Discovering an Algorithm Actually Learning Restricted Single Occurrence Regular Expression with Interleaving

Xiaofan Wang and Xiaolan Zhang
State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences
University of Chinese Academy of Sciences
{wangxf,zhangxl}@ios.ac.cn

Abstract. A recent paper [3,4] proposed an algorithm iSOIRE, which combines single-occurrence automaton (SOA) [1,2] and maximum independent set (MIS) to learn a subclass single-occurrence regular expressions with interleaving (SOIREs) and claims the learnt expression is SOIRE, which has unrestricted usage for interleaving. However, in reality, the learnt expression still has many restrictions for using interleaving, even does for Kleene-star or interation, i.e, the learnt expression is not an SOIRE, we prove that by examples. In this paper, for the algorithm iSOIRE, we first give the basic notions, then provide analyses about incorrectness, finally present the correct result learnt by iSOIRE. Our theoretical analyses demonstrate that the result derived by iSOIRE belongs to a subclass of SOIREs.

1 Basic Notions
We give the notions about single-occurrence regular expressions with interleaving (SOIRE), single-occurrence automaton (SOA).

Definition 1 (regular expression with interleaving). Let $\Sigma$ denote a finite set of alphabet symbols. The regular expression with interleaving are defined as follows. $\varepsilon$, $a \in \Sigma$ are regular expressions. For the regular expressions $r_1$ and $r_2$, the concatenation $r_1 \cdot r_2$, the kleene-star $r_1^*$, the disjunction $r_1 | r_2$, the interleaving $r_1 \& r_2$ are also regular expressions. Note that iteration $r_1^+$, optional $r_?^*$ are used as abbreviations of $r_1 r_1^*$, $r r_\varepsilon$. Usually, we omit concatenation operators in examples. $L(r_1 \& r_2) = L(r_1) \& L(r_2) = \bigcup_{s_1 \in L(r_1), s_2 \in L(r_2)} s_1 \& s_2$. For $u, v \in \Sigma^*$ and $a, b \in \Sigma$, $u \& \varepsilon = \varepsilon \& u = \{u\}$, and $(au)\& (bv) = \{a(u\&v)\} \cup \{b(au\&v)\}$.

Definition 2 (single-occurrence regular expression with interleaving (SOIRE)). Let $\Sigma$ be a finite alphabet. A single-occurrence regular expression with interleaving (SOIRE) is a regular expression with interleaving over $\Sigma$ in which every terminal symbol occurs at most once.

According to the definition, SOIREs have unrestricted usage for interleaving and other operators.
Definition 3 (single-occurrence automaton (SOA) [1,2]). Let $\Sigma$ be a finite alphabet, and let $q_0$, $q_f$ be distinct symbols that do not occur in $\Sigma$. A single-occurrence automaton (SOA) over $\Sigma$ is a finite directed graph $\mathcal{A} = (V, E)$ such that (1) $\{q_0, q_f\} \subseteq V$, and $V \subseteq \Sigma \cup \{q_0, q_f\}$. (2) $q_0$ has only outgoing edges, $q_f$ has only incoming edges, and every $v \in V \setminus \{q_0, q_f\}$ is visited during a walk from $q_0$ to $q_f$.

A string $a_1 \cdots a_n$ ($n \geq 0$) is accepted by an SOA $\mathcal{A}$, if and only if there is a path $q_0 \rightarrow a_1 \rightarrow \cdots \rightarrow a_n \rightarrow q_f$ in $\mathcal{A}$.

2 Analyses about the algorithm $i$SOIRE

First, we present analyses about the algorithm $i$SOIRE. Then, in term of the expression learnt by the algorithm $i$SOIRE, we obtain two conclusions and provide the corresponding proofs. The two conclusions reveal the expression learnt by the algorithm $i$SOIRE is not an SOIRE.

For any given finite sample, SOA is constructed by using algorithm 2T-INF [1], the algorithm $i$SOIRE [3,4] combines SOA and MIS to infer an expression called SOIRE. The main procedure $Soa2Soire$ [3,4] (see Figure 2(a)) is designed by revising only few steps of the algorithm $Soa2Sore$ [2] (see Figure 1), which is used to infer a single-occurrence regular expression (SORE) [1,2]. The main differences between algorithms $Soa2Soire$ and $Soa2Sore$ are in line 5 ~ line 8.

In algorithm $Soa2Sore$, in line 5 ~ line 8, if the input SOA built form given finite sample contains a strongly connected component $(U)$, $U$ is used to infer an expression $(r)$ and then is added an iteration operator ($^{\ast}$), i.e., $r^{\ast}$. The strongly connected component $(U)$ is contracted to a vertex labelled with $r^{\ast}$. However, in algorithm $Soa2Soire$, only for the strongly connected component $(U)$ where $|U| = 1$, $U$ is used to generate an expression $(a \in \Sigma, U = \{a\})$ and then is added an iteration operator ($^{\ast}$), i.e., $a^{\ast}$. For $|U| > 1$, $U$ is used to introduce interleaving & into inferred expression by calling subroutine $Merge$ (see Figure 2(b)).

Subroutine $Merge$ is used to return an expression, where the interleaving & is introduced. $filter(U,S)$ [3,4] denotes that, for each string $s \in S$, $filter$ extracts the substring consisting of symbols in $U$, where the order of the alphabetic symbols maintains the relative order of that in $s$. The input of the subroutine $Merge$ is the set of the strings extracted by $filter$. In $Merge$, first, in line 1 ~ line 8, all $\text{mis}$ [5] (the set of the distinct maximum independent set) is computed. Then, in line 9 ~ line 13, for each obtained maximum independent set $mis \in \text{all}_mis$, $mis$ is used to generate an expression by recursively calling $Soa2Soire$, where the input is the set of the strings extracted by $filter(mis, S)$. Each generated expression is input in $U'$. Finally, in line 14, subroutine $combine$ connects all expressions in $U'$ by using interleaving &. For example $U' = \{r_1, r_2, r_3\}$, $\text{combine}(U') = r_1 \& r_2 \& r_3$. The result returned by $Merge$ is the result of $combine$.

Since the algorithm $Soa2Soire$ presented in [3,4] does not provide the proof about the correctness, for any learnt expression, we check whether the learnt expression is SOIRE, which has unrestricted usage for interleaving and other oper-
Ators. However, we discover that the learnt expression still has many restrictions for using interleaving, even does for Kleene-Star or iteration, the learnt result is not an SOIRE. For better understanding, we provide the proofs by examples, which specify the learnt result is not an SOIRE.

**Algorithm 2: Sea2Sore**

```plaintext
1. Input: SOA $A = (V,E)$; Output: SOIRE $\mathcal{A}(A)$;
2. if $|E| = 0$ then return $\emptyset$;
3. else if $|V| = 2$ then return $\emptyset$;
4. else if $A$ has a cycle then
5.     Let $U$ be a strongly connected component of $A$;
6.     $B = A$.
7.     $A_{\text{contract}}(U, \text{head}())$;
8.     $A_{\text{contract}}(U, \text{tail}(A_{\text{contract}}(U, \text{head}())))$;
9.     if $A_{\text{contract}}(A_{\text{contract}}(U, \text{head}()), A_{\text{contract}}(U, \text{tail}(A_{\text{contract}}(U, \text{head}())))$)
10.     $A_{\text{contract}}(U, \text{head}())$;
11.     $A_{\text{contract}}(U, \text{tail}(A_{\text{contract}}(U, \text{head}())))$;
12.     if $|A_{\text{contract}}(U, \text{head}()) = 1$ then
13.         $A_{\text{contract}}(U, \text{tail}(A_{\text{contract}}(U, \text{head}())))$;
14.         $A_{\text{contract}}(U, \text{head}())$;
15.     if $|A_{\text{contract}}(U, \text{head}()) = 1$ then
16.         $A_{\text{contract}}(U, \text{tail}(A_{\text{contract}}(U, \text{head}())))$;
17.         $A_{\text{contract}}(U, \text{head}())$;
18.     return $A_{\text{contract}}(U, \text{tail}(A_{\text{contract}}(U, \text{head}())))$;
19.     else if $|A_{\text{contract}}(U, \text{head}()) = 1$ then
20.         $A_{\text{contract}}(U, \text{tail}(A_{\text{contract}}(U, \text{head}())))$;
21.         $A_{\text{contract}}(U, \text{head}())$;
22.     return $A_{\text{contract}}(U, \text{tail}(A_{\text{contract}}(U, \text{head}())))$;
```

Fig. 1: The algorithm Sea2Sore.

**Algorithm 3: SOIRE**

```plaintext
Input: SOIRE $\mathcal{A}(A)$; Output: SOIRE $\mathcal{A}(A)$;
1. if $|E| = 0$ then return $\emptyset$;
2. else if $|V| = 2$ then return $\emptyset$;
3. else if $A$ has a cycle then
4.     Let $U$ be a strongly connected component of $A$;
5.     if $|U| = 1$ then
6.         $A_{\text{contract}}(U, \text{head}())$;
7.     else $A_{\text{contract}}(U, \text{head}())$;
8.     $A_{\text{contract}}(U, \text{tail}(A_{\text{contract}}(U, \text{head}())))$;
9.     if $|A_{\text{contract}}(U, \text{head}()) = 1$ then
10.        $A_{\text{contract}}(U, \text{tail}(A_{\text{contract}}(U, \text{head}())))$;
11.        $A_{\text{contract}}(U, \text{head}())$;
12.     return $A_{\text{contract}}(U, \text{tail}(A_{\text{contract}}(U, \text{head}())))$;
13.     else if $|A_{\text{contract}}(U, \text{head}()) = 1$ then
14.        $A_{\text{contract}}(U, \text{tail}(A_{\text{contract}}(U, \text{head}())))$;
15.        $A_{\text{contract}}(U, \text{head}())$;
16.     return $A_{\text{contract}}(U, \text{tail}(A_{\text{contract}}(U, \text{head}())))$;
17.     else if $|A_{\text{contract}}(U, \text{head}()) = 1$ then
18.        $A_{\text{contract}}(U, \text{tail}(A_{\text{contract}}(U, \text{head}())))$;
19.        $A_{\text{contract}}(U, \text{head}())$;
20.     return $A_{\text{contract}}(U, \text{tail}(A_{\text{contract}}(U, \text{head}())))$;
```

Fig. 2: The algorithm SOIRE.

**Algorithm 4: Merge**

```plaintext
Input: SOIRE $\mathcal{A}(A)$; Output: SOIRE $\mathcal{A}(A)$;
1. if $|E| = 0$ then return $\emptyset$;
2. if $|V| = 2$ then return $\emptyset$;
3. else if $A$ has a cycle then
4.     Let $U$ be a strongly connected component of $A$;
5.     if $|U| = 1$ then
6.         $A_{\text{contract}}(U, \text{head}())$;
7.     else $A_{\text{contract}}(U, \text{head}())$;
8.     $A_{\text{contract}}(U, \text{tail}(A_{\text{contract}}(U, \text{head}())))$;
9.     if $|A_{\text{contract}}(U, \text{head}()) = 1$ then
10.        $A_{\text{contract}}(U, \text{tail}(A_{\text{contract}}(U, \text{head}())))$;
11.        $A_{\text{contract}}(U, \text{head}())$;
12.     return $A_{\text{contract}}(U, \text{tail}(A_{\text{contract}}(U, \text{head}())))$;
13.     else if $|A_{\text{contract}}(U, \text{head}()) = 1$ then
14.        $A_{\text{contract}}(U, \text{tail}(A_{\text{contract}}(U, \text{head}())))$;
15.        $A_{\text{contract}}(U, \text{head}())$;
16.     return $A_{\text{contract}}(U, \text{tail}(A_{\text{contract}}(U, \text{head}())))$;
```

(a) Sea2Sore  (b) Merge
For the given algorithm Soa2Soire, there are two conclusions:

1. For the learnt expression, Kleene-star * or iteration + is just on a single alphabet symbol.

   \textit{Proof.} In algorithm \textit{Soa2Soire}, in line 6 \textasciitilde line 7, \textit{plus()} \cite{2}, which should have used to add an iteration + on an expression, but only works on a single alphabet symbol. In the whole process of the recursion of the algorithm, \textit{plus()} does not occur in other cases. Since the initially obtained strongly connected components have been merge into a vertex (see subroutine \textit{Contract} \cite{2}), and the corresponding edges, which can form loops in SOA, have been removed (see subroutine \textit{bend} \cite{2}), even though for a node \(v:v.label()\) is an expression consisting of multiple alphabet symbols, \(v\) does not possible occur in a strongly connected component. Hence, we can assert that a class of expressions cannot be learnt, such as \((ab)^+, (a|b)^+, (a|bkc)^+, (a\&b)^+\) and \(((a|b)&(c|d)&(e|f))^*\).

2. The expressions of form \(a\&(b(c|d))\) cannot be learnt.

   \textit{Proof.} Assume that the expression \(a\&(b(c|d))\) can be correctly learnt by induction. In \textit{merge}, the interleaving \& is introduced by the subroutine \textit{combine}, in line 9 \textasciitilde line 13, any obtained maximum independent set \(\text{mis} \in \text{all\_mis}\) can form a sample \(S' (S' = \text{filter(mis}, S))\). And SOA \(A=2\text{T-INF}(S')\). Sample \(S'\) and SOA \(A\) are as inputs of the algorithm \textit{Soa2Soire}, let \textit{delta} denote the derived expression. According to the computation of \textit{filter}, the alphabet symbols in \(\delta\) are in the same maximum independent set \(\text{mis}\). Then in line 9, for each \(\text{mis}_i \in \text{all\_mis}\), the corresponding expression \(\delta_i\) is generated. \((S'_i = \text{filter(mis}_i, S))\); SOA \(A_i=2\text{T-INF}(S'_i)\); \(\delta_i = \text{Soa2Soire}(S'_i, A_i)\); \(\delta_i\) is put into \(U\), then \textit{combine}(\(U\)) = \(\delta_1\&\delta_2\&\cdots\&\delta_i\).

   For expression \(a\&(b(c|d))\), \(\text{mis}_1 = \{a\}\), \(\text{mis}_2 = \{b, c, d\}\), \(\text{mis}_3 = \{c\}\) and \(\text{mis}_4 = \{d\}\). Then for the initially constructed SOA, since \(b, c, d\) can form a maximum independent set \(\text{mis}\), then for \(\text{mis}_3 = \{c\}\) and \(\text{mis}_4 = \{d\}\) are not maximum independent sets, respectively. There is a contradiction, the initial assumption does not hold. The expressions of form \(a\&(b(c|d))\) cannot be learnt by \textit{Soa2Soire}.

Conclusion 1 and conclusion 2 have revealed that the expression learnt by \textit{Soa2Soire} still has many restrictions for using interleaving. Such as the expressions of form \((a|b)(c|d)^+\), \((a\&b)^+\), \(((a\&b)|(c|d)|d|e|f))^+\) and \(a\&(b(c|d))\). Meanwhile, the learnt expression has restrictions for using Kleene-star or iteration, even does for the expressions of form \((ab)^+\) and \((a\&b)^+\). Actually, for conclusion 2, if \(a, b, c\) and \(d\) are replaced with regular expressions, respectively. The same conclusions can be obtained. This implies that the expression learnt by \textit{Soa2Soire} is not an SOIRE.

3 \textbf{The correct result learnt by iSOIRE}

In Section 2, we have proved that the expression learnt by \textit{Soa2Soire} is not an SOIRE. However, we should check whether the expression returned by \textit{iSOIRE}
belongs to a subclass of SOIREs or not. Here, by analyzing the algorithm Soa2Soire, we propose a subclass of SOIREs called RSOIREs (see definition 4) and prove that the expression learnt by Soa2Soire is a RSOIRE.

**Definition 4 (restricted SOIREs).** A restricted SOIRE (RSOIRE) is a regular expression with interleaving over $\Sigma$ by the following grammar, and where every terminal symbol occurs at most once.

$$P := SP \mid PS \mid ST \mid PS$$

$$S := S \& S \mid T$$

$$T := T \mid TT \mid \varepsilon \mid a^*(a \in \Sigma)$$

**Example 1.** $(a^+|b)(c\&d), ad\&(b|c^*), and a^+|b^+\&c^*$ are RSOIREs. However, $(ab)\&(c|d)^+,(a|b\&c)d^*)^*, and a\&(b(c\&d))$ are SOIREs, not RSOIREs.

**Theorem 1.** For any given finite sample $S$, let $SOA_{A=2T-INF}(S)$, and $r = Soa2Soire(S, A)$. Then $r$ is a RSOIRE, and for any RSOIRE $r'$, $r'$ can be learnt by Soa2Soire.

**Proof.** 1. $r$ is a RSOIRE.

(1) For regular expressions $\varepsilon, a, a^*$, in algorithm Soa2Soire, in line 2, line 11 ~ line 16 and line 7, they can be derived, the correctness can be ensured by the corresponding correctness of algorithm Soa2Soire.

(2) For regular expressions $r_1r_2$ and $r_1|r_2$, assume that $r_1$ and $r_2$ can be correctly derived by Soa2Soire by induction. In line 11 ~ line 16 and line 22 ~ line 23, $r_1r_2$ and $r_1|r_2$ can be derived, the correctness can also be ensured by the corresponding correctness of algorithm Soa2Soire.

(3) For regular expression $r_1\&r_2\&\cdots\&r_k$ $(k \geq 2)$, assume that $r_i$ $(1 \leq i \leq k)$ can be correctly derived by Soa2Soire by induction. According to the conclusion 2 and the corresponding proof in Section 2, $r_i$ cannot be possible to contain the interleaving $\&$. The expression $r_1\&r_2\&\cdots\&r_k$ is computed by combine in Merge, $r_i$ is derived by computing the corresponding maximum independent set $mis_i$. And for any two distinct maximum independent sets $mis_i$ and $mis_j$ $(i \neq j)$, the symbols in $mis_i$ and the symbols in $mis_j$ can be interleaved. Thus, the expression $r_1\&r_2\&\cdots\&r_k$ can be correctly derived by Soa2Soire.

The expressions discussing in (1), (2) and (3), which are connected by using concatenation or disjunction, can form complexity expressions, and certainly they can be decomposed into the above discussed basic expressions. The grammar (3) and grammar (2) presented in definition 4 can generated the expressions discussing in (1) and (2), respectively. And the complexity expressions formed by the expressions discussing in (1), (2) and (3) can be generated by the grammar (1). This implies that any expression $r$ learnt by Soa2Soire is a RSOIRE.
2. For any RSOIRE \( r', r' \) can be learnt by \( \text{Soa2Soire} \).
   
   (1) For grammar (3), the corresponding generated regular expressions can be derived by \( \text{Soa2Soire} \), the correctness can be ensured by the corresponding correctness of algorithm \( \text{Soa2Sore} \).
   
   (2) For grammar (2), according to the proofs in 1, the corresponding generated regular expressions with interleaving can also be derived by \( \text{Soa2Soire} \).
   
   (3) For grammar (1), the generated complexity expressions can be decomposed into the expressions produced by grammar (2) or grammar (3), then the complexity expressions can also be derived by \( \text{Soa2Soire} \).

   This implies that, for an expression \( r' \) generated by the defined grammars, \( r' \) can be learnt by \( \text{Soa2Soire} \).

   We give a correct class of expression that can be learnt by \( \text{Soa2Soire} \), and present the corresponding proofs of correctness. Theorem 1 demonstrate that the expression learnt by \( \text{Soa2Soire} \) belongs to a subclass of SOIREs.

4 Conclusion

In this paper, we mainly provide analyses about the incorrectness about algorithm \( i\text{SOIRE} \), and then present the correct a class of expressions can be learnt by algorithm \( i\text{SOIRE} \), the corresponding proofs illustrate that the learnt expression belongs to a subclass of SOIREs. Since the algorithm \( i\text{SOIRE} \) can be used to learn other classes of expressions, such as \( k \)-occurrence regular expression with interleaving, the corresponding correctness depends on the correctness of the algorithm \( i\text{SOIRE} \). The comments in this paper can be provided as a reference.

References

1. Bex, G.J., Neven, F., Schwentick, T., Vansummeren, S.: Inference of concise regular expressions and DTDs. ACM Transactions on Database Systems 35(2), 1–47 (2010)
2. Freydenberger, D.D., Kötzing, T.: Fast learning of restricted regular expressions and DTDs. Theory of Computing Systems 57(4), 1114–1158 (2015)
3. Chen, H., Zhang, X., Zhang, L.: An effective algorithm for learning single occurrence regular expressions with interleaving. In: Proceedings of the 23rd International Database Applications & Engineering Symposium, IDEAS 2019, Athens, Greece, June 10-12, 2019. pp. 24:1–24:10 (2019), https://doi.org/10.1145/3331076.3331100
4. Chen, H., Zhang, X., Zhang, L.: An effective algorithm for learning single occurrence regular expressions with interleaving. arXiv preprint arXiv:1906.02074 (2019)
5. Peng, F., Chen, H.: Discovering restricted regular expressions with interleaving. In: Asia-Pacific Web Conference. pp. 104–115. Springer (2015)