APPENDIX

A. THE DETAILS OF PROOFS

A.1 Proof of Proposition 1

PROOF. According to Proposition 18 in [15], if Σ is a finite alphabet of n alphabet symbols, the number of pairwise non-equivalent SOREs over Σ is $s(n)$ with $n!2^{3n-r\log n} \leq s(n) \leq n!2^{7n}$, where r is a constant. This implies there is a finite number of non-equivalent SOREs.

For k-OREs, every symbol in Σ occurs at most k times, We treat the same symbol in a k-ORE as distinct, then, let Σ_k for k-OREs be a finite alphabet of nk alphabet symbols, the number of pairwise non-equivalent k-OREs over Σ_k is $s(nk)$ with $(nk)!2^{3nk-r\log(nk)} \leq s(nk) \leq (nk)!2^{7nk}$ This implies there is a finite number of non-equivalent k-OREs over Σ_k. Dk-OREs, which is the class of deterministic k-OREs, are subclass of k-OREs, the number of non-equivalent Dk-OREs over Σ_k is also finite.

Let $\mathcal{D} \in \{k\text{-OREs,D}k\text{-OREs}\}$. Assume that there is a language $L \subseteq \Sigma_k^*$ such that no expression $\alpha \in \mathcal{D}$ is $\mathcal{D}$-descriptive of L. If an expression $\alpha_1 \in \mathcal{D} : \mathcal{L}(\alpha_1) \supseteq L$, then there is an expression $\alpha_2 \in \mathcal{D} : \mathcal{L}(\alpha_1) \supset \mathcal{L}(\alpha_2) \supseteq L$. There are infinite expressions $\alpha_1, \alpha_2, \cdots, \alpha_i, \cdots \in \mathcal{D}$ such that $\mathcal{L}(\alpha_1) \supset \mathcal{L}(\alpha_2) \supset \cdots \supset \mathcal{L}(\alpha_i) \supset \cdots \supseteq L$. This contradicts the fact that there is only a finite number of non-equivalent k-OREs (and, hence, Dk-OREs) over Σ_k. Hence, for every language L, a k-ORE-descriptive k-ORE and a Dk-ORE-descriptive Dk-ORE must exist. $\square$

A.2 Proof of Theorem 1

PROOF. We first present some conclusions for the obtained SORE r, then the SORE r is proved to be a descriptive SORE by the derived conclusions. There are three conclusions for the r derived from *MinSore*. (1) r is a SORE. (2) $\mathcal{L}(r) \supseteq S$. (3) $\mathcal{L}(r)$ includes the minimum number of the strings, which are not recognized by the SOA A (SOA A_1 is referred to as SOA A for simplification).

For (1), an SOA as the input of the algorithm *MinSore*, contains distinct alphabet symbols as the label of the nodes. In algorithm *MinSore*, a regular expression is derived by modifying the SOA. And for every step, there are no duplicate alphabet symbols introducing into the label of a node in the SOA. Thus, a regular expression finally obtained from *MinSore* is a SORE.

We distinguish a number of different cases, depending on which clause was used for *MinSore*(A). Then, we conclude (2) and (3) by induction hypothesis. Since the labels of the nodes in a SOA are distinct, a node labelled with symbol a, is referred as node a. For a given finite language S, an SOA $A :=$**2T-INF**(S) and $r := MinSore(A)$, let $M(r, S, A) = 1$ denote the corresponding conclusions (2) and (3) are both hold.

Case 1: The clause in line 1 or the clause in line 2 was used. The case for (2) and (3) is trivial.

Case 2: The clause in line 3 was used. U is a strongly connected looped component of A, it indicates that, for a node a in U, there exists substrings $s_1 = a\cdots$ and $s_2 = \cdots a \cdots$ in S. B_0 is the SOA that U is extracted and processed by *bend*(), forbids the substrings such as s_1 and s_2 are both recognized. Let $r_0 :=$*MinSore*(B_0) and $r = r_0^+$, there exists the set S_{r_0} of the substrings extracted from S such that $B_0 =$**2T-INF**(S_{r_0}). Let SOA $A_u = A.extract(U)$, and S_u be the set of the substrings extracted from S such that $A_u =$**2T-INF**(S_u). For r_0, S_{r_0} and B_0, $M(r_0, S_{r_0}, B_0) = 1$ hold by induction, then, for r, S_u and A_u, $\mathcal{L}(r) = \mathcal{L}(r_0^+) = \mathcal{L}(r_0)^+ \supseteq S_{r_0}^+ \supseteq S_u$. Because r_0 is a SORE, to ensure the expression derived from A_u is also a SORE, let $r = r_0^+$, then, the substrings such as s_1 and s_2 can be generated by r. The SORE r guarantees the minimum number of the strings, which are accepted by r but not recognized by the SOA A_u, Thus, $M(r, S_u, A_u) = 1$.

Case 3: The clause in line 8 was used. Let $a = v.label()$ and $U = A.exclusive(v)$, U is the set of the nodes, which can only be reached by passing node v from the source of A. U includes node v. Let $B_0 = A.extract(U)$ and $r_0 = MinSore(B_0)$, there exists the set S_0 of the substrings which are extracted from S such that $B_0 =$**2T-INF**(S_0). Let v' be the node formed by $A.contract(U, r_0)$, v' represents the identification of the set of the substrings, which occur in S if and only if they begin with the symbol a or a' ($a' \in fst(a)$ if a is an expression.). This implies $\mathcal{L}(r_0)$ does not contain the additional strings, which do not begin with the symbol a or a' and are not recognized by B_0. For r_0, S_0 and B_0, $M(r_0, S_0, B_0) = 1$ holds by induction.

Case 4: The clause in line 12 was used. This case is mainly to add a node labeled ε by *addEpsilon*(). However, in the current SOA A, we first identify the pairs of edges $(v_1, v_2) \in A.E$, where $v_1 \in A.first()$, $v_2 \in A.\succ(q_0) \backslash A.first()$, and v_1 can reach the nodes which are not the successors of $A.q_0$. In addition, v_1 and v_2 are in the same strongly connected looped component (*sclc*) U. Let $V_s = \{v | v \in (A.\succ(v_1) \cup A.\succ v_2) \backslash (A.\succ(q_0) \cup \{A.q_f\})\}$, V_s is the set of the successors of the nodes v_1 and v_2, but not including $A.q_f$ and the successors of $A.q_0$. We remove edges (v_1, v_2). Otherwise, the clause in line 22 or 19 will be used, and then the clause in line 29 will be used, the expression $v_1.label()?v_2.label()?$ will be produced. Then, for nodes $v \in V_s$, v_1 and v_2, they are processed by *MinSore* to form a new node v', the corresponding label $v'.label()$ (i.e., an expression) will generate the strings not recognized by SOA A that each of them begins the symbol $l \in fst(v.label())$ instead of $l \in fst(v_1.label())$ or $l \in fst(v_2.label())$.

The edges (v_1, v_2) are removed[10]. Because v_1, v_2 ($v_1, v_2 \in A.first()$) and their successors $v \in V_s$ are in the same *sclc* U, according to the clause in line 3, the subexpression $v_1.label()|v_2.label$ under the iteration $^+$ will be computed if the clause in line 29 is used. And each successor $v \in V_s$ of the nodes v_1 and v_2 is processed by *MinSore*, according to the clause in 16, the edge $(v_1, A.q_f)$ is added, the node labeled ε will be introduced by *addEpsilon*() for the clause in 22. This implies that, for each successor $v \in V_s$ and its label, $v.label()?$ will be produced when the clause in line 29 is used. Then the concatenation $v_1.label()v_2.label()$ can be formed by the iteration $^+$, and can recognize any substrings in S generated by them. That the edges (v_1, v_2) are removed not only ensures a minimum number of the strings not recognized by the SOA A, but also guarantees the strings occurring in S can be dervied by $v_1.label()v_2.label()$. Let $A' = A.extract(U)$, S' is the set of the strings extracted from S such that $A' =$**2T-INF**(S'), By induction, the

[10]The edges (v_1, v_2) are removed such that the SOA A does not derive the expression of form $v_1.label()?v_2.label()?$, which generates the strings that begin with symbol $a \in fst(v.label)$ ($v \in V_s$) and are not recognized by the SOA A.

generated expression $v'.label()$ supports the corresponding conclusions (2) and (3), i.e., $M(v'.label(), S', A') = 1$.

Case 5: The clause in line 23 was used. Let v be the only successor of $A.q_0$, and $a = v.label()$. Let SOA A_a be the input of *MinSore* to generate a. Let SOA B_0 be the SOA which is obtained by $A.contract(\{A.q_0, v\}, q_0)$, and $r_0 = MinSore(B_0)$. There exists the set S_0 (resp. S_a) of the substrings, which can be extracted from S such that B_0 =**2T-INF**(S_0) (resp. A_a =**2T-INF**(S_a)). For $\mathcal{L}(r_0)$ (resp. $\mathcal{L}(a)$), S_0 (resp. S_a) and B_0 (resp. A_a), the correspond conclusions (2) and (3) hold by induction. i.e., $M(r_0, S_0, B_0) = 1$ and $M(a, S_a, A_a) = 1$. Then, let $r = concatenate(a, r_0) = ar_0$, for the language $S \subseteq S_aS_0$, $\mathcal{L}(r) = \mathcal{L}(ar_0) = \mathcal{L}(a)\mathcal{L}(r_0) \supseteq S_aS_0 \supseteq S$ for $\mathcal{L}(a) \supseteq S_a$ and $\mathcal{L}(r_0) \supseteq S_0$. This implies $\mathcal{L}(r)$ and S support conclusion (2). And the SOA A can be obtained by concatenating A_a with B_0, $\mathcal{L}(a)$ and $\mathcal{L}(r_0)$ support conclusion (3) for the SOA A_a and the SOA B_0, respectively. Then $\mathcal{L}(r)$ supports conclusion (3) for the SOA A. i.e., $M(r, S, A) = 1$.

Case 6: The clause in line 29 was used. Let u and v as chosen in line 29, and let $a = u.label()$ and $b = v.label()$. There exist corresponding samples S_a, S_b and SOAs A_a, A_b such that $\mathcal{L}(a)$ and $\mathcal{L}(b)$ both support conclusions (2) and (3) by induction. i.e., $M(a, S_a, A_a) = 1$ and $M(b, S_b, A_b) = 1$. $u, v \in A.first()$, this implies the strings accepted by the current SOA A begin with the substrings, which are either generated by a or generated by b. Let $r = or(u.label(), v.label()) = or(a, b) = a|b$. Then, $\mathcal{L}(r) = \mathcal{L}(a) \cup \mathcal{L}(b) \supseteq S$. $\mathcal{L}(a)$ with respect to the SOA A_a and $\mathcal{L}(b)$ with respect to the SOA A_b support conclusion (3), respectively, then for $\mathcal{L}(r)$ and the SOA A, the corresponding conclusion (3) holds. i.e., $M(r, S, A) = 1$.

The all cases have been analyzed. By induction, for the given finite language S and the corresponding SOA A, the final SORE r is obtained from $MinSore$, $\mathcal{L}(r)$ with respect to S (resp. with respect to SOA A) supports the conclusion (2) (resp. conclusion (3)). i.e., $M(r, S, A) = 1$. Assume r is not a descriptive SORE for S, then there exists a SORE δ such that $\mathcal{L}(r) \supset \mathcal{L}(\delta) \supseteq S$. Because of $\mathcal{L}(\textbf{2T-INF}(\mathcal{L}(r))) = \mathcal{L}(r)$ and $\mathcal{L}(\textbf{2T-INF}(\mathcal{L}(\delta))) = \mathcal{L}(\delta)$ (see Lemma 9 in [15]). And **2T-INF**(S) is SOA-descriptive of S [15], $\mathcal{L}(\textbf{2T-INF}(S))$ contains the all strings, which are recognized by the SOA A. Then, $|\mathcal{L}(r) \setminus \mathcal{L}(\textbf{2T-INF}(S))| > |\mathcal{L}(\delta) \setminus \mathcal{L}(\textbf{2T-INF}(S))|$, this implies $\mathcal{L}(r)$ does not include the minimum number of the strings, which is not recognized by the SOA A. This is a contradiction for the SORE r that $\mathcal{L}(r)$ with respect to the SOA A supports the conclusion (3). Thus, the above assumption does not hold, then the SORE r is a descriptive SORE for S. □

A.3 Proof of Theorem 2

PROOF. The initial constructed k_0-OA A_{k_0} for a given finite sample S exactly recognize S, i.e., $\mathcal{L}(A_{k_0}) = S$ which holds if only if $S \subseteq \mathcal{L}(A_{k_0})$ and $\mathcal{L}(A_{k_0}) \subseteq S$.

(1) $S \subseteq \mathcal{L}(A_{k_0})$. For a string $s \in S$, first, the suffix s_{ls} of s is computed, and then $s!s_{ls}$ is obtained. s is decomposed into the substrings s_1 and s_2, where $s_1 = s!s_{ls}$, $s_2 = s_{ls}$ and $s = s_1s_2$. Let $\widehat{S} = \{s!s_{ls}|s \in S\}$, the PTA G_p (resp. PTA G'_p) is constructed for $\widehat{S}$ (resp. s_{ls}). According to the definition of PTA, $s_1 \in \mathcal{L}(G_p)$ and $s_{ls} \in \mathcal{L}(G'_p)$. In Algorithm 3, PTAs G_p and G'_p are connected to a graph G, then $s = s_1s_2 \in \mathcal{L}(G)$. Some equivalent states in G are merged in lines 18∼22, G is transformed to a k_0-OA A_{k_0}, then $\mathcal{L}(G) = \mathcal{L}(A_{k_0})$, therefore, $s = s_1s_2 \in \mathcal{L}(A_{k_0})$. This implies that $\forall s \in S : s \in \mathcal{L}(A_{k_0})$. Thus, $S \subseteq \mathcal{L}(A_{k_0})$.

(2) $\mathcal{L}(A_{k_0}) \subseteq S$. Consider a path (string) $p \in A_{k_0}$, in Algorithm 3, before some equivalent states are merged in lines 18∼22, the constructed PTAs G_p and G'_p are connected to a graph G. For the graph G, a path in graph A_{k_0} also exists in the graph G, then the path $p \in \mathcal{L}(G)$. For graphs G_p and G'_p, there exist paths p_1 and p_2 such that $p_1 \in \mathcal{L}(G_p)$, $p_2 \in \mathcal{L}(G'_p)$ and $p = p_1p_2$. For $p_1 \in \mathcal{L}(G_p)$ (resp. $p_2 \in \mathcal{L}(G'_p)$), there exists $s_1 \in \widehat{S} : s_1 = p_1$ (resp. $s_2 = s_{ls}$). For a string $\widehat{s} \in \widehat{S}$, $\widehat{s}s_{ls} \in S$. Then, $p = p_1p_2 = s_1s_2 \in S$. This implies that $\forall p \in \mathcal{L}(A_{k_0}) : p \in S$. Thus, $\mathcal{L}(A_{k_0}) \subseteq S$.

A_{k_0} is a deterministic k_0-OA. For the PTAs G_p and G'_p, they are deterministic by definition. In line 4 of Algorithm 3, S_1 is obtained that G_p and G'_p are connected to form a deterministic k-OA ($k \geq k_0$) G. The finally k_0-OA A_{k_0} is obtained by merging some equivalent states in G in lines 18∼22, The obtained k_0-OA A_{k_0} is also deterministic. □

A.4 Proof of Theorem 3

PROOF. For any given finite sample S and value of k, according to the algorithm $ConsK$-OA(S, k), proofs are provided by distinguishing a number of different subroutines, which were used in $ConsK$-OA. Each subroutine has a minimal generalization for processing the current MA such that a descriptive k-OA (w.r.t. the class of deterministic k-OA) can be finally obtained.

$ConsK_0$-OA. Initially, the deterministic k_0-OA A_{k_0} is constructed for S, according to Theorem 2, $\mathcal{L}(A_{k_0}) = S$. If $k \geq k_0$, then A_{k_0} is returned as the constructed deterministic k-OA. A_{k_0} is descriptive of S (w.r.t. the class of deterministic k-OA).

If $k < k_0$, then to obtain the deterministic k-OA A_k, some states in A_{k_0} will be merged by calling subroutines $Determine$, $mergeMA$, $minMerge$, $MergeSym$ and $MergeEq$.

$Determine$. If an MA M is a non-deterministic MA, then $Determine$ returns a deterministic MA. Otherwise, $Determine$ returns M directly. Some states are compulsory to be merged for deterministic MA. The produced generalization for the current MA is inevitable.

mergeMA. *mergeMA* merges the specified set V of states in an MA $\mathscr{A}$. Assume that, the states in V are merged such that the finally returned MA has a minimal generalization, or the states in V are compulsory to be merged for deterministic MA.

$minMerge$. First, the specified pair of states (v_1, v_2) ($(v_1, v_2) \in D_a$) is searched. Condition (1) ensures that v_1 and v_2 are merged by *mergeMA* such that the returned MA has a minimal generalization. Condition (2) ensures that more states, whose labels use the symbols in $\mathscr{A}.\mathcal{S}$, can be merged such that the number of the states in the finally returned MA $\mathscr{A}$ is as small as possible. Condition (3) ensures that v_1 and v_2 ($[v_1v_2]_a^{\mathscr{A}}$) are chosen such that more states (whose labels use symbol a) can be merged. Then, v_1 and v_2 are merged by calling *mergeMA*.

For condition (1), $gdn(v_1, v_2)$ measures the generalization degree of the new formed MA after v_1 and v_2 were merged. More states can be merged according to conditions (2) and (3). If it is just states v_1 and v_2 merged to state v, then

$v.m = v_1.m \cup v_2.m$, and

$$gdn(v_1,v_2) = \sum_{u\in v.m} in(u)(\sum_{u\in v.m} out(u) - \sum_{a\in\Sigma} (\sum_{u\in Succ(v.m)} in_a(u) - 1)) - \sum_{u\in v.m} in(u)out(u). \quad (1)$$

Otherwise,

$$gdn(v_1,v_2) = gdn(v_1,v_2) + \sum_{(w_1,w_2)\in W} gdn(w_1,w_2). \quad (2)$$

where W is set of the pairs of states specified to be merged. According to Equation 1, it is easy to prove that Equation 2 holds. *minGdn* returns the set of pairs of states, where each pair of states (v_1,v_2) such that $gdn(v_1,v_2)$ has the minimum value for the pairs of states in specified set D_a. Thus, the searched states v_1,v_2 are merged by *mergeMA* has a minimal generalization for the current obtained MA, and *Determine* is possibly called, such that the finally returned MA has a minimal generalization by induction.

MergeSym. In *MergeSym*, P_l is the set of pairs of states. Each pair of states (v_1,v_2) in P_l is a direct edge in the current MA. While, P_l' is the set of pairs of states, for each pair of states (v_1,v_2) in P_l', v_2 is reachable from v_1, but (v_1,v_2) is not a direct edge in the current MA.

Let the current MA be $\mathscr{A}$. If Step (1) is chosen, then the pair of states (v_1,v_2) is selected from P_l that there is a longest path from $\mathscr{A}.q_0$ to v_2. v_1 and v_2 are merged to a new node v_1 such that a self loop is produced. Compared with the possibly produced the strongly connected loop component, self loop has a minimal generalization for MA $\mathscr{A}$.

If Step (2) is chosen, then subroutine *minMerge* is called, the returned MA has a minimal generalization by induction.

If Step (3) is chosen, then the pair of states (v_1,v_2) is selected from P_l' that there is a longest path from $\mathscr{A}.q_0$ to v_2. v_1 and v_2 are merged to a new node v_1 such that a strongly connected loop component is produced. Here, there does not exist any pairs of states that can be merged in step (1) and Step (2). i.e., v_1 and v_2 are compulsory to be merged for the states required to be merged for deterministic k-OA. Note that, there is a longest path from $\mathscr{A}.q_0$ to v_2. The produced strongly connected loop component has a minimal generalization for MA $\mathscr{A}$.

MergeEq. Let the current MA be $\mathscr{A}$. *MergeEq* returns an MA by recursively merging the two equivalent states in the MA $\mathscr{A}$. The two equivalent states are merged such that there is no generalization for MA $\mathscr{A}$.

Therefore, we prove all subroutines processing the current MA has a minimal generalization except the compulsory generalization for constructed deterministic k-OA. Since the initial constructed k_0-OA is equivalent to the given finite sample S, and the labels of the states in MA are converted to the corresponding labels in a k-OA, thus, the constructed deterministic k-OA has a minimal generalization for the given finite sample S, i.e., the constructed deterministic k-OA A_k is descriptive of S (w.r.t. the class of deterministic k-OA). □

A.5 Proof of Corollary 1

Proof. For any given finite sample S and value of k, a k-OA $A_k := ConsK\text{-OA}(S,k)$, according to Theorem 3, the deterministic k-OA A_k is descriptive of S (w.r.t. the class of deterministic k-OAs), then $\mathcal{L}(A_k) \supseteq S$. A k-ORE r_k is descriptive of S, then $\mathcal{L}(r_k) \supseteq S$. Consider $\overline{A}_k = marking(A_k)$, for a string $s = a_1 \cdots a_{|s|} \in S$, where $a_i \in \Sigma$ $(1 \le i \le |s|)$, there is a accepting run $q_0 a_1 \cdots a_{|s|} q_f$ in the k-OA A_k, the corresponding accepting run in $\overline{A}_k$ is $q_0 \overline{a_1} \cdots \overline{a_{|s|}} q_f$. Let $\overline{s} = \overline{a_1} \cdots \overline{a_{|s|}}$, then each $s \in S$, there exists a $\overline{s} = marking(s)$. Let $\overline{S} = \{\overline{s} | s \in S\}$, then $\overline{S} \subseteq \mathcal{L}(\overline{A}_k)$. Similarly, there exist marks such that $\overline{S} \subseteq \mathcal{L}(\overline{r_k})$

Therefore, $\overline{r_k}$(resp. $\overline{A}_k$) can be regarded as a SORE (resp. SOA), the k-ORE r_k is descriptive of S, then, the SORE $\overline{r_k}$ is descriptive of $\overline{S}$. The deterministic k-OA A_k is descriptive of S, then, the SOA $\overline{A}_k$ is descriptive of $\overline{S}$. According to Corollary 17 in [15], if the SORE $\overline{r_k}$ is descriptive of $\overline{S}$, then $\mathcal{L}(\overline{r_k}) \supseteq \mathcal{L}(\overline{A}_k)$. Thus, $\mathcal{L}(r_k) \supseteq \mathcal{L}(A_k)$. □

A.6 Proof of Theorem 4

Proof. For any given finite sample S and value of k, k-OA $A_k := ConsK\text{-OA}(S,k)$, then, according to Theorem 3, the deterministic k-OA A_k is descriptive of S (w.r.t. the class of deterministic k-OAs), then $\mathcal{L}(A_k) \supseteq S$. $r_k := Koa2Kore(A_k)$.

(1) $\mathcal{L}(r_k) \supseteq S$. $\overline{A}_k = marking(A_k)$, for $s \in S$, $s \in \mathcal{L}(A_k)$. For $\overline{A}_k$, there exist marks $\overline{s} = marking(s)$ such that $\overline{s} \in \overline{A}_k$. Let $\overline{S} = \{\overline{s} | \overline{s} \in \overline{A}_k, s \in S\}$, then for a string $\overline{s} \in \overline{S}$, $\overline{s} \in \overline{A}_k$. Therefore $\overline{S} \subseteq \overline{A}_k$. For sample $\overline{S}$, $\overline{A}_k$ is also a *SOA*-descriptive SOA.

In algorithm *Koa2Kore*, the SORE r is derived from the SOA $\overline{A}_k$ by using algorithm *MinSore*. For the sample $\overline{S}$, r is a SORE-descriptive SORE. Then, according to Corollary 17 in [15], $\mathcal{L}(r) \supseteq \mathcal{L}(\overline{A}_k)$. The k-ORE $r_k = \overline{r}$, therefore, $\mathcal{L}(r_k) \supseteq \mathcal{L}(A_k) \supseteq S$.

(2) r_k is k-ORE-descriptive of S. Assume that the k-ORE r_k is not descriptive of S, there exists a k-ORE δ_k such that $\mathcal{L}(r_k) \supset \mathcal{L}(\delta_k) \supseteq S$. For sample $S' = \overline{S}$, $\mathcal{L}(\overline{r_k}) = S'$, then there exist unified marks such that $\mathcal{L}(\overline{r_k}) \supset \mathcal{L}(\overline{\delta_k}) \supseteq S'$. $\overline{\delta_k}$ is a SORE. With respect to the class of SOREs, $\overline{r_k}$ is not a descriptive SORE for S'. There is a contradiction to the above conclusion that $\overline{r_k}$ is SORE-descriptive of $\overline{S}$. Thus, the k-ORE r_k is descriptive of S.

(3)If $A_k = A_{k_0}$, then $\mathcal{L}(r_k) = S$. According to Theorem 2, $\mathcal{L}(A_k) = S$, the k-ORE r_k is descriptive of S, then $\mathcal{L}(r_k) \supseteq \mathcal{L}(A_k) = S$. the k-ORE r_k is also descriptive of $\mathcal{L}(A_k)$. However, the k-ORE r_k is transformed from the k-OA A_k by using algorithm *MinSore*. Since $A_k = A_{k_0}$, the k-OA is constructed by connecting two PTAs built for S, and merging some equivalent states. Then, the graph A_k does not include strongly connected looped components, and each node v in A_k, which satisfied that $v_1 \to v$, $v \to v_2$ and $(v_1,v_2) \in A_k.E$, is not associated with any income edges. According the procedures in algorithm *MinSore*, the operator $^+$ does not be introduced into a k-ORE, and the operator ? does not be introduced into the subexpressions of a k-ORE, which should have been exactly matched by the substrings occurring in S. This implies there are no any generalizations in converting the k-OA to a k-ORE. Thus, $\mathcal{L}(r_k) = \mathcal{L}(A_k) = S$. □

A.7 Proof of Corollary 2

Proof. For any given finite sample S and value of k, $A_k := ConsK\text{-OA}(S,k)$, let $r_k := Koa2Kore(A_k)$, according to Theorem 3, the deterministic k-OA A_k is descriptive of S (w.r.t. the class of deterministic k-OAs). According to

Theorem 4, r_k is a k-ORE-descriptive k-ORE for S. Then, Corollary 1 shows that there is $\mathcal{L}(r_k) \supseteq \mathcal{L}(A_k) \supseteq S$.

If the k-OA A_k is a Glushkov representation of a target k-ORE r, i.e., $\mathcal{L}(A_k) = \mathcal{L}(r)$. Then, $\mathcal{L}(r_k) \supseteq \mathcal{L}(r) \supseteq S$. If $\mathcal{L}(r_k) \supset \mathcal{L}(r) \supseteq S$, then r is also a k-ORE such that r_k is not a descriptive k-ORE for S. A contradiction to Theorem 4. Thus, $\mathcal{L}(r_k) = \mathcal{L}(r)$, $Koa2Kore(A_k)$ is equivalent to r $\square$

A.8 Proof of Theorem 5

PROOF. For any given finite language S, $r_k := \mathit{InfKore}(S)$, the subroutine *Select*, which is required to select a deterministic expression, guarantees that the finally returned r_k is a deterministic k-ORE. Then assume that r_k is not descriptive of S (w.r.t. the class of deterministic k-OREs). There exists a deterministic k-ORE δ_k such that $\mathcal{L}(r_k) \supset \mathcal{L}(\delta_k) \supseteq S$. However, in algorithm *InfKore*(S), a k-ORE $r_k = Koa2Kore(A_k)$, where $A_k = \mathit{ConsK}\text{-OA}(S, k)$. According to Theorem 4, the k-ORE r_k is k-ORE-descriptive of S. δ_k is also a k-ORE. Then, with respect to the class of k-OREs, There is a contradiction for $\mathcal{L}(r_k) \supset \mathcal{L}(\delta_k) \supseteq S$. Thus, a deterministic k-ORE derived by *InfKore* is descriptive of S (w.r.t. the class of deterministic k-OREs). $\square$