Distributional Learning of Simple Context-Free Tree Grammars

Anna Kasprzik[1] and Ryo Yoshinaka[2,⋆]

[1] University of Trier, FB IV Informatik, 54286 Trier
kasprzik@informatik.uni-trier.de
[2] ERATO MINATO Project, Japan Science and Technology Agency
ry@ist.hokudai.ac.jp

Abstract. This paper demonstrates how existing distributional learning techniques for context-free grammars can be adapted to *simple context-free tree grammars* in a straightforward manner once the necessary notions and properties for string languages have been redefined for trees. Distributional learning is based on the decomposition of an object into a substructure and the remaining structure, and on their interrelations. A corresponding learning algorithm can emulate those relations in order to determine a correct grammar for the target language.

1 Introduction

This paper is settled in the area of *Grammatical Inference*, i.e., the study of algorithms that "learn" formal languages from only partial information. The class that has been studied most extensively with respect to its algorithmical learnability is that of regular string languages. The established learning algorithms for regular string languages have soon been extended to more complex structures, most notably to trees (see for example [1] for learning a subset of regular tree languages from finite positive data, and [2, 3, 4] for learning regular tree languages from queries and/or finite data).

However, recently a range of efforts have been made to explore other classes beyond the regular one. While the class of context-free languages as a whole does not seem to be learnable in any non-trivial setting considered so far, there exist several context-free subclasses with certain properties that make them learnable by accordingly adapted strategies which can be subsumed under the term of *distributional learning* (see for example [5, 6, 7, 8, 9, 10], and references therein).

Every member w of a context-free string language $L_* \subseteq \Sigma^*$ can be decomposed into a substring y and a context $\langle x, z\rangle \in \Sigma^* \times \Sigma^*$ such that $w = xyz$. The theory of distributional learning is concerned with the question which of those substrings and which of those contexts can be put back together to form another grammatical member of L_*. If L_* fulfils certain distributional properties, a corresponding learning algorithm can exploit information on these particular

⋆ The author is concurrently working in Hokkaido University. This work was supported in part by Mext Kakenhi (B-23700156)

J. Kivinen et al. (Eds.): ALT 2011, LNAI 6925, pp. 398–412, 2011.

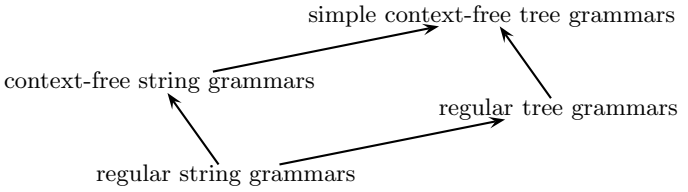

Fig. 1.

distributions for example in order to determine nonterminals and production rules of a grammar reflecting those interrelations, with the consequence that the resulting grammar correctly generates L_*.

The technique of distributional learning can be extended to languages based on more complex structures than the basic string once we have redefined such notions as a substructure and its counterpart, the *context* or *environment* of a certain substructure, along with an *unambiguous concatenation operation* for those two kinds of objects. Yoshinaka [10], for example, generalizes the learning algorithm given in [6] for context-free grammars (CFGs) to multiple CFGs. Moreover, Yoshinaka and Kanazawa [12] demonstrate that the techniques of distributional learning developed so far can be likewise adapted to other kinds of objects and formalisms in a rather abstract way via Abstract Categorial Grammars.

The contribution of our paper should be located in this line of research. We concentrate on the case of trees and consider *simple* (i.e., non-duplicating and non-deleting) *context-free tree grammars* (SCFTGs) as descriptions.[1] SCFTGs join two directions of generalization: They can be seen as a natural context-free counterpart of regular tree grammars and as a natural extension of CFGs to trees (Fig. 1). Moreover, the class of string languages that can be associated with SCFTGs is located in the so-called "mildly context-sensitive" family proposed by Joshi [11] which is of particular importance in computational linguistics. In this paper we demonstrate that distributional learning techniques for CFGs can be translated in a straightforward way to techniques for SCFTGs and thereby establish a parallel to the fact that learning techniques for regular string languages can be transferred directly to regular tree languages. We remark that although the distributional learning of SCFTGs can be seen as a special case covered by [12], in contrast to the generality of their discussion we obtain a much clearer and detailed view on the close correspondence between the distributional learning of CFGs and their immediate tree counterpart by basing our results directly on notions and results from formal (tree) language theory.

Organization of the paper: In Section 2 we define the necessary notational tools and we give the definition of SCFTGs. In Section 3 we establish the basic principles for the distributional learning of SCFTGs and then present and discuss two different distributional learning algorithms for them, in two different learning settings. We conclude and sketch some suggestions for future work in Section 4.

[1] SCFTGs have also been called *linear context-free tree grammars* in the literature.

2 Preliminaries

2.1 Trees and Substitutions

We presume a basic knowledge of standard tree terminology. For a more comprehensive introduction to trees and associated concepts and theorems, see [13] for example.

A *ranked alphabet* is a set of symbols Σ paired with a (total) function $\rho : \Sigma \to \mathbb{N}$. For $k \geq 0$, we write $\Sigma_k = \{a \in \Sigma \mid \rho(a) = k\}$ to denote the set of all symbols with rank k.

The set $\mathbb{T}_\Sigma$ of all *trees* over a ranked alphabet Σ is defined as the smallest set of expressions such that $t = f(t_1, \ldots, t_k) \in \mathbb{T}_\Sigma$ for every $k \geq 0$, $f \in \Sigma_k$, and $t_1, \ldots, t_k \in \mathbb{T}_\Sigma$. The tree $f()$ for $f \in \Sigma_0$ is often abbreviated to f. The *size* of a tree t, which is denoted by $|t|$, means the number of occurrences of symbols in t, i.e., $|f(t_1, \ldots, t_k)| = 1 + |t_1| + \cdots + |t_k|$.

A subset of $\mathbb{T}_\Sigma$ is called a *tree language*. For a finite tree language $L \subseteq \mathbb{T}_\Sigma$, we define the *size* of L by $\|L\| = \sum_{t \in L} |t|$.

Throughout this paper we assume a countably infinite set X of *ordered variables* $x_1, x_2, \ldots$ which have rank 0.

A tree $t \in \mathbb{T}_{\Sigma \cup X}$ is called an *m-stub* if only and all the variables $x_1, \ldots, x_m$ occur exactly once in t for some $m \in \mathbb{N}$.[2] The set of all m-stubs is denoted by $\mathbb{S}^m_\Sigma$ and we let $\mathbb{S}_\Sigma = \bigcup_{m \in \mathbb{N}} \mathbb{S}^m_\Sigma$. We note that $\mathbb{S}^0_\Sigma = \mathbb{T}_\Sigma$.

A *leaf substitution* η is a finite partial mapping from X to $\mathbb{T}_\Sigma$, which is extended to a function $\hat{\eta} : \mathbb{T}_{\Sigma \cup X} \to \mathbb{T}_{\Sigma \cup X}$ as follows:

- for $x \in X$, let $\hat{\eta}(x) = \eta(x)$ if $\eta(x)$ is defined and $\hat{\eta}(x) = x$ otherwise, and
- for $f \in \Sigma_k$, let $\hat{\eta}(f(t_1, \ldots, t_k)) = f(\hat{\eta}(t_1), \ldots, \hat{\eta}(t_k))$.

We identify a leaf substitution η with its extension $\hat{\eta}$. We often put η after an argument as a postfix operator, i.e., $t\eta$ instead of $\eta(t)$. A leaf substitution that maps $x_{i_1}, \ldots, x_{i_m}$ to $t_1, \ldots, t_m$, respectively, is often denoted as $[x_{i_1} \leftarrow t_1, \ldots, x_{i_m} \leftarrow t_m]$. In cases where the domain is just $\{x_1, \ldots, x_m\}$, we abbreviate such an operator $[x_1 \leftarrow t_1, \ldots, x_m \leftarrow t_m]$ to $[t_1, \ldots, t_m]$.

Let Δ and Σ be ranked alphabets. An *infix substitution* σ is a mapping from Δ to $\mathbb{S}_\Sigma$ such that $\sigma(f) \in \mathbb{S}^k_\Sigma$ for all $f \in \Delta_k$. σ is extended to a function $\hat{\sigma} : \mathbb{S}_{\Sigma \cup \Delta} \to \mathbb{S}_\Sigma$ as follows:

- $\hat{\sigma}(f(s_1, \ldots, s_k)) = \sigma(f)[\hat{\sigma}(s_1), \ldots, \hat{\sigma}(s_k)]$ for $f \in \Delta_k$,
- $\hat{\sigma}(f(s_1, \ldots, s_k)) = f(\hat{\sigma}(s_1), \ldots, \hat{\sigma}(s_k))$ for $f \notin \Delta_k$.

We identify an infix substitution σ with its extension $\hat{\sigma}$. The definition of an infix substitution coincides with that of a leaf substitution when the domain Δ consists only of symbols of rank 0. Hence we may denote an infix substitution that maps $Y_i \in \Delta$ to $s_i \in \mathbb{S}_\Sigma$ for $i = 1, \ldots, m$ as a postfix operator $[Y_1 \leftarrow s_1, \ldots, Y_m \leftarrow s_m]$ without confusion.

[2] What we call a stub is also called a 'context' in the literature on trees [13]. However, we avoid this established terminology since in our framework a stub fulfills the role of a substructure, and not of the remaining parts of a decomposition.

Example 1. Let $\Sigma = \Sigma_0 \cup \Sigma_1$ with $\Sigma_1 = \{a, b, c, d\}$, $\Sigma_0 = \{e\}$ and $\Delta = \Delta_1 = \{Y_1\}$. For $s = b(c(x_1)) \in \mathbb{S}^1_\Sigma$, the result of the application of an infix substitution $[Y_1 \leftarrow s]$ to a tree $t = a(Y_1(d(e))) \in \mathbb{T}_{\Sigma \cup \{Y_1\}}$ is $t[Y_1 \leftarrow s] = a(b(c(d(e))))$.

Let $\Sigma_0 = \{a, b, c\}$, $\Sigma_1 = \{h\}$, $\Sigma_2 = \{f, g\}$ and $\Delta = \Delta_2 = \{Y_2\}$. For $s = f(x_1, g(b, x_2)) \in \mathbb{S}^2_\Sigma$ and $t = h(Y_2(a, c)) \in \mathbb{T}_{\Sigma \cup \{Y_2\}}$, we have

$$t[Y_2 \leftarrow s] = h(Y_2(a, c))[Y_2 \leftarrow f(x_1, g(b, x_2))] = h(f(a, g(b, c))).$$

Lemma 1. *Let an infix substitution σ from Δ to $\mathbb{S}_\Sigma$ and a symbol $Y \in \Sigma$ be such that $Y \notin \Delta$ and $\sigma(a)$ contains no occurrence of Y for any $a \in \Delta$. Then $(s_1[Y \leftarrow s_2])\sigma = (s_1\sigma)[Y \leftarrow (s_2\sigma)]$.*

2.2 Simple Context-Free Tree Grammars

We are now ready to specify the grammars that generate the kind of languages we will consider as targets for our learning algorithms in Section 3.

Definition 1. Let $r \in \mathbb{N}$ be a natural number. We define an *r-simple context-free tree grammar (r-SCFTG)* as a 4-tuple $G = \langle \Sigma, N, I, P \rangle$ where

- Σ is a ranked alphabet of terminals with $\Sigma_m = \emptyset$ for all $m > r$,
- N is a ranked alphabet of nonterminals with $N_m = \emptyset$ for all $m > r$,
- $I \subseteq N_0$ is a set of initial symbols,[3] and
- P is a finite set of production rules of the form $A \to s$ with $A \in N_m$ and $s \in \mathbb{S}^m_\Sigma$ for some $m \geq 0$.

We let $P_m = \{\, A \to s \in P \mid A \in N_m \text{ and } s \in \mathbb{S}^m_\Sigma \,\}$.

For $u, v \in \mathbb{S}^n_{\Sigma \cup N}$, we write $u \Rightarrow_G v$ if there is $m \in \mathbb{N}$, $A \to s \in P_m$ and $t \in \mathbb{S}^n_{\Sigma \cup N \cup \{Y\}}$ such that Y has rank m and occurs just once in t and

- $u = t[Y \leftarrow A(x_1, \ldots, x_m)]$ and
- $v = t[Y \leftarrow s]$.

The relation $\Rightarrow^*_G$ on $\mathbb{S}_{\Sigma \cup N}$ is defined as the reflexive transitive closure of $\Rightarrow_G$.

The *stub language $\mathcal{L}(G, A)$ generated by $A \in N_m$* is the set

$$\mathcal{L}(G, A) = \{\, s \in \mathbb{S}^m_\Sigma \mid A(x_1, \ldots, x_m) \Rightarrow^*_G s \,\}.$$

We simply write $\mathcal{L}_A$ for $\mathcal{L}(G, A)$ where G is understood. The *tree language* generated by G is defined as $\mathcal{L}(G) = \bigcup_{A \in I} \mathcal{L}_A$.

Context-free (string) grammars (CFGs) can be interpreted as 1-SCFTGs and vice versa.

Lemma 2. *If $A \Rightarrow^*_G u[Y \leftarrow B(x_1, \ldots, x_m)]$ and $B \Rightarrow^*_G v$, then $A \Rightarrow^*_G u[Y \leftarrow v]$ for any $A \in N$, $B \in N_m$, $u \in \mathbb{S}_{\Sigma \cup N \cup \{Y\}}$, $v \in \mathbb{S}^m_{\Sigma \cup N}$ and Y of rank m.*

[3] Contrary to the standard definition of CFTGs, we allow multiple initial symbols. Obviously this generalization does not affect the expressive power of the formalism, yet our results will be conveniently presented in this non-standard form.

Definition 2. We call an m-stub s *non-permuting* if the variables $x_1, \ldots, x_m$ occur in this order from left to right in s.

An SCFTG is said to be *normal* if, for any rule $A \to s$, the stub $s \in \mathbb{S}_{\Sigma \cup N}$ is of one of the following two types:

I. $f(x_1, \ldots, x_k)$ for some $f \in \Sigma_k$ or
II. $B(x_1, \ldots, x_i, C(x_{i+1}, \ldots, x_j), x_{j+1}, \ldots, x_k)$ for some $B \in N_{k-j+i+1}$ and $C \in N_{j-i}$ with $0 \leq i \leq j \leq k$.

One can show that indeed every r-SCFTG admits an equivalent r-SCFTG in the normal form by a technique similar to the proofs for corresponding theorems in related formalisms (e.g., [14], [15]). For the rest of this paper we will assume all stubs of $\mathbb{S}^m_{\Sigma \cup N}$ to be non-permuting and all SCFTGs to be normal.

Proposition 1. *Fix a positive integer r. The uniform membership problem for r-SCFTGs, which asks whether $t \in \mathcal{L}(G)$, is decidable in polynomial time in the size of an r-SCFTG G and a tree t.*

Proof. The case of r-SCFTGs can be seen as a special case of a richer formalism that has a polynomial-time parsing algorithm if instances are restricted to r-SCFTGs (e.g., [16]). To make this paper self-contained, yet we outline a parsing algorithm for r-SCFTGs based on the standard CKY algorithm for CFGs. We have an $(m+1)$-dimensional table T_m where each dimension corresponds to a node position of the tree to be parsed for each $m = 0, \ldots, r$. We keep and update the tables so that each cell of T_m is occupied by nonterminals of rank m that can generate the m-stub determined by the corresponding $(m+1)$ positions. It is not hard to see that if instances are normal, such an algorithm runs in polynomial time, where the degree of the polynomial linearly depends on r.

3 Distributional Learning of Simple Context-Free Tree Grammars

3.1 Decomposition of Trees

Consider a context-free string language $L \subseteq \Sigma^*$, and the decompositions of members w of L into substrings w' and contexts $\langle u, v \rangle \in \Sigma^* \times \Sigma^*$ with $w = uw'v$. Distributional learning of CFGs is based on the analysis and emulation of the specific relations between those particular strings and contexts that when put together form a grammatical string of the language in question.

We (re)define suitable concepts corresponding to 'substrings' and 'contexts' for the tree case, with the goal of making distributional learning algorithms for strings directly translatable into distributional learning algorithms for trees. In our framework, the tree counterpart of a substring is a stub, which is the kind of object that a nonterminal of an SCFTG derives. We specify our tree counterpart of a context as follows.

Definition 3. An *m-environment* is a tree over $\Sigma \cup \{\#_m\}$ in which $\#_m$ occurs exactly once, where $\#_m \notin \Sigma$ is a special symbol of rank m. The set of all m-environments is denoted by $\mathbb{E}^m_\Sigma$. For an m-environment $e \in \mathbb{E}^m_\Sigma$ and an m-stub $s \in \mathbb{S}^m_\Sigma$, we define a binary operation $\odot_m$ by

$$e \odot_m s = e[\#_m \leftarrow s].$$

The domain and accordingly the range of the operation is naturally extended to sets in the standard way.

The subscript m of $\odot_m$ is often suppressed. We note that the result of the operation is always a tree in $\mathbb{T}_\Sigma$.

Definition 4 contains the essence of the substructure-environment relation for a certain language in the tree case.

Definition 4. For a tree language $L \subseteq \mathbb{T}_\Sigma$ and $m, r \in \mathbb{N}$, we let

$$\begin{aligned} Sub^m(L) &= \{\, s \in \mathbb{S}^m_\Sigma \mid e \odot s \in L \text{ for some } e \in \mathbb{E}^m_\Sigma \,\}, \\ Sub^{\leq r}(L) &= \bigcup_{m \leq r} Sub^m(L), \\ Env^m(L) &= \{\, e \in \mathbb{E}^m_\Sigma \mid e \odot s \in L \text{ for some } s \in \mathbb{S}^m_\Sigma \,\}, \\ Env^{\leq r}(L) &= \bigcup_{m \leq r} Env^m(L). \end{aligned}$$

We note that we always have $\#_0 \in Env^0(L)$ and $x_1 \in Sub^1(L)$ unless L is empty. $Sub^0(L)$ corresponds to the set of 'subtrees' in the usual sense.

Lemma 3. *Fix a positive integer r. For any finite language $L \subseteq \mathbb{T}_\Sigma$, one can compute the sets $Sub^{\leq r}(L)$ and $Env^{\leq r}(L)$ in polynomial time in $\|L\|$.*

We remark that the degree of the polynomial linearly depends on r.

Hereafter, we drop Σ to denote sets $\mathbb{S}_\Sigma, \mathbb{E}_\Sigma$ as $\mathbb{S}, \mathbb{E}$, when Σ is understood.

Example 2. Let $\Sigma = \Sigma_0 \cup \Sigma_2$ where $\Sigma_0 = \{a, b, c\}$ and $\Sigma_2 = \{f, g\}$. A tree $t = f(a, g(b, c)) \in \mathbb{T}$ can be decomposed in many ways:

$$\begin{aligned} t &= \#_0 \odot t && (\#_0 \in Env^0(t),\ t \in Sub^0(t)) \\ &= f(a, \#_0) \odot g(b, c) && (f(a, \#_0) \in Env^0(t),\ g(b, c) \in Sub^0(t)) \\ &= f(a, \#_1(b)) \odot g(x_1, c) && (f(a, \#_1(b)) \in Env^1(t),\ g(x_1, c) \in Sub^1(t)) \\ &= f(a, \#_2(b, c)) \odot g(x_1, x_2) && (f(a, \#_2(b, c)) \in Env^1(t),\ g(x_1, x_2) \in Sub^1(t)) \\ &= \#_2(a, c) \odot f(x_1, g(b, x_2)) && (\#_2(a, c) \in Env^2(t),\ f(x_1, g(b, x_2)) \in Sub^2(t)) \end{aligned}$$

and so on.

In the following subsections, we illustrate how these adapted notions result in distributional learning algorithms for trees by discussing some concrete examples in detail.

3.2 Substitutable Simple Context-Free Tree Languages

There are several properties that make context-free languages learnable using distributional techniques. Among those properties, we will pick the strongest one first: *Substitutability*. The class of substitutable CFGs and even the class of substitutable multiple context-free grammars (MCFGs) have been shown to be efficiently learnable from positive examples by Clark [6] and Yoshinaka [10], respectively. Compared to other existing distributional learning algorithms those learners are rather simple due to the great restriction of substitutability.

The learning setting we consider is the same as in the two references given above: *Identification in the limit from positive data* [17]. Let G_* be the target grammar. A learner $\mathcal{A}$ is given an infinite sequence of positive examples $t_1, t_2, \cdots \in \mathcal{L}(G_*)$ fulfilling the condition $\mathcal{L}(G_*) = \{\, t_i \mid i \geq 1 \,\}$. For each $n \geq 1$, $\mathcal{A}$ constructs a conjecture grammar G_n based on the data $t_1, \ldots, t_n$ received so far. We say that $\mathcal{A}$ *identifies* G_* *in the limit from positive data* if for any sequence of positive examples from $\mathcal{L}(G_*)$, there is a point n_0 such that $\mathcal{L}(G_{n_0}) = \mathcal{L}(G_*)$ and $G_n = G_{n_0}$ for all $n > n_0$.

This subsection presents an efficient algorithm that identifies every *r-substitutable* SCFTG in the limit from positive data.

Definition 5. A tree language $L \subseteq \mathbb{T}_\Sigma$ is said to be *r-substitutable* if the following holds:

- For any $m \leq r$ and $s_1, s_2 \in \mathbb{S}^m$, if there is $e_0 \in \mathbb{E}^m$ with $e_0 \odot s_1,\ e_0 \odot s_2 \in L$ then we have $e \odot s_1 \in L$ iff $e \odot s_2 \in L$ for all $e \in \mathbb{E}^m$.

In such a case we say that s_1 and s_2 are *substitutable* for each other. An *r-substitutable* SCFTG is an r-SCFTG G such that $\mathcal{L}(G)$ is r-substitutable.

Fix an r-substitutable SCFTG G_* as our learning target. For a finite set $D \subseteq \mathcal{L}(G_*)$ of positive examples, our learner constructs an SCFTG $\mathcal{G}_D = \langle \Sigma, N, I, P \rangle$ as follows. First, we take the substubs in $Sub^{\leq r}(D)$ as nonterminal symbols:

$$N_m = \{\, [\![s]\!] \mid s \in Sub^m(D) \,\} \text{ for } m \leq r,$$
$$I = \{\, [\![t]\!] \in N_0 \mid t \in D \,\}.$$

We want each nonterminal $[\![s]\!]$ to derive $s' \in \mathbb{S}$ if and only if s and s' are substitutable for each other. Our grammar $\mathcal{G}_D$ has rules of the following two types.

I. $[\![s]\!] \to a(x_1, \ldots, x_m)$ for $s \in Sub^m(D)$ and $a \in \Sigma_m$,
if there is $e \in Env^m(D)$ such that $e \odot s \in D$ and $e \odot a(x_1, \ldots, x_m) \in D$;

II. $[\![s]\!] \to [\![s_1]\!](x_1, \ldots, x_i, [\![s_2]\!](x_{i+1}, \ldots, x_j), x_{j+1}, \ldots, x_m)$ for $s \in Sub^m(D)$, $s_1 \in Sub^{m-j+i+1}(D)$, $s_2 \in Sub^{j-i}(D)$, if there is $e \in Env^m(D)$ such that $e \odot s \in D$ and $e \odot s_1[x_1, \ldots, x_i, s_2[x_{i+1}, \ldots, x_j], x_{j+1}, \ldots, x_m] \in D$.

In other words, we obtain a rule of Type I if the learner finds a common environment for s and $a(x_1, \ldots, x_m)$ and can thus conclude that those two stubs

are substitutable for each other, and accordingly a rule of Type II if the same occurs for s and $s_1[x_1, \ldots, x_i, s_2[x_{i+1}, \ldots, x_j], x_{j+1}, \ldots, x_m]$.

A *trivial* rule of Type I is $[\![a(x_1, \ldots, x_m)]\!] \to a(x_1, \ldots, x_m)$ for $a \in \Sigma_m$. Once the symbol a is observed in D, the learner constructs this trivial rule. Similarly, if $s \in Sub(D)$ is represented as $s = s_1[\vec{x}_1, s_2[\vec{x}_2], \vec{x}_3]$ for some $s_1, s_2 \in Sub(D)$, then we have the *trivial* rule of Type II

$$[\![s_1[\vec{x}_1, s_2[\vec{x}_2], \vec{x}_3]]\!] \to [\![s_1]\!](\vec{x}_1, [\![s_2]\!](\vec{x}_2), \vec{x}_3),$$

where $\vec{x}_1, \vec{x}_2, \vec{x}_3 = x_1, \ldots, x_m$. Successive applications of such trivial rules decompose a stub s into pieces in arbitrary ways and finally we obtain $[\![s]\!] \Rightarrow^*_{\mathcal{G}_D} s$ for all $[\![s]\!] \in N$.

Algorithm 1 shows our learner for r-substitutable SCFTGs. It is clear by Proposition 1 and Lemma 3 that the algorithm updates its conjecture $\hat{G}$ in polynomial time in the size $\|D\|$ of D.

Algorithm 1. Learning r-substitutable SCFTGs

Data: A sequence of positive examples $t_1, t_2, \ldots$
Result: A sequence of SCFTGs $G_1, G_2, \ldots$
let $\hat{G}$ be an SCFTG such that $\mathcal{L}(\hat{G}) = \varnothing$;
for $n = 1, 2, \ldots$ **do**
  read the next example t_n;
  if $t_n \notin \mathcal{L}(\hat{G})$ **then**
    let $\hat{G} = \mathcal{G}_D$ for $D = \{t_1, \ldots, t_n\}$;
  end if
  output $\hat{G}$ as G_n;
end for

Example 3. Let us sketch an example run for Algorithm 1.

Consider a target SCFTG $G_* = \langle \Sigma_0 \cup \Sigma_3, N_0^* \cup N_3^*, I^*, P^* \rangle$ where $\Sigma_0 = \{a, b, c, d, f\}$, $\Sigma_3 = \{g, h\}$, $N_0^* = \{A\}$, $N_3^* = \{B\}$, $I^* = \{A\}$, and P^* consists of the following three rules

$$A \to g(a, B(b, f, c), d),\ B \to h(x_1, x_2, x_3),\ B \to g(a, B(b, g(x_1, x_2, x_3), c), d)\,.$$

Figure 2 illustrates the rules of G_* and example trees $t_1 = g(a, h(b, f, c), d)$ and $t_2 = g(a, g(a, h(b, g(b, f, c), c), c), d)$ generated by G_*. Obviously G_* is not in normal form but since the learner only has to construct a grammar generating the same language as G_* we choose this representation for understandability. Suppose the learner is given the tree t_1 as its first datum. The learner will react by constructing a grammar $\mathcal{G}_{\{t_1\}} = \{\Sigma, N, I, P\}$ as specified above. At this first stage, I is the singleton of $[\![t_1]\!]$ and all the constructed rules in P are trivial ones. Thus this grammar does not generate any other tree but t_1 itself. Suppose the next datum given to the learner is t_2. This results in several additional nonterminals and one more start symbol $[\![t_2]\!]$. The learner observes

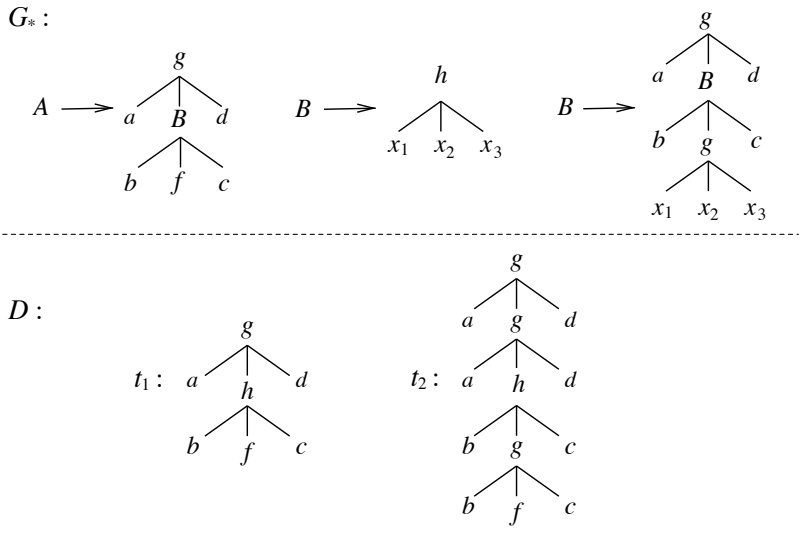

Fig. 2. Grammar G_* and sample set D for the example run of Algorithm 1

that $h(x_1, x_2, x_3)$ and $g(a, h(b, g(x_1, x_2, x_3), c), d)$ are substitutable for each other due to the environment $g(a, \#_3(b, f, c), d)$:

$$t_1 = g(a, \#_3(b, f, c), d) \odot h(x_1, x_2, x_3) \in D,$$
$$t_2 = g(a, \#_3(b, f, c), d) \odot g(a, h(b, g(x_1, x_2, x_3), c), d) \in D.$$

The learner constructs a rule

$$[\![h(x_1, x_2, x_3)]\!] \to [\![g(a, x_1, d)]\!]([\![h(b, g(x_1, x_2, x_3), c)]\!](x_1, x_2, x_3))$$

by the fact $g(a, h(b, g(x_1, x_2, x_3), c), d) = g(a, x_1, d)[x_1 \leftarrow h(b, g(x_1, x_2, x_3), c)]$. Successive applications of trivial rules result in a derivation

$$\begin{aligned}[\![h(x_1, x_2, x_3)]\!] &\underset{\mathcal{G}_D}{\Rightarrow} [\![g(a, x_1, d)]\!]([\![h(b, g(x_1, x_2, x_3), c)]\!](x_1, x_2, x_3)) \\ &\overset{*}{\underset{\mathcal{G}_D}{\Rightarrow}} g(a, [\![h(x_1, x_2, x_3)]\!](b, g(x_1, x_2, x_3), c), d),\end{aligned}$$

which simulates the rule $B \to g(a, B(b, g(x_1, x_2, x_3), c), d)$ of G_*. As a consequence, $\mathcal{G}_D$ generates every string in $\mathcal{L}(G_*)$.

Lemma 4. *$\mathcal{L}(\mathcal{G}_D) \subseteq \mathcal{L}(G_*)$ for any $D \subseteq \mathcal{L}(G_*)$.*

Proof. Let $\mathcal{G}_D = \langle \Sigma, N, I, P \rangle$ and σ an infix substitution from N to $\mathbb{S}_\Sigma$ given by $\sigma([\![s]\!]) = s$ for all $[\![s]\!] \in N$. By induction on the length of derivation, we show that if $[\![t]\!] \Rightarrow^*_{\mathcal{G}_D} t'$ for $[\![t]\!] \in I$, then $t'\sigma \in \mathcal{L}(G_*)$. If $[\![t]\!] \in I$, by definition it means $[\![t]\!]\sigma = t \in \mathcal{L}(G_*)$. Thus the claim holds for any zero-step derivation.

Suppose that the last rule applied in the derivation process from $[\![t]\!]$ to t' is

$$[\![s]\!] \to [\![s_1]\!](x_1, \ldots, x_i, [\![s_2]\!](x_{i+1}, \ldots, x_j), x_{j+1}, \ldots, x_m).$$

That is, there is $u \in \mathbb{E}^m$ such that

$$\begin{aligned}[\![t]\!] &\underset{\mathcal{G}_D}{\overset{*}{\Rightarrow}} u \odot [\![s]\!](x_1, \ldots, x_m) \\ &\underset{\mathcal{G}_D}{\Rightarrow} u \odot [\![s_1]\!](x_1, \ldots, x_i, [\![s_2]\!](x_{i+1}, \ldots, x_j), x_{j+1}, \ldots, x_m) = t'.\end{aligned}$$

For the sake of legibility, let us abbreviate $x_1, \ldots, x_i$ to $\vec{x}_1$, $x_{i+1}, \ldots, x_j$ to $\vec{x}_2$ and $x_{j+1}, \ldots, x_m$ to $\vec{x}_3$. By the induction hypothesis, we have

$$(u \odot [\![s]\!](\vec{x}_1, \vec{x}_2, \vec{x}_3))\sigma = u\sigma \odot s[\vec{x}_1, \vec{x}_2, \vec{x}_3] = u\sigma \odot s \in \mathcal{L}(G_*)$$

by Lemma 1. By the presence of the rule, we know that s and $s_1[\vec{x}_1, s_2[\vec{x}_2], \vec{x}_3]$ are substitutable for each other in $\mathcal{L}(G_*)$. The fact $u\sigma \odot s \in \mathcal{L}(G_*)$ implies $t'\sigma = u\sigma \odot s_1[\vec{x}_1, s_2[\vec{x}_2], \vec{x}_3] \in \mathcal{L}(G_*)$.

The case where the last applied rule is of Type I can be shown in the same way. □

Next we give a set of positive examples from which our learner constructs a right conjecture. Let the target grammar $G_* = \langle \Sigma, N^*, I^*, P^* \rangle$. For each nonterminal $A \in N^*_m$, arbitrarily fix an m-environment e_A such that $B \Rightarrow^*_{G_*} e_A \odot A(\vec{x})$ for some $B \in I^*$ and an m-stub s_A such that $A \Rightarrow^*_{G_*} s_A$. We define

$$\begin{aligned}D_* = &\{\, e_A \odot a(x_1, \ldots, x_m) \mid A \to a(x_1, \ldots, x_m) \in P^* \text{ with } a \in \Sigma \,\} \\ &\cup \{\, e_A \odot s_B[\vec{x}_1, s_C[\vec{x}_2], \vec{x}_3]] \mid A \to B(\vec{x}_1, C(\vec{x}_2), \vec{x}_3) \in P^* \text{ with } B, C \in N^* \,\}.\end{aligned}$$

Note that $e_A \odot s_A \in D_*$ for all $A \in N^*$ and $e_B = \#_0$ for all $B \in I^*$.

Lemma 5. *For any $D \subseteq \mathcal{L}(G_*)$, if $D \supseteq D_*$, we have $\mathcal{L}(\mathcal{G}_D) = \mathcal{L}(G_*)$.*

Proof. We show that every nonterminal A of G_* is simulated by $[\![s_A]\!]$ in $\mathcal{G}_D$. If $A \in I^*$, then $s_A \in D$ and thus $[\![s_A]\!] \in I$.

For a rule $A \to a(x_1, \ldots, x_m) \in P^*$, we have

$$e_A \odot s_A,\ e_A \odot a(x_1, \ldots, x_m) \in D_*.$$

By definition, $\mathcal{G}_D$ has the corresponding rule of Type I:

$$[\![s_A]\!] \to a(x_1, \ldots, x_m).$$

Let us consider a rule

$$A \to B(\vec{x}_1, C(\vec{x}_2), \vec{x}_3) \in P^*$$

of G_*. We have

$$e_A \odot s_A,\ e_A \odot s_B[\vec{x}_1, s_C[\vec{x}_2], \vec{x}_3]] \in D_*.$$

By definition, $\mathcal{G}_D$ has the corresponding rule of Type II:

$$[\![s_A]\!] \to [\![s_B]\!](\vec{x}_1, [\![s_C]\!](\vec{x}_2), \vec{x}_3)).$$

Consequently, every derivation of G_* is simulated by a derivation of $\mathcal{G}_D$. □

Therefore, once the learner gets a superset of D_*, it always conjectures an SCFTG that generates the target language. We also remark that the set D_* is not too big. D_* consists of at most $|P^*|$ positive examples and if we choose e_A and s_A to be minimal, each element in D_* is a minimal tree that is obtained using a rule of G_*.

Theorem 1. *Algorithm 1 identifies every r-substitutable* SCFTG *in the limit from positive data. It updates the conjecture in polynomial time and the number of examples needed to converge is bounded by a polynomial in the number of rules of the target grammar.*

3.3 Finite Environment Property

In [18], Clark defines the notion of a *Finite Context Property* for CFGs and proposes a corresponding learning algorithm. The learning setting he assumes is *identification in the limit from positive data and membership queries*, which is similar to the one from the previous subsection except that in addition the learner has access to a *membership oracle* to ask if a certain object is in the target language or not, and will receive an answer in constant time. This subsection defines an analogous property for SCFTGs and present a matching learning algorithm. We base our algorithm on the one proposed for strings in [19], which is a simpler version of Clark's original.

Definition 6. We say that an r-SCFTG G has the *p-Finite Environment Property* (p-FEP) if for every $A \in N_m$, there is $F_A \subseteq \mathbb{E}^m$ such that $|F_A| \leq p$ and

$$\mathcal{L}_A = \{\, s \in \mathbb{S}^m \mid F_A \odot s \subseteq \mathcal{L}(G_*) \,\}.$$

We call such an environment set F_A a *characterizing environment set of* A.

Before giving the overall view of our learning algorithm (Algorithm 2), we describe the construction and important properties of the SCFTG that the learner maintains as its current conjecture.

Let G_* be our learning target, an r-SCFTG with the p-FEP, and $L_* = \mathcal{L}(G_*)$. Suppose we have two finite sets $S \subseteq \mathbb{S}$ and $E \subseteq \mathbb{E}$ such that $a(x_1, \ldots, x_m) \in S$ for all $a \in \Sigma_m$ and $\#_0 \in E$, which are computed from given positive data in a way explained later. We define an SCFTG $\mathcal{G}_{r,p}(E, S) = \langle \Sigma, N, P, I \rangle$ as follows. While Algorithm 1 takes stubs as nonterminals, Algorithm 2 uses sets of environments as nonterminals.

$$N_m = \{\, [\![F]\!] \mid F \subseteq E \cap \mathbb{E}^m \text{ and } |F| \leq p \,\} \text{ for } m \leq r,$$
$$I = \{\, [\![\{\#_0\}]\!] \,\}.$$

Note that $|N_m| \leq |E|^p$. We have rules of two types, which require the aid of the oracle to be determined:

I. $[\![F]\!] \to a(x_1, \ldots, x_m)$ with $a \in \Sigma_m$ if $F \odot a(x_1, \ldots, x_m) \subseteq L_*$;

II. $[\![F]\!] \to [\![F_1]\!](\vec{x}_1, [\![F_2]\!](\vec{x}_2), \vec{x}_3)$ if

for all $s_1, s_2 \in S$, $F_1 \odot s_1,\ F_2 \odot s_2 \subseteq L_*$ implies $F \odot s_1[\vec{x}_1, s_2[\vec{x}_2], \vec{x}_3] \subseteq L_*$.

When learning an r-SCFTG with the p-FEP, we want each nonterminal $[\![F]\!]$ to derive s if and only if $F \odot s \subseteq L_*$, that is, we want F to be a characterizing environment set of $[\![F]\!]$. Conversely, for each nonterminal A of the target grammar G_*, we want our conjecture to simulate A by a nonterminal $[\![F_A]\!]$ where F_A is a characterizing environment set for A extracted from the given data. From this idea, rules of Type II of the form $[\![F]\!] \to [\![F_1]\!](\vec{x}_1, [\![F_2]\!](\vec{x}_2), \vec{x}_3)$ are justified if for all $s_1, s_2 \in \mathbb{S}$,

$$F_1 \odot s_1,\ F_2 \odot s_2 \subseteq L_* \text{ implies } F \odot s_1[\vec{x}_1, s_2[\vec{x}_2], \vec{x}_3] \subseteq L_* \tag{1}$$

since $s_1 \in \mathcal{L}_{[\![F_1]\!]}$, $s_2 \in \mathcal{L}_{[\![F_2]\!]}$ implies $s_1[\vec{x}_1, s_2[\vec{x}_2], \vec{x}_3] \in \mathcal{L}_{[\![F]\!]}$ under the presence of the rule. However, since $\mathbb{S}^m$ is infinite for each m, one cannot check this property (1). Instead, we use the finite set S. This is the idea behind the rule construction.

We say that a rule of Type II is *correct* if (1) holds.

Lemma 6. *If $E \subseteq F$ then $\mathcal{L}(\mathcal{G}_{r,p}(E, S)) \subseteq \mathcal{L}(\mathcal{G}_{r,p}(F, S))$.*

Proof. Every rule of $\mathcal{G}_{r,p}(E, S)$ is also that of $\mathcal{G}_{r,p}(F, S)$. □

Lemma 7. *If $S \subseteq T$ then $\mathcal{L}(\mathcal{G}_{r,p}(E, T)) \subseteq \mathcal{L}(\mathcal{G}_{r,p}(E, S))$.*

Proof. Every rule of $\mathcal{G}_{r,p}(E, T)$ is also that of $\mathcal{G}_{r,p}(E, S)$. □

We want every rule of the conjectured grammar to be correct. The following argument shows that for every finite set $E \subseteq \mathbb{E}$, there is a finite set $S \subseteq \mathbb{S}$ such that every rule of $\mathcal{G}_{r,p}(E, S)$ is correct.

For $F \subseteq \mathbb{E}^m$, $F_1 \subseteq \mathbb{E}^k$, $F_2 \subseteq \mathbb{E}^j$ and $i \leq m - j$ such that $m = k + j - 1$, if a rule of Type II

$$[\![F]\!] \to [\![F_1]\!](x_1, \ldots, x_i, [\![F_2]\!](x_{i+1}, \ldots, x_{i+j}), x_{i+j+1}, \ldots, x_m)$$

is not correct, there are $s_1, s_2 \in \mathbb{S}$ such that

$$F_1 \odot s_1, F_2 \odot s_2 \subseteq L_* \text{ and } F \odot s_1[x_1, ..., x_i, s_2[x_{i+1}, ..., x_{i+j}], x_{i+j+1}, ..., x_m] \nsubseteq L_*.$$

If $s_1, s_2 \in S$, the incorrect rule is suppressed. Hence, $2r|N|^3 \leq 2r|E|^{3p}$ stubs suffice to suppress all incorrect rules. We say that S is *fiducial on* E if every rule of $\mathcal{G}_{r,p}(E, S)$ is correct.

Lemma 8. *If every rule of $\hat{G} = \mathcal{G}_{r,p}(E, S)$ is correct, we have $\mathcal{L}(\hat{G}) \subseteq L_*$.*

Proof. One can prove by induction that for any $[\![F]\!] \in N$ and $s \in \mathbb{S}$, whenever $[\![F]\!] \Rightarrow^*_{\hat{G}} s$, it holds that $F \odot s \subseteq L_*$. By definition of I, we have proven the lemma. □

Lemma 9. *Let L_* be generated by an r-SCFTG G_* with the p-FEP. Then $L_* \subseteq \mathcal{L}(\mathcal{G}_{r,p}(E, S))$ if E includes a characterizing environment set F_A of A for every nonterminal A of G_*.*

Algorithm 2. Learning SCFTGs with the p-FEP

Data: A sequence of trees $t_1, t_2, \cdots \in \mathcal{L}(G_*)$; membership oracle $\mathcal{O}$;
Result: A sequence of r-SCFTGs $G_1, G_2, \ldots$;
let $D := \varnothing$; $E := \varnothing$; $S := \varnothing$; $\hat{G} := \mathcal{G}_{r,p}(E, S)$;
for $n = 1, 2, \ldots$ **do**
  let $D := D \cup \{t_n\}$; $S := Sub^{\leq r}(D)$;
  if $D \nsubseteq \mathcal{L}(\hat{G})$ **then**
    let $E := Env^{\leq r}(D)$;
  end if
  output $\hat{G} = \mathcal{G}_{r,p}(E, S)$ as G_n;
end for

Proof. Each nonterminal A of G_* is simulated by $[\![F_A]\!]$ in $\mathcal{G}_{r,p}(E, S)$ except the very first step of a derivation from an initial symbol of G_*, where that initial symbol of G_* is simulated by $[\![\{\#_0\}]\!]$ in $\mathcal{G}_{r,p}(E, S)$. □

Due to the nice properties presented in the lemmas above, our learning algorithm is quite simple. Whenever we get a positive example that is not generated by our current conjecture, we expand E (see Lemma 6). On the other hand, to suppress incorrect rules, we keep expanding S (Lemma 7).

Theorem 2. *Algorithm 2 identifies r-SCFTGs with the p-FEP in the limit from positive data and membership queries.*

Proof. Let $D_n = \{t_1, \ldots, t_n\}$. Lemma 9 ensures that Algorithm 2 does not update E infinitely many times, because characterizing environment sets of nonterminals in G_* are finite subsets of $Env(L_*)$ and at some point the learner will have seen all of them. Let $E_{m_0} = Env(D_{m_0})$ be the limit and $S_{n_0} = Sub(D_{n_0})$ be fiducial on E_{m_0}. Together with Lemma 8, we see that for any $n \geq \max\{m_0, n_0\}$, Algorithm 2 outputs $G_n = \mathcal{G}_{r,p}(E_{m_0}, S_{n_0})$ such that $\mathcal{L}(G_n) = \mathcal{L}(G_*)$. □

Some remarks on the efficiency of our algorithm: It is easy to see that $\|E\|, \|S\| \in O(\|D\|^{r+1})$ and we have at most $|E|^p$ nonterminals. We need at most a polynomial number of membership queries to determine rules among those nonterminals. All in all, Algorithm 2 updates its conjecture in polynomial time in the size $\|D\|$ of the data D. Moreover, we do not need too much data. To get characterizing environments of all nonterminals, $p|N^*|$ examples are enough, where N^* is the set of nonterminals of the target SCFTG G_*. To suppress incorrect rules, $O(r|E|^{3p})$ stubs are enough by Lemma 8.

4 Conclusion

We have demonstrated how existing distributional learning techniques for CFGs can be naturally extended to SCFTGs by giving the necessary theoretical foundations and by discussing two concrete efficient algorithms for the distributional learning of SCFTGs in detail. These are just two examples for the potential of our

translation – in fact, any distributional algorithm for strings should be translatable into an algorithm for trees by this method. This includes other learning settings such as membership and equivalence queries – see [8,5] for distributional algorithms based on this scenario.

We suggest *Tree Adjoining Grammars* (TAGs; [11]) as another prominent tree generating formalism that becomes learnable via our distributional techniques by modifying the definitions of substubs and environments accordingly. Moreover, the shift from strings to trees executed in this paper can be easily continued to context-free graph formalisms such as *hyperedge replacement grammars* [20] by giving a similar translation and specifying in an analogous manner how a learner would extract a sub-hypergraph from a given example of the target language to construct his hypothesis. We have focused on simple CFTGs as the most intuitive extension of CFGs to trees – a further conceivable direction for future work is the question of what happens if we allow subtrees to be duplicated.

Practical applications: As Clark [21] mentions in his conclusion, distributional learning techniques are adaptable to a probabilistic paradigm by focussing on the frequency with which the substrings and contexts of a language occur in (large) sets of given data. Hence it seems a promising approach for example in computational linguistics to consider applying probabilistically modified distributional techniques to large corpora of linguistic data such as the Penn treebank, from which a probabilistic CFG or also a TAG can be extracted (see [22], [23]), once we have specified distributional algorithms for TAGs as suggested above.

Acknowledgement. The authors are grateful to Alexander Clark and the anonymous reviewers for valuable comments that have improved the quality of this paper.

References

[1] López, D., Sempere, J.M., García, P.: Inference of reversible tree languages. IEEE Transactions on Systems, Man, and Cybernetics, Part B 34(4), 1658–1665 (2004)
[2] Drewes, F., Högberg, J.: Learning a regular tree language from a teacher. In: Ésik, Z., Fülöp, Z. (eds.) DLT 2003. LNCS, vol. 2710, pp. 279–291. Springer, Heidelberg (2003)
[3] Besombes, J., Marion, J.Y.: Learning tree languages from positive examples and membership queries. Theoretical Computer Science 382, 183–197 (2007)
[4] Oncina, J., Garcia, P.: Inference of recognizable tree sets. Technical report, DSIC II/47/93, Universidad de Valencia (1993)
[5] Shirakawa, H., Yokomori, T.: Polynomial-time MAT learning of c-deterministic context-free grammars. Transaction of Information Processing Society of Japan 34, 380–390 (1993)
[6] Clark, A., Eyraud, R.: Polynomial identification in the limit of substitutable context-free languages. Journal of Machine Learning Research 8, 1725–1745 (2007)
[7] Clark, A., Eyraud, R., Habrard, A.: Using contextual representations to efficiently learn context-free languages. Journal of Machine Learning Research 11, 2707–2744 (2010)
[8] Clark, A.: Distributional learning of some context-free languages with a minimally adequate teacher. In: [24], pp. 24–37

[9] Clark, A.: Towards general algorithms for grammatical inference. In: Hutter, M., Stephan, F., Vovk, V., Zeugmann, T. (eds.) ALT 2010. LNCS, vol. 6331, pp. 11–30. Springer, Heidelberg (2010)
[10] Yoshinaka, R.: Efficient learning of multiple context-free languages with multidimensional substitutability from positive data. Theor. Comput. Sci. 412(19), 1821–1831 (2011)
[11] Joshi, A.K.: Tree adjoining grammars: How much context-sensitivity is required to provide reasonable structural description. In: Dowty, D., Karttunen, L., Zwicky, A. (eds.) Natural Language Processing, Cambridge University Press, Cambridge (1985)
[12] Yoshinaka, R., Kanazawa, M.: Distributional learning of abstract categorial grammars. In: Pogodalla, S., Prost, J.-P. (eds.) LACL. LNCS, vol. 6736, pp. 251–266. Springer, Heidelberg (2011)
[13] Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree Automata Techniques and Applications (2008)
[14] Seki, H., Kato, Y.: On the generative power of multiple context-free grammars and macro grammars. IEICE Transactions 91-D(2), 209–221 (2008)
[15] Kanazawa, M., Salvati, S.: The copying power of well-nested multiple context-free grammars. In: Dediu, A.-H., Fernau, H., Martín-Vide, C. (eds.) LATA 2010. LNCS, vol. 6031, pp. 344–355. Springer, Heidelberg (2010)
[16] Lautemann, C.: The complexity of graph languages generated by hyperedge replacement. Acta. Inf. 27(5), 399–421 (1990)
[17] Gold, E.M.: Language identification in the limit. Information and Control 10(5), 447–474 (1967)
[18] Clark, A.: Learning context free grammars with the syntactic concept lattice. In: [24], pp. 38–51
[19] Yoshinaka, R.: Towards dual approaches for learning context-free grammars based on syntactic concept lattices. In: Mauri, G., Leporati, A. (eds.) DLT 2011. LNCS, vol. 6795, pp. 429–440. Springer, Heidelberg (2011)
[20] Habel, A., Kreowski, H.: Some structural aspects of hypergraph languages generated by hyperedge replacement. In: Brandenburg, F.J., Wirsing, M., Vidal-Naquet, G. (eds.) STACS 1987. LNCS, vol. 247, pp. 207–219. Springer, Heidelberg (1987)
[21] Clark, A.: Efficient, correct, unsupervised learning of context-sensitive languages. In: Proceedings of CoNLL. Association for Computational Linguistics, Uppsala (2010)
[22] Charniak, E.: Tree-bank grammars. In: Proceedings of the Thirteenth National Conference on Artificial Intelligence, pp. 1031–1036 (1996)
[23] Chen, J., Bangalore, S., Vijay-Shanker, K.: Automated extraction of tree-adjoining grammars from treebanks. Nat. Lang. Eng. 12, 251–299 (2006)
[24] Sempere, J.M., García, P. (eds.): ICGI 2010. LNCS, vol. 6339. Springer, Heidelberg (2010)