Polarization and abstraction of grammatical formalisms as methods for lexical disambiguation

Guillaume Bonfante, Bruno Guillaume, Guy Perrier

Polarization and abstraction of grammatical formalisms as methods for lexical disambiguation

Guillaume Bonfante, Bruno Guillaume and Guy Perrier

Abstract

In the context of lexicalized grammars, we propose general methods for lexical disambiguation based on polarization and abstraction of grammatical formalisms. Polarization makes their resource sensitivity explicit and abstraction aims at keeping essentially the mechanism of neutralization between polarities. If a grammar in the initial formalism happens to be lexicalized, parsing with the simplified grammar in the abstract formalism can be used efficiently for filtering lexical selections.

Introduction

There is a complexity issue if one consider exact parsing with large scale lexicalized grammars. Indeed, the number of way of associating to each word of a sentence a corresponding elementary structure—a tagging of the sentence—is the product of the number of lexical entries for each word. If such a selection is necessary in the parsing process, the procedure may have an exponential complexity in the length of the sentence.

In order to filter taggings, we can use probabilistic methods (Joshi and Srinivas, 1994) and keep only the most probable ones; but if we want to keep all successful taggings, we must use exact methods. Among these, one consists in abstracting information that is relevant for the filtering process, from the formalism F used for representing the concerned grammar G. In this way, we obtain a new formalism F_{abs} which is a simplification of F and the grammar G is translated into a grammar $abs(G)$ in the abstract framework F_{abs}. From this, disambiguating with G consists in parsing with $abs(G)$. The abstraction is relevant if parsing eliminates a maximum of bad taggings at a minimal cost. (Boullier, 2003) uses such a method for Lexicalized Tree Adjoining Grammars (LTAG) by abstracting a tree adjoining grammar into a context free grammar and further abstracting that one into a regular grammar. We also propose to apply abstraction but after a preprocessing polarization step.

The notion of polarity comes from Categorial Grammars (Moortgat, 1996) which ground syntactic composition on the resource sensitivity of natural languages and it is highlighted in Interaction Grammars (Perrier, 2003), which result from refining and making Categorial Grammars more flexible.

Polarization of a grammatical formalism F consists in adding polarities to its syntactic structures to obtain a polarized formalism F_{pol} in which neutralization of polarities is used for controlling syntactic composition. In this way, the resource sensitivity of syntactic composition is made explicit. (Kahane, 2004) shows that many grammatical formalisms can be polarized by generalizing the system of polarities used in Interaction Grammars.

To abstract a grammatical formalism, it is interesting to polarize it before because polarities allow original methods of abstraction.

The validity of our method is based on a concept of *morphism* (two instances of which being polarization and abstraction) which characterizes how one should transport a formalism into another.

In sections 1 and 2, we present the conceptual tools of grammatical formalism and morphism which will be used in the following.

In section 3, we define the operation of polarizing grammatical formalisms and in section 4, we describe how polarization is used then for abstracting these formalisms.

In section 5, we show how abstraction of grammatical formalisms grounds methods of lexical disambiguation, which reduce to parsing in simplified formalisms. We illustrate our purpose with an incremental and a bottom-up method.

In section 6, we present some experimental results which illustrate the flexibility of the ap-

proach.

1 Characterization of a grammatical formalism

Taking a slightly modified characterization of polarized unification grammars introduced by (Kahane, 2004) we define a grammatical formalism F (not necessarily polarized) as a quadruple $\langle \mathbf{Struct}_F, \mathbf{Sat}_F, \mathbf{Phon}_F, \mathbf{Rules}_F \rangle$:

1. $\mathbf{Struct}_F$ is a set of syntactic structures which are graphs[1] in which each edge and vertex may be associated with a label representing morpho-syntactic information; we assume that the set of labels associated with F is equipped with subsumption, a partial order denoted $\sqsubseteq$, and with unification, an operation denoted $\sqcup$, such that, for any labels l and l', either $l \sqcup l'$ is not defined, which is denoted $l \sqcup l' = \bot$, or $l \sqcup l'$ is the least upper bound of l and l'[2];
2. $\mathbf{Sat}_F$ is a subset of $\mathbf{Struct}_F$, which represents the saturated syntactic structures of grammatical sentences;
3. $\mathbf{Phon}_F$ is a function that projects every element of $\mathbf{Sat}_F$ in the sentence that has this element as its syntactic structure.
4. $\mathbf{Rules}_F$ is a set of composition rules between syntactic structures. Every element of $\mathbf{Rules}_F$ is a specific method for superposing parts of syntactic structures; this method defines the characteristics of the parts to be superposed and the unification operation between their labels. Notice that we do not ask rules to be deterministic.

The composition rules of syntactic structures, viewed as superposition rules, have the fundamental property of monotonicity: they add information without removing it. Hence, the definition above applies only to formalisms that can be expressed as constraint systems in opposition to transformational systems.

Let us give some examples of grammatical formalisms that comply with the definition above by examining how they do it.

- In LTAG, $\mathbf{Struct}_{\text{LTAG}}$ represents the set of derived trees, $\mathbf{Sat}_{\text{LTAG}}$ the set of derived trees with a root in the category *sentence* and without non terminal leaves. The projection $\mathbf{Phon}_{\text{LTAG}}$ is the canonical projection of a locally ordered tree on its leaves. Finally, $\mathbf{Rules}_{\text{LTAG}}$ is made up of two rules: *substitution* and *adjunction*. To view adjunction as a superposition rule, we resort to the monotone presentation of LTAG with quasi-trees introduced by (Vijay-Shanker, 1992).
- In Lambek Grammars (LG), $\mathbf{Struct}_{\text{LG}}$ is the set of partial proofs and these proofs can be represented in the form of incomplete Lambek proof nets labelled with phonological terms (de Groote, 1999). $\mathbf{Sat}_{\text{LG}}$ represents the set of complete proof nets with the category *sentence* as their conclusion and with syntactic categories labelled with words as their hypotheses. The projection $\mathbf{Phon}_{\text{LG}}$ returns the label of the conclusion of complete proof nets. $\mathbf{Rules}_{\text{LG}}$ is made up of two rules: a binary rule that consists in identifying two dual atomic formulas of two partial proof nets by means of an axiom link and a unary rule that consists in the same operation but inside the same partial proof net.

Now, inside a formalism defined as above, we can consider particular grammars:

> A grammar G of a formalism F is a subset $G \subseteq \mathbf{Struct}_F$ of its elementary syntactic structures.

A grammar is lexicalized if every element of G is anchored by a word in a lexicon. In LTAG, G is constituted of its initial and auxiliary trees. In LG, G is constituted of the syntactic trees of the formulas representing syntactic categories of words as hypotheses plus a partial proof net anchored by the period and including a conclusion in the category *sentence*.

From a grammar G defined in a formalism F, we build the set $\mathcal{D}(G)$ of its derived syntactic structures by applying the rules of $\mathbf{Rules}_F$ recursively from the elements of G. The language generated by the grammar is the projection $\mathcal{L}(G) = \mathbf{Phon}_F(\mathbf{Sat}_F \cap \mathcal{D}(G))$.

2 Morphisms between grammatical formalisms

Polarization and abstraction can be defined from a more general notion of morphism between grammatical formalisms. A morphism

[1] Usually trees or directed acyclic graphs.

[2] The least upper bound of l and l' can exist and, at the same time, $l \sqcup l'$ be not defined; if the operation of unification is defined everywhere, the set of labels is a semi-lattice.

from a grammatical formalism C to a grammatical formalism A is a function f from $\mathbf{Struct}_C$ to $\mathbf{Struct}_A$ with the following properties[3]:

(i) $f(\mathbf{Sat}_C) \subseteq \mathbf{Sat}_A$;

(ii) $\forall S \in \mathbf{Sat}_C, \mathbf{Phon}_A(f(S)) = \mathbf{Phon}_C(S)$;

(iii) if $S_1, \ldots, S_n$ are composed into a structure S in C by means of rules of $\mathbf{Rules}_C$, then $f(S_1), \ldots, f(S_n)$ can be composed into the structure $f(S)$ by means of rules of $\mathbf{Rules}_A$.

Given such a morphism f and a grammar G in C, the image of G by f denoted $f(G)$ is the grammar—in A—induced by the morphism. The three properties of morphism guarantee that the language generated by any grammar G of C is a subset of the language generated by $f(G)$. In other words, $\mathcal{L}(G) \subseteq \mathcal{L}(f(G))$.

We propose to use the notion of morphism in two ways:

- for polarizing grammatical formalisms and in this case, morphisms are isomorphisms; grammars are transposed from a formalism to another formalism with the same generative power; in other words, with the previous notations: $\mathcal{L}(G) = \mathcal{L}(f(G))$;
- for abstracting grammatical formalisms and this case, the transposition of grammars by morphisms entails simplification of grammars and extension of the generated languages; we have only: $\mathcal{L}(G) \subseteq \mathcal{L}(f(G))$.

An example of the use of abstraction for lexical disambiguation may be found in (Boullier, 2003)[4]. We propose to link polarization with abstraction because polarities allow original methods of abstraction. Polarization is used as a preprocessing step before the application of these methods.

3 Polarization of grammatical formalisms

The goal of polarizing a grammatical formalism is to make explicit the resource sensitivity that is hidden in syntactic composition, by adding polarities to the labels of its structures. When morpho-syntactic labels become polarized in syntactic structures, they get the status of consumable resources: a label that is associated with the polarity $+$ becomes an available resource whereas a label that is associated with the polarity $-$ becomes an expected resource; both combine for producing a saturated resource associated with the polarity $\mathring{=}$; labels associated with the polarity $=$ are neutral in this process. In a polarized formalism, the saturated structures are those that have all labels associated with the polarity $=$ or $\mathring{=}$. We call them neutral structures. The composition of structures is guided by a principle of neutralization: every positive (negative) label must unify with a negative (positive) label.

The polarization of a formalism must preserve its generative power: the language that is generated by a polarized grammar must be the same as that generated by the initial non-polarized grammar. This property of (weak and even strong) equivalence is guaranteed if the polarized formalism is isomorphic to the non-polarized formalism from which it stems. Formally, given a grammatical formalism F, any formalism F_{pol} with a morphism $pol : F \rightarrow F_{pol}$ is a polarization of F if:

(i) For any structure $S \in \mathbf{Struct}_F$, $pol(S)$ results from associating each label of S with one of the polarities: $+$, $-$, $=$, $\mathring{=}$; in others words, labels of F_{pol} are pairs (p, l) with p a polarity and l a label of F. The set of polarities $\{+, -, =, \mathring{=}\}$ is equipped with the operation of unification and the subsumption order defined by Figure 1. The operations of subsumption and unification on pairs are the pointwise operations. That is, for any pairs (p, l) and (p', l'),

$$(p, l) \sqsubseteq (p', l') \text{ iff } p \sqsubseteq p' \text{ and } l \sqsubseteq l'$$
$$(p, l) \sqcup (p', l') = (p \sqcup p', l \sqcup l')$$

(ii) $\mathbf{Sat}_{F_{pol}}$ is constituted of the neutral structures of $\mathbf{Struct}_{F_{pol}}$.

(iii) pol is an isomorphism whose inverse morphism is the function that ignores polarities and keeps invariant the rest of the structure.

Let us illustrate our purpose by taking again our two examples of formalisms.

- For LTAG (see figure 2), pol consists in labelling the root of elementary syntactic trees with the polarity $+$ and their non terminal leaves (substitution and foot nodes)

[3] An elegant definition of morphism could be given in a category-theoretical framework but we have chosen here a more elementary definition.

[4] Our definition of morphism must be slightly extended for embedding the proposal of (Boullier, 2003).

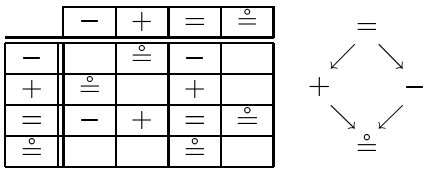

Figure 1: unification and subsumption between polarities

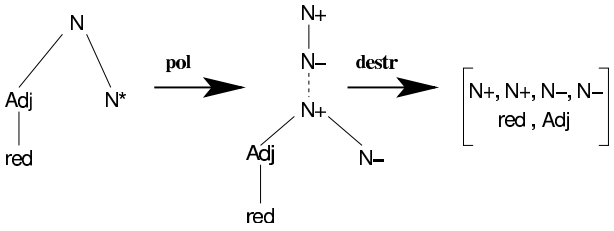

Figure 2: Syntactic structures associated with the adjective *red* in LTAG, LTAG_{pol}, $(\text{LTAG}_{pol})_{destr}$

with the polarity −. In every pair of quasi-nodes, the top quasi-node is labelled with the polarity − and the bottom quasi-node is labelled with the polarity +. With respect to the classical presentation of LTAG, initial trees must be completed by an axiom with two nodes of the type *sentence*: a root with the polarity = and its unique daughter with the polarity −. In this way, *pol* establishes a perfect bijection between the saturated structures of LTAG and the neutral structures of LTAG_{pol}. The rules of adjunction and substitution of $\textbf{Rules}_{\text{LTAG}_{pol}}$ mimic the corresponding rules in LTAG, taking into account polarities. We add a third composition rule, a unary rule which identifies the two quasi-nodes of a same pair. It is routine to check that *pol* is a polarisation.

- In LG(see figure 3), polarization is already present explicitly in the formalism: negative formulas and sub-formulas are input formulas, hypotheses whereas positive formulas and sub-formulas are output formulas, conclusions.

4 Abstraction of polarized grammatical formalisms

The originality of abstracting polarized formalisms is to keep a mechanism of neutralization between opposite polarities at the heart of the abstract formalism. Furthermore, we can choose different levels of abstraction by keeping more or less information from the initial formalism.

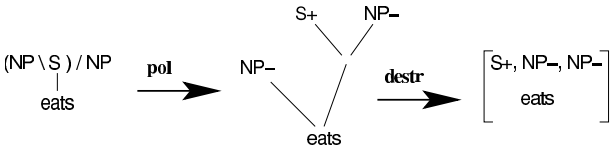

Figure 3: Syntactic structures associated with the transitive verb *eats* in LG, LG_{pol}, $(\text{LG}_{pol})_{destr}$

As an example, we propose a high degree abstraction, *destructuring*. Destructuring a polarized formalism consists in ignoring the structure from the initial syntactic objects to keep merely the multisets of polarized labels. Formally, given a polarized formalism P, we define the formalism P_{destr} as follows:

- Any element M of $\textbf{Struct}_{P_{destr}}$ is a multiset of labels. All elements of M are labels of P, except one exactly, the anchor, which is a neutral string.
- $\textbf{Sat}_{P_{destr}}$ is made up of multisets containing only neutral and saturated labels;
- The projection $\textbf{Phon}_{P_{destr}}$ returns the label of the anchor.
- $\textbf{Rules}_{P_{destr}}$ has two neutralization rules. A binary rule takes two multisets M_1 and M_2 from $\textbf{Struct}_{P_{destr}}$ as inputs; two unifiable labels $+l_1 \in M_1(M_2)$ and $-l_2 \in M_2(M_1)$ are selected. The rule returns the union of M_1 and M_2 in which $+l_1$ and $-l_2$ are unified and the two anchors are concatenated. The only change with the unary rule is that this operates inside the same multiset.

A morphism *destr* is associated to P_{destr} (see figure 2 and 3): it takes any structure S from $\textbf{Struct}_P$ as input and returns the multiset of its labels with an additionnal anchor. This anchor is the neutral string $\textbf{Phon}_P(S)$ if this one is defined.

An important property of P_{destr} is that it is not sensitive to word order: if a sentence is generated by a particular grammar of P_{destr}, by permuting the words of the sentence, we obtain another sentence generated by the grammar. Destructuring is an abstraction that applies to any polarized formalism but we can design abstractions with lower degree which are specific to particular formalisms (see Section 6).

5 Application to lexical disambiguation

Abstraction is the basis for a general method of lexical disambiguation. Given a lexicalized grammar G in a concrete formalism C, we consider a sentence $w_1 \dots w_n$. For each $1 \leq i \leq n$, let the word w_i have the following entries in the lexicon of G: $S_{i,1}, S_{i,2} \dots S_{i,m_i}$. A tagging of the sentence is a sequence $S_{1,k_1}, S_{2,k_2} \dots S_{n,k_n}$. We suppose now that we have given an abstraction morphism $abs : C \to C_{abs}$. As $\mathcal{L}(G) \subseteq \mathcal{L}(abs(G))$, any tagging in $abs(G)$ which has no solutions comes from a bad tagging in G. As a consequence, the methods we develop try to eliminate such bad taggings by parsing the sentence $w_1 w_2 \dots w_n$ within the grammar $abs(G)$.

We propose two procedures for parsing in the abstract formalism:

- an incremental procedure which is specific to the destructuring abstraction,
- a bottom-up procedure which can apply to various formalisms and abstractions.

5.1 Incremental procedure

We choose polarization followed by destructuring as abstraction. In other words: $abs = destr \circ pol$. Let us start with the particular case where unification of labels in C reduces to identity. In this case, parsing inside the formalism C_{abs} is greatly simplified because composition rules reduce to the neutralization of two labels $+l$ and $-l$. As a consequence, parsing reduces to a counting of positive and negative polarities present in the selected tagging for every label l: every positive label counts for +1 and every negative label for –1, the sum must be 0; since this counting must be done for every possible tagging and for every possible label, it is crucial to factorize counting. For this, we use automata, which drastically decrease the space (and also the time) complexity.

For every label l of C that appears with a polarity + or − in the possible taggings of the sentence $w_1 w_2 \dots w_n$, we build the automaton A_l as follows. The set of states of A_l is $[0..n] \times \mathbb{Z}$. For any state (i, c), i represents the position at the beginning of the word w_{i+1} in the sentence and c represents a positive or negative count of labels l. The initial state is $(0, 0)$, and the final state is $(n, 0)$. Transitions are labeled by lexicon entries $S_{i,j}$. Given any $S_{i,j}$, there is a transition $(i-1, x) \xrightarrow{S_{i,j}} (i, y)$ if y is the sum of x and the count of labels l in the multi-set $destr(S_{i,j})$.

Reaching state (i, c) from the initial state $(0, 0)$ means that

(a) the path taken is of the form $S_{1,j_1}, S_{2,j_2}, \dots, S_{i,j_i}$, that is a tagging of the first i words,

(b) c is the count of labels l present in the union of the multi-sets $abs(S_{1,j_1}), abs(S_{2,j_2}), \dots, abs(S_{i,j_i})$.

As a consequence, any path that leads to the final state corresponds to a neutral choice of tagging for this label l.

The algorithm is now simply to construct for each label l the automaton A_l and to make the intersection $A = \bigcap_{l \in Labels} A_l$ of all these automata. The result of the disambiguation is the set of paths from the initial state to the final state described by this intersection automaton. Notice that at each step of the construction of the intersection, one should prune automata from their blind states to ensure the efficiency of the procedure.

Now, in the general case, unification of labels in F does not reduce to identification, which introduces nondeterminism in the application of the neutralization rule. Parsing continues to reduce to counting polarities but now the counting of different labels is nondeterministic and interdependent. For instance, consider the multiset $\{+a, +b, -a \sqcup +b\}$ of three different elements. If we count the number of a, we find 0 if we consider that $+a$ is neutralized by $-a \sqcup b$ and +1 otherwise; in the first case, we find +1 for the count of b and in the second case, we find 0. Interdependency between the counts of different labels is very costly to be taken into account and in the following we ignore this property; therefore, in the previous exemple, we consider that the count of a is 0 or +1 and the count of b is also 0 or +1 independently from the first one. For expressing this, given a label l of F and a positive or negative label l' of F_{pol}, we define $P_l(l')$ as a segment of integers, which represents the possible counts of l found in l', as follows:

- if l' is positive, then $P_l(l') = \begin{cases} [\![1,1]\!] & \text{if } l \sqsubseteq l' \\ [\![0,0]\!] & \text{if } l \sqcup l' = \bot \\ [\![0,1]\!] & \text{otherwise} \end{cases}$
- if l' is negative, then $P_l(l') = \begin{cases} [\![-1,-1]\!] & \text{if } l \sqsubseteq l' \\ [\![0,0]\!] & \text{if } l \sqcup l' = \bot \\ [\![-1,0]\!] & \text{otherwise} \end{cases}$

We generalize the function P_l to count the number ol labels l present in a multi-set $abs(S)$: $P_l(S) = [\![inf, sup]\!]$with:

$$inf = \sum_{l' \in abs(S)} min(P_l(l'))$$
$$sup = \sum_{l' \in abs(S)} max(P_l(l'))$$

The method of disambiguation using automata presented above is still valid in the general case with the following change in the definition of a transition in the automaton A_l: given any $S_{i,j}$, there is a transition $(i-1, x) \xrightarrow{S_{i,j}} (i, y)$ if y is the sum of x and some element of $P_l(S_{i,j})$. With this change, the automaton A_l becomes nondeterministic.

The interest of the incremental procedure is that it is global to the sentence and that it ignores word order. This feature is interesting for generation where the question of disambiguation is crucial. This advantage is at the same time its drawback when we need to take word order and locality into account. Under this angle, the bottom-up procedure, which will be presented below, is a good complement to the incremental procedure.

5.2 Bottom-up procedure

We propose here another procedure adapted to a formalism C with the property of projectivity. Because of this property, it is possible to use a CKY-like algorithm in the abstract formalism C_{abs}. To parse a sentence $w_1 w_2 \cdots w_n$, we construct items of the form (i, j, S) with S an element of $\mathbf{Struct}_{C_{abs}}$ and i and j such that $w_{i+1} \ldots w_j$ represents the phonological form of S. We assume that $\mathbf{Rules}(C_{abs})$ has only unary and binary rules. Then, three rules are used for filling the chart:

initialization: the chart is initialized with items in the form $(i, i+1, abs(S_{i+1,k}))$;

reduction: if the chart contains an item (i, j, S), we add the item (i, j, S') such that S' is obtained by application of a unary composition rule to S;

concatenation: if the chart contains two item (i, j, S) and (j, k, S'), we add the item (i, k, S'') such that S'' is obtained by application of a binary composition rule to S and S'.

Parsing succeeds if the chart contains an item in the form $(0, n, S_0)$ such that S_0 is an element of $\mathbf{Sat}_{C_{abs}}$. From such an item, we can recover all taggings that are at its source if, for every application of a rule, we keep a pointer from the conclusion to the corresponding premisses. The other taggings are eliminated.

6 Experiments

In order to validate our methodology, we have written two toy English grammars for the LG and the LTAG formalisms. The point of the tests we have done is to observe the performance of the lexical disambiguation on highly ambiguous sentences. Hence, we have chosen the three following sentences which have exactly one correct reading:

(a) *the saw cut the butter.*
(b) *the butter that the present saw cut cooked well.*
(c) *the present saw that the man thinks that the butter was cut with cut well.*

For each test below, we give the execution time in ms (obtained with a PC Pentium III, 600Mhz) and the performance (number of selected taggings / number of possible taggings).

6.1 Incremental procedure

The incremental procedure gives the following results:

	LG		LTAG	
	ms	perf.	ms	perf.
(a)	1	3/36	3	3/96
(b)	42	126/12960	40	126/48384
(c)	318	761/248832	133	104/1548288

One may notice that the number of selected taggings/total taggings decrease with the length of the sentence. This is a general phenomenon explained in (Bonfante et al., 2003).

6.2 Bottom-up procedure

The execution time for the bottom-up procedure grows quickly with the ambiguity of the sentence. So this procedure is not very relevant if it is used alone. But, if it is used as a second step after the incremental procedure, it gives interesting results. We give below the results obtained with the *destr* abstraction.

	LG		LTAG	
	ms	perf.	ms	perf.
(a)	2	3/36	9	3/96
(b)	154	104/12960	339	82/48384
(c)	2260	266/248832	1821	58/1548288

Some other experiments show that we can improve performance or execution time with specific methods for each formalism which are less abstract than *destr*.

6.2.1 Tailor-made abstraction for LG

For the formalism LG, we can add some partial structural information to the polarized label. As the formalism is projective, we record some constraints about the continuous segment associated with a polarity. In this way, some neutralizations possible in the *destr* abstraction are not possible anymore if the two polarities have incompatible constraints (i.e. lie in different segments). The execution time is problematic but it might be controlled with a bound on the number of polarities in every multiset[5]:

	LG	
sentence	Time(ms)	Perf.
(a)	2	1/36
(b)	168	5/12 960
(c) with bound 6	2 364	3/248 832

Without bound for sentence (c), the running time is over 1 min.

6.2.2 Tailor-made abstraction for LTAG

Another abstraction is possible for LTAG: it consists in keeping, with each polarity, some information of the LTAG tree it comes from. Rather than bags where all polarized labels are brought together, we have four kind of polarized pieces: (1) a positive label coming from the root of an initial tree, (2) a negative label coming from a substitution node, (3) a couple of dual label coming from the root and the foot of an auxiliary tree or (4) a couple of dual label coming from the two parts of a quasi-node. **Rules** in this formalism reflect the two operations of LTAG; they do not mix polarities relative to adjunction with polarities relative to substitution. This improve execution time:

	LTAG	
	ms	perf.
(a)	6	3/96
(b)	89	58/48 384
(c)	272	54/1 548 288

Conclusion

The examples we have presented above should not be used for a definitive evaluation of particular methods, but more as a presentation of the flexibility of our program: polarization, abstraction and parsing for lexical disambiguation. We have presented one general tool (the destructuring abstraction) that may apply to various grammatical framework. But we think that abstractions should be considered for specific frameworks to be really efficient. One of our purpose is now to try the various tools we have developped to some large covering lexicons.

So far, we have not taken into account the traditional techniques based on probabilities. Our point is that these should be seen as an other way of abstracting grammars. Our hope is that our program is a good way to mix different methods, probabilistic or exact.

[5]This bound expresses the maximum number of syntactic dependencies between a constituent and the others in a sentence.

References

G. Bonfante, B. Guillaume, and G Perrier. 2003. Analyse syntaxique électrostatique. *Traitement Automatique des Langues*. To appear.

P. Boullier. 2003. Supertagging: a Non-Statistical Parsing-Based Approach. In *8th International Workshop on Parsing Technologies (IWPT'03), Nancy, France, 2003*, pages 55–66.

P. de Groote. 1999. An algebraic correctness criterion for intuitionistic multiplicative proofnets. *Theoretical Computer Science*, 224:115–134.

A. Joshi and B. Srinivas. 1994. Disambiguation of super parts of speech (or supertags) : Almost parsing. In *COLING'94, Kyoto*.

S. Kahane. 2004. Grammaires d'unification polarisées. In *TALN'2004, Fès, Maroc*.

M. Moortgat. 1996. Categorial Type Logics. In J. van Benthem and A. ter Meulen, editors, *Handbook of Logic and Language*, chapter 2. Elsevier.

G. Perrier. 2003. *Les grammaires d'interaction*. Habilitation à diriger des recherches, Université Nancy2.

K. Vijay-Shanker. 1992. Using description of trees in a tree adjoining grammar. *Computational Linguistics*, 18(4):481–517.