Word Order Constraints for Lexical Disambiguation of Interaction Grammars

Guillaume Bonfante, Bruno Guillaume, Mathieu Morey

Word Order Constraints for Lexical Disambiguation of Interaction Grammars

Guillaume Bonfante, Bruno Guillaume and Mathieu Morey
LORIA - Nancy-Université - INRIA Nancy Grand-Est
{guillaume.bonfante, bruno.guillaume, mathieu.morey}@loria.fr

Abstract

We propose a new method to perform lexical disambiguation of Interaction Grammars. It deals with the order constraints on words. Actually, the soundness of the method is due to an invariant property of the parsing of an Interaction Grammar. We show how this invariant can be computed statically from the grammar.

1 Introduction

Interaction Grammars are a lexicalized grammatical formalism. They share with Categorial Grammars the idea that words are composed of syntactic constituents with a notion of polarity. Some constituents are unsaturated: they "wait" for some resources and provide some other ones. Interaction Grammars also have, as Categorial Grammars, a mechanism to cope with the linear order on words.

We show in this paper that ordering constraints can be used to partially disambiguate the words of a sentence. But, first of all, let us state some of our wills. First, the issue considered here is to be thought in the context of syntactic analysis. We do not want to use statistical methods for lexical disambiguation since an error at that point cannot be recovered at the parsing step. Consequently, given a sentence, we accept to have more than one lexical tagging for it, as long as we can ensure to have the good ones (when they exist!).

Now, since we have to consider all possible lexical taggings to find the right ones, there is an immediate problem of complexity. Knowing that a word has typically about 10 corresponding lexical descriptions, for a short sentence of 10 words, we get 10^{10} possible taggings. It is not reasonable to treat them individually.

To avoid it, it is convenient to use an automaton to represent the set of all paths. This automaton has linear size with regard to the initial lexical ambiguity. The idea of using automata is not new. In particular, methods based on Hidden Markov Models (HMM) use such a technique for part-of-speech tagging [3, 4]. Using automata, one may conceive dynamic programming procedures, and consequently benefits from an exponential temporal speed up, together with the space one.

2 Interaction Grammars

We give here a very short and simplified description of IG and then, an example to illustrate them at work; we refer the reader to [2] for a complete and detailed presentation.

The final structure, used as output of the parsing process, is an ordered tree called **parse tree** (PT).

An example of a PT is given in Figure 1, on the right. A PT for a sentence contains the words of the sentence or the empty word ϵ in its leaves (the left-right order of the tree leaves follows the left-right order of words in the input sentence). The internal nodes of a PT represent the constituents of the sentence. The morpho-syntactic properties of these constituents are described with feature structures (only the category is shown in the figure).

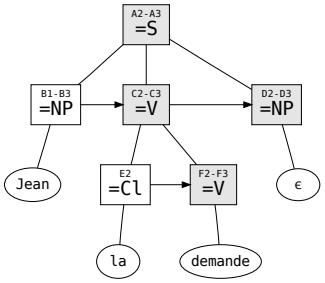

Figure 1: The PT of *"Jean la demande."* [John asks for it.]

As IG use the Model-Theoretic Syntax (MTS) framework, a PT is defined as the model of a set of constraints. Constraints are defined at the word level: words are associated to a set of constraints formally described as a **polarized tree description** (PTD). A PTD is a set of nodes provided with relations between these nodes. In Figure 2, the three PTDs given on the left are used to build the model above. The relations used in the PTDs are: dominance (lines) and immediate sisterhood (arrows). Nodes represent syntactic constituents and relations express structural dependencies between these constituents; moreover, nodes carry a polarity (polarities are $\{+, -, =, \sim\}$) which expresses a saturation constraint.

Now, we define a PT to be a **model** of a set of PTDs if there is a surjective function $\mathcal{I}$ from nodes of the PTDs to nodes of the PT such that:

- relations in the PTDs are realized in the PT: if M is a daughter (resp. immediate sister) of N in some PTD then $\mathcal{I}(M)$ is a daughter (resp. immediate sister) of $\mathcal{I}(N)$;
- each node N in the PT is saturated: the composition of the polarities of the nodes in $\mathcal{I}^{-1}(N)$ with the associative and commutative rule given in Table 3 is $=$;
- the feature structure of a node N in the PT is the unification of the feature structures of the nodes in $\mathcal{I}^{-1}(N)$.

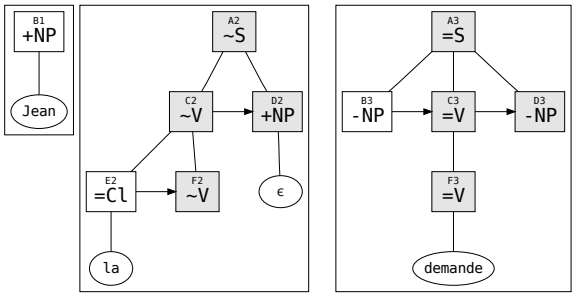

Figure 2: PTDs for the sentence *"Jean la demande."* [John asks for it.]

One of the strong points of IG is the flexibility given by the MTS approach: PTDs can be partially superposed to produce the final tree (superposition is limited in usual CG or in TAG for instance). In our example, the four grey nodes in the PTD which contains *"la"* are superposed to the four grey nodes in the PTD which contains *"demande"* to produce the four grey nodes in the model. In the full IG formalism, relations between nodes can be underspecified, for instance a PTD can impose a node to be an ancestor of another one without constraining the length of the path in the

model. In full IG, polarities are linked to features, not to nodes. A node can contain several polarities. The methods we present here can be straightforwardly extended to full IG (with unessential technical details).

An IG is made of:

- A finite set $\mathcal{W}$ of words;
- A finite set $\mathcal{G}$ of PTDs (without the word attached to them);
- A function $\ell : \mathcal{W} \longrightarrow \mathcal{P}(\mathcal{G})$ which associates words with set of PTDs.

	$\sim$	$-$	$+$	$=$
$\sim$	$\sim$	$-$	$+$	$=$
$-$	$-$		$=$	
$+$	$+$	$=$		
$=$	$=$			

Figure 3: Polarity composition

Now, to parse a sentence $S = w_1 \dots w_n$, we have to:

- first, for each w_i, choose one of the PTDs $d_i \in \ell(w_i)$ (we call **lexical tagging** a choice $\{d_1, \dots, d_n\}$ of one PTD for each word of the sentence);
- find a parse tree which is a model of the set of PTDs of the lexical tagging.

3 The Left-Right Principle

Computing a model of a sentence from a lexical tagging requires to saturate all the polarity constraints of its PTDs. To build a model, each node which is not saturated (with a polarity $+$, $-$ or $\sim$) has to be merged with its "companions", namely nodes from other PTDs that will saturate it.

Now let us take a look at the grammar, independently of any sentence, and try to find the "potential companions" of a given unsaturated node. Actually, as our grammar models word order constraints, it can be the case that using a node M to saturate a node N requires the PTD corresponding to M to be on the left (resp. on the right) of N. Therefore, for each unsaturated node N of the grammar $\mathcal{G}$, we can enumerate all of its potential companions using two possibly overlapping lists: its left potential companions (written $LPC(N)$) and its right potential companions (written $RPC(N)$). By extension, we say that $d \in LPC(N)$ (resp. $d \in RPC(N)$) for some PTD d whenever $\exists M \in d : M \in LPC(N)$ (resp. $\exists M \in d : M \in RPC(N)$).

Observe that constructing the LPC and the RPC sets can be done independently from any sentence, it is a property of the grammar which can be computed from the grammar itself.

Let us consider a sentence $w_1 w_2 \cdots w_n$ and one of its lexical taggings $d_1 d_2 \cdots d_n$. Suppose that there is one node $N \in d_i$ for some i for which there is neither some $d_j \in LPC(N)$ with $j < i$ nor some $d_k \in RPC(N)$ with $k > i$. Then, without performing deep parsing, we can state that such a lexical tagging has no model. So, it is a necessary condition of the success of parsing that there is no such node. We call this the Left-Right Principle.

4 Implementation of the Left-Right Principle with automata

We have seen above that the Left-Right principle applies to lexical taggings. As a matter of fact, in this section we keep the promise we made in the introduction of this paper: we show that it can be computed by means of automata, saving space and time. Actually, we propose two implementations of the Left-Right principle, an exact one and an approximate one. The latter is really fast and can be used as a first step before applying the first one.

4.1 Exact Left-Right disambiguation (ELR)

Given a sentence $w_1 \cdots w_n$, a PTD $d \in \ell(w_i)$ and a node N of d, we can build the *companionship automaton* $\mathcal{A}(w_i, d, N)$ for the sentence. It represents the saturation state of the polarity constraint corresponding to N after each choice of a PTD for a word.

Each state of $\mathcal{A}(w_i, d, N)$ is labelled with a couple (j, x), with j the position of the last considered word and x the saturation state (`Open` or `Close`). A state is labelled with `Close` when all of its incoming paths fulfill the polarity constraint of N. Otherwise the state is labelled with `Open`.

More formally, $\mathcal{A}(w_i, d, N)$ is defined as follows. States are a subset of $\mathbb{N} \times \{\texttt{Open}, \texttt{Close}\}$. Transitions $(j-1, x) \xrightarrow{d'} (j, y)$ are labelled by $d' \in \ell(w_j)$. The value of y is determined in the following way :

- if $(j = i) \wedge (d = d')$ then $y = x$,
- if $(j = i) \wedge (d \neq d')$ then $y = \texttt{Close}$,
- if $(j < i) \wedge (d' \in LPC(N))$ then $y = \texttt{Close}$,
- if $(j > i) \wedge (d' \in RPC(N))$ then $y = \texttt{Close}$,
- otherwise $y = \texttt{Open}$.

The initial state is $(0, \texttt{Open})$ and the unique accepting final state is $(n, \texttt{Close})$.

For every word w_i of the sentence, we construct the companionship automaton $\mathcal{A}(w_i, d, N)$ of every polarized node N in every PTD $d \in \ell(w_i)$. The intersection of these automata represents all the possible lexical taggings of the sentence which respect the Left-Right Principle. That is, we output:

$$\bigcap_{1 \leq i \leq n,\ d \in \ell(w_i),\ N \in d} \mathcal{A}(w_i, d, N)$$

Let us say that for the sentence *"Jean la demande."*, the possible PTDs are those described in Figure 4: $\ell(\text{"}Jean\text{"})$ ={Jean_NP}, $\ell(\text{"}la\text{"})$ ={la_Det, la_Clit, la_CN}, $\ell(\text{"}demande\text{"})$ ={demande_V, demande_CN}. Then Figure 5 represents the automaton $\mathcal{A}(\text{"}demande\text{"}, \text{demande\_V}, \text{D3})$.

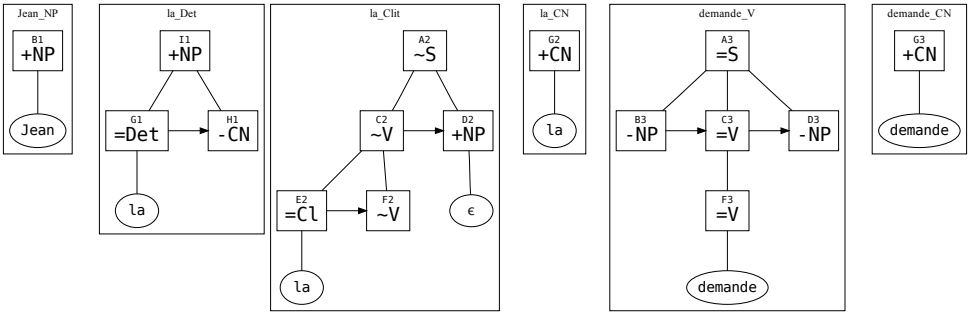

Figure 4: Possible PTDs for the sentence *"Jean la demande."*

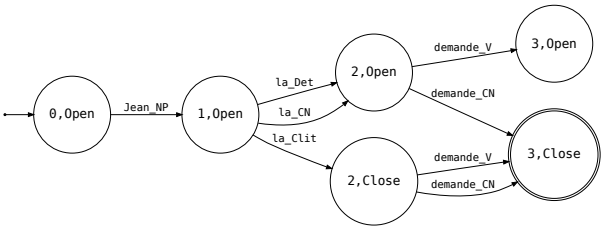

Figure 5: $\mathcal{A}$("*demande*", demande_V, D3) for the sentence *"Jean la demande."*

4.2 Quick and dirty approximation (QLR)

The issue with the previous algorithm is that it involves a large number of automata (actually $O(n)$) where n is the size of the input sentence. Each of these automata has size $O(n)$. The theoretical complexity of the intersection is then $O(n^n)$. Sometimes, we face the exponential. So, let us provide an algorithm which approximates the Principle.

Again, we consider a sentence $w_1 \cdots w_n$. Suppose that for some $d \in \ell(w_i)$ and some $N \in d$, there is no $d' \in LPC(N)$ with $d' \in \ell(w_j), j < i$ nor some $d'' \in RPC(N)$ with $d'' \in \ell(w_k), k > i$. Then lexical taggings containing d at position i have no model, hence, d can be removed.

This can be computed by a double-for loop: for each node N of a PTD $d \in \ell(w_i)$ in the sentence, verify that there is a companion node $M \in d'$, $d' \in \ell(w_j)$ for it. If it is not the case, simply remove the lexical choice d for the word w_i. Observe that the cost of this algorithm is $O(n^2)$.

Note that one must iterate this algorithm until a fix-point is reached. Indeed, removing a PTD which serves as a potential companion breaks the verification. Nevertheless, since for each step before the fix-point is reached, we remove at least one PTD, we iterate the double-for at most $O(n)$ times. The complexity of the whole algorithm is then $O(n^3)$.

Let us see on an example the difference between the two algorithms ELR and QLR. Take a very simple grammar, with a single word w associated to two PTDs d_1, d_2 such that $RPC(d_1) = LPC(d_1) = \{d_2\}, LPC(d_2) = RPC(d_2) = \{d_1\}$[1]. For the sentence ww, the algorithm QLR keeps the four lexical taggings d_1d_1, d_1d_2, d_2d_1 and d_2d_2, whereas ELR only keeps d_1d_2 and d_2d_1.

5 Experimental results

We present here some results obtained[2] with our methods. Our test corpus is composed of 189 sentences (with a mean length of 10 words) extracted from the French newspaper *"Le Monde"*. Our linguistic resources are a French IG [5] and a lexicon built from freely available French resources. In the table below, we give the filtering time for all sentences and filtering ratio for grammatical[3] sentences.

The filtering methods we consider are: POL (as a baseline) which uses a global filter based on polarity counting (described in [1]); QLR is the method described in 4.2 and ELR is described

[1]For instance, take the PTDs made of one node polarized $+N$ for d_1 and $-N$ for d_2.

[2]These results are obtained with a PC (Pentium® D930, 3.0Ghz, 4Go).

[3]Here grammatical means that they are accepted by the grammar.

in 4.1. Note that we have not reported experiments about the ELR method alone because the filtering time is too high for some sentences.

Values in the table are percentiles. For instance the value 1.20s (in the box) means that with the ELR+POL filtering, 75% of the 189 sentences are filtered in 1.20s **at most**. The ratio $3.06 \cdot 10^5$ (also in a box) means that 85% of the 133 sentences have a filtering ratio of $3.06 \cdot 10^5$ **at least**.

		POL	QLR	QLR + POL	ELR + POL
Filtering time	50%	0.38s	0.03s	0.07s	0.11s
for all sentences	75%	2.84s	0.07s	0.38s	1.20s
(189 sentences)	85%	9.83s	0.11s	0.98s	5.92s
Filtering ratio for	50%	$2.86 \cdot 10^5$	$8.06 \cdot 10^3$	$5.76 \cdot 10^7$	$1.70 \cdot 10^8$
grammatical sentences	75%	$3.07 \cdot 10^4$	$9.68 \cdot 10^2$	$1.01 \cdot 10^6$	$3.24 \cdot 10^6$
(133 sentences)	85%	$1.99 \cdot 10^4$	$2.66 \cdot 10^2$	$3.06 \cdot 10^5$	$7.52 \cdot 10^5$

It is clear from the first 2 experiments that the QLR is much more efficient (85% of the sentences can be filtered in less than 0.11s) but the ratio is lower than the baseline. The combination of the two methods (QLR+POL) greatly improves the baseline both in filtering time and ratio. The last experiment is more time consuming (bigger automata are built) but it is still usable in practice and shows the higher impact that can be reached with our methods (number of taggings divided by at least $7.52 \cdot 10^5$ for 85% of the sentences).

References

[1] G. Perrier G. Bonfante, B. Guillaume. Polarization and abstraction of grammatical formalisms as methods for lexical disambiguation. In *Proceedings of CoLing 2004*, 2004.

[2] Bruno Guillaume and Guy Perrier. Interaction Grammars. Research Report RR-6621, INRIA, 2008.

[3] Julien Kupiec. Robust Part-of-Speech Tagging Using a Hidden Markov Model. *Computer Speech and Language*, 6(3):225–242, 1992.

[4] Bernard Merialdo. Tagging English Text with a Probabilistic Model. *Computational linguistics*, 20:155–157, 1994.

[5] G. Perrier. A french interaction grammar. In *proceedings od the 6th International Conference on Recent Advances in Natural Language Processing*, pages 463–467, Borovets, Bulgaria, 2007.