An improved Earley parser with LTAG

Yannick de Kercadio

LIMSI-CNRS, BP 133, F-91403 Orsay cedex, FRANCE
TALANA, UFR de Linguistique, Université Paris 7
kercadio@talana.linguist.jussieu.fr or kercadio@limsi.fr

1 Introduction

This paper presents an adaptation of the Earley algorithm (EARLEY, 1968) for parsing with lexicalized tree-adjoining grammars (LTAGs).

This algorithm constructs the derivation tree following a top-down strategy and verifies the valid prefix property. Many earlier algorithm do not have both of this properties (SCHABES, 1994). The Earley-like algorithm described in (SCHABES and JOSHI, 1988) verifies the valid prefix property, but the algorithm presented here is thought to be easier to improve using some properties of LTAGs.

2 Representation of a LTAG with a set of rules

A LTAG is a context-free grammar (CFG) on trees, the elementary operations of which are the adjunction and the substitution. The Earley algorithm can be used for parsing with any CFG insofar as the elementary operation is the concatenation. Hence, the Earley algorithm cannot simply be used for LTAGs, but the meaning of an edge in the derivation tree needs to be specified in terms of words strings and concatenations.

Substitution and terminal nodes can be handled using ordinary context-free rules. Such a rule represents a node in the derivation tree and captures the linear word order of the derived string.

An adjunction can be seen as two correlated substitutions: the derived string of the part of the adjoined tree on the left of the foot node is inserted in some location while the other part of the string is inserted in some other location farther in the string. The string located between the two substitution points is the derived string of the subtree under the adjoined node. The correlation between these two substitutions is that either none or both of them should occur. thus a synchronization must be transmitted up to the second location in order to preserve this constraint.

The locations of these pairs of places follows a stack order: there is an equal number of "first places" and "second places" between two matching places. Therefore, a unique symbol (# hereafter) can be used to represent any "second place", while a βX notation can be used to represent a "first place" for an adjunction of a tree with root X.

The figure 1 shows a few rules representing some elementary trees. A star denotes a foot node in an auxiliary tree. The drawn links implements the correlation information between the two substitution points representing an adjunction. Because of the stack structure of this information, the links need not to be explicitly stored. Also note that these trees are flat (no VP). See (ABEILLÉ, 1991). This is not mandatory and the trees usually used for English can be encoded the same way.

As each node in the derivation tree represents an elementary tree, and as every elementary

I'd like to thank A. Abeillé, M.-H. Candito, F. Issac and P. Paroubek for their valuable help and advices

Rule for the transitive verb *to love* ($\alpha n0Vn1$), without adjunctions:

$$\alpha S \rightarrow \alpha N \quad \mathit{love} \quad \alpha N$$

Rule for the transitive verb *to love* ($\alpha n0Vn1$), with possible adjunctions on S and on V:

$$\begin{tikzcd}[column sep=small] \alpha S \rightarrow & \beta S \arrow[rrrrr, dash, bend left] & \alpha N & \beta V \arrow[rr, dash, bend left] & \mathit{love} & \# & \alpha N & \# \end{tikzcd}$$

Rule for the determiner *the* ($\beta DetN$), with a possible adjunction on the root N:

$$\begin{tikzcd}[column sep=small] \beta N \rightarrow & \beta N \arrow[rrr, dash, bend left] & \mathit{the} & * & \# \end{tikzcd}$$

Figure 1: Examples of rules

tree can be represented by a rule which capture the linear word order of the derived string, this is a way to capture the linear word order in the derivation tree. The usual derivation tree (as defined in (VIJAY-SHANKER, 1987)) can be obtained by linking the subtree of every "first place" to the left of the subtree of the matching "second place" and by storing the resulting structure under the "second place".

3 Earley-like parsing driven by the derivation tree

In this section, we show how the stacked relationships between the "first places" and "second places" can be represented in a structure which is suitable for the Earley algorithm.

Following Earley, a partial parsing can be represented by an item, which consists in a rule, a position in the rule (all the symbols located on its left have been recognized), and two lists of pairs of references to items. The first list keeps track of the requesters of the rule, that is to say the items which are waiting for the rule to be recognized in order to be shifted. The second element of each pair is used as a relay storage during the recognition of the second part of an auxiliary tree. The second list implements the previously mentioned stack of "first places". The first element of each pair it contains is the data part of the stack item. It is a reference to an item waiting on a foot symbol. The second element in each pair is used to implement the stack. It is a reference to an item waiting for an adjunction.

A number of primitive operations will be applied on this data structure. They are summed up in the table 2. When a primitive is applied on a given set, the second column indicates how many actions are to be taken. The rule and mark columns indicate which item is to be introduced. If no item with this rule and this position mark is present in the set, it is introduced with the indicated lists for the requesters list and the stack list. Otherwise, the indicated lists are merged with the ones of the existing item in the set. This merging step ensures that the spatial complexity has a polynomial upper bound.

The algorithm consists in working on each set in turn, following the word order. The initial set is initialized using **init**. Then an evolution stage applies a **predict** or **reduce** primitive on every newly introduced item, the type of which is chosen from the symbol in the rule which is right after the mark. For instance, if it is an αX (a substitution is expected), then **predict** α**(item, X)** is used. If there is no such symbol, them a **reduce** primitive is used, depending on the type (α or β) of the left part of the rule. This process is then run on each set in turn, replacing **inits** with a **shift** on every item expecting (i.e. with the mark right on the left of) the word associated with the current set.

The sentence is accepted if there is an item in the last set with a rule deriving the axiom (S), with the mark at the end of the rule, with an empty requesters list. It should be noted that this algorithm does not give an analysis of the sentence. An additional structure is required in each item to keep the analysis information.

primitive	**applied for each**	**rule**	**mark**	**req**	**stack**
init()	rule r with root αS	r	0	{}	{}
shift(item)	*once*	item.rule	item.mark + 1	item.req	item.stack
predict α(item, X)	rule r with root αX	r	0	{(item, -)}	{}
predict β(item, X)	rule r with root βX	r	0	{(item, -)}	{}
and	*once*	item.rule	item.mark + 1	item.req	{(-, item)}
predict $*$(item)	(x, y) in item.req	x.rule	x.mark + 1	x.req	{(item, x)}
reduce #(item)	(x, y) in item.stack, where x is not -	x.rule	x.mark + 1	{(item, y)}	x.stack
	(-, y) in item.stack	item.rule	item.mark + 1	item.req	y.stack
reduce α(item)	(x, y) in item.req	x.rule	x.mark + 1	x.req	x.stack
reduce β(item)	(x, y) in item.req	x.rule	x.mark + 1	x.req	y.stack

Figure 2: Primitives of the algorithm

However, every edge in the derivation tree is detected through the fact that a **reduce** primitive is run. This additional structure should cope with the ambiguities and permit a polynomial representation of ambiguities from other level of analysis (features unification, semantic analysis and so on). This is a quite general matter: the number of solutions to the problem of parsing being (potentially) exponential, a simple list of analyses would require an exponential time to be output. The usual assumption that the number of analyses is "small" is not acceptable in the context of parsing oral utterances (because of potential auto-repairing constructs). Therefore, the representation of the outputs should grow polynomially (and not exponentially) with the number of ambiguities.

4 Benefits in using this strategy

The top-down strategy of this algorithm has a trivial, but very useful property: this algorithm do not require the utterance to be cut into sentences in order to parse it. Instead, one can perform an **init** primitive in every set where a rule with the axiom as its left part and an empty requesters list is found. It has the effect of concurrently trying to parse a new sentence from this point. This property is very important when parsing oral utterances: there is no practical other way to find out where sentences begin and end.

Moreover, the combination of both the top-down strategy and the valid prefix property enables valuable performance improvements. Many of the LTAGs properties (SRINIVAS, 1997) can be used to avoid the introduction of unrelevant elementary trees, thus allowing the use of a richer grammar.

The data structures construct a derivation tree. Therefore, a rough semantic analysis can be performed to check whether some newly discovered potential edge in the derivation tree makes sense or not. If not, it can be invalidated as soon as it is discovered.

When features are used, they can be checked following only the derivation tree (the derived tree is not needed). This is due to the fact that the nodes in the derivation tree are more than simple atoms: they are the rules that have been used for parsing. Like with semantic analysis, the features unification can be done on partial analysis, after every reduction. However, it is not clear whether this would result in an improvement or not: the cost of the unification might overcome the benefits of invalidating some partial analysis as soon as possible.

Due to the lexicalization, terminals (words) are put in the trees during lexicon access. When a rule is invocated in a set S, it always contains at least one terminal (lexicalization). All the symbols on the left of the first terminal have to be recognized before the set where this terminal is to be found. This is a way to filter the candidate rules for recognizing these symbols.

Former parsers already used the span of trees to eliminate trees that are too large to parse the sentence (XTAG, for instance), but this algorithm permits considering the span properties locally, at every prediction stage.

Last but not least, the data structures used for this algorithm can be enriched in successive analysis stages. That is to say, when no analysis is found, it is possible to enrich the sets with new rules. This property is useful to construct a fault tolerant parser, accepting unknown words, using weighted syntactic rules (the weights indicating whether a given rule is linguistically perfect or somewhat deviant), and accounting for auto-repairing sequences in an oral utterance.

5 Prospects

Using these properties enables the design of an efficient oral-specific robust parser using a grammar of the written language (ABEILLÉ, 1991). We plan to incorporate a syntactic LTAG-based component in a working real-time speech understanding system (GAUVAIN et al., 1997,) to improve its recognition performances.

References

ABEILLÉ, A. 1991. *Une grammaire lexicalisée d'Arbres Adjoints pour le français.* Ph.D. thesis, Université Paris 7.

ABEILLÉ, A., K. BISHOP, S. COTE, and Y. SCHABES. 1990. A lexicalized tree adjoining grammar for english. Technical report, Department of Computer and Information Science, University of Pennsylvania, Philadelphia.

EARLEY, J. C. 1968. *An efficient context-free parsing algorithm.* Ph.D. thesis, Carnegie-Mellon University, Pittsburgh.

GAUVAIN, J.-L., S. BENNACEF, L. DEVILLERS, L. LAMEL, and S. ROSSET. 1997. Spoken language component of the mask kiosk. In K. Varghese and S. Pfleger, editors, *Human Comfort and Security of Information Systems.* Springer-Verlag, pages 93–103.

SCHABES, Y. 1994. Left-to-right parsing of lexicalized tree-adjoining grammars. *Computational Linguistics,* 10(4):506–524.

SCHABES, Y. and A. K. JOSHI. 1988. An earley-type parsing algorithm of tree adjoining languages. In *26th Meeting of the Association for Computational Linguistics,* Buffalo.

SRINIVAS, B. 1997. *Complexity of lexical descriptions and its relevance to partial parsing.* Ph.D. thesis, Department of Computer and Information Science, University of Pennsylvania, Philadelphia.

VIJAY-SHANKER, K. 1987. *A study of tree adjoining grammars.* Ph.D. thesis, Department of Computer and Information Science, University of Pennsylvania, Philadelphia.

VIJAY-SHANKER, K. and A. K. JOSHI. 1985. Some computational properties of tree adjoining grammars. In *23rd Meeting of the Association for Computational Linguistics,* pages 82–93, Chicago.