Program Synthesis Co-Design

Hila Peleg – Technion

@HilaCodes
Program Synthesis

This is what I want
I’ll tell you what I want

Int -> [Maybe a] -> a
I’ll tell you what I want

Looks like $x \cdot f(a, b, ???)$
I’ll tell you what I want

\[ \exists m. \forall x. \phi(x) \land \neg \psi(x) \]
\[ \lor \psi(f(x)) \]
\[ \Rightarrow (\forall e. [m](x) = e \Rightarrow \forall (e)) \]
\[ \land (\exists y. p(y) \leftrightarrow q(x)) \lor \tau(m) = \tau_1 \]
I’ll tell you what I want
Program Synthesis

\( \varphi \)
The search is highly customized

Search by example

Search by type

Search by natural language
Interaction models

How do I express what I want?

Is this what I wanted?

This is what I want

\[ \varphi \]

\[ p \]
Interaction models are part of the package

Interaction models are part of the package $\varphi_p$. This is what I want.
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

LooPy: interactive program synthesis with control structures [OOPSLA’21]
Example I: interaction ideas

Interaction idea: assignment order is hard for users

Now to synthesize it:

```
  for c in s[1]:
      if c == last:
          count += 1
      else:
          rs += str(count) + last
          count = 1
          last = c
  return rs
```

LooPy: interactive program synthesis with control structures [OOPSLA’21]
Example II: synthesizer constraints

Independent iterations are fine:

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

LooPy: interactive program synthesis with control structures [OOPSLA’21]
Example II: synthesizer constraints

Independent iterations are fine:

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

Data dependencies make everything hard:

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

LooPy: interactive program synthesis with control structures [OOPSLA’21]
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} \rightarrow 0
{arr \mapsto [1], s \mapsto 0} \rightarrow 1
{arr \mapsto [1,2], s \mapsto 1} \rightarrow 3
```

LooPy: interactive program synthesis with control structures [OOPSLA’21]
Example II: synthesizer constraints

Independent iterations are fine: Data dependencies make everything hard:

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

⇓

```
s = 0
for x in arr:
s = ??
⇒
```

Common solution: trace-complete specifications

```
arr ↦ \[
\], s ↦ 0 → 0
```

```
arr ↦ \[1\], s ↦ 0 → 1
```

```
arr ↦ \[1,2\], s ↦ 1 → 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

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

**Example:**
- Input: [1, 9, 5]
- Sorted: [1, 5, 9]
- Median: 5

```
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**

```plaintext
> 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**

```
/ 
├── input
│   ├── length
│   └── 2
└── .
```

Exclude

Retain
Read-Eval-Synth Loop (RESL)
Challenge 1: Local specifications interfere with synthesis algorithm

Challenge 2: Loops are hard for the synthesizer
Challenge 1: Synthesis breaks

Why did this happen?

\[
\{ \text{input} \rightarrow [1,9,5] \} \rightarrow 5
\]

\[
\text{Retain} (\text{input.length} / 2)
\]

No program

\[
\text{input.sort()} [\text{Math.floor(input.length} / 2)]
\]
is in the synthesizer’s space...
Searching the space by enumeration

height 0

\[ S \rightarrow E \]
\[ E \rightarrow 0|1|x \]
Searching the space by enumeration

height 0

\[ S \rightarrow \quad E \]

\[ E \rightarrow \quad 0\mid1\midx \]
Searching the space by enumeration

height 0

\[
\begin{array}{ccc}
0 & 1 & x \\
\end{array}
\]

height 1

\[
\begin{array}{cccccccc}
0+0 & 0+1 & 0+x & 1+0 & 1+1 & 1+x & x+0 \\
x+1 & x+x &
\end{array}
\]

\[
\begin{align*}
S & \rightarrow \quad E \\
E & \rightarrow \quad 0|1|x
\end{align*}
\]
Searching the space by enumeration

height 0

height 1

\[
S \rightarrow E
\]
\[
E \rightarrow 0|1|x
\]
Searching the space by enumeration

height 0

\[
\begin{array}{ccc}
0 & 1 & x \\
\end{array}
\]

height 1

\[
\begin{array}{cccccccc}
0+0 & 0+1 & 0+x & 1+0 & 1+1 & 1+x & x+0 \\
x+1 & x+x & 0*0 & 0*1 & 0*x & 1*0 & 1*1 \\
1*x & x*0 & x*1 & x*x \\
\end{array}
\]

height 2

\[
\begin{array}{cccccccc}
0+0+0 & 0+0+1 & 0+0+x & 0+1+0 & 0+1+1 & \ldots \\
\end{array}
\]

\[
S \rightarrow E \\
E \rightarrow 0|1|x
\]
Searching the space by enumeration

| Height 0 | | | |
|---|---|---|
| **0** | **1** | **x** |
| <0,0> | <1,1> | <1,3> |

| Height 1 | | | |
|---|---|---|
| **0+0** | **0+1** | **0+x** |
| <0,0> | <1,1> | <1,3> |
| **1+0** | **1+1** | **1+x** |
| <1,1> | <2,2> | <2,4> |
| **x+0** | **x+1** | **x+x** |
| <1,3> | <2,4> | <2,6> |
| **0*0** | **0*1** | **0*x** |
| <0,0> | <0,0> | <0,0> |
| **1*0** | **1*1** | **x*1** |
| <0,0> | <1,1> | <1,3> |
| **x*0** | **x*x** | **x*1** |
| <0,0> | <1,9> | <1,3> |

| Height 2 | | | |
|---|---|---|
| **0+0+0** | **0+0+1** | **0+0+x** |
| <0,0> | <1,1> | <1,3> |
| **0+1+0** | **0+1+1** | **0+1+x** |
| <1,1> | <2,2> | <1,9> |

\[ S \to E \]
\[ E \to 0|1|x \]

\[ i_0 = \{ x \mapsto 1 \} \]
\[ i_1 = \{ x \mapsto 3 \} \]
Searching the space by enumeration

height 0

\[
\begin{array}{ccc}
\text{0} & \text{1} & \text{x} \\
<0,0> & <1,1> & <1,3>
\end{array}
\]

height 1

\[
\begin{array}{cccc}
\text{0+0} & \text{0+1} & \text{0+x} & \text{1+0} \\
<0,0> & <1,1> & <1,3> & <2,2>
\end{array}
\]

\[
\begin{array}{cccc}
\text{1+1} & \text{1+x} & \text{x+0} \\
<2,4> & <2,4> & <1,3>
\end{array}
\]

i_0 = \{x \mapsto 1\}

i_1 = \{x \mapsto 3 \}

height 2

\[
\begin{array}{cccc}
\text{0+0+0} & \text{0+0+1} & \text{0+0+x} & \text{0+1+0} & \text{0+1+1} \\
<0,0> & <1,1> & <1,3> & <1,1> & <2,2>
\end{array}
\]

...
Searching the space by enumeration

\[
\begin{align*}
S & \rightarrow E \\
E & \rightarrow 0 \mid 1 \mid x
\end{align*}
\]

\[
i_0 = \{ x \mapsto 1 \} \\
i_1 = \{ x \mapsto 3 \}
\]
Searching the space by enumeration

\[ S \rightarrow E \]
\[ E \rightarrow 0|1|x \]

\[ i_0 = \{ x \mapsto 1 \} \]
\[ i_1 = \{ x \mapsto 3 \} \]
Searching the space by enumeration

height 0

\[
\begin{array}{ccc}
0 & 1 & x \\
<0,0> & <1,1> & <1,3>
\end{array}
\]

height 1

\[
\begin{array}{ccc}
0+x & 1+1 & 1+x & x+0 \\
<1,3> & <2,2> & <2,4> & <1,3>
\end{array}
\]

\[
\begin{array}{ccc}
x+1 & x+x \\
<2,4> & <2,6>
\end{array}
\]

\[
\begin{array}{ccc}
x+x \\
<2,4>
\end{array}
\]

\[
\begin{array}{ccc}
1*x & x*1 & x*x \\
<1,3> & <1,3> & <1,9>
\end{array}
\]

i₀ = \{x → 1\}

i₁ = \{x → 3\}

S → E
E → 0|1|x

height 2

\[
\begin{array}{ccc}
0+0+x & 0+1+1 \\
<1,3> & <2,2>
\end{array}
\]

...
Searching the space by enumeration

height 0

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>x</td>
</tr>
<tr>
<td>&lt;0,0&gt;</td>
<td>&lt;1,1&gt;</td>
<td>&lt;1,3&gt;</td>
</tr>
</tbody>
</table>

height 1

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1+1</td>
<td>1+x</td>
<td></td>
</tr>
<tr>
<td>&lt;2,2&gt;</td>
<td>&lt;2,4&gt;</td>
<td></td>
</tr>
</tbody>
</table>

| x+x |
| <2,6> |

| x*x |
| <1,9> |

height 2

...
Challenge 1 (ctd): why does synthesis break?

\[
\{ \text{input} \rightarrow [1,9,5] \} \rightarrow 5
\]

\[\text{Retain(input.length / 2)}\]

height 1 \hspace{1cm} ... \hspace{1cm} \text{input.length} \hspace{1cm} ... \hspace{1cm} \]

\(<3>\)

height 2 \hspace{1cm} ... \hspace{1cm} \text{input.length} / 2 \hspace{1cm} ... \hspace{1cm} \]

\(<1.5>\)

\[i_0 = \{ \text{input} \rightarrow [1,9,5] \}\]
Challenge 1 (ctd): why does synthesis break?

\[
\begin{align*}
\{ \text{input} & \mapsto [1,9,5] \} \rightarrow 5 \\
\text{Retain}(\text{input.length} / 2)
\end{align*}
\]

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

height 1

\[
\begin{align*}
2+1 & \quad \text{...} \quad \text{input.length} \quad \text{...} \\
<3> & \quad \text{...} \quad <3>
\end{align*}
\]

height 2

\[
\begin{align*}
\text{...} \quad \text{input.length} / 2 \quad \text{...} \\
\text{...} \quad <1.5>
\end{align*}
\]
Challenge 1 (ctd): why does synthesis break?

\[
\begin{align*}
\{ \text{input} \mapsto [1,9,5] \} &\rightarrow 5 \\
\text{Retain}(\text{input.length} / 2) \\
\end{align*}
\]

\[
\begin{align*}
\text{height 1} & \quad 2+1 \\
& \quad <3> \\
\text{height 2} & \quad \ldots \quad \text{input.length} / 2 \quad \ldots \\
& \quad <1.5> \\
\end{align*}
\]

\[i_0 = \{ \text{input} \mapsto [1,9,5] \}\]
Challenge 1 (ctd): why does synthesis break?

\[
\begin{align*}
\{ \text{input} & \mapsto [1,9,5] \} \rightarrow 5 \\
\text{Retain} & (\text{input.length} / 2)
\end{align*}
\]

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

height 1

2+1

<3>

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?

\[
\{ \text{input} \mapsto [1,9,5] \} \rightarrow 5 \\
\text{Retain}(\text{input.length} / 2)
\]

height 1

\[
\begin{array}{ccc}
2+1 & \ldots & \text{input.length} & \ldots \\
<3, \text{not part of retain} & & <3, \text{part of retain}
\end{array}
\]

height 2

\[
\begin{array}{ccc}
\ldots & \text{input.length} / 2 & \ldots \\
<1.5, \text{has retained expr}
\end{array}
\]
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:

<table>
<thead>
<tr>
<th>Type Constraints:</th>
<th>Number:</th>
<th>require</th>
<th>prohibit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean:</td>
<td>require</td>
<td>prohibit</td>
<td></td>
</tr>
<tr>
<td>String:</td>
<td>require</td>
<td>prohibit</td>
<td></td>
</tr>
<tr>
<td>Object:</td>
<td>require</td>
<td>prohibit</td>
<td></td>
</tr>
<tr>
<td>Array:</td>
<td>require</td>
<td>prohibit</td>
<td></td>
</tr>
</tbody>
</table>
RESL challenges

Challenge 1: Local specifications interfere with synthesis algorithm

Challenge 2: Loops are hard for the synthesizer

synthesizer

user interaction
Challenge 2: loops are hard for the synthesizer

\[
> \text{filter}([1, 9, 5, 2, 4], \quad e \Rightarrow e \mod 2 = 0) \\
[2, 4]
\]
Challenge 2: loops are hard for the synthesizer

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

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

\[
\text{height 3} \quad \text{...} \quad \text{input.filter(e => e \% 2 = 0)} \quad \text{...}
\]

\[
<[2,4]>
\]
Challenge 2: loops are hard for the synthesizer

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

height 3 ... \texttt{input.filter(e \mapsto e \% 2 = 0)} ...
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)` ...

\[ \langle \text{false, false, false, true, true} \rangle \]
RESL co-design result

- add local specifications
- more forms of spec available
- require users to introduce loops
- need to find inner inputs for loops

synthesizer

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

algorithmic part
we understand

magic
Magic, an incomplete definition

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

“incomplete assistants”
Available magics

Hard subproblem

Model?
Available magics

- Users
- ML Models
- SAT
- SMT

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

This is what I want

"Augusta Ada King"

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

abbreviate()

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

"A.A.K"
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

synthesizer

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