Dependency Constraints for Lexical Disambiguation

Guillaume Bonfante, Bruno Guillaume, Mathieu Morey

Dependency Constraints for Lexical Disambiguation

Guillaume Bonfante
LORIA INPL
guillaume.bonfante@loria.fr

Bruno Guillaume
LORIA INRIA
bruno.guillaume@loria.fr

Mathieu Morey
LORIA Nancy-Université
mathieu.morey@loria.fr

Abstract

We propose a generic method to perform lexical disambiguation in lexicalized grammatical formalisms. It relies on dependency constraints between words. The soundness of the method is due to invariant properties of the parsing in a given grammar that can be computed statically from the grammar.

1 Introduction

In this work, we propose a method of lexical disambiguation based on the notion of dependencies. As this has been done by Boullier in (Bou03), our method is not based on statistics, nor on heuristics, but it is based on a necessary condition of the deep parsing. 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!). This property is particulary useful for deep parsing which won't fail due to an error at the disambiguation step.

In modern linguistics, Lucien Tesnière developed a formal and sophisticated theory with dependencies (Tes59). Nowadays, many current grammatical formalisms rely more or less explicitly on the notion of dependencies between words. The most straightforward examples are formalism in the Dependency Grammars family but it is also true of the phrase structure based formalisms which consider that words introduce incomplete syntactic structures which must be completed by other words. This idea is at the core of Categorial Grammars (CG) (Lam58) and all its trends such as Abstract Categorial Grammars (ACG) (dG01) or Combinatory Categorial Grammars (CCG) (Ste00), being mostly encoded in their type system. Dependencies in CG were studied in (MM91) and for CCG, in (CHS02; KK09). Other formalisms can be viewed as modeling and using dependencies, such as Tree Adjoining Grammars (TAG) (Jos87) with their substitution and adjunction operations. Dependencies for TAG are studied in (JR03). More recently, (MGP09) shows that it is also possible to extract a dependency structure from a syntactic analysis in Interaction Grammars (IG) (GP08).

Another much more recent concept of polarity can be used in grammatical formalisms to express that words introduce incomplete syntactic structures. IG directly use polarities to describe these structures but it is also possible to use polarities in other formalisms in order to make explicit the more or less implicit notion of incomplete structures: for instance, in CG (Lam08) or in TAG (Kah06; BGP04; GK05). On this regard, (MGP09) exhibits a direct link between polarities and dependencies. This encourages us to say that in many respects dependencies and polarities are two sides of the same coin.

The aim of this paper is to show that dependencies can be used to express constraints on the taggings of sentence and hence these dependency constraints can be used to partially disambiguate the words of a sentence. We will see that, in practice, using the link between dependencies and polarities, these dependency constraints can be computed directly from polarized structures.

Concerning disambiguation, 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 this, 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 (Kup92; Mer94). Using automata, we benefit from dynamic programming procedures,

and consequently from an exponential temporal ans space speed up.

2 Abstract Grammatical Framework

Our filtering method is applicable to any lexicalized grammatical formalism which exhibits some basic properties. In this section we establish these properties and define from them the notion of Abstract Grammatical Framework (AGF).

Formally, an **Abstract Grammatical Framework** is an n-tuple $(\mathcal{V}, \mathcal{S}, \mathcal{G}, \mathbf{anc}, \mathcal{F}, \mathbf{p}, \mathbf{dep})$ where:

- $\mathcal{V}$ is the **vocabulary**: a finite set of words of the modeled natural language;
- $\mathcal{S}$ is the set of **syntactic structures** used by the formalism;
- $\mathcal{G} \subset \mathcal{S}$ is the **grammar**: the finite set of initial syntactic structures; a finite list $[t_1, \ldots, t_n]$ of elements of $\mathcal{G}$ is called a **lexical tagging**;
- $\mathbf{anc} : \mathcal{G} \rightarrow \mathcal{V}$ maps initial syntactic structures to their anchors;
- $\mathcal{F} \subset \mathcal{S}$ is the set of final syntactic structures that the parsing process builds (for instance trees);
- $\mathbf{p}$ is the **parsing function** from lexical taggings to finite subsets of $\mathcal{F}$;
- $\mathbf{dep}$ is the **dependency function** which maps a couple of a lexical tagging and a final syntactic structures to dependency structures.

Note that the $\mathbf{anc}$ function implies that the grammar is lexicalized: each initial structure in $\mathcal{G}$ is associated to an element of $\mathcal{V}$. Note also that no particular property is required on the dependency structures that are obtained with the $\mathbf{dep}$ function, they can be non-projective for instance.

We call **lexicon** the function (written ℓ) from $\mathcal{V}$ to subsets of $\mathcal{G}$ defined by:

$$\ell(w) = \{t \in \mathcal{G} \mid \mathbf{anc}(t) = w\}.$$

We will say that a lexical tagging $[t_1, \ldots, t_n]$ is a lexical tagging of the sentence $[\mathbf{anc}(t_1), \ldots, \mathbf{anc}(t_n)]$.

The final structures in $\mathbf{p}\ (L) \subset \mathcal{F}$ are called the **parsing solutions** of L.

In the following, in our examples, we will consider the ambiguous French sentence (1).

(1) *"La belle ferme la porte"*

Example 1 *We consider the following toy AGF, suited for parsing our sentence:*

- $\mathcal{V} =$ { "la", "belle", "ferme", "porte" }*;*
- *the grammar $\mathcal{G}$ is given in the table below: each $\times$ corresponds to an element in $\mathcal{G}$, written with the category and the French word as subscript. For instance, the French word* "porte" *can be either a common noun ("door") or a transitive verb ("hangs"); hence $\mathcal{G}$ contains the 2 elements $\texttt{CN}_{porte}$ and $\texttt{TrV}_{porte}$.*

	la	*belle*	*ferme*	*porte*
Det	$\times$			
LAdj		$\times$	$\times$	
RAdj		$\times$	$\times$	
CN	$\times$	$\times$	$\times$	$\times$
Clit	$\times$			
TrV			$\times$	$\times$
IntrV			$\times$	

In our example, categories stands for, respectively: determiner, left adjective, right adjective, common noun, clitic pronoun, transitive verb and intransitive verb.

With respect to our lexicon, for sentence (1), there are $3 \times 3 \times 5 \times 3 \times 2 = 270$ lexical taggings.

The parsing function $\mathbf{p}$ is such that 3 lexical taggings have one solution and the 267 remaining ones have no solution; we do not need to precise the final structures, so we only give the English translation as the result of the parsing function:

- $\mathbf{p}([\texttt{Det}_{la}, \texttt{CN}_{belle}, \texttt{TrV}_{ferme}, \texttt{Det}_{la}, \texttt{CN}_{porte}]) =$ {*"The nice girl closes the door"*}
- $\mathbf{p}([\texttt{Det}_{la}, \texttt{LAdj}_{belle}, \texttt{CN}_{ferme}, \texttt{Clit}_{la}, \texttt{TrV}_{porte}]) =$ {*"The nice farm hangs it"*}
- $\mathbf{p}([\texttt{Det}_{la}, \texttt{CN}_{belle}, \texttt{RAdj}_{ferme}, \texttt{Clit}_{la}, \texttt{TrV}_{porte}]) =$ {*"The firm nice girl hangs it"*}

3 The Companionship Principle

We have stated in the previous section the framework and the definitions required to describe our principle.

3.1 Potential Companion

We say that $u \in \mathcal{G}$ is a **companion** of $t \in \mathcal{G}$ if $\mathbf{anc}(t)$ and $\mathbf{anc}(u)$ are linked by a dependency in $\mathbf{dep}(L, t)$ for some lexical tagging L which contains t and u and some $t \in \mathbf{p}(L)$. The subset of elements of $\mathcal{G}$ that are companions of t is called the **potential companion set** of t.

The Companionship Principle says that if a lexical tagging contains some t but no potential companion of t, then it can be removed.

In what follows, we will generalize a bit this idea in two ways. First, the same t can be implied in more than one kind of dependency and hence it can have several different companion sets with respect to the different kind of dependencies. Secondly, it can be the case that some companion t has to be on the right (resp. on the left) to fulfill its duty so we will consider pairs of sets rather than sets. These generalizations are done through the notion of atomic constraints defined below.

3.2 Atomic constraints

We say that a pair $(\mathcal{L}, \mathcal{R})$ of subsets of $\mathcal{G}$ is an **atomic constraint** for an initial structure $t \in \mathcal{G}$ if for each lexical tagging $L = [t_1, \ldots, t_n]$ such that $\mathbf{p}(L) \neq \emptyset$ and $t = t_i$ for some i then:

- either there is some $j < i$ such that $t_j \in \mathcal{L}$,
- or there is some $j > i$ such that $t_j \in \mathcal{R}$.

In other words, $(\mathcal{L}, \mathcal{R})$ lists the potential companions of t, respectively on the left and on the right.

A **system of constraints** for a grammar $\mathcal{G}$ is a function $\mathcal{C}$ which associates a finite set of atomic constraints to each element of $\mathcal{G}$.

The Companionship Principle is an immediate consequence of the definition of atomic constraints. It can be stated as the necessary condition:

The Companionship Principle
If a lexical tagging $[t_1, \ldots, t_n]$ has a solution then for all i and for all atomic constraints $(\mathcal{L}, \mathcal{R}) \in \mathcal{C}(t_i)$

- $\{t_1, \ldots, t_{i-1}\} \cap \mathcal{L} \neq \emptyset$
- or $\{t_{i+1}, \ldots, t_n\} \cap \mathcal{R} \neq \emptyset$.

Example 2 *Often, constraints can be expressed independently of the anchors. In our example, we use the category to refer to the subset of $\mathcal{G}$ of structures defined with this category:* $\texttt{LAdj}$ *for instance refers to the subset* $\{\texttt{LAdj}_{belle}, \texttt{LAdj}_{ferme}\}$.

We complete the example of the previous section with the following constraints[1]*:*

❶ $t \in \texttt{CN} \Rightarrow (\texttt{Det}, \emptyset) \in \mathcal{C}(t)$
❷ $t \in \texttt{LAdj} \Rightarrow (\emptyset, \texttt{CN}) \in \mathcal{C}(t)$
❸ $t \in \texttt{RAdj} \Rightarrow (\texttt{CN}, \emptyset) \in \mathcal{C}(t)$
❹ $t \in \texttt{Det} \Rightarrow (\emptyset, \texttt{CN}) \in \mathcal{C}(t)$ ❺ $t \in \texttt{Det} \Rightarrow (\texttt{TrV}, \texttt{TrV} \cup \texttt{IntrV}) \in \mathcal{C}(t)$
❻ $t \in \texttt{TrV} \Rightarrow (\texttt{Clit}, \texttt{Det}) \in \mathcal{C}(t)$ ❼ $t \in \texttt{TrV} \Rightarrow (\texttt{Det}, \emptyset) \in \mathcal{C}(t)$
❽ $t \in \texttt{IntrV} \Rightarrow (\texttt{Det}, \emptyset) \in \mathcal{C}(t)$
❾ $t \in \texttt{Clit} \Rightarrow (\emptyset, \texttt{TrV}) \in \mathcal{C}(t)$

The two constraints ❹ and ❺ for instance express that every determiner is implied in two dependencies. First, it must find a common noun on its right to build a noun phrase. Second, the noun phrase has to be used in a verbal construction.

Now, let us consider the lexical tagging: $[\texttt{Det}_{la}, \texttt{LAdj}_{belle}, \texttt{TrV}_{ferme}, \texttt{Clit}_{la}, \texttt{CN}_{porte}]$ *and the constraint ❾ (a clitic is waiting for a transitive verb on its right). This constraint is not fulfilled by the tagging so this tagging has no solution.*

3.3 The "Companionship Principle" language

Actually, a lexical tagging is an element of the formal language $\mathcal{G}^*$ and we can consider the following three languages. First, $\mathcal{G}^*$ itself. Second, the set $C \subseteq \mathcal{G}^*$ corresponds to the lexical taggings which can be parsed. The aim of lexical disambiguation is then to exhibit for each sentence $[w_1, \ldots, w_n]$ all the lexical taggings that are within C. Third, the Companionship Principle defines the language P of lexical taggings which verify this Principle. P squeezes between the two latter sets $C \subseteq P \subseteq \mathcal{G}^*$. Remarkably, the language P can be described as a regular language. Since C is presumably not a regular language (at least for natural languages!), P is a better regular approximation than the trivial $\mathcal{G}^*$.

Let us consider one lexical entry t and an atomic constraint $(\mathcal{L}, \mathcal{R}) \in \mathcal{C}(t)$. Then, the set of lexical taggings verifying this constraint can be described as

$$L_{t:(\mathcal{L},\mathcal{R})} = \complement((\complement\mathcal{L})^* t (\complement\mathcal{R})^*)$$

where $\complement$ denoting the complement of a set.

[1] these constraints are relative to our toy grammar and are not linguistically valid in a larger context.

Since P is defined as the lexical taggings verifying all constraints, it is

$$P = \bigcup_{(\mathcal{L},\mathcal{R})\in\mathcal{C}(t)} L_{t:(\mathcal{L},\mathcal{R})}$$

which is a regular expression.

From the Companionship Principle, we derive a lexical disambiguation Principle which simply tests tagging candidates with P. Notice that P can be statically computed (at least, in theory) from the grammar itself.

Example 3 *For instance, for our example grammar, this automaton is given in the figure 1 where c=Clit, n=CN, d=Det, i=IntrV, l=LAdj, r=RAdj and t=TrV.*

A rough approximation of the size of the automaton corresponding to P can be easily computed. Since each automaton $L_{t:(\mathcal{L},\mathcal{R})}$ has 4 states, P has at most 4^m states where m is the number of atomic constraints. For instance, the grammar used in the experiments contains more than one atomic constraint for each lexical entry, and $m > |\mathcal{G}| > 10^6$. Computing P by brute-force is then intractable.

4 Implementation of the Companionship Principle with automata

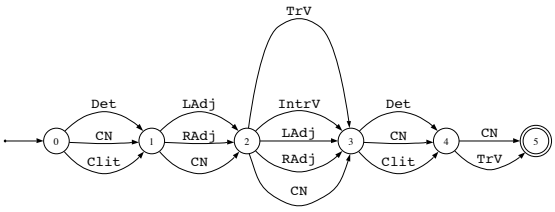
We have seen above that the Companionship Principle applies to lexical taggings. As a matter of fact, in this section we keep the promise we made in the introduction of this paper: we show that it can be computed by means of automata, saving space and time. Actually, we propose two implementations of the Companionship 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 Automaton to represent sets of lexical taggings

The number of lexical taggings to consider for a sentence can be exponential in the length of the sentence. In many cases, an acyclic automaton with elements of $\mathcal{G}$ on the transitions can efficiently represent a large set of lexical taggings: each path of the automaton is interpreted as a lexical tagging. We call such an automaton a **lexical taggings automaton** (LTA).

For instance, with a given sentence $[w_1, \ldots, w_n]$ the number of lexical taggings to consider at the beginning of the parsing process is $\Pi_{1\leq i\leq n}|\ell(w_i)|$, hence the number of taggings grows exponentially with the length of the sentence. This set of taggings can be efficiently represented as the set of paths of the automaton with $n+1$ states $s_0, \ldots, s_n$ and with a transition from s_{i-1} to s_i with the label t for each $t \in \ell(w_i)$. This automaton has $\sum_{1\leq i\leq n}|\ell(w_i)|$ transitions.

Example 4 *With the data of the previous examples, we have the initial automaton:*



To improve readability, only the categories are given on the edges, while the French words can be inferred from the position in the automaton.

4.2 Exact Companionship Principle (ECP)

Suppose we have a LTA $\mathcal{A}$ for a sentence $[w_1, \ldots, w_n]$. For each transition t and for each atomic constraint in $(\mathcal{L}, \mathcal{R}) \in \mathcal{C}(t)$, we construct an automaton $\mathcal{A}_{t,\mathcal{L},\mathcal{R}}$ in the following way.

Each state s of $\mathcal{A}_{t,\mathcal{L},\mathcal{R}}$ is labeled with a triple composed of a state of the automaton $\mathcal{A}$ and two booleans. The intended meaning of the first boolean is to say that each path reaching this state passes through the transition t. The second boolean means that the atomic constraint $(\mathcal{L}, \mathcal{R})$ is necessarily fulfilled.

The initial state is labeled $(s_0, \mathtt{F}, \mathtt{F})$ where s_0 is the initial state of $\mathcal{A}$ and other states are labeled as follows: if $s \xrightarrow{u} s'$ in $\mathcal{A}$ then, in $\mathcal{A}_{t,\mathcal{L},\mathcal{R}}$, we have:

1. $(s, \mathtt{F}, b) \xrightarrow{u} (s', \mathtt{T}, b)$ if $u = t$
2. $(s, \mathtt{F}, b) \xrightarrow{u} (s', \mathtt{F}, \mathtt{T})$ if $u \in \mathcal{L}$
3. $(s, \mathtt{F}, b) \xrightarrow{u} (s', \mathtt{F}, b)$ if $u \notin \mathcal{L}$
4. $(s, \mathtt{T}, b) \xrightarrow{u} (s', \mathtt{T}, \mathtt{T})$ if $u \in \mathcal{R}$
5. $(s, \mathtt{T}, b) \xrightarrow{u} (s', \mathtt{T}, b)$ if $u \notin \mathcal{R}$

where $b \in \{\mathtt{T}, \mathtt{F}\}$. It is then routine to show that, for each state labeled (s, b_1, b_2):

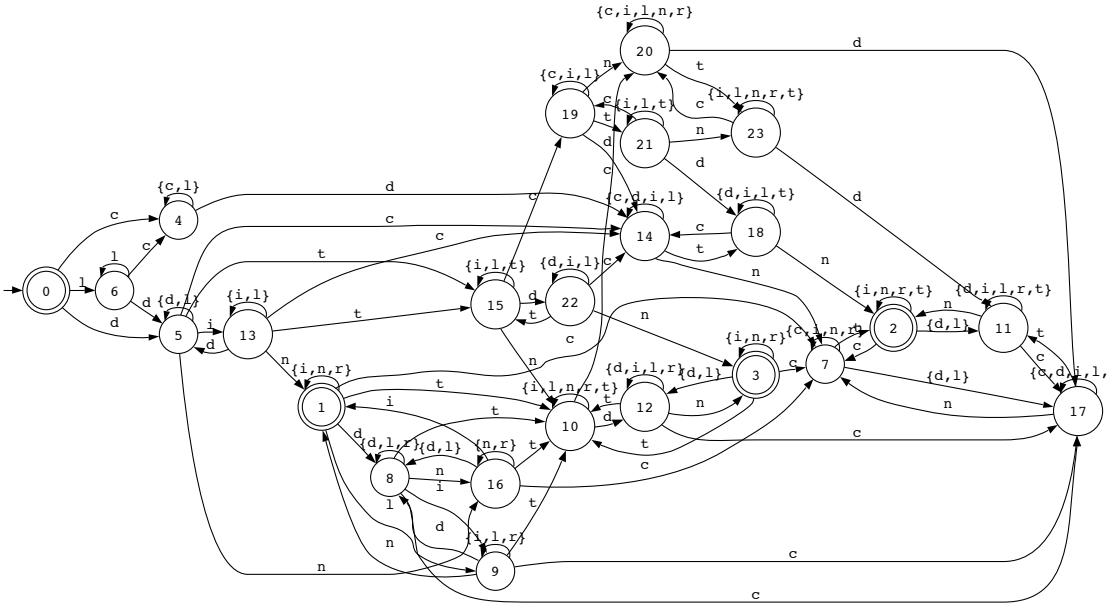


Figure 1: The P language for $\mathcal{G}$

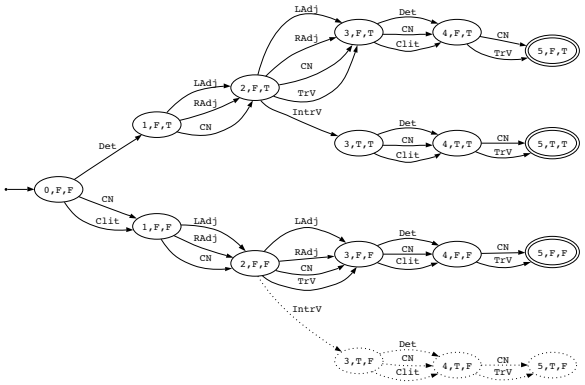
- b_1 is T iff all paths from the initial state to s contain the transition t;
- b_2 is T iff for all paths p reaching this state, either there is some $u \in \mathcal{L}$ or p goes through t and there is some $u \in \mathcal{R}$. In other words, the constraint (whether we go through t or not) is fulfilled.

In conclusion, a path ending with $(s_f, \mathtt{T}, \mathtt{F})$ with s_f a final state of $\mathcal{A}$ is built with transitions 1, 3 and 5 only and hence contains t but no transition able to fulfill the constraint. The final states are:

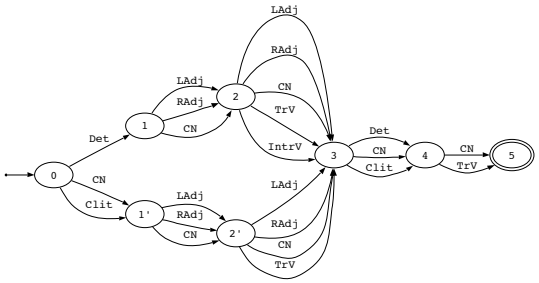
- $(s_f, \mathtt{F}, b)$: each path ending here does not contains the edge t and is not concerned by the constraints, if is kept,
- $(s_f, \mathtt{T}, \mathtt{T})$ each path ending here contains the edge t but it contains also either a transition 2 or 4, so the constraint is fulfilled by there these paths.

The size of these automata is easily bounded by $4n$ where n is the size of A. Using a slightly more intricated presentation, we built automata of size $2n$.

Example 5 *We give below the automaton $\mathcal{A}$ for the atomic constraint ❽ (an intransitive verb is waiting for a determiner on its left):*



The dotted part of the graph corresponds to the part of the automaton that can be safely removed. After minimization, we finally obtain:



This automaton contains 234 paths (36 lexical taggings are removed by this constraint).

For each transition t of the lexical taggings automaton and for each constraint $(\mathcal{L}, \mathcal{R}) \in \mathcal{C}(t)$, we construct the atomic constraint automaton $\mathcal{A}_{t,\mathcal{L},\mathcal{R}}$.

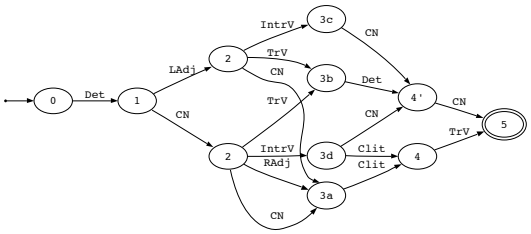
The intersection of these automata represents all the possible lexical taggings of the sentence which respect the Companionship Principle. That is, we output :

$$\mathcal{A}_{CP} = \bigcap_{1 \leq i \leq n,\ t \in \mathcal{A}; (\mathcal{L},\mathcal{R}) \in \mathcal{C}(t)} \mathcal{A}_{t,\mathcal{L},\mathcal{R}}$$

It can be shown that the automaton is the same that the one obtained by intersection with the automaton of the language defined in 3.3:

$$\mathcal{A}_{CP} = \mathcal{A} \cap P.$$

Example 6 *In our example, the intersection of the 9 automata built for the atomic constraints is given below:*



This automaton has 8 paths: there are 8 lexical taggings which fulfill every constraint.

4.3 Approximation: the Quick Companionship Principle (QCP)

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. The idea is to consider at the same time all the paths that contain some transition.

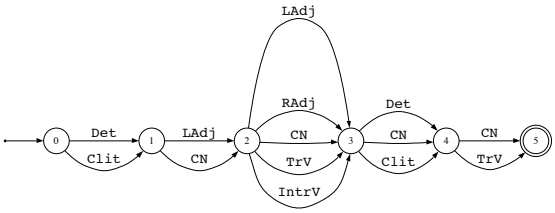
We consider a LTA $\mathcal{A}$. We write $\prec_{\mathcal{A}}$ the precedence relation on transitions in an automaton $\mathcal{A}$. We define $l_{\mathcal{A}}(t) = \{u \in \mathcal{G}, u \prec_{\mathcal{A}} t\}$ and $r_{\mathcal{A}}(t) = \{u \in \mathcal{G}, t \prec_{\mathcal{A}} u\}$.

For each transition $s \xrightarrow{t} s'$ and each constraint $(\mathcal{L}, \mathcal{R}) \in \mathcal{C}(t)$, if $l_{\mathcal{A}}(t) \cap \mathcal{L} = \emptyset$ and $r_{\mathcal{A}}(t) \cap \mathcal{R} = \emptyset$, then none of the lexical taggings which use the transition t has a solution and the transition t can be safely removed from the automaton.

This can be computed by a double-for loop: for each atomic constraint of each transition, verify that either the left context or the right context of the transition contains some structure to solve the constraint. Observe that the cost of this algorithm is $O(n^2)$, where n is the size of the input automaton.

Note that one must iterate this algorithm until a fixpoint is reached. Indeed, removing a transition which serves as a potential companion breaks the verification. Nevertheless, since for each step before the fixpoint is reached, we remove at least one transition, we iterate the double-for at most $O(n)$ times. The complexity of the whole algorithm is then $O(n^3)$. In practice, we have observed that the complexity is close to $O(n^2)$: only 2 or 3 loops are enough to reach the fixpoint.

Example 7 *If we apply the QCP to the automaton of Example 4, in the first step, only the transition* $0 \xrightarrow{\texttt{CN}} 1$ *is removed by applying the atomic constraint* ❶*. In the next step, the transition* $1 \xrightarrow{\texttt{RAdj}} 2$ *is removed by applying the atomic constraint* ❸*. The fixpoint is reached and the output automaton (with 120 paths) is:*



5 The Generalized Companionship Principle

In practice, of course, we have to face the problem of the computation of the constraints. In large coverage grammar, the size of $\mathcal{G}$ is too big to compute all the constraints in advance. However, as we have seen in example 2 we can identify subsets of $\mathcal{G}$ that have the same constraints; the same way, we can use these subsets to give a more concise presentation of the $\mathcal{L}$ and $\mathcal{R}$ sets of the atomic constraints. This motivates us to define a Generalized Principle which is stated on a quotient set of $\mathcal{G}$.

5.1 Generalized atomic constraints

Let $\mathcal{U}$ be a set of subsets of $\mathcal{G}$ that are a partition of $\mathcal{G}$. For $t \in \mathcal{G}$, we write $\overline{t}$ the subset of $\mathcal{U}$ which contains t.

We say that a pair $(\mathcal{L}, \mathcal{R})$ of subsets of $\mathcal{U}$ is a **generalized atomic constraint** for $u \in \mathcal{U}$ if for each lexical tagging $L = [t_1, \ldots, t_n]$ such that $\mathbf{p}(L) \neq \emptyset$ and $u = \overline{t_i}$ for some i then:

- either there is some $j < i$ such that $\overline{t_j} \in \mathcal{L}$,
- or there is some $j > i$ such that $\overline{t_j} \in \mathcal{R}$.

A **system of generalized constraints** for a partition $\mathcal{U}$ of a grammar $\mathcal{G}$ is a function $\mathcal{C}$ which associates a finite set of generalized atomic constraints to each element of $\mathcal{U}$.

5.2 The Generalized Principle

The Generalized Companionship Principle is then an immediate consequence of the previous definition and can be stated as the necessary condition:

The Generalized Companionship Principle
If a lexical tagging $[t_1, \ldots, t_n]$ has a solution then for all i and for all generalized atomic constraints $(\mathcal{L}, \mathcal{R}) \in \mathcal{C}(\overline{t_i})$

- $\{\overline{t_1}, \ldots, \overline{t_{i-1}}\} \cap \mathcal{L} \neq \emptyset$
- or $\{\overline{t_{i+1}}, \ldots, \overline{t_n}\} \cap \mathcal{R} \neq \emptyset$.

Example 8 *The constraints given in example 2 are in fact generalized atomic constraints on the set (recall that we write* `LAdj` *then 2 elements set* $\{\texttt{LAdj}_{belle}, \texttt{LAdj}_{ferme}\}$*):*

$$\mathcal{U} = \{\texttt{Det}, \texttt{LAdj}, \texttt{RAdj}, \texttt{CN}, \texttt{Clit}, \texttt{TrV}, \texttt{IntrV}\}.$$

Then the constraints are expressed on $|\mathcal{U}| = 7$ *elements and not on* $|\mathcal{G}| = 13$.

A generalized atomic constraint on $\mathcal{U}$ can, of course, be expressed as a set of atomic constraints on $\mathcal{G}$: let $u \in \mathcal{U}$ and $t \in \mathcal{G}$ such that $\overline{t} = u$

$$(\mathcal{L}, \mathcal{R}) \in \mathcal{C}(u) \Longrightarrow \left(\bigcup_{L \in \mathcal{L}} L, \bigcup_{R \in \mathcal{R}} R \right) \in \mathcal{C}(t).$$

5.3 Implementation of lexicalized grammars

In implementations of large coverage linguistic resources, it is very common to have, first, the description of the set of "different" structures needed to describe the modeled natural language and then an anchoring mechanism that explains how words of the lexicon are linked to these structures. We call **unanchored grammar** the set $\mathcal{U}$ of different structures (not yet related to words) that are needed to describe the grammar. In this context, the lexicon is split in two parts:

- a function $\overline{\ell}$ from $\mathcal{V}$ to subsets of $\mathcal{U}$,
- an anchoring function α which builds the grammar elements from a word $w \in \mathcal{V}$ and an unanchored structure $u \in \overline{\ell}(w)$; we suppose that α verifies that $\mathbf{anc}(\alpha(w, u)) = w$.

In applications, we suppose that $\mathcal{U}$, $\overline{\ell}$ and α are given. In this context, we define the grammar as the codomain of the anchoring function:

$$\mathcal{G} = \bigcup_{w \in \mathcal{V}, u \in \overline{\ell}(w)} \alpha(w, u)$$

Now, we can define generalized constraints on the unanchored grammar, which are independent of the lexicon and can be computed statically for a given unanchored grammar.

6 Application to Interaction Grammars

In this section, we apply the Companionship Principle to the Interaction Grammars formalism. We first give a short and simplified description of IG and an example to illustrate them at work; we refer the reader to (GP08) for a complete and detailed presentation.

6.1 Interaction Grammars

We illustrate some of the important features on the French sentence (2). In this sentence, *"la"* is an object clitic pronoun which is placed before the verb whereas the canonical place for the (non-clitic) object is on the right of the verb.

(2) *"Jean la demande."* [John asks for it]

The set $\mathcal{F}$ of final structures, used as output of the parsing process, contains ordered trees called **parse trees** (PT). An example of a PT for the sentence (2) is given in Figure 2. 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).

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

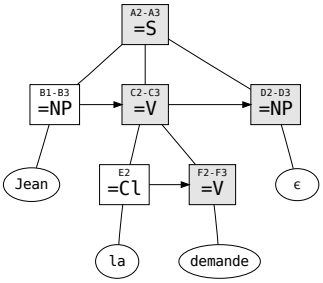


Figure 2: The PT of sentence (2)

relations between these nodes. The three PTDs used to build the model above are given in Figure 3. The relations used in the PTDs are: immediate dominance (lines) and immediate sisterhood (arrows). Nodes represent syntactic constituents and relations express structural dependencies between these constituents.

Moreover, nodes carry a polarity: the set of polarities is $\{+, -, =, \sim\}$. A $+$ (resp.$-$) polarity represents an available (resp. needed) resource, a $\sim$ polarity describes a node which is unsaturated. Each $+$ must be associated to exactly one $-$ (and vice versa) and each $\sim$ must be associated to at least another polarity.

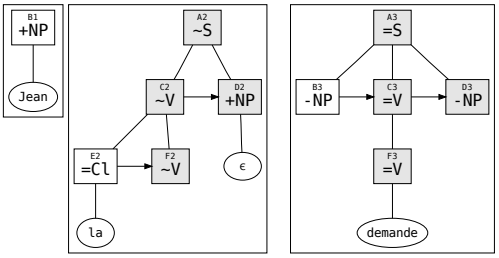


Figure 3: PTDs for the sentence (2)

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 4 is $=$;
- the feature structure of a node N in the PT is the unification of the feature structures of the nodes in $\mathcal{I}^{-1}(N)$.

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 (whereas 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.

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

Figure 4: Polarity composition

In order to give a idea of the full IG system, we briefly give here the main differences between our presentation and the full system.

- Dominance relations 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. This is mainly used to model unbounded extraction.
- Sisterhood relations can also be underspecified: when the order on subconstituents is not total, it can be modeled without using several PTDs.
- Polarities are attached to features rather than nodes: it sometimes gives more freedom to the grammar writer when the same constituent plays a role in different constructions.
- Feature values can be shared between several nodes: once again, this is a way to factorize the unanchored grammar.

The application of the Companionship Principle is described on the reduced IG but it can be straightforwardly extended to full IG with unessential technical details.

Following notation of 5.3, an IG is made of:

- A finite set $\mathcal{V}$ of words;
- A finite set $\mathcal{U}$ of unanchored PTDs (without any word attached to them);
- A lexicon function $\overline{\ell}$ from $\mathcal{V}$ to subsets of $\mathcal{U}$.

When $t \in \overline{\ell}(w)$, we can construct the anchored PTD $\alpha(w, u)$. Technically, in each unanchored PTD u, a place is marked to be the anchor, i.e. to be replaced by the word during the anchoring process. Moreover, the anchoring process can also be used to refine some features. The fact that the feature can be refined gives more flexibility and more compactness to the unanchored grammar construction. In the French IG grammar, the same unanchored PTD can be used for masculine or feminine common nouns and the gender is specified during the anchoring to produce distinct anchored PTDs for masculine and feminine nouns. $\mathcal{G}$ is defined by:

$$\mathcal{G} = \bigcup_{w \in \mathcal{V}, u \in \overline{\ell}(w)} \alpha(w, u)$$

The parsing solutions of a lexical tagging is the set of PTs that are models of the list of PTDs described by the lexical tagging:

$$\mathbf{p}(L) = \{t \in \mathcal{F} \mid t \text{ is a model of } L\}$$

With the definitions of this section, an IG is a special case of AGF as defined in section 2.

6.2 Companionship Principle for IG

In order to apply the Companionship Principle, we have to explain how the generalized atomic constraints are built for a given grammar. One way is to look at dependency structures but in IG polarities are build in and then we can read the dependency information we need directly on polarities. A requirement to build a model is the saturation of all the polarities. This requirement can be expressed using atomic constraints. Each time a PTD contains an unsaturated polarity $+$, $-$ or $\sim$, we have to find some other compatible dual polarity somewhere else in the grammar to saturate it.

From the general MTS definition of IG above, we can define a step by step process to build models of a lexical tagging. The idea is to build incrementally the interpretation function $\mathcal{I}$ with the atomic operation of **node merging**. In this atomic operation, we choose two nodes and make the hypothesis that they have the same image through $\mathcal{I}$ and hence that they can be identified.

Now, suppose that the unanchored PTD u contains some unsaturated polarity p. We can use the atomic operation of node merging to test if the unanchored PTD u' can be used to saturate the polarity p. Let $\mathcal{L}$ (resp $\mathcal{R}$) be the set of PTDs that can be used on the left (resp. on the right) of u to saturate p, then $(\mathcal{L}, \mathcal{R})$ is a generalized atomic constraint in $\mathcal{C}(u)$.

7 Companionship Principle for other formalisms

As we said in the introduction, many current grammatical formalisms can more or less directly be used to generate dependency structures and hence are candidate for disambiguation with the Companionship Principle. With IG, we have seen that dependencies are strongly related to polarities and dependency constraints in IG are built with the polarity system.

We give below two short examples of polarity use to define atomic constraints on TAG and on CG. We use, as for IG, the polarity view of dependencies to describe how the constraints are build.

7.1 Tree Adjoining Grammars

Feature-based Tree Adjoining Grammars (hereafter FTAG) (Jos87) are a unification based version of Tree Adjoining Grammars. An FTAG consists of a set of elementary trees and of two tree composition operations: substitution and adjunction. There are two kinds of trees: auxiliary and initial. Substitution inserts a tree t with root r onto a leaf node l of another tree t' under the condition that l is marked as a place for substitution and l and r have compatible feature structures. Adjunction inserts an auxiliary tree t into a tree t' by splitting a node n of t' under the condition that the feature structures of the root and foot nodes of t are compatible with the *t*op and *b*ottom ones of n.

Getting the generalized atomic constraints and the model building procedure for lexical tagging is extremely similar to what was previously described for IG if we extend the polarization procedure which was described in (GK05) to do polarity based filtering in FTAG. The idea is that for each initial tree t, its root of category C is marked as having the polarity $+C$, and its substitution nodes of category S are marked as having the polarity

$-S$. A first constraint set contains trees t' whose root is polarized $+S$ and such that feature structures are unifiable. A second constraint sets contains trees t'' which have a leaf that is polarized $-C$. We can extend this procedure to auxiliary trees: each auxiliary tree t of category A needs to be inserted in a node of category A of another tree t'. This gives us a constraint in the spirit of the $\sim$ polarity in IG: $\mathcal{C}(t)$ contains all the trees t' in which t could be inserted[2].

7.2 Categorial Grammars

In their type system, Categorial Grammars encode linearity constraints and dependencies between constituents. For example, a transitive verb is typed $NP\backslash S/NP$, meaning that it waits for a subject NP on its left and an object NP on its right. This type can be straightforwardly decomposed as two $-NP$ and one $+S$ polarities. Then again, getting the generalized atomic constraints is immediate and in the same spirit as what was described for IG.

8 Experimental results

The experiments are done with a IG French grammar and a set of sentences taken from the newspaper *Le Monde*.

The French grammar we consider (Per07) contains $|\mathcal{U}| = 2\,088$ unanchored trees. It covers 88% of the grammatical sentences and rejects 85% of the ungrammatical ones on the TSNLP (LORP$^+$96) corpus.

The constraints have been computed on the unanchored grammar as explained in section 5: each tree contains several polarities and therefore several atomic constraints. For the whole grammar, there is a total of 20 627 atomic constraints. It takes 2 days to compute the set of constraints and the results can be stored in a constraints file of 10MB. Of course, an atomic constraint is more interesting when the sizes of $\mathcal{L}$ and $\mathcal{R}$ are small. In our grammar, 50% of the constraints set (either $\mathcal{R}$ or $\mathcal{L}$) contain at most 40 elements and 80% of these sets contain at most 200 elements over 2 088.

8.1 The QCP method

The QCP method (section 4) was applied to 68 500 sentences of various length (figure 5). The mean time for a given length is reported in figure 6: the time is almost linear and below 0.2 second even for long sentences.

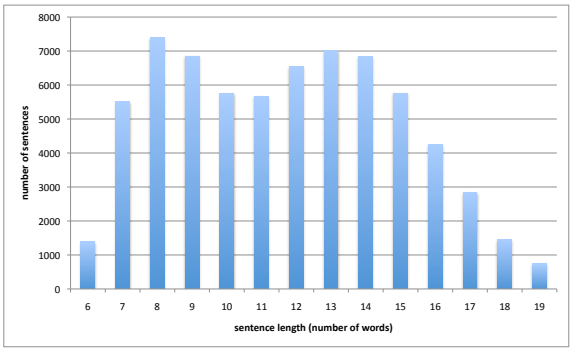


Figure 5: number of sentence of each length

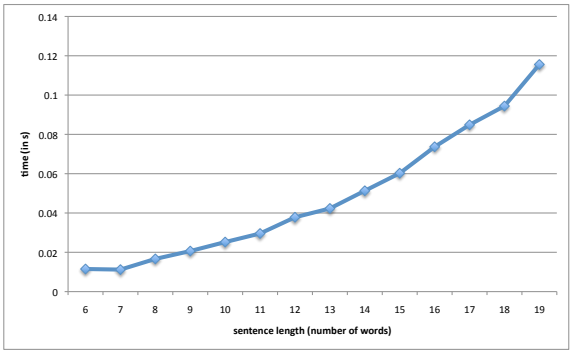


Figure 6: mean execution time (in s)

As we have observed above, the number of lexical taggings is exponential in the length of the sentence. As the number n of lexical taggings is a priori exponential in the sentence length, we will consider the $\log$. Moreover, we use a raw corpus, some sentences can be considered as agrammatical by the grammar; in this case it may happen that the disambiguation method removes all taggings. This is the reason why we will consider $\log_{10}(1+n)$ to avoid undefined values when $n = 0$.

In figure 7, for each sentence length, we give the mean value of $\log_{10}(1+n)$ where n is:

- the initial number of lexical taggings, in the upper curve with squares;
- the number of lexical taggings after the QCP, in the lower curve with diamonds.

We can then observe that the two curves are linear and that the QCP has a significant impact. For instance, for sentences of length 14, the mean log value goes from more than 11 down to less than 7: the number of path is divided by 10 000 over 10^{11}.

[2]Note that in the adjunction case, the constraint is not oriented and then $\mathcal{L} = \mathcal{R}$

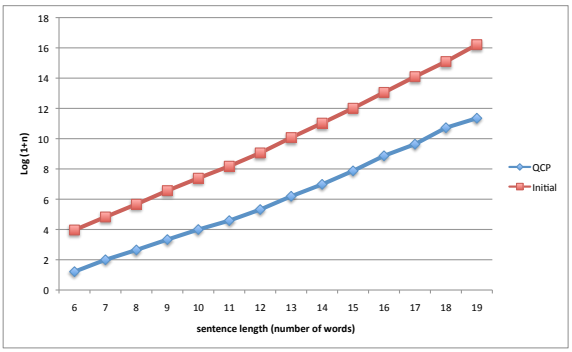


Figure 7: number of taggings (initial and after QCP)

8.2 The ECP method

As expected, the ECP method is more time consuming and for some sentences the time and/or memory required is problematic. To be able to apply the ECP to a large number of sentences, we have used it after another filtering method based on polarities and described in (BGP04).

In our experiment, 31 000 sentences were used. For each sentence, we have computed 3 different filters, each one being finer than the previous one:

- **QCP** the Quick Companionship Principle (like in the previous subsection)
- **QCP+POL** QCP followed by a filtering technique based on polarity counting
- **QCP+POL+ECP** the Exact Companionship Principle applied to the previous filter

We give in figure 8 the number of sentences of each length in the corpus we consider.

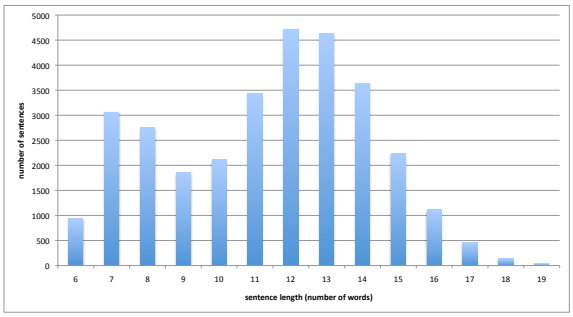


Figure 8: number of sentences of each length

In figure 9, we report the mean computation time for each length: it confirms that the ECP is more time consuming and goes up to 5s for our long sentences.

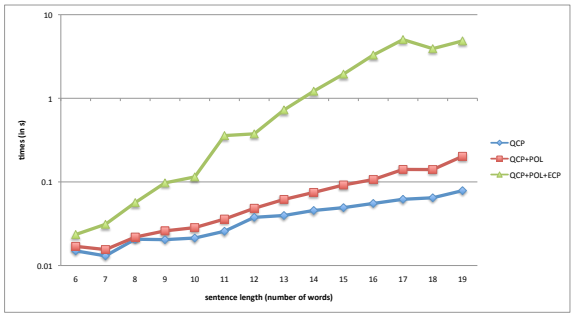


Figure 9: mean execution time (in s)

Finally, as before, we report the number of lexical taggings that each method returns. In figure 10, we give the mean value of $\log_{10}(1+n)$ where n is either the initial number of lexical taggings or the number of lexical taggings returned by our methods.

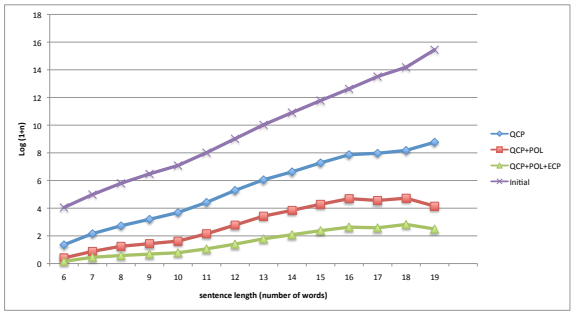


Figure 10: number of taggings (initial and after the 3 disambiguation methods)

We can observe that the slope of the lines corresponds to the mean word ambiguity: if the mean ambiguity is a then the number of taggings for a sentence of length n is about a^n and then $log(a^n) = n \cdot log(a)$. As a consequence, the mean ambiguity can be read as 10^s where s is the slope in the last figure. Computing the mean ambiguity (for sentence of length 17) we get 6.2 for the raw data and 1.4 after the filtering.

9 Conclusion

We have presented a disambiguation method based on dependency constraints which allows to filter out many wrong lexical taggings before entering the deep parsing. As this method relies on the computation of static constraints on the linguistic data and not on a statistical model, we can be sure that we will never remove any correct lex-

ical tagging. Moreover, we manage to apply our methods to an interesting set of data and prove that it is efficient for a large coverage grammar and not only for a toy grammar.

These results are also an encouragement to develop further this kind of disambiguation methods. In the near future, we would like to explore some improvements.

First, we have seen that our principle cannot be computed on the whole grammar and that in implementation, we consider unanchored structures. We would like to explore the possibility to compute on the fly finer constraints (relative to the full grammar) for each sentence. We believe that this can eliminate some more taggings before entering the deep parsing.

Concerning the ECP, as we have seen, there is a kind of interplay between the efficiency of the filtering and the time of the computation. We would like to explore the possibility to define some intermediate way between QCP and ECP either by using approximate automata or using the ECP but only on a subset of elements where it is known to be efficient.

Another challenging method we would like to investigate is to use the Companionship Principle not only as a disambiguation method but as a guide for the deep parsing. Actually, we have observed for at least 20% of the words that dependencies are completely determined by the filtering methods. If deep parsing can be adapted to use this observation (this is the case for IG), this can be of great help.

Finally, we can improve the filtering using both worlds: the Companionship Principle and the polarity counting method. Two different constraints cannot be fulfilled by the same potential companion: this may allow to discover some more lexical taggings that can be safely removed.

References

G. Bonfante, B. Guillaume, and G. Perrier. Polarization and abstraction of grammatical formalisms as methods for lexical disambiguation. In *CoLing 2004*, pages 303–309, Genève, Switzerland, 2004.

P. Boullier. Supertagging : A non-statistical parsing-based approach. In *Pro- ceedings of the 8th International Workshop on Parsing Technologies (IWPT 03)*, pages 55–65, Nancy, France, 2003.

S. Clark, J. Hockenmaier, and M. Steedman. Building Deep Dependency Structures with a Wide-Coverage CCG Parser. In *Proceedings of ACL'02*, pages 327–334, Philadephia, PA, 2002.

Ph. de Groote. Towards abstract categorial grammars. In *Association for Computational Linguistics, 39th Annual Meeting and 10th Conference of the European Chapter, Proceedings of the Conference*, pages 148–155, 2001.

C. Gardent and E. Kow. Generating and selecting grammatical paraphrases. *Proceedings of the ENLG*, Aug 2005.

B. Guillaume and G. Perrier. Interaction Grammars. Research Report RR-6621, INRIA, 2008.

A. Joshi. An Introduction to Tree Adjoining Grammars. *Mathematics of Language*, 1987.

A. Joshi and O. Rambow. A Formalism for Dependency Grammar Based on Tree Adjoining Grammar. In *Proceedings of the Conference on Meaning-Text Theory*, 2003.

S. Kahane. Polarized unification grammar. In *Proceedings of Coling-ACL'02*, Sydney, 2006.

A. Koller and M. Kuhlmann. Dependency trees and the strong generative capacity of ccg. In *EACL' 2009*, Athens, Greece, 2009.

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

J. Lambek. The mathematics of sentence structure. *American mathematical monthly*, pages 154–170, 1958.

F. Lamarche. Proof Nets for Intuitionistic Linear Logic: Essential Nets. Technical report, INRIA, 2008.

S. Lehmann, S. Oepen, S. Regnier-Prost, K. Netter, V. Lux, J. Klein, K. Falkedal, F. Fouvry, D. Estival, E. Dauphin, H. Compagnion, J. Baur, L. Balkan, and D. Arnold. Tsnlp: Test suites for natural language processing. In *Proceedings of the 16th conference on Computational linguistics*, pages 711–716, 1996.

B. Merialdo. Tagging English Text with a Probabilistic Model. *Computational linguistics*, 20:155–157, 1994.

J. Marchand, B. Guillaume, and G. Perrier. Analyse en dépendances à l'aide des grammaires d'interaction. In *Actes de TALN 09*, Senlis, France, 2009.

M. Moortgat and G. Morrill. Heads and phrases. type calculus for dependency and constituent structure. In *Journal of Language, Logic and Information*, 1991.

G. Perrier. A French Interaction Grammar. In *RANLP 2007*, pages 463–467, Borovets Bulgarie, 2007.

M. Steedman. *The Syntactic Process*. MIT Press, 2000.

L. Tesnière. *Éléments de syntaxe structurale*. Klinksieck, 1959.