Appendix: Mapping natural-language problems to formal-language solutions using structured neural representations

Kezhen Chen [1 2] Qiuyuan Huang [1] Hamid Palangi [1] Paul Smolensky [1 3] Kenneth D. Forbus [2] Jianfeng Gao [1]

1. Implementations of TP-N2F for experiments

In this section, we present details of the experiments of TP-N2F on the two datasets. We present the implementation of TP-N2F on each dataset.

The MathQA dataset consists of about 37k math word problems ((80/12/8)% training/dev/testing problems), each with a corresponding list of multi-choice options and an straight-line operation sequence program to solve the problem. An example from the dataset is presented in the Appendix A.4. In this task, TP-N2F is deployed to generate the operation sequence given the question. The generated operations are executed to generate the solution for the given math problem. We use the execution script from (Amini et al., 2019) to execute the generated operation sequence and compute the multi-choice accuracy for each problem. During our experiments we observed that there are about 30% noisy examples (on which the execution script fails to get the correct answer on the ground truth program). Therefore, we report both execution accuracy (the final multi-choice answer after running the execution engine) and operation sequence accuracy (where the generated operation sequence must match the ground truth sequence exactly).

The AlgoLisp dataset (Polosukhin & Skidanov, 2018) is a program synthesis dataset, which has 79k/9k/10k training/dev/testing samples. Each sample contains a problem description, a corresponding Lisp program tree, and 10 input-output testing pairs. We parse the program tree into a straight-line sequence of commands from leaves to root and (as in MathQA) use the symbol $\#_i$ to indicate the result of the i^{th} command (generated previously by the model). A dataset sample with our parsed command sequence is presented in the Appendix A.4. AlgoLisp provides an execution script to run the generated program and has three evaluation metrics: accuracy of passing all test cases (Acc), accuracy of passing 50% of test cases (50p-Acc), and accuracy of generating an exactly matched program (M-Acc). AlgoLisp has about 10% noise data (where the execution script fails to pass all test cases on the ground truth program), so we report results both on the full test set and the cleaned test set (in which all noisy testing samples are removed).

We use $d_{\mathrm{R}}, n_{\mathrm{R}}, d_{\mathrm{F}}, n_{\mathrm{F}}$ to indicate the TP-N2F encoder hyperparameters, the dimension of role vectors, the number of roles, the dimension of filler vectors and the number of fillers. $d_{Rel}, d_{Arg}, d_{Pos}$ indicate the TP-N2F decoder hyperparameters, the dimension of relation vectors, the dimension of argument vectors, and the dimension of position vectors.

In the experiment on the MathQA dataset, we use $n_{\mathrm{F}} = 150$, $n_{\mathrm{R}} = 50$, $d_{\mathrm{F}} = 30$, $d_{\mathrm{R}} = 20$, $d_{Rel} = 20$, $d_{Arg} = 10$, $d_{Pos} = 5$ and we train the model for 60 epochs with learning rate 0.00115. The reasoning module only contains one layer. As most of the math operators in this dataset are binary, we replace all operators taking three arguments with a set of binary operators based on hand-encoded rules, and for all operators taking one argument, a padding symbol is appended. For the baseline SEQ2PROG-orig, TP2LSTM and LSTM2TP, we use hidden size 100, single-direction, one-layer LSTM. For the SEQ2PROG-best, we performed a hyperparameter search on the hidden size for both encoder and decoder; the best score is reported.

In the experiment on the AlgoLisp dataset, we use $n_{\mathrm{F}} = 150$, $n_{\mathrm{R}} = 50$, $d_{\mathrm{F}} = 30$, $d_{\mathrm{R}} = 30$, $d_{Rel} = 30$, $d_{Arg} = 20$, $d_{Pos} = 5$ and we train the model for 50 epochs with learning rate 0.00115. We also use one-layer in the reasoning module like in MathQA. For this dataset, most function calls take three arguments so we simply add padding symbols for those functions with fewer than three arguments.

2. Analysis from ablation studies

We performed some ablation studies. The explanation studies and findings are discussed here. As TP-N2F model usually needs more parameters for TPRs, we tested the baseline LSTM2LSTM+attention model with similar number of parameters (increasing the hidden size in the encoder and decoder). We found that the performance of baseline model decreased when they had similar degree of parameters. We also tested different number of layers of the reasoning MLP. Each layer of the MLP is a linear layer following Tanh activation function. From ablation studies, the performance with 1, 2 and 3 layers were similar. As the number of layers increase, the performance reduced. Finally, we tested using the tensor product of last hidden states of Role-LSTM and

Filler-LSTM instead of the sum of all tensor products. Experiments showed that using tensor product sums had better performance than using last hidden states.

3. Detailed equations of TP-N2F

3.1. TP-N2F encoder

Filler-LSTM in TP-N2F encoder

This is a standard LSTM, governed by the equations:

$$\boldsymbol{f}_{\mathrm{f}}^{t} = \varphi(\boldsymbol{U}_{\mathrm{ff}}\, \boldsymbol{w}^{t} + \boldsymbol{V}_{\mathrm{ff}}\, \flat(\mathbf{T}^{t-1}) + \boldsymbol{b}_{\mathrm{ff}}) \tag{1}$$

$$\boldsymbol{g}_{\mathrm{f}}^{t} = \tanh(\boldsymbol{U}_{\mathrm{fg}}\, \boldsymbol{w}^{t} + \boldsymbol{V}_{\mathrm{fg}}\, \flat(\mathbf{T}^{t-1}) + \boldsymbol{b}_{\mathrm{fg}}) \tag{2}$$

$$\boldsymbol{i}_{\mathrm{f}}^{t} = \varphi(\boldsymbol{U}_{\mathrm{fi}}\, \boldsymbol{w}^{t} + \boldsymbol{V}_{\mathrm{fi}}\, \flat(\mathbf{T}^{t-1}) + \boldsymbol{b}_{\mathrm{fi}}) \tag{3}$$

$$\boldsymbol{o}_{\mathrm{f}}^{t} = \varphi(\boldsymbol{U}_{\mathrm{fo}}\, \boldsymbol{w}^{t} + \boldsymbol{V}_{\mathrm{fo}}\, \flat(\mathbf{T}^{t-1}) + \boldsymbol{b}_{\mathrm{fo}}) \tag{4}$$

$$\boldsymbol{c}_{\mathrm{f}}^{t} = \boldsymbol{f}_{\mathrm{f}}^{t} \odot \boldsymbol{c}_{\mathrm{f}}^{t-1} + \boldsymbol{i}_{\mathrm{f}}^{t} \odot \boldsymbol{g}_{\mathrm{f}}^{t} \tag{5}$$

$$\boldsymbol{h}_{\mathrm{f}}^{t} = \boldsymbol{o}_{\mathrm{f}}^{t} \odot \tanh(\boldsymbol{c}_{f}^{t}) \tag{6}$$

$\varphi, \tanh$ are the logistic sigmoid and tanh functions applied elementwise. $\flat$ flattens (reshapes) a matrix in $\mathbb{R}^{d_{\mathrm{F}} \times d_{\mathrm{R}}}$ into a vector in $\mathbb{R}^{d_{\mathrm{T}}}$, where $d_{\mathrm{T}} = d_{\mathrm{F}} d_{\mathrm{R}}$. $\odot$ is elementwise multiplication. The variables have the following dimensions:

$$\begin{aligned}
&\boldsymbol{f}_{\mathrm{f}}^{t}, \boldsymbol{g}_{\mathrm{f}}^{t}, \boldsymbol{i}_{\mathrm{f}}^{t}, \boldsymbol{o}_{\mathrm{f}}^{t}, \boldsymbol{c}_{\mathrm{f}}^{t}, \boldsymbol{h}_{\mathrm{f}}^{t}, \boldsymbol{b}_{\mathrm{ff}}, \boldsymbol{b}_{\mathrm{fg}}, \boldsymbol{b}_{\mathrm{fi}}, \boldsymbol{b}_{\mathrm{fo}}, \flat(\mathbf{T}^{t-1}) \in \mathbb{R}^{d_{\mathrm{T}}} \\
&w^{t} \in \mathbb{R}^{d} \\
&\boldsymbol{U}_{\mathrm{ff}}, \boldsymbol{U}_{\mathrm{fg}}, \boldsymbol{U}_{\mathrm{fi}}, \boldsymbol{U}_{\mathrm{fo}} \in \mathbb{R}^{d_{\mathrm{T}} \times d} \\
&\boldsymbol{V}_{\mathrm{ff}}, \boldsymbol{V}_{\mathrm{fg}}, \boldsymbol{V}_{\mathrm{fi}}, \boldsymbol{V}_{\mathrm{fo}} \in \mathbb{R}^{d_{\mathrm{T}} \times d_{\mathrm{T}}}
\end{aligned}$$

Filler vector

The filler vector for input token w^t is $\boldsymbol{f}^t$, defined through an attention vector over possible fillers, $\boldsymbol{a}_{\mathrm{f}}^{t}$:

$$\boldsymbol{a}_{\mathrm{f}}^{t} = \mathrm{softmax}((\boldsymbol{W}_{\mathrm{fa}}\, \boldsymbol{h}_{\mathrm{f}}^{t})/T) \tag{7}$$

$$\boldsymbol{f}^{t} = \boldsymbol{W}_{\mathrm{f}}\, \boldsymbol{a}_{\mathrm{f}}^{t} \tag{8}$$

(W_{f} is the same as $\boldsymbol{F}$ of Sec.2 in the paper) The variables' dimensions are:

$$\begin{aligned}
&\boldsymbol{W}_{\mathrm{fa}} \in \mathbb{R}^{n_{\mathrm{F}} \times d_{\mathrm{T}}} \\
&\boldsymbol{a}_{\mathrm{f}}^{t} \in \mathbb{R}^{n_{\mathrm{F}}} \\
&\boldsymbol{W}_{\mathrm{f}} \in \mathbb{R}^{d_{\mathrm{F}} \times n_{\mathrm{F}}} \\
&\boldsymbol{f}^{t} \in \mathbb{R}^{d_{\mathrm{F}}}
\end{aligned}$$

T is the temperature factor, which is fixed at 0.1.

Role-LSTM in TP-N2F encoder

Similar to the Filler-LSTM, the Role-LSTM is also a standard LSTM, governed by the equations:

$$\boldsymbol{f}_{\mathrm{r}}^{t} = \varphi(\boldsymbol{U}_{\mathrm{rf}}\, \boldsymbol{w}^{t} + \boldsymbol{V}_{\mathrm{rf}}\, \flat(\mathbf{T}^{t-1}) + \boldsymbol{b}_{\mathrm{rf}}) \tag{9}$$

$$\boldsymbol{g}_{\mathrm{r}}^{t} = \tanh(\boldsymbol{U}_{\mathrm{rg}}\, \boldsymbol{w}^{t} + \boldsymbol{V}_{\mathrm{rg}}\, \flat(\mathbf{T}^{t-1}) + \boldsymbol{b}_{\mathrm{rg}}) \tag{10}$$

$$\boldsymbol{i}_{\mathrm{r}}^{t} = \varphi(\boldsymbol{U}_{\mathrm{ri}}\, \boldsymbol{w}^{t} + \boldsymbol{V}_{\mathrm{ri}}\, \flat(\mathbf{T}^{t-1}) + \boldsymbol{b}_{\mathrm{ri}}) \tag{11}$$

$$\boldsymbol{o}_{\mathrm{r}}^{t} = \varphi(\boldsymbol{U}_{\mathrm{ro}}\, \boldsymbol{w}^{t} + \boldsymbol{V}_{\mathrm{ro}}\, \flat(\mathbf{T}^{t-1}) + \boldsymbol{b}_{\mathrm{ro}}) \tag{12}$$

$$\boldsymbol{c}_{\mathrm{r}}^{t} = \boldsymbol{f}_{\mathrm{r}}^{t} \odot \boldsymbol{c}_{\mathrm{r}}^{t-1} + \boldsymbol{i}_{\mathrm{r}}^{t} \odot \boldsymbol{g}_{\mathrm{r}}^{t} \tag{13}$$

$$\boldsymbol{h}_{\mathrm{r}}^{t} = \boldsymbol{o}_{\mathrm{r}}^{t} \odot \tanh(\boldsymbol{c}_{\mathrm{r}}^{t}) \tag{14}$$

The variable dimensions are:

$$\begin{aligned}
&\boldsymbol{f}_{\mathrm{r}}^{t}, \boldsymbol{g}_{\mathrm{r}}^{t}, \boldsymbol{i}_{\mathrm{r}}^{t}, \boldsymbol{o}_{\mathrm{r}}^{t}, \boldsymbol{c}_{\mathrm{r}}^{t}, \boldsymbol{h}_{\mathrm{r}}^{t}, \boldsymbol{b}_{\mathrm{rf}}, \boldsymbol{b}_{\mathrm{rg}}, \boldsymbol{b}_{\mathrm{ri}}, \boldsymbol{b}_{\mathrm{ro}}, \flat(\mathbf{T}^{t-1}) \in \mathbb{R}^{d_{\mathrm{T}}} \\
&w^{t} \in \mathbb{R}^{d} \\
&\boldsymbol{U}_{\mathrm{rf}}, \boldsymbol{U}_{\mathrm{rg}}, \boldsymbol{U}_{\mathrm{ri}}, \boldsymbol{U}_{\mathrm{ro}} \in \mathbb{R}^{d_{\mathrm{T}} \times d} \\
&\boldsymbol{V}_{\mathrm{rf}}, \boldsymbol{V}_{\mathrm{rg}}, \boldsymbol{V}_{\mathrm{ri}}, \boldsymbol{V}_{\mathrm{ro}} \in \mathbb{R}^{d_{\mathrm{T}} \times d_{\mathrm{T}}}
\end{aligned}$$

Role vector

The role vector for input token w^t is determined analogously to its filler vector:

$$\boldsymbol{a}_{\mathrm{r}}^{t} = \mathrm{softmax}((\boldsymbol{W}_{\mathrm{ra}}\, \boldsymbol{h}_{\mathrm{r}}^{t})/T) \tag{15}$$

$$\boldsymbol{r}^{t} = \boldsymbol{W}_{\mathrm{r}}\, \boldsymbol{a}_{\mathrm{r}}^{t} \tag{16}$$

The dimensions are:

$$\begin{aligned}
&\boldsymbol{W}_{\mathrm{ra}} \in \mathbb{R}^{n_{\mathrm{R}} \times d_{\mathrm{T}}} \\
&\boldsymbol{a}_{\mathrm{r}}^{t} \in \mathbb{R}^{n_{\mathrm{R}}} \\
&\boldsymbol{W}_{\mathrm{r}} \in \mathbb{R}^{d_{\mathrm{R}} \times n_{\mathrm{R}}} \\
&\boldsymbol{r}^{t} \in \mathbb{R}^{d_{\mathrm{R}}}
\end{aligned}$$

Binding

The TPR for the filler/role binding for token w^t is then:

$$\boldsymbol{T}_{t} = \boldsymbol{r}^{t} \otimes \boldsymbol{f}^{t} \tag{17}$$

where

$$\boldsymbol{T}^{t} \in \mathbb{R}^{d_{\mathrm{R}} \times d_{\mathrm{F}}}$$

3.2. Structure Mapping

$$\mathbf{H}^{0} = f_{\mathrm{mapping}}(\boldsymbol{T}_{t}) \tag{18}$$

$\mathbf{H}^{0} \in \mathbb{R}^{d_{\mathrm{H}}}$, where $d_{\mathrm{H}} = d_{\mathrm{A}}, d_{\mathrm{O}}, d_{\mathrm{P}}$ are dimension of argument vector, operator vector and position vector. f_{mapping} is implemented with a MLP (linear layer followed by a tanh) for mapping the $\boldsymbol{T}_{t} \in \mathbb{R}^{d_{\mathrm{T}}}$ to the initial state of decoder $\mathbf{H}^{0}$.

3.3. TP-N2F decoder

Tuple-LSTM

The output tuples are also generated via a standard LSTM:

$$\boldsymbol{w}_d^t = \gamma(\boldsymbol{w}_{Rel}^{t-1}, \boldsymbol{w}_{Arg1}^{t-1}, \boldsymbol{w}_{Arg2}^{t-1}) \tag{19}$$

$$\boldsymbol{f}^t = \varphi(\boldsymbol{U}_\mathrm{f}\, \boldsymbol{w}_d^t + \boldsymbol{V}_\mathrm{f}\, \flat(\mathbf{H}^{t-1}) + \boldsymbol{b}_\mathrm{f}) \tag{20}$$

$$\boldsymbol{g}^t = \tanh(\boldsymbol{U}_\mathrm{g}\, \boldsymbol{w}_d^t + \boldsymbol{V}_\mathrm{g}\, \flat(\mathbf{H}^{t-1}) + \boldsymbol{b}_\mathrm{g}) \tag{21}$$

$$\boldsymbol{i}^t = \varphi(\boldsymbol{U}_\mathrm{i}\, \boldsymbol{w}_d^t + \boldsymbol{V}_\mathrm{i}\, \flat(\mathbf{H}^{t-1}) + \boldsymbol{b}_\mathrm{i}) \tag{22}$$

$$\boldsymbol{o}^t = \varphi(\boldsymbol{U}_\mathrm{o}\, \boldsymbol{w}_d^t + \boldsymbol{V}_\mathrm{o}\, \flat(\mathbf{H}^{t-1}) + \boldsymbol{b}_\mathrm{o}) \tag{23}$$

$$\boldsymbol{c}^t = \boldsymbol{f}^t \odot \boldsymbol{c}^{t-1} + \boldsymbol{i}^t \odot \boldsymbol{g}^t \tag{24}$$

$$\boldsymbol{h}_\mathrm{input}^t = \boldsymbol{o}^t \odot \tanh(\boldsymbol{c}^t) \tag{25}$$

$$\mathbf{H}^t = \mathrm{Atten}(\boldsymbol{h}_\mathrm{input}^t, [\boldsymbol{T}_0, ..., \boldsymbol{T}_{n-1}]) \tag{26}$$

Here, γ is the concatenation function. $\boldsymbol{w}_{Rel}^{t-1}$ is the trained embedding vector for the Relation of the input binary tuple, $\boldsymbol{w}_{Arg1}^{t-1}$ is the embedding vector for the first argument and $\boldsymbol{w}_{Arg2}^{t-1}$ is the embedding vector for the second argument. Then the input for the Tuple LSTM is the concatenation of the embedding vectors of relation and arguments, with dimension d_dec.

$$\boldsymbol{f}^t, \boldsymbol{g}^t, \boldsymbol{i}^t, \boldsymbol{o}^t, \boldsymbol{c}^t, \boldsymbol{h}_\mathrm{input}^t, \boldsymbol{b}_\mathrm{f}, \boldsymbol{b}_\mathrm{g}, \boldsymbol{b}_\mathrm{i}, \boldsymbol{b}_\mathrm{o}, \flat(\mathbf{H}^{t-1}) \in \mathbb{R}^{d_\mathrm{H}}$$
$$\boldsymbol{w}_d^t \in \mathbb{R}^{d_\mathrm{dec}}$$
$$\boldsymbol{U}_\mathrm{f}, \boldsymbol{U}_\mathrm{g}, \boldsymbol{U}_\mathrm{i}, \boldsymbol{U}_\mathrm{o} \in \mathbb{R}^{d_\mathrm{H} \times d_\mathrm{dec}}$$
$$\boldsymbol{V}_\mathrm{f}, \boldsymbol{V}_\mathrm{g}, \boldsymbol{V}_\mathrm{i}, \boldsymbol{V}_\mathrm{o} \in \mathbb{R}^{d_\mathrm{H} \times d_\mathrm{H}}$$
$$\mathbf{H}^t \in \mathbb{R}^{d_\mathrm{H}}$$

Atten is the attention mechanism used in Luong et al. (2015), which computes the dot product between $\boldsymbol{h}_\mathrm{input}^t$ and each $\boldsymbol{T}_{t'}$. Then a linear function is used on the concatenation of $\boldsymbol{h}_\mathrm{input}^t$ and the softmax scores on all dot products to generate $\mathbf{H}^t$. The following equations show the attention mechanism:

$$\boldsymbol{d}^t = \mathrm{score}(\boldsymbol{h}_\mathrm{input}^t, \mathbf{C}_T) \tag{27}$$

$$\boldsymbol{s}^t = \mathbf{C}_T\, \mathrm{softmax}(\boldsymbol{d}^t) \tag{28}$$

$$\mathbf{H}^t = \boldsymbol{K}\gamma(\boldsymbol{h}_\mathrm{input}^t, \boldsymbol{s}^t) \tag{29}$$

score is the score function of the attention. In this paper, the score function is dot product.

$$\mathbf{C}_T \in \mathbb{R}^{d_\mathrm{H} \times n}$$
$$\boldsymbol{d}_t \in \mathbb{R}^n$$
$$\boldsymbol{s}_t \in \mathbb{R}^{d_\mathrm{H}}$$
$$\boldsymbol{K} \in \mathbb{R}^{d_\mathrm{H} \times (d_\mathrm{T} + n)}$$

Unbinding

At each timestep t, the 2-step unbinding process described in Sec.3.1.2 of the paper operates first on an encoding of the triple as a whole, $\mathbf{H}$, using two unbinding vectors $\boldsymbol{p}_i'$ that are learned but fixed for all tuples. This first unbinding gives an encoding of the two operator-argument bindings, $\mathbf{B}_i$. The second unbinding operates on the $\mathbf{B}_i$, using a generated unbinding vector for the operator, $\boldsymbol{r}_{rel}'$, giving encodings of the arguments, $\boldsymbol{a}_i$. The generated unbinding vector for the operator, $\boldsymbol{r}'$, and the generated encodings of the arguments, $\boldsymbol{a}_i$, each produce a probability distribution over symbolic operator outputs Rel and symbolic argument outputs Arg_i; these probabilities are used in the cross-entropy loss function. For generating a single symbolic output, the most-probable symbols are selected.

$$\mathbf{B}_1^t = \mathbf{H}^t\, \boldsymbol{p}_1' \tag{30}$$

$$\mathbf{B}_2^t = \mathbf{H}^t\, \boldsymbol{p}_2' \tag{31}$$

$$\boldsymbol{r}_{rel}'^t = \boldsymbol{W}_\mathrm{dual}\,(B_1^t + B_2^t) \tag{32}$$

$$\boldsymbol{a}_1^t = \mathbf{B}_1^t\, \boldsymbol{r}_{rel}'^t \tag{33}$$

$$\boldsymbol{a}_2^t = \mathbf{B}_2^t\, \boldsymbol{r}_{rel}'^t \tag{34}$$

$$\boldsymbol{l}_{r_{rel}}^t = \mathbf{L}_{r_{rel}}^t\, \boldsymbol{r}_{rel}'^t \tag{35}$$

$$\boldsymbol{l}_{a_1}^t = \mathbf{L}_a^t\, \boldsymbol{a}_1^t \tag{36}$$

$$\boldsymbol{l}_{a_2}^t = \mathbf{L}_a^t\, \boldsymbol{a}_2^t \tag{37}$$

$$Rel^t = \mathrm{argmax}(\mathrm{softmax}(\boldsymbol{l}_r^t)) \tag{38}$$

$$Arg1^t = \mathrm{argmax}(\mathrm{softmax}(\boldsymbol{l}_{a_1}^t)) \tag{39}$$

$$Arg2^t = \mathrm{argmax}(\mathrm{softmax}(\boldsymbol{l}_{a_2}^t)) \tag{40}$$

The dimensions are:

$$\boldsymbol{r}_{rel}'^t \in \mathbb{R}^{d_\mathrm{O}}$$
$$\boldsymbol{a}_1^t, \boldsymbol{a}_2^t \in \mathbb{R}^{d_\mathrm{A}}$$
$$\boldsymbol{p}_1', \boldsymbol{p}_2' \in \mathbb{R}^{d_\mathrm{P}}$$
$$\mathbf{B}_1^t, \mathbf{B}_2^t \in \mathbb{R}^{d_\mathrm{A} \times d_\mathrm{O}}$$
$$\boldsymbol{W}_\mathrm{dual} \in \mathbb{R}^{d_\mathrm{H}}$$
$$\boldsymbol{L}_r^t \in \mathbb{R}^{n_\mathrm{O} \times d_\mathrm{O}}$$
$$\mathbf{L}_a^t \in \mathbb{R}^{n_\mathrm{A} \times d_\mathrm{A}}$$
$$\boldsymbol{l}_r^t \in \mathbb{R}^{n_\mathrm{R}}$$
$$\boldsymbol{l}_{a_1}^t, \boldsymbol{l}_{a_2}^t \in \mathbb{R}^{n_\mathrm{A}}$$

4. Dataset samples

4.0.1. Data sample from MathQA dataset

Problem: The present polulation of a town is 3888. Population increase rate is 20%. Find the population of town after 1 year?
Options: a) 2500, b) 2100, c) 3500, d) 3600, e) 2700
Operations: multiply(n0,n1), divide(#0,const-100), add(n0,#1)

4.0.2. Data sample from AlgoLisp dataset

Problem: Consider an array of numbers and a number, decrements each element in the given array by the given number, what is the given array?
Program Nested List: (map a (partial1 b –))
Command-Sequence: (partial1 b –), (map a #0)

5. Generated programs comparison

In this section, we display some generated samples from the two datasets, where the TP-N2F model generates correct programs but LSTM-Seq2Seq does not.

Question: A train running at the speed of 50 km per hour crosses a post in 4 seconds. What is the length of the train?
TP-N2F(correct):
(multiply,n0,const1000) (divide,#0,const3600) (multiply,n1,#1)
LSTM(wrong):
(multiply,n0,const0.2778) (multiply,n1,#0)

Question: 20 is subtracted from 60 percent of a number, the result is 88. Find the number?
TP-N2F(correct):
(add,n0,n2) (divide,n1,const100) (divide,#0,#1)
LSTM(wrong):
(add,n0,n2) (divide,n1,const100) (divide,#0,#1) (multiply,#2,n3) (subtract,#3,n0)

Question: The population of a village is 14300. It increases annually at the rate of 15 percent. What will be its population after 2 years?
TP-N2F(correct):
(divide,n1,const100) (add,#0,const1) (power,#1,n2) (multiply,n0,#2)
LSTM(wrong):
(multiply,const4,const100) (sqrt,#0)

Question: There are two groups of students in the sixth grade. There are 45 students in group a, and 55 students in group b. If, on a particular day, 20 percent of the students in group a forget their homework, and 40 percent of the students in group b forget their homework, then what percentage of the sixth graders forgot their homework?
TP-N2F(correct):
(add,n0,n1) (multiply,n0,n2) (multiply,n1,n3) (divide,#1,const100) (divide,#2,const100) (add,#3,#4) (divide,#5,#0) (multiply,#6,const100)
LSTM(wrong):
(multiply,n0,n1) (subtract,n0,n1) (divide,#0,#1)

Question: 1 divided by 0.05 is equal to
TP-N2F(correct):
(divide,n0,n1)
LSTM(wrong):
(divide,n0,n1) (multiply,n2,#0)

Question: Consider a number a, compute factorial of a
TP-N2F(correct):
(¡=,arg1,1) (-,arg1,1) (self,#1) (*,#2,arg1) (if,#0,1,#3) (lambda1,#4) (invoke1,#5,a)
LSTM(wrong):
(¡=,arg1,1) (-,arg1,1) (self,#1) (*,#2,arg1) (if,#0,1,#3) (lambda1,#4) (len,a) (invoke1,#5,#6)

Question: Given an array of numbers and numbers b and c, add c to elements of the product of elements of the given array and b, what is the product of elements of the given array and b?
TP-N2F(correct):
(partial, b,*) (partial1,c,+) (map,a,#0) (map,#2,#1)
LSTM(wrong):
(partial1,b,+) (partial1,c,+) (map,a,#0) (map,#2,#1)

Question: You are given an array of numbers a and numbers b, c and d , let how many times you can replace the median in a with sum of its digits before it becomes a single digit number and b be the coordinates of one end and c and d be the coordinates of another end of segment e , your task is to find the length of segment e rounded down
TP-N2F(correct):
(digits arg1) (len #0) (== #1 1) (digits arg1) (reduce #3 0 +) (self #4) (+ 1 #5) (if #2 0 #6) (lambda1 #7) (sort a) (len a) (/ #10 2) (deref #9 #11) (invoke1 #8 #12) (- #13 c) (digits arg1) (len #15) (== #16 1) (digits arg1) (reduce #18 0 +) (self #19) (+ 1 #20) (if #17 0 #21) (lambda1 #22) (sort a) (len a) (/ #25 2) (deref #24 #26) (invoke1 #23 #27) (- #28 c) (* #14 #29) (- b d) (- b d) (* #31 #32) (+ #30 #33) (sqrt #34) (floor #35)
LSTM(wrong): (digits arg1) (len #0) (== #1 1) (digits arg1) (reduce #3 0 +) (self #4) (+ 1 #5) (if #2 0 #6) (lambda1 #7) (sort a) (len a) (/ #10 2) (deref #9 #11) (invoke1 #8 #12 c) (- #13) (- b d) (- b d) (* #15 #16) (* #14 #17) (+ #18) (sqrt #19) (floor #20)

Question: Given numbers a , b , c and e , let d be c , reverse digits in d , let a and the number in the range from 1 to b inclusive that has the maximum value when its digits are reversed be the coordinates of one end and d and e be the coordinates of another end of segment f , find the length of segment f squared
TP-N2F(correct):

(digits c) (reverse #0) (* arg1 10) (+ #2 arg2) (lambda2 #3) (reduce #1 0 #4) (- a #5) (digits c) (reverse #7) (* arg1 10) (+ #9 arg2) (lambda2 #10) (reduce #8 0 #11) (- a #12) (* #6 #13) (+ b 1) (range 0 #15) (digits arg1) (reverse #17) (* arg1 10) (+ #19 arg2) (lambda2 #20) (reduce #18 0 #21) (digits arg2) (reverse #23) (* arg1 10) (+ #25 arg2) (lambda2 #26) (reduce #24 0 #27) (¿ #22 #28) (if #29 arg1 arg2) (lambda2 #30) (reduce #16 0 #31) (- #32 e) (+ b 1) (range 0 #34) (digits arg1) (reverse #36) (* arg1 10) (+ #38 arg2) (lambda2 #39) (reduce #37 0 #40) (digits arg2) (reverse #42) (* arg1 10) (+ #44 arg2) (lambda2 #45) (reduce #43 0 #46) (¿ #41 #47) (if #48 arg1 arg2) (lambda2 #49) (reduce #35 0 #50) (- #51 e) (* #33 #52) (+ #14 #53)

LSTM(wrong):

(- a d) (- a d) (* #0 #1) (digits c) (reverse #3) (* arg1 10) (+ #5 arg2) (lambda2 #6) (reduce #4 0 #7) (- #8 e) (+ b 1) (range 0 #10) (digits arg1) (reverse #12) (* arg1 10) (+ #14 arg2) (lambda2 #15) (reduce #13 0 #16) (digits arg2) (reverse #18) (* arg1 10) (+ #20 arg2) (lambda2 #21) (reduce #19 0 #22) (¿ #17 #23) (if #24 arg1 arg2) (lambda2 #25) (reduce #11 0 #26) (- #27 e) (* #9 #28) (+ #2 #29)

6. Analysis of TP-N2F encoder

For TP-N2F encoder, we extract the Softmax scores for fillers and roles of natural-language. We dropped the scores that are less than 0.1 to keep the significant fillers and roles for each word. After analyzing a subset of questions, we find that fillers tend to represent the semantic information and words or phrases with same meaning tend to be assigned the same filler. Roles tend to represent the structured schemes of sentences. For example, in AlgoLisp dataset, ”decrement”, ”difference of” and ”decremented by” are assigned to filler 43. ”increment” and ”add” are assigned to filler 105. In MathQA dataset, ”positive integer”, ”positive number” and ”positive digits” are assigned to filler 27. Figure 1 shows the visualization of fillers for four examples from AlgoLisp dataset. From the figure, ”consider” and ”you are given” are assigned to the filler 146. ”what is” and ”find” are assigned to filler 120. Figure 2 presents the visualization of selected for the four examples. Role 12 indicates the target of the questions needs to be solved and Role 3 indicates the provided information to solve the questions.

7. Analysis of TP-N2F decoder

For TP-N2F decoder, we run K-means clustering on both datasets with $k = 3, 4, 5, 6$ clusters and the results are displayed in Figure 3 and Figure 4. As described before, unbinding-vectors for operators or functions with similar semantics tend to be closer to each other. For example, in the MathQA dataset, arithmetic operators such as *add, subtract, multiply, divide* are clustered together at middle, and operators related to geometry such as *square* or *volume* are clustered together at bottom left. In AlgoLisp dataset, basic arithmetic functions are clustered at middle, and string processing functions are clustered at right.

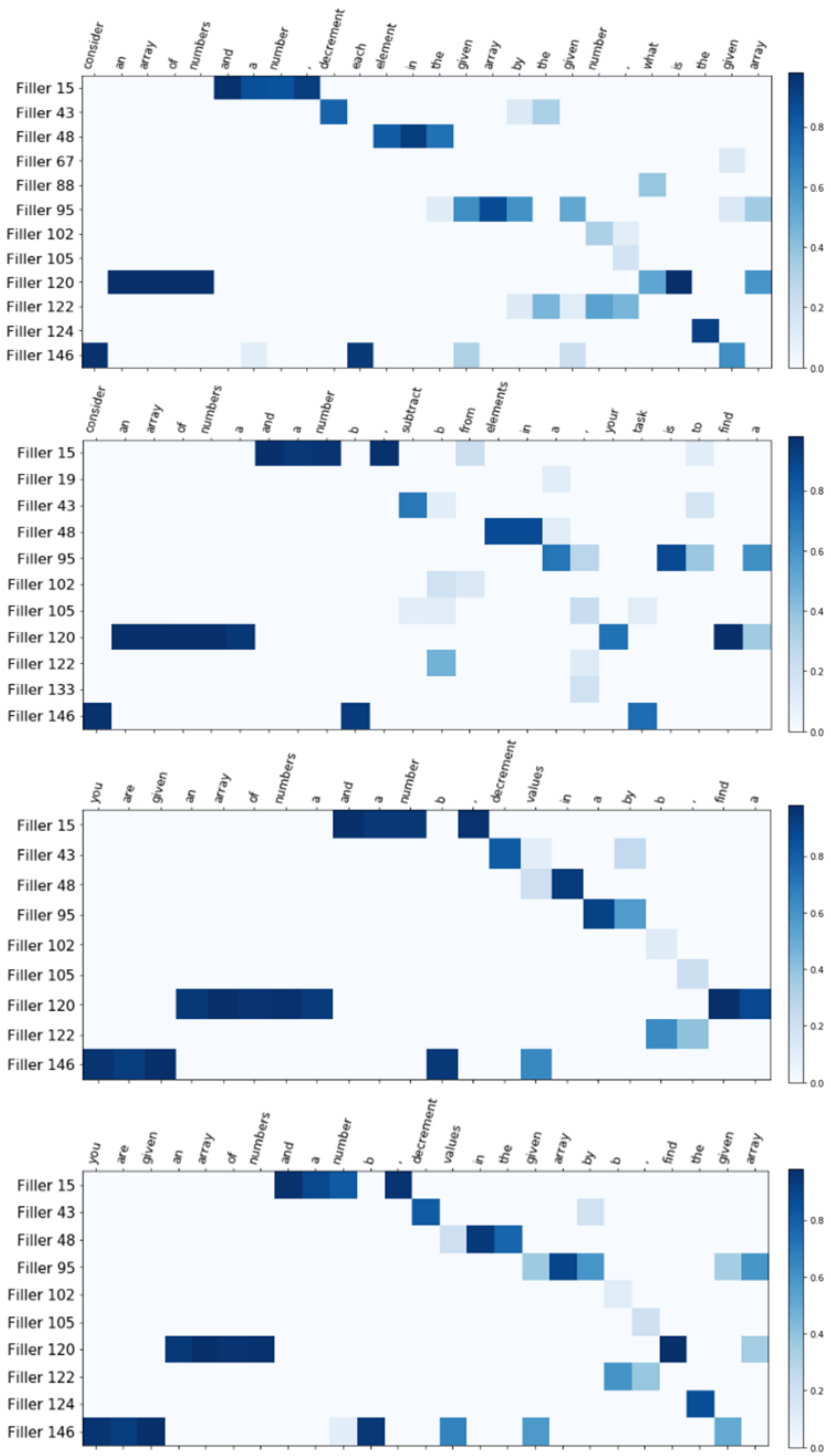

Figure 1. Visualizations of selected fillers for four examples

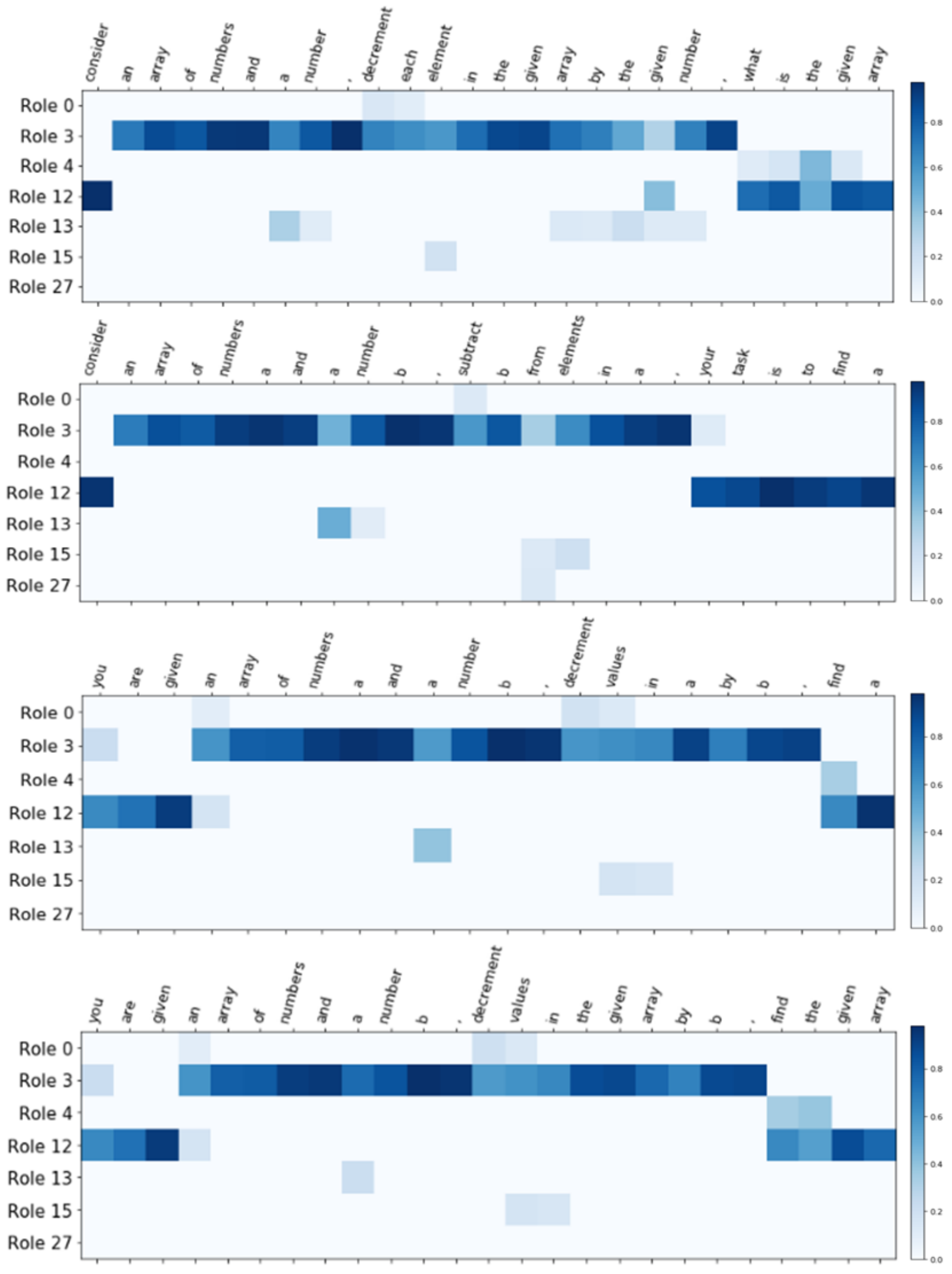

Figure 2. Visualizations of selected roles for four examples

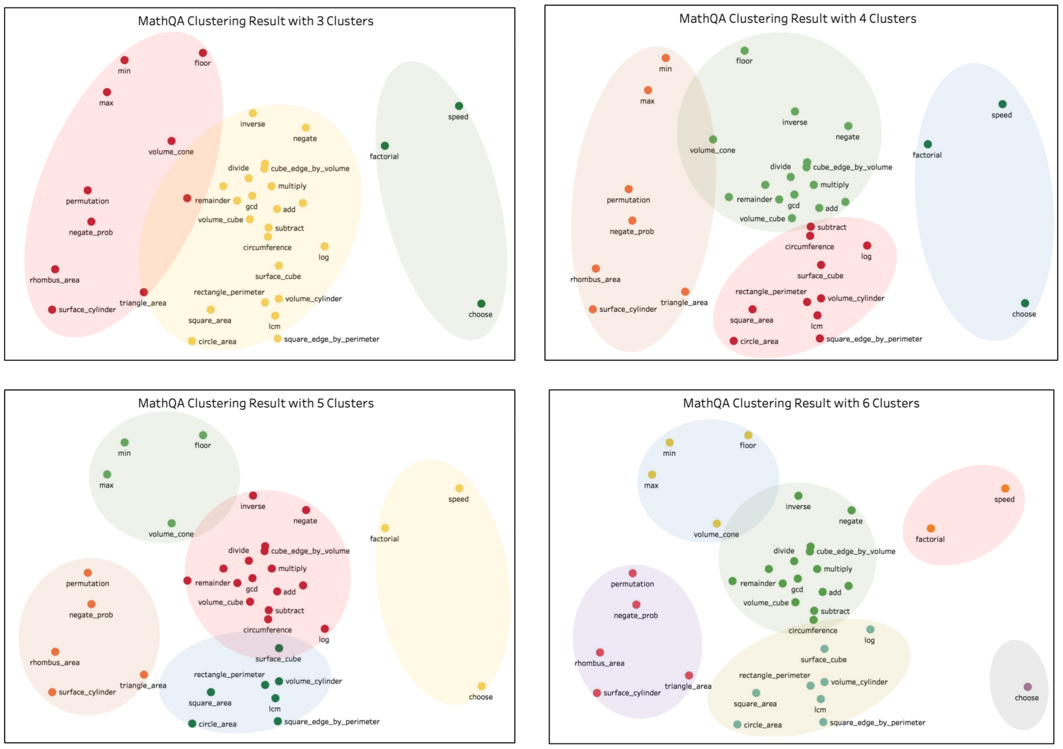

Figure 3. MathQA clustering results

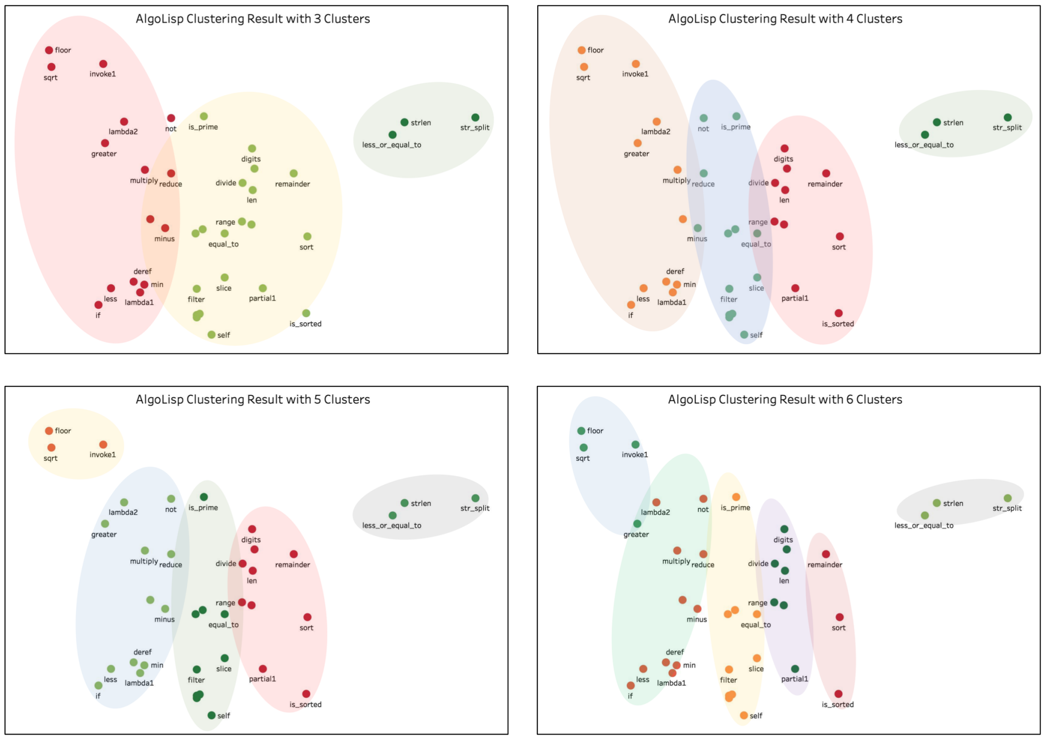

Figure 4. AlgoLisp clustering results

References

Amini, A., Gabriel, S., Lin, P., Kedziorski, R. K., Choi, Y., and Hajishirzi, H. Mathqa: Towards interpretable math word problem solving with operation-based formalisms. In *NACCL*, 2019.

Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

Luong, M.-T., Pham, H., and Manning, C. D. Effective approaches to attention-based neural machine translation. *EMNLP*, pp. 533–536, 2015.

Polosukhin, I. and Skidanov, A. Neural program search: Solving programming tasks from description and examples. In *ICLR workshop*, 2018.