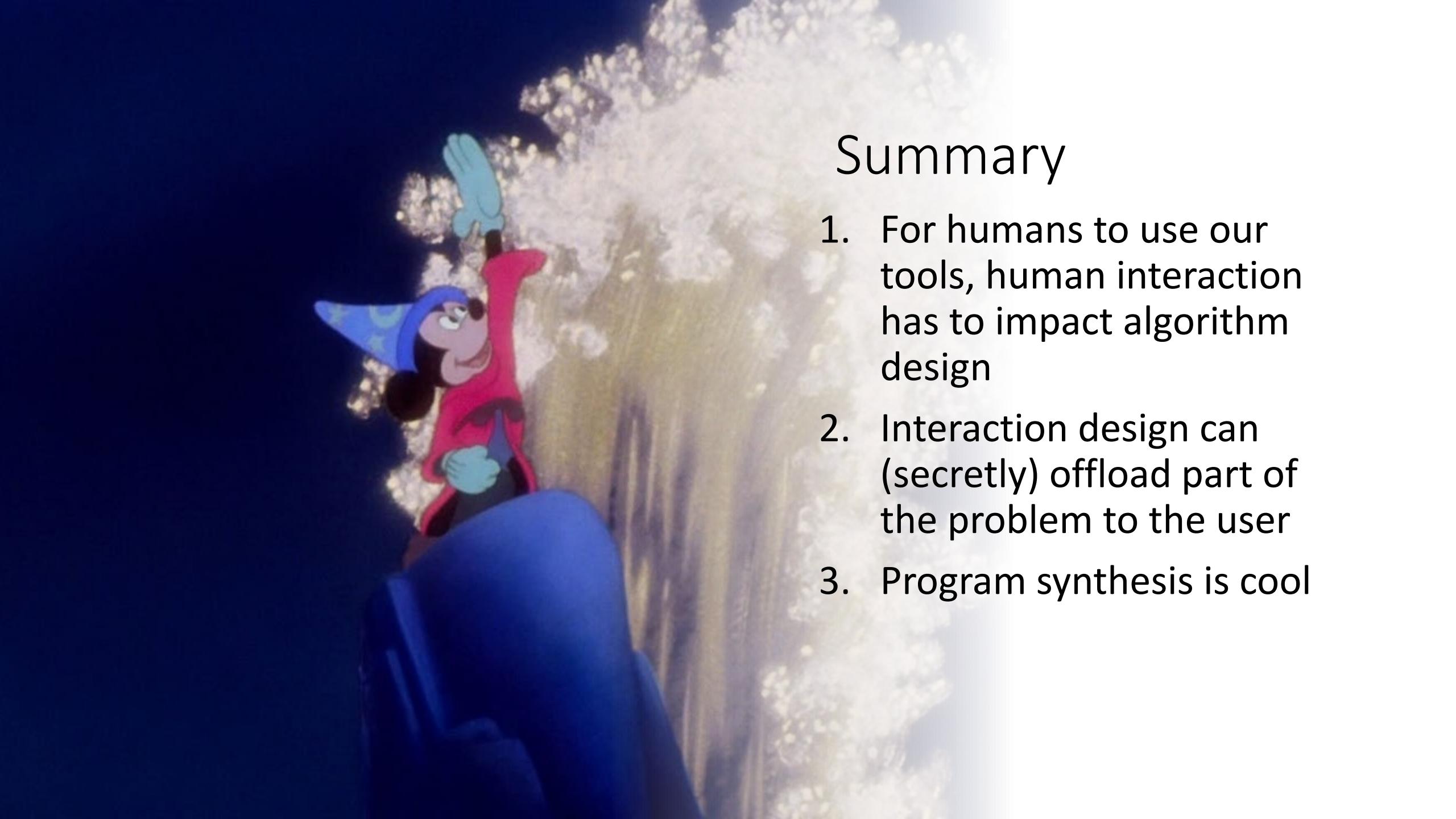
Shraddha Barke



Sorin Lerner



Nadia Polikarpova

A cartoon character with a blue pointed hat and a red coat is standing next to a large tree trunk. The character is looking up at a bright, glowing light source that appears to be emanating from the tree. The background is dark, suggesting it might be night or a forest setting.

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