SoftRegex: Generating Regex from Natural Language Descriptions using Softened Regex Equivalence

Jun-U Park
Yonsei University
Seoul, Republic of Korea
junupark@yonsei.ac.kr

Sang-Ki Ko
Kangwon National University
Kangwon, Republic of Korea
narame7@gmail.com

Marco Cognetta[*]
Yonsei University
Seoul, Republic of Korea
cognetta.marco@gmail.com

Yo-Sub Han
Yonsei University
Seoul, Republic of Korea
emmous@yonsei.ac.kr

Abstract

We continue the study of generating semantically correct regular expressions from natural language descriptions (NL). The current state-of-the-art model, SemRegex, produces regular expressions from NLs by rewarding the reinforced learning based on the semantic (rather than syntactic) equivalence between two regular expressions. Since the regular expression equivalence problem is PSPACE-complete, we introduce the EQ_Reg model for computing the similarity of two regular expressions using deep neural networks. Our EQ_Reg model essentially softens the equivalence of two regular expressions when used as a reward function. We then propose a new regex generation model, SoftRegex, using the EQ_Reg model, and empirically demonstrate that SoftRegex substantially reduces the training time (by a factor of at least 3.6) and produces state-of-the-art results on three benchmark datasets.

1 Introduction

Regular expressions are an efficient tool to represent structured data with specific rules in a variety of fields such as natural language processing or text classification. However, it is not always an easy task to write an exact regular expression for those who do not have a deep knowledge of regular expressions or when the expression is very complicated, and incorrect or sloppy regular expressions may cause unexpected consequences in practice (Bispo et al., 2006; Zhang et al., 2018). Indeed, even a single character difference between regular expressions can cause them to represent completely different sets of strings. As such, researchers have begun working on a system than generates a regular expression from a natural language description provided by a human while reducing possible errors caused by incorrect regular expressions (Liu et al., 2019). Recently, Locascio et al. (2016) designed the Deep-Regex model based on the sequence-to-sequence (Seq2Seq) model (Sutskever et al., 2014) using minimal domain knowledge during the learning phase while still accurately predicting regular expressions from NLs. Later, Zhong et al. (2018a) improved the performance by training on not only syntactic content of the expressions (i.e. the exact textual representation of the expression that was used), but also the semantic content (the regular language described by the expression). However, the reward function in the SemRegex model (Zhong et al., 2018a) that determines if the predicted regular expression is semantically equivalent to the ground truth expression is known to be PSPACE-complete and is a bottleneck in practice (Stockmeyer and Meyer, 1973). Thus, if we can solve this problem (even approximately) more quickly, then we can decrease the required learning time in the natural-language-to-regular expression (NL-RX) model. Reward shaping (Mataric, 1994; Ng et al., 1999) is a well-known mechanism to estimate the reward of an action in reinforced learning without executing the action. As a reward shaping mechanism for NL-RX, we build a new model, EQ_Reg, based on deep learning that estimates the equivalence probability of two regular expressions and use it to improve the NL-RX training speed substantially. During the NL-

[*]Now at Google.

RX model training phase, EQ_Reg can quickly determine the equivalence of the predicted expression and the true expression, and pass this value as a reward for reinforcement learning. Our new NL-RX model together with the EQ_Reg model as a reward function is called SoftRegex. We demonstrate that SoftRegex substantially reduces the training time and produces state-of-the-art results on three benchmark datasets.

2 Related Work

Generating Regular Expressions: Ranta (1998) studied rule-based techniques for the conversion between multi-languages and regular expressions. Kushman and Barzilay (2013) built a parsing model that translates a natural language sentence to a regular expression, and provided a dataset which is now a popular benchmark dataset for related research. Locascio et al. (2016) proposed the Deep-Regex model based on Seq2Seq for generating regular expressions from natural language descriptions together with a dataset of 10,000 NL-RX pairs. However, due to the limitations of the standard Seq2Seq model, the Deep-Regex model can only generate regular expressions similar in shape to the training data. The SemRegex model improved the Deep-Regex model by using reinforcement learning based on the determinisitic finite automata (DFA) equivalence oracle (which determines if two regular expressions describe the same language) as a reward function. This model can generate correct regular expressions that may not resemble the ground truth answer.

Comparing Regular Expressions: A regular language can have several syntactically different regular expressions. We say that two regular expressions are equivalent if they both define the same language. The most basic method to deciding whether or not two regular expressions are equivalent is to convert both regular expressions to DFAs. However, the time and space complexity of converting regular expressions to DFAs are both exponential (Hopcroft and Ullman, 1979). Stockmeyer and Meyer (1973) showed that it is PSPACE-complete to decide if two regular expressions generate the same set of words. Thus, deciding equivalence for two regular expressions is a bottleneck calculation even for small inputs.

3 Methods

3.1 NL-RX Model

We apply the Seq2Seq model with an attention mechanism (Luong et al., 2015). We utilize LSTM (Hochreiter and Schmidhuber, 1997) cells in the Seq2Seq model, which consists of an encoder and decoder. The encoder generates a latent vector from the given natural language description. Concurrently, the decoder receives the latent vector from the encoder and generates an output.

Maximum Likelihood Estimation (MLE): Let θ be all parameters in the Seq2Seq model. The probability that the model outputs regular expressions R from a natural language input S is

$$p_\theta(R \mid S) = \prod_{t=1}^{T} p_\theta(r_t \mid r_{<t}, S). \qquad (1)$$

Here, r_t represents a predicted token at time step t. The Deep-Regex model trains itself to find the proper r_t using MLE and optimizes through minimizing loss.

Policy Gradient: The SemRegex model additionally trains the NL-RX model via a policy gradient (Williams, 1992) by rewarding the model if it generates regular expressions that are semantically equivalent to the ground truth. The reward function returns 1 if the output regular expression is equivalent to the answer, and 0 otherwise. In short, the SemRegex model trains itself to maximize the following objective function:

$$J(\theta) = \sum_{(S,R)\in D} p(R \mid S) r(R), \qquad (2)$$

where D is a training set, $p(R \mid S)$ represents the probability that the model predicts regular expression for given sentence, and $r(R)$ is the return value of the reward function that receives the predicted regular expression.

3.2 EQ_Reg Model

The reward function in the SemRegex model is based on a regular expression equivalence oracle. This test is time intensive (the regular expression equivalence problem is PSPACE-complete) and returns only a binary value (0/1) representing the equivalence. Therefore, the equivalence test is a major hurdle when training large amounts of data, which is closely related to model performance. Recently, there have been a few attempts to tackle intractable problems using deep learning (Prates

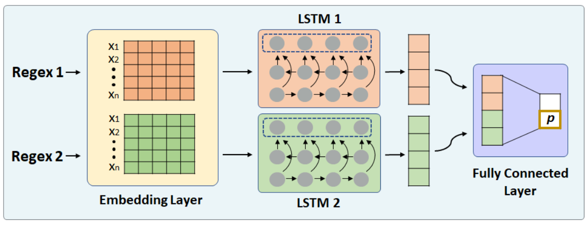


Figure 1: The configuration of the EQ_Reg model

et al., 2019; Selsam et al., 2019). This motivates us to study a different approach that can speed up the equivalence test based on deep learning and propose the EQ_Reg model that returns an equivalence probability of two regular expressions. We can then reward the NL-RX model with continuous values (the equivalence probability of two regular expressions) and significantly boost the learning speed of the NL-RX model using the EQ_Reg model.

The EQ_Reg model converts the input regular expressions to two sequences of embedding vectors. Next, we feed the embedding vector sequences into two different LSTM layers. Each LSTM layer converts the sequences to latent vectors. Since the regular expression is a grammatical expression, we present its context using a bidirectional RNN method (Graves et al., 2013) as

$$\overrightarrow{h}_t = \sigma(W_{x\overrightarrow{h}}x_t + W_{\overrightarrow{h}\overrightarrow{h}}\overrightarrow{h}_{t-1} + b_{\overrightarrow{h}}), \tag{3}$$

$$\overleftarrow{h}_t = \sigma(W_{x\overleftarrow{h}}x_t + W_{\overleftarrow{h}\overleftarrow{h}}\overleftarrow{h}_{t-1} + b_{\overleftarrow{h}}), \tag{4}$$

$$y_t = \sigma(W_{y\overrightarrow{h}}\overrightarrow{h} + W_{y\overleftarrow{h}}\overleftarrow{h}) + b_y, \tag{5}$$

where $\overrightarrow{h}$ is the forward hidden sequence and $\overleftarrow{h}$ is the backward hidden sequence. We concatenate the latent vectors from each LSTM layer into one vector and so that a single vector contains both regular expressions information. Finally, the fully connected layer predicts the equivalence of the two regular expressions from their concatenated vector.

4 Experimental Setup

We run our experiments on a computer with the following specifications: AMD Ryzen 7 1700 8-core with 64GB RAM and a GTX 1080Ti GPU on Ubuntu 16.04.1. Our source code is written using PyTorch 0.4.0 (Ketkar, 2017).

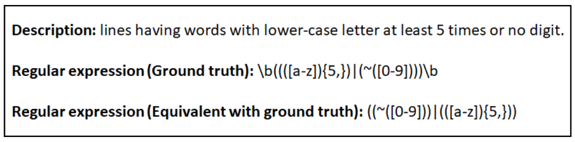
4.1 EQ_Reg Model

Datasets: Locascio et al. (2016) created a set of NL-RX pair data by arbitrarily creating and combining data in a tree form. We define the *depth* of a regular expression in this dataset as the depth of the tree that generated the NL-RX pair (see Appendix A). Similar to Locascio et al. (2016), we randomly generate regular expression pairs up to depth three and label the equivalence between each pair. We sample approximately 200,000 pairs using this method with a ratio of equivalent and non-equivalent pairs of about 2:1. We prepare three sets of data having different depths for test data. One set is made up of only regular expressions with depth at most 3, which is the same as the training data (10,000 pairs). The other two have depths 4 and 5, respectively, and each contain 1,000 pairs of regular expressions of which half are equivalent.

Model Settings: We set the two LSTMs in our model to not have shared parameters as we found this gives better performance. The embedding dimension size is set to 4, since the size of vocabulary for regular expressions is relatively small compared to natural languages. We configure two independent LSTMs with 1 layer to receive the two regular expressions. The LSTM layers use the average value of the sequence output values as their final outputs and pass them to the next layer. We set the batch size to 256 and the learning rate to 0.1 and use a stochastic gradient descent optimizer (Bottou, 2010).

4.2 SoftRegex Model

Datasets: We use three public datasets to compare SoftRegex with the-state-of-the-art model, SemRegex. The KB13 (Kushman and Barzilay, 2013) dataset was constructed by regex experts and is relatively small. On the other hand, NL-RX-Synth is data generated automatically and NL-RX-Turk is made from ordinary people by paraphrasing NL descriptions in NL-RX-Synth using Mechanical Turk (Locascio et al., 2016). Both datasets have 10,000 pairs of NL-RX data. We follow the previous work in splitting the data (train: 65%, dev: 10%, test: 25%).


Description: lines having words with lower-case letter at least 5 times or no digit.

Regular expression (Ground truth): \b((([a-z]){5,})|(~([0-9])))\b

Regular expression (Equivalent with ground truth): ((~([0-9]))|(([a-z]){5,}))

Figure 2: An example NL-RX pair and another regular expression that is semantically equivalent to the ground truth.

Model settings: We arrange the SoftRegex architecture based on SemRegex. We embed the input sequence tokens with dimension size 128, stack two LSTM layers, and set the hidden state dimension size to 256. We set the batch size to 32 and learning rate to 0.05 with the ADADELTA optimizer (Zeiler, 2012). We substitute in the EQ_Reg model as the reward function which gives high reward value 1 if our model generate a regular expression that is equivalent to the ground truth (Figure 2).

5 Results and Analysis

5.1 Model Performance

EQ_Reg: We test the EQ_Reg model with the three datasets from Section 4.1. The F1-score of test set 1 (depth 3) is 0.986, test set 2 (depth 4) is 0.868, and test set 3 (depth 5) is 0.853. As expected, we can see the model is showing high performance for test data with the same range of depth with the training data. In addition, although the EQ_Reg model has a simple structure, it shows reasonable accuracy for more complex data that has not been previously seen. Here, we can notice the model does not just remember the structure of the training data but generalizes to solve the equivalence problem by understanding the semantics of regular expressions.

SoftRegex: Table 1 shows the experimental results of Deep-Regex, SemRegex, and our SoftRegex model. The average accuracy of 10 evaluations is given. The distinguishing test cases method is based on the membership test of samples for the case when an oracle is not available and is described in (Zhong et al., 2018a). The accuracy of SoftRegex is similar to or better than SemRegex (Oracle) and always better than Deep Regex and SemRegex (Distinguishing Test Cases). Figure 3 shows the average training time per epoch over 30 epochs for each of the three models (SoftRegex and both SemRegex variants) and datasets. The training time with EQ_Reg vastly outperforms the training time for SemRegex. In the worst case (the NL-RX-Synth dataset), we see that our new method is still 3.6 times faster than that of SemRegex. Though the speedup described in experimental result may appear constant, our softened equivalence approximately decides a PSPACE-complete problem in linear time to the length of regular expressions, which would otherwise take exponential time.

Methods	KB13	NL-RX-Synth	NL-RX-Turk
Deep-Regex	65.6%	89.7%	58.2%
SemRegex (Oracle)	78.2%	91.6%	62.3%
SemRegex (Distinguishing Test Cases)	77.5%	90.2%	61.3%
SoftRegex	78.2%	91.4%	62.8%

Table 1: The accuracy of NL-RX models.

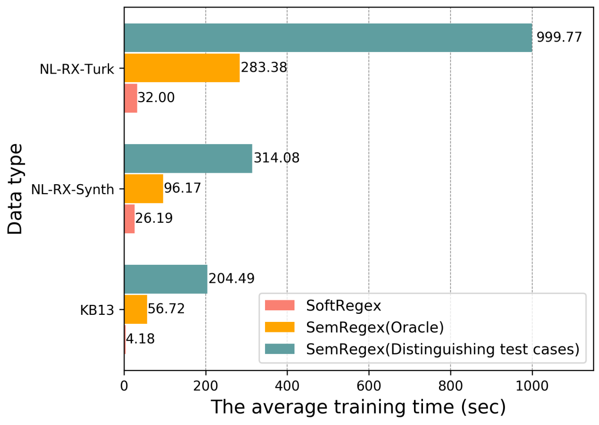


Figure 3: The average training time of NL-RX models.

5.2 Error Analysis

Zhong et al. (2018b) pointed out some problems in the NL-RX datasets. Specifically, there are some ambiguities since Locascio et al. (2016) tried to obtain data from machine-generated sentences. Thus, there are situations that even expert humans cannot accurately classify. On the other hand, the NL-RX-Turk dataset is unreliable in that it is generated by non-experts who are paraphrasing previously generated data. We investigate all 921 incorrect predictions of SoftRegex in NL-RX-Turk and categorize the resulting errors into 4 types (Table 2). The 354 type-1 errors are caused by ambiguity in the natural language description where SoftRegex prediction matches one interpretation of the natural language description but not the ground truth. The 256 type-2 errors are from incorrect regex descriptions. These errors occur when descriptions from NL-RX-Synth are misunderstood by the human annotators. The 56 type-3 errors are from incorrect user-generated data (e.g. typos in the descriptions). The remaining 296 type-4 errors are true errors that our model failed to give correct answers for. For the

NL Example	Predicted Answer	Ground Truth	Error Type
lines containing 3 or more capital letters	.*([[A-Z]]){3,}.*	(.*[A-Z].*){3,}	Description ambiguity (type-1)
lines without a capital letter or string ‘dog’	~(([A-Z])|(dog))	(~[A-Z]))|((dog)+)	Wrong description (type-2)
lines with 5 or more words without characters	\b~(.)\b	\b~((.){5,})\b	Typo in description (type-3)
lines with words and 4 lower-case letters	\b.*[a-z].*\b	\b([a-z]){4,}\b	True error (type-4)

Table 2: Example of errors caused by SoftRegex for NL-RX-Turk.

type-1 errors, we need to train more NL-RX data where a single NL is paired with several equivalent regex to cope with the NL ambiguity problem. The type-2 and type-3 errors are caused from crowd-sourcing, which might be hard to detect in the current learning model. We may consider a rule-based prescreening procedure. Finally, for the type-4 errors, we plan to incorporate the copying mechanism (Gu et al., 2016) and the sequence-to-tree translation model (Dong and Lapata, 2016) to enhance the performance of our model considering the deterministic relationship between input and output sequences and hierarchical structure of output regular expressions. Although errors are mostly caused by data errors that we can handle easily, we chose to conduct our experiments under the same condition as SemRegex to provide a fair comparison with prior work.

6 Conclusions

Recently, there have been several successful attempts to generate regular expressions from natural language. The current state-of-the-art model is based on reinforced learning using the (in)equivalence of two regular expressions. The regular expression equivalence procedure is a PSPACE-complete problem and, thus, is a major bottleneck in training both in time and space. We sidestep this bottleneck by using the EQ_Reg model as reward shaping, which gives a regex equivalence probability for two regular expressions, and build a new NL-RX model called SoftRegex. Our SoftRegex model with EQ_Reg as a reward function substantially speeds up the training phase (at least 3.6 times faster than SemRegex) while having similar or better performance on a series of standard benchmarks.

Acknowledgments

Park and Han were supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2018-0-00276).

Ko was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIP) (No. 2017-0-00255).

References

João Bispo, Ioannis Sourdis, João M. P. Cardoso, and Stamatis Vassiliadis. 2006. Regular expression matching for reconfigurable packet inspection. In *2006 IEEE International Conference on Field Programmable Technology, FPT 2006, Bangkok, Thailand, December 13-15, 2006*, pages 119–126.

Léon Bottou. 2010. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*.

Li Dong and Mirella Lapata. 2016. Language to logical form with neural attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016*, pages 33–43.

Alex Graves, Navdeep Jaitly, and Abdel-rahman Mohamed. 2013. Hybrid speech recognition with deep bidirectional LSTM. In *2013 IEEE Workshop on Automatic Speech Recognition and Understanding*, pages 273–278.

Jiatao Gu, Zhengdong Lu, Hang Li, and Victor O. K. Li. 2016. Incorporating copying mechanism in sequence-to-sequence learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016*.

Sepp Hochreiter and Jrgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

John E Hopcroft and Jeffry D. Ullman. 1979. *Introduction to automata theory, languages, and computation*.

Nikhil Ketkar. 2017. Introduction to pytorch. In *Deep learning with python*, pages 195–208.

Nate Kushman and Regina Barzilay. 2013. Using semantic unification to generate regular expressions from natural language. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 826–836.

Xiao Liu, Yufei Jiang, and Dinghao Wu. 2019. A lightweight framework for regular expression verification. In *2019 IEEE 19th International Symposium on High Assurance Systems Engineering (HASE)*, pages 1–8.

Nicholas Locascio, Karthik Narasimhan, Eduardo DeLeon, Nate Kushman, and Regina Barzilay. 2016. Neural generation of regular expressions from natural language with minimal domain knowledge. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1918–1923.

Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421.

Maja J Mataric. 1994. Reward functions for accelerated learning. In *Machine Learning Proceedings 1994*, pages 181–189.

Andrew Y Ng, Daishi Harada, and Stuart Russell. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pages 278–287.

Marcelo Prates, Pedro HC Avelar, Henrique Lemos, Luis C Lamb, and Moshe Y Vardi. 2019. Learning to solve np-complete problems: A graph neural network for decision tsp. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4731–4738.

Aarne Ranta. 1998. A multilingual natural-language interface to regular expressions. In *Proceedings of the International Workshop on Finite State Methods in Natural Language Processing*, pages 79–90.

Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill. 2019. Learning a SAT solver from single-bit supervision. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*.

Larry J Stockmeyer and Albert R Meyer. 1973. Word problems requiring exponential time (preliminary report). In *Proceedings of the fifth annual ACM symposium on Theory of computing*, pages 1–9.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.

Matthew D Zeiler. 2012. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

Shanshan Zhang, Lihong He, Slobodan Vucetic, and Eduard Dragut. 2018. Regular expression guided entity mention mining from noisy web data. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1991–2000.

Zexuan Zhong, Jiaqi Guo, Wei Yang, Jian Peng, Tao Xie, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2018a. Semregex: A semantics-based approach for generating regular expressions from natural language specifications. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1608–1618.

Zexuan Zhong, Jiaqi Guo, Wei Yang, Tao Xie, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2018b. Generating regular expressions from natural language specifications: Are we there yet? In *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*.

A Supplemental Material

We now describe how Locascio et al. (2016) generated their synthetic regular expression data. To begin, they manually mapped primitive regular expression operations, such as union and concatenation, to natural language. They then defined a small alphabet on which the operations could be performed. In the end, their system has 15 operations and 6 types of characters in the vocabulary. From this, they were able to build up NL-RX pairs automatically by creating a parse tree of repeated applications of operations to an initial regular expression. The operations (non-terminals) and alphabet (terminals) are shown in Table 3.

Figure 4 gives an example of how a NL-RX pair is generated by creating a parse tree. It can be seen that it builds from the bottom up by starting with a series of words or characters (terminals) and applying some primitive operations (non-terminals). Simultaneously, natural language descriptions of terminals and non-terminals are composed together to create a semantically identical natural language description. In this example, the NL-RX pair has depth 2.

Non-Terminals		
x&y → x and y	x|y → x or y	∼(x) → not x
.*x.*y → x followed by y	.*x.* → contains x	x{N, } → x, N or more times
x&y&z → x and y and z	x|y|z → x or y or z	x{1, N} → at most N times
x.* → starts with x	.*x → ends with x	\bx\b → words with x
(x)+ → x, at least once	(x)* → x, zero or more times	x → only x

Terminals		
[AEIOUaeiou] → a vowel	[0-9] → a number	word → the string word
[A-Z] → an uppercase letter	[a-z] → a lowercase letter	. → a character

Table 3: The non-terminal and terminal operations with their corresponding natural language description as constructed by Locascio et al. (2016).

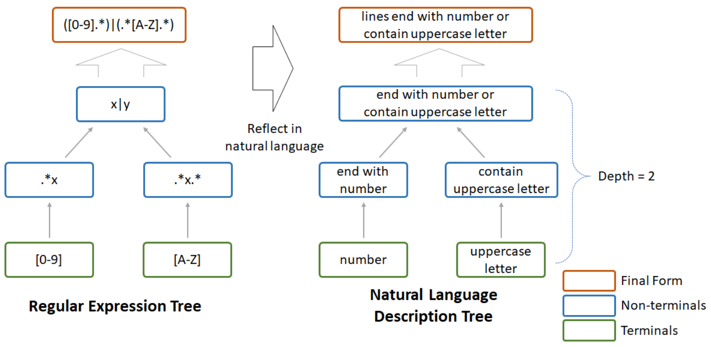


Figure 4: An example of how Locascio et al. (2016) generate the dataset of NL-RX pairs.