Hypertags

Alexandra KINYON
Talana / Lattice, Univ. Paris 7
UFRL case 7003
2 pl Jussieu 75005 Paris, France
Alexandra.Kinyon@linguist.jussieu.fr

Abstract

Srinivas (97) enriches traditional morpho-syntactic POS tagging with syntactic information by introducing Supertags. Unfortunately, words are assigned on average a much higher number of Supertags than traditional POS. In this paper, we develop the notion of Hypertag, first introduced in Kinyon (00a) and in Kinyon (00b), which allows to factor the information contained in several Supertags into a single structure and to encode functional information in a systematic manner. We show why other possible solutions based on mathematical properties of trees are unsatisfactory and also discuss the practical usefulness of this approach.

Introduction

As a first step prior to parsing, traditional Part of Speech (POS) tagging assigns limited morpho-syntactic information to lexical items. These labels can be more or less fine-grained depending on the tagset , but syntactic information is often absent or limited. Also, most lexical items are assigned several POS. Although lexical ambiguities are dealt with by POS taggers, either in a rule-based or in probabilistic manner, it is useful to delay this decision at a further parsing step (e.g. Giguet (98) shows that knowing constituent boundaries is crucial for solving lexical ambiguity correctly). In order to do so, it would help to be able to encode several POS into one compact representation.

In order to assign richer syntactic information to lexical items Joshi & Srinivas (94) and Srinivas (97) introduce the notion of Supertags, developed within the framework of Tree Adjoining Grammars (TAG). The idea behind Supertags is to assign to each word in a sentence, instead of a traditional POS, an "elementary tree", which constitutes a primitive syntactic structure within the TAG framework. A supertagged text can then be inputed to a parser or shallow parser, thus alleviating the task of the parser. Several problems remain though:

- Even when no lexical ambiguity occurs, each word can anchor several trees (several hundreds for some verbs)[1]. On average for English a word is associated with 1.5 POS and with 9 supertags (Joshi (99)). One common solution to the problem is to only retain the "best" supertag for each word, or eventually the 3 best supertags for each word, but then early decision has an adverse effect on the quality of parsing if the wrong supertag(s) have been kept : one typically obtains between 75% and 92% accuracy when supertagging, depending on the type of text being supertagged and on the technique used) (cf Srinivas (97), Chen & al (99), Srinivas & Joshi (99)). This means that it may be the case that every word in 4 will be assigned the wrong supertag, whereas typical POS taggers usually achieve an accuracy above 95%.
- Supertagged texts rely heavily on the TAG framework and therefore may be difficult to exploit without being familiar with this formalism.
- Supertagged texts are difficult to read and thus difficult to annotate manually.
- Some structural information contained in Supertags is redundant
- Some information is missing, especially with respect to syntactic functions[2].

So our idea is to investigate how supertags can be underspecified so that instead of associating a set of supertags to each word, one could associate one single structure, which we call hypertag, and which contains the same information as a set of supertags as well as functional information

Our practical goal is fourfolds :

a) delaying decision for parsing

b) obtaining a compact and readable representation, which can be manually annotated

---

1 See Barrier & al. (00) for precise data for French, using the FTAG wide-coverage grammar developped at TALANA, University of Paris 7.

2 The usefulness of functional information in POS tagging has also been discussed within the reductionist paradigm (cf Voutilainen & Tapanainen (93)).

as a step towards building a treebank for French (cf Abeillé & al. (00a), Clément & Kinyon (00)).
c) extracting linguistic information on a large scale such as lexical preferences for verb subcategorization frames. (cf Kinyon (99a))
d) Building an efficient, but nonetheless psycholinguistically motivated, processing model for TAGs (cf Kinyon (99b))

Thus, in addition of being well-defined computational objects (Point a), hypertags should be "readable" (point b) and also motivated from a linguistic point of view (Points c & d).

In the first part of this paper, we briefly introduce the LTAG framework and give examples of supertags. In a second part, we investigate several potential ways to underspecify supertags, and show why these solutions are unsatisfactory. In a third part, we explain the solution we have adopted, building up on the notion of MetaGrammar introduced by Candito (96) and Candito (99). Finally, we discuss how this approach can be used in practice, and why it is interesting for frameworks other than LTAGs.

1 Brief Overview of LTAGs

A LTAG consists of a finite set of **elementary trees** of finite depth. Each elementary tree must "anchor" one or more lexical item(s). The principal anchor is called "head", other anchors are called "co-heads". All leaves in elementary trees are either "anchor", "foot node" (noted *) or "substitution node" (noted ↓).. These trees are of 2 types : **auxiliary** or **initial**[3]. A tree has at most 1 foot-node. A tree with a foot node is an auxiliary tree. Trees that are not auxiliary are initial. Elementary trees combine with 2 operations : **substitution** and **adjunction**, but we won't develop this point since it is orthogonal to our concern and refer to Joshi (87) for more details. Morphosyntactic features are encoded in atomic feature structures associated to nodes in elementary trees, in order to handle phenomena such as agreement.
Moreover, linguistic constraints on the well-formedness of elementary trees have been formulated :

- Predicate Argument Cooccurence Principle : there must be a leaf node for each realized argument of the head of an elementary tree.
- Semantic consistency : No elementary tree is semantically void
- Semantic minimality : an elementary tree corresponds at most to one semantic unit

Figure 1 shows a non exhaustive set of Supertags (i.e. elementary trees) which can be assigned to "beats"[4] , which is a verb in trees α1 (canonical tree), α2 (object extraction), β1 (object relative) and β2 (subject relative) and a noun in tree α3. So an LTAG can be seen as a large dictionary, were in addition of traditional POS, lexical entries are associated with several structures encoding their morphological as well as some of their syntactic properties, these structures being very similar to small constituent trees.

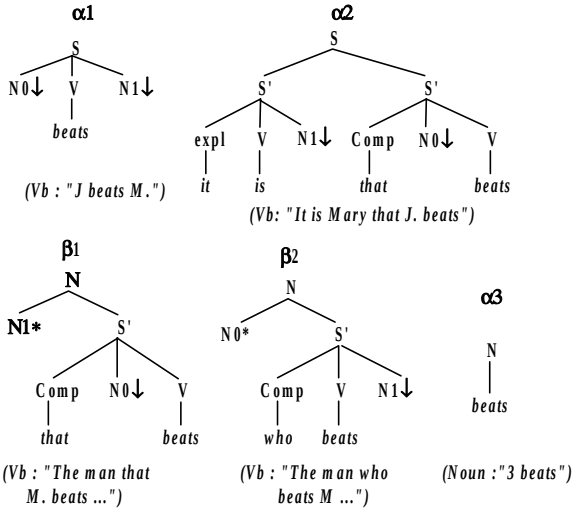

FIGURE 1 : some supertags for "beats"

2 Underspecifying Supertags

The idea of underspecifying constituent trees (and thus elementary trees) is not new. Several solutions have been proposed in the past. We will now investigate how these solutions could potentially be used to encode a set of supertags in a compact manner.

2.1 Parse forest

Since elementary trees are constituent structures, one could represent a set of elementary trees with a graph instead of a tree (cf. Tomita (91)). This approach is not particularly interesting though. For example, if one considers the trees α1 and β1 from figure 1, it is obvious that they hardly have any structural information in common, not even the category of their root. Therefore, representing these 2 structures in a graph would not help. Moreover, packed

[3] Traditionally initial trees are called α, and auxiliary trees β

[4] For sake of readability, morphological features are not shown.

structures are notoriously difficult to manipulate and yield unreadable output.

2.2 Logical formulae

With this approach, developped for instance in Kallmeyer (99), a tree can be represented by a logical formula, where each pair of nodes is either in relation of dominance, or in relation of precedance. This allows to resort to 1st order logic to represent a set of trees by underspecifying dominance and/or precedence relations . Unfortunately, this yields an output which is difficult to read. Also, the approach relies only on mathematical properties of trees (i.e. no linguistic motivations)

2.3 Linear types of trees

This approach, introduced in Srinivas (97), used in other work (e.g. Halber (99)) is more specific to TAGs. The idea is to relax constraints on the order of nodes in a tree as well as on internal nodes. A linear type consists in a 7-tuple <A,B,C,D,E,F,G> where A is the root of the tree, B is the category of the anchor, C is the lexical anchor, D is a set of nodes which can receive an adjunction, E is a set of co-anchors, F a set of nodes marked for substitution, and G a potential foot node (or nil in case the tree is initial). In addition, elements of E and F are marked + if they are to the left of the anchor, - if they are to the right.

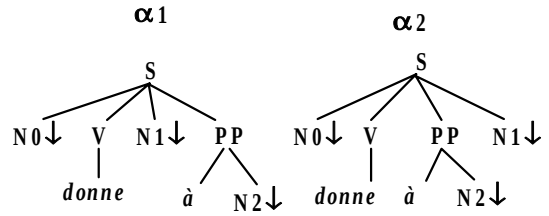

FIGURE 2 :
two trees with the same linear type

For example, the tree NOdonneN1àN2 for "*Jean donne une pomme à Marie*" (J. gives an apple to M.) and the tree N0donneàN2N1 for "*Jean donne à Marie une pomme*" (J. gives M. an apple) which are shown on Figure 2, yield the unique linear type (a)

(a) <S,V,donne,{S,V,PP},{à+},{N0-,N1+,N2+}, nil>

(b) <S,V,gives,{S,V,PP}, {to+}, {N0-,N1+,N2+} ,nil>

This approach is robust, but not really linguistic : it will allow to refer to trees that are not initially in the grammar. For instance, the linear type (b) will correctly allow the sentence "*John gives an apple to Mary*", but also incorrectly allow *"*John gives to Mary an apple*". Moreover, linear types are not easily readable[5]. Finally, trees that have more structural differences than just the ordering of branches will yield different linear types. So, the tree N0giveN1toN2 (*J. gives an apple to M.*) yields the linear type (b), whereas the tree N0giveN2N1 (*J. gives M. an apple*) yields a different linear type (c), and thus both linear types should label "gives". Therefore, it is impossible to label "gives" with one unique linear type.

(c) <S,V,gives,{S,V}, {}, {N0-,N1+,N2+} ,nil>

2.4. Partition approach

This approach, which we have investigated, consists in building equivalence classes to partition the grammar, each lexical item then anchors one class instead of a set of trees. But building such a partition is prohibitively costly : a wide coverage grammar for French contains approx. 5000 elementary trees (cf Abeillé & al. (99), (00b)), which means that we have 2^{5000} possible subsets. Also, it does not work from a linguistic point of view :

(a) Quand Jean a brisé la glace ?
(When did J. break the ice ?)

(b) Jean a brisé la glace *(J. broke the ice)*

(c) Quelle chaise Jean a brisé ce matin ?
(Which chair did J. break this morning ?)

In (a) *brisé* potentially anchors N0briseN1 (canonical transitive), WhN0brise (object extraction) and N0BriseGlace (tree for idiom). But in (b), we would like *brisé* not to anchor WhN0brise since there is no Wh element in the sentence, therefore these three trees should not belong to the same equivallence class : We can have class A={N0briseN1,N0BriseGlace} and ClassB={WhN0brise}. But then, in (c), *brisé* potentially anchors WhN0brise and N0briseN1 but not N0BriseGlace since *glace* does not appear in the sentence. So N0VN1 and N0BriseGlace should not be in the same equivalence class. This hints that the only realistic partition of the grammar would be the one were each class contains only one tree, which is pretty useless.

4. Exploiting a MetaGrammar

Candito (96), (99) has developed a tool to generate semi-automatically elementary trees She use an additional layer of linguistic description, called the metagrammar (MG), which imposes a general organization for syntactic information in a 3 dimensional hierarchy :

---

[5] This type of format was considered as a step towards creating a treebank for French (cf Abeillé & al 00a), but unfortunately proved impossible to manually annotate.

- **Dimension 1:** initial subcategorization
- **Dimension 2:** redistribution of functions and transitivity alternations
- **Dimension 3:** surface realization of arguments, clause type and word order

Each terminal class in dimension 1 describes a possible initial subcategorization (i.e. a tree family). Each terminal class in dimension 2 describes a list of ordered redistributions of functions (e.g. it allows to add an argument for causatives). Finally, each terminal class in dimension 3 represents the surface realization of a (final) function (e.g. cliticized, extracted ...).

Each class in the hierarchy corresponds to the partial description of a tree (cf. Rogers & Vijay-Shanker (94)). An elementary tree is generated by inheriting from one terminal class in dimension 1, from one terminal class in dimension 2 and from n terminal classes in dimension 3 (were n is the number of arguments of the elementary tree). [6] The hierarchy is partially handwritten. Then crossing of linguistic phenomena (e.g. passive + extraction), terminal classes, and from there elementary trees are generated automatically off line. This allows to obtain a grammar which can then be used to parse online. When the grammar is generated, it is straight forward to keep track of the terminal classes each elementary tree inherited from : Figure 3 shows seven elementary trees which can supertag "*donne*" (*gives*), as well as the inheritance patterns[7] associated to each of these supertags. All the examples below will refer to this figure.

The key idea then is to represent a set of elementary trees by a disjunction for each dimension of the hierarchy. Therefore, a hypertag consists in 3 disjunctions (one for dimension 1, one for dimension 2 and one for dimension 3). The cross-product of the disjunctions can then be performed automatically and from there the set of elementary trees referred to by the hypertag will

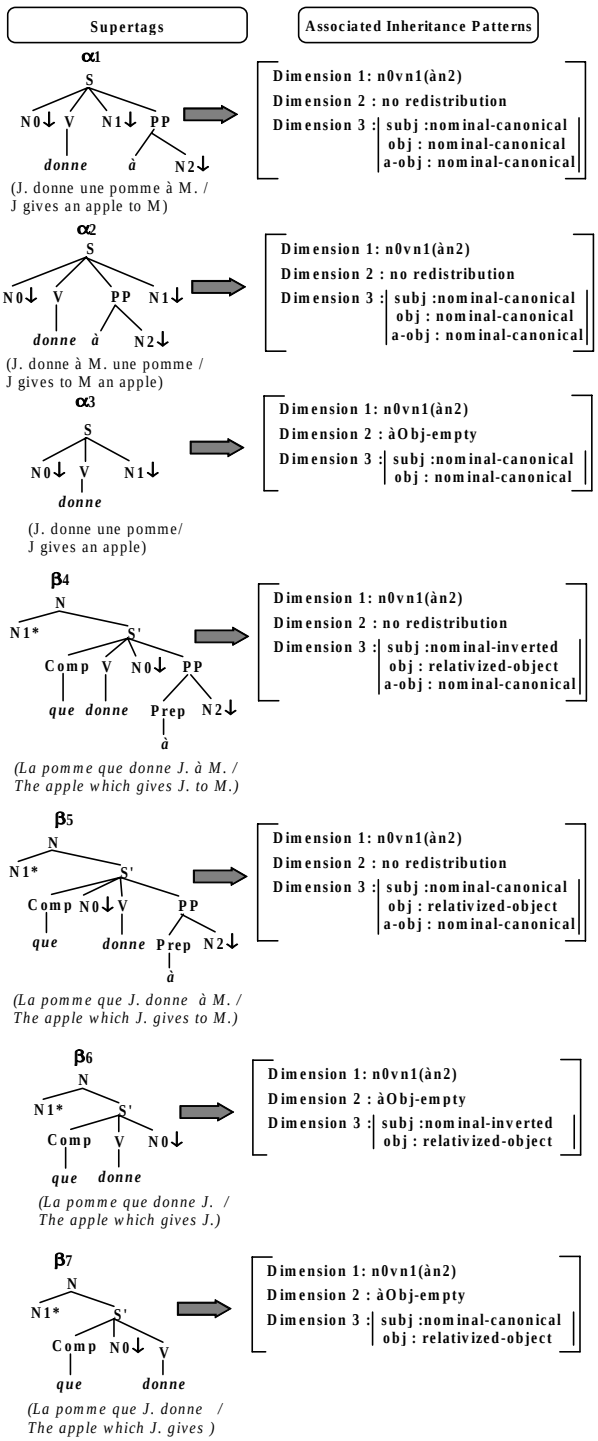

FIGURE 3 : SuperTags and associated inheritance patterns

be automatically retrieved We will now illustrate this, first by showing how hypertags are built, and then by explaining how a set of trees (and thus of supertags) is retrieved from the information contained in a hypertag.

4.1 Building hypertags : a detailed example

Let us start with a simple exemple were we want "donner" to be assigned the supertags α1 (*J. donne une pomme à M./ J. gives an apple to M.*) and α2 (*J donne à M. une pomme/J. gives M. an*

---

[6] The idea to use the MG to obtain a compact representation of a set of SuperTags was briefly sketched in Candito (99) and Abeillé & al. (99), by resorting to MetaFeatures, but the approach here is slightly different since only information about the classes in the hierarchy is used .

[7] We call inheritance patterns the structure used to store all the terminal classes a tree has inherited from.

apple). On figure 3, one notices that these 2 trees inherited exactly from the same classes : the relative order of the two complements is left unspecified in the hierarchy, thus one same description will yield both trees. In this case, the hypertag will thus simply be identical to the inheritance pattern of these 2 trees :

Dimension 1:	n0vn1(àn2)
Dimension 2 :	no redistribution
Dimension 3	subj :nominal-canonical obj : nominal-canonical a-obj: nominal-canonical

Let's now add tree α3 (*J. donne une pomme / J. gives an apple*) to this hypertag. This tree had its second object declared empty in dimension 2 (thus it inherits only two terminal classes from dimension 3, since it has only 2 arguments realized). The hypertag now becomes[8] :

Dim. 1:	n0vn1(àn2)
Dim. 2 :	no redistribution **OR àObj- empty**
Dim. 3	subj :nominal-canonical obj : nominal-canonical a-obj: nominal-canonical

Let's now add the tree β4 for the object relative to this hypertag. This tree has been generated by inheriting in dimension 3 from the terminal class "nominal inverted" for its subject and from the class "relativized object" for its object. This information is simply added in the hypertag, which now becomes :

Dim. :	n0vn1(àn2)
Dim. 2 :	no redistribution OR àObj- empty
Dim. 3	subj :nominal-canonical **OR nominal-inverted** obj : nominal-canonical **OR relativized-object** a-obj: nominal-canonical

Also note that for this last example the structural properties of β4 were quite different than those of α1, α2 and α3 (for instance, it has a root of category N and not S). But this has little importance since a generalization is made in linguistic terms without explicitly relying on the shape of trees.

It is also clear that hypertags are built in a monotonic fashion : each supertag added to a hypertag just adds information. Hypertags allow to label each word with a unique structure[9]. and contain rich syntactic and functional information about lexical items (For our example here the word *donne/gives*). They are linguistically motivated, but also yield a readable output. They can be enriched or modified by human annotators or easily fed to a parser or shallow parser.

4.2 Retrieving information from hypertags

Retrieving information from hypertags is pretty straightforward. For example, to recover the set of supertags contained in a hypertag, one just needs to perform the cross-product between the 3 dimensions of the hypertag, as shown on Figure 4, in order to obtain all inheritance patterns. These inheritance patterns are then matched with the inheritance patterns contained in the grammar (i.e. the right column in Figure 3) to recover all the appropriate supertags. Inheritance patterns which are generated but don't match any existing trees in the grammar are simply discarded.

We observe that the 4 supertags α1, α2 and α3 and β4 which we had explicitly added to the hypertag in 4.1 are correctly retrieved. But also, the supertags β5, β6 and β7 are retrieved, which we did not explicitly intend since we never added them to the hypertag. But if a word can anchor the 4 first trees, then it will also necessarily anchor the three last ones : for instance we had added the canonical tree without a second object realized into the hypertag (tree α2), as well as the tree for the object relative with a second object realized realized (tree β4), so it is expected that the tree for the object relative without a second object realized can be retrieved from the hypertag (tree β6) even though we never explicitly added it. In fact, the automatic crossing of disjunctions in the hypertag insures consistency.

Also note that no particular mechanism is needed for dimension 3 to handle arguments which are not realized : if *àObj-empty* is inherited from dimension 2, then only *subject* and *object* will inherit from dimension three (since only arguments that are realized inherit from that dimension when the grammar is generated).

Information can be modified at runtime in a hypertag, depending on the context of lexical items. For example "*relativized-object*" can be suppressed in dimension 2 from the hypertag shown on Figure 4, in case no Wh element is encountered in a sentence. Then, the correct set of supertags will still be retrieved from the

---

[8] What has been added to a supertag is shown in bold characters.

[9] We presented a simple example for sake of clarity, but traditional POS ambiguity is handled in the same way, except that disjunctions are then added in dimension 1 as well.

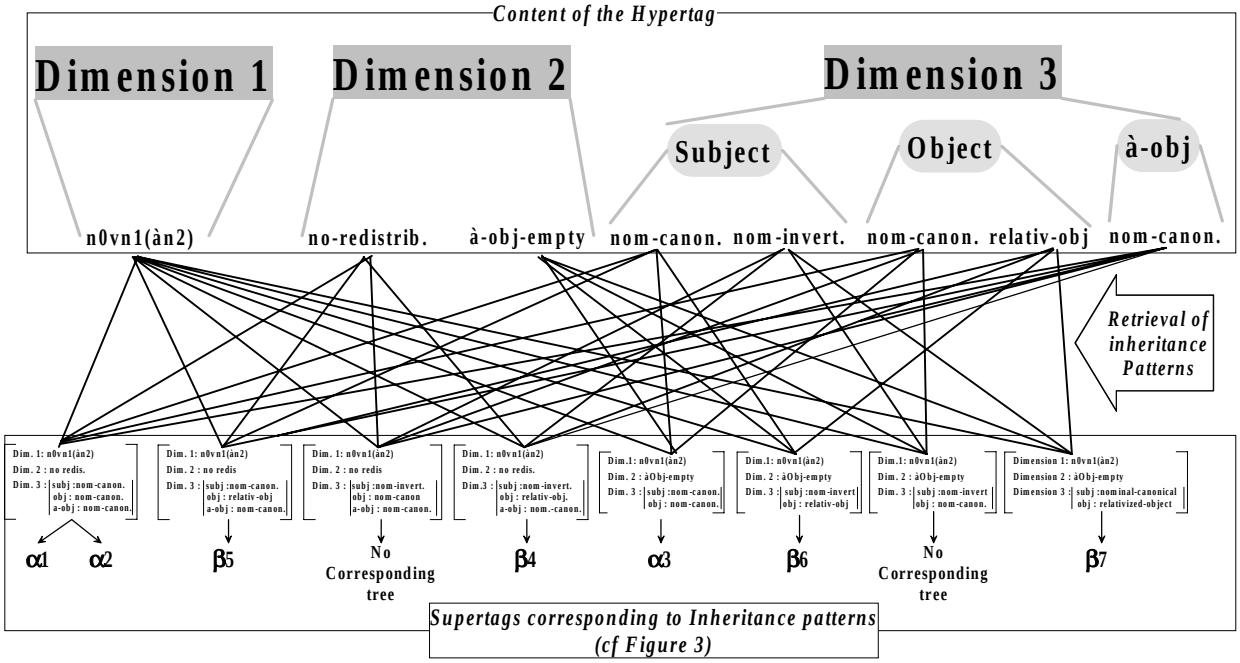

FIGURE 4 : Retrieving Inheritance patterns and Supertags from a Hypertag

hypertag by automatic crossing (that is, trees α1, α2 and α3), since the other inheritance patterns generated won't refer to any tree in the grammar (here, no tree inherits in dimension 3 *subject:inverted-nominal*, without inheriting also *object: relativized-object*)

4.3 Practical use

We have seen that an LTAG can be seen as a dictionary, in which each lexical entry is associated to a set of elementary trees. With hypertags, each lexical entry is now paired with one unique structure. Therefore, automatically hypertagging a text is easy and involves a simple dictionary lookup. The equivalent of finding the "right" supertag for each lexical item in a text (i.e. reducing ambiguity) then consists in dynamically removing information from hypertags (i.e. suppressing elements in disjunctions). This can be achieved by specific rules, which are currently being developed. The resulting output can then easily be manually annotated in order to build a gold-standard corpus : manually removing linguistically relevant pieces from information in a disjunction from a single structure is simpler than dealing with a set of trees. In addition of obvious advantages in terms of display (tree structures, especially when presented in a non graphical way, are unreadable), the task itself becomes easier because topological problems are solved automatically: annotators need just answer questions such as "does this verb have an extracted object ?", "is the subject of this verb inverted ?" to decide which terminal classe(s) must be kept[10] .We believe that these questions are easier to answer than "Which of these trees have a node N1 marked wh+ at address 1.1 ?" (for an extracted object).

Moreover, supertagged text are difficult to use outside of an LTAG framework, contrary to hypertagged texts, which contain higher level general linguistic information. An example would be searching and extracting syntactic data on a large scale : suppose one wants to extract all the occurrences where a given verb V has a relativized object. To do so on a hypertagged text simply involves performing a "grep" on all lines containing a V whose hypertag contains *dimension 3 : objet:relativized-object* , without knowing anything about the LTAG framework. Performing the same task with a supertagged text involves knowing how LTAGs encode relativized objects in elementary trees and scanning potential trees associated with V. Another example would be using a hypertagged text as an input to a parser based on a framework other than LTAGs : for instance, information in hypertags could be used by an LFG parser to constrain the construction of an F-structure, whereas it's unclear how this could be achieved with supertags.

10 This of course implies that one must be very careful in choosing evocative names for terminal classes.

The need to "featurize" Supertags, in order to pack ambiguity and add functional information has also been discussed for text generation in Danlos (98) and more recently in Srinivas & Rambow (00). It would be interesting to compare their approach with that of hypertags.

Conclusion

We have introduced the notion of Hypertags. Hypertags allow to assign one unique structure to lexical items. Moreover this structure is readable, linguistically and computationally motivated, and contains much richer syntactic information than traditional POS, thus a hypertagger would be a good candidate as the front end of a parser. It allows in practice to build large annotated resources which are useful for extracting syntactic information on a large scale, without being dependant on a given grammatical formalism.

We have shown how hypertags are built, how information can be retrieved from them. Further work will investigate how hypertags can be combined directly.

References

Abeillé A., Candito M.H., Kinyon A. (1999) FTAG : current status & parsing scheme. Proc. Vextal'99. Venice.

Abeillé A., Clément L., Kinyon A. (2000a) Building a Treebank for French. Proc. LREC'2000. Athens.

Abeillé A., Candito M.H., Kinyon A. (2000b) Current status of FTAG. Proc TAG+5. Paris.

Barrier N. Barrier S. Kinyon A. (2000). Lexik : a maintenance tool for FTAG. Proc. TAG+5. Paris.

Candito M-H. (1996) A principle-based hierarchical representation of LTAGs, Proc. COLING'96 Kopenhagen.

Candito M.-H, (1999) Représentation modulaire et paramétrable de grammaires électroniques lexicalisées. Application au français et à l'italien. PhD dissertation. University Paris 7.

Chen J., Srinivas B., Vijay-Shanker K. 1999 New Models for Improving Supertag Disambiguation. Proc. EACL'99 pp. 188-195. Bergen.

Clément L, Kinyon A. (2000) Chunking, marking and searching a morphosyntactically annotated corpus for French . Proc. ACIDCA'2000. Monastir.

Danlos L (1998) GTAG : un formalisme lexicalisé pour la génération automatique de TAG. TAL 39:2.

Giguet E. (1998) Méthodes pour l'analyse automatique de structures formelles sur documents multilingues.PhD thesis. Université de Caen.

Halber A. (1999) Stratégie d'analyse pour la compréhension de la parole : vers une approche à base de Grammaires d'Arbres Adjoints Lexicalisées. PhD thesis. ENST. Paris

Joshi A. (1987) An introduction to Tree Adjoining Grammars. In Mathematics of Language. A. Manaster-Ramer (eds). John Benjamins Publishing Company. Amsterdam.Philadelphia. pp. 87-114.

Joshi A. (1999) Explorations of a domain of locality. CLIN'99. Utrecht.

Joshi A. Srinivas B. (1994) Disambiguation of Super Parts of Speech (or Supertags) : Almost parsing. Proceeding COLING'94. Kyoto.

Kallmeyer L (1999) Tree Description Grammars and Underspecified Representations. PhD thesis, Universität Tübingen.

Kinyon A. (1999a) Parsing preferences with LTAGs : exploiting the derivation tree. Proc. ACL'99.College Park, Md

Kinyon A. (1999b) Some remarks about the psycholinguistic relevance of LTAGs. Proc. CLIN'99. Utrecht.

Srinivas B. (1997) Complexity of lexical descriptions and its relevance for partial parsing, PhD thesis, Univ. of Pennsylvania.

Srinivas B., Joshi A. (1999) Supertagging : An approach to almost parsing. Computational Linguistics 25:2.

Srinivas B. Rambow O. (2000) Using TAGs, a Tree Model, and a Language Model for Generation. Proc. TAG+5. Paris.

Tomita M. (1991) Generalized LR Parsing. Masaru Tomita (eds). Kluwer academic publishers..

Rogers J., Vijay-Shanker K. (1994) Obtaining trees from their descriptions : an application to TAGs. Computational Intelligence, 10:4 pp 401-421.

Voutilainen A. Tapanainen P. (1993) Ambiguity resolution in a reductionistic parser. Proc. EACL'93.