Neural Guided Constraint Logic Programming for Program Synthesis

Lisa Zhang\textsuperscript{1,2}, Gregory Rosenblatt\textsuperscript{4}, Ethan Fetaya\textsuperscript{1,2}, Renjie Liao\textsuperscript{1,2,3}, William E. Byrd\textsuperscript{4}, Matthew Mithg\textsuperscript{3}, Raquel Urtasun\textsuperscript{1,2,3}, Richard Zemel\textsuperscript{1,2}

\textsuperscript{1}University of Toronto, \textsuperscript{2}Vector Institute, \textsuperscript{3}Uber ATG, \textsuperscript{4}University of Alabama at Birmingham

Programming By Example

\textbf{Goal}: synthesize program specified in terms of input/output examples.

\textbf{Approaches}: enumerative type-based search methods like \(\lambda^c\), Myth, Escher, Machine Learning work uses methods like conditional program generation, differentiable programming, and neural guided synthesis.

\textbf{Our approach}: use a ML agent to guide the search, but additionally give ML agent internal state of symbolic system.

Background

We use the constraint logic programming language miniKanren as the symbolic system.

- miniKanren is flexible: can synthesize dynamically typed recursive programs
- write a relational interpreter in miniKanren: a relational form (evalo P I O) of interpreter (eval P I) = 0
- relations like evalo can be thought of as constraints
- query miniKanren to find solutions to \(P\) in (evalo \(P\) I O) by iteratively expanding relation evalo with its definition: (evalo \(P\) I O) \(\Rightarrow\) DISJ \(\Rightarrow\) (evalo (quote \(P\)) I O) -> (evalo (car \(P\)) I O) -> (evalo (cdr \(P\)) I O) -> (evalo (cons \(P\)) I O) -> (evalo (var \(P\)) I O) ...

As we choose branches of DISJ to expand, we search through possible programs \(\lambda^c\).

Our Approach: Neural Guide

Build a machine learning agent to choose branches of DISJ to expand, taking constraints as inputs.

Experimental Results

We report on two sets of results, with both experiments using the same trained weights.

- Test Problems Solved (%): held-out, dynamically-typed improper list construction problems.
- \textbf{Generalization}: Largest \(N\) for which synthesis of a family of programs succeeded.

Method & Test Problems Solved (%)

<table>
<thead>
<tr>
<th>Method</th>
<th>Test Problems Solved (%)</th>
<th>Repeat(N)</th>
<th>DropLast(N)</th>
<th>BringToFront(N)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive</td>
<td>27%</td>
<td>6 (time)</td>
<td>2 (time)</td>
<td>- (time)</td>
</tr>
<tr>
<td>+Heuristics</td>
<td>82%</td>
<td>11 (time)</td>
<td>3 (time)</td>
<td>- (time)</td>
</tr>
<tr>
<td>RNN (No Constraints)</td>
<td>93%</td>
<td>9 (time)</td>
<td>3 (time)</td>
<td>6 (time)</td>
</tr>
<tr>
<td>GNN + Constraints</td>
<td>88%</td>
<td>20+</td>
<td>6 (time)</td>
<td>5 (time)</td>
</tr>
<tr>
<td>RNN + Constraints</td>
<td>99%</td>
<td>20+</td>
<td>6 (time)</td>
<td>5 (time)</td>
</tr>
<tr>
<td>(\lambda^c)</td>
<td>4 (memory)</td>
<td>3 (error)</td>
<td>3 (error)</td>
<td>- (error)</td>
</tr>
<tr>
<td>Escher</td>
<td>80%</td>
<td>10 (error)</td>
<td>1 (oracle)</td>
<td>- (oracle)</td>
</tr>
<tr>
<td>Myth</td>
<td>90%</td>
<td>20+</td>
<td>- (error)</td>
<td>- (error)</td>
</tr>
<tr>
<td>RobustFill beam 5000</td>
<td>100%</td>
<td>3</td>
<td>1</td>
<td>- (error)</td>
</tr>
</tbody>
</table>

Discussion & Future Work

- RNN with constraints performed almost perfectly in test problems.
- RNN / GNN with constraints has the potential to scale to larger programs.
- Thus far we have used a small subset of Lisp, without recursion. We would like to expand to synthesize programs in larger subsets of the Lisp language, and recursive programs.

Model Choices

- RNN + Constraints computes constraint embeddings using LSTMs, treating constraints as sequences.
- GNN + Constraints computes constraint embeddings using a Graph Neural Network (GNN), treating constraints as graphs.
- RNN (\textbf{No Constraints}) scores candidate programs directly by embedding the candidate program, input sequence, and output sequence using LSTMs.

Training the models:

- Autogenerate training problems: generate a program, then generate input/output examples for the program. We use miniKanren to do this.
- Since we know a ground truth program during training, we know which candidate program is correct at each step.
- Expand 2 partial programs per step during training.

Why use constraints?

- Evaluating whether a partial program is plausible should be easier than generating a program.
- ML Agent essentially learns a flavour of constraint satisfaction.
- Constraints contain relevant portions of the input/output, acting as an attentional mechanism.
- Constraints are roughly the same length, whereas programs can be long, so we should be able to scale to larger programs by using constraints.