1

Parsing with probabilistic strictly locally testable tree languages

Jose Luis Verdú-Mas, Rafael C. Carrasco, Jorge Calera-Rubio

Departament de Llenguatges i Sistemes Informàtics. Universidad de Alicante, E-03071 Alicante (Spain)

Index Terms

parsing with probabilistic grammars, stochastic learning, tree grammars.

Abstract

Probabilistic k-testable models (usually known as k-gram models in the case of strings) can be easily identified from samples and allow for smoothing techniques to deal with unseen events during pattern classification. In this paper, we introduce the family of stochastic k-testable tree languages and describe how these models can approximate any stochastic rational tree language. The model is applied to the task of learning a probabilistic k-testable model from a sample of parsed sentences. In particular, a parser for a natural language grammar that incorporates smoothing is shown.

I. INTRODUCTION

Stochastic models based on k-grams predict the probability of the next symbol in a sequence as a function of the $k-1$ previous symbols and have been widely used in natural language processing [1]–[3]. From a theoretical (albeit not historical) point of view, k-gram models can be regarded as a probabilistic extension of strictly locally testable string languages [4]. Strictly locally testable models or k-testable models [1], a kind of models that classify strings by looking

Work supported by the Spanish Comisión Interministerial de Ciencia y Tecnología through grants TIC2000-1599-C02 and TIC2003-08681-C02-01.

[1]In their original denomination [4], these models were called *k-testable in the strict sense* but, in the following, we will simply refer to them as k-testable.

at substrings of length at most k, are easy to learn from samples and can be dealt with using efficient and simple algorithms [5]–[7]. Is is also straightforward to incorporate probabilities into the model, although data sparseness often leads to the assignment of null probabilities to many strings. To alleviate this, there is a number of well-known techniques [8], [9] to smooth the distribution.

When hierarchical relations are established among the data components, trees become a more adequate representation than strings of symbols. This is the case, for example, in natural language parsing [10]–[13] or structured text processing [14], [15]. Context-free grammars [16] provide a traditional formalism that handles structural information. This kind of grammars can be easily written and updated by humans, although it is difficult to learn them automatically. For instance, it is hard to find the appropriate degree of generalization unless some information about the size of the target grammar is available [17].

The class of parse trees generated by a context-free grammar can be characterized as a rational tree language [18], [19] and any rational tree language can be recognized by a bottom-up (also called *frontier-to-root* or *ascending*) finite-state tree automaton.[2] The class of k-testable tree languages [21] is a proper subclass of the class of rational tree languages, where the effect of events that have occurred beyond a certain depth window are ignored when processing a tree. Then, k-testable tree models can be regarded as a special case of automata whose states are simply subtrees truncated to a certain depth. If valence (that is, number of children) remains bounded, the number of states in the automaton remains finite (and observable).

Stochastic models that assign a probability to a tree can be useful to select the best parse tree for a sentence and resolve structural ambiguity. For instance, in many natural-language processing applications, obtaining the correct syntactical structure for a sentence is an important intermediate step before assigning an interpretation to it. But ambiguous parses are very common in real natural-language sentences (e.g., those longer than 15 words; the fact that many ambiguous parse examples in books sound a bit awkward is due to the fact that they involve short sentences [22], [23, p. 411]). Choosing the correct parse for a given sentence is a crucial task if one wants to interpret the meaning of the sentence, due to the *principle of compositionality* [24,

[2]It is known [18], [20] that the class of languages that can be recognized by deterministic top-down (root-to-frontier or descending) tree automata is a proper subset of those recognizable by bottom-up automata.

p. 358], which states, informally, that the interpretation of a sentence is obtained by *composing* the meanings of its constituents according to the groupings defined by the parse tree[3].

As will be introduced in this paper, a class of probabilistic models analogous to the k-gram models can be defined for tree languages. In this probabilistic extension, the automaton behaves as a top-down generating device but the computation is still performed bottom-up: the state that generates a subtree can be recognized by a finite-state computation over the generated subtree or, more precisely, over those nodes lying within a depth k in the subtree.

Our notation is introduced in section II and some known properties of strictly locally testable tree languages are presented in section III. In section IV, we show how to obtain from a given, general, tree automaton the k-testable automaton that minimizes the relative entropy or Kullback-Leibler divergence [27] to the original one. In section V we justify how the best model from a stochastic sample is obtained. The application of these models to describe the syntactical structure of sentences is presented in section VI, together with some techniques to smooth the distributions. Experiments using one of these models, called *child-annotated* model for short, for structural disambiguation are presented in section VII and compared with other family-annotated models. Finally, conclusions are given in section VIII.

II. TREES AND TREE AUTOMATA

Given a finite set of symbols (called the *alphabet*) $\Sigma = \{\sigma_1, \ldots, \sigma_{|\Sigma|}\}$, the language T_Σ of Σ-trees can be defined recursively. On the one hand, all symbols in Σ are Σ-trees whose depth is zero. On the other hand, given a symbol $\sigma \in \Sigma$ and $m > 0$ Σ-trees $t_1, \ldots, t_m$, $\sigma(t_1 \cdots t_m)$ is a new Σ-tree in T_Σ of depth $1 + \max_i\{\text{depth}(t_i)\}$ and with root label σ. For instance, the depth 3 $\{a, b\}$-tree $a(a(a(ab))b)$ in $T_{\{a,b\}}$ is depicted in fig. 1. In the following, we will assume that Σ is given and then, we will simply use the name *trees* for all $t \in T_\Sigma$ and *labels* for all $\sigma \in \Sigma$. Any subset of T_Σ will be called a *tree language*. The *set of subtrees* of a tree t is

$$\text{sub}(t) = \begin{cases} \{\sigma\} & \text{if } t = \sigma \in \Sigma \\ \{t\} \cup \bigcup_{i=1}^{m} \text{sub}(t_i) & \text{if } t = \sigma(t_1 \cdots t_m) \in T_\Sigma - \Sigma \end{cases} \tag{1}$$

[3]The principle of compositionality in natural language is akin to the concept of *syntax-directed translation* used in compiler construction [25, p. 25] and to the principles which inspire the *syntactic transfer* architecture used in machine translation systems [26].

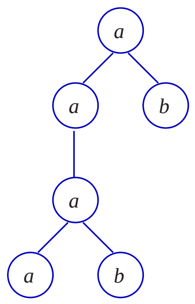

Fig. 1: A graphical representation of the tree $a(a(a(ab))b)$.

Finite-state automata are processing devices with a finite number of states. In contrast to the case of strings, where the automaton computes a new state for every prefix [16], a frontier-to-root tree automaton processes the tree in a bottom-up fashion and a state is computed for every subtree. If the subtree consists of a root node with m subtrees, the state is computed as a function of the root label and of the m states obtained after processing the subtrees. Therefore, the automaton needs a collection of transition functions, one for each possible value of m.

Formally, a *deterministic finite-state tree automaton* (DTA) is defined as $A = (Q, \Sigma, \Delta, F)$, where $Q = \{q_1, \ldots, q_{|Q|}\}$ is a finite set of states, $\Sigma = \{\sigma_1, \ldots, \sigma_{|\Sigma|}\}$ is the alphabet, $F \subseteq Q$ is the subset of accepting states and $\Delta = \{\delta_0, \delta_1, \ldots, \delta_M\}$ is a collection of transition functions of the form $\delta_m : \Sigma \times Q^m \to Q$. For every tree $t \in T_\Sigma$, the output $\delta(t) \in Q$ when A operates on t is

$$\delta(t) = \begin{cases} \delta_0(\sigma) & \text{if } t = \sigma \in \Sigma \\ \delta_m(\sigma, \delta(t_1), \ldots, \delta(t_m)) & \text{if } t = \sigma(t_1 \cdots t_m) \in T_\Sigma - \Sigma \end{cases} \tag{2}$$

For instance, if $\delta_0(a) = q_1$, $\delta_0(b) = q_2$, $\delta_2(a, q_1, q_2) = q_2$ and $\delta_1(a, q_2) = q_1$, the result of the operation of A on tree $a(a(a(ab))b)$ depicted in fig. 1, is $\delta(t) = \delta_2(a, \delta(a(a(ab))), \delta(b))$. Recursively, one gets $\delta(t) = \delta_2(a, q_1, q_2) = q_2$, as shown in fig. II.

The tree language $L(q)$ accepted by state $q \in Q$ is the subset of trees in T_Σ with output q

$$L(q) = \{t \in T_\Sigma : \delta(t) = q\} \tag{3}$$

and the tree language $L(A)$ recognized by the automaton A is the subset of trees in T_Σ whose output is a state in F

$$L(A) = \bigcup_{q \in F} L(q). \tag{4}$$

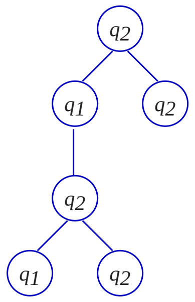

Fig. 2: Output states after the computation of the tree automaton A on $a(a(a(ab))b)$.

Each language that can be recognized by a DTA is a *rational tree language*. In case that a transition function δ_m is a partial mapping, all undefined transitions are assumed to lead to a special *absorption* state $\perp \notin F$. With this assumption, the languages $L(q)$ define a partition of T_Σ.

Probabilistic tree automata generate a probability distribution over the trees in T_Σ by using only a finite number of parameters and are more naturally seen as top-down generating devices. In particular, a probabilistic DTA incorporates a probability for every transition in the automaton, normalized so that the probabilities of the transitions leading to the same state add up to one. In other words, there is a collection of functions $P = \{p_0, p_1, p_2, \ldots, p_M\}$ of the type $p_m : \Sigma \times Q^m \rightarrow [0,1]$ such that they satisfy, for all $q \in Q$,

$$\sum_{\sigma\in\Sigma} \sum_{m=0}^{M} \sum_{\substack{i_1,\ldots,i_m \in Q: \\ \delta_m(\sigma,i_1,\ldots,i_m)=q}} p_m(\sigma, i_1, \ldots, i_m) \quad = \quad 1 \tag{5}$$

With this normalization, P defines a probability distribution[4] over every $L(q)$. In order to define a probability distribution over T_Σ, every *probabilistic deterministic tree automaton* (PDTA) $A = (Q, \Sigma, \delta, P, \rho)$ provides also a function $\rho : Q \rightarrow [0,1]$ which, for every $q \in Q$, gives the probability of the class $L(q)$ and satisfies

$$\sum_{q\in Q} \rho(q) = 1. \tag{6}$$

Then, the probability of a tree t in the stochastic language generated by A is defined as

$$p(t|A) = \rho(\delta(t))\, \pi(t) \tag{7}$$

[4]Although additional constraints are necessary to guarantee its consistency, see [28].

where $\pi(t)$ is the product of the probabilities of all the transitions performed when A operates on t, that is,

$$\pi(t) = \begin{cases} p_0(\sigma) & \text{if } t = \sigma \in \Sigma \\ p_m(\sigma, \delta(t_1), \ldots, \delta(t_m))\ \pi(t_1) \cdots \pi(t_m) & \text{if } t = \sigma(t_1 \cdots t_m) \in T_\Sigma - \Sigma \end{cases} \tag{8}$$

For instance, in the example used to draw fig. II, the probability of tree $a(a(a(ab))b)$ computed by the PDTA is $\rho(q_2)p_2(a, q_1, q_2)^2 p_1(a, q_2) p_0(b)^2 p_0(a)$. Any probability distribution over T_Σ defined by a PDTA will be called a *stochastic rational tree language*.

III. Strictly locally testable tree languages

Strictly locally testable languages are characterized, in the case of strings, by defining

1) the allowed set of substrings of length k and,
2) the sets of allowed prefixes and suffixes of length strictly smaller than k.

In the case of trees, as described by Knuutila [21], the concept of k-fork plays the role of the substrings and the k-root and k-subtrees play the role of the prefixes and suffixes. Every k-fork contains a node and all its descendents lying at a depth smaller that k. The k-root of a tree is its topmost k-fork and the k-subtrees are all the subtrees whose depth is smaller than k. These concepts are illustrated in fig. 3 and formally defined below.

For all $k > 0$ and for all trees $t = \sigma(t_1 \cdots t_m) \in T_\Sigma$, the *$k$-root* of t is the tree in T_Σ defined as

$$r_k(\sigma(t_1 \cdots t_m)) = \begin{cases} \sigma & \text{if } k = 1 \\ \sigma(r_{k-1}(t_1) \cdots r_{k-1}(t_m)) & \text{otherwise} \end{cases} \tag{9}$$

In case that $m = 0$, that is $t = \sigma \in \Sigma$, then $r_k(\sigma) = \sigma$.

On the other hand, the set $f_k(t)$ of *k-forks* and the set $s_k(t)$ of *k-subtrees* of a tree t are defined for all $k > 0$ as follows:

$$f_k(\sigma(t_1 \cdots t_m)) = \begin{cases} \emptyset & \text{if depth}(\sigma(t_1 \cdots t_m)) < k - 1 \\ \{r_k(\sigma(t_1 \cdots t_m))\} \cup \bigcup_{i=1}^{m} f_k(t_i) & \text{otherwise} \end{cases} \tag{10}$$

$$s_k(\sigma(t_1 \cdots t_m)) = \begin{cases} \sigma(t_1 \cdots t_m) \cup \bigcup_{i=1}^{m} s_k(t_i) & \text{if depth}(\sigma(t_1 \cdots t_m)) < k \\ \bigcup_{i=1}^{m} s_k(t_i) & \text{otherwise} \end{cases} \tag{11}$$

In the particular case $t = \sigma \in \Sigma$, then $s_k(t) = f_1(t) = \sigma$ and $f_k(t) = \emptyset$ for all $k > 1$. For instance, if $t = a(a(a(ab))b)$, then one gets $r_2(t) = \{a(ab)\}$, $f_3(t) = \{a(a(a)b), a(a(a(b)))\}$ and $s_2(t) = \{a(ab), a, b\}$, as shown in fig. 3.

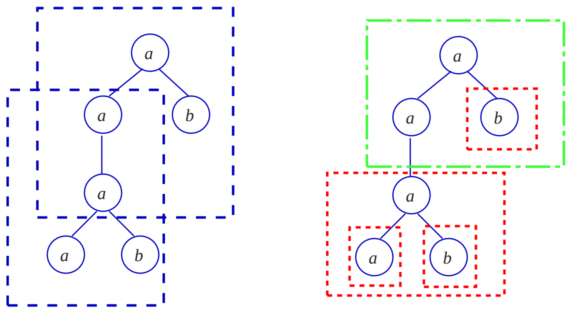

Fig. 3: Left: set of 3-forks contained in $a(a(a(ab))b)$. Right: 2-root (dash-dotted) and 2-subtrees.

A k-testable tree language consists of trees whose k-forks, $(k-1)$-subtrees and $(k-1)$-root belong to predefined sets. More precisely, a tree language $T \subseteq T_\Sigma$ is k-*testable* (with $k \geq 2$) if there exist three finite subsets $\mathcal{R}, \mathcal{F}, \mathcal{S} \subseteq T_\Sigma$ such that

$$t \in T \;\Leftrightarrow\; r_{k-1}(t) \in \mathcal{R} \;\wedge\; f_k(t) \subseteq \mathcal{F} \;\wedge\; s_{k-1}(t) \subseteq \mathcal{S}. \tag{12}$$

Without loss of generality, it can be assumed that $\mathcal{R} \subseteq r_{k-1}(\mathcal{F} \cup \mathcal{S})$ and $\mathcal{S} \subseteq s_{k-1}(\mathcal{F})$. With these assumptions, it can be proved that any k-testable language is rational [21]. Indeed, the DTA (Q, Σ, Δ, F) with

$$\begin{aligned} Q &= r_{k-1}(\mathcal{F} \cup \mathcal{S}) \\ F &= \mathcal{R} \\ \delta_m(\sigma, t_1, \ldots, t_m) &= \begin{cases} r_{k-1}(\sigma(t_1 \cdots t_m)) & \text{if } \sigma(t_1 \cdots t_m) \in \mathcal{F} \cup \mathcal{S} \\ \perp & \text{otherwise} \end{cases} \end{aligned} \tag{13}$$

recognizes the language T defined by equation (12). We will call the automaton obtained in this way a k-testable DTA.

As proved by Knuutila [21], the k-testable class can be identified in the limit [29] from positive samples and, given a sample set $\Omega = \{\tau_1, \tau_2, \tau_3, \ldots, \tau_{|\Omega|}\}$ of trees in the language T, using

$\mathcal{R} = r_{k-1}(\Omega)$, $\mathcal{F} = f_k(\Omega)$ and $\mathcal{S} = s_{k-1}(\Omega)$ defines the minimal k-testable model that recognizes Ω. As an illustration, consider the single-example sample $\Omega = \{a(a(a(ab))b)\}$ and $k = 3$. Then one gets the set of states $Q = \{a(ab), a(a), a, b\}$ with acceptance subset $F = \{a(ab)\}$. The defined transitions are $\delta_0(a) = a$, $\delta_0(b) = b$, $\delta_2(a, a, b) = a(ab)$, $\delta_2(a, a(a), b) = a(ab)$ and $\delta_1(a, a(ab)) = a(a)$.

In the following, we will call a *probabilistic k-testable tree automaton* any PDTA whose structure (that is, the transitions and states) is built according to (13) and will call *stochastic k-testable tree language* the distribution generated by a PDTA of this type. Therefore, stochastic k-testable tree languages are a proper subclass of stochastic rational tree languages. However, as it will be shown in the next section, a k-testable model that approximates a given stochastic rational tree language can be efficiently obtained.

IV. Approximating probabilistic DTA by k-testable automata

This section describes a procedure to build a probabilistic k-testable tree automaton upon a given DTA so that the relative entropy [27] to a second probabilistic automaton is minimal. In the case of strings, Stolcke and Segal [30] describe a procedure to obtain a k-gram model from a probabilistic context-free grammar, although its relation to the relative entropy is not shown. Here we will start by defining the states and transitions of the k-testable tree automaton and, then, we will describe how the probabilities are computed.

A. States and transitions

Assume that we are given a general probabilistic DTA $A = (Q, \Sigma, \Delta, P, \rho)$ and, for a certain $k > 1$, we want to obtain a probabilistic k-testable tree automaton $A' = (Q', \Sigma, \Delta', P', \rho')$ such that the relative entropy (see sect. IV-B) between A and A' is minimal. In order to find the states and transitions in A', one needs to know the set $\Theta^{[k]}$ of k-forks and $(k-1)$-subtrees that A may generate, that is,

$$\Theta^{[k]} = \{r_k(t) : t \in T_\Sigma \wedge \delta(t) \neq \perp\} = \bigcup_{q \neq \perp} r_k(L(q)) \tag{14}$$

The sets $r_k(L(q))$ can be efficiently built by means of an iterative procedure:

$$\begin{aligned} r_1(L(q)) \quad &= \quad \{\sigma \in \Sigma : \delta_0(\sigma) = q\} \cup \\ &\qquad \{\sigma \in \Sigma : \exists i_1, ..., i_m \in Q : \delta_m(\sigma, i_1, ..., i_m) = q\}. \end{aligned}$$

And for all $k > 1$,

$$r_k(L(q)) = \{\sigma \in \Sigma : \delta_0(\sigma) = q\} \cup \{\sigma(t_1 \cdots t_m) : \exists i_1, ..., i_m \in Q : \delta_m(\sigma, i_1, ..., i_m) = q \bigwedge \forall n\ [t_n \in r_{k-1}(L(i_n))]\ \} \quad (15)$$

From (13), the set of states Q' is simply given by $Q' = r_{k-1}(\Theta^{[k]}) = \Theta^{[k-1]}$ and Δ' contains $\delta'_m(\sigma, t_1, \ldots, t_m) = r_{k-1}(\sigma(t_1 \cdots t_m))$ if $\sigma(t_1 \cdots t_m) \in \Theta^{[k]}$ (and equals $\perp$ otherwise). For instance, let Δ in the automaton A contain the following transitions:

$$\begin{aligned} \delta_0(a) &= \delta_2(a, q_1, q_2) = q_1 \\ \delta_0(b) &= \delta_2(a, q_1, q_1) = q_2 \end{aligned} \quad (16)$$

with the rest being undefined. In such case, A is not k-testable and $\Theta^{[1]} = \{a, b\}$, $Q' = \Theta^{[2]} = \{a, b, a(aa), a(ab)\}$ and $\Theta^{[3]}$ contains the 14 trees depicted in fig. 4. Each tree in $\Theta^{[3]}$ corresponds to a defined transition in Δ': for instance, the first tree in the third line in the figure corresponds to $\delta_2(a, a(ab), b) = a(ab)$.

B. Probabilities

Next, we proceed to compute the probabilities in P' and ρ' that minimize the relative entropy. Indeed, the procedure will be also valid to obtain an approximate PDTA with the structure of any given DTA. Recall that the relative entropy $H(A, A')$ between two PDTA A and A' is $H(A, A') = G(A, A') - G(A, A)$ with

$$G(A, A') = -\sum_{t \in T_\Sigma} p(t|A) \log p(t|A') \quad (17)$$

The component that depends on the model A', that is $G(A, A')$, can be computed as the addition of two terms [31]:

$$G_\rho(A, A') = -\sum_{\substack{i \in Q \\ j \in Q'}} \eta_{ij}\ \rho(i)\ \log \rho'(j) \quad (18)$$

and

$$G_\pi(A, A') = \sum_{m=0}^{M} \sum_{\sigma \in \Sigma} \sum_{i_1, i_2, ..., i_m \in Q} \sum_{j_1, j_2, ..., j_m \in Q'} C_{\delta_m(\sigma, i_1, i_2, ..., i_m)}\ p_m(\sigma, i_1, i_2, ..., i_m) \log p'_m(\sigma, j_1, j_2, ..., j_m)\ \eta_{i_1 j_1} \eta_{i_2 j_2} ... \eta_{i_m j_m} \quad (19)$$

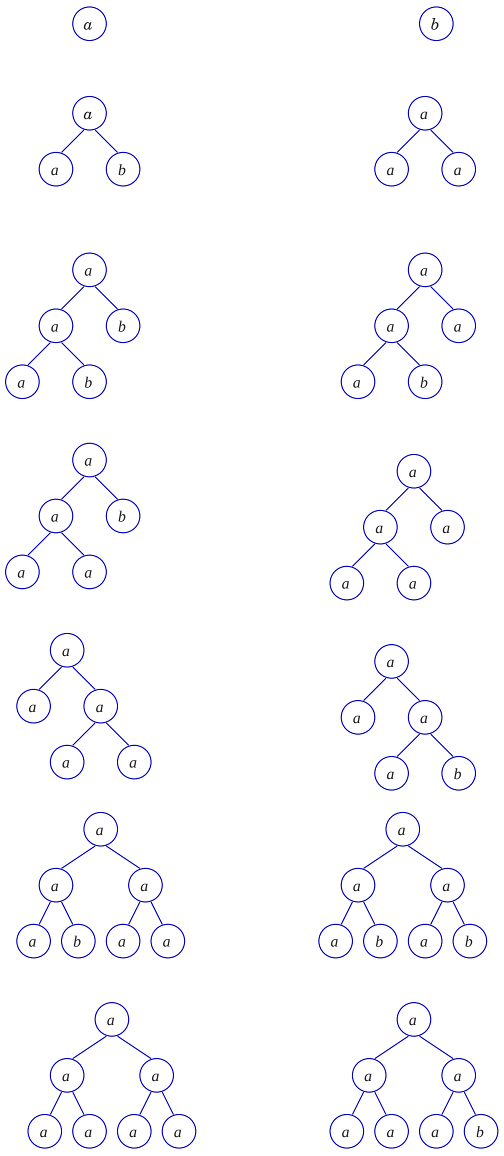

Fig. 4: Set $\Theta^{[3]}$ of 2-subtrees and 3-forks generated by the automaton A with the transitions defined in eq. (16).

where C_q is the expected number, according to the automaton A, of nodes of type q in a tree and η_{ij} represents the probability with respect to A that a node of type $i \in Q$ expands as a subtree t such that $\delta'(t) = j$. All the coefficients C_q and η_{ij} can be efficiently computed using iterative procedures as described in [31]:

$$\eta_{ij} = \sum_{m=0}^{M} \sum_{\sigma \in \Sigma} \sum_{\substack{i_1,\ldots,i_m \in Q: \\ \delta_m(\sigma,i_1,\ldots,i_m)=i}} \sum_{\substack{j_1,\ldots,j_m \in Q': \\ \delta'_m(\sigma,j_1,\ldots,j_m)=j}} p_m(\sigma, i_1, \ldots, i_m)\ \eta_{i_1 j_1} \eta_{i_2 j_2} \ldots \eta_{i_m j_m} \tag{20}$$

$$C_i = \rho(i) + \sum_{j \in Q} C_j \sum_{m=1}^{M} \sum_{\sigma \in \Sigma} \sum_{\substack{j_1, j_2, ..., j_m \in Q: \\ \delta(\sigma, j_1, ..., j_m) = j}} p_m(\sigma, j_1, j_2, ..., j_m) \sum_{n=1}^{m} D(i, j_n) \tag{21}$$

where $D(i, j) = 1$ if $i = j$ and zero otherwise.

In order to minimize G_ρ one has to take into account that ρ' is normalized according to eq. (6). Then, we will add a Lagrange multiplier μ when differentiating with respect to $\rho'(j)$

$$\frac{\partial G_\rho}{\partial \rho'(j)} + \mu = 0 \tag{22}$$

Then, one gets

$$\rho'(j) = \frac{1}{\mu} \sum_{i \in Q} \eta_{ij} \ \rho(i) \tag{23}$$

where $\mu = 1$ as a consequence of eq. (6) and the normalization $\sum_{j \in Q'} \eta_{ij} = 1$.

On the other hand, the probabilities in P' are tied by the set of normalizations (5). This means that when minimizing with respect to parameter $p'_m(\sigma, j_1, \ldots, j_m)$ a Lagrange multiplier $\nu_{\delta'_m(\sigma, j_1, \ldots, j_m)}$ has to be included. In this case, one gets

$$p'_m(\sigma, j_1, j_2, ..., j_m) = \\ \frac{1}{\nu_{\delta'_m(\sigma, j_1, j_2, ..., j_m)}} \sum_{i_1, i_2, ..., i_m \in Q} C_{\delta_m(\sigma, i_1, i_2, ..., i_m)} \ p_m(\sigma, i_1, i_2, ..., i_m) \ \eta_{i_1 j_1} \eta_{i_2 j_2} \cdots \eta_{i_m j_m} \tag{24}$$

The normalization (5), together with (20), entails that $\nu_j = \sum_{i \in Q} C_i \ \eta_{ij}$.

It is worth remarking that the equations (23) and (24) are also valid in case A is any given non-deterministic tree automaton, provided that the expressions (20) and (21) are suitably rewritten.

As an illustration, tables I and II describe a k-testable automaton A' with $k = 3$ that approximates the (non-k-testable) PDTA with Δ defined in eq. (16) and the following transition probabilities

$$\rho(q_1) = 1, \rho(q_2) = 0$$
$$p_0(a) = p_0(b) = 0.75,$$
$$p_2(a, q_1, q_2) = p_2(a, q_1, q_1) = 0.25$$

Table III shows how the relative entropy between the original model and the approximate one is small compared to the entropy of the automaton (in this example, $H(A) = G(A, A) = 1.6226$ bits). The Kullback-Leibler divergence or relative entropy between the exact model A

and the approximate one A' is computed using eqs. (18) to (21). As expected, the quality of the approximation improves as k increases at the expense of a considerable growth in the number of states, so that moderate values of k are to be preferred.

V. Learning stochastic strictly locally testable tree languages

A number of algorithms have been proposed in the past to build automata from stochastic samples. In some cases the result is a non-deterministic automaton [17], [32]; in others the automata are deterministic [33], [34]. Here, a stochastic sample $\Omega = \tau_1, \tau_2, \ldots \tau_{|\Omega|}$ consists of a sequence of (possibly repeated) trees generated according to an unknown probability distribution and a procedure to learn a probabilistic model from Ω can be derived using the results in former section.

Given a stochastic sample Ω, we will denote with $\hat{\Omega}$ the set of (different) trees in Ω. It is always possible to build a trivial PDTA A such that, for every tree t in the sample, $p(t|A)$ coincides with the relative frequency of t in the sample as follows:

$$Q = \text{sub}(\hat{\Omega})$$

$$\delta_m(\sigma, t_1, ..., t_m) = \begin{cases} \sigma(t_1 \cdots t_m) & \text{if } \sigma(t_1 \cdots t_m) \in Q \\ \perp & \text{otherwise} \end{cases} \tag{25}$$

$$p_m(\sigma, t_1, ..., t_m) = 1 \text{ for all } \sigma(t_1 \cdots t_m) \in Q$$

$$\rho(t) = \tfrac{1}{|\Omega|} \textstyle\sum_{n=1}^{|\Omega|} D(t, \tau_n)$$

Note that $\delta(t) = t$ for all trees in the sample.

On the other hand, as described after eq. (13), the minimal k-testable model A' recognizing $\hat{\Omega}$ has states $Q' = \Theta^{[k-1]}$ and transitions $\delta'_m(\sigma, t_1, ..., t_m) = r_{k-1}(\sigma(t_1 \cdots t_m))$ with

$$\Theta^{[k]} = \bigcup_{n=1}^{|\Omega|} f_k(\tau_n) \cup s_{k-1}(\tau_n) \tag{26}$$

Then, $\delta'(t) = r_{k-1}(t)$ for all $t \in \hat{\Omega}$.

In order to find the probabilities ρ' in the corresponding PDTA one should note that, in this case, η_{ij} is 1 if $r_{k-1}(i) = j$ and zero otherwise. Then, using (23), one gets

$$\begin{aligned} \rho'(j) &= \frac{1}{|\Omega|} \sum_{n=1}^{|\Omega|} \sum_{i \in Q} D(r_{k-1}(i), j)\ D(i, \tau_n) = \\ &= \frac{1}{|\Omega|} \sum_{n=1}^{|\Omega|} D(j, r_{k-1}(\tau_n)) \end{aligned} \tag{27}$$

Finally, in order to obtain P', one should take into account that in this case $C_i = \frac{1}{|\Omega|} \sum_n \Phi(i, \tau_n)$ where $\Phi(t, \tau)$ counts the number of nodes in τ that expand a subtree s such that $s = \tau$. Then, one gets for the denominator in (24)

$$\frac{1}{|\Omega|} \sum_{n=1}^{|\Omega|} E^{[k-1]}(\sigma(t_1 \cdots t_m), \tau_n) \tag{28}$$

where

$$E^{[k-1]}(j, \tau_n) = \sum_{i \in Q} \Phi(i, \tau_n) D(r_{k-1}(i), r_{k-1}(j)) \tag{29}$$

counts the number of nodes in τ_n that expand a subtree s such that $\delta'(s) = \delta'(j)$.

On the other hand, $r_{k-1}(i_1) = j_1, \ldots,$ and $r_{k-1}(i_m) = j_m$ if and only if $r_k(\sigma(i_1, ..., i_m)) = \sigma(j_1, ..., j_m)$. Taking this into account one gets for the numerator in (24)

$$\frac{1}{|\Omega|} \sum_{n=1}^{|\Omega|} \sum_{i_1 \cdots i_m \in Q} \Phi(\sigma(i_1 \cdots i_m), \tau_n) D(r_k(\sigma(i_1 \cdots i_m)), \sigma(j_1 \cdots j_m)) \tag{30}$$

and, therefore,

$$p'_m(\sigma, j_1, \ldots, j_m) = \frac{\sum_n E^{[k]}(\sigma(j_1 \cdots j_m), \tau_n)}{\sum_n E^{[k-1]}(\sigma(j_1 \cdots j_m), \tau_n)} \tag{31}$$

As expected, the equations (27) and (31) show that the closest automaton A' to the sample Ω is obtained when each transition is assigned a probability that is proportional to the number of times the transition is performed when processing the sample. In practice, it is useful to store the probabilities given by these equations as pairs (numerator, denominator), so that if a new tree τ is added to the sample Ω, the automaton A' can be easily updated to account for the additional information. For this incremental update, it suffices to increment each term in the pair with the partial sums obtained for τ.

In our example with $k = 3$ presented at the end of section III, where $\tau = a(a(a(ab))b)$, the relevant values (see fig. 3) are:

$$E^{[3]}(a(a(a)b),\tau) = E^{[3]}(a(a(ab)),\tau) = E^{[3]}(a(ab),\tau) = E^{[3]}(a,\tau) = 1$$
$$E^{[3]}(b,\tau) = 2$$
$$E^{[2]}(a(a),\tau) = E^{[2]}(a,\tau) = 1$$
$$E^{[2]}(a(a(a)b),\tau) = E^{[2]}(a(ab),\tau) = E^{[2]}(b,\tau) = 2.$$

Then one gets for the five transitions involved:

$$p_2(a,a(a),b)) = p_2(a,a,b) = 1/2$$
$$p_0(a) = p_0(b) = p_1(a,a(ab)) = 1$$

and $\rho(q_2) = 1$. All other probabilities are zero. If we compute, for instance, the probability of $a(a(a(ab)))b)$ with this model, the result is 1/4.

VI. Smoothed offspring annotated models of syntactical structure

Context-free grammars may be considered the customary way of representing syntactical structure in natural language sentences. A set of rather radical hypotheses as to how humans select the best parse tree [35] propose that a great deal of syntactic disambiguation may actually occur without the use of any semantic information, that is, just by selecting a preferred parse tree. It may be argued that the preference of a parse tree with respect to another is largely due to the relative frequencies with which those choices have lead to a successful interpretation. This sets the ground for a family of techniques which use a probabilistic scoring of parses to find the correct parse in each case.

Probabilistic scorings depend on parameters which are usually estimated from data, that is, from parsed text corpora such as the Penn Treebank [36]. The most straightforward approach is that of *treebank grammars* [37]. Treebank grammars are probabilistic context-free grammars (PCFG) in which the probability that a particular nonterminal is expanded according to a given rule is estimated as the relative frequency of that expansion by simply counting the number of times it appears in a manually-parsed corpus. Our probabilistic k-testable models can be regarded as *offspring-annotated* models where the expansion probabilities are dependent on the future expansions of children and include treebank grammars as a special case. Other models, such as Johnson's [38] *parent-annotated* models (or more generally, *ancestor annotated* models) and IBM history-based grammars [23, p. 423] [39] or Bod's and Scha's Data-Oriented parsing [40] offer an alternate approach in which the probability of expansion of a given nonterminal is made dependent on the previous expansions. An interesting property of many of these models is that, even though they may be seen as context-dependent, they may still be easily rewritten as context-free models in terms of specialized versions of the original nonterminals.

For instance, if $\Omega = \tau_1, \tau_2, \ldots, \tau_{|\Omega|}$ is a treebank, that is, a stochastic sample of parse trees, the alphabet Σ can be safely partitioned into the subset $s_1(\hat{\Omega})$ of labels that may only appear at leaves (usually lexical forms or part of speech tags) and its complementary subset $\Sigma - s_1(\hat{\Omega})$ of labels at internal nodes (syntactic tags). Then, we can define a probabilistic k-testable grammar as $G^{[k]} = (V^{[k]}, T, I, R^{[k]}, p^{[k]})$ where $V^{[k]} = \{I\} \cup r_{k-1}(f_k(\hat{\Omega}) \cup s_{k-1}(\hat{\Omega})) - s_1(\hat{\Omega})$ is the set of non-terminals; $T = s_1(\hat{\Omega})$ is the set of terminals; I is the start symbol; $R^{[k]}$ is the set of production rules and $p^{[k]}$ a probability function, builts as follows:

- For every tree $t \in r_{k-1}(\hat{\Omega})$ add the rule $I \rightarrow t$ to $R^{[k]}$ and compute its probability as

$$p^{[k]}(I \rightarrow t) = \frac{1}{|\Omega|} \sum_{n=1}^{|\Omega|} D(t, r_{k-1}(\tau_n))$$

- For every tree $\sigma(t_1 t_2 \cdots t_m) \in f_k(\hat{\Omega})$ add the rule $r_{k-1}(\sigma(t_1 t_2 \cdots t_m)) \rightarrow t_1 \ t_2 \ \ldots \ t_m$ to $R^{[k]}$ and compute its probability as

$$p^{[k]}(r_{k-1}(\sigma(t_1 t_2 \cdots t_m)) \longrightarrow t_1 \ t_2 \ \ldots \ t_m) = \frac{\sum_n E^{[k]}(\sigma(t_1 t_2 \cdots t_m), \tau_n)}{\sum_n E^{[k-1]}(\sigma(t_1 t_2 \cdots t_m)), \tau_n)}$$

- For every tree $\sigma(t_1 t_2 \ldots t_m) \in s_{k-1}(\hat{\Omega}) - T$ add the rule $\sigma(t_1 t_2 \cdots t_m) \rightarrow t_1 \ t_2 \ \ldots \ t_m$ to $R^{[k]}$, its probability being one:

$$p^{[k]}(\sigma(t_1 t_2 \ldots t_m) \longrightarrow t_1 \ t_2 \ \ldots \ t_m) = 1$$

The above rules and probabilities are analogous to those given by eqs. (27) and (31) and define a consistent probabilistic context-free grammar. [5]

With this definition, when $k = 2$, only the label of the node is taken into account and the k-testable model coincides with the simple rule-counting approach used in treebank grammars [37]. As an illustration, consider a tiny sample containing only the parse tree t depicted in fig. 5. Then, $r_1(t) =$ S, $f_2(t) = \{$S(NP VP), NP(N), VP(V NP), NP(NP PP), PP(P NP)$\}$, $s_1(t) = \{N, V, P\}$

[5]As shown in [41] and [42], maximum likelihood PCFGs preserve consistency.

and the PCFG is $G^{[2]} = (\{\text{I}, \text{S}, \text{NP,VP,PP}\}, \{\text{N,V,P}\}, \text{I}, R^{[2]}, p^{[2]})$ with rules and probabilities

$$\begin{array}{rcll} \text{I} & \to & \text{S} & (1) \\ \text{S} & \to & \text{NP VP} & (1) \\ \text{NP} & \to & \text{NP PP} & (1) \\ \text{NP} & \to & \text{N} & (0.75) \\ \text{VP} & \to & \text{VP PP} & (0.25) \\ \text{VP} & \to & \text{V NP} & (1) \\ \text{PP} & \to & \text{P NP} & (1) \end{array}$$

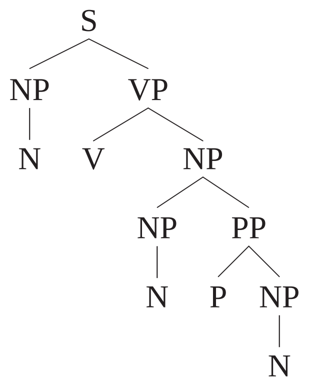

Fig. 5: An example of parse tree.

When $k = 3$ one gets a *child-annotated* model, that is, non-terminal symbols are of the form $\sigma(\sigma_1\,\sigma_2\,\ldots\,\sigma_m)$. In our example, $r_2(t)$ = S(NP VP), $f_3(t)$ = {S(NP(N) VP(V NP)), VP(V NP(NP PP)), NP(NP(N) PP(P NP)), PP(P NP(N))}, $s_2(t)$ = {NP(N), N, V, P} and the PCFG is $G^{[3]}$ = ({I, S(NP VP), NP(N), VP(V NP), NP(NP PP), PP(P NP)}, {N,V,P}, I, $R^{[3]}, p^{[3]}$) with rules and probabilities

$$\begin{array}{rcll} \text{I} & \longrightarrow & \text{S(NP VP)} & (1) \\ \text{S(NP VP)} & \longrightarrow & \text{NP(N) VP(V NP)} & (1) \\ \text{VP(V NP)} & \longrightarrow & \text{V NP(NP PP)} & (1) \\ \text{NP(NP PP)} & \longrightarrow & \text{NP(N) PP(P NP)} & (1) \\ \text{PP(P NP)} & \longrightarrow & \text{P NP(N)} & (1) \\ \text{NP(N)} & \longrightarrow & \text{N} & (1) \end{array}$$

In general, k-testable grammars with larger values of k contain more specialized rules and, therefore, are less ambiguous and allow for faster parsing. In contrast, typical treebank grammars have 100% coverage (as remarked in [37]) unlike with higher order grammars where sentences without a valid parse tree are not uncommon. Therefore, the use of smoothing techniques becomes necessary if one wants to use these models for parsing. Two classical techniques of this type are linear interpolation and backing-off [9].

Smoothing through *linear interpolation* is performed by computing the probability of events as a weighted average of the probabilities given by different models. For instance, the smoothed probability of a k-testable model can be computed as a weighted average of the probability given by the grammar $G^{[k]}$ and that given by the grammar $G^{[k-1]}$:

$$p(t|G^{[k]}_{\text{INT}}) = (1-\lambda)\, p(t|G^{[k]}) + \lambda\, p(t|G^{[k-1]}) \tag{32}$$

In turn, $p(t|G^{[k-1]})$ can be replaced by its smoothed version. The parameter $\lambda \in [0,1]$ can be chosen to maximize the likelihood of the sample. In practice, linear interpolation can be slow as it is dominated by the lower-order, more ambiguous, parse.

In contrast, backing-off allows one to avoid lower-order parsing when possible:

$$p^{[k]}(t|G^{[k]}_{\text{TREE}}) = \begin{cases} (1-\lambda)\, p(t|G^{[k]}) & \text{if } p(t|G^{[k]}) > 0 \\ \frac{\lambda}{\Lambda}\, p(t|G^{[k-1]}) & \text{otherwise} \end{cases} \tag{33}$$

where Λ is a normalization factor that can be computed as

$$\Lambda = 1 - \sum_{t:p(t|G^{[k]})>0} p(t|G^{[k-1]}). \tag{34}$$

If the parameter λ is small, this approach essentially parses the sentence with the higher-order grammar unless no parse tree is provided by this grammar. Because only in such a case the lower-order model is called, backing-off is faster than linear interpolation. However, the lack of a single rule in the sample can force the parser to use the lower-order model and, then, loose all the higher-order information for a whole sentence.

Here, we introduce an alternative approach: the use of rule-based backing-off. In this approach, the set of rules is generalized with the probabilities

$$p((r_{k-1}(\sigma(t_1 \cdots t_m)) \to t_1\, t_2 \cdots t_m | G^{[k]}_{\text{RULE}}) =$$
$$\begin{cases} (1-\lambda)\; p^{[k]}(r_{k-1}(\sigma(t_1 \cdots t_m)) \to t_1\, t_2 \cdots t_m) \text{ if } p^{[k]}(r_{k-1}(\sigma(t_1 \cdots t_m)) \to t_1\, t_2 \cdots t_m) > 0 \\ \frac{\lambda}{\Lambda(r_{k-1}(\sigma(t_1 \cdots t_m)))} \prod_{i: r_{k-2}(t_i) \neq t_i} p^{[k-1]}(r_{k-2}(t_i) \to t_i) \text{ otherwise} \end{cases} \tag{35}$$

where $t_1, \ldots, t_m$ are parse trees and

$$\Lambda(r_{k-1}(\sigma(t_1 \cdots t_m))) = 1 - \sum_{r_{k-1}(\sigma(t_1 \cdots t_m)) \to \tau_1 \cdots \tau_m \in R^{[k]}} \prod_{i: r_{k-2}(\tau_i) \neq \tau_i} p^{[k-1]}(r_{k-2}(\tau_i) \to \tau_i) \tag{36}$$

Using this rule-based backing-off requires the implementation of specific parsers, as building the whole grammar is unfeasible due to the large number of implicit rules (even if only those assigned a strictly positive probability in the last case of eq. (35) are considered). An alternative scheme that requires only minor modifications is using a quasi-equivalent grammar G' built as follows:

1) Add the rules in $R^{[k]}$ to R' with probabilities
$$p'(X \to \alpha) = (1-\lambda)\, p^{[k]}(X \to \alpha)$$
2) For every variable $\sigma(t_1\, t_2 \cdots t_m)$ in $V^{[k]}$ add a rule $\sigma(t_1\, t_2\, \cdots\, t_m) \to t_1\, t_2 \ldots t_m$ to R' whose probability is
$$p'(\sigma(t_1\, t_2\, \cdots\, t_m) \to t_1\, t_2 \ldots\, t_m) = \frac{\lambda}{\Lambda(\sigma(t_1 \cdots t_m))}$$
3) For every rule $r_{k-2}(\sigma(t_1\, t_2 \cdots t_m)) \to t_1\, t_2 \cdots\, t_m$ in $R^{[k-1]}$ add $r_{k-2}(\sigma(t_1\, t_2 \cdots t_m)) \to r_{k-1}(\sigma(t_1\, t_2 \cdots\, t_m))$ to R' and the probability
$$p'(r_{k-2}(\sigma(t_1\, t_2 \cdots t_m)) \to r_{k-1}(\sigma(t_1\, t_2 \cdots\, t_m))) =$$
$$p^{[k-1]}(r_{k-2}(\sigma(t_1\, t_2 \cdots t_m)) \to t_1\, t_2 \cdots\, t_m)$$

As shown in fig. 6, the parse trees generated by this grammar G' and their probabilities are, after an adequate projection ζ, identical to those generated by using (35): if a node with label X has as single descendent a subtree $Y(\alpha)$ and the depth of label X is smaller than that of label Y, the projection operation ζ has to be applied so that $\zeta(X(Y(\alpha))) = Y(\alpha)$. Then, G' can be used with a standard chart parser provided that some care is taken to avoid selecting a projected subtree whenever the same subtree can be obtained without projection. This can be checked by redefining the suitable comparison operator so that t_1 is not a better tree than t_2 if $\zeta(t_1) = \zeta(t_2) = t_2$ (even if $p(t_1|G') > p(t_2|G')$.

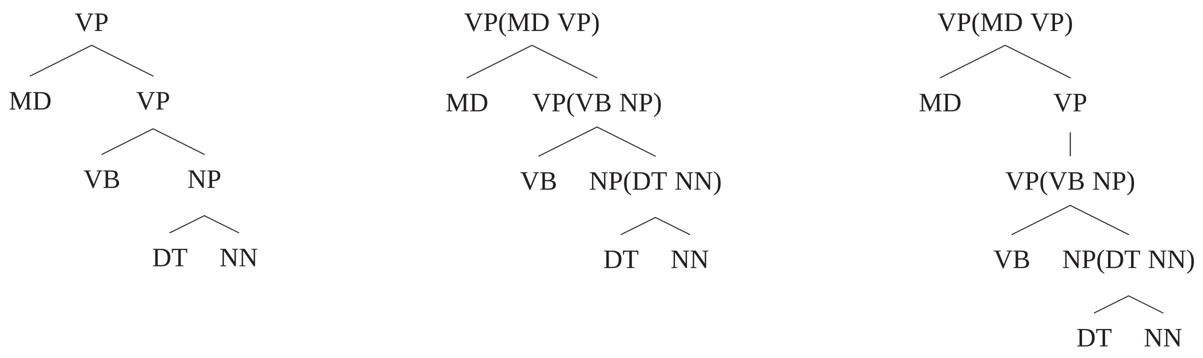

Fig. 6: Parse tree obtaiend with $G^{[2]}$ (left); parse tree obtained with $G^{[3]}$, invalid if the rule VP(MD VP) $\rightarrow$ MD VP(VB NP) is missing in $R^{[3]}$ (middle); parse tree obtained with G' where backoff rules VP(MD VP) $\rightarrow$ MD VP and VP $\rightarrow$ VP(VB NP) have been applied (right).

VII. Experiments

A. General conditions

We have performed a series of experiments to assess the structural disambiguation performance of offspring-annotated models as compared to standard treebank grammars, that is, to compare their relative ability for selecting the best parse tree. To better put these comparisons in context, we have also evaluated Johnson's [38] parent annotation scheme. To build training corpora and test sets of parse trees, we have used English parse trees from the Penn Treebank, release 3, with small, basically structure-preserving modifications (consistently with previous work [37], [38]):

- insertion of a root node, $I =$ ROOT, in all sentences, to encompass the sentence and final periods, etc.;
- removal of non-syntactic annotations (prefixes and suffixes) from constituent labels (for instance, NP-SBJ is reduced to NP);
- removal of empty constituents; and
- collapse of single-child nodes with the parent node when they have the same label.[6]

In all experiments the training corpus, consisted of all of the trees (41,532) in sections 02 to 22 of the *Wall Street Journal* portion of Penn Treebank, modified as above. This gives a total

[6]As most cases are labelly mistakes.

number of more than 600,000 subtrees. The test set contained all sentences in section 23 having less than 40 words.

All grammar models were written as standard context-free grammars, and a chart parser [43] was used to obtain the most likely parse for each sentence in the training set. This parse was compared to the corresponding gold-standard tree in the test set using the customary PARSEVAL evaluation metric [22], [23, p. 432] after deannotating the most likely tree delivered by the parser. PARSEVAL gives partial credit to incorrect parses by establishing three measures:

- *labeled precision* (P) is the fraction of correctly-labeled nonterminal bracketing (constituents) in the most likely parse which match the gold-standard parse,
- *labeled recall* (R) is the fraction of brackets in the gold-standard parse which are found in the most likely parse with the same label, and
- *crossing brackets* (X) refers to the fraction of constituents in one parse cross over constituent boundaries in the other parse.

The crossing brackets measure does not take constituent labels into account and will not be shown here. Some authors (see, e.g. [44]) have questioned partial-credit evaluation metrics such as the PARSEVAL measures. In particular, if one wants to use a probability model to perform structural disambiguation before assigning some kind of interpretation to the parsed sentence, it may well be argued that the exact match between the gold-standard tree and the most likely tree is the only possible relevant measure. It is however known that the Penn Treebank, even in its release 3, still suffers from problems. One of the problems worth mentioning (discussed in detail by Krotov et al. [45]) is the presence of far too many partially bracketed constructs according to rules like NP $\rightarrow$ NN NN CC NN NNS which lead to very flat trees, when one can, in the same treebank, find rules such as NP $\rightarrow$ NN NN, NP $\rightarrow$ NN NNS and NP $\rightarrow$ NP CC NP, which would lead to more structured parses such as that depicted in fig. 7.

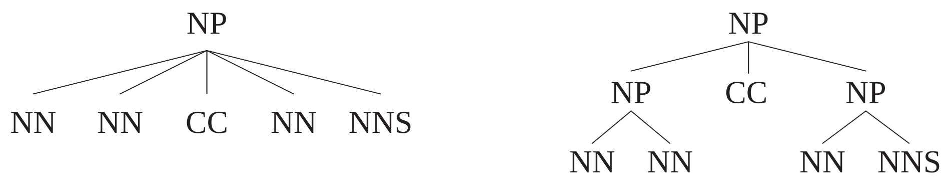

Fig. 7: Flat and structured parse of a sentence

Some of these flat parses may indeed be too flat to be useful for semantic purposes and have little linguistic plausibility; therefore, if one gets a more refined parse, one may consider it to be the one leading to the correct interpretation or not, but it surely contains more information than the flat, unstructured one. For this reason, we have chosen to give, in addition to the exact-match figure, the percentage of trees having 100% recall, because these are the trees in which the most likely parse is either exactly the gold-standard parse or a refinement thereof in the sense of the previous example.

B. Structural disambiguation results

The models which were evaluated are the following:

1) A standard treebank grammar [37], with no annotation of node labels (15,140 rules).
2) A parent-annotated grammar [38], with information at each node of the parent's label (23,020 rules).
3) A linear interpolation of $k = 3$ and $k = 2$ k-testable grammars with parameter $\lambda = 0.7$ selected to maximize the likelihood of the sample.
4) A 3-testable grammar (92,830 rules) smoothed with tree-based backing-off.
5) A 3-testable grammar smoothed with rule-based backing-off (adding 10,250 rules of the second type described at the end of former section) with parameter $\lambda = 0.005$ selected to maximize the sum of recall and precision.

As expected, the number of rules in the model increases as more information is conveyed by the node label, although this increase is not extreme. On the other hand, as the generalization power decreases, some sentences in the test would become unparsable by the k-testable grammar if the unsmoothed model was used. For instance, in our experiments, the child-annotated model was able to parse 94.6% of the sentences.

The results in table IV show that the best parsing performance is obtained using a child-annotated grammar with a rule-based backing-off. The results obtained with parent-annotated models are comparable (except for 100% recall). However, parsers using child-annotated grammars are much faster because the number of possible parse trees considered is drastically reduced.

It may be worth mentioning that parse trees produced by child-annotated models tend to be more structured and refined than parent-annotated and unannotated parses which tend to use rules that lead to flat trees in the sense mentioned above. Favoring structured parses may be convenient in some applications (especially where meaning has to be compositionally computed) but may be less convenient than flat parses when the structure obtained is incorrect.

VIII. Conclusions

We have introduced a class of probabilistic models (the strictly k-testable class) that describes tree languages and can be easily estimated from treebank samples. We have found an efficient procedure to compute the probabilities of the k-testable model that approximates any given stochastic rational tree language. The relative entropy between the k-testable model and the original one can also be efficiently computed and, as expected, the divergence becomes smaller as k increases. These strictly locally testable models can be updated incrementally and are adequate to apply backing-off techniques [9] or to be used as starting point by more complex learning algorithms [34], [46].

Fixed-depth probabilistic models may be seen as a special PCFG in which nonterminals are specialized for each possible offspring (up to a certain depth) and so are the expansion probabilities of rules. This may help model linguistic phenomena such as the differential distribution of determinants in noun phrases acting like a subject or like an object. We have introduced a procedure to build k-testable grammars smoothed by using a rule-based backing-off approach. In particular, we have compared the parsing performance of these models with that of unannotated PCFG and of parent-annotated PCFG [38]. As future work we plan to study the use of statistical confidence criteria to eliminate unnecessary annotations by merging states, therefore reducing the number of parameters to be estimated.

Acknowledgments

We thank Mikel L. Forcada for his useful suggestions and help with linguistic insight.

References

[1] G. Riccardi, R. Pieraccini, and E. Bocchieri, “Stochastic automata for language modeling,” *Computer Speech and Language*, vol. 10(4), pp. 265–293, 1996.

[2] J. Hu, W. Turin, and M. K. Brown, “Language modeling with stochastic automata,” in *Proc. ICSLP ’96*, vol. 1, Philadelphia, PA, 1996, pp. 406–409. [Online]. Available: http://www.asel.udel.edu/icslp/cdrom/vol1/996/a996.pdf

[3] Y. Esteve, F. Bechet, A. Nasr, and R. D. Mori, “Stochastic finite state automata language model triggered by dialogue states,” in *Proc. of Eurospeech*, 2001, pp. 725–728.

[4] R. McNaughton and S. Papert, *Counter-Free Automata*. Cambridge, Mass.: MIT Press, 1971.

[5] P. García and E. Vidal, "Inference of k-testable languages in the strict sense and application to syntactic pattern recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 9, pp. 920–925, sep 1990.

[6] T. Yokomori, "On polynomial-time learnability in the limit of strictly deterministic automata," *Machine Learning*, vol. 19, no. 2, pp. 153–179, 1995.

[7] T. Yokomori, N. Ishida, and S. Kobayashi, "Learning local languages and its application to protein alpha-chain identification," in *Proc. 27th Hawaii International Conference on System Sciences*, 1994, pp. 113–122. [Online]. Available: citeseer.ist.psu.edu/article/yokomori96learning.html

[8] K. W. Church and W. Gale, "A comparison of the enhanced Good-Turing and deleted estimation methods for estimating probabilities of English bigrams," *Computer Speech and Language*, vol. 5, pp. 19–54, 1991.

[9] H. Ney, U. Essen, and R. Kneser, "On the estimation of small probabilities by leaving-one-out," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 17, no. 12, pp. 1202–1212, 1995.

[10] E. Charniak, *Statistical Language Learning*. MIT Press, 1993.

[11] K. Sima'an, R. Bod, S. Krauwer, and R. Scha, "Efficient disambiguation by means of stochastic tree substitution grammars," in *Proc. of the Int. Conf. on New Methods in Language Processing. Manchester, UK, 14–16 Sep 1994*, D. B. Jones and H. L. Somers, Eds. London, UK: UCL Press, 1996, pp. 50–58.

[12] A. Stolcke, "An efficient context-free parsing algorithm that computes prefix probabilities," *Computational Linguistics*, vol. 21, no. 2, pp. 165–201, 1995.

[13] M. Thorup, "Disambiguating grammars by exclusion of sub-parse trees," *Acta Informatica*, vol. 33, no. 6, pp. 511–522, 1996.

[14] P. Prescod, "Formalizing XML and SGML instances with forest automata theory," University of Waterloo, Department of Computer Science, Waterloo, Ontario, Tech. Rep., 2000, draft Technical Paper.

[15] M. Murata, "Transformation of documents and schemas by patterns and contextual conditions," in *Principles of Document Processing, Third International Workshop, PODP'96 Proceedings*, C. K. Nicholas and D. Wood, Eds., vol. 1293, 1997, pp. 153–169.

[16] J. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Language, and Computation*. Reading, MA: Addison–Wesley, 1979.

[17] Y. Sakakibara, M. Brown, R. C. Underwood, I. S. Mian, and D. Haussler, "Stochastic context-free grammars for modeling RNA," in *Proceedings of the 27th Annual Hawaii International Conference on System Sciences. Volume 5 : Biotechnology Computing*, L. Hunter, Ed. Los Alamitos, CA, USA: IEEE Computer Society Press, Jan. 1994, pp. 284–294.

[18] F. Gécseg and M. Steinby, *Tree Automata*. Budapest: Akadémiai Kiadó, 1984.

[19] Y. Sakakibara, "Efficient learning of context-free grammars from positive structural examples," *Information and Computation*, vol. 97, no. 1, pp. 23–60, Mar. 1992.

[20] M. Nivat and A. Podelski, "Minimal ascending and descending tree automata," *SIAM Journal on Computing*, vol. 26, no. 1, pp. 39–58, 1997.

[21] T. Knuutila, "Inference of k-testable tree languages," in *Advances in Structural and Syntactic Pattern Recognition (Proc. Intl. Workshop on Structural and Syntactic Pattern Recognition, Bern, Switzerland)*, H. Bunke, Ed. World Scientific, 1993.

[22] E. Black, S. Abney, D. Flickinger, C. Gdaniec, R. Grishman, P. Harrison, D. Hindle, R. Ingria, F. Jelinek, J. Klavans, M. Liberman, M. Marcus, S. Roukos, B. Santorini, and T. Strzalkowski, "A procedure for quantitatively comparing the

syntatic coverage of English grammars," in *Proc. Speech and Natural Language Workshop 1991*. San Mateo, CA: Morgan Kauffmann, 1991, pp. 306–311.

[23] C. Manning and H. Schütze, *Foundations of Statistical Natural Language Processing*. Cambridge, US: The MIT Press, 1999.

[24] A. Radford, M. Atkinson, D. Britain, H. Clahsen, and A. Spencer, *Linguistics: an introduction*. Cambridge: Cambridge Univ. Press, 1999.

[25] A. V. Aho, R. Sethi, and J. D. Ullman, *Compilers Principles, Techniques, and Tools*. Addison Wesley, 1986.

[26] W. J. Hutchins and H. L. Somers, *An Introduction to Machine Translation*. New York: Academic Press, 1992.

[27] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, ser. Wiley Series in Telecommunications. New York, NY, USA: John Wiley & Sons, 1991.

[28] C. S. Wetherell, "Probabilistic languages: A review and some open questions," *ACM Computing Surveys*, vol. 12, no. 4, pp. 361–379, Dec. 1980.

[29] E. Gold, "Language identification in the limit," *Information and Control*, vol. 10, pp. 447–474, 1967.

[30] A. Stolcke and J. Segal, "Precise n-gram probabilities from stochastic context-free grammars," International Computer Science Institute, Berkeley, CA, Tech. Rep. TR-94-007, Jan. 1994.

[31] J. Calera-Rubio and R. C. Carrasco, "Computing the relative entropy between regular tree languages," *Information Processing Letters*, vol. 68, no. 6, pp. 283–289, 1998.

[32] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–285, Feb. 1989.

[33] R. C. Carrasco and J. Oncina, "Learning deterministic regular grammars from stochastic samples in polynomial time," *RAIRO (Theoretical Informatics and Applications)*, vol. 33, no. 1, pp. 1–20, 1999.

[34] R. C. Carrasco, J. Oncina, and J. Calera-Rubio, "Stochastic inference of regular tree languages," *Machine Learning*, vol. 44, no. 1/2, pp. 185–197, 2001.

[35] L. Frazier and K. Rayner, "Making and correcting errors during sentence comprehension: Eye movements in the analysis of structurally ambiguous sentences," *Cognitive Psychology*, vol. 14, pp. 178–210, 1982.

[36] M. P. Marcus, B. Santorini, and M. Marcinkiewicz, "Building a large annotated corpus of English: the Penn Treebank," *Computational Linguistics*, vol. 19, pp. 313–330, 1993.

[37] E. Charniak, "Tree-bank grammars," in *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference*. Menlo Park: AAAI Press/MIT Press, 1996, pp. 1031–1036.

[38] M. Johnson, "PCFG models of linguistic tree representations," *Computational Linguistics*, vol. 24, no. 4, pp. 613–632, 1998.

[39] E. Black, F. Jelinek, J. D. Lafferty, D. M. Magerman, R. L. Mercer, and S. Roukos, "Towards history-based grammars: Using richer models for probabilistic parsing," in *Proceedings of the DARPA Speech and Natural Language Workshop*, 1992, pp. 31–37.

[40] R. Bod and R. Scha, "Data-oriented language processing: An overview," Departement of Computational Linguistics, University of Amsterdam, The Netherlands, Tech. Rep. LP-96-13, 1996.

[41] J. Sáchez and J. Benedí, "Consistency of stochastic context-free grammars from probabilistic estimation based on growth transformations," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 9, pp. 1052–1055, 1997.

[42] Z. Chi and S. Geman, “Estimation of probabilistic context-free grammars,” *Computational Linguistics*, vol. 24, no. 2, pp. 299–305, 1998.

[43] J. Chappelier and M. Rajman, “A generalized CYK algorithm for parsing stochastic CFG,” in *Proceedings of Tabulation in Parsing and Deduction (TAPD’98)*, Paris (FRANCE), Apr. 1998, pp. 133–137. [Online]. Available: ftp://ftp.inria.fr/INRIA/Projects/Atoll/TAPD98/chappelier.ps.gz

[44] J. Carroll, T. Briscoe, and A. Sanfilippo, “Parser evaluation: A survey and a new proposal,” in *Proceedings of the International Conference on Language Resources and Evaluation*, Granada, Spain, 1998, pp. 447–454.

[45] A. Krotov, R. Gaizauskas, M. Hepple, and Y. Wilks, “Compacting the Penn Treebank grammar,” in *Proceedings of COLING/ACL’98*, 1998, pp. 699–703.

[46] F. Pereira and Y. Schabes, “Inside-outside re-estimation from partially bracketed corpora,” in *Proceedings of the 30th annual meeting of the ACL*, Newark, 1992, pp. 128–135.

$\rho'(a) = 0.75$	$\rho'(b) = 0$
$\rho'(a(ab)) = 0.1875$	$\rho'(a(aa)) = 0.0625$

TABLE I: Probabilities ρ' for the approximate 3-testable automaton A'. Note that the possible arguments of ρ' are the 2-roots of the trees in fig. 4.

$p_0(a) = 1$	$p_0(b) = 1$
$p_2(a, a, b) = 0.75$	$p_2(a, a, a) = 0.2812$
$p_2(a, a(ab), b) = 0.1875$	$p_2(a, a(ab), a) = 0.07031$
$p_2(a, a(aa), b) = 0.0625$	$p_2(a, a(aa), a) = 0.02344$
$p_2(a, a, a(aa)) = 0.3984$	$p_2(a, a, a(ab)) = 0.07031$
$p_2(a, a(ab), a(aa)) = 0.0996$	$p_2(a, a(ab), a(ab)) = 0.0175$
$p_2(a, a(aa), a(aa)) = 0.0332$	$p_2(a, a(aa), a(ab)) = 0.0059$

TABLE II: Probabilities in P' for the approximate 3-testable model after the same example automaton A. Each probability corresponds to a tree in fig. 4. The probabilities in the same block add up to 1.

k	$H(A, A')$	$\lvert\Delta'\rvert$
2	0.348687	4
3	0.165707	14
4	0.021793	100
5	0.010357	7102

TABLE III: Kullback-Leibler divergence (bits) in the example between the approximate automaton A' and the original one A (whose entropy is $H(A) = 1.6226$) for different values of k. The second column shows the number of transitions in Δ', that is, the size of the set $\Theta^{[k]}$.

GRAMMAR	R	P	$f_{R=100\%}$	EXACT
1 (Treebank grammar)	70.7%	76.1%	10.4%	10.0%
2 (Parent annotated)	80.0%	81.9%	18.5%	16.3%
3 (Linear interpolation)	80.2%	78.6%	21.7%	17.4%
4 (Tree-based back-off)	78.9%	74.2%	19.4%	17.1%
5 (Rule-based back-off)	82.4%	81.3%	25.7%	17.5%

TABLE IV: Parsing results with different grammars: labeled recall R, labeled precision P, fraction of sentences with total labeled recall $f_{R=100\%}$, fraction of exact matches.