Program Synthesis

Alex Polozov
Microsoft Research AI

polozov@microsoft.com
(circa 1950-1960)

Sort

Given: $A[1..n]$  
Produce: $B[1..n]$ ensuring $B_1 \leq \cdots \leq B_n$ and $B$ is a permutation of $A$
Modern Program Synthesis

- Input-output examples
- Natural language

User Intent

- Programming language
- Domain-specific language

Program Space

Search Algorithm
- Enumerative/deductive search
- Neural networks

Program
Application: Question answering

Natural language question

User Intent

Search Algorithm

Command interpretations + query plans

Program Space
Application: Data wrangling

Input-output examples

User Intent

Search Algorithm

String → String data transforms

Program Space

Excel Flash Fill
Application: Code generation

Program context/template

User Intent

Programming language

Program Space

Search Algorithm

17 October 2018

Advanced ML Day 2018
Outline

▶ Introduction
▶ Programming by Examples
▶ Neural-Guided Program Search
▶ Neural Program Synthesis
Outline

- Introduction
- Programming by Examples
- Neural-Guided Program Search
- Neural Program Synthesis
Flash Fill
[Gulwani, 2011]

<table>
<thead>
<tr>
<th>First Name</th>
<th>Middle Name</th>
<th>Last Name</th>
<th>Combined Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>Kristy</td>
<td>Mills</td>
<td>Mills, Mary K.</td>
</tr>
<tr>
<td>Craig</td>
<td>John</td>
<td>Garland</td>
<td>Garland, Craig J.</td>
</tr>
<tr>
<td>Andrew</td>
<td>John</td>
<td>Simpson</td>
<td></td>
</tr>
<tr>
<td>Rupert</td>
<td>L</td>
<td>Richards</td>
<td>Richards, Rupert L.</td>
</tr>
<tr>
<td>Joe</td>
<td></td>
<td>Bryant</td>
<td></td>
</tr>
<tr>
<td>Jane</td>
<td></td>
<td>Hart</td>
<td></td>
</tr>
<tr>
<td>Bob</td>
<td>JD</td>
<td>Bond</td>
<td>Bond, Bob JD.</td>
</tr>
</tbody>
</table>

**Code**

```
Concat(input[2], Const("", ""), input[0], Const(" "),
       let v = input[1] in Substring(v, AbsPos(v, 0), RegexPos(v, Uppercase, ",", -1)),
       Const("."))
```
Flash Fill

[Gulwani, 2011]

```
if (input[1] == '') then Concat(input[2], Const("", ""), input[0], Const(".")) else Concat(input[2], Const("", ""), input[0], Const("")),
let v = input[1] in Substring(v, AbsPos(v, 0), RegexPos(v, UpperCase, '', -1)), Const("."))
```

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>First Name</td>
<td>Middle Name</td>
<td>Last Name</td>
<td>Combined Name</td>
</tr>
<tr>
<td>2</td>
<td>Mary</td>
<td>Kristy</td>
<td>Mills</td>
<td>Mills, Mary K.</td>
</tr>
<tr>
<td>3</td>
<td>Craig</td>
<td>John</td>
<td>Garland</td>
<td>Garland, Craig J.</td>
</tr>
<tr>
<td>4</td>
<td>Andrew</td>
<td>Simpson</td>
<td>Simpson</td>
<td>Simpson, A.</td>
</tr>
<tr>
<td>5</td>
<td>Rupert L</td>
<td>Richards</td>
<td>Richards</td>
<td>Richards, Rupert L.</td>
</tr>
<tr>
<td>6</td>
<td>Joe</td>
<td>Bryant</td>
<td>Bryant</td>
<td>Bryant, J.</td>
</tr>
<tr>
<td>7</td>
<td>Jane</td>
<td>Hart</td>
<td>Hart</td>
<td>Hart, J.</td>
</tr>
<tr>
<td>8</td>
<td>Bob</td>
<td>JD</td>
<td>Bond</td>
<td>Bond, Bob JD.</td>
</tr>
</tbody>
</table>

17 October 2018 Advanced ML Day 2018
if (input[1] == '') then Concat(input[2], Const("", ""), input[0], Const(".\"")) else Concat(input[2], Const("", ""), input[0], Const(" ")),
let v = input[1] in Substring(v, AbsPos(v, 0), RegexPos(v, Uppercase, "", -1)),
from the beginning till the last uppercase character
PBE Challenges

1. The synthesized program *must* satisfy the examples.

2. Examples are severely ambiguous.
   - Should learn the *intended* program, not just one that satisfies the examples
   - How do we know when to stop?

3. Interactive UX requires quick turnaround.

4. The DSL must be expressive but concise.
@input string[] inputs;

@start string transform := atom | Concat(atom, transform);
string atom := Const(s)
    | let v = std.Kth(inputs, k) in
      Substring(v, std.Pair(pos, pos));
uint? pos := AbsPos(v, k)
    | RegexPos(v, std.Pair(r, r), k);

string s;     Regex r;     int k;
Deductive Search in PBE

[Polozov & Gulwani, 2015]

```
Input x       Output y
alice liddell To: al
bob o’reilly To: bo

1. Select a hole (initially root).
2. Select a DSL operator to insert in the hole.

```

```
partial Abstract Syntax Tree (AST)

```

"To: "

```
Concat

Atom

```

```
Concat

Constant

```

```
Output

al
bo

```
Deductive Search in PBE

[Polozov & Gulwani, 2015]

Input $x$  
| alice liddell  | To: al |
| bob o’reilly   | To: bo |

Output $y$

1. Select a hole (initially root).
2. Select a DSL operator to insert in the hole.
3. Propagate the examples.
Deductive Search in PBE

[Polozov & Gulwani, 2015]

```
def propagate_concat(output_spec φ = {x ⊸ y}):
    return φ_prefix = {x ⊸ y[1:1] ∨ y[2:2] ∨ ...}
```

1. Select a hole (initially root).
2. Select a DSL operator to insert in the hole.
3. Propagate the examples.

**Input x**
- alice liddell  To: al
- bob o’reilly  To: bo

**Output y**
- a
- l
- b
- o
Deductive Search in PBE

[Polozov & Gulwani, 2015]

Input $x$ | Output $y$
---|---
alice liddell | To: al
bob o’reilly | To: bo

1. Select a hole (initially root).
2. Select a DSL operator to insert in the hole.
3. Propagate the examples.

- Correct by construction
- Can incorporate program ranking
- Example propagation easy to write for many operations & domains
Microsoft PROSE SDK
https://microsoft.github.io/prose
Microsoft PROSE SDK

Program Synthesis Framework

Synthesis Strategies → PROSE

App

Synthesizer

I/O Examples

DSL Definition

Programs
Deductive Search in PBE
[Polozov & Gulwani, 2015]

Input $x$  |  Output $y$
---|---
alice liddell  |  To: al
bob o’reilly  |  To: bo

1. Select a hole (initially root).
2. Select a DSL operator to insert in the hole.
3. Propagate the examples.

- Correct by construction
- Can incorporate program ranking
- Example propagation easy to write for many operations & domains
Outline

- Introduction
- Programming by Examples
- Neural-Guided Program Search
- Neural Program Synthesis
Deductive Search

I/O Examples → Search Space → Program

Why so slow? Explores the entire search space (unless deduction prunes some of it)
Machine-learned insights

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>alice liddell</td>
<td>To: al</td>
</tr>
<tr>
<td>bob o’reilly</td>
<td>To: bo</td>
</tr>
</tbody>
</table>

Can’t be a substring, requires concatenation
DeepCoder: Learning to Write Programs
[Balog et al., 2017]

Idea: Order the search space based on a priority list from DNN before starting
Neural-Guided Deductive Search

[Kalyan et al., 2018]

**Idea:** Order the search space based on a priority list from DNN *at each step*
Neural-Guided Deductive Search

[Kalyan et al., 2018]

**Idea:** Order the search space based on a priority list from DNN at each step

![Diagram of search space and I/O examples leading to a program with search priorities]
Neural-Guided Deductive Search

[Kalyan et al., 2018]

Idea: Order the search space based on a priority list from DNN at each step
Search branch prediction

- Collect a complete dataset of intermediate search results:

  at a search branch $N := F_1(...) | F_2(...) | \cdots | F_k(\ldots)$
  
  given a spec $\phi = \{x \mapsto y\}$
  
  produced programs $P_1, \ldots, P_k$ with scores $h(P_1, \phi), \ldots, h(P_k, \phi)$

- Learn a predictive model $f$ s.t. $f(F_j, \phi) \approx h(P_j, \phi)$

- Train using squared-error loss over program scores:

  $$\mathcal{L}(f; F_j, \phi) = \left[f(F_j, \phi) - h(P_j, \phi)\right]^2$$
Model

- Guides the search toward higher-quality programs
- Eliminates unproductive search branches
- Balances performance vs. generalization using branch-and-bound
“Neuro-Symbolic” Architectures

Guiding a symbolic search

Verifying neutrally generated programs
Adding semantic structure to data
Natural Language → SQL

**Q:** When did Greece hold its last Summer Olympics?

**SELECT** `MAX(Year)` **FROM** `SummerOlympics` **WHERE** `Country='Greece'`;

**A:** 2004
Pointer-SQL

[Wang et al., 2017]

- Copying mechanism for column names
- Constant token
- Copying mechanism for words/values

What's the number of round with opponent Haugar?

LSTM ↔ LSTM

season
competition
round
opponent ...

= Haugar

opponent

WHERE opponent
Natural Language → SQL

**Q:** When did **Greece** hold its **last** Summer Olympics?

```sql
SELECT MAX(Year) FROM SummerOlympics WHERE Country='Greece';
```

**A:** 2004

**Execution Guidance** during inference:
1. Backtrack if partial query has type error
2. Backtrack if “... WHERE COND” is empty
3. Constrain the decoder using grammar types
4. Nondeterministic oracles during training

### Relevant Information:
1. Column names
2. Column data types
3. Matches between table data and question
Execution Guidance [Wang et al., 2018]

When did Greece hold its last Summer Olympics after 1950?

+ [Year, City, Country, Nations]

LSTM Decoder

1) Eliminate syntax errors from the beam after partial execution
2) Eliminate clauses with empty outputs from the beam after partial execution
3) Restrict the output vocab at each step to enforce SQL grammar correctness

SELECT MAX Year FROM SummerOlympics WHERE Country = 'Greece' AND Year > 1950

SUM City

MIN Country

City = 1950
Coarse→Fine sketch generation

[Solar-Lezama, 2006] [Murali et al., 2018] [Dong & Lapata, 2018]
Outline

- Introduction
- Programming by Examples
- Neural-Guided Program Search
- Neural Program Synthesis
Programs: the ML Perspective

Approach 1: Sequence of Words (re-using NLP ideas)
Programs: the ML Perspective

**Approach 1: Sequence of Words**

(re-using NLP ideas)

Programs are different from natural language:

- Semantics for keywords already known
- Many words (APIs, local methods) only used seldomly
- Long-distance dependencies common
Programs: the ML Perspective

Approach 1.1: Sequence/tree of words (re-using NLP ideas)

Programs are different from natural language:
- Semantics for keywords already known
- Many words (APIs, local methods) only used seldomly
- Long-distance dependencies common
Programs: the ML Perspective

Approach 1.1: Sequence/tree of words  (re-using NLP ideas)

Programs are different from natural language:
- Semantics for keywords already known
- Many words (APIs, local methods) only used seldomly
- Long-distance dependencies common

Approach 2: Graphs
- Nodes labelled by semantic information
- Edges for semantic relationships
Programs: the ML Perspective

Approach 1.1: Sequence/tree of words (re-using NLP ideas)

Programs are different from natural language:
- Semantics for keywords already known
- Many words (APIs, local methods) only used seldomly
- Long-distance dependencies common

Approach 2: Graphs
- Nodes labelled by semantic information
- Edges for semantic relationships
Programs as Graphs: Key Idea

```c
int SumPositive(int[] arr, int lim) {
    int sum = 0;
    for (int i = 0; i < lim; i++)
        if (arr[i] > 0)
            sum += arr[i];

    return sum;
}
```
Programs as Graphs: Key Idea

```c
int SumPositive(int[] arr, int lim) {
    int sum = 0;
    for (int i = 0; i < lim; ++i)
        if (arr[i] > 0)
            sum += arr[i];
    return sum;
}
```
Programs as Graphs: Syntax

```
Assert.NotNull(clazz);
```

- Next Token
- AST Child

Diagram:
- ExpressionStatement
  - InvocationExpression
    - MemberAccessExpression
      - Assert
      - .
      - NotNull
      - ( ...
    - ArgumentList

17 October 2018
Advanced ML Day 2018
Programs as Graphs: Data Flow

```plaintext
(x, y) = Foo();
while (x > 0)
    x = x + y;
```

- Last Write
- Last Use
- Computed From
Graph Neural Networks: Extending RNNs

[Li et al., 2016] [Allamanis et al., 2018]

embed('the') = 

---

Chain structured data
(e.g. text)

---

△ Recurrent unit
Graph Neural Networks: States

\[ v_1 \quad v_2 \quad v_3 \quad v_6 \quad v_5 \quad v_4 \quad v_7 \quad v_8 \quad v_9 \]
Graph Neural Networks: States

Label Embedding

|   | -0.7 | 12   | 0.18 | -0.9 | 0.32 |

Graph:

- $v_1$
- $v_2$
- $v_3$
- $v_4$
- $v_5$
- $v_6$
- $v_7$
- $v_8$
- $v_9$
Graph Neural Networks: States

Edge Type 1 / $NN_1$
Graph Neural Networks: States

- Edge Type 1 / $NN_1$
- Edge Type 2 / $NN_2$
Graph Neural Networks: Propagation

- $NN_1$
- $NN_2$
- Recurrent unit
Graph Neural Networks: Propagation

- $NN_1$
- $NN_2$
- Recurrent unit
Expression Completion Task

```csharp
int methParamCount = 0;
if (paramCount > 0) {
    IParameterTypeInfo[] moduleParamArr =
        GetParamTypeInfo(Dummy.Signature, paramCount);
    methParamCount = moduleParamArr.Length;
}
if (  ) {
    IParameterTypeInfo[] moduleParamArr =
        GetParamTypeInfo(Dummy.Signature,
            paramCount - methParamCount);
}
```
Expression Completion Task

```csharp
int methParamCount = 0;
if (paramCount > 0) {
    IParameterTypeInfo[] moduleParamArr =
        GetParamTypeInfo(Dummy.Signature, paramCount);
    methParamCount = moduleParamArr.Length;
}
if (/* placeholder */) {
    IParameterTypeInfo[] moduleParamArr =
        GetParamTypeInfo(Dummy.Signature,
                         paramCount - methParamCount);
}
```

"Classic" Graph Neural Network
Expression Completion Task

```
int methParamCount = 0;
if (paramCount > 0) {
    IParameterTypeInfo[] moduleParamArr =
        GetParamTypeInfo(Dummy.Signature, paramCount);
    methParamCount = moduleParamArr.Length;
}
if (/* Hole */) {
    IParameterTypeInfo[] moduleParamArr =
        GetParamTypeInfo(Dummy.Signature,
                         paramCount - methParamCount);
}
```

"Classic" Graph Neural Network
Expression Completion Task

```csharp
int methParamCount = 0;
if (paramCount > 0) {
    IParmParameterTypeInformation[] moduleParamArr =
        GetParamTypeInformations(Dummy.Signature, paramCount);
    methParamCount = moduleParamArr.Length;
}
if ( methParamCount ) {
    IParmParameterTypeInformation[] moduleParamArr =
        GetParamTypeInformations(Dummy.Signature,
                                  paramCount - methParamCount);
}
```
Expression Completion Task

```csharp
int methParamCount = 0;
if (paramCount > 0) {
    I ParameterTypeInformation[] moduleParamArr =
    Get ParamTypeInformations(Dummy.Signature, paramCount);
    methParamCount = moduleParamArr.Length;
}
if (paramCount > methParamCount) {
    I ParameterTypeInformation[] moduleParamArr =
    Get ParamTypeInformations(Dummy.Signature,
        paramCount - methParamCount);
}
```
Sequential Graph Neural Network

[Brockschmidt et al., 2018]

Expression

Expression

- 

Expression

Variables in Scope

i

j
Sequential Graph Neural Network
[Brockschmidt et al., 2018]

Variables in Scope
- i
- j

Expression
- Expression
  - i
  - Expression

Diagram represents a graph neural network for sequential data.
Sequential Graph Neural Network
[Brockschmidt et al., 2018]
Sequential Graph Neural Network
[Brockschmidt et al., 2018]

Variables in Scope:
- i
- j

Diagram:
- Expression
  - i
  - Expression
  - -
  - Expression
  - +
  - Expression
Sequential Graph Neural Network
[Brockschmidt et al., 2018]
Sequential Graph Neural Network
[Brock and Schmidhuber, 2018]

Variables in Scope

Expression

Expression

Expression

Expression

Expression

Expression

Expression
Sequential Graph Neural Network
[Brockschmidt et al., 2018]
Examples

```csharp
int methParamCount = 0;
if (paramCount > 0) {
    IParameterTypeInfo[] moduleParamArr =
        GetParamTypeInformations(Dummy.Signature, paramCount);
    methParamCount = moduleParamArr.Length;
}
if (paramCount > methParamCount) {
    IParameterTypeInfo[] moduleParamArr =
        GetParamTypeInformations(Dummy.Signature,
                                   paramCount - methParamCount);
}

caretPos--;
if (caretPos < 0) {
    caretPos = 0;
}

int len = inputString.Length;
if (caretPos >= len) {
    caretPos = len - 1;
}

G → NAG:
len++ (33.6%)
len-1 (21.9%)
len+1 (14.6%)

char character = originalName[i];
if (character == '<') {
    ++startTagCount;
    builder.Append(' ');  
} else if (startTagCount > 0) {
    if (character == '>') {
        --startTagCount;
    }

G → NAG:
character == UNK_CHAR_LITERAL (75.5%)
character == ' ' (2.6%)
character != 'UNK_CHAR (2.5%)
```

17 October 2018

Advanced ML Day 2018
Application: Code generation

Program context/template

User Intent

Search Algorithm

Programming language

Program Space

17 October 2018

Advanced ML Day 2018
Further reading


Microsoft PROSE SDK. [https://microsoft.github.io/prose/](https://microsoft.github.io/prose/)


Takeaways

• Program synthesis = Language + Spec + Search algorithm

• For PBE, enumerative/deductive search ensures correctness
  • Performance improved using neural-guided search

• For other specs (language, context), use graph-based or recurrent neural networks
  • Ensure correctness using execution guidance & program structure

• Solicit or learn hints: sketches, grammars, ranking functions

• Make use of existing frameworks: PROSE, Sketch, Rosette