Anaphor resolution in unrestricted texts with partial parsing[1]

A. Ferrández; M. Palomar
Dept. Languages and Information Systems
Alicante University - Apt. 99
03080 - Alicante - Spain
antonio@dlsi.ua.es mpalomar@dlsi.ua.es

L. Moreno
Dept. Information Systems and Computation
Valencia University of Technology
lmoreno@dsic.upv.es

Abstract

In this paper we deal with several kinds of anaphora in unrestricted texts. These kinds of anaphora are pronominal references, surface-count anaphora and one-anaphora. In order to solve these anaphors we work on the output of a part-of-speech tagger, on which we automatically apply a partial parsing from the formalism: *Slot Unification Grammar*, which has been implemented in Prolog. We only use the following kinds of information: lexical (the lemma of each word), morphologic (person, number, gender) and syntactic. Finally we show the experimental results, and the restrictions and preferences that we have used for anaphor resolution with partial parsing.

Introduction

Nowadays there are two different approaches to anaphor resolution: integrated and alternative. The former is based on the integration of different kinds of knowledge (e.g. syntactic or semantic information) whereas the latter is based on statistical, neural networks or the principles of reasoning with uncertainty: e.g. Connoly (1994) and Mitkov (1997).

Our system can be included into the first approach. In these integrated approaches the semantic and domain knowledge information is very expensive in relation to computational processing. As a consequence, current anaphor resolution implementations mainly rely on constraints and preference heuristics which employ information originated from morphosyntactic or shallow semantic analysis, e.g. in Baldwin (1997). These approaches, however, perform remarkably well. In Lappin and Leass (1994) it is described an algorithm for pronominal anaphor resolution with a high rate of correct analyses: 85%. This one operates primarily on syntactic information only. In Kennedy and Boguraev (1996) it is proposed an algorithm for anaphor resolution which is a modified and extended version of that developed by Lappin and Leass (1994). In contrast to that work, this algorithm does not require in-depth, full, syntactic parsing of text. The modifications enable the resolution process to work from the output of a POS tagger, enriched only with annotations of grammatical function of lexical items in the input text stream. The advantage of this algorithm is that anaphor resolution can be realized within NLP frameworks which do not -or cannot- employ robust and reliable parsing components. Quantitative evaluation shows the anaphor resolution algorithm described here to run at a rate of 75% accuracy. Our framework will allow us a similar approach to that of Kennedy and Boguraev (1996), but we will automatically get syntactic information from partial parsing. Moreover, our proposal will also be applied to other kinds of anaphors such as surface-count anaphora or one-anaphora.

There are some other approaches that work on the output of a POS tagger, e.g. that of Mitkov and Stys (1997), in which it is proposed another knowledge-poor approach to resolving pronouns in technical manuals in both English and Polish. This approach is a modification of the reported in Mitkov (1997). Here, the knowledge is limited to a small noun phrase grammar, a list of terms and

[1] This paper has been supported by the CICYT number TIC97-0671-C02-01 / 02

a set of antecedent indicators (definiteness, giveness, term preference, lexical reiteration, ...). We will work in a similar way to this approach, since we use some of its antecedent indicators, but we automatically apply a partial parsing that allows us to deal with other kinds of anaphors as well as pronouns.

In this work we are going to apply a partial parsing on the output of a POS tagger in order to solve anaphora problem. We will work over the corpus used within CRATER[2]. This corpus contains the International Telecommunications Union CCITT handbook, also known as *The Blue Book*, in English, French and Spanish versions. This corpus is the most important collection of telecommunication texts and contains 5M words, automatically tagged by the Spanish version of the Xerox tagger. We will use the system *Slot Unification Grammar (SUG)* in order to get a partial parsing on the output of this tagger.

SUG is a logical formalism based on unification, which is an extension of *Definite Clause Grammars (DCG)*. It is called *Slot Unification Grammar* due to the slot structures generated by the parser. SUG has been developed with the aim of extending DCG in order to facilitate the resolution of several Natural Language Processing (NLP) problems in a modular way. This system has been firstly proposed in Ferrández (1997a), and it has been previously applied to anaphor resolution in Ferrández (1997b).

We have used SUG instead of other well known formalisms such as *Head Driven Phrase Structure Grammar (HPSG)*, *Lexical Functional Grammar (LFG)* or *Slot Grammars (SG)*, because SUG allows a modular and computational treatment of NLP problems, and it facilitates its integration with a POS tagger.

In the following section we will briefly describe SUG formalism in order to facilitate the undertanding of this paper. In section 2 we will propose a SUG grammar to accomplish the partial parsing of the unrestricted text and the interface to work with the output of the POS tagger. In section 3 we will explain the algorithm used to anaphor resolution and its constraints and preferences. And, finally, in section 4 we will offer some figures of the evaluation of the system.

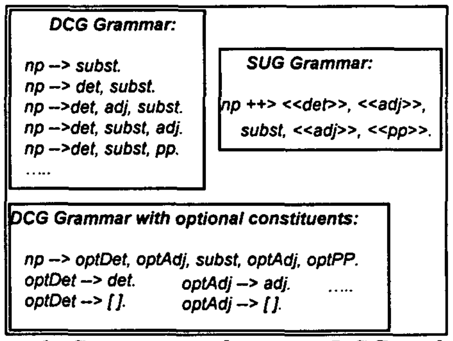
[2] http://138.87.135.33/~mdavies/roanoke.htm

1 Slot Unification Grammar

In this section we will briefly describe SUG formalism. We will only show some of the capabilities of SUG in order to undertand this paper. For further details on SUG it is necessary to consult Ferrández (1997a).

SUG can be defined as this quadruple: *(NT,T,P,H)*, where *NT* and *T* are a finite set of nonterminal and terminal symbols respectively; moreover $NT \cap T = \emptyset$. *P* is a finite set of pairs α ++> β where $\alpha \in NT$, $\beta \in (T \cup NT)^* \cup$ *{procedures calls}*, and these pairs are called *production rules*. Finally *H* is a set of production rules which only has the first member of the production rule, i.e. α, and α*'s* name is either *coordinated, juxtaposition, fusion, basicWord* or *isWord.*

SUG's production rules adds to those of DCG that each subconstituent of β could be omitted in the sentence if it is noted between the *optional operator*: << *constituent* >>. It is a well-known fact that we can get optional constituents in DCG from making use of a nonterminal symbol (e.g. *optA*, with *optA→A* and *optA→[ ]*). However this skill obliges us to add new nonterminal symbols, whereas SUG allows us to get it without adding any new one. We can get an example from Figure 1, in which we can see the reduction of grammatical rules in SUG.



```
DCG Grammar:

np --> subst.
np --> det, subst.
np -->det, adj, subst.
np -->det, subst, adj.
np -->det, subst, pp.
.....
```

```
SUG Grammar:

np ++> <<det>>, <<adj>>,
   subst, <<adj>>, <<pp>>.
```

```
DCG Grammar with optional constituents:

np --> optDet, optAdj, subst, optAdj, optPP.
optDet --> det.        optAdj --> adj.      .....
optDet --> [ ].        optAdj --> [ ].
```

Figure 1. Comparison between DCG and SUG with reference to optional constituents.

Furthermore, this optional operator has the possibility of reminding whether the optional constituent has been parsed in the sentence or not. This information will be very useful in the resolution of NLP problems such as ellipsis or extraposition. This fact is carried out by adding a label to the optional constituent, e.g. << ***SSNP*** ***:*** *np* >>. This label will be an uninstantiated Prolog variable if constituent *np* is missing, so Prolog predicate *var (SSNP)* would success.

We have developed a translator which turns SUG rules into Prolog clauses. This translator has been run under SICStus Prolog 2.1 and Arity Prolog 5.1, and it will translate into Prolog each SUG production rule. This translator will provide what we call *slot structure* (henceforth *SS*).

This *SS* stores the syntactic, morphologic and semantic information of every constituent of the grammar. Each SS consists of a structure with functor the name of the constituent (*np, vp, ...*). Its first argument corresponds to another structure with functor *conc* which includes all the arguments of the constituent (*Number, Gender, SemanticType*). The second one corresponds to the λp of the final logical formula of the constituent. And the remaining arguments correspond to the SS of its subconstituents. In this SS the parser leaves as uninstantiated Prolog variables ("_") the slots corresponding to the optional constituents that do not appear in the sentence, in this way, we know what has been parsed and what has not. From now on we will show each SS with λp and *conc* only if it is necessary, in order to get simplicity.

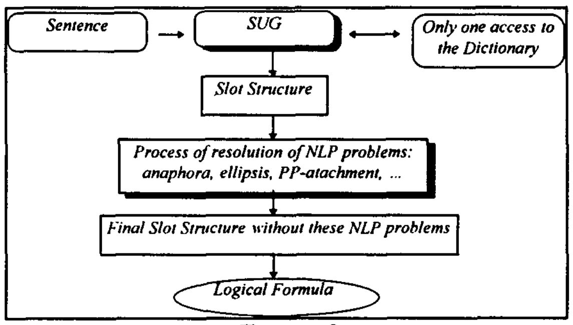

Sentence
SUG
Only one access to the Dictionary
Slot Structure
Process of resolution of NLP problems: anaphora, ellipsis, PP-atachment, ...
Final Slot Structure without these NLP problems
Logical Formula

Figure 2

Now we would like to make clear the process in which we obtain the final logical formula. First of all we parse the sentence, and then we get its SS. After that, it would be the moment in which we could try to solve NLP problems such as extraposition, ellipsis, PP-attachment and anaphora. The solution will consist of a new SS which will be used to obtain the final logical formula. This process has been summed up in Figure 2. We would like to emphasize that this skill of resolution allows us to produce modular NLP systems in which grammatical rules, logical formulas and the module of resolution of NLP problems are quite independent from each other.

Our SUG parser will access the dictionary only once during the whole process of parsing in order to avoid repeated access to the same word from the dictionary. It stores the information of each word on a list before starting the parse and it will work with this structure instead of the list of words of a DCG parser in Prolog; e.g. DCG list: *[this, book, is, mine]*, SUG list: *[**word** (this, [adj (sing, dem), pron (sing, dem)]), **word** (book, [noun (...)]), ...]*. Each element from the SUG list is a structure with name *word* and with two arguments. The first one corresponds to the same word of the sentence like a Prolog atom. The second one corresponds to a structure list which refers to the lexical entries of the word. That is to say that every time the parser has to access a lexical entry of a word, it will look it up in this list; it will not access the dictionary ever again.

2 Partial parsing with SUG

In Abney (1997) it is considered necessary to carry out a partial parsing on the unrestricted text instead of a complete parsing, both due to errors and the unavoidable incompleteness of lexicon and grammar. It is also difficult to do a global search efficiently with unrestricted text, due to the length of sentences and the ambiguity of grammars. Partial parsing is considered a response to these difficulties. Partial parsing techniques aim to recover syntactic information efficiently and reliably from unrestricted text, by sacrificing completeness and depth of analysis.

In this section we will show the application of SUG in partial parsing. We are going to take the output of a POS tagger as input, and after apply a partial parsing with SUG. The previously mentioned corpus *The blue book* is going to be worked on, which has been automatically tagged by the Spanish version of the Xerox tagger. Each word in a tagged sentence has the following syntax: *(surfaceForm, lemma, TAG)*.

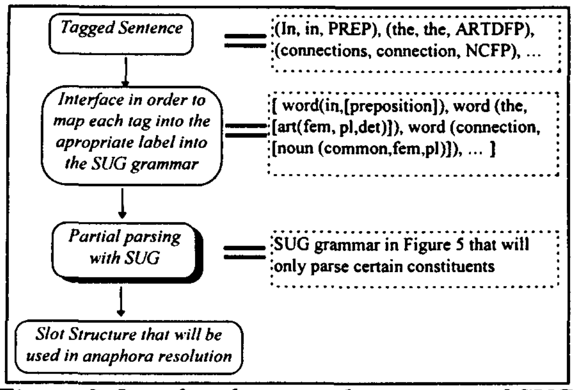

Tagged Sentence
(In, in, PREP), (the, the, ARTDFP), (connections, connection, NCFP), ...
Interface in order to map each tag into the apropriate label into the SUG grammar
[word(in,[preposition]), word (the, [art(fem, pl,det)]), word (connection, [noun (common,fem,pl)]), ...]
Partial parsing with SUG
SUG grammar in Figure 5 that will only parse certain constituents
Slot Structure that will be used in anaphora resolution

Figure 3. Interface between the tagger and SUG.

We will proceed in the way that is described in Figure 3. Firstly the tagged sentence is turned into the SUG list format, where each Xerox tag is mapped into the apropriate label into the SUG grammar, e.g. the Xerox tag *(connections, connection, NCFP)* is mapped into the SUG tag *word (connection, [noun (common, fem, pl)])*. Finally, this SUG list of words will be taken as input for the grammar described in Figure 4. This grammar will carry out the partial parsing of the text, and the SUG parser will produce the SS that will be used in the algorithm, which is proposed for anaphor resolution. This simple interface between the tagger and SUG is one of the advantages of the modularity that presents SUG. It will allow us to work with different dictionaries or taggers with the same SUG grammar. This is due to the fact that in this system there is a great independence between the grammar, the lexicon, the process of dealing with NLP problems and the process of obtaining the final logical formula.

```
sentence ++>
  << PP:pp >>, << NP:np >>, <<P:pronoun>>,
  << V:verb >>, <<C:conj>>,
  <# [ ],
     remainingSentence(PP,NP,P,V,C)  #> .
remainingSentence(PP,NP,P,V,C) ++>
  <## ( {( var(PP), var(NP), var(P), var(V), var(C))},   [W]),
       ( _                                          ,    _ )
  ##>,
  sentence.
%———— Grammatical rules for each constituent to parse
coordinated( pp, simplePP ).
simplePP ++> preposition, np.
coordinated( np, simpleNP (_) ).
simpleNP (substantiveType) ++> <<determiner>>,
              <<adjective>>, noun, <<pp>>.
simpleNP (adjectiveType) ++> <<determiner>>, adjective,
              <<pp>>.
...
```

Figure 4. Partial parsing with SUG.

The grammar in Figure 4 will only parse coordinated prepositional phrases (*pp*), coordinated noun phrases (*np*), pronouns (*p*), conjunctions (*conj*) and verbs (*verb*) in whatever order that they appear in the text and it will allow us to work in a similar way that the algorithm mentioned in Kennedy and Boguraev (1996). But in our approach we will automatically get the syntactic information from this grammar. The SS returned by the parser will consist of a sequence of these constituents: *pp, np, p, conj, verb* and *free words*. The attachments (e.g. of the *pp*) will be postponed to the module of resolution of NLP problems, which could work jointly with the algorithm for anaphor resolution (in a similar way to the approach proposed in Azzam (1995)). The *free words* will consist of constituents that are not covered by the grammar (e.g. adverbs) or words that are not important for the anaphor resolution. The output of the whole system will consist of a sequence of the logical formulas of each constituent.

Here *sentence* will be the initial symbol of the grammar and the partial parsing will be applied with the rules shown in Figure 4. If we want a complete parsing, we just have to substitute these rules for the following: *sentence ++> np, vp*, and obviously we will have to add the grammatical rule for a verbal phrase (*vp*).

3 The algorithm

In this section we are going to propose an algorithm which can deal with discourse anaphora in unrestricted texts with partial parsing. It is based on the process of parsing described in Figure 3. So this process will take the output of a POS tagging as input, and it will be applied after the partial parsing of a sentence (using the grammar described in Figure 4) and before obtaining its logical formula.

This algorithm is shown in Figure 5 and it will deal with pronominal references, surface-count anaphora and one-anaphora. This algorithm will take a slot structure (SS) that consists of a sequence of the following constituents: *np, pp, p, conj* and *verbs* and it will return a new one without anaphors. Every possible antecedent (noun phrases) will be stored in a *list of* *antecedents*, that will be used to solve the anaphors. Another structure will be stored in this list for each antecedent: *paral (Sent, Clause, PosVerb, NumConst, NumCoord)*. This structure will be used to deduce the parallelism with partial parsing between an anaphor and its antecedent. Its first argument, *Sent*, is the sentence in which the antecedent appears. The second one is the clause in which it appears. Consider that the beginning of a new clause has been found when we parse a *free* conjuction (we do not refer to the conjunctions that join the coordinated noun and prepositional phrases). The third one is the position of the antecedent with reference to the verb of the clause: before (*bv*) or after (*av*). The fourth one is the number of constituent in the sentence and the fifth one is the number of coordinated constituent if it is included in a coordinated *np* or *pp*. For example in: *He said that Peter and John bought a book*, we have the following: $paral_{He}$ *(S, 1, bv, 1, 1)*, $paral_{John}$ *(S, 2, bv, 4, 2)* and $paral_{book}$ *(S,2,av,6,1)*.

```
Parse a sentence. We obtain its slot structure (SS1).
For each anaphor in SS1:
  Select the antecedents of the previous X sentences
  depending on the kind of anaphor in L0
  Apply constraints (depending on the kind of anaphor) to L0
  with a result of L1:
  Case of:
    |L1| = 1 Then:
      This one will be the antecedent of the anaphor
    |L1| > 1 Then:
      Apply preferences (depending on the kind of anaphor) to
      L1, with a result of L2:
      The first one of L2 will be the selected antecedent
Update SS1 with each antecedent of each anaphor with a
result of SS2.
```

Figure 5. Algorithm for anaphor resolution.

At the same time that we are searching for antecedents, we will also search for anaphors and whenever we found an anaphor this algorithm will be applied. The kind of anaphors we are going to search are the following: pronouns (*he, she, ...*), pronominal noun phrases formed by: *determiner + pronoun* (*the second, the former, ...*), noun phrases with the structure: *determiner + adjective + "one"* (*the red one*, this anaphors in Spanish[3] are noun phrases in which the noun has been omitted: *el rojo*). We will identify such anaphors from its SS (its functor and its number and type of arguments). For example, the one-anaphor in Spanish will have the following SUG rule: *np* ++> *<<determiner>>*, *adjective*, *<<pp>>*, and the following SS: *np (determiner (...), adjective (...), pp (...))*.

The number of previous sentences considered in the resolution of an anaphor will be determined by the kind of anaphor itself. For pronominal references will be considered the antecedents in the same sentence or in the previous sentence if it is in the same paragraph, unlike to one-anaphora which have more lexical information, so we will consider the antecedents in the same paragraph. We will be able to know the number of sentence because this information will be stored jointly with the SS of every antecedent: for each sentence will be assigned a different Prolog variable and all the antecedents in this sentence will have this variable in its *paral* structure.

The algorithm will apply a set of constraints to the list of possible antecedents in order to discount candidates. If there is only one candidate, this one will be the antecedent of the anaphor. Otherwise, if there are still more than one candidates left, a set of preferences will be applied that will sort the list of remaining antecedents, and the selected antecedent will be the first one. It is important to remark that these constraints and preferences could be different for each kind of anaphor.

Next the constraints and preferences are going to be briefly explained. Morphosyntactic agreement (person, gender and number) will be checked by unification of the structure *conc* described in section 1. It is a strong constraint on reference, but it is not absolute: *At the zoo, a monkey scampered between two elephants. One snorted at it*[4], or in: <u>*John and Bill*</u>$_i$ *went into the shop. They*$_i$ *bought a book*. To solve the second example we will store a new antecedent with *plural* number which includes all the coordinated noun phrases (in this case *John and Bill*). We will detect the coordination of noun phrases from the SS returned by the SUG fact *coordinated*. In one-

[3] We are going to work with Spanish unrestricted texts, but whenever it is possible, all the examples will be translated into English in order to facilitate its understanding.

[4] In this paper we will not deal with problems caused by quantification.

anaphora we have considered the number agreement as a preference instead of a constraint in order to solve sentences like this: *Wendy didn't give either boy* <u>*a green shirt*</u>$_i$, *but she gave Sue* <u>*two red ones*</u>$_j$, where the anaphor and its antecedent do not agree in number (so they do not co-refer to the same entity of the discourse).

The c-command constraints will be applied on the syntactic information stored in the SS of each constituent and its structure *paral*. For example the following constraint: "A pronominal NP must be interpreted as non-coreferential with any NP that c-commands it", e.g. *Zelda*$_i$ *bores her*$_j$. It is accomplished by the information stored in their structures: *paral*$_i$ *(Sent1, Clause1, ...)* and *paral*$_j$ *(Sent1, Clause1, ...)* which means that they are in the same sentence and clause. However in *John*$_i$ *was late for work, because he*$_i$ *slept in*, here *John* and *he* can be coreferential because they are in different clauses separated by the conjunction *because*: *paral*$_{John}$ *(Sent1, Clause1, ...), paral*$_{he}$ *(Sent1, Clause2, ...)*. But in *John*$_i$ *and he*$_j$ *bought a book*, the pronoun will not corefer with *John* although there is a conjunction between them because they are in the same coordinated noun phrase, which is known from: *paral*$_i$ *(<u>S1, C1, bv, 1</u>, 1)* and *paral*$_j$ *(<u>S1, C1, bv, 1</u>, 2)*. In sentences like *(John*$_i$*'s portrait of him*$_j$*)*$_{NP}$ *is interesting* and *This is (the man*$_i$ *who he*$_j$ *saw)*$_{NP}$ the coreference is not permitted because the pronoun and the antecedent are in the same constituent *NP* (they are in the same slot structure: *np (det (the), noun (man), relSent (...)*). As well in *John bought a book for Peter*$_i$ *and for a friend of him*$_i$, the pronoun can corefer with *Peter* although they belong to the same coordinated constituent because the pronoun is an adjunct of the second coordinated constituent. From the reflexivity constraints in *Mary*$_i$ *loves herself*$_i$, we can conclude the antecedent of *herself* is *Mary* because they are in the same clause.

In relation to preferences, they will be different for each kind of anaphor: the non-reflexive pronouns will prefer the antecedent in the same sentence and clause, and if there are still more than one antecedent left, those in the same position with reference to the verb: *syntactic parallelism*. Moreover we have added some other preferences, e.g. a non-reflexive pronoun would not be allowed to have an antecedent that appear in the same clause due to reflexivity constraints: *Jack*$_i$ *saw Sam*$_j$ *at the party. Sam*$_j$ *gave him*$_i$ *a drink*. If after applying these preferences, there are more than one antecedent left, we will choose the antecedent most recently mentioned.

In order to solve *surface-count anaphora* we will use the SS returned by the SUG fact *coordinated*. This fact allows the coordination of constituents with the same or different form: *Peter, your daughter and she* and it will allow us to access whatever coordinated constituent in the order we wish. That is to say, its SS: *np (simpleNP (Peter), conj(','), np (simpleNP (det (your), noun (daughter)), conj (and), np (simpleNP (pron (she)), _, _)))*, and their structures *paral* with their fifth argument will tell us the number of coordinated constituent: *paral*$_{Peter}$ *(S, C, V, P, 1), paral*$_{daughter}$ *(S, C, V, P, 2)*, ... In this way the anaphor: *the second one* will choose an antecedent with a structure *paral* with a value of *2* in its fifth argument.

To solve *one-anaphora* we will apply the following preference: we will choose the antecedents with a similar structure. For example, in *Wendy didn't give either boy* <u>*a green tie-dyed T-shirt*</u>$_i$, *but she gave Sue* <u>*a blue one*</u>$_j$, the antecedent *a green tie-dyed T-shirt* would be chosen instead of *Wendy* or *Sue* because they have similar SS (a determiner, a common noun and an adjective): *np (noun(Wendy)), np*$_i$ *(X, det (a), adj ([green, tie-dyed]*[5]*), noun (T-shirt)*) and *np*$_j$ *(Y, det (a), adj ([blue]), pron (one))*. This SS will allow decomposition of the description (i.e. *green* can be broken off) and the solution of the anaphora will be: *np (Y, det (a), adj ([blue]), noun (T-shirt))*. It is important to remark that the solution will have a different variable[6] (*Y*) than its antecedent (*X*). It means the anaphor and its antecedent do not co-refer, so the anaphor refers to a new entity in the discourse. However in *John bought a* <u>*red dark apple*</u>$_i$ *and a green pear. He ate* <u>*the red one*</u>$_i$, the anaphor will co-refer with *a red dark apple*. We will distinguish both cases

[5] This list of adjectives is provided by the SUG fact *juxtaposition*.

[6] This variable corresponds to the λp of the final logical formula of the constituent (see section 1).

because in the second one the anaphor and its antecedent share the same modifiers[7] (*red*) and they agree in number.

4 Evaluation of the system

We have run our system on part of the previously mentioned corpus (9600 words), and we have got the following figures. Our system has detected 100% of the anaphors described in this paper, and the partial parsing described in Figure 4, has parsed 81% of words with a very simple grammar[8]. The medium length of the sentences with anaphors is 48 words. For pronominal references we have a 83% accuracy in detecting the position of the antecedent. For one-anaphora and surface-count anaphora, we have not got significant figures since there were not so many anaphors as we wished (only 5 anaphors with a 80% accuracy). The reason why some of the references have failed is mainly due to the lack of semantic information and due to the problem of attachments between different parsed constituents[9].

Conclusions

In this paper we have proposed a computational approach to the resolution of pronominal references, surface-count anaphora and one-anaphora. This approach works on the output of a POS tagger, on which we will automatically apply a partial parsing from the formalism: *Slot Unification Grammar*. We have only used lexical, morphologic and syntactic information. We have slightly[10] improved the accuracy (83%) in pronominal references to the work of Kennedy and Boguraev (1996) (75%), but we have also improved that approach since we automatically apply a partial parsing and we deal with other kinds of anaphors.

As a future aim we will include semantic information in our algorithm in order to check the improvement that we get with it. This information will be stored in a dictionary which could be automatically consulted (since this semantic information is not provided by the tagger).

[7] It is obvious that we will probably need more semantic information in order to solve these anaphors, but in this paper we are not going to consider this information since the tagger does not provide it.

[8] We could easily improve this percentage from adding more constituents to the grammar (e.g. adverbs or punctuation marks).

[9] To solve this problem is also necessary semantic information.

[10] It is difficult to compare both measures because we have worked on different texts (Spanish texts).

References

Abney S. (1997) Part-of-Speech Tagging and Partial Parsing. In Steve Young and Gerrit Bloothooft (eds) Corpus-based methods in language and speech processing. Kluwer Academic Publishers

Azzam S. (1995) An Algorithm to Co-Ordinate Anaphor resolution and PPS Disambiguation Process. EACL

Baldwin B. (1997) CogNIAC: high precision coreference with limited knowledge and linguistic resources. ACL/EACL workshop on Operational factors in practical, robust anaphor resolution

Connoly D., Burger J. and Day D. (1994) A Machine learning approach to anaphoric reference. International Conference on New Methods in Language Processing, UMIST

Ferrández A., Palomar M. and Moreno L. (1997a) Slot Unification Grammar. Joint Conference on Declarative Programming. APPIA-GULP-PRODE

Ferrández A., Palomar M. and Moreno L. (1997b) Slot Unificacion Grammar and anaphor resolution. Recent Advances in Natural Language Processing

Kennedy C. and Boguraev B. (1996) Anaphora for Everyone: Pronominal Anaphor resolution without a Parser. COLING

Lappin S. and Leass H. (1994) An algorithm for pronominal anaphor resolution. Computational Linguistics, 20(4)

Mitkov R. (1997) Pronoun resolution: the practical alternative". In S. Botley, T. McEnery (eds) Discourse Anaphora and Anaphor Resolution, Univ. College London Press

Mitkov R. (1995) An uncertainty reasoning approach to anaphor resolution. Natural Language Pacific Rim Symposium. Seoul. Korea

Mitkov R. and Stys M. (1997) Robust reference resolution with limited knowledge: high precision genre-specific approach for English and Polish. Recent Advances in Natural Language Processing