Parsing of Partially Bracketed Structures for Parse Selection

Mark-Jan Nederhof
School of Computer Science
University of St Andrews
St Andrews, United Kingdom

Ricardo Sánchez-Sáez
Instituto Tecnológico de Informática
Universidad Politècnica de Valencia
Valencia, Spain

Abstract

We consider the problem of parsing a sentence that is partially annotated with information about where phrases start and end. The application domain is interactive parse selection with probabilistic grammars. It is explained that the main obstacle is spurious ambiguity. The proposed solution is first described in terms of appropriately constrained synchronous grammars, and then in terms of a computational model for parsing. Experiments show the feasibility for a practical grammar.

1 Introduction

In interactive parse selection, the objective is to obtain correct parses of sentences in a corpus, by means of an iterative process, alternately drawing upon a language model and human linguistic judgement. In a first step, the most likely parse is computed on the basis of the model. This parse is displayed to the human annotator, who looks for possible errors and enters corrections. Each correction takes the form of an occurrence of a phrase that the parse should contain. The model is then consulted anew, to recompute the most likely parse, but now under the constraint that all occurrences of phrases entered previously by the linguist must be included. This process is repeated until no more errors remain. Applications can be found in creation of treebanks (Marcus et al., 1993) and computer-assisted translation (Barrachina et al., 2009).

Apart from the exact language model used in the process, there are various ways to implement interactive parse selection. One obvious approach is to demand that errors are corrected strictly from left to right. That is, where the occurrence of a phrase is asserted by the annotator, it is implicitly assumed that all phrases in the latest proposed parse that are wholly contained in the preceding prefix are correct. This means that these structures in a left-hand portion of the parse tree can no longer change in future iterations.

Another degree of freedom in the design of interactive parse selection is the exact information that the human annotator provides about occurrences of phrases. The most obvious choice would be a triple consisting of the beginning, the end, and the syntactic category ('noun phrase', 'prepositional phrase', etc.). If desired, the category could be omitted or underspecified. This approach has been implemented for example by Sánchez-Sáez et al. (2009; 2010).

The main motivation for interactive parse selection is that it saves the human annotator manual labour, by automatic prediction of at least parts of parses that very often are correct. With the criterion of minimizing human effort, it not clear however that the optimal design of interactive parse selection is of the kind outlined above, with a strictly left-to-right strategy, and with specification of both the beginning and the end for each corrected phrase. One objection against the left-to-right strategy is that errors may be temporarily overlooked. Typical implementations may allow backtracking to deal with this situation, but backtracking entails that work needs to be redone.

One objection against having to specify the beginning as well as the end of a corrected phrase is firstly that this requires more mouse clicks or keyboard strokes than if, say, only the correct beginning of a phrase were specified. Furthermore, for long and complex sentences, it may be tedious to determine both phrase boundaries.

For these reasons we explore a less rigid alter-

Proceedings of the 12th International Conference on Parsing Technologies, pages 231–240,
October 5-7, 2011, Dublin City University.

native, namely to allow the human annotator to specify only the beginning of a phrase, or only the end of a phrase. This is formalized in the remainder of this paper as an unmatched open bracket, or an unmatched close bracket. Such a bracket may be labelled with a category or not. Parse selection is not constrained to be unidirectional, and at each iteration, brackets can be placed at arbitrary positions in the input sentence, and thereupon the most likely parse is (re-)computed that is consistent with the provided brackets so far.

In our notation, the unmatched brackets are written as square brackets. We refer to them as 'unmatched' because the user need not specify both the beginning and end of a phrase. However, users *may* specify both the beginning of a phrase and the end of the same phrase, by a square open and a square close bracket, if they so choose.

Next to unmatched brackets, we also allow matched brackets in the input, which in our notation are written as round brackets. These must always occur in pairs of one open bracket and one close bracket, specified together by the user. Unlike square brackets, pairs of round brackets must be properly nested. As square brackets, round brackets may be labelled by categories or may be unlabelled. Sentences enriched with matched and unmatched brackets will be called *partially bracketed* strings.

As we will informally illustrate in the following section, parsing of partially bracketed strings by context-free grammars causes particular problems. The main issue is spurious ambiguity, by which one input string may be parsed in different ways all corresponding to one and the same parse tree by the input grammar. Where the language model is used to compute the most likely parse, performance may suffer from having one computation computed in more than one way. A more serious consequence is that computation of inside probabilities is hindered by subparses being represented more than once. Also n-best parsing algorithms no longer work correctly without further refinements.

The main contribution of this article is to offer a solution to avoiding all spurious ambiguity. Our theoretical framework is that of order-preserving synchronous context-free grammars, to be summarized in Section 3. With this machinery, mappings between unbracketed, bracketed and partially bracketed strings will be presented in Section 4. A sketch of a proof that spurious ambiguity is avoided as claimed is the subject of Section 5. The actual parsing process, which is based on Earley's algorithm, is presented in Section 6.

Section 7 discusses an implementation. The practicality of our approach is demonstrated by experiments measuring running time. In addition, some possible optimizations are proposed. We end our paper with conclusions, in Section 8.

The issue of avoiding spurious ambiguity was considered before by (Wieling et al., 2005). Our treatment differs in that the solution is at the same time more precise, in terms of synchronous CFGs rather than grammar transformations, and more succinct, using simpler constraints on allowable structures. Also novel is our parsing algorithm, which is versed towards practical application. Earlier work on partially bracketed strings, such as that by Pereira and Schabes (1992), has involved matching brackets only.

2 Informal Illustration

Let us consider the following example context-free grammar:

$$\begin{array}{lcl} NP & \rightarrow & Adj\ NP \mid N \\ Adj & \rightarrow & \mathbf{big} \mid \mathbf{angry} \\ N & \rightarrow & \mathbf{dog} \end{array}$$

(The vertical bar separates alternative right-hand sides for the same left-hand side nonterminal symbol.)

The language of all fully bracketed strings is generated by the following grammar:

$$\begin{array}{lcl} NP & \rightarrow & (_{NP}\ Adj\ NP\ )_{NP} \mid (_{NP}\ N\ )_{NP} \\ Adj & \rightarrow & (_{Adj}\ \mathbf{big}\ )_{Adj} \mid (_{Adj}\ \mathbf{angry}\ )_{Adj} \\ N & \rightarrow & (_{N}\ \mathbf{dog}\ )_{N} \end{array}$$

We can make such bracketed strings less precise by:

1. omitting labels of categories at brackets, and/or
2. replacing a matching pair of round brackets by a pair of square brackets, of which zero, one or both may then be omitted.

For example, we can 'fuzzify' a fully bracketed string:

$$\begin{array}{l} (_{NP}\ (_{Adj}\ \mathbf{big}\ )_{Adj}\ (_{NP}\ (_{Adj}\ \mathbf{angry}\ )_{Adj} \\ \quad (_{NP}\ (_{N}\ \mathbf{dog}\ )_{N}\ )_{NP}\ )_{NP}\ )_{NP} \end{array} \quad (1)$$

by a partially bracketed string:

$$\textbf{big angry (dog) }]_{NP} \quad (2)$$

Note there is little point in omitting the label of one round bracket from a pair of matching brackets without also omitting it from the other bracket in the pair. This is because the one omitted label could be reconstructed from the remaining label by casual inspection of the string.

The language of all partially bracketed strings can thus be naively specified by a context-free grammar where next to a rule of the form:

$$A \rightarrow (_A\ \alpha\ )_A$$

we also have a rule of the form:

$$A \rightarrow (\ \alpha\ )$$

and nine rules of the form:

$$A \rightarrow y^{(l)}\ \alpha\ y^{(r)}$$

where $y^{(l)}$ is one of $[_A$, $[$, or the empty string ε, and $y^{(r)}$ is one of $]_A$, $]$, or ε.

However, with the resulting grammar, the partially bracketed string in (2) can be parsed in five different ways, four of which are illustrated in Figure 1. The trees in (a) and (b) differ in the placement of $]_{NP}$ to the right of a right-most path in the parse tree with more than one node labelled *NP*; in fact, there are three different nodes to which $]_{NP}$ can be attached. The trees in (c) and (d) differ from those in (a) and (b) in the placement of the pair of unlabelled round brackets on either side of a path in the parse tree involving only unit rules (that is, rules with right-hand sides of length one).

Because the original grammar was unambiguous, the existence of five different parse trees for the partially bracketed string should be seen as spurious ambiguity, given the task of finding parses of "**big angry dog**" that are consistent with the collection of brackets inserted into that string. As we will explain in more detail later in this article, this problem of spurious ambiguity is avoided by demanding that brackets occur as high as possible. In the example, this means that the round brackets are placed around "*N*" as in (c) or (d) rather than around "**dog**" as in (a) or (b). Further, $]_{NP}$ is attached to the highest possible node labelled *NP* as in (d) rather than (c). Thus, our parser would return only the tree in (d). It is not difficult to see there is always a unique way of placing brackets as high as possible.

3 Preliminaries

In order to formalize the main problem and its solution, we turn to a restricted type of synchronous context-free grammar (SCFG), in notation similar to that in Satta and Peserico (2005). A SCFG $\mathcal{G}$ defines a relation between a source language generated by a context-free grammar $\mathcal{G}_1$ and a target language generated by a context-free grammar $\mathcal{G}_2$. In the general case, each 'synchronous' rule in $\mathcal{G}$ has the form $\langle A \rightarrow \alpha, B \rightarrow \beta \rangle$, where $A \rightarrow \alpha$ is a rule in $\mathcal{G}_1$ and $B \rightarrow \beta$ is a rule in $\mathcal{G}_2$. The number of nonterminal symbols in α must equal that in β.

Each such synchronous rule $\langle A \rightarrow \alpha, B \rightarrow \beta \rangle$ is also associated with a bijection from the nonterminal occurrences in α to the nonterminal occurrences in β. By this bijection, one may express a reordering of constituents between source and target structures. In this paper, we will consider a restricted type of SCFG without such reordering, or put differently, the bijection implicitly maps the i-th nonterminal occurrence in α to the i-th nonterminal occurrence in β. We will call this restriction OP-SCFG (*order-preserving* SCFG).

Let $\mathcal{G}$ be an OP-SCFG as above, with S_1 the start symbol of $\mathcal{G}_1$ and S_2 the start symbol of $\mathcal{G}_2$. We first define relations $\mathcal{T}_{\mathcal{G}}^{(A,B)}$ between pairs of languages, for A a nonterminal in $\mathcal{G}_1$ and B a nonterminal in $\mathcal{G}_2$, to be the smallest relations such that:

1. existence of a synchronous rule

$$\langle A \rightarrow w_0 A_1 w_1 \cdots w_{n-1} A_m w_m,\ B \rightarrow v_0 B_1 v_1 \cdots v_{m-1} B_m v_m \rangle \quad (3)$$

2. existence for $1 \leq i \leq m$ of strings x_i and y_i such that $x_i\ \mathcal{T}_{\mathcal{G}}^{(A_i,B_i)}\ y_i$

together imply that:

$$w_0 x_1 w_1 \cdots x_m w_m\ \mathcal{T}_{\mathcal{G}}^{(A,B)}\ v_0 y_1 v_1 \cdots y_m v_m$$

The transduction $\mathcal{T}_{\mathcal{G}}$ generated by $\mathcal{G}$ is now defined to be $\mathcal{T}_{\mathcal{G}}^{(S_1,S_2)}$.

For each 4-tuple (x, A, B, y) such that $x\ \mathcal{T}_{\mathcal{G}}^{(A,B)}\ y$ we can build at least one derivation tree that shows in reverse how $x\ \mathcal{T}_{\mathcal{G}}^{(A,B)}\ y$ was obtained. More precisely, such derivation trees can be inductively defined as follows. Let there be a synchronous rule as in (3), and let each t_i, with $1 \leq i \leq m$, be a derivation tree associated with a 4-tuple (x_i, A_i, B_i, y_i) such that $x_i\ \mathcal{T}_{\mathcal{G}}^{(A_i,B_i)}\ y_i$.

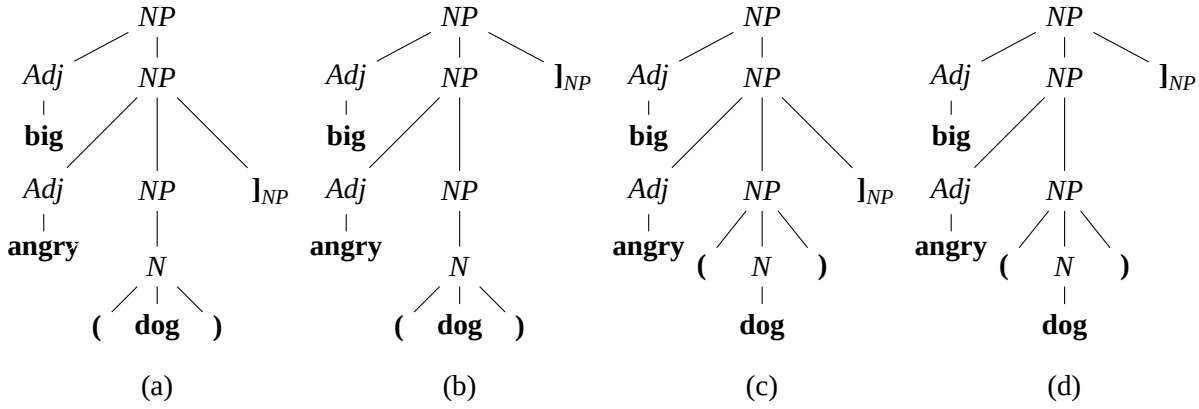

Figure 1: Four parses of the same partially bracketed string.

Then a derivation tree can be constructed for the 4-tuple (x, A, B, y), where $x = w_0 x_1 w_1 \cdots x_m w_m$ and $y = v_0 y_1 v_1 \cdots y_m v_m$, by a root node labelled by the above synchronous rule, and its (ordered) children are the roots of t_i.

Similar to the notion of ambiguity in CFGs, we say that a pair (x, y) is ambiguous for a fixed SCFG if more than one derivation tree can be associated with the 4-tuple (x, S_1, S_2, y). We say the SCFG is ambiguous if there is at least one ambiguous pair (x, y). For an example of these concepts, see the following section.

In this paper, we will assume there are no epsilon rules, i.e., rules with empty right-hand sides.

4 Bracketed and Partially Bracketed Strings

The parsing problem for a CFG $\mathcal{G}$ can be described in terms of a transduction from unbracketed strings to fully bracketed strings, by means of the OP-SCFG $\mathcal{G}_{bracket}$ that has one synchronous rule:

$$\langle A \to \alpha, A \to \mathbf{(}_A\ \alpha\ \mathbf{)}_A \rangle$$

for each rule $A \to \alpha$ in $\mathcal{G}$. The strings y such that $x\ \mathcal{T}_{\mathcal{G}_{bracket}}\ y$ each describe one parse of x according to the input CFG $\mathcal{G}$.

A naive transduction to fuzzify fully bracketed strings can be defined in terms of a SCFG $\mathcal{G}_{naive}$ of which the synchronous rules are, for each rule $A \to \alpha$ in the input grammar:

- $\langle A \to \mathbf{(}_A\ \alpha\ \mathbf{)}_A, A \to \mathbf{(}_A\ \alpha\ \mathbf{)}_A \rangle$,
- $\langle A \to \mathbf{(}_A\ \alpha\ \mathbf{)}_A, A \to \mathbf{(}\ \alpha\ \mathbf{)} \rangle$,
- $\langle A \to \mathbf{(}_A\ \alpha\ \mathbf{)}_A, A \to y^{(l)}\ \alpha\ y^{(r)} \rangle$, for all nine combinations of $y^{(l)} \in \{\mathbf{[}_A, \mathbf{[}, \varepsilon\}$ and $y^{(r)} \in \{\mathbf{]}_A, \mathbf{]}, \varepsilon\}$.

The problem with spurious ambiguity that was the subject of discussion in Section 2 can now be expressed in formal terms, as ambiguity of SCFG $\mathcal{G}_{naive}$. Concretely, one and the same fully bracketed string can be mapped to one and the same partially bracketed string in different ways. This is particularly relevant if the transduction is used in reverse, mapping a given partially bracketed string to fully bracketed strings, or in other words, building parse trees according to the input grammar. As explained in the introduction, the problems this causes include increased running time, and failure of probabilistic models and n-best algorithms.

To illustrate this, let us revisit the example from Figure 1, which corresponds to the following input/output pair:

$$\begin{aligned} &\big(\ \mathbf{(}_{NP}\ \mathbf{(}_{Adj}\ \mathbf{big}\ \mathbf{)}_{Adj}\ \mathbf{(}_{NP}\ \mathbf{(}_{Adj}\ \mathbf{angry}\ \mathbf{)}_{Adj} \\ &\qquad\qquad \mathbf{(}_{NP}\ \mathbf{(}_{N}\ \mathbf{dog}\ \mathbf{)}_{N}\ \mathbf{)}_{NP}\ \mathbf{)}_{NP}\mathbf{)}_{NP}\ , \\ &\quad \mathbf{big\ angry\ (\ dog\ )\ ]}_{NP}\ \big) \end{aligned} \tag{4}$$

With $\mathcal{G}_{naive}$, there are five different derivation trees through which this pair can be obtained. For example, the tree in Figure 1 (d), which we regard as the preferred one, corresponds to application of the following rules:

$\langle\ NP \to \mathbf{(}_{NP}\ Adj\ NP\ \mathbf{)}_{NP}\ ,\ NP \to Adj\ NP\ \mathbf{]}_{NP}\ \rangle$,
$\langle\ Adj \to \mathbf{(}_{Adj}\ \mathbf{big}\ \mathbf{)}_{Adj}\ ,\ Adj \to \mathbf{big}\ \rangle$,
$\langle\ NP \to \mathbf{(}_{NP}\ Adj\ NP\ \mathbf{)}_{NP}\ ,\ NP \to Adj\ NP\ \rangle$,
$\langle\ Adj \to \mathbf{(}_{Adj}\ \mathbf{angry}\ \mathbf{)}_{Adj}\ ,\ Adj \to \mathbf{angry}\ \rangle$,
$\langle\ NP \to \mathbf{(}_{NP}\ N\ \mathbf{)}_{NP}\ ,\ NP \to \mathbf{(}\ N\ \mathbf{)}\ \rangle$,
$\langle\ N \to \mathbf{(}_{N}\ \mathbf{dog}\ \mathbf{)}_{N}\ ,\ N \to \mathbf{dog}\ \rangle$

As solution we propose a refined SCFG $\mathcal{G}_{fuzzy}$. It specifies the same transduction as $\mathcal{G}_{naive}$, but

now without ambiguity. Intuitively, we ensure that any brackets in a partially bracketed string are attached as high as possible. Because there can be at most one way of doing this that corresponds to a fully bracketed string, the implication is that there can be at most one derivation tree for each combination of a source string x and a target string y.

The SCFG $\mathcal{G}_{fuzzy}$ conceptually keeps records of where brackets are attached by replacing nonterminals A in all possible ways by nonterminals $A(s^{(l)}, s^{(r)})$, where the values $s^{(l)}$ and $s^{(r)}$ are either ϵ or a bracket. If such a value is a bracket, that means that this bracket was attached to the nearest descendant to which it could be attached, in the left-most (for $s^{(l)}$) or right-most (for $s^{(r)}$) path downwards in the parse tree. The definition of $\mathcal{G}_{fuzzy}$ disallow situations where this bracket could be moved to the current node, because that would imply that the bracket is not attached as high as possible. Note that there is a finite number of choices for $s^{(l)}$ and $s^{(r)}$, so that the rules are still within context-free power.

The synchronous rules of $\mathcal{G}_{fuzzy}$ are specified in Table 1. As expressed by (5) and (6), the round brackets from the source language can be replaced with unlabelled or square brackets. If in the target language there is an open round bracket, either labelled or unlabelled, then the close bracket must be of the same form, and vice versa.

As is clear from (7), only nonterminals are extended with two arguments. Terminals in Σ are copied unchanged from source language to target language. The first line of (8) says that if the current rule is a unit rule, and if a pair of round brackets were attached to a node further down, possibly along more occurrences of unit rules, then there must be a reason why these brackets cannot be attached to the current node, and the only reason can be that other brackets $y^{(l)} \neq \varepsilon$ or $y^{(r)} \neq \varepsilon$ are already attached to the current node. The second and third lines similarly exclude situations where a square bracket was placed further down, while it could be attached to the current node.

The information about brackets attached to the current node, or brackets from further down if no brackets are attached to the current node, is passed on bottom-up, as expressed by (9). For technical reasons, we need to augment the grammar with a new start symbol, as shown in (10).

5 Correctness

Two properties of $\mathcal{G}_{fuzzy}$ are of interest. The first is that $\mathcal{T}_{\mathcal{G}_{fuzzy}}$ equals $\mathcal{T}_{\mathcal{G}_{naive}}$, and the second is that $\mathcal{G}_{fuzzy}$ is unambiguous. The first proof is tedious, but the intuition is simple: if we are given a derivation tree associated with 4-tuple (x, S, S, y) where $x\ \mathcal{T}_{\mathcal{G}_{naive}}\ y$, for a fully bracketed string x and partially bracketed string y, then we can systematically change this derivation tree, to preserve x and y but move brackets to be attached as high as possible in the parse tree of y. With this attachment of brackets, we can straightforwardly map this derivation tree into a derivation tree associated with $(x, S^\dagger, S^\dagger, y)$ where $x\ \mathcal{T}_{\mathcal{G}_{fuzzy}}\ y$, ensuring that the constraint in (8) is satisfied. Conversely, if $x\ \mathcal{T}_{\mathcal{G}_{fuzzy}}\ y$ then clearly $x\ \mathcal{T}_{\mathcal{G}_{naive}}\ y$.

The unambiguity of $\mathcal{G}_{fuzzy}$ can be argued as follows. First, if $\mathcal{G}_{fuzzy}$ were ambiguous, then there would be a pair (A, B) of nonterminals and a pair (x, y) of strings, with the length of xy minimal and AB minimal according to some fixed lexicographical ordering, such that two or more different derivation trees can be associated with 4-tuple (x, A, B, y) where $x\ \mathcal{T}^{(A,B)}_{\mathcal{G}_{fuzzy}}\ y$. Because of the minimality of xy and AB, the two derivation trees must differ in the synchronous rule at the root, each of which must be of the form in (5). The remainder of the proof consists of a large number of case distinctions, and in each the task is to show a contradiction, by violation of (8). For example, suppose y starts with **[** and the two synchronous rules differ in that one has $y^{(l)} =$ **[** and the second has $y^{(l)} = \varepsilon$. Then in the second case, the **[** must be generated by a rule further down in the derivation tree, which would violate (8).

6 Parsing

In this section we simplify the discussion by looking only at the context-free rules in the right parts of the synchronous rules defined in Table 1. These rules generate the right projection of $\mathcal{T}_{\mathcal{G}_{fuzzy}}$. We will discuss a recognition algorithm on the basis of these rules, with the tacit assumption that this can be extended to a parsing algorithm, to obtain fully bracketed strings as output, for a given partially bracketed strings as input.

Naively, one could use any parsing algorithm instantiated to the set of rules in the right parts of (5). For example, one could use the classical Earley algorithm (Earley, 1970; Stolcke, 1995). This manipulates items of the form $[\![\langle A \rightarrow \alpha \bullet$

For given CFG $\mathcal{G}$, with set N of nonterminals, set Σ of terminals, and start symbol $S \in N$, the SCFG $\mathcal{G}_{fuzzy}$ has one synchronous rule:

$$\langle\ A \to (_A\ X_1 \cdots X_m\ )_A,\ \ A(s_0^{(l)}, s_0^{(r)}) \to y^{(l)}\ Y_1 \cdots Y_m\ y^{(r)}\ \rangle \tag{5}$$

for each rule $A \to X_1 \cdots X_m$ in $\mathcal{G}$, for each choice of the pair:

$$(y^{(l)}, y^{(r)}) \in \{((_A,)_A), ((,))\} \cup \{[_A, [, \varepsilon\} \times \{]_A,], \varepsilon\} \tag{6}$$

and for each choice of pairs:

$$(s_i^{(l)}, s_i^{(r)}) \in \{((_B,)_B) \mid B \in N\} \cup \{((,))\} \cup (\{[_B \mid B \in N\} \cup \{[, \varepsilon\}) \times (\{]_B \mid B \in N\} \cup \{], \varepsilon\})$$

for each i ($1 \le i \le m$) such that $X_i \in N$, and $(s_i^{(l)}, s_i^{(r)}) = (\varepsilon, \varepsilon)$ for each i such that $X_i \in \Sigma$, and:

$$Y_i = \begin{cases} X_i(s_i^{(l)}, s_i^{(r)}) & \text{if } X_i \in N \\ X_i & \text{if } X_i \in \Sigma \end{cases} \tag{7}$$

under the following constraints:

$$\begin{array}{l} m = 1 \wedge (s_1^{(l)}, s_m^{(r)}) \in \{((_A,)_A), ((,))\}\ \to\ (y^{(l)} \neq \varepsilon\ \vee y^{(r)} \neq \varepsilon)\ \wedge \\ (s_1^{(l)} = [_A \vee s_1^{(l)} = [)\ \to\ y^{(l)} \neq \varepsilon\ \wedge \\ (s_m^{(r)} =]_A \vee s_m^{(r)} =])\ \to\ y^{(r)} \neq \varepsilon \end{array} \tag{8}$$

and:

$$(s_0^{(l)}, s_0^{(r)}) = \begin{cases} (y^{(l)}, y^{(r)}) & \text{if } (y^{(l)}, y^{(r)}) \in \{((_A,)_A), ((,))\} \\ (s_1^{(l)}, s_m^{(r)}) & \text{if } m = 1 \wedge y^{(l)} = y^{(r)} = \varepsilon\ \wedge (s_1^{(l)}, s_m^{(r)}) \in \{((_B,)_B) \mid B \in N \setminus \{A\}\} \\ (s^{(l)}, s^{(r)}) & \text{otherwise, where:} \\ & \quad s^{(l)} = \begin{cases} y^{(l)} & \text{if } y^{(l)} \in \{[_A, [\} \\ s_1^{(l)} & \text{if } y^{(l)} = \varepsilon \wedge s_1^{(l)} \in \{[_B \mid B \in N \setminus \{A\}\} \\ \varepsilon & \text{otherwise} \end{cases} \\ & \quad s^{(r)} = \begin{cases} y^{(r)} & \text{if } y^{(r)} \in \{]_A,]\} \\ s_m^{(r)} & \text{if } y^{(r)} = \varepsilon \wedge s_m^{(r)} \in \{]_B \mid B \in N \setminus \{A\}\} \\ \varepsilon & \text{otherwise} \end{cases} \end{cases} \tag{9}$$

$\mathcal{G}_{fuzzy}$ further has one synchronous rule:

$$\langle\ S^\dagger \to S,\ \ S^\dagger \to S(s^{(l)}, s^{(r)})\ \rangle \tag{10}$$

for each choice of the pair:

$$(s^{(l)}, s^{(r)}) \in \{((_B,)_B) \mid B \in N\} \cup \{((,))\} \cup (\{[_B \mid B \in N\} \cup \{[, \varepsilon\}) \times (\{]_B \mid B \in N\} \cup \{], \varepsilon\})$$

where $S^\dagger$ is a new symbol.

Table 1: The SCFG $\mathcal{G}_{fuzzy}$ constructed out of CFG $\mathcal{G}$.

$\beta\rangle, i, j]\!]$, which means that of a rule $A \rightarrow \alpha\beta$ from the grammar, the first part α has been processed and was found to generate the substring $a_{i+1} \cdots a_j$ of a fixed input string $a_1 \cdots a_n$.

The problem is that there is a very large number of rules as in the right parts of (5). This number is in fact exponential in the length m of the largest rule from the original grammar $\mathcal{G}$. Casual inspection of the constraints on the rules in Table 1 reveals however that only the values of $s_0^{(l)}$, $s_0^{(r)}$, $y^{(l)}$, $s_1^{(l)}$, $s_m^{(r)}$, $y^{(r)}$ are ever used in the current rule. The values of $s_i^{(l)}$ for $1 < i \leq m$ and the values of $s_i^{(r)}$ for $1 \leq i < m$ can be safely ignored.

We therefore let the parser manipulate items of the form $[\![\langle A(s_0^{(l)}, s_0^{(r)}) \rightarrow \alpha' \bullet \beta' \rangle, [s_1^{(l)}, s_m^{(r)}], i, j]\!]$ where $\alpha'\beta' = y^{(l)}; \alpha; y^{(r)}$ and $A \rightarrow \alpha$ is a rule in $\mathcal{G}$. Each such item can be seen as an abbreviation of an item for the Earley algorithm for the right parts in (5), leaving out fields that are not useful. The values of $s_0^{(l)}$, $s_0^{(r)}$, $y^{(l)}$, $s_1^{(l)}$, $s_m^{(r)}$, $y^{(r)}$ are initially the empty string, and are gradually filled in as these values become known.

The recognition algorithm is presented as deduction system in Table 2. Step (11) is the initialization step of Earley's algorithm and Step (12) straightforwardly corresponds to the predictor step. Steps (14), (15) and (17) correspond to the scanner step of Earley's algorithm. In the side condition of (17), the constraints on the allowable combinations of $y^{(l)}$, $s_1^{(l)}$, $s_m^{(r)}$, $y^{(r)}$ are checked, and $s_0^{(l)}$ and $s_0^{(r)}$ are determined on the basis of the other values.

The steps (13) and (16) have no direct equivalent in Earley's original algorithm. They are motivated by the intention to give uniform treatment to context-free rules with and without brackets. Step (18) straightforwardly corresponds to the completer step of Earley's algorithm, given the abbreviated form of our items, as explained above. Acceptance is expressed by step (19).

Deduction systems like this have a common algorithmic interpretation; see for example McAllester (2002). The time complexity of our algorithm is $\mathcal{O}(n^3 \cdot |\mathcal{G}|^2)$, where $|\mathcal{G}|$ is the size of the input grammar. This can be brought down as usual to $\mathcal{O}(n^3 \cdot |\mathcal{G}|)$ using techniques from Graham et al. (1980), which bring the complexity in line with the best known practical parsing algorithms for context-free grammars. Note that if the values of $s_0^{(l)}$, $s_0^{(r)}$, $y^{(l)}$, $s_1^{(l)}$, $s_m^{(r)}$, $y^{(r)}$ in an item $[\![\langle A(s_0^{(l)}, s_0^{(r)}) \rightarrow \alpha' \bullet \beta' \rangle, [s_1^{(l)}, s_m^{(r)}], i, j]\!]$, with $\alpha'\beta' = y^{(l)}; \alpha; y^{(r)}$, are anything other than ϵ, then they are uniquely determined by i and j. It is for this reason that the number of possible labelled and unlabelled brackets does not contribute an extra factor to the time complexity.

The recognition algorithm can also be straightforwardly extended to compute the most likely parse or the inside probability, similarly to how this is done by Jelinek et al. (1992). Note that unambiguity is essential in the latter case. The probabilities manipulated by the parser would then be based on the probabilities of rules from the original grammar, similarly to how this is normally done in probabilistic parsing algorithms based on Earley's algorithm (Stolcke, 1995; Nederhof, 2003).

7 Experiments

We have implemented the construction from Table 1 and the parsing algorithm from Table 2. Our aim was to assess the feasibility in practical terms. The latter algorithm was based on an implementation of the standard Earley algorithm, which we used as a base line. The implementation language is C++ and the experiments were performed on a laptop computer with a 2.66 GHz Intel Core 2 Duo processor.

First, a context-free grammar was extracted from sections 2-21 of the Penn Treebank, with NoTransform and NoEmpties as in (Klein and Manning, 2001), and unary rules were collapsed. This grammar has 84613 rules, 372 nonterminals and 44389 terminals (words). With this grammar, we parsed the (unbracketed) sentences from section 23 that had length 10 or less. Of these, 92 sentences were outside the language generated by the grammar and were discarded. Parsing of the remaining 178 sentences using the standard Earley algorithm took 8m27s in total.

Next, we tried to construct a context-free grammar that generates partially bracketed sentences without spurious ambiguity, as the right-projection of the construction in Table 1. Predictably, this was found to be infeasible for any but very small subsets of our input grammar, because of the exponential behaviour in the length of right-hand sides.

Lastly, the extended Earley algorithm from Table 2 was applied on the same 178 sentences, but now in partially bracketed form. We started with the fully bracketed sentences, as they appear in

Initialize

$$\frac{}{[\![\langle S(\varepsilon,\varepsilon) \to \bullet; \alpha; \rangle, [\varepsilon,\varepsilon], 0, 0]\!]} \left\{ (S \to \alpha) \in \mathcal{G} \right. \tag{11}$$

Predict

$$\frac{[\![\langle A(\varepsilon,\varepsilon) \to y^{(l)}; \alpha \bullet B\beta; \rangle, [s_1^{(l)}, s_m^{(r)}], i, j]\!]}{[\![\langle B(\varepsilon,\varepsilon) \to \bullet; \gamma; \rangle, [\varepsilon,\varepsilon], i, i]\!]} \left\{ (B \to \gamma) \in \mathcal{G} \right. \tag{12}$$

No open

$$\frac{[\![\langle A(\varepsilon,\varepsilon) \to \bullet; \alpha; \rangle, [\varepsilon,\varepsilon], i, i]\!]}{[\![\langle A(\varepsilon,\varepsilon) \to ; \bullet\alpha; \rangle, [\varepsilon,\varepsilon], i, i]\!]} \tag{13}$$

Open

$$\frac{[\![\langle A(\varepsilon,\varepsilon) \to \bullet; \alpha; \rangle, [\varepsilon,\varepsilon], i, i]\!]}{[\![\langle A(\varepsilon,\varepsilon) \to y^{(l)}; \bullet\alpha; \rangle, [\varepsilon,\varepsilon], i, i+1]\!]} \left\{ y^{(l)} = a_{i+1} \in \{(_A, (, [_A, [\right. \tag{14}$$

Scan

$$\frac{[\![\langle A(\varepsilon,\varepsilon) \to y^{(l)}; \alpha \bullet a\beta; \rangle, [s_1^{(l)}, s_m^{(r)}], i, j]\!]}{[\![\langle A(\varepsilon,\varepsilon) \to y^{(l)}; \alpha a \bullet \beta; \rangle, [s_1^{(l)}, s_m^{(r)}], i, j+1]\!]} \left\{ a = a_{j+1} \right. \tag{15}$$

No close

$$\frac{[\![\langle A(\varepsilon,\varepsilon) \to y^{(l)}; \alpha\bullet; \rangle, [s_1^{(l)}, s_m^{(r)}], i, j]\!]}{[\![\langle A(s_0^{(l)}, s_0^{(r)}) \to y^{(l)}; \alpha; \bullet\rangle, [s_1^{(l)}, s_m^{(r)}], i, j]\!]} \begin{cases} \text{the constraint in (8) with } y^{(r)} = \varepsilon \\ (s_0^{(l)}, s_0^{(r)}) \text{ defined by (9)} \end{cases} \tag{16}$$

Close

$$\frac{[\![\langle A(\varepsilon,\varepsilon) \to y^{(l)}; \alpha\bullet; \rangle, [s_1^{(l)}, s_m^{(r)}], i, j]\!]}{[\![\langle A(s_0^{(l)}, s_0^{(r)}) \to y^{(l)}; \alpha; y^{(r)}\bullet\rangle, [s_1^{(l)}, s_m^{(r)}], i, j+1]\!]} \begin{cases} y^{(r)} = a_{j+1} \in \{)_A,),]_A,]\} \\ \text{the constraint in (8)} \\ (s_0^{(l)}, s_0^{(r)}) \text{ defined by (9)} \end{cases} \tag{17}$$

Complete

$$\frac{\begin{array}{l}[\![\langle A(\varepsilon,\varepsilon) \to y^{(l)}; \alpha \bullet B\beta; \rangle, [s_1^{(l)}, s_m^{(r)}], i, j]\!] \\ [\![\langle B(t_0^{(l)}, t_0^{(r)}) \to z^{(l)}; \gamma; z^{(r)}\bullet\rangle, [t_1^{(l)}, t_{m'}^{(r)}], j, k]\!]\end{array}}{[\![\langle A(\varepsilon,\varepsilon) \to y^{(l)}; \alpha B \bullet \beta; \rangle, [u_1^{(l)}, u_m^{(r)}], i, k]\!]} \begin{cases} u_1^{(l)} = \begin{cases} t_0^{(l)} & \text{if } \alpha = \varepsilon \\ s_1^{(l)} & \text{otherwise} \end{cases} \\ u_m^{(r)} = \begin{cases} t_0^{(r)} & \text{if } \beta = \varepsilon \\ s_m^{(r)} & \text{otherwise} \end{cases} \end{cases} \tag{18}$$

Accept

$$\frac{[\![\langle S(s_0^{(l)}, s_0^{(r)}) \to y^{(l)}; \gamma; y^{(r)}\bullet\rangle, [s_1^{(l)}, s_m^{(r)}], 0, n]\!]}{\textit{accept}} \tag{19}$$

Table 2: Recognition of partially bracketed strings.

p	time
0.0	25m33s
0.2	68m08s
0.4	51m07s
0.6	34m36s
0.8	21m02s
1.0	11m48s

Table 3: CPU time for recognition of sentences of length ≤ 10 from section 23 of the Penn Treebank, for a varying probability p.

the treebank, and then randomly omitted a varying percentage of brackets and category labels.

This process of 'fuzzifying' a fully bracketed sentence proceeds in stages, with one potential step turning a bracket pair $(_A\ )_A$ into $[_A\ ]_A$. If this step does not happen, there is another potential step turning $(_A\ )_A$ into $(\ )$. For each labelled square bracket individually, a step may remove the label, which is then optionally followed by a step removing the bracket altogether. The process is parameterized with a value p, which expresses the probability that a step of fuzzifying does not happen. Hence, $p = 0$ means that all annotation is removed and $p = 1$ means that all brackets and labels are kept.

The results are given in Table 3. The first row of the table corresponds to the sentences in unannotated form. The running time is higher than the baseline of the standard Earley algorithm. This was to be expected, as there are some extra steps in Table 2, introduced for handling of brackets, and these steps are performed even if the input contains no brackets at all. Nonetheless, the running time is of the same order of magnitude.

In the next few rows of the table we see that the running time increases further. Again, this is to be expected, as the presence of brackets induces multiple instances of parsing items where there would be only one in the unbracketed case. When close to 100 % of the brackets are maintained, the running time again decreases. This is because the brackets reduce ambiguity.

One may object that the parsing of fully bracketed sentences should take close to 0 seconds, as those sentences are already parsed. However, we have not introduced any further optimizations to the Earley algorithm apart from those presented in Table 2, and the predictive nature of the algorithm leads to many steps creating different partial analyses at an open bracket of which all but one is discarded upon finding the matching close bracket.

The main objective was to investigate to which extent parsing of partially bracketed structures is possible, under the constraint that no spurious ambiguity should arise. Our experiments show that the running time is of the same order of magnitude as parsing of unbracketed strings using the standard Earley algorithm. Further refinements can be expected to reduce the running time.

For example, we found a straightforward optimization that is realized by letting parts of the check from condition (8) happen as soon as possible, rather than delaying them until completion of an item in (16) or (17). In concrete terms, the compatibility of an open bracket $y^{(l)}$ and the value of $s_1^{(l)}$ coming from the first member in the right-hand side can be checked as soon as that first member is known. This and other optimizations lead to a more involved formulation however, and for presentational reasons we abstain from further discussion.

Further note that the general ideas that led to Table 2 starting from the construction in Table 1 can as easily be used to derive other parsing algorithms for partially bracketed strings, using any other parsing strategy such as the bottom-up Earley algorithm (Sikkel, 1997) and left-corner parsing (Rosenkrantz and Stearns, 1970).

8 Conclusions

This paper has introduced a sound and elegant theoretical framework for processing partially bracketed strings. That is, an input string may contain any combination of matched or unmatched and labelled or unlabelled brackets.

Our theory, which uses synchronous CFGs, led us to a procedure based on Earley's algorithm. Its effectiveness was shown in experiments using a practical grammar. Despite having implemented few optimizations, we found that the time measurements show promising results, and considerable speed-ups may be expected by further refinement of the implementation. Use for the purpose of interactive parse selection therefore seems feasible. Further work will be needed to determine to which extent linguists benefit from being able to specify partially bracketed structures.

Acknowledgements

This work is partially supported by the Spanish MICINN under the MIPRCV Consolider Ingenio 2010 (CSD2007-00018), MITTRAL (TIN2009-14633-C03-01), and Prometeo (PROMETEO/2009/014) research projects, and the FPU fellowship AP2006-01363.

References

S. Barrachina, O. Bender, F. Casacuberta, J. Civera, E. Cubel, S. Khadivi, A. Lagarda, H. Ney, J. Tomás, E. Vidal, and J.-M. Vilar. 2009. Statistical approaches to computer-assisted translation. *Computational Linguistics*, 35(1):3–28.

J. Earley. 1970. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102, February.

S.L. Graham, M.A. Harrison, and W.L. Ruzzo. 1980. An improved context-free recognizer. *ACM Transactions on Programming Languages and Systems*, 2:415–462.

F. Jelinek, J.D. Lafferty, and R.L. Mercer. 1992. Basic methods of probabilistic context free grammars. In P. Laface and R. De Mori, editors, *Speech Recognition and Understanding — Recent Advances, Trends and Applications*, pages 345–360. Springer-Verlag.

D. Klein and C.D. Manning. 2001. Parsing with treebank grammars: Empirical bounds, theoretical models, and the structure of the Penn treebank. In *39th Annual Meeting and 10th Conference of the European Chapter of the Association for Computational Linguistics, Proceedings of the Conference*, pages 338–345, Toulouse, France, July.

M.P. Marcus, B. Santorini, and M.A. Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn treebank. *Computational Linguistics*, 19(2):313–330.

D. McAllester. 2002. On the complexity analysis of static analyses. *Journal of the ACM*, 49(4):512–537.

M.-J. Nederhof. 2003. Weighted deductive parsing and Knuth's algorithm. *Computational Linguistics*, 29(1):135–143.

F. Pereira and Y. Schabes. 1992. Inside-outside reestimation from partially bracketed corpora. In *30th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, pages 128–135, Newark, Delaware, USA, June–July.

D.J. Rosenkrantz and R.E. Stearns. 1970. Properties of deterministic top-down grammars. *Information and Control*, 17:226–256.

R. Sánchez-Sáez, J.A. Sánchez, and J.M. Benedí. 2009. Interactive predictive parsing. In *Proceedings of the 11th International Conference on Parsing Technologies*, pages 222–225, Paris, France, October.

R. Sánchez-Sáez, J.A. Sánchez, and J.M. Benedí. 2010. Confidence measures for error discrimination in an interactive predictive parsing framework. In *The 23rd International Conference on Computational Linguistics, Posters Volume*, pages 1220–1228, Beijing, China, August.

G. Satta and E. Peserico. 2005. Some computational complexity results for synchronous context-free grammars. In *Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 803–810.

K. Sikkel. 1997. *Parsing Schemata*. Springer-Verlag.

A. Stolcke. 1995. An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. *Computational Linguistics*, 21(2):167–201.

M. Wieling, M.-J. Nederhof, and G. Van Noord. 2005. Parsing partially bracketed input. In *Computational Linguistics in the Netherlands*, pages 1–16.