PoS Disambiguation and Partial Parsing Bidirectional Interaction

Montserrat Marimon Felipe^{*} Jordi Porta Zamorano[‡]

*Grup d'Investigació en Lingüística Computacional Universitat de Barcelona montse@gilcub.es

[‡]Departamento de Lingüística Computacional Real Academia Española porta@rae.es

Abstract

This paper presents Latch; a system for PoS disambiguation and partial parsing that has been developed for Spanish. In this system, chunks can be recognized and can be referred to like ordinary words in the disambiguation process. This way, sentences are simplified so that the disambiguator can operate interpreting a chunk as a word and chunk head information as a word analysis. This interaction of PoS disambiguation and partial parsing reduces the effort needed for writing rules considerably. Furthermore, the methodology we propose improves both efficiency and results.

1. Introduction

Rule-based part of speech (PoS) tagging aims at disambiguating texts –choosing the appropriate tag for each lexical item– looking at the sentential context. Distributional generalizations are implemented by a set of constraint rules promoting and/or discarding analyses.

This linguistic approach thus requires a high effort for writing an exhaustive grammar to deal with ambiguities which, quite often, go beyond the definition of morphosyntax. Ambiguities within a lexicon encoding morphosyntactic information are mainly due to homonomy and multiple functions of affixes. Morphosyntactic corpus annotation (PoS tagging) is the first application of such a lexicon, which, in turn, usually constitutes a prerequisite for further (and more complex) types of analyses (e.g., syntactic, semantic annotation). Lexical entries are frequently encoded taking them into account, which results in an increase of the number of ambiguities due to semantic and/or syntactic distinctions which are not marked by any feature.

In addition, a lineal distributional generalization does not always provide with enough information for effective disambiguation. Phenomena like non-local agreement and discontinuity are difficult to express without information about constituency structure and head information.

The system we present here –Latch¹– marks chunks and uses that information for PoS disambiguation. Chunks (possible formed by ambiguous word sequences) can be recognized by the system and can be referred to like ordinary words in the disambiguation process. This way, sentences are simplified so that the disambiguator can operate interpreting a chunk as a word and chunk head information as a word analysis. This interaction of PoS disambiguation and partial parsing reduces the effort needed for writing rules considerably. We need less and simpler rules. Furthermore, the methodology we propose improves both efficiency and results.

We will first present the Latch formalism, and we will

see how we make use of a mechanism for macro processing and a literate programming writing style to increase the expressivity and abstraction of the rule system. Then, we will present our grammar rules, having been built up as a set of modular components defined along such criteria as efficiency and reliability degree. Here we will see how grammar development may be improved by using chunks.

2. The Latch Formalism

Latch is a lean formalism for rule-based disambiguation and chunking. Rules take the following syntactic form²:

$$\langle rule \rangle ::= @TOK \langle rule name \rangle \\ \langle left context \rangle^* \langle tpivot \rangle^n \langle right context \rangle^* \\ [\& \langle score \rangle^n] \\ | @ \langle chunk type \rangle \langle rule name \rangle \\ \langle left context \rangle^* \langle cpivot \rangle^n \langle right context \rangle^*$$

where:

- (*rule name*) is the rule identifier, which is displayed when the tracing facility is enabled.³.
- \$\langle tpivot \rangle^n\$ is a non-empty sequence of elements (n > 0) which specifies the lexical elements to be disambiguated by the rule. Its syntax is:

$$\langle tpivot \rangle$$
 ::= $\langle [\langle guard \rangle - \rangle] \langle pattern \rangle \rangle$

In its simpler form, $\langle pattern \rangle$ is a tuple of $\langle full-form \rangle$, $\langle lemma \rangle$ and $\langle MSD \rangle^4$ which uses regular expressions⁵

¹Linguistic Annotator Tagger and Chunker.

²As usually: [] denotes optionality; $\langle sym \rangle$ marks non-terminal symbols; ^{*n*}, ⁺ and ^{*} are repetition operators; | separate alternative expressions; and other symbols are terminals.

³Therefore, the choice of mnemonic names is strongly recommended.

⁴Morphosyntactic description.

⁵Regular expressions are implemented with the GNU regex library.

for each of the three elements where omission is interpreted as a regular expression .*. Rules may thus refer to words, expressing lexical preferences, and/or morphosyntactic analyses, where underspecification is expressed by relaxing regular expressions.

In its general form, *patterns* can be formed using the logic connectives && (and), || (or), and !! (not), as expressed in the following grammar:

One of the main features of Latch is that it allows the grammar developer to specify a precondition –what we call $\langle guard \rangle$ – to the lexical element the rule is referred to. In this way, analyses are promoted/rejected not only on the basis of their contextual conditions, but also taking into account such further constraints.

Such a precondition is expressed by means of universally $(\&\{\langle pattern \rangle\}\&)$ or existentially $(\#\{\langle pattern \rangle\}\#)$ quantified patterns, which, in turn, may be combined using the logic connectives && (and), || (or) and !! (not).

These elements appearing in the $\langle guard \rangle^6$ allows us to define concepts like 'class'⁷, to define an ambiguity class, which will constrain the analysis/analyses to be promoted/rejected by the rule, as we exemplify in (2):

```
(2) < #{@@N.*}# && #{@@A.*}# &&
    !!#{!!@@N.* && !!@@A.*}#
    -> @@N.* >
```

In addition, we may also define the concept (ambiguity) 'superclass'⁸ in order to deal with any ambiguity type which is subsumed by the specified ambiguity, which we exemplify in (3):

(3) $\langle \# \{ @@N. * \} \# \&\& \# \{ @@A. * \} \# - > @@N. * >$

During the development of a grammar it is important to provide with means to reason about the meaning of a rule, when intuitive interpretation of a rule is not clear. The Latch formalism has been designed to allow syntactical translation from rules to predicate calculus. In (4) and (5) we show how the *guard* in (2) and (3) is expressed and translated into predicate calculus.

(4) $\exists x Noun(x) \land \exists y Adj(y) \land \neg \exists z (\neg Noun(z) \land \neg Adj(z))$ (5) $\exists x Noun(x) \land \exists y Adj(y)$

As we will see in section 4.3., logical equivalence provides clues to express such constraints in a computationally efficient way.

- (*chunk type*) can be VCHUNK (*volatile chunks*) or PCHUNK (*persistent chunks*) for chunk rules.
- $\langle cpivot \rangle^n$ is a non-empty sequence of element constraints (n > 0) which are grouped into an opaque element.

Latch allows the grammar developer to specify a head element, which must be unique, by marking it with $\langle \rangle$. The production rule for $\langle cpivot \rangle$ is:

$$\langle cpivot \rangle ::= \langle element \rangle >$$

 $| \langle element \rangle >$

Head descriptions are then projected to the chunk element so that they can be used by the disambiguation rules both at the rule pivot to disambiguate the head of the chunk, and at the context conditions.

The concepts 'class' and 'superclass' may also be used to define chunks that will include ambiguous lexical items. Besides, we may also use the concept of 'subclass'⁹ to refer to analyses which are either unambiguous or ambiguous between them. We exemplify in (6), and in (7) we present how it is expressed and translated into predicate calculus:

The following is an example of the pivot of a chunk rule building a nominal chunk¹⁰.

• The *context conditions* are placed to the left and/or to the right of the rule pivot, indicating the elements preceding and/or following the readings to be disambiguated as well as the elements to be grouped. There is no limit to the number of contextual elements nor

⁶As well as in the contextual conditions, as we will see below. ⁷#{A}# && #{B}# && !!#{!!A && !!B}#: analyses match either A or B. It covers the ambiguity A|B.

⁸#{A}# && #{B}#: there is an analysis matching A or B. It covers the ambiguities A|B, A|B|C, ...

 $^{{}^{9}\&}amp;\{A \mid \mid B\}\&$: analyses match either A, B or A and B. It covers: A, B, A|B.

¹⁰Covering 'articles + premodifying adjectives or participles (possibly ambiguous between them) + unambiguous nouns'; e.g., *un/el detallado informe (the/a detailed report)*.

to the position they occur with respect to the pivot element. Constraints can go beyond neighbouring tokens. Our rules can have one of the following context expressions:

- The *immediate context operator* (<) states that there is an adjacent element that satisfies the element constraint.
- The unbounded context operator (<<) expresses that there is an element somewhere to the left that satisfies the element constraint.
- The constrained unbounded context operator

 (!<) expresses that there is an element somewhere to the left that satisfies the context condition but all elements between boundaries satisfy a constraint.

Similarly, >, >> and >! are defined for right contexts.

Constraint sequences are possible by concatenating context expressions by means of this relative ordering operators, but note that in " $C_1 < C_2 << C_3 << C_4$ " backtracking is required in order to search for possible anchoring contexts.

As within the pivot, the concepts of 'class', 'subclass' and 'superclass' may be used to refer to ambiguous contextual elements.

The last component of lexical rules is (score)ⁿ; an optional sequence of positive or negative scores the grammar developer assigns to a rule in order to promote or to reject the (possibly preconditioned) analyses specified by pivots on the basis of the contextual elements. The default value is '+1' when no score is provided to a rule.

In this *voting constraint* (Oflazer and Tür, 1997) system, in order to avoid undesirable interference between rules we give higher votes to rules that are more specific; i.e., rules that make reference to specific ambiguity classes or to specific lexical items (lemma or full-form).

3. Evaluation Mechanism

Rules operate within an SGML element (the default being the sentence SGML tags < S>...</S>).

The evaluator operates by cycles of two phases: disambiguation and chunk construction. During the first one, the evaluator tries to anchor rule pivots and, if context constraints are satisfied, scores are added to the pivots. Given a threshold based on the maximum score reached by some analyses, analyses scored below that are eliminated. The process of chunk construction is a recursive process so chunks can be built up by other chunks. Volatile chunks can be used for other chunk construction, but they are eliminated at the end of this phase and only persistent chunks survive.

The process can iterate once, or cycle until rules have no effect on text.

4. Increasing the Formalism Expressivity

4.1. Macros

As we have seen, Latch is a lean formalism where linguistic analyses are patterns that can be universally $(\& \{pattern\}\&)$ or existentially $(\#\{pattern\}\#)$ quantified. In order to increase the expressivity and the abstraction of rules, Latch is provided with a mechanism for macro processing: m4.

At a first level of abstraction, let's call it *vocabulary abstraction*, macro definitions may be used to make the rules independent from the tagset one may be using, which, in addition, frequently increases their readability. Here, Latch tags can be defined as fine grained as desired, also simulating underspecification:

```
(9) define(`NOUN',`Nc.*|Np.*')
    define(`NOUN_SG',`N..s*')
```

Using the same rules by another tagset would just require updating the macro definitions.

It is also important at this level of abstraction the use of macros to mark some syntactic and/or semantic distinctions which, going beyond the definition of morphosyntax, have not been encoded in the lexicon one might be using but which may have a relevant role in disambiguation. Examples below include the marking of degree adverbs and control and raising verbs:

(10) define(`DEG_ADV',`más|menos|muy|...')
 define(`V_Ovp',`deber|aconsejar|...')

Note that, by recursive macro expansion, hierarchical abstractions can be expressed in a very simple way:

```
(11) define('V_Ovp', 'V_RAIS|V_CTRL|...')
    define('V_RAIS', 'deber|...')
    define('V_CTRL', 'aconsejar|...')
```

At this abstraction level, we may also have parameterized macros. Examples of this type of macros are the ones we have implemented to deal with the concepts of 'class'. 'subclass' and 'superclass' we saw in section 2:

(12) a. class($elem_1, \ldots, elem_n$) b. subclass($elem_1, \ldots, elem_n$) c. superclass($elem_1, \ldots, elem_n$)

Examples (2), (3) and (8) we presented in section 2 will be obtained by macro expansion of:

(13) <class(@@NOUN,@@ADJ)->@@NOUN>
 <superclass(@@NOUN,@@ADJ)->@@NOUN>
 <subclass(@@NOUN,@@ADJ)>

Parameterized macros, in addition, allow us to collect some sequences of contextual elements. Let's call this type *structural abstraction*. Examples of this type of macros are the ones we have defined for subcategorization. Let's see the one for ditransitive verbs; i.e., taking a direct object (DO) and an indirect object (IO)

If we define macros for DO and IO:

the final expansion of (14 b.) in a rule like (16) will be (17):

```
(17) @TOK AdjVerb_PreNPPP_Verb
    <#{@@A.*}# && #{@@V.*}# &&
    !!#{!!@@A.* && !!@@V.*}#->@@V.*>
    > #{@@N.*}# && !!#{!!@@N.*}#
    > #{@@A.*}# && !!#{!!@@A.*}#
    > #{@@A.*}# && !!#{!!@@A.*}#
    % 1
```

Finally, we have the *rule abstraction* level where we define what we called schemata. An schema has the following syntax:

schema ($expression, var, val_1, \ldots, val_n$)

It duplicates *expression* n times, replacing *var* with *val*_{*i*} at iteration *i*th.

This can be used to simulate unification in order to reduce the number of rules to be implemented dealing, for example, with agreement (18) and subcategorization patterns (19):

```
(18) schema(
    @TOK ArtClit_PreNounAgr_DetAgr
    <class(@@TdAgr.,@@CLIT->@@TdAgr.>
        class(@@N.Agr.)
        % 1,
        Agr, fs, fp, ms, mp)
```

```
(19) schema(
    @TOK AdjVerb_Preprep_Verb
    class(BOS@@) <
        <class(@@ADJ,@@VERB->@@VERB>
        > subcat(PP(Prep))
        > class(EOS)
        % 1,
        Prep, a, hacia)
define('PP','> class(@$1@PREP)
```

```
> class(@@NOUN)')
```

which are expanded into (20) and (21) respectively:

```
c. @TOK ArtClit_PreNounms_Detms
    <class(@@Tdms.,@@CLIT->@@Tdms.>
        class(@@N.ms.)
        % 1
```

```
(21) a. @TOK AdjVerbLc_Prea_Verb
    class(BOS@@) <
        <class(@@ADJ,@@VERB->@@VERB>
        > class(@@ADPREP)
        > class(@@NOUN)
        > class(EOS@@)
        % 1
    b. @TOK AdjVerb_Prehacia_Verb
        class(BOS@@) <
        <class(BOS@@) <
        <class(@@ADJ,@@VERB->@@VERB>
        > class(@hacia@PREP)
        > class(@@NOUN)
```

> class(EOS@@)

```
81
```

4.2. Literate Programming

Grammar rules have been implemented in a literate programming system, namely **noweb**. Briefly, literate programming, invented by Donald Knuth (Knuth, 1984), is a philosophy for writing programs where (i)- you get to write the code in any order you want, independently of the order it will be executed; and (ii)- code and documentation can be intermingled. Thus, in the grammar we will present, sections correspond to the different ambiguity classes we found. There, rules are classified according to the different criteria we will present.

4.3. About Rule Coding Style and Efficiency

We will now see how quantified element definitions may be optimized, strongly recommended for those ones frequently used along the grammar.

One of the patterns more frequently used is 'class'. The macro 'class' generates code of this kind " $\#\{A_1\}\#\&\&$...&& $\#\{A_n\}\#\&\&$!! $\#\{!!A_1\&\&...\&\&!!A_1\}\#$ " when called with arguments A_1, \ldots, A_n . But there is a particular very frequent case when the macro is called with just one argument. The optimal expansion of the macro in this case is " $\&\{A\}\&$ " because the following sequence of logical equivalences holds:

$$\exists x \ A(x) \land \neg \exists x \neg A(x)$$

$$\equiv \ \exists x \ A(x) \land \forall x \ A(x)$$

$$\equiv \ \forall x \ A(x)$$

This optimization is carried out during the macro expansion since the macro takes into account the number of arguments.

Another example is the disjunction of existentially quantified elements. For instance, $\langle \# \{ @@A \} \# | | \\ \# \{ @@B \} \# \rangle$ could be rewritten as $\langle \# \{ @@A | B \} \# \rangle$. Since

the disjunction of contexts in the former is equivalent to the disjunction of regular expressions in the latter.

Another kind of optimization is done to the implementation. Based on the observation that, due to the laws of sum, lexical rules can be applied in any order, a multi-threaded version of the code has been implemented for parallel rule application. This results in a better performance on multiprocessor systems.

5. Grammar Definition

Our main goal was to implement a grammar which, on the one hand, could be easily and efficiently adapted to deal with different input text, and, on the other hand, it should also allow us to sacrifice its recall when an increase of its precision was preferable¹¹. Taking this into account, we defined our grammar as a set of modular components –defined along different criteria– to be activated (or deactivated) depending both on the type of input text and on the application. Our first task here was to make an exhaustive study of the ambiguities we found in our lexicon, then, rules were defined accordingly on the basis on such criteria as efficiency and reliability degree which constitute the different modules of our grammars and which we present in the following subsections.

5.1. Disambiguation Constraint Rules

- 1. We distinguished a first set of rules which are applied to specific lexical forms, (most of them) independently of the context in which they occur, to eliminate very rare readings. Examples are the ambiguity of *a* and *de* between 'noun/preposition'.
- 2. Our second subgrammar contains a very small set of rules which either remove readings that are always illegitimate in a given context (e.g., verbal reading following unambiguous determiners) or promote readings in context where no other reading is appropriate (e.g., verbs following unambiguous clitics). These are very simple and efficient rules where: (i)- contexts are limited to one element either preceding or following the rule pivot; (ii)- rule pivots are referred to categories rather than subtypes or lexical instantiation of such categories; and, (iii)- no precondition is specified in the focus of the rule, meaning that they apply whatever their lexical instantiation is.
- 3. Then, we have a set of rules which disambiguate on the basis of agreement within phrases. They eliminate (nominal, adjectival, participle and pronominal) readings which do not agree either in gender or in number with the tokens immediately preceding and/or following them.

4. Our fourth subgrammar is the biggest one. While in the previous set of rules we implemented very simple distributional generalizations dealing with a few categories, this is an exhaustive grammar dealing with all ambiguity types we found in our lexicon. Here we distinguished two subtypes of rules.

The first group of rules deals with the ambiguities between open class lexical items: adjectives, nouns and verbs. Even though we wanted to avoid as much as possible redundant constraints, since that clearly affected the performance of the grammar, our main goal was to ensure the reliability of the constraints. Here, the notion of (ambiguity) class played a crucial role, all rule pivots were specified w.r.t. the ambiguity we wanted to resolve. This strategy, in addition, allowed us to leave apart the disambiguation of 'adjective/participle' and 'adjective/noun' ambiguities, which need semantic, or, even, pragmatic information to be reliably solved. Within this group contextual constraints are not limited to a single element, but may include up to three or four elements, basically, establishing the distributional requirement within phrases.

Our second group deals with both the ambiguities between close classes and the ambiguity types we classified as 'special lexical items', covering ambiguities between open and close classes, which though they are not very numerous, they contain many of the most frequent words. To deal with them, we followed two different strategies. Where possible we employed the notion of superclass to reduce the number of rules. This, however, was only possible when no many ambiguity classes could be subsumed by the precondition. But the biggest set of ambiguities were dealt with contextual rules where the focus made reference to specific lexical items. This strategy was partly motivated by the fact that the application of constraint rules coded in the previous subgrammars added new ambiguity types to our original typology. Let's see an example. In the set of rules defined in the second subgrammar, we included a contextual constraint eliminating the determiner reading following another determiner or an article. If such a rule applies, the ambiguities coming from the lexicon between 'indefinite pronoun/indefinite adjective/indefinite determine/adverb' is reduced to 'indefinite pronoun/indefinite adjective/adverb', an ambiguity we did not have before in our lexicon. Dealing with so many special ambiguity classes by means of rules specifying an ambiguity class or superclass would have required adding too many rules to deal with the new classes. Besides, even when the resulting ambiguity coincided with one already present in the lexicon, this was not a "real" ambiguity case, so we could not deal with them the same way.

5.2. Chunk Rules: PoS Disambiguation and Partial Parsing Bidirectional Interaction

Our next step was to increase the distributional generalizations and to disambiguate on the basis of the order con-

¹¹The terms recall and precision, originally proposed by the developers of the ENGCC system (Karlsson et al., 1995), are broadly used in rule-based PoS tagging to evaluate the resulting disambiguated text. A recall of 100% means that all tokens have received the appropriate analysis; a precision of 100% means that there is no superfluous reading. When recall and precision are the same, then this value is called accuracy –which happens when all tokens received just one analysis (as happens when statistical methods are applied).

straints –or requirements– of sentential constituents. For this, we developed and integrated into the system a partial parser that built up 'chunks'. Our main goal was twofold. On the one hand, we aimed at interpreting a chunk as a lexical items and head information as a word analysis, both when appearing at the contextual conditions and the rules focus; on the other hand, to avoid rule diversification to deal with all possible phrasal elements.

5. Thus, the fifth subgrammar includes both the rules building chunks and the disambiguation constraints rules referring to them.

Basically, our notion of chunks follows very closely Abney's proposal (Abney, 1996). In this way, chunks represent intra-clausal constituents which are defined on purely syntactic basis, rather than semantically or lexically. Chunks are thus defined along categorial dimension, extending from any premodifying –or specifying– element up to the head element. However, we differ from Abney's in that we do not only recognize and mark what he names *maximal chunks* but we also mark those chunks which may be contained within another one, as for example adjectival phrases preceding the nominal head (22)

(22) [_{NX}mi [_{AX}mejor] amiga] my best friend

Been this mainly motivated to make the task of chunk definition simpler and to ease its maintenance and updating. With one simple rule where prenominal adjectives are included within the nominal chunks, we deal not only with adjectival chunks having a unique element, the head element, but also with modified and coordinated adjectives (23)

- (23) a. mucho más importante much more important
 - b. pequeñas y medianas empresas small and medium companies

Note that, by specifying contextual conditions we build up chunks which are part of another chunk, which otherwise we would rather do not build, as for example the one for coordinated element. Finally, the notions of 'class', 'subclass' and 'superclass' may also be used –and useful– in the chunk building process in order to include ambiguous items within them.

Let's now move to the disambiguation rules included in this module. The strategy we have followed here is similar to that of the previous one. Specific rules have been developed to deal with the different ambiguity classes by making use of the class and superclass concepts. The main innovation of this set of rules is that distributional constraints may refer to –or may be restricted to, when necessary– chunks. Such constraints appear at the rule pivot when we define rules to disambiguate the head element of a chunk, and at the contextual elements. Contextual constraints can thus go beyond constituents in a very simple, efficient and reliable way. One rule is enough to refer to whatever elements it includes. 6. Our first module groups together those rules that have been heuristically defined.

6. Future Work

In this paper we have presented a system which integrates partial parsing into PoS tagging. Even though we are still at the development phase, the first results look very promising. Future work will be concentrated on extending the rules and to test our system on a larger corpus than the one we have used so far (of about 20,000 words).

Our final goal is to integrate our system into a deep grammar in order to reduce the non-determinism of the high level processing.

7. References

- Abney, Steven, 1996. Chunk stylebook. Work in progress. Available at: http://sfs.nphil.uni-tuebingen.de/ ~abney/Papers.html#96i.
- Karlsson, F., A. Voutilainen, J. Heikkilä, and
 A. Anttila (eds.), 1995. Constraint Grammar: A Language-Independent System for Parsing Unrestricted Text. Berlin and New York: Mouton de Gruyter.
- Knuth, Donald E., 1984. Literate programming. *The Computer Journal*, 27(2):97–111.
- Oflazer, K. and G. Tür, 1997. Morphological disambiguation by voting constraints. In Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and 8th Conference of the European Chapter of the Association for Computational Linguistics (ACL/EACL'97). Madrid, Spain. Available at: http://xxx.lanl.gov/abs/cmp-lg/9704011.