Goal: We want to automatically write code from the kinds of specifications humans can easily provide, such as examples or natural language instruction.

List Processing from IO:
- [1, 2, 3, 4, 5] → [2, 4]
- [7, 8, 0, 9] → [8, 0]

Text Editing from IO:
- Max Nye → Nye, M.
- Luke Hewitt → Hewitt, L.

Natural language + IO → code
- “Consider an array of numbers, find elements in the given array not divisible by two”
  - [1, 2, 3, 4, 5] → [1, 3, 5]
  - [7, 8, 0, 9] → [7, 9]
How might people solve problems like this?

Given:

- [1, 2, 3, 4, 5] → [2, 4]
- [0, 6, 2, 7] → [0, 6, 2]
- [5, 10, 5, 1, 8] → [10, 8]

Goal: Write a program which maps inputs to outputs
How might people solve problems like this?

People use a flexible trade-off between pattern recognition and reasoning.
Easy problem:

Spec:
[1, 2, 3, 4, 5] → [2, 4]
[0, 6, 2, 7] → [0, 6, 2]
[5, 10, 5, 1, 8] → [10, 8]

Solution:
Easy problem:

Spec:

\[
\begin{align*}
[1, 2, 3, 4, 5] & \rightarrow [2, 4] \\
[0, 6, 2, 7] & \rightarrow [0, 6, 2] \\
[5, 10, 5, 1, 8] & \rightarrow [10, 8]
\end{align*}
\]

Solution:

\[
\text{filter(\text{lambda } x: x\%2==0, \text{input})}
\]
Easy problem:

Spec:
[1, 2, 3, 4, 5] → [2, 4]
[0, 6, 2, 7] → [0, 6, 2]
[5, 10, 5, 1, 8] → [10, 8]

Solution:

```python
filter(lambda x: x%2==0, input)
```

Fast, using pattern recognition
More difficult problem:

Spec:

\[ [3, 4, 5, 6, 7] \rightarrow [4, 7] \]
\[ [10, 8, 7, 3, 2, 1] \rightarrow [10, 7, 1] \]
\[ [5, 1, 2, 13, 4] \rightarrow [1, 13, 4] \]

Solution:
More difficult problem:

Spec:
[3, 4, 5, 6, 7] → [4, 7]
[10, 8, 7, 3, 2, 1] → [10, 7, 1]
[5, 1, 2, 13, 4] → [1, 13, 4]

Solution:

filter(<SOMETHING>, input)

(Fast, using pattern recognition)
More difficult problem:

Spec:

\[
\begin{align*}
[3, 4, 5, 6, 7] & \rightarrow [4, 7] \\
[10, 8, 7, 3, 2, 1] & \rightarrow [10, 7, 1] \\
[5, 1, 2, 13, 4] & \rightarrow [1, 13, 4]
\end{align*}
\]

Solution:

\[
\text{filter}(\text{<SOMETHING>}, \text{input}) \quad \rightarrow \quad \text{filter}(\lambda x: x \% 3 == 1, \text{input})
\]

(Fast, using pattern recognition)
More difficult problem:

Spec:

\[ [3, 4, 5, 6, 7] \rightarrow [4, 7] \]
\[ [10, 8, 7, 3, 2, 1] \rightarrow [10, 7, 1] \]
\[ [5, 1, 2, 13, 4] \rightarrow [1, 13, 4] \]

Solution:

\begin{align*}
\text{filter(} & <\text{SOMETHING}>, \text{ input)} \quad \text{\text{\text{\rightarrow}}} \quad \text{filter(} & \text{lambda } x: x\%3==1, \text{ input)} \\
\text{Symbolic reasoning} \quad \text{\rightarrow} \quad \text{(Slow)}
\end{align*}
Very difficult problem:

Spec:

\[ [2, 5, 0, 16, 12] \rightarrow 0 \]
\[ [4, 23, 11, 9, 25] \rightarrow 25 \]
\[ [3, 29, 30, 14, 16] \rightarrow 14 \]
Very difficult problem:

Spec:
\[
\begin{align*}
[2, 5, 0, 16, 12] &\rightarrow 0 \\
[4, 23, 11, 9, 25] &\rightarrow 25 \\
[3, 29, 30, 14, 16] &\rightarrow 14 \\
[1, 7, 6, 9, 5] &\rightarrow 7 \\
[5, 5, 1, 8, 8, 12, 4] &\rightarrow 12 \\
[0, 4, 8, 5, 1] &\rightarrow 0 \\
[3, 7, 2, 9, 1] &\rightarrow 9 \\
[1, 0, 3, 7, 3, 8] &\rightarrow 0
\end{align*}
\]
Very difficult problem:

Spec:
[2, 5, 0, 16, 12] → 0
[4, 23, 11, 9, 25] → 25
[3, 29, 30, 14, 16] → 14

Solution:

\(<\text{SOMETHING}>\) → input[\text{input}[0]]

Symbolic reasoning

(Slow)
Q: How do we model this? A: Program sketches

[3, 4, 5, 6, 7] → [4, 7]
[10, 8, 7, 3, 2, 1] → [10, 7, 1]

Pattern recognition (e.g., neural network)

filter(<HOLE>, input)

Symbolic reasoning (e.g., guess and check)

filter(lambda x: x%3==1, input)

Flexible trade-off between pattern recognition and reasoning

Our system: **SketchAdapt**

- **Neural sketch generator**
  - Program specification: $[3, 4, 5, 6, 7] \rightarrow [4, 7]$  
  - $[10, 8, 7, 1] \rightarrow [10, 7, 1]$  
  - $[5, 1, 13, 4] \rightarrow [1, 13, 4]$  

- **Neural recognizer**
  - Production probabilities: $0.25, 0.05, 0.02, 0.01, 0.25, 0.30, \ldots$

- **Symbolic enumerator**
  - Filter: $\text{filter}(\lambda x: x \% 3 == 1, \text{input})$

- **Learned neural network**

- **Filter**
  - $\text{filter}(\text{<HOLE>}, \text{input})$
Our system: **SketchAdapt**

Program specification

Neural sketch generator

filter(<HOLE>, input)

Full program

Neural recognizer

Symbolic enumerator

filter(lambda x: x%3==1, input)

Production probabilities

Learned neural network

Sketch generator:
RNN that proposes program sketches (c.f. RobustFill)

Devlin et al, 2017
Balog et al, 2016

Program sketch

\[
\begin{align*}
[3, 4, 5, 6, 7] &\rightarrow [4, 7] \\
[10, 8, 7, 1] &\rightarrow [10, 7, 1] \\
[5, 1, 13, 4] &\rightarrow [1, 13, 4]
\end{align*}
\]
Our system: **SketchAdapt**

- **Neural sketch generator**
  - Filter \( \langle \text{HOLE} \rangle \), input

- **Symbolic enumerator**
  - Filter \( \lambda x: x \% 3 == 1 \), input

- **Neural recognizer**

Program specification: \([3, 4, 5, 6, 7] \rightarrow [4, 7] \)
\([10, 8, 7, 1] \rightarrow [10, 7, 1] \)
\([5, 1, 13, 4] \rightarrow [1, 13, 4] \)

**Production probabilities**

- Neural recognizer
- Symbolic synthesizer: **enumerator** that fills in sketches, guided by neural recognizer (c.f. DeepCoder)
- Sketch generator: RNN that proposes program sketches (c.f. RobustFill)

---

Devlin et al, 2017
Balog et al, 2016
Results: list processing

SketchAdapt can recognize familiar problems and generalize to unfamiliar problems

Trained on length 3 programs

Length 3 test programs: SketchAdapt

Graph showing the percentage of problems solved versus the number of candidates evaluated per problem. The graph compares different methods: Ours, Pattern recognition only (neural network), and Reasoning only (symbolic enumeration).
Results: list processing

SketchAdapt can recognize familiar problems and generalize to unfamiliar problems

- Trained on length 3 programs
  - Length 3 test programs:
  - Length 4 test programs:
    - Trained on length 3 programs

<table>
<thead>
<tr>
<th>Spec</th>
<th>Program</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consider an array of numbers, find elements in the given array not divisible by two</td>
<td><code>(filter (lambda ( == (% arg1 2 ) 1))</code></td>
</tr>
<tr>
<td>You are given an array of numbers, your task is to compute median in the given array with its digits reversed</td>
<td><code>(reduce(reverse(digits(deref (sort a) (/ (len a) 2)))) 0 (lambda2 (+(* arg1 10) arg2)))</code></td>
</tr>
</tbody>
</table>
Natural language + IO examples → Code

Requires less data than pure neural approaches:

![Chart showing performance of different models with varying numbers of training programs used. The chart includes bars for 'Our model', 'Generator only (RobustFill)', and 'Synthesizer only (Deepcoder)'. The x-axis represents the number of training programs used (2000, 4000, 6000, 8000, 79214), and the y-axis represents the percentage of test programs solved. The chart highlights SketchAdapt.]
Natural language + IO examples → Code

Requires less data than pure neural approaches:

Generalizes to unseen concepts:

Table 5. Algolisp generalization results: Trained on 8000 programs, excluding ‘Odd’ concept:

<table>
<thead>
<tr>
<th>Model</th>
<th>Even</th>
<th>Odd</th>
</tr>
</thead>
<tbody>
<tr>
<td>SketchAdapt (Ours)</td>
<td>34.4</td>
<td>29.8</td>
</tr>
<tr>
<td>Synthesizer only</td>
<td>23.7</td>
<td>0.0</td>
</tr>
<tr>
<td>Generator only</td>
<td>4.5</td>
<td>1.1</td>
</tr>
</tbody>
</table>
Learn to Infer Program Sketches

Come see our poster:
Today (Thurs) 06:30 - 09:00 PM @ Pacific Ballroom #182

[Image of neural network diagram]

- Learned neural network
- [1, 3, -4, 3] -> 3
- [-3, 0, 2, -1] -> 2
- [7, -4, -5, 2] -> 2
- Neural sketch generator, \( q_\phi([-|\mathcal{X}|]) \)
- Count > 0 (Map (HOLE))
- Enumerator
- Recognizer, \( r_\psi(\mathcal{X}, s) \)
- Production probabilities, \( \theta \)
- Count > 0 (Map +1 input)
- Program spec, \( \mathcal{X} \)
- Program sketch, \( s \)
- Program synthesizer
- Full program, \( F \)