Learning restricted regular expressions with interleaving from XML data

1 Inference Algorithm

For the set of given sample S, $L(SOA(S))$ is a minimal-inclusion generalization of S using *2T-INF* [2]. Finding a *maximum independent set* (MIS) for a graph G is a NP-hard problem. Hence we use the method *clique_removal*() [1] to find an approximate result. *all_mis* is the set contained all MISs iteratively obtained from G. $sym(A)$ is the set of all symbols occurring in A. $G.setln()$ is to assign each node a level number ln. $G.isSL(i)$ returns $true$ if i is a skip level and $false$ otherwise. $Combine(V, \text{“}|\text{”})$ (or $Combine(V, \text{“}\&\text{”})$) is to combine all elements in V with union ($|$) (or interleaving ($\&$)) operator. The input is S and the output is an *ESIRE* R. The main procedures and the pseudo-code of the algorithm *GenESIRE* are as follows.

1. Construct a graph $G(V,E)$=$SOA(S)$ for S using *2T-INF* [2].
2. For each node v with a self-loop, rename it with v^+ and remove the self-loop.
3. For each non-trival strongly connected component NTSCC, call function *Repair*$(NTSCC, S)$. Replace the NTSCC with a new node and label it with the return value of *Repair*$(NTSCC, S)$. All relations with any node in NTSCC of G rebuild the relations with the new node.
4. Assign the level numbers for the new graph and compute all the skip levels.
5. Nodes of each level are converted into one or more chain factors.

Algorithm Analysis For a graph $G(V,E)$=$SOA(S)$, let n=$|V|$ and m=$|E|$. It costs time $O(n)$ to find all nodes with loops and $O(m+n)$ to find all NTSCCs. The time complexity of *clique_removal*() is $O(n^2 + m)$. For each NTSCC, computation of *all_mis* costs time $O(n^3 + m)$. For each mis, there is no NTSCCs at all. Hence *Repair()* only costs time $O(n^3+m)$. The number of NTSCCs in a SOA is finite. Then computing *all_mis* for all NTSCCs also costs time $O(n^3 + m)$. Assigning level numbers and computing the skip numbers will be finished in time $O(m + n)$. All nodes will be converted into specific chain factors of *ESIRE* in $O(n)$. Therefore, the time complexity of *GenESIRE* is $O(n^3 + m)$.

Theorem 1. *Let* α=$GenESIRE(SOA(S))$ *where* S *is a set of given sample. If there exists another ESIRE* β *such that* $S \subseteq L(\beta) \subset L(\alpha)$, $L(\beta)$=$L(\alpha)$ *must hold.*

Proof. We construct SOA for S, α, β as G_s, G_α, G_β respectively. Obviously, we have $sym(G_s)$=$sym(G_\alpha)$=$sym(G_\beta)$. Let α=$\alpha_1\alpha_2\cdots\alpha_n$, β=$\beta_1\beta_2\cdots\beta_m$. Now we first consider α_1. α_1 contains all nodes with with ln=1. We use V_S and V_T to denote the sets of nodes with each node containing only one terminal symbol and multiple terminal symbols, respectively.

1. $V_S \neq \emptyset$, V_T=$\emptyset$ and ln=1 is not a skip level.

Let V_S=$\{v_1,v_2,\cdots,v_k\}$. α_1=$(v_1|v_2|\cdots|v_k)$. According to the algorithm, for each

Algorithm 1 *GenESIRE(S)*

Input: A set of strings S
Output: An *ESIRE* R

Construct the $G(V, E) = SOA(S)$ using *2T-INF* [2];
Rename each node v with loop v^+ and remove the loop; For each $NTSCC$, call algorithm *Repair(NCSCC,S)*. Then we get a new one $G' = (V', E')$;
$G.setln()$; $R \leftarrow \varepsilon$; $ln = 1$;
while $ln \leq$ (ln *of G'.snk*) $- 1$ **do**
  $V_T \leftarrow$ all nodes with level number ln and $length(sym(v)) \geq 2$;
  $V_S \leftarrow$ all nodes with level number ln and $length(sym(v))=1$;
  $A \leftarrow Combine(V_S$, "|"); $B \leftarrow Combine(V_T$, "|");
  if $A \neq \emptyset$ and $B = \emptyset$ **then**
    if $!G.isSL(ln)$ **then**
      $R \leftarrow R \cdot A$;
    else
      $R \leftarrow R \cdot A^?$;
    end if
  end if
  if $A = \emptyset$ and $B \neq \emptyset$ **then**
    if $!G.isSL(ln)$ **then**
      $R \leftarrow R \cdot B$;
    else
      $R \leftarrow R \cdot B^?$;
    end if
  end if
  if $A \neq \emptyset$ and $B \neq \emptyset$ **then**
    $R \leftarrow R \cdot A^? \cdot B^?$;
  end if
end while
return R

Algorithm 2 *Repair(V,S)*

Input: A set of nodes V and a set of given sample S
Output: A regular expression $newRE$

$pattern \leftarrow V$; $S' \leftarrow \bigcup_{s \in S} Filter(pattern, s)$; Compute sets $CS(S')$, $NCS(S')$ using $POR(S')$;
if $CS(S') == \emptyset$ **then**
  return $(Graph(CS).combine(V))^+$;
else
  $G = Graph(CS)$;
  while $G.nodes() \neq \emptyset$ **do**
    $v = clique\_removal(G)$; $G = G \backslash v$; $all\_mis.append(v)$;
  end while
  for each $mis \in all\_mis$ **do**
    $sub\_ex = GenESIRE(\bigcup_{s \in S} Filter(mis, s))$; $RE_{mis}.append((\varepsilon \in S')?\ sub\_ex^? : sub\_ex)$;
  end for
  $newRE \leftarrow Combine(RE_{mis}$, "&");
  return $newRE$
end if

node v_i there is an edge connected with *src* and no edge between any two nodes v_i and v_j. Now we prove $sym(\alpha_1)$=$sym(\beta_1)$. If there is a symbol $a \in sym(\alpha_1)$ but $a \notin sym(\beta_1)$, there exists some string $s_0 \in S$ and $s_0 \in L(\alpha)$ started with a. However, $s_0 \notin L(\beta)$ which causes a contradiction with $S \subseteq L(\beta)$. If there is a symbol $a \in sym(\beta_1)$ but $a \notin sym(\alpha_1)$, with similar analysis, it will lead to a contradiction with $L(\beta) \subset L(\alpha)$. Therefore, $sym(\alpha_1)$=$sym(\beta_1)$ which can also hold on for the following cases.

If there are concatenation ($\cdot$) or interleaving (&) operators in β_1, there must be edges between two nodes v_i, v_j to illustrate the occurrence orders. However, this will generate strings not in $L(\alpha)$, which is a contradiction with $L(\beta) \subset L(\alpha)$. For unary operators, each symbol in β_1 must be the same with α_1, otherwise it will lead to a contradiction with $L(S) \subset L(\beta)$ or $L(\beta) \subset L(\alpha)$. Therefore, we have α_1=β_1 and $L(\alpha_1)$=$L(\beta_1)$.

2. $V_S \neq \emptyset$, $V_T = \emptyset$ and ln=1 is a skip level.
Let $V_S = \{v_1, v_2, \cdots, v_k\}$. $\alpha_1 = (v_1|v_2|\cdots|v_k)^?$. According to the algorithm *GenESIRE*, there must be edges for *src* to some node $v \in V \backslash \{src\} \backslash V_S$. Similar with the proof in case 1, we know that β_1 must be added with optional operator ?, otherwise $L(\beta)$ can not cover the whole set S which is a contradiction. Therefore, we have α_1=β_1 and $L(\alpha_1)$=$L(\beta_1)$.

3. $V_S = \emptyset$, $|V_T| \neq \emptyset$ and ln=1 is not a skip level.
Let $V_T = \{A_1, A_2, \cdots, A_k\}$ where A_i is the node consisting of multiple symbols. α_1=$(A_1|A_2|\cdots|A_k)$. Similar with the proof in case 1, we know that there is an edge from *src* to node A_i and there is no edge between any node A_i and A_j where $i, j \in [1, k]$. Symbols of any two nodes $sym(A_i)$ and $sym(A_j)$ can not occur in one string. Therefore we know that β_1 must be in the same form of β_1=$(B_1|B_2|\cdots|B_k)$ in order to satisfy the condition $L(S) \subseteq L(\beta) \subset L(\alpha)$. Suppose that B_i in β_1 is in some specific order, it is easy to conclude that $sym(A_i)$=$sym(B_i)$. Now we prove A_i=B_i.

According to the algorithm $Repair(S')$, A_i is a sequence of *ESs* connected with interleaving &. Suppose that A_i=$s_i^1 \& s_i^2 \& \cdots \& s_i^q$ where s_i^p is an *ES* and $p \in [1, q]$. Each s_i^p is a maximum independent set obtained by $Graph(CS)$. Symbols within s_i^p are ordered determined by *NCS* while symbols between s_i^p and s_i^r are unordered. Both *CS* and *NCS* are computed from $POR(S')$ (S'=$\bigcup_{s \in S} Filter(sym(A_i), s)$). Remember β is also an *ESIRE*. Suppose any symbols $a \in sym(s_i^p)$ and $b \in sym(s_i^r)$, if they were connected by concatenation operator instead of interleaving, only one partial order ($a \prec b$ or $b \prec a$) would appear in strings generated by $L(\beta)$. This is a contradiction with condition $S \subseteq L(\beta)$. If symbols a and b within s_i^p were connected by interleaving operator instead of concatenation or union, then partial orders $a \prec b$ and $b \prec a$ would both appear in strings in $L(\beta)$ while can not in $L(\alpha)$. This causes another contradiction with $L(\beta) \subset L(\alpha)$. Therefore we can conclude that β_1 is in the form of $g_i^1 \& g_i^2 \& \cdots \& g_i^q$ in which $sym(s_i^p)$=$sym(g_i^p)$ (g_i^p in β_1 is ordered in accordance with A_i) and symbols within g_i^p only have two kinds of binary operators: concatenation and union.

Next, the specific form of s_i^p is determined by calling the algorithm *GenESIRE* in which V_T=∅ and V_S≠∅. This is just the situation which has been proved in case 1 and case 2. We can easily concluded that s_i^p=g_i^p. Therefore, we have A_i=B_i, α_1=β_1 and $L(\alpha_1)$=$L(\beta_1)$.

4. V_S=∅, $|V_T|$≠∅ and ln=1 is a skip level.

Let $V_T = \{A_1, A_2, \cdots, A_k\}$ where A_i is the node consisting of multiple symbols. α_1=$(A_1|A_2|\cdots|A_k)^?$. Similar with the proof in case 2, there exists an edge from src to node $v \in V \setminus \{src\} \setminus V_T$. In order to accept all strings in S, β_1 must have optional operator also. Therefore, we have α_1=β_1 and $L(\alpha_1)$=$L(\beta_1)$.

5. V_S≠∅, $|V_T|$≠∅.

According to the analysis above, all factors have optional operator in α_1. β_1 must be in the same form in order to satisfy the condition $S \subseteq L(\beta) \subset L(\alpha)$. Therefore, $L(\alpha_1)$=$L(\beta_1)$.

From the analysis above, we can conclude that $L(\alpha_1)$=$L(\beta_1)$ holds. Then we consider α'=$\alpha_2 \cdots \alpha_n$ and β'=$\beta_2 \cdots \beta_m$. We first construct a new set of strings S'. For each string $s \in S$, if $sym(s) \cap sym(\alpha_1)$=∅, then $s \in S'$; otherwise, replace all alphabet in $sym(\alpha_1)$ as ε and add it to S'. We let $G_{S'}$=$SOA(S')$. Clearly, we can find if we merge α_1 together with src ($\alpha_1 \cdot src$) and consider it as the new src, then the new graph is the same with $G_{S'}$ and α'=$GenESIRE(G_{S'})$. We can use the same proof procedure as above until ln=n. We therefore can conclude that $L(\alpha')$=$L(\beta')$ and $L(\alpha)$=$L(\beta)$.

References

1. Boppana, R., Halldrsson, M.M.: Approximating Maximum Independent Set by Excluding Subgraphs. Bit Numerical Mathematics 32(2), 180–196 (1992)
2. Garcia, P., Vidal, E.: Inference of k-Testable Languages in the Strict Sense and Application to Syntactic Pattern Recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence 12(9), 920–925 (2002)