

# Smoothing and Compression with Stochastic $k$ -testable Tree Languages<sup>\*</sup>

Juan Ramón Rico-Juan, Jorge Calera-Rubio<sup>1</sup> and  
Rafael C. Carrasco

*Departament de Llenguatges i Sistemes Informàtics  
Universitat d'Alacant, E-03071 Alacant, Spain*

---

## Abstract

In this paper, we describe some techniques to learn probabilistic  $k$ -testable tree models, a generalization of the well known  $k$ -gram models, that can be used to compress or classify structured data. These models are easy to infer from samples and allow for incremental updates. Moreover, as shown here, backing-off schemes can be defined to solve data sparseness, a problem that often arises when using trees to represent the data. These features make them suitable to compress structured data files at a better rate than string-based methods.

*Key words:* tree grammars, stochastic models, smoothing, backing-off, data compression

---

## 1 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 modeling [1, 2], speech recognition [3] and data compression [4]. Indeed, probabilistic models are a key component of arithmetic data compression algorithms [5, 6]. In classification tasks, the need of probabilistic models often arises when the Bayes' decision rule for minimum error rate is applied: given a sequence  $S = s_1s_2\dots$  of observations,

---

<sup>\*</sup> Work supported by the Spanish Comisión Interministerial de Ciencia y Tecnología through grant TIC2000-1599-C02.

<sup>1</sup> Corresponding author (calera@dlsi.ua.es)

the model  $M$  that maximizes the conditional probability  $P(M|S)$  also maximizes  $P(S|M)P(M)$  and, therefore, a model  $P(S|M)$  for the generation of sequences is needed. If the probabilities are computed as  $P(S = s_1s_2 \dots s_t|M) = p_M(s_1)p_M(s_2|s_1) \dots p_M(s_t|s_1s_2 \dots s_{t-1})$  and the conditional probabilities are assumed to depend only on the last  $k-1$  words,  $p_M(s_t|s_1 \dots s_{t-1}) = p_M(s_t|s_{t-k+1} \dots s_{t-1})$ , the resulting Markov chain model [7] is known as  $k$ -gram model.

From a theoretical (although not historical) point of view,  $k$ -gram models can be regarded as a probabilistic extension of locally testable languages [8]. Informally, a string language  $L$  is locally testable if every string  $w$  can be recognized as a string in  $L$  just by looking at all the substrings in  $w$  of length at most  $k$ . These models are easy to learn and can be efficiently processed [9, 10].

Whenever hierarchical relations are established among the pattern components, trees become a more natural representation of the input. For instance, probabilistic tree grammars have been widely used to tackle ambiguity in natural language parsing [11, 12, 13, 14] or to process structured text [15, 16, 17]. Context-free grammars [18] provide a traditional formalism that handles structural information. This kind of grammars can be easily written and updated by linguists, 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 [19].

The class of parse trees generated by a context-free grammar can be characterized as a rational tree language [20, 21, 22]. In particular, the class of  $k$ -testable tree languages [23] 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. These tree languages can be defined either using the formalism of tree automata (from the acceptor point of view) or using tree grammars (from the generation point of view). Both approaches are equivalent and previous work [23] has proposed identification algorithms for locally testable tree languages.

Therefore, learning tree languages represents a step towards identifying context-free languages. In case structural descriptions are available (that is, samples of unlabelled parse trees), the tree grammar can be identified by means of learning algorithms [21]. Even if structured samples are only partially available, this information can also be used to improve the learning process [24]. On their own, the identification of probabilistic models is of interest in classification tasks. Although some general algorithms have been proposed before [25], they require large data collections to infer even small probabilistic tree grammars. In contrast, probabilistic  $k$ -testable models (for instance,  $k$ -grams) represent an approximation to rational languages that provide a better description of small and medium-sized samples.

Probabilistic  $k$ -testable models have been applied to natural language processing before [26]. Here we also used the Penn Treebank data collection [27] in order to test the algorithm. However, instead of defining an ad hoc classification task, we have addressed another interesting problem: how much the inferred model helps to compress the type of data it describes. In case the model conveys a lot of information about the samples, a better compression rate will be obtained. For this purpose, some adaptation of the algorithm is needed: in particular, it is very important that any unseen tree is assigned a non-null probability.

Summarizing, in this paper, we explore the applicability of probabilistic locally testable models to describe structured data and complete the results presented in [28]. A brief description of these models can be found in section 2 and the basic guidelines to infer this kind of models are presented in section 3. We have checked (section 4) that using these models for adaptive compression of tree data improves performance compared to the traditional string-based arithmetic compression. Finally, how a backing-off scheme can be defined for classification tasks is described in section 5.

## 2 Probabilistic locally testable tree languages

Given an *alphabet*, that is, a finite set of symbols  $\Sigma = \{\sigma_1, \dots, \sigma_{|\Sigma|}\}$ , the set  $T_\Sigma$  of  $\Sigma$ -trees is defined as the language generated by the context-free grammar  $G = (\Sigma', \{T, F\}, T, R)$ , where the alphabet  $\Sigma'$  contains  $\Sigma$  plus the left and right parenthesis and whose set of productions  $R$  contains the rules:

- $T \longrightarrow \sigma \mid \sigma(F)$  for all  $\sigma \in \Sigma$
- $F \longrightarrow T \mid TF$

The *depth* of a tree  $t$  is

$$\text{depth}(t) = \begin{cases} 0 & \text{if } t = \sigma \in \Sigma \\ 1 + \max_{j=1}^m \{\text{depth}(t_j)\} & \text{if } t = \sigma(t_1 \dots t_m) \in T_\Sigma - \Sigma \end{cases} \quad (1)$$

and the subset of trees is

$$\text{sub}(t) = \begin{cases} \{\sigma\} & \text{if } t = \sigma \in \Sigma \\ \{t\} \cup \bigcup_n \text{sub}(t_n) & \text{if } t = \sigma(t_1 \dots t_m) \in T_\Sigma - \Sigma \end{cases} \quad (2)$$

For instance, the  $\Sigma$ -tree  $a(a(a(ab))b)$  belongs to  $T_{\{a,b\}}$  and its depth is 3. Its graphical representation is depicted in fig.1.

A *deterministic finite-state tree automaton* (DTA) is defined as a four-tuple  $A = (Q, \Sigma, \Delta, F)$ , where  $Q = \{q_1, \dots, q_{|Q|}\}$  is a finite set of states,  $\Sigma = \{\sigma_1, \dots, \sigma_{|\Sigma|}\}$  is the alphabet,  $F \subseteq Q$  is the subset of accepting states and  $\Delta = \{\delta_0, \delta_1, \dots, \delta_M\}$  is a collection of  $M$  transition functions of the form  $\delta_m : \Sigma \times Q^m \rightarrow Q$ . For all trees  $t \in T_\Sigma$ , the result  $\delta(t) \in Q$  of the operation of  $A$  on  $t$  is

$$\delta(t) = \begin{cases} \delta_0(\sigma) & \text{if } t = \sigma \in \Sigma \\ \delta_m(\sigma, \delta(t_1), \dots, \delta(t_m)) & \text{if } t = \sigma(t_1 \dots t_m) \in T_\Sigma - \Sigma \end{cases} \quad (3)$$

By convention, undefined transitions lead to an absorption state  $\perp \notin F$ .

For instance, if  $Q = \{q_1, q_2\}$ ,  $\Sigma = \{a, b\}$  and  $\delta_0(a) = q_1$ ,  $\delta_0(b) = q_2$ ,  $\delta_2(a, q_1, q_2) = q_2$  and  $\delta_1(a, q_2) = q_1$ , the result of the operation of  $A$  on tree  $t = a(a(a(ab))b)$ , plotted in fig.1, is  $\delta_2(a, \delta(a(a(ab))), \delta(b))$ . Recursively, one gets  $\delta(t) = \delta(a, q_1, q_2) = q_2$ .

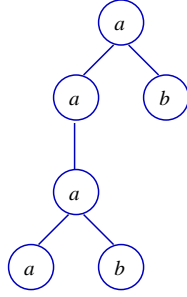


Fig. 1. A graphical representation of the tree  $a(a(a(ab))b)$ .

The tree language  $L(A)$  recognized by the automaton  $A$  is the subset of  $T_\Sigma$

$$L(A) = \{t \in T_\Sigma : \delta(t) \in F\}. \quad (4)$$

Every language that can be recognized by a DTA is called a *rational tree language* [20]. Rational tree languages can also be defined as the class of languages generated by *regular tree grammars* [22]. A regular tree grammar  $G = (N, \Sigma, S, R)$  consists of a finite set of variables  $V$ , an alphabet  $\Sigma$ , an axiom  $S \in V$ , and a set  $R$  of production rules. Every production rule has the form  $A \rightarrow \beta$  where  $A$  is a variable and  $\beta \in T_{V \cup \Sigma}$ . However, every regular tree grammar can be easily normalized so that all productions are of the form  $A \rightarrow \sigma(A_1 \dots A_n)$ , where  $\sigma \in \Sigma$  and  $A_1, \dots, A_n \in V$ . This form explicitly shows the equivalence between regular tree grammars and top-down tree automata [29, 22] which are, in turn, equivalent to the DTA defined here.

Probabilistic tree automata generate a probability distribution over the trees in  $T_\Sigma$ . A probabilistic DTA incorporates a probability for every transition in

the automaton, with the normalization that the probabilities of the transitions leading to the same state  $q \in Q$  must add up to one. In other words, there is a collection of functions  $P = \{p_0, p_1, p_2, \dots, p_M\}$  of the type  $p_m : \Sigma \times Q^m \rightarrow [0, 1]$  such that they satisfy, for all  $q \in Q$ ,

$$\sum_{\sigma \in \Sigma} \sum_{m=0}^M \sum_{\substack{q_1, \dots, q_m \in Q: \\ \delta_m(\sigma, q_1, \dots, q_m) = q}} p_m(\sigma, q_1, \dots, q_m) = 1 \quad (5)$$

With this normalization,  $P$  defines a distribution over every language  $L(q) = \{t \in T_\sigma : \delta(t) = q\}$ . In order to define a probability distribution over  $T_\Sigma$ , every *probabilistic deterministic tree automaton*  $A = (Q, V, \delta, P, \rho)$  provides a function  $\rho : Q \rightarrow [0, 1]$  which, for every  $q \in Q$ , gives the probability that a tree satisfies  $\delta(t) = q$ . Then, the probability of a tree  $t$  in the language generated by the probabilistic DTA  $A$  is given by

$$p(t|A) = \rho(\delta(t)) \pi(t) \quad (6)$$

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

$$\pi(t) = \begin{cases} p_0(\sigma) & \text{if } t = \sigma \in \Sigma \\ p_m(\sigma, \delta(t_1), \dots, \delta(t_m)) \pi(t_1) \cdots \pi(t_m) & \text{if } t = \sigma(t_1 \dots t_m) \in T_\Sigma - \Sigma \end{cases} \quad (7)$$

The equations (6) and (7) define a probability distribution  $p(t|A)$  provided that consistency is preserved:

$$\sum_{t \in T_\Sigma} p(t|A) = 1. \quad (8)$$

As shown by Chaudhuri et al. [30] and Sánchez et al. [31], context-free grammars whose probabilities are estimated from random samples are consistent. In the following, the probabilities of the DTA (or the equivalent regular tree grammar) will be extracted from random samples and, therefore, consistency is always preserved.

Locally testable languages, in the case of strings, are characterized by defining the set of substrings of length  $k$  together with prefixes and suffixes of length strictly smaller than  $k$  to check near the string boundaries [9, 10]. In the case of trees, as described by Knuutila [23], the concept of  $k$ -fork plays the role of the substrings and the  $k$ -root and  $k$ -subtrees play the role of prefixes and suffixes. For any  $k > 0$ , every  $k$ -fork contains a node and all its descendants lying at a depth smaller than  $k$ . The  $k$ -root of a tree is its shallowest  $k$ -fork and the  $k$ -subtrees are all the subtrees whose depth is smaller than  $k$ . These concepts are illustrated in fig. 2 for the tree  $t = a(a(a(ab))b)$ . In this example,  $r_2(t) = \{a(ab)\}$ ,  $f_3(t) = \{a(a(a)b), a(a(a(b)))\}$  and  $s_2(t) = \{a(ab), a, b\}$ .

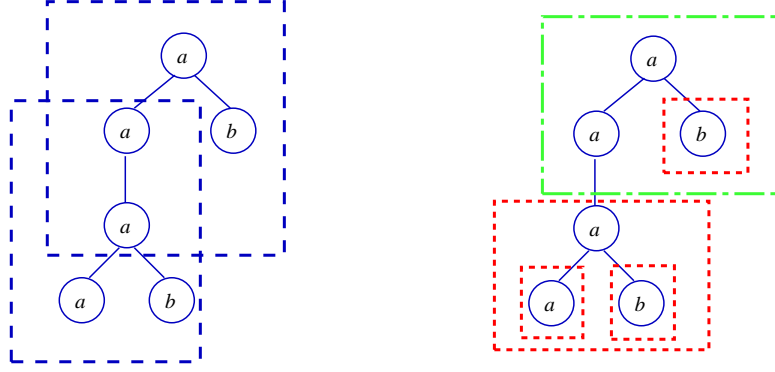


Fig. 2. Left: Set of 3-forks contained in  $a(a(a(ab))b)$ . Right: 2-root and 2-subtrees.

A tree language  $T$  is a *strictly  $k$ -testable* language (with  $k \geq 2$ ) if there exist three finite subsets  $\mathcal{R}, \mathcal{F}, \mathcal{S} \subseteq T_\Sigma$  such that

$$t \in T \Leftrightarrow r_{k-1}(t) \subseteq \mathcal{R} \wedge f_k(t) \subseteq \mathcal{F} \wedge s_{k-1}(t) \subseteq \mathcal{S}. \quad (9)$$

We will denote a language defined in this way as  $T = L_k(\mathcal{R}, \mathcal{F}, \mathcal{S})$ . Languages of this type are rational [23]. Indeed, it is straightforward to build a DTA  $A = (Q, \Sigma, \Delta, F)$  that recognizes  $L_k(\mathcal{R}, \mathcal{F}, \mathcal{S})$ . For this purpose, it suffices to choose:

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

Therefore, every strictly  $k$ -testable tree language ( $k$ -testable tree language in the following) is a rational tree language.

### 3 Inference of stochastic locally testable tree languages

As shown in [23], the class of  $k$ -testable tree languages is identifiable in the limit [32] from positive samples. In other words, there exists an algorithm that for every  $k$ -testable tree language  $L$  and after a finite number of examples outputs a DTA recognizing  $L$ . Essentially, the procedure to obtain the DTA given the sample  $S$  builds the DTA for  $L_k(r_{k-1}(S), f_k(S), s_{k-1}(S))$  following the construction at the end of former section.

For instance, for the single example sample  $S = \{a(a(a(ab))b)\}$  and  $k = 3$  one gets the set of states  $Q = \{a(ab), a(a), a, b\}$  with final subset  $F = \{a(ab)\}$  and transitions  $\delta_0(a) = a$ ,  $\delta_0(b) = b$ ,  $\delta_2(a, a, b) = a(ab)$ ,  $\delta_2(a, a(a), b) = a(ab)$  and  $\delta_1(a, a(ab)) = a(a)$ .

We can extend this learning procedure to the case where the sample is stochastically generated. A stochastic sample  $S = \{\tau_1, \tau_2, \dots, \tau_{|S|}\}$  consists of a sequence of trees generated according to an unknown probability distribution. The assumption that the underlying transition scheme (that is, the states  $Q$  and the collection of transition functions  $\Delta$ ) correspond to a  $k$ -testable DTA allows one to infer a probabilistic DTA from  $S$  in a simple way. For this purpose, one should note that the likelihood of the stochastic sample  $S$  is maximized [2] if the stochastic model assigns to every tree  $\tau$  in the sample a probability equal to the relative frequency of  $\tau$  in  $S$ . In other words, every transition in  $\Delta$  will be assigned a probability which coincides with the relative number of times the rule is used when the trees in the sample are parsed by the DTA. Therefore, the procedure to infer a probabilistic DTA from a stochastic sample  $S = \{\tau_1, \tau_2, \dots, \tau_{|S|}\}$  works as follows. The set of states is built as

$$Q = r_{k-1}(f_k(S) \cup s_{k-1}(S)); \quad (10)$$

and  $\Delta$  contains the transitions

$$\delta(\sigma, t_1, \dots, t_m) = \begin{cases} r_{k-1}(\sigma(t_1 \dots t_m)) & \text{if } \sigma(t_1 \dots t_m) \in f_k(S) \cup s_{k-1}(S) \\ \perp & \text{otherwise} \end{cases} \quad (11)$$

with probabilities

$$p_m(\sigma, t_1, \dots, t_m) = \frac{C^{[k]}(\sigma(t_1 \dots t_m), S)}{C^{[k-1]}(r_{k-1}(\sigma(t_1 \dots t_m)), S)} \quad (12)$$

where  $C(t, S) = \sum_{i=1}^{|S|} C(t, \tau_i)$  and  $C(t, \tau_i)$  counts the number of  $k$ -forks and  $(k-1)$ -subtrees<sup>2</sup> isomorphic to  $t$  found in  $\tau_i$ ; finally, the subset of accepting states is

$$F = r_{k-1}(S) \quad (13)$$

and the probabilities  $\rho(t)$  are estimated for every  $t \in F$  as

$$\rho(t) = \frac{1}{|S|} D^{[k]}(t, S) \quad (14)$$

where  $D^{[k]}(t, S) = \sum_{i=1}^{|S|} D^{[k]}(t, \tau_i)$  and  $D^{[k]}(t, \tau_i) = 1$  if  $r_{k-1}(\tau) = t$  and zero otherwise.

It is useful to store the above probabilities as the quotient of two terms, as given by equations (12) and (14). In this way, if a new tree (or subtree)  $\tau$  is provided, 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 equations with the partial sums obtained for  $\tau$ .

---

<sup>2</sup> Note that a tree  $\tau$  may contain, depending on the depth of  $t$ , either  $k$ -forks or  $(k-1)$ -subtrees isomorphic to  $t$  but not both simultaneously.

## 4 An application: tree data compression

In this section, we explore the application of the class of models considered here to the task of tree data compression. Because stochastic modeling becomes a key ingredient in arithmetic compression [5, 33], one expects that probabilistic tree models that provide a better description of the file content will allow for a more effective file compression. Conversely, compression performance can be used as a measure of the quality of the probabilistic model.

Recall that an arithmetic encoder uses at step  $n$  the cumulative range of probabilities  $[l(e_n), h(e_n)]$  that the model assigns to the event  $e_n$  having probability  $h(e_n) - l(e_n)$ . Starting with  $low_0 = 0$  and  $high_0 = 1$ , a new interval is iteratively computed as follows

$$\begin{aligned} low_{n+1} &= low_n + (high_n - low_n)l(e_n) \\ high_{n+1} &= low_n + (high_n - low_n)h(e_n) \end{aligned} \tag{15}$$

The output of the encoder is any number in the range obtained after the whole file is processed and implementing this computation using integer arithmetics is a subtle task as shown in [5]. An important issue is that the probabilistic model should never assign a null probability to any event that can be observed, that is,  $h(e_n) - l(e_n)$  has to be always strictly positive.

Our procedure to compress tree data follows the guidelines of prediction by partial matching (PPM, [34]). Similarly to the case of strings, a probabilistic model  $M^{[k]}$  predicts the next code to be transmitted based on the previous context. In this case, the context is given by the  $(k - 1)$ -fork (or, when appropriate, by the  $(k - 1)$ -root or  $(k - 1)$ -subtree) in the tree above the node. For instance, if  $k = 3$ , the possible expansions of the nodes shadowed in fig. 3 depend on the whole context marked with a square. In that case, the probability  $p_2(a, a(a), b)$  of the expansion  $t = a(a(a)b)$  given the observed state  $q = r_2(t) = a(ab)$  is needed. In the following, we will say that a tree  $t$  is an expansion if  $t \in f_k(S) \cup s_{k-1}(S)$  and say that  $t$  is a root expansion if  $t \in r_{k-1}(S)$ .

In case the  $k$ -testable model  $M^{[k]}$  contains no information about the expansion  $t = \sigma(t_1 \dots t_m)$ , the  $(k - 1)$ -order model  $M^{[k-1]}$  is used instead to compute the probabilities of the subexpansions  $t_1, \dots, t_m$ . This backing-off procedure is repeated recursively till either a) a suitable model is found for the subexpansion under consideration or b)  $k = 1$  and the ground model  $M^{[1]}$  is applied. On the return, when each state expansion is known, the models  $M^{[1]}$  to  $M^{[k]}$  are updated.

Some important features differentiate the procedure from the standard string

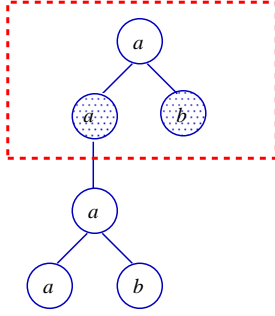


Fig. 3. Trailing context ( $k = 3$ ) for the expansion of the shadowed nodes.

PPM compression and are worth to comment:

- (1) In contrast to strings, where left-to-right processing is the natural choice, different orders are possible to perform a walk on the tree. Breadth-first traversal offers the advantage that the model can be updated before the whole tree is expanded. This improves compression of files consisting of a single tree or a small amount of them.
- (2) Each model  $M^{[k]}$  with  $k > 1$  consists of a collection of counters  $C^{[k]}(t)$  and  $D^{[k]}(t)$  needed for the estimates (12) and (14) respectively. There is a counter for every different argument  $t$ , where  $t$  is a  $k$ -fork or  $(k - 1)$ -subtree in the first case and a  $(k - 1)$ -root in the second case. In addition, one needs counters for the escape codes:  $D^{[k]}(\epsilon_r)$  for the root escape code  $\epsilon_r$  and  $C^{[k]}(\epsilon_q)$  for the escape code  $\epsilon_q$  associated to the state  $q$ . All these counters are initialized to zero, updated with the processed part of the tree, and they will be used to estimate the probabilities as follows. The probability in model  $M^{[k]}$  of the expansion  $t$  from state  $q = \sigma(t_1 \dots t_m)$  is given by

$$\alpha(k, t) = \frac{C^{[k]}(t)}{C^{[k-1]}(q) + C^{[k]}(\epsilon_q)} \quad (16)$$

The above formula can also be used for the escape probabilities  $\alpha(k, \epsilon_q)$ . Conversely, the probability in  $M^{[k]}$  of the root expansion  $t$  is

$$\beta(k, t) = \frac{D^{[k]}(t)}{|S| + D^{[k]}(\epsilon_r)} \quad (17)$$

This equation also holds for the escape code probability  $\beta(k, \epsilon_r)$ .

- (3) The ground model  $M^{[1]}$  is used when no information is available about a expansion  $t = \sigma(a_1 \dots a_m)$  with  $a_1, \dots, a_m \in \Sigma$  or  $t = \sigma$ . In such case, the state is  $q = r_1(t) = \sigma$  and one needs to code the number  $m$  of descendents together with, if  $m > 0$ , their labels  $a_1, \dots, a_m$ . Therefore, this ground model has two components. On the one hand, a collection of counters  $E_\sigma(m)$  stores how many nodes labeled  $\sigma$  expanded  $m$  subtrees. As the maximal tree-width  $M$  is in advance unknown, we initially use  $M$  counters per symbol,  $E_\sigma(0), \dots, E_\sigma(M - 1)$ , plus an additional one

$E_\sigma(\epsilon_\sigma)$  that stores how often  $M$  additional counters are needed. Then, the probability that a node labeled  $\sigma$  has  $m$  descendents is computed as follows

$$\gamma(\sigma, m) = \frac{1 + E_\sigma(m)}{1 + ME_\sigma(\epsilon_\sigma) + \sum_{i=0}^{ME_\sigma(\epsilon_\sigma)} E_\sigma(i)} \quad (18)$$

The same equation formally holds for  $\gamma(\sigma, \epsilon_\sigma)$ .

In addition,  $M^{[1]}$  keeps a counter  $C^{[1]}(\sigma)$  storing the number of nodes labeled with  $\sigma$  in the sample. They are used to assign a probability to every symbol as follows:

$$\alpha(1, \sigma) = \frac{1 + C^{[1]}(\sigma)}{|\Sigma| + \sum_{a \in \Sigma} C^{[1]}(a)} \quad (19)$$

With all these ingredients, the tree compression algorithm executes repeatedly a function, `tcompress` schematically represented in fig. 4, for every tree  $\tau$  in the input and the maximal order  $k_{\max}$  allowed for the models. The `tcompress`

```

algorithm tcompress( $k, \tau$ )      [for  $k > 1$ ]
  r_encode( $r_{k-1}(\tau)$ )
  do ( $\forall x$  subtree of  $\tau$  in breadth-first order)
     $t \leftarrow r_k(x)$ 
    if ( $1 + \text{depth}(t) \geq k - 1$ ) then
      f_encode( $k, t$ )
    endif
  enddo
endalgorithm

```

Fig. 4. Main tree compression algorithm.

algorithm calls a function `f_encode` (plotted in fig. 5) that encodes the  $k$ -forks and  $(k-1)$ -subtrees<sup>3</sup> and a similar function `r_encode` (plotted in fig. 7) that encodes the  $(k-1)$ -root. All these functions use a generic procedure `send`( $\varphi, \eta, x$ ) that generates the input for the arithmetic encoder, that is, the cumulative range of probabilities assigned to the event by the model. In this function,  $\varphi$  is either  $\alpha$ ,  $\beta$ , or  $\gamma$ , the parameter  $\eta$  represents a  $k$ - or  $\sigma$ -value and  $x$  is a tree or  $m$ -value. Both  $\varphi$  and  $\eta$  specify the table to be used, that is, the equation (16–19) and its first argument, while  $x$  is the parameter that selects the table entry (the second argument in these equations). The corresponding decoding functions are implemented in a similar fashion and can be found in the appendix.

In order to illustrate how the algorithm works we apply it with parameter  $k_{\max} = 3$  to the trees in 8. The steps performed are traced in table 9.

In this case, encoding starts with the 2-root  $a(ba)$ , but  $M^{[3]}$  is still empty.

<sup>3</sup> This means that the argument  $t$  for `f_encode` satisfies  $k - 1 \leq 1 + \text{depth}(t) \leq k$ .

```

function f_encode( $k, t$ )
   $q \leftarrow r_{k-1}(t)$            [ $k > 1$  is always true]
  if ( $C^{[k]}(t) > 0$ ) then           [ $t$  found in  $M^{[k]}$ ]
    send( $\alpha, k, t$ )
    update( $k, t$ )
  else           [ $t$  not found in  $M^{[k]}$ ]
    send( $\alpha, k, \epsilon_q$ )
    inc( $C^{[k]}(\epsilon_q)$ )
    if ( $k > 2$ ) then           [use  $M^{[k-1]}$  for  $t = \sigma(t_1 \dots t_m)$ ]
      do ( $j = 1, \dots, m$ )
        if ( $1 + \text{depth}(t_j) \geq k - 2$ ) then
          f_encode( $k - 1, t_j$ )
        endif
      enddo
    else           [use  $M^{[1]}$  for  $t = \sigma(a_1 \dots a_m)$ ]
      do (while  $m \geq ME_\sigma(\epsilon_\sigma)$ )
        send( $\gamma, \sigma, \epsilon_\sigma$ )
        inc( $E_\sigma(\epsilon_\sigma)$ )
      enddo
      send( $\gamma, \sigma, m$ )
      inc( $E_\sigma(m)$ )
      do ( $j = 1, \dots, m$ )
        send( $\alpha, 1, a_j$ )
        inc( $C^{[1]}(a_j)$ )
      enddo
    endif
    inc( $C^{[k]}(t)$ )
  endif
endfunction

```

Fig. 5. Encoding function. The command `inc` increments the counter by one. Function `update` is expanded in fig. 6.

Then a escape code is generated (with probability 1) and a smaller context ( $k = 2$ ) is tried in order to encode the 1-root  $a$ . As  $M^{[2]}$  is also empty a second escape code is generated (also with probability 1) and the ground model  $M^{[1]}$  is called. At this point, there is no context for the next symbol and then, the code for the label  $a$  is emitted and  $M^{[2]}$  tries to encode  $a(ba)$  from the state  $a$ , generating a call to `encode_p` with  $k = 2$ . As this expansion is still unseen, the escape code of state  $a$  is generated and the ground model is used to encode the number of children and their labels. This ends step 1 in the table.

In step 2, the 2-fork  $a(ba)$  has to be expanded as  $a(ba(ba))$ , but there is still no information in  $M^{[3]}$  so, the corresponding escape code is emitted and  $M^{[2]}$  is called with the smaller context  $b$ . Again, the state is not in the model, a

```

function update( $k, t$ )
  if ( $k > 2$ ) then                                [ $t = \sigma(t_1 \dots t_m)$ ]
    do ( $j = 1, \dots, m$ )
      if ( $1 + \text{depth}(t_j) \geq k - 2$ ) then update( $k - 1, t_j$ )
    enddo
  else                                             [ $t = \sigma(a_1 \dots a_m)$ ]
    inc( $E_\sigma(m)$ )
    do ( $j = 1, \dots, m$ )
      inc( $C^{[1]}(a_j)$ )
    enddo
  endif
  inc( $C^{[k]}(t)$ )
endfunction

```

Fig. 6. Function update.

```

function r_encode ( $k, t$ )
  if ( $D^{[k]}(t) > 0$ ) then                          [use  $M^{[k]}$ ]
    send( $\beta, k, t$ )
    do ( $i = 2, \dots, k$ )
      inc( $D^{[i]}(r_{i-1}(t))$ )
      update( $i - 1, r_{i-1}(t)$ )
    enddo
  else                                             [use  $M^{[k-1]}$ ]
    send( $\beta, k, \epsilon_r$ )
    inc( $D^{[k]}(\epsilon_r)$ )
    if ( $k > 2$ ) then                                [ $t = \sigma(t_1 \dots t_m)$ ]
      r_encode ( $k - 1, r_{k-1}(t)$ )
      if ( $1 + \text{depth}(t) \geq k - 1$ ) f_encode ( $k - 1, t$ )
    else                                           [ $t = \sigma$ ]
      send( $\alpha, 1, \sigma$ )
      inc( $C^{[1]}(\sigma)$ )
    endif
    inc( $D^{[k]}(t)$ )
  endif
endfunction

```

Fig. 7. Root encoding function.

escape code is emitted and the ground model is called to encode zero children. Then,  $M^{[2]}$  is called with context  $a$  but now, this expansion is already stored in  $M^{[2]}$  since after step 1, the models were updated. Therefore, the code of this expansion is generated.

The execution is still not finished, as the fact that  $a$  and  $b$  are leaves in the fork

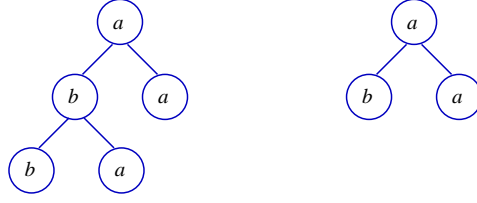


Fig. 8. Trees used to generate a sample trace of the encoding algorithm.

Step	Function	Action
1	encode_r( $a(ba), 3$ )	send_r( $\varepsilon, 3$ )
	encode_r( $a, 2$ )	send_r( $\varepsilon, 2$ )
		send_1( $a$ )
	encode_p( $a(ba), 2$ )	send( $\varepsilon, a, 2$ )
		send_1M( $2, a$ )
		send_1( $b$ )
		send_1( $a$ )
2	encode_p( $a(ba(ba)), 3$ )	send( $\varepsilon, a(ba), 3$ )
	encode_p( $b, 2$ )	send( $\varepsilon, b, 2$ )
		send_1M( $0, b$ )
	encode_p( $a(ba), 2$ )	send( $a(ba), a, 2$ )
3	encode_p( $a(ba), 3$ )	send( $\varepsilon, a(ba), 3$ )
	encode_p( $b, 2$ )	send( $b, b, 2$ )
	encode_p( $a, 2$ )	send( $\varepsilon, a, 2$ )
		send_1M( $0, a$ )
4	encode_r( $a(ba), 3$ )	send_r( $a(ba), 3$ )
	encode_p( $a(ba), a(ba), 3$ )	send( $a(ba), a(ba), 3$ )

Fig. 9. Trace of the algorithm when processing the trees in fig. 8

$a(ab)$  is still not coded and the model  $M^{[3]}$  does not contain such information. In step 3, after a new escape code, the  $M^{[2]}$  code corresponding to state  $b$  without children is generated and a second escape code is emitted because  $M^{[2]}$  does not contain an expansion of state  $a$  with no children. Then, the ground model is used for this purpose.

Finally, step 4 describes the calls performed while the second tree is encoded. At this point  $M^{[3]}$  has the information needed about the state and the expansion and two calls are enough for the encoding.

We checked this model with a 6.129 MByte file consisting of parse trees (with structural and part-of-speech tags) contained in the Penn Treebank [27]. The compression rate obtained, 13.94, is to be compared with 7.08 obtained using `gzip` with best compression options, 10.92 obtained with `bzip2` with longest block-size or 9.08 using trigram-based arithmetic compression. This is a clear indication that the probabilistic  $k$ -testable tree models provide the arithmetic encoding algorithm with a suitable description of the structured data contained in the corpus.

## 5 Multilevel smoothing of tree languages

A standard procedure to avoid null probabilities is the backing-off method that has been extensively studied for string models (see, for instance, [2]). In this section we introduce a backing-off procedure that has been successfully used in classification tasks. Different schemes are possible but the results are not very sensitive to the details of this choice [2] and our main purpose is to illustrate the difficulties and differences with the string case. Indeed, the most important difference comes from the fact that the number of descendants of a node is not bounded: on the one hand a special ground model is needed and, on the other hand, special care has to be taken when calculating the normalization factors as the number of unseen events is not known.

Following the standard approach, the template used to back-off a probability distribution  $p_A$  with a second, more general, one  $p_B$  is as follows:

$$p(x) = \begin{cases} p_A(x)(1 - \lambda(x)) & \text{if } p_A(x) > 0 \\ \frac{\Lambda}{F} p_B(x) & \text{otherwise} \end{cases} \quad (20)$$

The distribution  $p_A$  is decreased with a discounting function  $\lambda$  such that  $0 < \lambda(x) < 1$  and originates a discount term

$$\Lambda = \sum_{x:p_A(x)>0} \lambda(x)p_A(x)$$

which is distributed among the unseen events. The factor  $F$  is needed to keep the normalization:

$$F = \sum_{x:p_A(x)=0} p_B(x)$$

In our case, the probability of a tree  $\tau = \sigma(\tau_1 \dots \tau_m)$  in model  $M^{[k]}$  is computed using the definitions (6) and (7) adapted to a  $k$ -testable language, so that

$p(\tau|M^{[k]}) = \rho^{[k]}(r_{k-1}(\tau))\pi^{[k]}(\tau)$  and

$$\pi^{[k]}(\tau) = \begin{cases} p_0^{[k]}(\sigma) & \text{if } \tau = \sigma \in \Sigma \\ p_m^{[k]}(\sigma, r_{k-1}(\tau_1), \dots, r_{k-1}(\tau_m)) \prod_{i=1}^m \pi^{[k]}(\tau_i) & \text{if } \tau = \sigma(\tau_1 \dots \tau_m) \in T_\Sigma - \Sigma \end{cases} \quad (21)$$

We will define a backing-off scheme for the three types of elementary probabilities in Eq. (21):  $p_0(\sigma)$ ,  $p_m(\sigma, t_1, \dots, t_m)$  with  $m > 0$  and  $\rho(t)$  following the template (20). For this purpose, we will use counters  $C^{[k]}$ ,  $D^{[k]}$  and  $E_\sigma$  with the same meaning as in former section.

Sometimes, we will need to predict the number  $m$  of descendents of a node labeled  $\sigma$ . For this purpose we define

$$p_\sigma^{[1]}(m) = \begin{cases} \frac{E_\sigma(m)}{C^{[1]}(\sigma)} (1 - \lambda_\sigma^{[1]}(m)) & \text{if } E_\sigma(m) > 0 \\ \frac{\Lambda_\sigma^{[1]}}{F_\sigma^{[1]}} P_{\mu_\sigma}(m) & \text{if } E_\sigma(m) = 0 \wedge C^{[1]}(\sigma) > 0 \\ P_\mu(m) & \text{otherwise} \end{cases} \quad (22)$$

where  $\lambda_\sigma^{[1]}$  are strictly positive discounting functions that generate, for each  $\sigma$ , a global discounting factor

$$\Lambda_\sigma^{[1]} = \frac{1}{C^{[1]}(\sigma)} \sum_m E_\sigma(m) \lambda_\sigma^{[1]}(m) \quad (23)$$

which is normalized with

$$F_\sigma^{[1]} = 1 - \sum_{n: E_\sigma(n) > 0} P_{\mu_\sigma}(n) \quad (24)$$

Here,  $P_\mu$  and  $P_{\mu_\sigma}$  are non-vanishing probability distributions (for instance, Poisson-like) over the positive integers such that their expectation value coincides with the average number  $\mu$  of descendents of the nodes in the sample or the average number  $\mu_\sigma$  of descendents of nodes labeled  $\sigma$ , that is

$$\mu_\sigma = \frac{\sum_m m E_\sigma(m)}{\sum_m E_\sigma(m)} \quad (25)$$

Note that  $\Lambda_\sigma^{[1]} > 0$  when  $C^{[1]}(\sigma) > 0$ , as this implies  $E_\sigma(m) > 0$  for some  $m$ .

The ground model  $M^{[1]}$  has as a second ingredient: the probabilities  $p^{[1]}(\sigma)$  that a node has label  $\sigma$ :

$$p^{[1]}(\sigma) = \begin{cases} \frac{C^{[1]}(\sigma)}{N} (1 - \lambda^{[1]}(\sigma)) & \text{if } C^{[1]}(\sigma) > 0 \\ \frac{\Lambda^{[1]}}{F^{[1]}} & \text{otherwise} \end{cases} \quad (26)$$

where  $\lambda^{[1]}(\sigma)$  is a strictly positive function,  $N = \sum_a C^{[1]}(a)$  is the number of

nodes in the sample, the normalizing factor  $F^{[1]}$  is the number of different alphabet symbols not found in  $S$  and

$$\Lambda^{[1]} = \frac{1}{N} \sum_a C^{[1]}(a) \lambda^{[1]}(a) \quad (27)$$

Note that any non-empty sample (that is, with  $N > 0$ ) has  $C^{[1]}(\sigma) > 0$  for some  $\sigma$  and, therefore,  $\Lambda^{[1]} > 0$ .

With these ingredients, we are ready to define backing-off schemes for the elementary probabilities with  $k = 2$ : given  $t = \sigma(a_1 \dots a_m)$ , they are computed as follows

$$p_m^{[2]}(\sigma, a_1, \dots, a_m) = \begin{cases} \frac{C^{[2]}(t)}{C^{[1]}(\sigma)} (1 - \lambda^{[2]}(t)) & \text{if } C^{[2]}(t) > 0 \\ \frac{\Lambda^{[2]}(\sigma)}{F^{[2]}(\sigma)} p_\sigma^{[1]}(m) \prod_{i=1}^m p^{[1]}(a_i) & \text{otherwise} \end{cases} \quad (28)$$

where

$$\Lambda^{[2]}(\sigma) = \sum_{u: C^{[2]}(u) > 0 \wedge r_1(u) = \sigma} \frac{C^{[2]}(u)}{C^{[1]}(\sigma)} \lambda^{[2]}(u) \quad (29)$$

and the normalization factor is

$$F^{[2]}(\sigma) = 1 - p_0^{[2]}(\sigma) - \sum_{m > 0: E_\sigma(m) > 0} p_\sigma^{[1]}(m) \sum_{\substack{a_1, \dots, a_m: \\ C^{[2]}(\sigma(a_1 \dots a_m)) > 0}} \prod_{i=1}^m p^{[1]}(a_i) \quad (30)$$

Again, it is easy to realize that the value of any probability of this type is strictly positive.

In case  $k > 2$ ,  $p_0^{[k]}(\sigma) = 1$ . However, the probabilities of type  $p_m^{[k]}$  with  $m > 0$ , given  $t = \sigma(t_1 \dots t_m)$  with  $t_i = a_i(s_{i1} \dots s_{im_i})$ , are defined as

$$p_m^{[k]}(\sigma, t_1, \dots, t_m) = \begin{cases} \frac{C^{[k]}(t)}{C^{[k-1]}(r_{k-1}(t))} (1 - \lambda^{[k]}(t)) & \text{if } C^{[k]}(t) > 0 \\ \frac{\Lambda^{[k]}(r_{k-1}(t))}{F^{[k]}(r_{k-1}(t))} \prod_{i=1}^m p_{m_i}^{[k-1]}(a_i, s_{i1}, \dots, s_{im_i}) & \text{otherwise} \end{cases} \quad (31)$$

where

$$\Lambda^{[k]}(q) = \frac{1}{C^{[k-1]}(t)} \sum_{u: C^{[k]}(u) > 0 \wedge r_{k-1}(u) = q} C^{[k]}(u) \lambda^{[k]}(u) \quad (32)$$

and the normalization factor is

$$F^{[k]}(\sigma(\tau_1 \dots \tau_m)) = 1 - \sum_{u_1, \dots, u_m: C^{[k]}(\sigma(u_1 \dots u_m)) > 0 \wedge r_{k-2}(u_i) = \tau_i} \prod_{i=1}^m p_{m_i}^{[k-1]}(a_i, v_{i1}, \dots, v_{im_i}) \quad (33)$$

where  $u_i = a_i(v_{i1}, \dots, v_{im_i})$ .

Finally,  $\rho^{[k]}$  probabilities can be defined in case  $k = 2$  as

$$\rho^{[2]}(\sigma) = \begin{cases} \frac{D^{[2]}(\sigma)}{|S|} (1 - \theta^{[2]}(\sigma)) & \text{if } D^{[2]}(\sigma) > 0 \\ \frac{\Theta^{[2]}}{G^{[2]}} p_{\sigma}^{[1]}(0) & \text{otherwise} \end{cases} \quad (34)$$

where the discounting factor  $\Theta^{[2]}$  is

$$\Theta^{[2]} = \frac{1}{|S|} \sum_{a: D^{[2]}(a) > 0} D^{[2]}(a) \theta^{[2]}(a) \quad (35)$$

and its corresponding normalization

$$G^{[2]} = 1 - \sum_{a: D^{[2]}(a) > 0} p^{[1]}(a) \quad (36)$$

In case  $k > 2$  we use instead for  $t = \sigma(t_1 \dots t_m)$

$$\rho^{[k]}(\sigma(t_1 \dots t_m)) = \begin{cases} \frac{D^{[k]}(t)}{|S|} (1 - \theta^{[k]}(t)) & \text{if } D^{[k]}(\sigma(t_1 \dots t_m)) > 0 \\ \frac{\Theta^{[k]}}{G^{[k]}} \rho^{[k-1]}(r_{k-2}(t)) p_m^{[k-1]}(\sigma, t_1, \dots, t_m) & \text{otherwise} \end{cases} \quad (37)$$

where

$$\Theta^{[k]} = \frac{1}{|S|} \sum_{u: D^{[k]}(u) > 0} D^{[k]}(u) \theta^{[k]}(u) \quad (38)$$

and the suitable normalizing factor is

$$G^{[k]} = 1 - \sum_a \sum_m \sum_{\substack{u_1, \dots, u_m: \\ D^{[k]}(a(u_1 \dots u_m)) > 0}} \rho^{[k-1]}(r_{k-2}(a(u_1 \dots u_m))) p_m^{[k-1]}(a, u_1, \dots, u_m) \quad (39)$$

All these equations provide an efficient backing-off scheme that guarantees that no event is assigned a null probability.

As an illustration, consider the tiny sample  $S = \{a(a(ab)b)\}$ . The probabilities of the events seen in the sample are summarized in the following table

$\rho^{[3]}(a(ab)) = (1 - \theta_{a(ab)}^{[3]})$	$\rho^{[2]}(a) = (1 - \theta_a^{[2]})$
$p_2^{[3]}(a, a(a), b) = \frac{1}{2}(1 - \lambda_{a(ab)}^{[3]})$	$p_2^{[2]}(a, a, b) = \frac{2}{4}(1 - \lambda_a^{[2]})$
$p_2^{[3]}(a, a, b) = \frac{1}{2}(1 - \lambda_{a(ab)}^{[3]})$	$p_1^{[2]}(a, a) = \frac{1}{4}(1 - \lambda_a^{[2]})$
$p_1^{[3]}(a, a(ab)) = (1 - \lambda_{a(a)}^{[3]})$	$p_0^{[2]}(a) = \frac{1}{4}(1 - \lambda_a^{[2]})$
$p_0^{[3]}(a) = p_0^{[3]}(b) = 1$	$p_0^{[2]}(b) = \frac{2}{2}(1 - \lambda_b^{[2]})$

In order to obtain, for instance, the probability of tree  $a(a(ab)b)$  with  $k = 3$

one needs

$$p(|a(a(ab)b)|M^{[3]}) = \rho^{[3]}(a(ab))p_2^{[3]}(a, a(ab), b)p_2^{[3]}(a, a, b)p_0^{[3]}(a)p_0^{[3]}(b)^2 \quad (40)$$

where, only for the second factor, backing-off needs to be applied

$$p_2^{[3]}(a, a(ab), b) = \frac{\lambda_{a(ab)}^{[3]}}{1 - p_1^{[2]}(a, a)p_0^{[2]}(b) - p_0^{[2]}(a)p_0^{[2]}(b)}p_2^{[2]}(a, a, b)p_0^{[2]}(b) \quad (41)$$

Substituting the above expression in (40), the result can be now computed using only the probabilities in the table.

## 6 Conclusion

We have described how a probabilistic extension of the  $k$ -testable tree languages allows for simple statistical learning procedures and can be used to implement arithmetic compression algorithms or backing-off techniques. The models of this type can be updated incrementally and may be inferred using medium-sized samples with better results than some state merging methods [25] which tend to output too simple models. The higher compression rates achieved when processing tree data files show that this class of models provide a suitable description of the structured data.

## References

- [1] Peter F. Brown, Vincent J. Della Pietra, Peter V. deSouza, Jenifer C. Lai, and Robert L. Mercer. Class-based  $n$ -gram models of natural language. *Computational Linguistics*, 18(4):467–479, 1992.
- [2] 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*, 17(12):1202–1212, 1995.
- [3] Frederick Jelinek. *Statistical Methods for Speech Recognition*. The MIT Press, Cambridge, Massachusetts, 1998.
- [4] Frank Rubin. Experiments in text file compression. *Communications of the ACM*, 19(11):617–623, 1976.
- [5] I. H. Witten, R. M. Neal, and J. G. Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, 1987.
- [6] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley Series in Telecommunications. John Wiley & Sons, New York, NY, USA, 1991.
- [7] K. L. Chung. *Markov Chains with Stationary Transition Probabilities*. Springer, Berlin, 2 edition, 1967.

- [8] Yechezkel Zalcstein. Locally testable languages. *Journal of Computer and System Sciences*, 6(2):151–167, April 1972.
- [9] Pedro García and Enrique Vidal. Inference of  $k$ -testable languages in the strict sense and application to syntactic pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(9):920–925, sep 1990.
- [10] Takashi Yokomori. On polynomial-time learnability in the limit of strictly deterministic automata. *Machine Learning*, 19(2):153–179, 1995.
- [11] Eugene Charniak. *Statistical Language Learning*. MIT Press, 1993.
- [12] K. Sima'an, R. Bod, S. Krauwer, and R. Scha. Efficient disambiguation by means of stochastic tree substitution grammars. In Daniel B. Jones and Harold L. Somers, editors, *Proc. of the Int. Conf. on New Methods in Language Processing. Manchester, UK, 14–16 Sep 1994*, pages 50–58, London, UK, 1996. UCL Press.
- [13] Andreas Stolcke. An efficient context-free parsing algorithm that computes prefix probabilities. *Computational Linguistics*, 21(2):165–201, 1995.
- [14] Mikkel Thorup. Disambiguating grammars by exclusion of sub-parse trees. *Acta Informatica*, 33(6):511–522, 1996.
- [15] Paul Prescod. Formalizing XML and SGML instances with forest automata theory. Technical report, University of Waterloo, Department of Computer Science, Waterloo, Ontario, 2000. Draft Technical Paper.
- [16] M. Murata. Transformation of documents and schemas by patterns and contextual conditions. In Charles K. Nicholas and Derick Wood, editors, *Principles of Document Processing, Third International Workshop, PODP'96 Proceedings*, volume 1293, pages 153–169, 1997.
- [17] Boris Chidlovskii. Using regular tree automata as XML schemas. In J. Hoppenbrouwers, T. de Souza Lima, M. Papazoglou, and A. Sheth, editors, *Proc. IEEE Advances on Digital Libraries Conference 2000*, pages 89–98, 2000.
- [18] J. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Language, and Computation*. Addison–Wesley, Reading, MA, 1979.
- [19] Y. Sakakibara, M. Brown, R. C. Underwood, I. S. Mian, and D. Hausler. Stochastic context-free grammars for modeling RNA. In Lawrence Hunter, editor, *Proceedings of the 27th Annual Hawaii International Conference on System Sciences. Volume 5 : Biotechnology Computing*, pages 284–294, Los Alamitos, CA, USA, January 1994. IEEE Computer Society Press.
- [20] Ferenc Gécseg and Magnus Steinby. *Tree Automata*. Akadémiai Kiadó, Budapest, 1984.
- [21] Yasubumi Sakakibara. Efficient learning of context-free grammars from positive structural examples. *Information and Computation*, 97(1):23–60, March 1992.
- [22] Hubert Common, Max Dauchet, Rémy Gilleron, Florent Jacquemard, Denis Lugiez, Sophie Tison, and Marc Tommasi. Tree automata

- techniques and applications. Draft book; available electronically on <http://www.grappa.univ-lille3.fr/tata>, 2002.
- [23] Timo Knuutila. Inference of  $k$ -testable tree languages. In H. Bunke, editor, *Advances in Structural and Syntactic Pattern Recognition (Proc. Intl. Workshop on Structural and Syntactic Pattern Recognition, Bern, Switzerland)*. World Scientific, 1993.
  - [24] F. Pereira and Y. Schabes. Inside-outside re-estimation from partially bracketed corpora. In *Proceedings of the 30th annual meeting of the ACL*, pages 128–135, Newark, 1992.
  - [25] Rafael C. Carrasco, Jose Oncina, and Jorge Calera-Rubio. Stochastic inference of regular tree languages. *Machine Learning*, 44(1/2):185–197, 2001.
  - [26] Jose Luis Verdú-Mas, Mikel L. Forcada, Rafael C. Carrasco, and Jorge Calera-Rubio. Tree  $k$ -grammar models for natural language modelling and parsing. In Terry Caelli, Adnan Amin, Robert P. W. Duin, Mohamed S. Kamel, and Dick de Ridder, editors, *Structural, Syntactic, and Statistical Pattern Recognition, Joint IAPR International Workshops SSPR 2002 and SPR 2002, Windsor, Ontario, Canada, Proceedings*, volume 2396 of *Lecture Notes in Computer Science*, pages 53–63. Springer, 2002.
  - [27] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of english: the penn treebank. *Computational Linguistics*, 19:313–330, 1993.
  - [28] Juan Ramón Rico-Juan, Jorge Calera-Rubio, and Rafael C. Carrasco. Stochastic  $k$ -testable tree languages and applications. In Menno van Zaanen Pieter W. Adriaans, Henning Fernau, editor, *Grammatical Inference: Algorithms and Applications, 6th International Colloquium: ICGI 2002*, volume 2484 of *Lecture Notes in Computer Science*, pages 199–212, 2002.
  - [29] Maurice Nivat and Andreas Podelski. Minimal ascending and descending tree automata. *SIAM Journal on Computing*, 26(1):39–58, 1997.
  - [30] R. Chaudhuri, S. Pham, and O.N. Garcia. Solution of an open problem on probabilistic grammars. *IEEE Transactions on Computers*, 32(8):758–750, 1983.
  - [31] J.A. Sánchez and J.M. Benedí. Consistency of stochastic context-free grammars from probabilistic estimation based on growth transformations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(9):1052–1055, 1997.
  - [32] E.M. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.
  - [33] I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufman Publishing, San Francisco, 2nd edition, 1999.
  - [34] John G. Cleary and Ian H. Witten. Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communicaton*, 32(4):396–402, 1984.

## A Decoding functions

The corresponding decoding functions can be found in fig. A.2 and A.1. These functions call procedures `get` that play the inverse role of function `send` in compression.

```
function r_decode ( $k$ )
   $t \leftarrow \text{get}(\beta, k)$ 
  if ( $t! = \epsilon_r$ ) then
    do( $i = 2, \dots, k$ )
      inc( $D^{[i]}(r_{i-1}(t))$ )
      update( $i - 1, r_{i-1}(t)$ )
    enddo
  else
    inc( $D^{[k]}(\epsilon_r)$ )
    if  $k > 2$  then
       $q \leftarrow \text{r\_decode}(k - 1)$ 
       $t \leftarrow \text{f\_decode}(q)$ 
    else
       $\sigma = \text{get}(\alpha, 1)$ 
      inc( $C^{[1]}(\sigma)$ )
       $t \leftarrow \sigma$ 
    endif
    inc( $D^{[k]}(t)$ )
  endif
  return  $t$ 
endfunction
```

Fig. A.1. Root decoding function.

```

function f_decode (k)
  k ← 2 + depth(q)
  t ← get(α, k, q)
  if (t! = εq) then
    update(t)
  else
    inc(C[k](εq))
    if (k > 2) then           [q = σ(τ1 ... τm)]
      do (j = 1, ..., m)
        if (1 + depth(τj) = k - 1) then
          tj = f_decode(τj)
        endif
      enddo
      t ← σ(t1 ... tm)
    else                       [q = σ]
      do (while get(γ, σ) = εσ)
        inc(Eσ(εσ))
      enddo
      m ← get(γ, σ)
      inc(Eσ(m))
      do (j = 1, ..., m)
        aj ← get(α, 1)
        inc (C[1](aj))
      enddo
      t ← σ(a1 ... am)
    endif
    inc(C[k](t))
  endif
return t
endfunction

```

Fig. A.2. Decoding function.