Neural Guided Constraint Logic Programming for Program Synthesis

Lisa Zhang[1,2], Gregory Rosenblatt[4], Ethan Fetaya[1,2], Renjie Liao[1,2,3], William E. Byrd[4], Matthew Might[4], Raquel Urtasun[1,2,3], Richard Zemel[1,2]

[1]University of Toronto, [2]Vector Institute, [3]Uber ATG, [4]University of Alabama at Birmingham

Programing By Example

Goal: synthesize program specified in terms of input/output examples.
Approaches: enumerative type-based search methods like λ^2, Myth, Escher; Machine Learning work uses methods like conditional program generation, differentiable programming, and neural guided synthesis.
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.

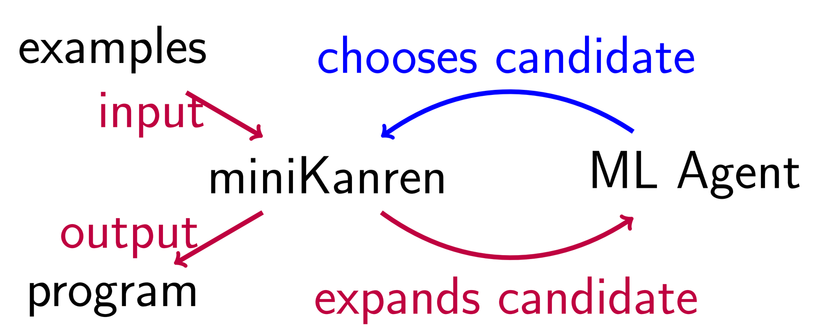
- miniKanren is flexible: can synthesize dynamically typed recursive programs
- write a **relational interpreter** in miniKanren: a relational form `(evalo P I O)` of interpreter `(eval P I) = O`
- relations like *evalo* can be thought of as constraints
- query miniKanren to find solutions to P in `(evalo P I O)` by iteratively expanding relation evalo with its definition:

```
(evalo P I O)
  ⇒ DISJ →(evalo (quote A) I O)
         →(evalo (car B) I O)
         →(evalo (cdr C) I O)
         →(evalo (cons D E) I O)
         →(evalo (var F) I O)
          ...
```

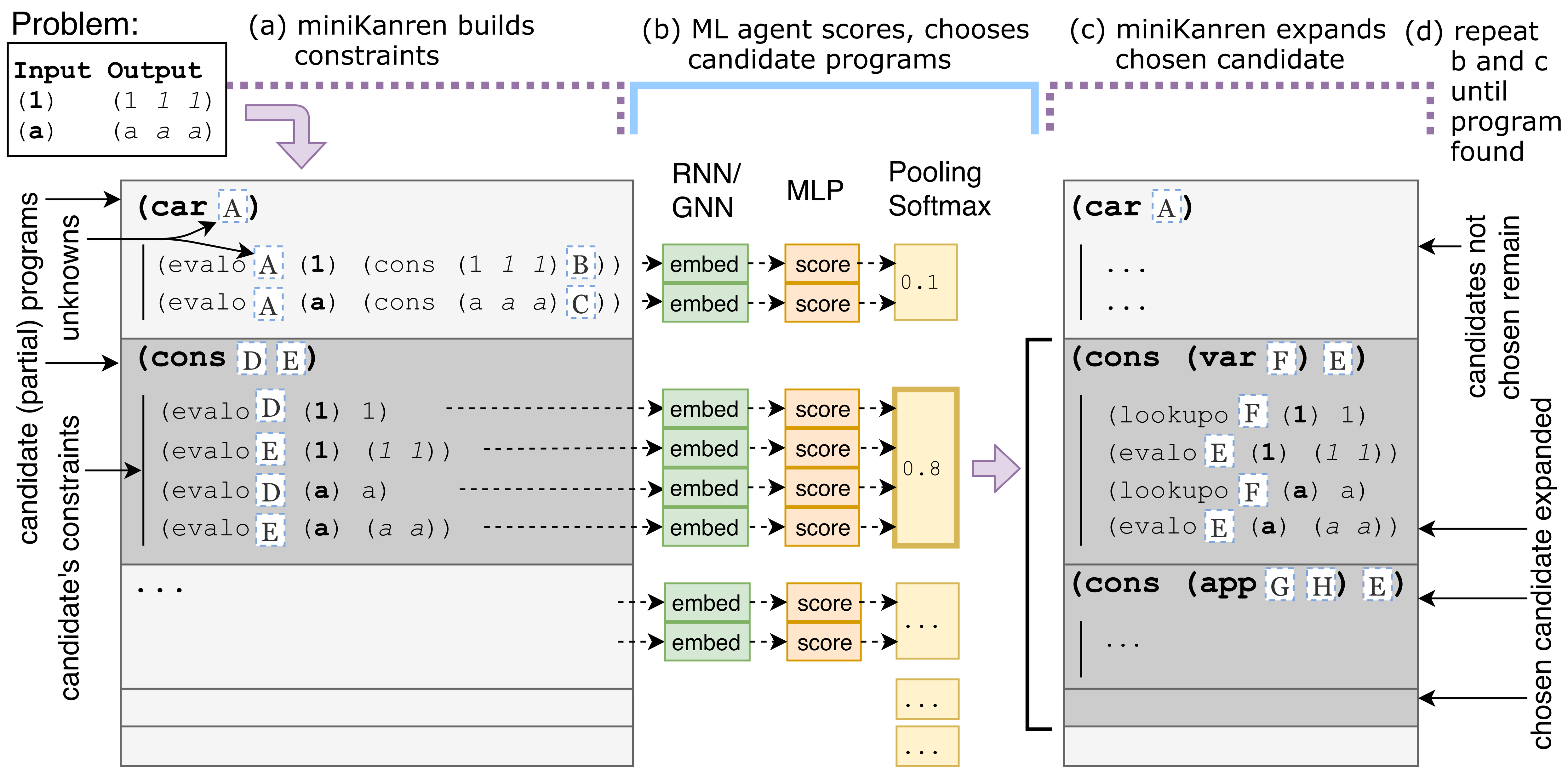
- As we choose branches of DISJ to expand, we search through possible programs P.

Our Approach: Neural Guide

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



Synthesis Steps



(a) miniKanren builds constraints representing the PBE problem; candidate programs contain unknowns, whose values are restricted by constraints: in the second candidate, the `evalo` constraints decompose the output into two portions to be synthesized independently
(b) a neural network operating on the constraints scores candidates; each constraint is embedded and scored separately, then pooled per candidate; scores determine which candidate to expand
(c) miniKanren expands the chosen candidate `(cons D E)`, so that different completions of unknown D are added to the set of candidates
(d) this process continues until a fully-specified program (with no logic variables) is found

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.
- **Generalization**: Largest N for which synthesis of a family of programs succeeded.

Method	Test Problems Solved (%)	Generalization		
		Repeat(N)	DropLast(N)	BringToFront(N)
Naive	27%	6 (time)	2 (time)	- (time)
+Heuristics	82%	11 (time)	3 (time)	- (time)
RNN (No Constraints)	93%	9 (time)	3 (time)	2 (time)
GNN + Constraints	88%	**20+**	**6** (time)	**6** (time)
RNN + Constraints	99%	**20+**	**6** (time)	5 (time)
λ^2		4 (memory)	3 (error)	3 (error)
Escher		10 (error)	1 (oracle)	- (oracle)
Myth		**20+**	- (error)	- (error)
RobustFill beam 5000	**100%**	3	1	- (error)

- Repeat(N): repeat a token N times
- DropLast(N): drop the last element in an N element list
- BringToFront(N): bring the last element to the front in an N element list
- Failure modes: out of **time**, out of **memory**, requires **oracle**, other **error**

Model Choices

We test different models for scoring candidates:

- **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 (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.

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.

Research reported in this publication was supported in part by the Natural Sciences and Engineering Research Council of Canada, and the National Center For Advancing Translational Sciences of the National Institutes of Health under Award Number OT2TR002517. R.L. was supported by Connaught International Scholarship. The content is solely the responsibility of the authors and does not necessarily represent the official views of the funding agencies.