数理解析研究所講究録 1375 巻 2004 年 106-112

Polynomial-Time Identification of an Extension of Very Simple Grammars from Positive Data

東京大学大学院学際情報学府　吉仲 亮 (Ryo Yoshinaka)
ry@iii.u-tokyo.ac.jp
Graduate School of Interdisciplinary Information Studies, University of Tokyo

Abstract

In this paper, we study the learning efficiency of a subclass of simple deterministic languages. We define a superclass of very simple grammars and show that the class is identifiable in the limit from positive data by presenting an algorithm which updates its conjectures in polynomial time in the size of provided data. Moreover, we investigate an algorithm which decides the inclusion problem for the class efficiently, which is a sub-algorithm of the learning algorithm.

1 Introduction

While there are few subclasses of context-free grammars (CFGs) known to be learnable efficiently from positive data only, Yokomori [6] shows that the class of very simple grammars (VSGs) is identifiable in the limit from positive data by an algorithm which updates its conjecture in polynomial time in the size of provided data. In this paper, we define a superclass of very simple grammars, called the *right-unique simple grammars (RSGs)* and show that the class is also efficiently learnable by an algorithm, which is based on the learning algorithm for VSGs by Yokomori. Our algorithm has a sub-algorithm which decides the inclusion whether $L(G') \subseteq L(G)$ between an RSG G and an arbitrary CFG G', which is based on (and runs faster than) the algorithm for the inclusion problem for VSGs proposed by Wakatsuki and Tomita [5].

For more detailed discussion (in particular for the proofs of the lemmas and theorems omitted in this paper) on the issues treated in this paper, see the master's thesis by Yoshinaka [7], from which this paper excerpts. The thesis includes a slight improvement on the algorithm for the inclusion problem for RSGs.

2 Preliminaries

ε denotes the empty string and $\varnothing$ denotes the empty set. $|\cdot|$ denotes the length of the string or the cardinality of the set. A context-free grammar (CFG) G is denoted by a 4-tuple (N, Σ, P, S) where N is the finite set of nonterminal symbols, Σ is the finite set of terminal symbols disjoint from N, P is the finite set of production rules and $S \in N$ is the start symbol. Nonterminals are denoted by upper case letters from the beginning of the alphabet, $A, B, \ldots$ etc. and terminals are denoted by lower case letters from the beginning of the alphabet, $a, b, \ldots$ etc. Sequences of nonterminals are denoted by lower case letters from the beginning of the Greek alphabet, $\alpha, \beta, \ldots$ etc. and sequences of terminals (strings) are denoted by lower case letters from the end of the alphabet, $x, y, \ldots$ etc. We define a binary relation $\Rightarrow_G$ as $\Gamma \Rightarrow_G \Delta$ for $\Gamma, \Delta \in (\Sigma \cup N)^*$ iff (if and only if) $\Gamma = \Delta_1 A \Delta_3$, $\Delta = \Delta_1 \Delta_2 \Delta_3$ and $A \to \Delta_2 \in P$. We some times write $\Rightarrow$ instead of $\Rightarrow_G$ if no confusion occurs. $\overset{+}{\Rightarrow}$ is the transitive closure of $\Rightarrow$. $\overset{*}{\Rightarrow}$ is the reflexive and transitive closure of $\Rightarrow$. We define the language $L(G)$ by a grammar G as $L(G) = L(G, S)$ where $L(G, A) = \{w \in \Sigma^* | A \overset{*}{\Rightarrow} w\}$. A CFG G is reduced iff for every nonterminal $A \in N$, there is $xyz \in L(G)$ such that $S \overset{*}{\Rightarrow} xAz \overset{*}{\Rightarrow} xyz$.

Definition 1. A *positive data* of a language L is a surjection from the set of natural numbers $\mathbb{N}$ to L.

For a positive data R of L, let R_n denote $\{R(0), \ldots, R(n-1)\}$. An algorithm $\mathcal{A}$ *converges to* G *for* R iff there is $k \in \mathbb{N}$ such that $\mathcal{A}(R_n) = G$ for all $n \geq k$, where $\mathcal{A}(R_n) = G$ means that the output of $\mathcal{A}$ is G for the input R_n. We say that $\mathcal{A}$ *learns* a class $\mathcal{L}$ of languages if for any $L \in \mathcal{L}$ and any positive data R for L, $\mathcal{A}$ converges to G such that $L(G) = L$.

The term "the efficiency of a learning algorithm" is controversial. Since the size of the input data R_n increases infinitely, every learning algorithm can be modified into more "efficient" one which updates its conjecture faster. In order to make the discussion constructive, we introduce the following two common constraints, which is satisfied by our learning algorithm for RSGs.

Definition 2. A grammar G is *consistent* with L iff $L \subseteq L(G)$. A learning algorithm $\mathcal{A}$ is *consistent* iff the output is always consistent with the input, i.e., $R_n \subseteq \mathcal{A}(R_n)$ for every positive data R and a natural number n. An algorithm $\mathcal{A}$ is *conservative* iff $R(n) \in \mathcal{A}(R_n)$ implies $\mathcal{A}(R_{n+1}) = \mathcal{A}(R_n)$ for every R and n.

The learning algorithm for VSGs by Yokomori [6] updates its conjecture in $\mathcal{R}_n^{|\Sigma|}$ under the above two constraints, where $\mathcal{R}_n = \sum_{0 \le k < n} |R(k)|$ is the total length of the input data R_n.[1] In this paper, we define a superclass of VSGs and discuss its learning efficiency.

Definition 3. A CFG G is in *Greibach normal form* if every production rule is in the form of $A \to a\alpha$ for some $a \in \Sigma$ and $\alpha \in N^*$. A CFG G in Greibach normal form is a *simple grammar* iff $A \to a\alpha$, $A \to a\beta \in P$ implies $\alpha = \beta$. A simple grammar G is *very simple* iff $A \to a\alpha$, $B \to a\beta \in P$ implies $A = B$ and $\alpha = \beta$. A simple grammar G is *right-unique* iff $A \to a\alpha$, $B \to a\beta \in P$ implies $\alpha = \beta$.

The class of RSGs is a proper superclass of VSGs. The following example of an RSG expresses formulas of first order logic, which cannot be expressed by a VSG since a VSG cannot distinguish "variables" (represented by the nonterminal V below) from "terms" (by T).

Example 4. Let an RSG be such that $N = \{S, T, V, C, L, R\}$, $\Sigma = \{\mathtt{p}, \mathtt{q}, \mathtt{f}, \mathtt{g}, \mathtt{a}, \mathtt{b}, \mathtt{x}, \mathtt{y}, \neg, \vee, \exists, (,), .\}$, and $P = \{S \to \neg S,\ S \to \vee LSCSR,\ S \to \exists VS,\ S \to \mathtt{p}LTR,\ S \to \mathtt{q}LTCTR,\ T \to \mathtt{f}LTR,\ T \to \mathtt{g}LTCTR,\ T \to \mathtt{a},\ T \to \mathtt{b},\ T \to \mathtt{x},\ T \to \mathtt{y},\ V \to \mathtt{x},\ V \to \mathtt{y},\ C \to .,\ L \to (,\ R \to)\}$. Then, for example, there is a derivation $S \Rightarrow \neg S \Rightarrow \neg\exists VS \Rightarrow \neg\exists \mathtt{x}S \Rightarrow \neg\exists \mathtt{x}\mathtt{p}LTR \stackrel{*}{\Rightarrow} \neg\exists \mathtt{x}\mathtt{p}(T) \Rightarrow \neg\exists \mathtt{x}\mathtt{p}(\mathtt{g}LTCTR) \stackrel{*}{\Rightarrow} \neg\exists \mathtt{x}\mathtt{p}(\mathtt{g}(\mathtt{a}.\mathtt{x}))$.

Angluin [1] shows that the class of k-reversible languages is learnable from positive data efficiently for any natural number k, where a regular language L is *k-reversible* iff $\{x_1yz_1, x_2yz_1, x_1yz_2\} \subseteq L$ and $|y| = k$ implies $x_2yz_2 \in L$. While it is shown by Yokomori [6] that if L is a regular and very simple language then L is zero-reversible, in contrast, there is a regular and right-unique simple language L which is not k-reversible for any k, e.g., $L = \{ac^nde, ac^ndf, bc^nde | n \ge 0\}$ defined by an RSG as $P = \{S \to aCF, S \to bCE, C \to cC, C \to d, E \to e, F \to e, F \to f\}$.

Theorem 5. The class of RSGs is closed under none of the following; union, intersection, complement, concatenation, Kleene closure ($*$, $+$), (ε-free) homomorphism, inverse homomorphism, or reversal.

3 A Learning Algorithm for RSGs

Under the constraint of the consistency and conservatism, a learning algorithm must output a grammar which represents a minimal language in the languages including the input data. The following function defined on an RSG is very important to study the properties of RSGs.

Definition 6 (shape). For an RSG G, a function $\#_G$ mapping from $(\Sigma \cup N)^*$ to the set of integers $\mathbb{Z}$ is defined as follows;

- $\#_G(a) = |\alpha| - 1$ for $A \to a\alpha$
- $\#_G(A) = -1$ for $A \in N$
- $\#_G(\Gamma\Delta) = \#_G(\Gamma) + \#_G(\Delta)$ (homomorphism)

We call $\#_G$ the *shape* of G.

A function $\$_G : \Sigma^* \to \mathbb{N}$ is defined as follows;

- $\$_G(\varepsilon) = 0$
- $\$_G(x) = \max\{1 - \#_G(x') | x' \text{ is a proper prefix of } x\}$ for $x \neq \varepsilon$

If $\alpha \stackrel{*}{\Rightarrow} x\beta$, then $\#(\alpha) = \#(x\beta)$ (i.e., $\#(x) = |\beta| - |\alpha|$). $\$(x)$ denotes the necessary and sufficient length of a sequence of nonterminals to derive x by a left-most derivation, because $|\beta'| = |\alpha| + \#(x') \ge 1$ for $\alpha \stackrel{*}{\Rightarrow} x'\beta' \stackrel{+}{\Rightarrow} x\beta$. That is, $\alpha \stackrel{*}{\Rightarrow} x\beta$ implies $\alpha' \stackrel{*}{\Rightarrow} x\beta'$ for the prefix α' of length $\$(x)$ of α and $|\beta'| = \$(x) + \#(x)$.

Lemma 7 (right-uniqueness). For two derivations $\alpha_1 \stackrel{*}{\Rightarrow} x\beta_1$ and $\alpha_2 \stackrel{*}{\Rightarrow} x\beta_2$, if $|\alpha_1| = |\alpha_2| = \$(x)$, then $\beta_1 = \beta_2$.

[1] Yokomori additionally claims that the algorithm satisfies the condition for polynomial-time identification proposed by Pitt [4]. The author does not think the conclusion is false but has a question on his proof.

Definition 8 (consistent shape). The shape $\#$ of some RSG (a homomorphism $\#$ from Σ^* to $\mathbb{Z}$ such that $\#(a) \geq -1$ for all $a \in \Sigma$) is *consistent with a language* L iff there is an RSG G whose shape is $\#$ such that $L \subseteq L(G)$.

Lemma 9. A shape $\#$ is consistent with L iff (1) $\#(w) = -1$ and (2) $\#(w') \geq 0$ for all the proper prefixes w' of w for every $w \in L$. The condition (2) can be replaced with (2') $\$(w) = 1$, where $\$(x) = \max\{1 - \#(x') | x' \text{ is a proper prefix of } x\}$.

If $\#$ is consistent with L, then $xay \in L$ implies $\#(a) < |y|$.

Corollary 10. For a provided data R_n, all the consistent shapes can be enumerated in finite steps $O(\mathcal{R}_n^{|\Sigma|-1})$ ($\mathcal{R}_n = \sum_{0 \leq k < n} |R(k)|$).

In order to obtain the set of consistent shapes more fast, in practice, it is a good idea to construct and solve the simultaneous linear equations which represent the condition (1) of Lemma 9, that is, $\sum_{a \in \Sigma} (\mathrm{occ}(a, w) \cdot \#(a)) = -1$ for each $w \in R_n$ where $\mathrm{occ}(a, w)$ denotes the number of occurrences of a in w. But, unfortunately, such a strategy does not improve the worst-case time complexity theoretically.

Suppose that an input R_n and a consistent shape $\#$ are given. Let $\mathcal{G}_\# = \{G | \#_G = \#\}$ be the subclass of RSGs whose shapes are all $\#$. Then, it is easy to find the minimum grammar G_0 in $\mathcal{G}_\#$ such that $R_n \subseteq L(G_0)$ as follows. We assume that the right side of each rule of each grammar in $\mathcal{G}_\#$ is in the form of $?_a \to aA_{a,0} \ldots A_{a,\#(a)}$ and the set of nonterminals is $N_\# = \{S\} \cup \{A_{a,i} | a \in \Sigma, 0 \leq i \leq \#(a)\}$. This assumption loses no generality. Then, we determine the left side of the rules of G_0 by simulating the derivations of all $w \in R_n$. We can complete such a simulation without fail. For example, suppose a shape $\#(a, b, c, d) = (1, 0, -1, -1)$ given. The right side of the rules of G_0 is determined as

$$?_a \to aA_0A_1,\ ?_b \to bB_0,\ ?_c \to c,\ ?_d \to d.$$

If $abcbd \in R_n$, then we simulate the derivation as

$$S \Rightarrow aA_0A_1 \Rightarrow abB_0A_1 \Rightarrow abcA_1 \Rightarrow abcbB_0 \Rightarrow abcbd,$$

and therefore, the rules are determined as follows;

$$S \to aA_0A_1,\ A_0 \to bB_0,\ A_1 \to bB_0,\ B_0 \to c,\ B_0 \to d.$$

As seen above, the distinctions between RSGs in $\mathcal{G}_\#$ are what nonterminals are in $?_a$ for each $a \in \Sigma$. In other words, (the equivalence classes of) $\mathcal{G}_\#$ forms a finite Boolean algebra isomorphic to the power set $\mathcal{M}_\#$ of $\Sigma \times N_\#$. The correspondence between $G \in \mathcal{G}_\#$ and $M_G \in \mathcal{M}_\#$ is that $A_{b,k} \to aA_{a,0} \ldots A_{a,\#(a)}$ is in G iff $(a, A_{b,k}) \in M_G$. Then it is easy to verify that $L(G_1) \cap L(G_2) = L(G_1 \sqcap G_2)$ holds, where $\sqcap$ is defined as $G = G_1 \sqcap G_2$ iff $M_G = M_{G_1} \cap M_{G_2}$. This assures that if $\#$ is consistent with a language L then the minimum grammar G_0 in $\mathcal{G}_\#$ such that $L \subseteq L(G_0)$ is uniquely determined.

Summarizing the above, for a given input,

- We can enumerate shapes consistent with the input.
- We can compute the minimum consistent grammar with the input for a given shape consistent with the input.

The *minimum* grammar for a fixed shape is, however, not necessarily a *minimal* grammar in all the consistent RSGs with the input. There are a number of consistent shapes for the input. The remained task is to choose a *minimal* grammar in the *minimum* grammars. For this task, it is enough to show an algorithm which decides the inclusion of RSGs with different shapes.[2] We present such an algorithm in the next section.

Corollary 11. The inclusion problem for RSGs which are constructed from a provided data R_n and consistent shapes can be solved in $O(|\Sigma|^4 \mathcal{R}_n^{10})$ steps.

Therefore, we can choose a minimal grammar consistent with a given input. Moreover, it is easy to see that the algorithm converges to an RSG which represents the target language, because (1) the set $\mathcal{S}_n$ of shapes consistent with R_n is finite, (2) $\mathcal{S}_n \subseteq \mathcal{S}_{n+1}$ if all the terminals of the target grammar appear in R_n, and (3) finitely many essentially different grammars have a same shape ($\mathcal{M}_\#$ is finite).

Theorem 12. The algorithm in Figure 1 learns the class of right-unique simple grammars conservatively and consistently, which updates its conjecture in $O(|\Sigma|^4 \mathcal{R}^{|\Sigma|+9})$ steps.

[2] Yokomori [6] shows that we can choose a minimal VSG in a number of VSGs without deciding the inclusion. But such a method is not applicable to RSGs. That is a main difference between Yokomori's algorithm and ours.

Algorithm; Learning RSGs
Input $R_n = \{R(0), \ldots, R(n-1)\}$ and the previous conjecture G;
if $R(n-1) \in L(G)$ **then** output G and **halt**; **fi**
let $\mathcal{S} = \{\# \mid -1 \leq \#(a) < |y_a|\}$, where $|y_a| = \min\{|y| \mid xay \in R_n\}$;
eliminate inconsistent shapes with R_n from $\mathcal{S}$;
if $\mathcal{S} = \varnothing$ **then** output "this is not a right-unique simple language." and **halt**;
let $\mathcal{G}$ be $\varnothing$;
for $\#_i \in \mathcal{S}$ **do** construct the minimum grammar G_i whose shape is $\#_i$ and add G_i to $\mathcal{G}$; **od**
by comparing each grammar in $\mathcal{G}$, output a minimal grammar and **halt**;
End Algorithm

Figure 1: A Learning Algorithm for RSGs

4 An algorithm for Inclusion Problems of RSGs

It is already shown that the inclusion problem for some superclasses of right-unique simple grammars is decidable by Linna [3] and Greibach and Friedman [2]. But these algorithms, which take exponential time in $\mathcal{R}_n$, are not enough efficient to be adopted as a sub-algorithm of our learning algorithm. On the other hand, Wakatsuki and Tomita [5] show that the inclusion problem for VSGs is decidable in polynomial time in the *thicknesses* of compared grammars. The thickness τ_G of a CFG G is defined as $\tau_G = \max_{A \in N} \tau_G(A)$ for $\tau_G(A) = \min_{A \stackrel{*}{\Rightarrow} w} |w|$. Because the length of a shortest string in the language by a CFG cannot be bounded by a polynomial in the description size of the grammar generally, it is thought to be reasonable to adopt the thickness as a parameter for an evaluation of the computational complexity of an algorithm which treats CFGs. Based on their algorithm, we investigate an algorithm which decides the inclusion problem for RSGs more efficiently than theirs.

Theorem 13. For an RSG $G = (N, \Sigma, P, S)$ and an arbitrary CFG $G' = (N', \Sigma, P', S')$ in Greibach normal form, the question whether $L(G') \subseteq L(G)$ is decidable in $O(|\Sigma|^4|N|^2|G'|^6\tau_{G'}^2)$ by the algorithm in Figure 2, where $|G'|$ denotes the description size of G' defined as $|G'| = \sum_{A' \to a\alpha' \in P'}(3 + |\alpha'|)$.

Note that, though we assume that G' is in Greibach normal form here for a convenience, but that is inessential restriction at all.

Our algorithm checks whether $\#_G$ is consistent with $L(G')$ first. This is a necessary condition for $L(G') \subseteq L(G)$. Through this section, we assume that both G and G' are reduced.

Definition 14 (extended shape). Suppose that $\#_G$ is consistent with $L(G')$. In such a case, we can define $\#_G(A')$ for $A' \in N'$ by identifying A' as a string in $L(G', A')$.

- $\#_G(A') = \#_G(y)$ for some $A \stackrel{*}{\Rightarrow}_{G'} y$

This is well defined, because for different two strings $y_1, y_2 \in L(G', A')$ and a derivation $S' \stackrel{*}{\Rightarrow}_{G'} xA'z \stackrel{*}{\Rightarrow} xy_iz$, we obtain $\#_G(xy_iz) = -1$ and $\#_G(A') = \#_G(y_i) = -1 - \#_G(xz)$. In addition, we define $\$_G(A')$ as follows;

- $\$_G(\Gamma) = \max\{\$_G(y) | \Gamma' \stackrel{*}{\Rightarrow}_{G'} y\}$ for $\Gamma \in (\Sigma \cup N')^*$,
- $\$_G(A') = \max\{\$_G(y) | A' \stackrel{*}{\Rightarrow}_{G'} y\}$ in particular.

It is easy to see that $\$(A')$ and $\$(\Gamma)$ has a finite value if $\#_G$ is consistent with $L(G')$.

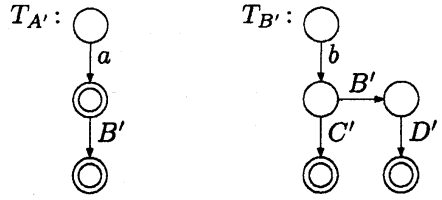
Lemma 15. Suppose that $\#_G$ is consistent with $L(G')$. Then,

- $\#_G(A') = \#_G(a\alpha')$ for all rules $A' \to a\alpha' \in P'$ of G',
- $\$_G(A') = \max\{\$_G(a\alpha') | A' \to a\alpha' \in P'\}$,
- $\$_G(X_1 \ldots X_m) = \max\{-\#_G(X_1 \ldots X_{k-1}) + \$_G(X_k) | 1 \leq k \leq m\}$ for $X_i \in \Sigma \cup N'$.

The algorithm in Figure 2 decides the consistency of $\#_G$ with $L(G')$ in Stage 1. The correctness of the procedure is guaranteed by the above lemma.

Suppose that we obtain a conclusion that $\#_G$ is consistent with $L(G')$ (otherwise $L(G') \not\subseteq L(G)$). Secondly, the algorithm emulates the derivations of G' by G in Stage 2. We conclude $L(G') \subseteq L(G)$ iff this emulation has been done with no errors. Although $L(G')$ may be infinite, we can do that. Hereafter,

we write $\widetilde{\#}$ and $\widetilde{\$}$ for the extended $\#_G$ and $\$_G$ computed in Stage 1. The rules of G' are represented by a set of trees. Each tree corresponds to one nonterminal of G' and there is a path from the root node which represents a rule in G'. Thus, each tree $T_{A'}$ represents the derivations of $L(G', A')$. For example, the trees which represent the rules $\{A' \to a,\ A' \to aB',\ B' \to bC',\ B' \to bB'D'\}$ are described as follows:



For each node in $T_{A'}$, we say *the address* of the node is $\langle A' \to a\alpha' \rangle$ if the path from the root node is $\langle a\alpha' \rangle$ (i.e., the edges on the path are labeled with $a, A'_1, \ldots, A'_{|\alpha'|}$ in this order where $\alpha' = A'_1 \ldots A'_{|\alpha'|}$). Note that, in this case, there is a rule $A' \to_{G'} \alpha'\alpha''$ for some $\alpha'' \in N'^*$ but not necessarily a rule $A' \to_{G'} a\alpha'$. We call the node whose address represents a rule in G' *final node* (indicated by double circles in the above). All the leaf nodes are final nodes, but the converse is not necessary.

Each node is labeled with a sequence of subsets of $N \cup \overline{N}$ where $\overline{N}$ is a twin of N (however the labels of root nodes consist of subsets of $\overline{N}$ only), while each edge is, in contrast, labeled with a single terminal or nonterminal in G'. The length of the label on the node of address $\langle A' \to a\alpha' \rangle$ is $\widetilde{\$}(A') + \widetilde{\#}(a\alpha')$ and the length of the label on the root node of $T_{A'}$ is $\$(A')$. In particular, the length of the label on every final node in $T_{A'}$ is $\widetilde{\$}(A') + \widetilde{\#}(A')$. The algorithm begins with the forest called *skeleton forest* whose all node labels are sequences of the empty sets. The algorithm adds some members of $N \cup \overline{N}$ to labels on nodes step by step in order to complete the forest as in Figure 2. Hereafter we use upper case letters from the end of the alphabet, $X, Y, Z, \ldots$ for subsets of $N \cup \overline{N}$ and upper case letters of the Greek alphabet, $\Gamma, \Delta, \Theta, \ldots$ for sequences of subsets of $N \cup \overline{N}$.

Suppose that the forest is completed (this means $L(G') \subseteq L(G)$) and that the root node of $T_{A'}$ is labeled with $X_1 \ldots X_m$ $(m = \widetilde{\$}(A'))$, $\overline{A_i} \in X_i$, $A' \Rightarrow_{G'} a\alpha'\beta' \stackrel{*}{\Rightarrow} ay\beta'$, and the node of address $\langle A' \to a\alpha' \rangle$ is labeled with $Y_1 \ldots Y_n$ $(n = m + \widetilde{\#}(a\alpha'))$. Then, there are $B_1, \ldots, B_n$ such that $A_1 \ldots A_m \stackrel{*}{\Rightarrow}_G ayB_1 \ldots B_n$, where $B_i \in Y_i$ for $1 \leq i \leq \widetilde{\$}(ay) + \widetilde{\#}(ay)$ and $\overline{A_{j-\widetilde{\#}(ay)}} = \overline{B_j} \in Y_j$ for $\widetilde{\$}(ay) + \widetilde{\#}(ay) < j \leq n$. In other words, $B_i \in N$ is determined by y but independently from any A_i by the right-uniqueness and $\overline{B_j} \in \overline{N}$ appears only when it has already appeared before deriving y. This is the difference between N and $\overline{N}$.

In an uncompleted forest, thus, the algorithm updates each node label as follows: Suppose that an edge labeled with A' connects a node labeled with Γ and its child node in some tree. If there are nonterminals which appear in Γ but not in the label on the root node of $T_{A'}$, then we must add them into the appropriate position in the label on the root node of $T_{A'}$. If some final node of $T_{A'}$ is labeled with nonempty sets, then by referring Γ and the labels on the root node and the final nodes of $T_{A'}$, we determine what nonterminals in $N \cup \overline{N}$ should be added to appropriate position in the label on the child node. Therefore, recursively we can determine all the labels on nodes in the forest.

If it occurs that there are a tree $T_{A'}$ and $\overline{A} \in X$ where the root node of $T_{A'}$ is labeled with $X\Gamma$ such that $A' \to_{G'} a\alpha' \in P'$ but $A \to_G a\alpha \notin P$ for any $\alpha \in N^*$, then the algorithm concludes $L(G') \not\subseteq L(G)$. Excluding such a case, when it occurs that the algorithm cannot modify any labels on trees, the algorithm concludes $L(G') \subseteq L(G)$. The algorithm terminates in finite steps, since each X in each node label has the upper bound $N \cup \overline{N}$.

The formal definitions of notations used in the algorithm is given as follows:

Definition 16. Let $\Gamma = X_1 \ldots X_n$, $\Delta = Y_1 \ldots Y_m$ and $\Theta = Z_1 \ldots Z_l$.

- $\Gamma \subseteq \Delta$ iff $n = m$ and $X_i \subseteq Y_i$ for all i.
- $\Theta = \Gamma \cap \Delta$ iff $n = m = l$ and $Z_i = X_i \cap Y_i$ for all i.
- $\Theta = \Gamma \cup \Delta$ iff $n = m = l$ and $Z_i = X_i \cup Y_i$ for all i.
- $\mathrm{Pre}(\Gamma, k) = X_1 \ldots X_k$ is defined only if $k \leq n$.
- $\mathrm{Start}(A')$ denotes the label on the root node of $T_{A'}$.
- $\mathrm{Final}(A')$ denotes the union of labels on the final nodes of $T_{A'}$.
- $\overline{X} = \{\overline{A} | A \in X \text{ or } \overline{A} \in X\}$, and $\overline{\Gamma_1\Gamma_2} = \overline{\Gamma_1}\ \overline{\Gamma_2}$

- Roughly speaking, Derive(Γ, A') (or Derive(Γ, a)) expresses the sequences of nonterminals which appear when some elements of Γ derive strings in $L(G', A')$ (or a respectively).
Let $\Gamma = X_1 \dots X_n$ and $A \to_G aB_1 \dots B_k$ for some $A \in N$. Then Derive(Γ, a) is defined as

$$\text{Derive}(\Gamma, a) = \{B_1\} \dots \{B_k\} X_2 \dots X_n.$$

For Final$(A') = Y_1 \dots Y_{\widetilde{\$}(A')+\widetilde{\#}(A')}$, Derive$(\Gamma, A')$ is defined as

$$\begin{aligned}
\text{Derive}(\Gamma, A') &= \Delta_1 \varnothing^{n-\widetilde{\$}(A')} \cup \varnothing^m \Delta_2 \varnothing^{n-\widetilde{\$}(A')} \cup \varnothing^{\widetilde{\$}(A')+\widetilde{\#}(A')} \Delta_3 \\
\text{where} \quad m &= \max\{0, \widetilde{\#}(A')\}, \\
\Delta_1 &= N^{\widetilde{\$}(A')+\widetilde{\#}(A')} \cap \text{Final}(A'), \\
\Delta_2 &= Z_1 \dots Z_k \text{ where } k = \min\{\widetilde{\$}(A'), \widetilde{\$}(A') + \widetilde{\#}(A')\} \\
&\quad \text{and } Z_i = \{A | A \in X_{i-\widetilde{\#}(A')} \text{ and } \overline{A} \in Y_i\} \cup \{\overline{A} | \overline{A} \in X_{i-\widetilde{\#}(A')} \text{ and } \overline{A} \in Y_i\}, \\
\Delta_3 &= X_{\widetilde{\$}(A')+1} \dots X_n.
\end{aligned}$$

5 Further Discussions

We have presented an algorithm which efficiently learns RSGs from positive data and an algorithm which decides the inclusion problem for RSGs. We see that the function shape of a grammar plays a important role in these algorithms. Then, it is natural to ask whether similar issues on the learning efficiency and inclusion problem is applicable to the subclass of simple grammars in which the shape is well defined, i.e., $A \to a\alpha, B \to a\beta$ implies $|\alpha| = |\beta|$. We can show an algorithm which solves the inclusion problem for the subclass in polynomial in the thicknesses and description sizes in the compared grammars. But, it is easy to see that the subclass is not learnable from positive data.

References

[1] Dana Angluin. Inference of reversible languages. *Journal of the Association for Computing Machinery*, 29(3):741–765, 1982.

[2] Sheila A. Greibach and Emily P. Friedman. Superdeterministic pdas: A subcase with a decidable inclusion problem. *Journal of the Association for Computing Machinery*, 27(4):675–700, 1980.

[3] Matti Linna. Two decidability results for deterministic pushdown automata. *Journal of Computer and System Science*, 18:92–107, 1979.

[4] Leonard Pitt. Inductive inference, dfas, and computational complexity. In *Proceedings of 2nd Workshop on Analogical and Inductive Inference*, Lecture Notes in Artificial Intelligence, pages 18–44, 1989.

[5] Mitsuo Wakatsuki and Etsuji Tomita. A fast algorithm for checking the inclusion for very simple deterministic pushdown automata. *IEICE transactions on information and systems*, E76-D(10):1224–1233, 1993.

[6] Takashi Yokomori. Polynomial-time identification of very simple grammars from positive data. *Theoretical Computer Science*, 298:179–206, 2003.

[7] Ryo Yoshinaka. A study on the mathematical properties and learning efficiency of very simple grammars and some extensions. Master's thesis, Graduate School of Interdisciplinary Information Studies, University of Tokyo, 2003.

Algorithm; Whether $L(G') \subseteq L(G)$**?**
Input $G' = (N', \Sigma, P', S')$ and $G = (N, \Sigma, P, S)$;
— *Stage 1.1. Compute* $\widetilde{\#}_G(N')$ —
let $\widetilde{\#} := \#_G$;
let every rule in G' be *unchecked*;
while there remains a *unchecked* rule **do**
take some *unchecked* rule $A' \to_{G'} a\alpha'$ where $\widetilde{\#}(\alpha')$ is already defined;
if $\widetilde{\#}(A')$ is not defined yet **then** define $\widetilde{\#}(A') := \widetilde{\#}(a\alpha')$;
elseif $\widetilde{\#}(A') \neq \widetilde{\#}(a\alpha')$ **then** output "$L(G') \not\subseteq L(G)$" and **halt**;
fi
check the rule $A' \to_{G'} a\alpha'$;
if $\widetilde{\#}(S') \neq -1$ **then** output "$L(G') \not\subseteq L(G)$" and **halt**; **fi**
od
— *Stage 1.2. Compute* $\widetilde{\$}_G(N')$ —
let $\widetilde{\$}(A') := 1$ for all $A' \in N'$;
while there occurs some change in $\widetilde{\$}$ **do**
for each nonterminal $A' \in N'$ **do**
let $\widetilde{\$}(A') := \max\{\widetilde{\$}(a\alpha') | A' \to_{G'} a\alpha' \in P'$ for some a and $\alpha'\}$;
od
if $\widetilde{\$}(S') \neq 1$ **then** output "$L(G') \not\subseteq L(G)$" and **halt**; **fi**
od
— *Stage 2. Decide the Inclusion* —
create the skeleton forest, where the root node of $T_{A'}$ is labeled with $\langle \varnothing^{\widetilde{\$}(A')} \rangle$
and the node of address $\langle A' \to a\alpha' \rangle$ is labeled with $\langle \varnothing^{\widetilde{\$}(A') + \widetilde{\#}(a\alpha')} \rangle$;
let Start$(S') := \{\overline{S}\}$ (label the root node of $T_{S'}$ with $\langle \{\overline{S}\} \rangle$);
while the forest is modified **do**
for an edge $\langle A' \rangle$ whose parent node is $\langle \Gamma \rangle$
do add Pre$(\overline{\Gamma}, \widetilde{\$}(A'))$ to Start(A'); **od**
for an edge $\langle A' \rangle$ connecting a parent node $\langle \Gamma \rangle$ and its child $\langle \Delta \rangle$
do add Derive(Γ, A') to Δ; **od**
for an edge $\langle a \rangle$ connecting a root node $\langle \Gamma \rangle$ and its child $\langle \Delta \rangle$
do add Derive(Γ, a) to Δ; **od**
if there are $A \in N$, $A' \in N'$ and $a \in \Sigma$ such that
$\overline{A} \in$ Pre(Start$(A'), 1)$, $A' \to_{G'} a\alpha' \in P'$ for some α' but $A \to_G a\alpha \notin P$ for any α
then output "$L(G') \not\subseteq L(G)$" and **halt**;
fi
od
output "$L(G') \subseteq L(G)$" and **halt**;
End Algorithm

Figure 2: An Algorithm for the Inclusion Problem for RSGs