A Noise-tolerant Differentiable Learning Approach for Single Occurrence Regular Expression with Interleaving

Rongzhen Ye[1], Tianqu Zhuang[1], Hai Wan[1*], Jianfeng Du[2*], Weilin Luo[1], Pingjia Liang[1]

[1] School of Computer Science and Engineering, Sun Yat-sen University
[2] Guangzhou Key Laboratory of Multilingual Intelligent Processing, Guangdong University of Foreign Studies
yerzh@mail2.sysu.edu.cn, zhangzhq58@mail2.sysu.edu.cn, wanhai@mail.sysu.edu.cn, jfdu@gdufs.edu.cn,
luowlin3@mail2.sysu.edu.cn, liangpj3@mail2.sysu.edu.cn

Abstract

We study the problem of learning a *single occurrence regular expression with interleaving* (SOIRE) from a set of text strings possibly with *noise*. SOIRE fully supports interleaving and covers a large portion of regular expressions used in practice. Learning SOIREs is challenging because it requires heavy computation and text strings usually contain noise in practice. Most of the previous studies only learn restricted SOIREs and are not robust on noisy data. To tackle these issues, we propose a noise-tolerant differentiable learning approach `SOIREDL` for SOIRE. We design a neural network to simulate SOIRE matching and theoretically prove that certain assignments of the set of parameters learnt by the neural network, called *faithful encodings*, are one-to-one corresponding to SOIREs for a bounded size. Based on this correspondence, we interpret the target SOIRE from an assignment of the set of parameters of the neural network by exploring the nearest faithful encodings. Experimental results show that `SOIREDL` outperforms the state-of-the-art approaches, especially on noisy data.

Introduction

Learning regular expressions (REs) is a fundamental task in Machine Learning. For example, REs are the target in EXtensible Markup Language (XML) schema inference, for covering a set of text strings about an XML element. The regular expression with interleaving, denoted as RE(&), is an extension of regular expressions, where the operator interleaving (&) is added to interleave two strings. RE(&) has been widely used in various areas, ranging from XML database system (Makoto and Clark 2003; Colazzo et al. 2013; Martens et al. 2017) to system verification (Gischer 1981; Bojanczyk et al. 2006) and natural language processing (Kuhlmann and Satta 2009; Nivre 2009), *etc.*

We focus on a subclass of RE(&), *single occurrence regular expression with interleaving* (SOIRE), where each symbol occurs at most once in a SOIRE. Learning SOIREs is still meaningful, as SOIREs have the second highest coverage rate of REs on the schema database Relax NG among all well-known subclasses (Li et al. 2019b). Although a subclass is generally easier to learn than the full class, it is still challenging to learn SOIREs. On one hand, learning SOIREs is a search problem requiring heavy computation. On the other hand, real-life text strings usually contain noise (Kearns and Li 1988; Galassi and Giordana 2005; Bex et al. 2006). For example, in XML schema inference, the XML data may contain incorrect symbols (Bex et al. 2006). The presence of noise makes the problem of learning SOIREs more challenging.

There have been a number of proposals for learning either the full class or its subclasses of SOIRE, from a set of text strings (Freydenberger and Kötzing 2015; Zhang et al. 2018; Li et al. 2019a, 2020b). However, they are hard to guarantee that the learnt REs reach the full declared expressive power. For example, Li et al. (2019a) claimed to learn SOIREs, but Wang and Zhang (2021) showed that they just learn special cases of SOIRE. Besides, existing approaches are not robust on noisy data because any modification to given strings will alter the patterns in learnt REs. As far as we know, there is no approach to learning SOIREs that works well with both noise-free data and noisy data.

In this paper, we propose a noise-tolerant differentiable learning approach `SOIREDL` for SOIREs. Specifically, we design a neural network to simulate SOIRE matching for text strings. Since existing work for SOIRE matching (Wang 2021b, 2022) is not suitable for differentiable learning, we propose a new SOIRE matching algorithm `SOIRETM` based on the syntax tree of SOIRE, and accordingly, an algorithm for converting `SOIRETM` to a neural network. We theoretically prove that certain assignments of the set of parameters learnt by the neural network, called *faithful encodings*, one-to-one correspond to SOIREs for a bounded size. This correspondence allows us to interpret the target SOIRE from an assignment of the set of learnt parameters of the neural network by exploring the nearest faithful encodings.

To evaluate the performance of `SOIREDL` on noisy data, we extract 30 SOIREs from the RE database built by Li et al. (2018) to make a group of datasets covering five domains with different noise levels. Experimental results show that at all noise levels, the average accuracy of `SOIREDL` is higher than state-of-the-art (SOTA) approaches and the faithfulness of `SOIREDL` is always beyond 80%. In particular, the average accuracy of `SOIREDL` only decreases slightly with increasing noise levels, which suggests that `SOIREDL` is robust on noisy data.

*Both Hai Wan and Jianfeng Du are corresponding authors.

Related Work

Matching algorithms for regular expressions. It has been shown (Mayer and Stockmeyer 1994) that the matching problem of RE(&) is NP-hard. For SOIREs, Wang (2021b) proposed a finite automata with interleaving, written FA(&), to solve the matching problem. Wang (2022) proposed a single occurrence finite automata, written SFA(&, #), for matching single occurrence regular expressions with interleaving and counting. All above studies do not consider converting the matching algorithm into a neural network. In contrast, we present not only a matching algorithm for SOIREs but also the way to convert it to a neural network.

Learning approaches for regular expressions. There also exists work for learning different subclasses of REs from a set of text strings, such as the deterministic regular expressions with counting (Wang and Chen 2018), the deterministic regular expressions with unorder (Wang and Chen 2020), and the deterministic regular expression with counting and unorder (Wang 2021a). Some subclasses of the extension RE(&) have also been explored, such as chain regular expression with interleaving (ICHARE) (Zhang et al. 2018), restricted SOIRE (RSOIRE) (Li et al. 2019a), and k-occurrence regular expression with interleaving (kOIRE) (Li et al. 2020a). All of them are learnt from positive strings only. Li et al. (2020b) learnt a subclass of ICHARE, called SIRE, from both positive and negative strings based on a genetic algorithm. The relation of expressive powers supported by these classes is SIRE $\subset$ ICHARE $\subset$ RSOIRE $\subset$ SOIRE $\subset$ kOIRE. Li et al. (2021) proposed a natural language processing based RE synthesizer to learn REs from natural language descriptions together with positive and negative strings. This problem setting is different from ours.

Differentiable learning. Differentiable learning has attracted much research interest recently. Most studies focus on learning logical rules from knowledge bases (Yang, Yang, and Cohen 2017; Sadeghian et al. 2019; Cohen, Yang, and Mazaitis 2020; Huang et al. 2021) or on neural logic programming (Yang and Song 2020; Gao et al. 2022; Wang et al. 2020). In particular, some studies (Rocktäschel and Riedel 2017; Minervini et al. 2020b,a) focus on neural theorem proving. They convert the symbolic operations into differentiable modules to enhance the reasoning ability of the neural network. Mensch and Blondel (2018) utilized a strongly convex regularizer to smooth the max operator and convert a broad class of dynamic programming (DP) algorithms into differentiable operators. Wang et al. (2019) proposed a differentiable MaxSAT solver integrated into the deep learning networks to solve the problems like visual Sudoku, which has implicit satisfiability constraints. For REs, Jiang et al. (2020) injected a weighted finite-state automaton (FSA) of REs into the recurrent neural network (RNN) to improve the performance of text classification. Further, Jiang, Jin, and Tu (2021) injected a finite-state transducer of REs into RNN for slot filling. Both of the above methods fine-tune initial regular expressions given from the expert knowledge to obtain better results. Different from all above studies, we further study the one-to-one correspondence between parameters of a neural network and SOIREs.

Preliminaries

A regular expression with interleaving, written RE(&), over an alphabet Σ is defined recursively as follows (Mayer and Stockmeyer 1994):

$$r := \epsilon \big| a \big| r_1^* \big| r_1 \cdot r_2 \big| r_1 \& r_2 \big| r_1 | r_2$$

where ϵ is empty string, the symbol $a \in \Sigma$, and r_1, r_2 are RE(&). The operator $*$ denotes Kleene-Star, $\cdot$ denotes concatenation (it can be omitted if there is no ambiguity), $\&$ denotes interleaving, $|$ denotes disjunction. The operators $?$ and $+$ are commonly used for repetition. They are defined as $r^? := r|\epsilon$ and $r^+ := r \cdot r^*$, respectively. The operator $\&$ for two strings s_1, s_2 is defined as follows:

$$s_1 \& s_2 := \begin{cases} s_2 & \text{if } s_1 = \epsilon \\ s_1 & \text{if } s_2 = \epsilon \\ a(s_1' \& s_2) | b(s_1 \& s_2') & otherwise \end{cases}$$

where $s_1 = as_1'$, $s_2 = bs_2'$, $a, b \in \Sigma$.

Definition 1 (Single occurrence regular expression with interleaving (SOIRE) (Li et al. 2019a))**.** *SOIRE is a RE(&) where each symbol occurs at most once in the expression.*

Example 1. *$(a\&b)c^*$ is a SOIRE. $(a\&b)a^*$ is a RE(&) but not a SOIRE, because the symbol a occurs twice.*

Each SOIRE can be expressed by its prefix notation where operators are written in front of operands rather than written in the middle as the infix notation. By $\texttt{PreForm}(r)$ we denote the prefix notation of a SOIRE r. For the SOIRE $r = (a\&b)c^*$ given in Example 1, $\texttt{PreForm}(r)$ is $\cdot\&ab*c$.

Syntax tree. Each SOIRE can also be represented as a binary tree, called *syntax tree*, where each inner vertex in the tree represents an operator and each leaf represents a symbol. By $\texttt{RE2Tree}(r)$ we denote the syntax tree of a SOIRE r. In a syntax tree, each leaf represents a symbol in Σ and each symbol occurs at most once, while each inner vertex represents an operator and the number of children of it is equal to the number of its operands. Figure 1 shows the syntax tree of $(a\&b)c^*$ in Example 1. The size of a SOIRE r, denoted by $|r|$, is defined as the number of vertices in the syntax tree of r. For example, the size of $(a\&b)c^*$ is 6. Obviously, the preorder traversal sequence of the syntax tree of r is $\texttt{PreForm}(r)$ and each subtree represents a subexpression. In this preorder traversal sequence, vertex $t+1$ is the left child for any inner vertex t. We use $r^t (1 \le t \le |r|)$ to denote the corresponding SOIRE of the subtree of $\texttt{RE2Tree}(r)$ whose root is vertex t. Further, if vertex t represents a binary operator, we use η^t to denote the sequential number of its right child. For $r = (a\&b)c^*$ in Example 1, $r^2 = a\&b$ and $\eta^2 = 4$.

From prefix notation to syntax tree. We show a way to realize $\texttt{RE2Tree}(r)$. It scans $\texttt{PreForm}(r)$ from back to front and maintains a stack of syntax trees. Take Figure 1 for instance, $\texttt{PreForm}(r) = \cdot\&ab*c$. $\texttt{RE2Tree}(r)$ scans $\texttt{PreForm}(r)$ from c to $\cdot$. When scanning c, push c (v_6) into the stack. When scanning $*$, pop c (v_6) from the stack and make c the left child of $*$, then push $*c$ (v_5) into the stack. When scanning b and a, push b (v_4) and a (v_3) into the stack. When scanning $\&$, pop a (v_3) and b (v_4) from the stack and

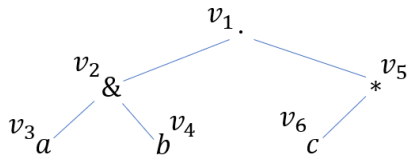

Figure 1: The syntax tree of SOIRE $(a\&b)c^*$. v_1, …, v_6 represent $\cdot$, $\&$, a, b, $*$ and c, respectively.

make a the left child of $\&$ and b the right child, then push $\&ab$ (v_2) into the stack. In this way, the syntax tree in Figure 1 is built. We use $\texttt{Tree2RE}(\xi)$ to denote the inverse function of $\texttt{RE2Tree}(r)$, which returns $\texttt{PreForm}(r)$ from the syntax tree $\xi = \texttt{RE2Tree}(r)$ by preorder traversal.

SOIRE matching. By $r \models s$ we denote that a SOIRE r matches a string s. Then the problem of SOIRE matching is to check whether $r \models s$, which can be decided in the following way:

$$r \models s := \begin{cases} s = a & \text{if } r = a \\ s = \epsilon \vee r_1 \models s & \text{if } r = r_1^? \\ s = \epsilon \vee \exists s_1 \exists s_2, (s = s_1 s_2 \wedge s_2 \neq \epsilon \\ \wedge r_1^* \models s_1 \wedge r_1 \models s_2) & \text{if } r = r_1^* \\ \exists s_1 \exists s_2, (s = s_1 s_2 \wedge r_1^* \models s_1 \wedge r_1 \models s_2) & \text{if } r = r_1^+ \\ \exists s_1 \exists s_2, (s = s_1 s_2 \wedge r_1 \models s_1 \wedge r_2 \models s_2) & \text{if } r = r_1 \cdot r_2 \\ \exists s_1 \exists s_2, (s = s_1 \& s_2 \wedge r_1 \models s_1 \wedge r_2 \models s_2) & \text{if } r = r_1 \& r_2 \\ r_1 \models s \vee r_2 \models s & \text{if } r = r_1 | r_2 \end{cases} \quad (1)$$

where r_1, r_2 are SOIREs and $a \in \{\epsilon\} \cup \Sigma$.

SOIRE learning. Given a set of strings $\Pi = \Pi^+ \cup \Pi^-$, where Π^+ (*resp.* Π^-) denotes the set of strings with the positive (*resp.* negative) label, the problem of SOIRE learning is to find a SOIRE r that maximizes $\text{acc}(r)$ defined below.

$$\text{acc}(r) = \frac{|\{s | r \models s, s \in \Pi^+\}| + |\{s | r \not\models s, s \in \Pi^-\}|}{|\Pi|} \quad (2)$$

The Proposed SOIREDL Approach

In this section, we first describe a new matching algorithm SOIRETM, then show how to convert SOIRETM to a neural network. Afterwards, we show correspondence between parameters of the neural network and SOIREs. Finally, we show how to interpret the target SOIRE from the parameters.

First of all, we introduce a variant problem of SOIRE matching, called *filter matching*. Filter matching for a SOIRE r and a string s is to check if r matches $filter(s, \alpha(r))$, where $\alpha(r)$ denotes the set of symbol in a SOIRE r, and the filter function $filter(s, V)$ returns a string that only retains symbols in V, where $V \subseteq \Sigma$. For Example 1 and $s = dbac$, the corresponding problem of filter matching is to check if $(a\&b)c^*$ matches $filter(dbac, \{a, b, c\}) = bac$, as $\alpha((a\&b)c^*) = \{a, b, c\}$.

Given a SOIRE r and a string s, we use $g_{i,j}^t \in \{0, 1\}$ $(1 \leq t \leq |r|, 1 \leq i, j \leq |s|)$ to denote whether r^t matches $filter(s_{i,j}, \alpha(r^t))$, where $s_{i,j}$ denotes the substring of s from i to j, and where $s_{1,0} = \epsilon$ specially. If r^t matches $filter(s_{i,j}, \alpha(r^t))$, then $g_{i,j}^t = 1$, otherwise $g_{i,j}^t = 0$. In particular, $g_{1,0}^t$ denotes if r^t matches ϵ since for all t from 1 to $|r|$, $filter(s_{1,0}, \alpha(r^t)) = \epsilon$. For Example 1 and $s = dbac$, $g_{1,2}^2$ denotes if $a\&b$ matches ba and $g_{1,2}^2 = 1$.

SOIRE	Semantics of $r \models filter(s, \alpha(r))$
$r = a \in \Sigma$	1. $filter(s, a) = a$.
$r = r_1^?$ $= \epsilon \vert r_1$	1. $filter(s, \alpha(r_1^?)) = \epsilon$. 2. $r_1 \models filter(s, \alpha(r_1))$.
$r = r_1^*$ $= \epsilon \vert r_1$ $\vert r_1^* \cdot r_1$	1. $filter(s, \alpha(r_1^*)) = \epsilon$. 2. $r_1 \models filter(s, \alpha(r_1))$. 3. $r_1^* \models filter(s_1, \alpha(r_1^*))$ and $r_1 \models filter(s_2, \alpha(r_1))$, where $s = s_1 s_2 (s_1, s_2 \neq \epsilon)$.
$r = r_1^+$ $= r_1$ $\vert r_1^+ \cdot r_1$	1. $r_1 \models filter(s, \alpha(r_1))$. 2. $r_1^+ \models filter(s_1, \alpha(r_1^+))$ and $r_1 \models filter(s_2, \alpha(r_1))$, where $s = s_1 s_2 (s_1, s_2 \neq \epsilon)$.
$r = r_1 \cdot r_2$	1. $r_1 \models filter(s, \alpha(r_1 \cdot r_2))$ and $r_2 \models \epsilon$. 2. $r_2 \models filter(s, \alpha(r_1 \cdot r_2))$ and $r_1 \models \epsilon$. 3. $r_1 \models filter(s_1, \alpha(r_1 \cdot r_2))$ and $r_2 \models filter(s_2, \alpha(r_1 \cdot r_2))$, where $s = s_1 s_2 (s_1, s_2 \neq \epsilon)$.
$r = r_1 \& r_2$	$r_1 \models filter(s, \alpha(r_1))$ and $r_2 \models filter(s, \alpha(r_2))$.
$r = r_1 \vert r_2$	1. $r_1 \models filter(s, \alpha(r_1 \vert r_2))$. 2. $r_2 \models filter(s, \alpha(r_1 \vert r_2))$.

Table 1: The semantics of filter matching, where r, r_1, r_2 are SOIREs and s, s_1, s_2 are strings. $r \models filter(s, \alpha(r))$ if and only if at least one condition on the right is satisfied.

SOIRE Matching by SOIRETM

We propose a new matching algorithm for SOIRE, named SOIRETM, based on dynamic programming. Generally, SOIRETM divides the original matching problem into smaller ones to conquer. We observe that SOIRE matching can be simplified to filter matching, as shown in the following theorem.

Theorem 1. *Given a SOIRE r and a string s, $r \models s$ iff $filter(s, \alpha(r)) = s$ and $r \models filter(s, \alpha(r))$.*[1]

Theorem 1 presents a necessary and sufficient condition for SOIRE matching. The condition $filter(s, \alpha(r)) = s$ guarantees that $\alpha(r)$ contains all symbols in s, which is easy to check. Therefore, the primary problem is to check if r matches $filter(s, \alpha(r))$.

We build the syntax tree of r and compute the result of filter matching of s and r, namely $g_{1,|s|}^1$. The proposed algorithm SOIRETM is detailed in Algorithm 1. Initially, line 1 checks if $filter(s, \alpha(r)) = s$. Then we calculate $g_{i,j}^t$ from shorter substrings to longer ones and from bottom to top of the syntax tree. The statements in Line 7-20 conform to the semantics of each operator, given by Table 1 and Lemma 2.

[1] All the proofs of theorems/lemmas/propositions are provided in appendix of (Ye et al. 2022).

Algorithm 1 Matching Algorithm for SOIRE, SOIRETM.

Input: A SOIRE r and a string s.
Output: The answer of whether r matches s.

1: **if** $filter(s, \alpha(r)) \neq s$ **then**
2: **Return** 0;
3: Build the syntax tree of r by RE2Tree(r);
4: Let $flag_{i,j}^{t,t'}$ denote $1[filter(s_{i,j}, \alpha(r^t)) = filter(s_{i,j}, \alpha(r^{t'}))]$, where $1[\mu] = 1$ iff μ is true;
5: **for all** substring $s_{i,j}$ from ϵ to s (shorter to longer) **do**
6: **for** $t \leftarrow |r|$ downto 1 **do**
7: **if** $r^t = a \in \alpha(r)$ **then**
8: $g_{i,j}^t \leftarrow 1[filter(s_{i,j}, \{a\}) = a]$;
9: **else if** $r^t = (r^{t+1})^?$ **then**
10: $g_{i,j}^t \leftarrow 1[filter(s_{i,j}, \alpha(r^t)) = \epsilon] \vee g_{i,j}^{t+1}$;
11: **else if** $r^t = (r^{t+1})^*$ **then**
12: $g_{i,j}^t \leftarrow 1[filter(s_{i,j}, \alpha(r^t)) = \epsilon] \vee g_{i,j}^{t+1} \vee \bigvee_{k=i}^{j-1}(g_{i,k}^t \wedge g_{k+1,j}^{t+1})$;
13: **else if** $r^t = (r^{t+1})^+$ **then**
14: $g_{i,j}^t \leftarrow g_{i,j}^{t+1} \vee \bigvee_{k=i}^{j-1}(g_{i,k}^t \wedge g_{k+1,j}^{t+1})$;
15: **else if** $r^t = r^{t+1} \cdot r^{\eta^t}$ **then**
16: $g_{i,j}^t \leftarrow (flag_{i,j}^{t,t+1} \wedge g_{i,j}^{t+1} \wedge g_{1,0}^{\eta^t}) \vee (flag_{i,j}^{t,\eta^t} \wedge g_{i,j}^{\eta^t} \wedge g_{1,0}^{t+1}) \vee \bigvee_{k=i}^{j-1}(flag_{i,k}^{t,t+1} \wedge g_{i,k}^{t+1} \wedge flag_{k+1,j}^{t,\eta^t} \wedge g_{k+1,j}^{\eta^t})$;
17: **else if** $r^t = r^{t+1} \& r^{\eta^t}$ **then**
18: $g_{i,j}^t \leftarrow g_{i,j}^{t+1} \wedge g_{i,j}^{\eta^t}$;
19: **else if** $r^t = r^{t+1} | r^{\eta^t}$ **then**
20: $g_{i,j}^t \leftarrow (flag_{i,j}^{t,t+1} \wedge g_{i,j}^{t+1}) \vee (flag_{i,j}^{t,\eta^t} \wedge g_{i,j}^{\eta^t})$;
21: **Return** $g_{1,|s|}^1$;

Lemma 2. *Given a SOIRE r and a string s. If $r = r_1 \cdot r_2$ or $r = r_1 \& r_2$ or $r = r_1 | r_2$, then for all $i \in \{1, 2\}$, $r_i \models filter(s, \alpha(r))$ iff $filter(s, \alpha(r_i)) = filter(s, \alpha(r))$ and $r_i \models filter(s, \alpha(r_i))$.*

The time complexity of Algorithm 1 is $O(|s|^3|r|)$. It is sound and complete according to the following theorem.

Theorem 3. *Given a SOIRE r and a string s, $r \models s$ iff SOIRETM$(r, s) = 1$.*

From SOIRETM to Neural Network

We now detail how to convert SOIRETM to a trainable neural network to simulate SOIRE matching.

SOIRETM uses the syntax tree for SOIRE matching, so the trainable parameters of an expected neural network can be defined by constructs of the syntax tree. There are two parts of parameters used in an expected neural network, $\theta = (w, u)$, where $w \in [0, 1]^{T \times |\mathbb{B}|}$, $u \in [0, 1]^{T \times T}$ and $\mathbb{B} = \Sigma \cup \{?, *, +, \cdot, \&, |, \text{none}\}$, and where T is the bounded size of the target SOIRE. For $1 \leq t \leq T, a \in \mathbb{B}$, w_a^t denotes the probability of vertex t representing a symbol in Σ or an ordinary operator or the none operator. For $1 \leq t \leq T$ and $t + 2 \leq t' \leq T$, $u_{t'}^t$ denotes the probability of vertex t choosing vertex t' as its right child. The total number of the parameters to be learnt is $T|\mathbb{B}| + \frac{(T-1)(T-2)}{2}$. Example 2 shows the parameters of the neural network $\theta = (w, u)$ corresponding to the syntax tree in Figure 1.

Example 2. *When $T = 6$, $w_\cdot^1$, $w_\&^2$, w_a^3, w_b^4, w_*^5, w_c^6, u_5^1, u_4^2 are* 1*s, whereas other parameters are* 0*s. When $T = 8$, w_{none}^7, w_{none}^8 are* 1*s in addition to the above parameters.*

There are four parts that should be considered during the conversion of SOIRETM to neural network: $\alpha(r^t)$, $flag_{i,j}^{t,t'}$, $g_{i,j}^t$, and the return value of Algorithm 1.

Recall that $\alpha(r^t)$ represents the set of symbols in r^t, which is also the set of symbols occurring in the subtree whose root is t. We use ρ_a^t to denote the probability of symbol $a \in \Sigma$ that occurs in the subtree whose root is t. We calculate ρ_a^t from bottom to top of the syntax tree by Equation 3, where $\sigma_{01}(x) = \min(\max(x, 0), 1)$. For all $t > T$ and $a \in \Sigma$, ρ_a^t is set to 0. Note that $\min(x) = -\max(-x)$, and the max function amounts to ReLU and can be approximated by a more differentiable LeakyReLU function.

$$\rho_a^t = \sigma_{01}(w_a^t + \sum_{o \in \{?,*,+,\cdot,\&,|\}} w_o^t \rho_a^{t+1} + \sum_{o \in \{\cdot,\&,|\}} w_o^t \sum_{t'=t+2}^{T} u_{t'}^t \rho_a^{t'}) \tag{3}$$

For converting $flag_{i,j}^{t,t'}$, we treat it as the probability that there does not exist a symbol occurring in both $s_{i,j}$ and $\alpha(r^t)$ but not occurring in $\alpha(r^{t'})$, as defined in Equation 4. Note that t' can only be either $t + 1$ or η^t.

$$flag_{i,j}^{t,t'} = 1 - \sigma_{01}(\sum_{a \in \Sigma} \sigma_{01}(1[a \in s_{i,j}] + (\rho_a^t - \rho_a^{t'}) - 1)) \tag{4}$$

For converting $g_{i,j}^t$, we introduce $p_{i,j}^t(?)$ (*resp.* $p_{i,j}^t(*)$ or $p_{i,j}^t(+)$) to denote the probability that the right-hand side of Line 10 (*resp.* 12 or 14) in Algorithm 1 evaluates to 1, as well as $p_{i,j}^t(\cdot, t')$ (*resp.* $p_{i,j}^t(\&, t')$ or $p_{i,j}^t(|, t')$) the probability that the right-hand side of Line 16 (*resp.* 18 or 20) with η^t substituted by t' evaluates to 1. For example, $p_{1,3}^8(\&, 10)$ denotes the probability that $g_{1,3}^9 \wedge g_{1,3}^{10}$ evaluates to 1. Since a right-hand side may contain logical connectives $\wedge$ or $\vee$, we apply the transformations given in Table 2 to estimate the ultimate probability that the right-hand side evaluates to 1, where the special transformation is only used in the third term of Line 12 and Line 16 as well as the second term of Line 14 since these terms may have a large number of operands, while anywhere else the general transformations are used. With the probabilities that the right-hand sides evaluate to 1, the probability that $g_{i,j}^t$ evaluates to 1 can be defined recursively by Equation 5, where we reuse $g_{i,j}^t$ to represent such a probability.

$$g_{i,j}^t = \sum_{a \in \Sigma} w_a^t \cdot 1[filter(s_{i,j}, a) = a] + \sum_{o \in \{?,*,+\}} w_o^t p_{i,j}^t(o) + \sum_{o \in \{\cdot,\&,|\}} w_o^t \sum_{t'=t+2}^{T} u_{t'}^t p_{i,j}^t(o, t') \tag{5}$$

Logical operation	General transformation	Special transformation
$A \wedge B$	$\min(p_A, p_B)$	-
$A \vee B$	$\sigma_{01}(p_A + p_B)$	$\max(p_A, p_B)$

Table 2: The transformation from logical operations to numerical computations, where A and B are formulae, and p_A (*resp.* p_B) is the probability that A (*resp.* B) evaluates to 1.

For converting the return value of Algorithm 1, we consider Line 1 and Line 21 in Algorithm 1 and compute the return value by Equation 6, where the second term in Equation 6 ensures that all symbols occurring in s appear in the target SOIRE too.

$$\hat{y} = g^1_{1,|s|} - \max_{a \in \Sigma} \sigma_{01}(1[a \in s] - \rho^1_a) \tag{6}$$

The converted neural network is trained to minimize the objective function $\frac{1}{2}(\hat{y}-y)^2$, where $y \in \{0, 1\}$ is the ground-truth label for r matching s.

Faithful Encoding

We simply refer to an assignment of the set of trainable parameters of the converted neural network as an *encoding* of SOIREs. We find that encodings can one-to-one correspond to prefix notations of SOIREs for a bounded size, when it satisfies certain conditions given by Definition 2.

Definition 2 (Faithful encoding)**.** *An encoding $\theta = (w, u)$ of SOIREs with length T is said to be* faithful *if it satisfies all the following conditions:*

1. $\forall 1 \le t \le T, w^t$ *is a one-hot vector.*
2. $\forall 1 \le t \le T, u^t$ *is either a one-hot vector or an all-zero vector.*
3. $\forall 1 \le t \le T, \sum_{t'=t+2}^{T} u^t_{t'} + \sum_{a \in \Sigma \cup \{?,*,+,\mathit{none}\}} w^t_a = 1.$
4. $\forall 1 \le t \le T-1, w^{t+1}_{\mathit{none}} - w^t_{\mathit{none}} \ge 0.$
5. $\forall 2 \le t \le T, \sum_{a \in \{?,+,*,\cdot,\&,|\}} w^{t-1}_a + \sum_{t'=1}^{t-2} u^{t'}_t + w^t_{\mathit{none}} = 1.$
6. $\forall 3 \le t \le T, \forall 1 \le p \le t-2, (t-1-p)u^p_t + \sum_{p'=p+1}^{t-1} \sum_{t'=t+1}^{T} u^{p'}_{t'} \le t-1-p.$
7. $\forall a \in \Sigma, \sum_{t=1}^{T} w^t_a \le 1.$

All conditions in a faithful encoding are translated from the construction constraints of a syntax tree. Condition 1 guarantees that each vertex in the syntax tree represents a symbol, an ordinary operator or the none operator. Condition 2 guarantees that each vertex has at most one right child. Condition 3 guarantees that each vertex either has a right child, or represents a symbol, a unary operator or the none operator. Condition 4 guarantees that if vertex t represents the none operator, then vertex $t+1$ is also the none operator. Condition 5 guarantees that if vertex t represents the none operator, then vertex t is not the child of any vertex; otherwise, vertex t is the child of exactly one vertex. Condition 6 guarantees that the vertices are numbered in the order of pre-order traversal. Condition 7 guarantees that each symbol in Σ occurs at most once in the syntax tree. Example 2 shows two faithful encodings with lengths 6 and 8, respectively.

By $\texttt{Enc2Pre}(\theta)$ we denote the prefix notation of the SOIRE interpreted from a faithful encoding θ. $\texttt{Enc2Pre}(\theta)$ decodes w^t and $u^{t'}_t$ $(1 \le t' \le t-2)$ into the syntax tree of r from $t = 1$ to T progressively until $w^t_{\text{none}} = 1$, and then translates the constructed syntax tree of r to the prefix notation of r. Take Example 2 for instance, $\texttt{Enc2Pre}(\theta) = \cdot\&ab*c$ is decoded from the faithful encoding θ with length $T = 8$.

Proposition 4 shows that $\texttt{Enc2Pre}(\theta)$ always yields the prefix notation of a SOIRE.

Proposition 4. *For any faithful encoding θ, there exists a SOIRE r such that $\texttt{Enc2Pre}(\theta) = \texttt{PreForm}(r)$.*

Proposition 5 and Proposition 6 show that $\texttt{Enc2Pre}(\theta)$ is surjective and injective, respectively, for a bounded size.

Proposition 5. *Given a bounded size $T \in \mathbb{Z}^+$, for any SOIRE r such that $|r| \le T$, there exists a faithful encoding θ with length T such that $\texttt{Enc2Pre}(\theta) = \texttt{PreForm}(r)$.*

Proposition 6. *Given a bounded size $T \in \mathbb{Z}^+$, for any two different faithful encodings θ_1 and θ_2 with length T, we have $\texttt{Enc2Pre}(\theta_1) \neq \texttt{Enc2Pre}(\theta_2)$.*

Since $\texttt{Enc2Pre}(\theta)$ is both injective and surjective, faithful encodings are one-to-one corresponding to the prefix notations of SOIREs for a bounded size.

Theorem 7. *Given a bounded size $T \in \mathbb{Z}^+$, prefix notations of SOIREs r with $|r| \le T$ and faithful encodings θ with length T have a one-to-one correspondence, i.e., $\texttt{Enc2Pre}(\theta) = \texttt{PreForm}(r)$.*

To make an encoding more faithful, we add one regularization for each condition to the objective function[2].

SOIRE Interpretation

We apply beam search to find a faithful encoding nearby the learnt encoding and then interpret it to the target SOIRE. The algorithm is shown in Algorithm 2. The interpretation steps are conducted from bottom to top of the syntax tree. We keep β candidate SOIREs for each subtree according to the score of each candidate SOIRE, which is defined as the geometric mean of the probabilities of all operators and symbols. At each step, we select different operators and candidate SOIREs from the left child and right child (if any) and merge them to generate new candidates. At last, we calculate the accuracy of each SOIRE in the last step on the training set and pick out the SOIRE with the highest accuracy.

Evaluation

We conduct experiments to evaluate the performance of SOIREDL on both noise-free data and noisy data.

Datasets. We extract 30 SOIREs from the RE database built by Li et al. (2018) and generate datasets with noise from them. The SOIREs are randomly chosen from different classes: SIRE, ICHARE, RSOIRE, SOIRE. We set Σ as 10 letters. For each SOIRE r, we generate a dataset (Π^+, Π^-) randomly, making sure that for all $s \in \Pi^-$, there exists

[2]Details of regularizations are provided in (Ye et al. 2022).

Algorithm 2 SOIRE Interpretation.

Input: A training set $(\Pi^+_{\text{train}}, \Pi^-_{\text{train}})$ on the alphabet Σ, an encoding $\theta = (w, u)$ and the the beam width β.
Output: The infix notation of the target SOIRE.

Let T be the first dimension of w;
Let C^t be the set of candidate solutions (r, e) of the subtree with the root t, where r is the infix notation of a SOIRE and e is its score;
for t from T down to 1 **do**
  for all $a \in \Sigma$ **do**
    Add (a, w^t_a) into C^t;
  for all $(r_i, e_i) \in C^{t+1}$ **do**
    Add $(r^?_i, e_i w^t_?)$ into C^t;
    Add $(r^*_i, e_i w^t_*)$ into C^t;
    Add $(r^+_i, e_i w^t_+)$ into C^t;
  for t' from $t+2$ to T **do**
    for all $((r_i, e_i), (r_j, e_j)) \in C^{t+1} \times C^{t'}$ **do**
      if $\alpha(r_i) \cap \alpha(r_j) = \emptyset$ **then**
        Add $((r_i) \cdot (r_j), e_i e_j w^t_\cdot)$ into C^t;
        Add $((r_i)\&(r_j), e_i e_j w^t_\&)$ into C^t;
        Add $((r_i)|(r_j), e_i e_j w^t_|)$ into C^t;
  Sort all (r, e) in C^t according to the descending order of $e^{\frac{1}{|r|}}$ and keep only the top-β elements;
Get the accuracy of r on $(\Pi^+_{\text{train}}, \Pi^-_{\text{train}})$ for all r in C^1;
Return r in C^1 that has the highest accuracy;

s' such that $r \models s'$ and s can be modified to s' by deleting a character, inserting a character at any position, replacing a character with another one, or moving a character to any other position. For example, string abc can be modified to ac, $abac$, acc or bca. We set the maximum length of strings to 20 in the dataset. For training sets and test sets, we set $|\Pi^+| = |\Pi^-| = 250$, whereas for validation sets, we set $|\Pi^+| = |\Pi^-| = 50$. The noise levels are set to $\delta = \{0, 0.05, 0.1, 0.15, 0.2\}$, where for each δ, we reverse the labels for $|\Pi^+|\delta$ positive strings and $|\Pi^-|\delta$ negative strings in the training and validation sets.

Competitors. We choose approaches `iSOIRE` (Li et al. 2019a), `GenICHARE` (Zhang et al. 2018), `iSIRE` (Li et al. 2020b) and `RE2RNN` (Jiang et al. 2020) as competitors. `iSOIRE` learns RSOIREs and `GenICHARE` learns ICHAREs from positive strings only. `iSIRE` learns SIREs from both positive and negative strings. `RE2RNN` embeds a weighted FSA to improve the performance on text classification. It can also learn an automaton if we randomly initialize the parameters. Thus we also compare the performance between `SOIREDL` and `RE2RNN`.

Settings. We implement `iSOIRE`, `GenICHARE`, `iSIRE` according to their papers, and reuse the source code of `RE2RNN`. The hyper-parameters of `RE2RNN` are set as default, except that the number of states is set to 100 and the threshold in interpretation to 0.12 for achieving the best accuracy. We train `SOIREDL` with the AdamW optimizer. The hyper-parameters of `SOIREDL` are set as follows: the bounded size T is $4|\Sigma| - 2$ according to Proposition 8, the batch size is 64, the regularization coefficient λ is 0, and the beam width β is 500. The optimal λ is selected from $\{0, 10^{-3}, 10^{-2}, 10^{-1}, 1, 10\}$ and β from $\{10, 50, 100, 300, 500, 1000\}$ for maximizing the accuracy on the validation set. `iSOIRE`, `GenICHARE` and `iSIRE` use the union of the training and validation sets for learning.

All experiments were conducted on a Linux machine equipped with an Intel Xeon Gold 6248R processor with 126 GB RAM and a single NVIDIA A100. We train `SOIREDL` with five learning rates $0.01, 0.05, 0.1, 0.15, 0.2$ and select the SOIRE achieving the best accuracy on the validation set. Therefore, the running time of `SOIREDL` is the sum of training time and interpretation time with five learning rates. The time limit for each approach is set to 5000 seconds.

Proposition 8. *For any SOIRE r over Σ, there exists another SOIRE r' over Σ and a faithful encoding θ with length $4|\Sigma| - 2$ such that* $\texttt{PreForm}(r') = \texttt{Enc2Pre}(\theta)$ *and* $\{s | r' \models s\} = \{s | r \models s\}$.

Evaluate metrics. We use accuracy on the test set to evaluate the performance of all approaches. For `SOIREDL` and `RE2RNN`, We introduce faithfulness defined as $\frac{N_=}{|\Pi^+|+|\Pi^-|}$ to evaluate the consistency between the neural network and the interpreted SOIRE (`SOIREDL`) or automata (`RE2RNN`), where $N_=$ is the number of test strings that the neural network and the SOIRE or automata predicts the same label.

Comparison on noise-free data. The results of different approaches on noise-free data are shown in Table 3. For learning from both positive and negative strings, `SOIREDL` outperforms `iSIRE` and `RE2RNN` on almost all datasets, achieves comparable performance with `iSOIRE` and `GenICHARE` (both of which learn from positive strings only), and achieves the highest average accuracy among all approaches. Regarding the accuracy of the intermediate neural network, `SOIREDL` is also superior to `RE2RNN`. These results show that `SOIREDL` achieves the SOTA performance on noise-free data.

Comparison on noisy data. The average accuracy of different approaches on noisy data is shown in Figure 2 (a). The performance of `iSOIRE` and `GenICHARE` drops sharply when noise is present, suggesting that they are not robust on noisy data. The average accuracy of `RE2RNN` also decreases quickly when the noise level increases, and so does the neural network of it. Both `SOIREDL` and `iSIRE` perform well on noisy data. The average accuracy of `SOIREDL` slightly decreases when the noise level increases, but it still keeps higher than others at all noise levels. This suggests that `SOIREDL` is the most robust on noisy data.

Comparison in terms of faithfulness. The average faithfulness of `SOIREDL` and `RE2RNN` is shown in Figure 3. Obviously, the faithfulness of `SOIREDL` is much higher than that of `RE2RNN` and keeps over 80% at all noise levels, suggesting that the neural network of `SOIREDL` and its interpreted SOIRE are more consistent in performing SOIRE matching.

Case study. We pick one RE from each subclass of SOIREs considered in our experiments to show their learnt results, reported in Table 4. Although the ground-truth REs and the SOIREs learnt by `SOIREDL` are not exactly the same in the subclasses ICHARE and RSOIRE, they still belong to the

Data set	Positive and negative strings			Positive strings only				
	iSIRE	RE2RNN	SOIREDL	iSOIRE	GenICHARE	iSIRE	RE2RNN	SOIREDL
1	86.0	52.4 (94.4)	**100.0** (100.0)	**89.4**	**89.4**	83.8	48.0 (50.0)	57.0 (57.0)
2	77.4	48.8 (91.4)	**100.0** (100.0)	**100.0**	**100.0**	**100.0**	50.4 (50.0)	**100.0** (100.0)
3	90.8	50.6 (77.4)	**100.0** (100.0)	**100.0**	97.2	95.6	50.6 (49.6)	65.4 (65.4)
4	72.4	49.0 (80.2)	**99.6** (73.4)	**73.8**	72.6	73.4	49.6 (49.8)	61.4 (61.4)
5	**90.8**	52.6 (91.4)	58.2 (87.2)	**100.0**	**100.0**	86.2	49.6 (50.2)	52.8 (52.8)
6	77.8	51.6 (62.2)	**93.4** (93.0)	**100.0**	**100.0**	70.4	50.2 (50.0)	52.6 (52.6)
7	81.2	52.2 (95.8)	**99.2** (99.2)	**100.0**	96.4	89.0	49.2 (49.8)	69.4 (69.4)
8	76.2	49.8 (88.0)	**100.0** (100.0)	**100.0**	**100.0**	**100.0**	49.8 (50.0)	**100.0** (100.0)
9	93.8	44.2 (48.4)	**98.4** (98.4)	**99.8**	98.8	98.0	47.2 (49.6)	81.6 (81.6)
10	94.8	50.2 (89.8)	**99.8** (100.0)	**99.8**	**99.8**	82.8	44.4 (50.2)	61.2 (61.2)
11	**91.0**	52.4 (88.6)	89.2 (91.0)	**100.0**	**100.0**	91.4	47.6 (50.2)	57.4 (57.4)
12	78.6	51.2 (98.4)	**100.0** (100.0)	**100.0**	**100.0**	**100.0**	50.8 (49.4)	**100.0** (100.0)
13	96.6	52.8 (50.2)	**100.0** (100.0)	**100.0**	**100.0**	83.6	51.2 (49.6)	67.0 (67.0)
14	74.4	50.0 (79.0)	**84.8** (71.6)	70.0	**74.4**	68.8	49.0 (50.0)	54.4 (54.4)
15	95.2	54.0 (49.8)	**96.2** (100.0)	**100.0**	**100.0**	**100.0**	47.8 (50.2)	66.2 (66.2)
16	**96.8**	49.0 (71.4)	94.8 (100.0)	**100.0**	**100.0**	75.4	49.4 (50.0)	75.4 (75.4)
17	91.0	46.2 (87.2)	**100.0** (100.0)	**100.0**	**100.0**	**100.0**	50.6 (50.0)	**100.0** (100.0)
18	81.0	42.8 (78.8)	**87.4** (99.2)	87.4	**100.0**	82.0	40.2 (50.2)	53.0 (53.0)
19	88.2	50.0 (63.8)	**100.0** (93.4)	**100.0**	**100.0**	88.0	50.4 (50.0)	54.6 (54.6)
20	**93.4**	45.2 (54.4)	83.4 (90.0)	**100.0**	99.2	95.8	47.4 (50.0)	56.0 (51.2)
21	69.8	48.8 (96.2)	**100.0** (100.0)	**71.2**	**71.2**	**71.2**	49.8 (50.4)	**71.2** (71.2)
22	90.6	47.6 (50.6)	**100.0** (100.0)	**100.0**	**100.0**	91.4	50.0 (50.2)	58.2 (58.2)
23	85.6	32.8 (84.4)	**86.8** (65.2)	**90.0**	**90.0**	88.0	50.0 (48.8)	56.8 (56.8)
24	69.4	49.0 (57.2)	**74.6** (76.8)	**77.6**	76.0	71.4	47.6 (50.0)	54.2 (54.2)
25	67.8	54.2 (93.4)	**99.8** (74.6)	**70.0**	**70.0**	**70.0**	50.2 (50.0)	69.6 (69.6)
26	**80.2**	50.4 (50.4)	63.8 (98.2)	**100.0**	**100.0**	85.2	51.0 (49.6)	63.8 (63.8)
27	92.0	52.0 (91.6)	**96.4** (96.4)	**100.0**	**100.0**	97.4	52.2 (50.2)	67.6 (67.6)
28	65.0	54.8 (95.2)	**100.0** (98.6)	**65.6**	**65.6**	**65.6**	50.0 (50.0)	**65.6** (65.6)
29	93.4	56.2 (75.8)	**97.6** (97.2)	**100.0**	99.8	81.2	49.2 (49.8)	54.2 (54.2)
30	60.4	41.4 (61.6)	**78.8** (79.6)	**61.0**	**61.0**	60.4	49.8 (49.8)	54.8 (54.8)
Avg.	83.4	49.4 (76.6)	**92.7** (92.8)	91.9	**92.0**	84.9	49.1 (49.9)	66.7 (66.6)

Table 3: Accuracy (%) on noise-free data with best results in bold. For X(Y), X denotes the accuracy of the learnt SOIRE or automaton, and Y the accuracy of the neural network.

Subclass	Dataset	Ground-truth SOIREs	SOIREDL
SIRE	13	$a^?\&b^*\&c^?$	$a^?\&c^?\&b^*$
ICHARE	22	$(a\|b)^* c^* d^*$	$(a^*\&b^*) c^* d^*$
RSOIRE	3	$a^+\|(b\|c)^* \|d^+$	$(b^+\|c)^* \|a^*\|d^*$
SOIRE	1	$((a\|b) c^*)^+ d$	$((a\|b) c^*)^+ d$

Table 4: The ground-truth SOIREs and the SOIREs learnt by SOIREDL on different subclasses of SOIREs.

same subclass. These results show that SOIREDL is able to learn different subclasses of SOIREs.

We also analyse the relation between the accuracy of the SOIRE learnt by SOIREDL and the size of the ground-truth SOIRE. Figure 4 shows that the accuracy decreases when the size of the ground-truth SOIRE increases. This may be due to the difficulty for a neural network to capture the long-distance dependency in SOIRE matching.

The necessity for using negative strings. The results of learning from positive strings only are shown in Table 3 and Figure 2 (b). Neural networks do not perform well because they prefer to classify unseen strings as positive ones when training on positive strings only. Even worse, when noise is present, the accuracy of any competitor drops sharply to no more than 60% as shown in Figure 2 (b). This suggests that negative strings are crucial in effectively learning with noisy data, possibly because they infer what kinds of strings that the target SOIRE cannot match. By learning from positive and negative strings, both SOIREDL and RE2RNN achieve significantly better performance; in particular, SOIREDL outperforms iSOIRE and GenICHARE in terms of average accuracy, as shown in Table 3 and Figure 2 (a).

Conclusion and Future Work

In this paper, we have proposed a noise-tolerant differentiable learning approach SOIREDL and a matching algorithm SOIRETM based on the syntax tree for SOIREs. The neural network introduced in SOIREDL simulates the matching process of SOIRETM. Theoretically, the faith-

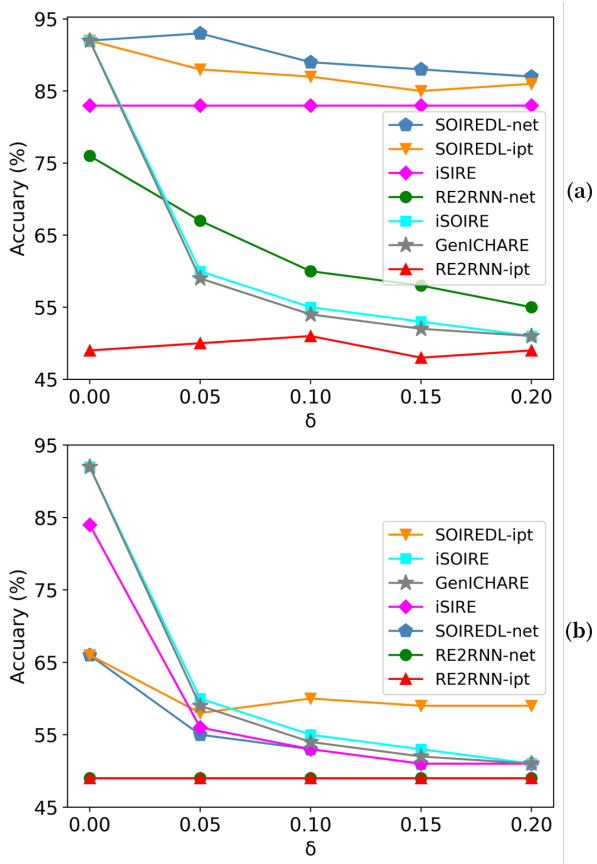

Figure 2: Average accuracy (%) on test sets at different noise levels δ. SOIREDL-ipt and RE2RNN-ipt represent the learnt SOIREs or automata, whereas SOIREDL-net and RE2RNN-net represent the neural networks. (a) Positive and negative strings. (b) Positive strings only.

ful encodings learnt by SOIREDL one-to-one correspond to SOIREs for a bounded size. Experimental results have demonstrated higher performance compared with the SOTA approaches. In the future work, we will tackle the problem of long dependency in SOIRE matching and extend our approach to other subclasses of REs.

Acknowledgments

We thank Kunxun Qi for discussion on the paper and anonymous referees for helpful comments.

This paper was supported by the National Natural Science Foundation of China (No. 62276284, 61976232, 61876204), the National Key Research and Development Program of China (No. 2021YFA1000504), Guangdong Basic and Applied Basic Research Foundation (No. 2022A1515011355), Guangzhou Science and Technology Project (No. 202201011699), Guizhou Science Support Project (No. 2022-259), as well as Humanities and Social Science Research Project of Ministry of Education (No. 18YJCZH006).

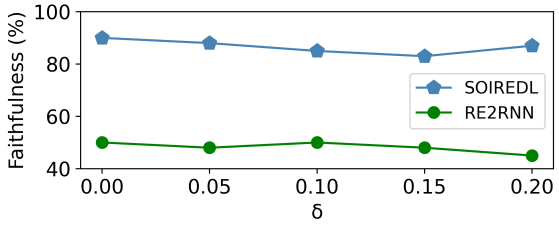

Figure 3: Average faithfulness (%) of SOIREDL and RE2RNN on test sets at different noise levels δ.

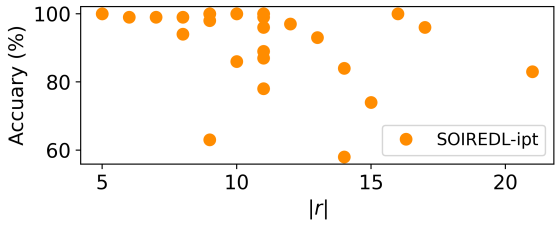

Figure 4: The relation of the accuracy (%) of a SOIRE learnt by SOIREDL and the size $|r|$ of the ground-truth SOIRE r.

References

Bex, G. J.; Neven, F.; Schwentick, T.; and Tuyls, K. 2006. Inference of Concise DTDs from XML Data. In *VLDB*, 115–126.

Bojanczyk, M.; Muscholl, A.; Schwentick, T.; Segoufin, L.; and David, C. 2006. Two-Variable Logic on Words with Data. In *LICS*, 7–16.

Cohen, W. W.; Yang, F.; and Mazaitis, K. 2020. TensorLog: A Probabilistic Database Implemented Using Deep-Learning Infrastructure. *J. Artif. Intell. Res.*, 67: 285–325.

Colazzo, D.; Ghelli, G.; Pardini, L.; and Sartiani, C. 2013. Efficient asymmetric inclusion of regular expressions with interleaving and counting for XML type-checking. *Theor. Comput. Sci.*, 492: 88–116.

Freydenberger, D. D.; and Kötzing, T. 2015. Fast Learning of Restricted Regular Expressions and DTDs. *Theory Comput. Syst.*, 57(4): 1114–1158.

Galassi, U.; and Giordana, A. 2005. Learning Regular Expressions from Noisy Sequences. In *SARA*, volume 3607, 92–106.

Gao, K.; Inoue, K.; Cao, Y.; and Wang, H. 2022. Learning First-Order Rules with Differentiable Logic Program Semantics. In *IJCAI*, 3008–3014.

Gischer, J. L. 1981. Shuffle Languages, Petri Nets, and Context-Sensitive Grammars. *Commun. ACM*, 24(9): 597–605.

Huang, J.; Li, Z.; Chen, B.; Samel, K.; Naik, M.; Song, L.; and Si, X. 2021. Scallop: From Probabilistic Deductive Databases to Scalable Differentiable Reasoning. In *NeurIPS*, 25134–25145.

Jiang, C.; Jin, Z.; and Tu, K. 2021. Neuralizing Regular Expressions for Slot Filling. In *EMNLP*, 9481–9498.

Jiang, C.; Zhao, Y.; Chu, S.; Shen, L.; and Tu, K. 2020. Cold-Start and Interpretability: Turning Regular Expressions into Trainable Recurrent Neural Networks. In *EMNLP*, 3193–3207.

Kearns, M. J.; and Li, M. 1988. Learning in the Presence of Malicious Errors (Extended Abstract). In *STOC*, 267–280.

Kuhlmann, M.; and Satta, G. 2009. Treebank Grammar Techniques for Non-Projective Dependency Parsing. In *EACL*, 478–486.

Li, Y.; Cao, J.; Chen, H.; Ge, T.; Xu, Z.; and Peng, Q. 2020a. FlashSchema: Achieving High Quality XML Schemas with Powerful Inference Algorithms and Large-scale Schema Data. In *ICDE*, 1962–1965.

Li, Y.; Chen, H.; Zhang, L.; Huang, B.; and Zhang, J. 2020b. Inferring Restricted Regular Expressions with Interleaving from Positive and Negative Samples. In *PAKDD*, volume 12085, 769–781.

Li, Y.; Chen, H.; Zhang, X.; and Zhang, L. 2019a. An effective algorithm for learning single occurrence regular expressions with interleaving. In *IDEAS*, 24:1–24:10.

Li, Y.; Chu, X.; Mou, X.; Dong, C.; and Chen, H. 2018. Practical Study of Deterministic Regular Expressions from Large-scale XML and Schema Data. In *IDEAS*, 45–53.

Li, Y.; Li, S.; Xu, Z.; Cao, J.; Chen, Z.; Hu, Y.; Chen, H.; and Cheung, S. 2021. TRANSREGEX: Multi-modal Regular Expression Synthesis by Generate-and-Repair. In *ICSE*, 1210–1222.

Li, Y.; Zhang, X.; Cao, J.; Chen, H.; and Gao, C. 2019b. Learning k-Occurrence Regular Expressions with Interleaving. In *DASFAA*, volume 11447, 70–85.

Makoto, M.; and Clark, J. 2003. RELAX NG. https://relaxng.org/. Accessed: 2022-06-01.

Martens, W.; Neven, F.; Niewerth, M.; and Schwentick, T. 2017. BonXai: Combining the Simplicity of DTD with the Expressiveness of XML Schema. *ACM Trans. Database Syst.*, 42(3): 15:1–15:42.

Mayer, A. J.; and Stockmeyer, L. J. 1994. The Complexity of Word Problems - This Time with Interleaving. *Inf. Comput.*, 115(2): 293–311.

Mensch, A.; and Blondel, M. 2018. Differentiable Dynamic Programming for Structured Prediction and Attention. In *ICML*, volume 80, 3459–3468.

Minervini, P.; Bosnjak, M.; Rocktäschel, T.; Riedel, S.; and Grefenstette, E. 2020a. Differentiable Reasoning on Large Knowledge Bases and Natural Language. In *AAAI*, 5182–5190.

Minervini, P.; Riedel, S.; Stenetorp, P.; Grefenstette, E.; and Rocktäschel, T. 2020b. Learning Reasoning Strategies in End-to-End Differentiable Proving. In *ICML*, volume 119, 6938–6949.

Nivre, J. 2009. Non-Projective Dependency Parsing in Expected Linear Time. In *ACL*, 351–359.

Rocktäschel, T.; and Riedel, S. 2017. End-to-end Differentiable Proving. In *NeurIPS*, 3788–3800.

Sadeghian, A.; Armandpour, M.; Ding, P.; and Wang, D. Z. 2019. DRUM: End-To-End Differentiable Rule Mining On Knowledge Graphs. In *NeurIPS*, 15321–15331.

Wang, P.; Donti, P. L.; Wilder, B.; and Kolter, J. Z. 2019. SATNet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *ICML*, volume 97, 6545–6554.

Wang, P.; Stepanova, D.; Domokos, C.; and Kolter, J. Z. 2020. Differentiable learning of numerical rules in knowledge graphs. In *ICLR*.

Wang, X. 2021a. Inferring Deterministic Regular Expression with Unorder and Counting. In *DASFAA*, volume 12682, 235–252.

Wang, X. 2021b. Learning Finite Automata with Shuffle. In *PAKDD*, volume 12713, 308–320.

Wang, X. 2022. Membership Algorithm for Single-Occurrence Regular Expressions with Shuffle and Counting. In *DASFAA*, volume 13245, 526–542.

Wang, X.; and Chen, H. 2018. Inferring Deterministic Regular Expression with Counting. In *Conceptual Modeling - 37th International Conference, ER*, volume 11157, 184–199.

Wang, X.; and Chen, H. 2020. Inferring Deterministic Regular Expression with Unorder. In *SOFSEM*, volume 12011, 325–337.

Wang, X.; and Zhang, X. 2021. Discovering an Algorithm Actually Learning Restricted Single Occurrence Regular Expression with Interleaving. *CoRR*, abs/2103.10546.

Yang, F.; Yang, Z.; and Cohen, W. W. 2017. Differentiable Learning of Logical Rules for Knowledge Base Reasoning. In *NeurIPS*, 2319–2328.

Yang, Y.; and Song, L. 2020. Learn to Explain Efficiently via Neural Logic Inductive Learning. In *ICLR*.

Ye, R.; Zhuang, T.; Wan, H.; Du, J.; Luo, W.; and Liang, P. 2022. A Noise-tolerant Differentiable Learning Approach for Single Occurrence Regular Expression with Interleaving. arXiv:2212.00373.

Zhang, X.; Li, Y.; Cui, F.; Dong, C.; and Chen, H. 2018. Inference of a Concise Regular Expression Considering Interleaving from XML Documents. In *PAKDD*, volume 10938, 389–401.