Comment on "An Effective Algorithm for Learning Single Occurrence Regular Expressions with Interleaving" by Li et al.

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] by Li et al. proposed an algorithm *i*SOIRE, 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 *i*SOIRE, we first give the basic notions, then provide analyses about incorrectness, finally present the correct result learnt by *i*SOIRE. Our theoretical analyses demonstrate that the result derived by *i*SOIRE 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 Σ denote a finite set of alphabet symbols. The regular expression with interleaving are defined as follows. ε, $a \in \Sigma$ 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 $r_1 r_1^*$, $r|\varepsilon$. Usually, we omit concatenation operators in examples. $\mathcal{L}(r_1\&r_2) = \mathcal{L}(r_1)\&\mathcal{L}(r_2) = \bigcup_{s_1 \in \mathcal{L}(r_1), s_2 \in \mathcal{L}(r_2)} s_1\&s_2$. For $u, v \in \Sigma^*$ and $a, b \in \Sigma$, $u\&\varepsilon = \varepsilon\&u = \{u\}$, and $(au)\&(bv) = \{a(u\&bv)\} \cup \{b(au\&v)\}$.*

Definition 2 (single-occurrence regular expression with interleaving (SOIRE)). *Let Σ be a finite alphabet. A single-occurrence regular expression with interleaving (SOIRE) is a regular expression with interleaving over Σ 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 Σ be a finite alphabet, and let q_0, q_f be distinct symbols that do not occur in Σ. A single-occurrence automaton (SOA) over Σ is a finite directed graph $\mathscr{A} = (V, E)$ such that (1) $\{q_0, q_f\} \in V$, 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 $\mathscr{A}$, if and only if there is a path $q_0 \to a_1 \to \cdots \to a_n \to q_f$ in $\mathscr{A}$.

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 $\sim$ line 8.

In algorithm $Soa2Sore$, in line 5 $\sim$ 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 ($^+$), i.e., r^+. The strongly connected component (U) is contracted to a vertex labelled with r^+. 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 ($^+$), i.e., a^+. 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 $\sim$ line 8, all_mis [5] (the set of the distinct maximum independent set) is computed. Then, in line 9 $\sim$ line 13, for each obtained maximum independent set $mis \in all\_mis$, 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 $U' = \{r_1, r_2, r_3\}$, $combine(U') = r_1 \& r_2 \& r_3$. 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 operators. 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: `Soa2Sore`

Input: SOA $A = (V, E)$;
Output: SORE minimally generalizing $\mathcal{L}(A)$;
if $|E| = 0$ **then return** $\emptyset$;
else if $|V| = 2$ **then return** ε ;
else if *A has a cycle* **then**
  Let U be a strongly connected looped component of A;
  $B_0 \leftarrow A.\mathrm{extract}(U).\mathrm{bend}()$;
  $A.\mathrm{contract}(U, \mathrm{plus}(\texttt{Soa2Sore}(B_0)))$;
else if $A.\mathrm{succ}(A.\texttt{src}) \neq A.\mathrm{first}()$ **then**
  $A.\mathrm{addEpsilon}()$;
else if $|A.\mathrm{first}()| = 1$ **then**
  Let v be the only successor of `src`;
  $\ell \leftarrow v.\mathrm{label}()$;
  $A.\mathrm{contract}(\{A.\texttt{src}, v\}, \texttt{src})$;
  $\ell' \leftarrow \texttt{Soa2Sore}(A)$;
  return $\mathrm{concatenate}(\ell, \ell')$;
else if $\exists v \in A.\mathrm{first}()$*:* $A.\mathrm{exclusive}(v) \neq \{v\}$ **then**
  Let v be such that $A.\mathrm{exclusive}(v) \neq \{v\}$;
  $U \leftarrow A.\mathrm{exclusive}(v)$;
  $A.\mathrm{contract}(U, \texttt{Soa2Sore}(A.\mathrm{extract}(U)))$;
else
  Let $u, v \in A.\mathrm{first}()$ with $u \neq v$ s.t. $A.\mathrm{reach}(u) \cap A.\mathrm{reach}(v)$ is $\subseteq$-maximal;
  $A.\mathrm{contract}(\{u, v\}, \mathrm{or}(u.\mathrm{label}(), v.\mathrm{label}()))$;
return $\texttt{Soa2Sore}(A)$;

Fig. 1: The algorithm *Soa2Sore*.

Algorithm 3: Soa2Soire

Input: a set of positive sample S; an SOA $\mathcal{A} = (V,E)$
Output: an SOIRE
if $|E| = 0$ **then return** $\varnothing$;
else if $|V| = 2$ **then return** ε;
else if *$\mathcal{A}$ has a cycle* **then**
  Let U be a strongly connected component of $\mathcal{A}$;
  if $|U| = 1$ **then**
    Let v be the only vertice of U;
    $\mathcal{A}$.contract(U,plus(v.label()));
  else $\mathcal{A}$.contract(U,Merge(filter(U, S)));
else if $\mathcal{A}$.succ($\mathcal{A}$.src) $\neq \mathcal{A}$.first() **then**
  $\mathcal{A}$.addEpsilon();
else if $|\mathcal{A}$.first()$| = 1$ **then**
  Let v be the only successor of *src*;
  $\delta \leftarrow v$.label();
  $\mathcal{A}$.contract($\{\mathcal{A}$.src,$v\}$,src);
  $\delta' \leftarrow$ Soa2Soire(S,$\mathcal{A}$);
  return concatenate(δ,δ');
else if $\exists v \in \mathcal{A}$.first(), $\mathcal{A}$.exclusive(v) $\neq \{v\}$ **then**
  Let v be such that $\mathcal{A}$.exclusive(v) $\neq \{v\}$;
  $U \leftarrow \mathcal{A}$.exclusive(v);
  $\mathcal{A}$.contract(U,Soa2Soire(S,$\mathcal{A}$.extract(U)));
else
  Let u,$v \in \mathcal{A}$.first() with $u \neq v$ s.t. $\mathcal{A}$.reach(u) $\cap$ $\mathcal{A}$.reach(v) is $\subseteq$-maximal;
  $\mathcal{A}$.contract($\{u,v\}$,or(u.label(),v.label()));
return Soa2Soire(S,$\mathcal{A}$);

(a) *Soa2Soire*

Algorithm 4 : Merge

Input: a set of positive sample S
Output: an epression ζ
constraint_tr $\leftarrow$ cs(S);
$U \leftarrow \varnothing$;
$G \leftarrow$ Graph(*constraint_tr*);
all_mis $\leftarrow \varnothing$;
while $|G.nodes()| > 0$ **do**
  $W \leftarrow$ clique_removal(G) [12];
  $G \leftarrow G \setminus W$;
  all_mis.append(G)
foreach *mis* $\in$ *all_mis* **do**
  $S' \leftarrow$ filter(*mis*, S)
  Construct SOA $\mathcal{A}$ for S' using method 2T-INF [20];
  $\delta \leftarrow$ Soa2Soire(S',$\mathcal{A}$)
  U.append(δ)
return $\zeta \leftarrow$ combine(U)

(b) *Merge*

Fig. 2: The algorithm *i*SOIRE.

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.

   Proof. In algorithm $Soa2Soire$, in line 6 $\sim$ line 7, $plus()$ [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, $plus()$ does not occur in other cases. Since the initially obtained strongly connected components have been merge into a vertex (see subroutine $Contract$ [2]), and the corresponding edges, which can form loops in SOA, have been removed (see subroutine $bend$ [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 $(ab)^+$, $(a|b)^+$, $(a|b\&c)^+$, $(a\&b)^+$ and $((a|b)\&(c|d)\&(e|f))^*$.

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

   Proof. Assume that the expression $a\&(b(c\&d))$ can be correctly learnt by induction. In $merge$, the interleaving & is introduced by the subroutine $combine$, in line 9 $\sim$ line 13, any obtained maximum independent set $mis \in all\_mis$ can form a sample S' ($S' = filter(mis, S)$). And SOA A=2T-INF(S'). Sample S' and SOA A are as inputs of the algorithm $Soa2Soire$, let $delta$ denote the derived expression. According to the computation of $filter$, the alphabet symbols in δ are in the same maximum independent set mis. Then in line 9, for each $mis_i \in all\_mis$, the corresponding expression δ_i is generated. ($S'_i = filter(mis_i, S)$; SOA A_i=2T-INF(S'_i); $\delta_i = Soa2Soire(S'_i, A_i)$;) δ_i is put into U, then $combine(U) = \delta_1\&\delta_2\&\cdots\&_i$.
   For expression $a\&(b(c\&d))$, $mis_1 = \{a\}$, $mis_2 = \{b, c, d\}$, $mis_3 = \{c\}$ and $mis_4 = \{d\}$. Then for the initially constructed SOA, since b, c, d can form a maximum independent set mis_2, then for $mis_3 = \{c\}$ and $mis_4 = \{d\}$ 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 $Soa2Soire$.

Conclusion 1 and conclusion 2 have revealed that the expression learnt by $Soa2Soire$ still has many restrictions for using interleaving. Such as the expressions of form $(a|b\&c)^+$, $(a\&b)^+$, $((a|b)\&(c|d)\&(e|f))^*$ 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 $Soa2Soire$ is not an SOIRE.

3 The correct result learnt by iSOIRE

In Section 2, we have proved that the expression learnt by $Soa2Soire$ is not an SOIRE. However, we should check whether the expression returned by 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 Σ by the following grammar, and where every terminal symbol occurs at most once.*

$$P := SP \Big| PS \Big| S \Big| T \Big| P|S \tag{1}$$

$$S := S\&S \Big| T \tag{2}$$

$$T := T|T \Big| TT \Big| \varepsilon \Big| a \Big| a^*(a \in \Sigma) \tag{3}$$

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 ε, a, a^*, in algorithm $Soa2Soire$, in line 2, line 11 $\sim$ line 16 and line 7, they can be derived, the correctness can be ensured by the corresponding correctness of algorithm $Soa2Sore$.

(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 $\sim$ line 16 and line 22 $\sim$ 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 $Soa2Sore$.

(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 $Soa2Soire$.
   (1) For grammar (3), the corresponding generated regular expressions can be derived by $Soa2Soire$, the correctness can be ensured by the corresponding correctness of algorithm $Soa2Sore$.
   (2) For grammar (2), according to the proofs in 1, the corresponding generated regular expressions with interleaving can also be derived by $Soa2Soire$.
   (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 $Soa2Soire$.
   This implies that, for an expression r' generated by the defined grammars, r' can be learnt by $Soa2Soire$.

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

4 Conclusion

In this paper, we mainly provide analyses about the incorrectness about algorithm iSOIRE, and then present the correct a class of expressions can be learnt by algorithm iSOIRE, the corresponding proofs illustrate that the learnt expression belongs to a subclass of SOIREs. Since the algorithm iSOIRE 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 iSOIRE. 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. Li, Y., 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. Li, Y., 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)