SLOT UNIFICATION GRAMMAR

A. Ferrández
Dept. Languages and Information Systems
ALICANTE UNIVERSITY
antonio@dlsi.ua.es

M.Palomar
Dept. Languages and Information Systems
ALICANTE UNIVERSITY
mpalomar@dlsi.ua.es

L.Moreno
Dept. Information Systems and Computation
VALENCIA UNIVERSITY OF TECHNOLOGY
lmoreno@dsic.upv.es

Keywords: Grammar, coordination, ellipsis, anaphora, Definite Clause Grammar
Postal address: Carretera San Vicente S/N. 03080 ALICANTE, Spain
Telephone number: +34-6-590-3400 ext. 2445 **Fax number:** +34-6-590-3772

Abstract

In this paper we propose a framework which we call *Slot Unification Grammar (SUG)*. This framework allows to integrate different kinds of knowledge, all of them necessary for resolution of several problems in Natural Language Processing (NLP). These problems, such as coordination, extraposition of constituents, ellipsis and anaphora, sometimes occur simultaneously and it is very useful to solve them in a modular way. This framework includes a grammatical formalism which makes easier the solution of these problems. This formalism is based on unification and it is an extension of Definite Clause Grammar (DCG). SUG extends DCG in the following characteristics: (1) SUG allows optional subconstituents in any production rule without adding new nonterminal symbols; (2) SUG solves coordination and juxtaposition by means of its predicates *coordination* and *juxtaposition*. We have implemented SUG in Prolog, extending DCG in the following way: (3) SUG automatically returns a *slot structure* which integrates syntactic, morphologic and semantic information of every constituent of the grammar; (4) SUG will access to the dictionary only once during the whole process of parsing; (5) SUG will generate the logical formula of a sentence after solving the above problems, allowing us to generate very modular NLP systems.

1. Introduction

In the last few years a great number of grammatical formalisms have been developed. The Metamorphosis Grammar (MG) introduced by [1] is the first formalism that allowed to describe grammar rules for syntactical variations of Prolog which make transparent to the user the manipulation of the input string during analysis and includes new characteristics (left side of rules can have more than one logical symbol: nonterminal symbol followed by some terminal symbols). In [11] was developed Definite Clause Grammar (DCG), which is a particular case of MG because left side of rules has only one nonterminal logical symbol.

In this work we propose a logical formalism based on unification, which is an extension of DCG. We call it *Slot Unification Grammar* (SUG). SUG is developed with the aim of extending DCG in order to make easier the resolution of several NLP problems in a modular way. SUG extends DCG in five characteristics which have been enumerated in the previous summary. These new characteristics will be fully developed in the following sections.

2. SUG with regard to DCG

A DCG can be defined as a quadruple *(NT,T,P,S)*, where *NT* is a finite set of nonterminal symbols, *T* is a finite set of terminal symbols disjoint with *NT*, *P* is a finite set of pairs $\alpha \rightarrow \beta$ where $\alpha \in NT$, $\beta \in (T \cup NT)^* \cup \{procedures\ calls\}$, and *S* is the set of initial symbols of the grammar. The elements of *P* are called *production rules.*

SUG adds to the previous definition of DCG that its production rules are noted $\alpha +\!+\!> \beta$, and each subconstituent of β could be omitted in the sentence if it is noted between the operator << and >>. It is a well-known fact that we can get optional constituents in DCG by means of making use of a nonterminal *optA*, with $optA \rightarrow A$ and $optA \rightarrow [\ ]$. However this skill obliges us to add new nonterminal symbols, whereas SUG allows us to get it without adding any new one. Moreover SUG will remind which constituent has been parsed in a sentence and which one not, by means of the final *slot structure* returned by the parser (see section 3.1). This information will be very useful in ellipsis and extraposition resolution. We can get an example from Figure 1, in which we can see the reduction of grammatical rules in SUG.

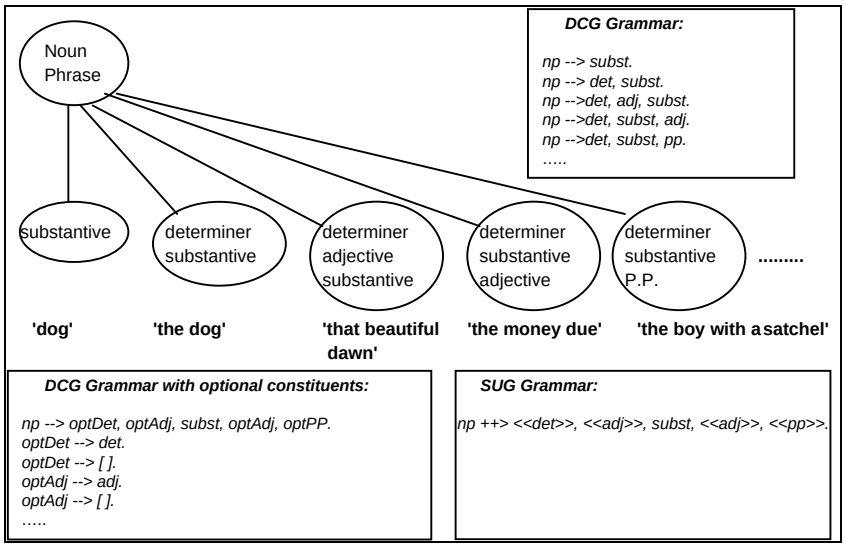

Figure 1: Comparison between DCG and SUG: optional constituents.

As well as SUG production rules such as α*+++>*β, SUG adds what we call SUG facts. These SUG facts have only the first member of the production rule, i.e. α, and α*'s* name is either *coordination, juxtaposition, fusion, basicWord* or *isWord*.

The fact *coordination(A,SimpleA)* is used to solve coordination of constituents. It would be equivalent to DCG rules *A→simpleA,conjunction,A* and *A→simpleA*. There is another sort of coordination which occurs without conjunctions. It is called *juxtaposition* and is implemented in SUG by means of the fact *juxtaposition(A,SimpleA)*. This fact would be equivalent to DCG rules *A→simpleA,A* and *A→simpleA*. We can get an example from adjectives. A noun can take an indefinite number of adjectives in front of it, as long as the meanings add up to something coherent: *That huge dark greyish cat is nice*. Here we will apply the juxtaposition rule for adjectives shown in Figure 3 lines 36-39. Both facts will store syntactical structure of coordinated constituents in the final slot structure which will allow us to establish relations of parallelism between them. They will also allow us to coordinate constituents of different forms as it is shown in Figure 3 lines 8-23 in the coordination of noun phrases of pronoun, verbal and substantive type.

We can also reduce the number of rules by means of SUG fact *fusion(rule1, rule2, …, ruleN)*. With this fact, all these $rule_i$ with common constituents will be merged in only one rule. We can get an example from Figure 2. In this way, when we parse the noun phrase of verbal type, we will parse only once the gerund verb and after that we will try to parse either a *np* or a *pp*. In this way, we can improve the efficiency of the grammar, since *verb(gerund)* will be parsed only once, and the final slot structure will tell us what has been parsed and what not.

```
fusion(
   (np(verbalType)++>verb(gerund), <<np(substType)>>),
   (np(verbalType)++>verb(gerund), <<pp>>)
).
```

Result of the merging:

np(verbalType) + + > verb(gerund), { << np(substType) >> , << pp >> }

Figure 2: SUG predicate "fusion".

We can see a grammar in SUG notation in Figure 3, which will be used in the following examples of this paper.

```
%------------------ SENTENCE ------------------------------------------------------------------ % Initial symbol
coordination(                                                          % "I wrote a letter but she didn't post it"
        sentence,                        % It's equivalent to: "sentence → simpleSentence,conjunction,sentence"
        simpleSentence
).
simpleSentence ++> nounPhrase(Number), verbalPhrase(Number) )                    % ++> means SUG rule
%------------------ NOUN PHRASE ---------------------------------------------------------
coordination(                    % Coordination of NPs of different forms: "She, Peter and the black boy ..."
        nounPhrase(_,_),                 % nounPhrase→ simpleNounPhrase,conjunction,nounPhrase
        simpleNounPhrase(_,_)            % The arguments mean that can be coordinated noun phrases
).                                       % of different forms and numbers such as "she and those boys"
simpleNounPhrase(Number, substantiveType) ++>                          % NP of substantive type
        <<determiner(Number)>>,                                        % Optional constituent
        <<adjective>>,
        substantive(Number),                                           % Obligatory constituent
        <<adjective>>,
        <<prepositionalPhrase>>.
simpleNounPhrase(Number, pronounType) ++>                              % NP of pronoun type
        pronoun(Number).                 % Obligatory constituent. The main word is a pronoun
fusion(                                                                % It merges these rules in only one rule
  (simpleNounPhrase(Number, verbalType)++>verb(gerund), <<np(substType,Number)>>),
  (simpleNounPhrase(Number, verbalType)++>verb(gerund), <<pp>>)
).
%------------------ PREPOSITIONAL PHRASE ------------------------------------------
coordination(                                                          % "for Peter and for John"
        prepositionalPhrase,             % It's equivalent to: "pp → simplePP,conjunction,PP"
        simplePrepositionalPhrase
).
simplePrepositionalPhrase ++>
        <<adverb(negation)>>, preposition, nounPhrase(_, _).           % "for she and not for Peter"
%------------------ VERBAL PHRASE -------------------------------------------------------
verbalPhrase(Number) ++>
        verb(Number),                                                  % Obligatory constituent
        <<nounPhrase(_,_)>>, <<prepositionalPhrase>>.                  % Optional constituents
%------------------ ADJECTIVES ----------------------------------------------------------
juxtaposition(                                                         % "That huge dark greyish cat is nice"
        adjective,                       % It's equivalent to: "adjective→simpleAdjective,adjective"
        simpleAdjective
).
%------------------ BASIC CONSTITUENTS --------------------------------------------------
basicWord( preposition ).                                              % Prepositions
basicWord( determiner(Number) ).                                       % Determiners
basicWord( conjunction ).                                              % Conjunctions
basicWord( pronoun(Number) ).                                          % Pronouns
basicWord( substantive(Number) ).                                      % Substantives
basicWord( verb(Number) ).                                             % Verbs
basicWord( simpleAdjective ).                                          % Adjectives
basicWord( adverb(Type) ).                                             % Adverb
%------------------ DICTIONARY ----------------------------------------------------------
isWord('mine',[pronoun(singular), subst(singular)]).
isWord('the', [determiner(singular)]).
isWord('black', [adjective]).
isWord('boy', [substantive(singular)]).
isWord('not', [adverb(negation)]).
isWord('for', [preposition]).
...
```

Figure 3: An example of a grammar in SUG notation with "sentence" as initial symbol.

The SUG fact *basicWord(constituent(arg1, arg2, …, argN))* allows us to define the basic constituents of the grammar, in which *constituent* is augmented with so many arguments as it is needed (e.g. *Number* or *Type*). This fact would be equivalent to DCG rule *constituent(arg1, arg2, …, argN)→[Word]*, but it is also used in SUG in order to add all morphological and lexical information in the final *slot structure* of the constituent. This fact is related with *isWord(word, listOfLexicalEntriesOfWord)* which will be fully explained in section 3.2.

Finally we would like to make a brief remark about the resemblance (in their names) between *Slot Unification Grammars (SUG)* and *Slot Grammars (SG)* [7]. SUG is quite different to SG since SG is a lexicalist grammar whereas SUG is not.

3. Implementing SUG

3.1 Slot structure returned by the parser

In previous sections we have described SUG like an extension of DCG. There are translators which turns DCG rules into Prolog clauses by adding two extra arguments to every symbol that is not in brackets or braces. Each DCG production rule such as $s \rightarrow np\ vp$, is translated into a Prolog clause for a predicate with two arguments, like this: *s(L1,L3) :- np(L1,L2), vp(L2,L3)*, where *L1*, *L2* and *L3* are the lists of words of the remaining sentence to parse.

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 like *s(Number) ++> np(Number) vp(Number)*. This translation will add three extra arguments to every symbol that is not in brackets or braces: *s(Number,SSs,L1,L3) :- np(Number,SSnp,L1,L2) vp(Number,SSvp,L2,L3)*. The last two extra arguments corresponds to the list of words to parse in a similar way that in DCG. The first one corresponds to what we call *slot structure (SS)*.

This *SS* stores the syntactic, morphologic and semantic information of every constituent of the grammar. All this information is automatically stored as it is shown in Figure 4. 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 (morphologic and semantic information: *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 variables ("_") the slots corresponding to the optional constituents that do not appear in the sentence, in this way, we can know what has been parsed and what not. Sometimes is very useful to know what has not been parsed, like in ellipsis resolution (Figure 13) or extraposition of constituents. The semantic information corresponds to semantic constraints which can be included in a syntactic parser. We have applied IRSAS method [9, 10] in order to incorporate these semantic constraints in parsing. From now on we will show each SS with λp and *conc* only if it is necessary, in order to get simplicity.

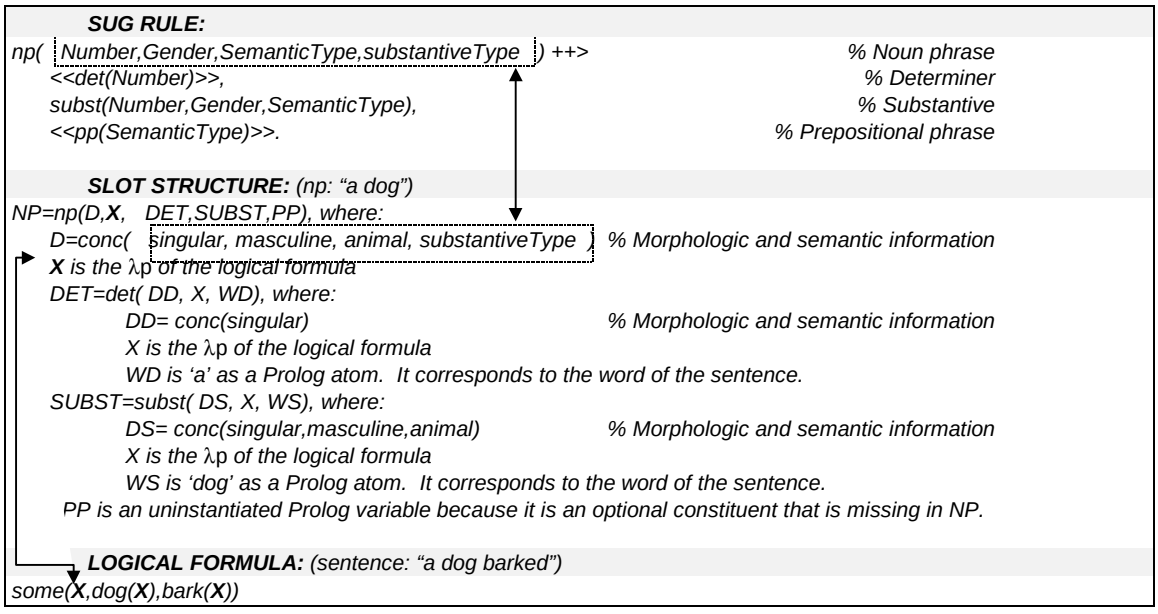

Figure 4: Slot structure returned by parsing "a dog".

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 problems such as extraposition of constituents, ellipsis [4, 11] and anaphora [4]. 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 5.

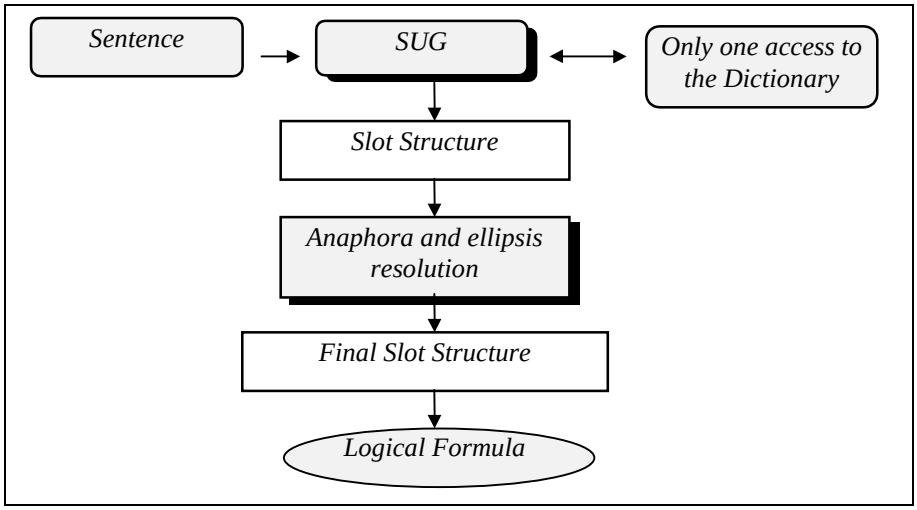

Figure 5

We would like to emphasize that this skill of resolution allows us to produce very modular NLP systems in which grammatical rules and logical formulas are quite independent of the module of resolution of ellipsis and anaphora.

3.2 Reducing the times we access the dictionary

We store the lexical entries of the words in the dictionary. It is obvious that the dictionary will have a great size in a real application of natural language processing, so it is useful to optimize its access. Moreover, during backtracking points we have repeated access to the same word from the dictionary.

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 as it is shown in Figure 6, and it will work with this structure instead of the list of words of DCG parser. That is to say, 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.

Each element from the 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.

The dictionary has been implemented in Prolog through SUG predicate *isWord(word, listOfLexicalEntriesOfWord)* in order to create this structure efficiently. This predicate has two arguments. The first one refers to the word as a Prolog atom, and the second one to the list of lexical entries. There is just one clause for each word as we can see in Figure 3 lines 50-55.

This list of words allows SUG parser to apply a look-ahead search in each node of the backtracking tree. We could apply the following *pruning algorithm*: "when a node of the backtracking tree has more than one daughter, instead of visiting the first one, we will analyse the following words in the sentence, and we will decide on the only subtree that we will visit depending on the type of these words". It is possible to implement this pruning algorithm by means of looking at the list of words, instead of repeated accesses to the dictionary which could lead us to an inefficiency. We can see an example in Figure 7. If we are parsing the verbal phrase in *He speaks to his mother*, we will know in advance that there is not a noun phrase because the word after the verb is a preposition (*to*). Thus we can reduce the path of the tree as it is shown in this figure.

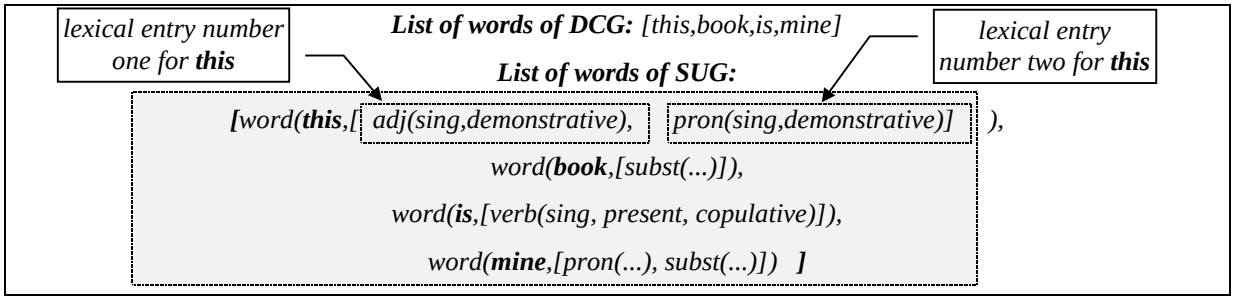

Figure 6: List of words of SUG parser in "This book is mine"

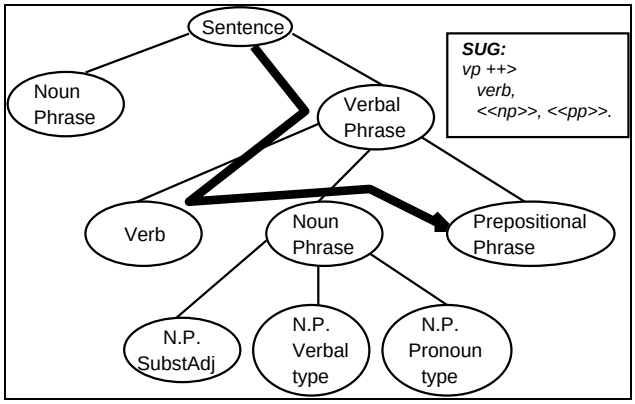

Figure 7: Parsing the vp "…speaks to his mother".

3.3 Evaluation of the system

We have implemented a SUG spanish grammar. This grammar includes coordination of sentences, noun phrases, prepositional phrases, adjectives and prepositions. It also includes juxtaposition of adjectives. Noun phrases have been divided into several categories: substantive type, adjective type, verbal type and pronoun type. We have used arguments in constituents of the grammar to enforce number, person and gender agreement. To sum up, this grammar involves a wide range of grammatical sentences in which we have tested the efficiency of the framework proposed in this work.

We have got the results that are shown from Figure 8 to Figure 11. In all these figures we have analysed efficiency of SUG. For this analysis we have increased gradually the number of coordinated constituents. For example, initially we parse the sentence *"John said it to Susan and Peter said it to me"* and after that we parse *"John and Sally said it to Susan, Peter said it to me and John said it to him"*, and so on. We can get from these results a good relation of complexity of parsing in relation with the size of the grammar, about n^2, with n the number of words of the sentence. We should point out that the time has been taken in hundredths of second, and the order of complexity ($n^{Complexity}$) has been got from the formula: *Complexity=(log(T1/T0) / (log(N1/N0))*, with *T0* the time of parsing for a sentence with *N0* words.

In Figure 8 we can see the average percentages of time involved in the parsing of constituents. In Figure 10 we have added a conjunction at the last of each sentence in order to parse grammatically wrong sentences, for example the sentence *"John and Sally said it to Susan and"*. In these sentences we have tried that the parser have to search through every backtracking point. This case produces a big deal of ambiguity in the parsing, since that the conjunction *"and"* might mean a new coordinated constituent. In Figure 11 we have parsed the same sentences that in Figure 10 but without the final conjunction, so they are gramatically correct sentences, for example *"John and Sally said it to Susan"*.

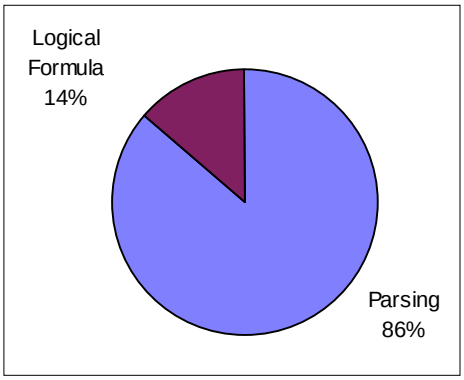

Figure 8

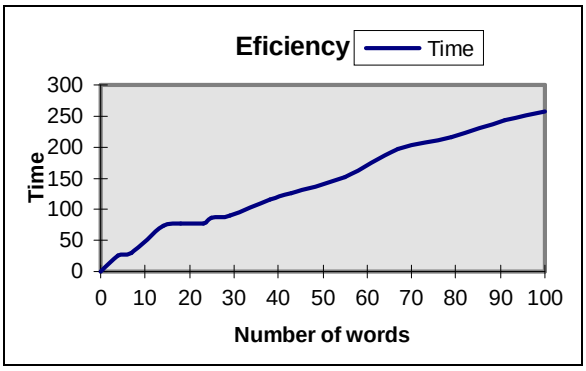

Figure 9: Sentences in Figure 10.

Words	*Time*	*log(T1/T0)*	*log(N1/N0)*	*Complexity*
4	25,25	0,14705342	0,55961579	0,26
7	29,25	0,92818552	0,69314718	0,92
14	74	0,03648829	0,25131443	0,03
18	76,75	0,00325204	0,24512246	0,0032
23	77	0,11344463	0,08338161	1,36
25	86,25	0,05085842	0,14842001	0,34
29	90,75	0,2497828	0,27029033	0,92
38	116,5	0,0803728	0,12361396	0,65
43	126,25	0,18890053	0,24613307	0,76
55	152,5	0,25349784	0,19735943	1,28
67	196,5	0,09808218	0,16475523	0,59
79	216,75	0,11431683	0,14141165	0,80
91	243	0,05601464	0,09431068	0,59
100	257			

Figure 10: Gramatically wrong sentences.

Words	*Time*	*log(T1/T0)*	*log(N1/N0)*	*Complexity*
3	43,75	-0,32059889	0,69314718	-0,46
6	31,75	0,38692153	0,77318989	0,50
13	46,75	0,30228087	0,26826399	1,12
17	63,25	0,10496518	0,25782911	0,40
22	70,25	0,23657606	0,08701138	2,71
24	89	0,07571182	0,15415068	0,49
28	96	0,17622663	0,2787134	0,63
37	114,5	0,09966749	0,12675171	0,78
42	126,5	0,35825472	0,25131443	1,42
54	181	0,21314902	0,2006707	1,06
66	224	0,34409616	0,16705408	2,05
78	316	0,14552407	0,14310084	1,01
90	365,5	0,21903114	0,09531018	2,29
99	455			

Figure 11: Gramatically correct sentences.

4. Some examples of anaphora and ellipsis resolution

In this section we will show some examples of the application of SUG to anaphora and ellipsis resolution. It is important to remark that we are not going to focus on generalizing the resolution of anaphora and ellipsis with

SUG. We just intend to show how this framework can be applied to some kinds of anaphora and ellipsis resolution.

We can apply SUG to resolution of anaphora problem since its slot structure (SS) offers a data structure which includes all necessary information to solve a reference. In addition to syntactic, morphologic and semantic information, it also allows to establishes a relation of parallelism between constituents. That parallelism is particularly important in resolving *surface count anaphora* [5] (*the former, the latter, the first, the second, ...*), which requires that the surface structure of the sentence (or at least the order of possible referents) be retained. We can get an example from Figure 12, in which we solve anaphors by means of the SS returned from the coordination of sentences and noun phrases. This SS allows us to solve anaphor *she* due to morphologic agreement (by means of unification of structure *conc* of both noun phrases) and syntactic parallelism between the subjects of both coordinated sentences. It will also solve *the former* since the SS of coordination of noun phrases allows us to access whatever coordinated noun phrase in the order we wish. In this example we use SUG rules in Figure 3 lines 2-5 and 8-11. It is easy to generalize this parallelism to intersentential cases in order to solve *discourse anaphora*, since this SS allows us to establish relations of parallelism between different sentences.

The issue that anaphora resolution can be treated in a similar way that ellipsis is very important in situations where both problems occur simultaneously, e.g. *VP-anaphora*, or when the anaphor is completely null: *Ross$_i$ carefully folded his trousers and Ø$_i$ climbed into bed*. We can have a look at the example in Figure 13. It is easy to see that VP-anaphora occurs in the second sentence since its verbal phrase consists of a pro-verb (*do*) with no complement and a modifier (*too*). We have previously mentioned that the parser leaves as uninstantiated variables the slots corresponding to the optional constituents that do not appear in the sentence. That is what occurs in this figure, where *DO_2* is missing whereas in the first sentence does not (*DO_1*). Later on, we can recover the missing constituents by unification of their variables, e.g. *DO_2=DO_1*.

But if we look at this example carefully, we can see that there is an ambiguity called the *sloppy identity problem*. This ambiguity makes that two readings can be derived: *λp kiss(p, wifeOf(Jack))* or *λp kiss(p, wifeOf(p))*, where *p=Sam*. SUG foresees these cases by means of its SS. In every SS of a constituent, SUG adds a variable which corresponds with *λp* of the logical formula (Figure 4). When we are solving ellipsis, by means of unifying the variables shown in Figure 13 (*DO_2= DO_1*), we can decide between also unify both *λp* or not, that is the way we can decide between the two readings above.

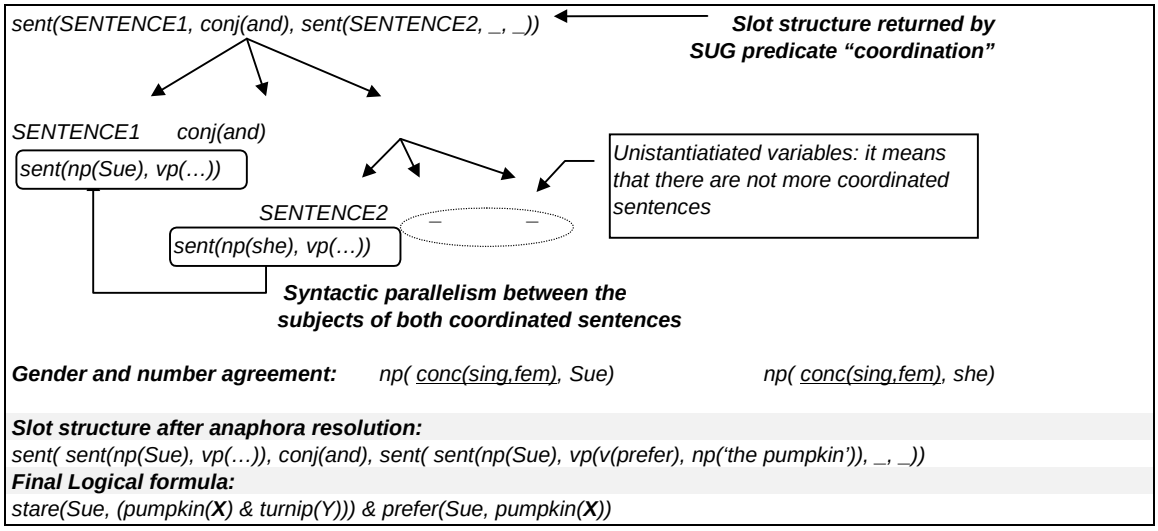

Figure 12: "Sue$_i$ stared at the pumpkin$_j$ and the turnip, and she$_i$ preferred the former$_j$"

It is important to remark that this handling of ellipsis is also able to reject ungrammatical sentences. For example, in *The man$_i$ devoured Ø$_j$ and Ø$_i$ finished the cake$_j$* obliged us to allow for all complements to be optional, and in the ellipsis resolution process, the missing complements will be got by unification. But we are also able to reject *The man devoured* where *devoured* is a transitive verb, since we will check that obligatory complements after ellipsis resolution and before the logical formula is obtained. We can get it by means of Prolog predicates *TypeOfVerb==transitive*, *var(DirectObject)*, *var(IndirectObject)*, where if it successes, SUG parser will reject the sentence (*TypeOfVerb* will be obtained from the structure *conc* of the slot structure of the verb; *DirectObject* and *IndirectObject* are the slot structures of the verb complements).

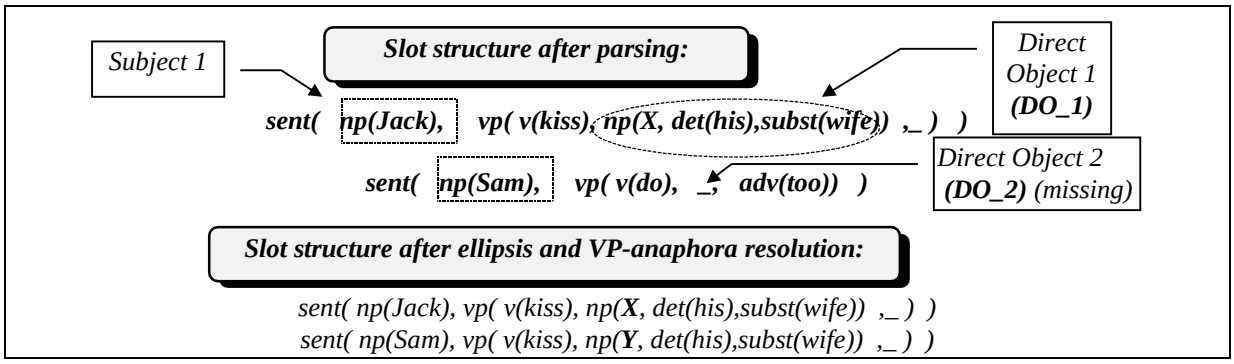

Figure 13: Solving ellipsis and VP-anaphora in "Jack kissed his wife. Sam did too"

5. Conclusions

In this paper we have proposed a framework which integrates different kinds of knowledge, all of them necessary for resolution of several Natural Language Processing problems. This framework, *SUG*, is an extension of *DCGs*, with a reduction in the number of grammatical rules. We have implemented a parser of this grammar in Prolog. This parser automatically returns an *slot structure*, which stores all the necessary information to solve coordination, anaphora, ellipsis and extraposition in a very modular way.

As a future aim we intend to generalize our approach to ellipsis and anaphora resolution by using our grammar in a similar way that the algorithm proposed in [6]. This algorithm does not require in-depth, full, syntactic parsing of text. We could implement it by means of SUG grammar in Figure 14. This grammar would parse only prepositional phrases (*pp*), noun phrases (*np*) and verbs (*verb*) in an unrestricted text. The SS returned by the parser will consist of a sequence of these constituents. Figure 14 is using some features of SUG that we have not previously mentioned in this paper, such as the operator << ***SSA:****a* >>. SUG predicate <<*a*>> has already been mentioned in section 2. Here, this predicate has been added the possibility of accessing to its SS, *SSA*, which will be an uninstantiated Prolog variable if constituent *a* is missing (it will success Prolog predicate *var(SSA)*). Operator <# *a*, *b* #> allows us to run predicate *a*, and if it does not success, then it will be run *b*. Operator <## *a*, *b* ##> is similar but it will also success when *a* and *b* fail.

```
sentence ++>                                          % We will parse sentences
   << PP:pp >>, << NP:np >>, << V:verb >>,            % Constituents to parse
   <# [ ],                                            % It means that there is no more sentences left
      remainingSentence(PP,NP,V)  #> .
remainingSentence(PP,NP,V) ++>
   <## ( {( var(PP), var(NP), var(V))},   [W]),       % It means that W is neither PP,NP nor V,
         ( _                          ,    _ ) ##>,  % so we parse it as a free word
    sentence.
pp ++> preposition, np.                               % Grammar rules for each constituent to parse
…
```

Figure 14

References

[1] Colmerauer, A. *"Metamorphosis Grammars"*. Informe Interno (Univ. d'Aix-Marseille II). 1975

[2] Covington, M. *"Natural Language Processing for Prolog Programmers"*. Prentice Hall. 1994

[3] Ferrández, A.; Moreno, L.; Palomar, M. *"Un formalismo para el tratamiento gramatical de la coordinación: Gramática de Unificación de Huecos"*. Novatica, 115. 1995

[4] Ferrández, A.; Palomar, M.; Moreno, L. *"Slot Unification Grammar and anaphora resolution"*. Submitted to ACL'97 Workshop on anaphora resolution. 1997

[5] Hirst, G.*"Anaphora in Natural Language Understanding"*.Springer-Verlag. 1981

[6] Kennedy, C.; Boguraev, B. *"Anaphora for Everyone: Pronominal Anaphora Resolution without a Parser"*. Proc. 16th Int. Conf. on Computational Linguistics, COLING, Vol. I, 113-118, Copenhagen, Denmark. 1996

[7] McCord, M.C. *"Slot Grammar: a system for simpler construction of practical natural language grammars"*.Natural Language and Logic, International Scientific Symposium, edited by R.Studer. Lecture Notes in Computer Science. Springer Verlag. 1990

[8] Moreno, L.; Andrés, F.; Palomar, M. *"Incorporar Restricciones Semánticas en el Análisis Sintáctico: IRSAS"*. Procesamiento del Lenguaje Natural n.12. 1992.

[9] Moreno, L.; Palomar, M. *"Semantic Constraints in a Syntactic Parser: Queries-Answering to Database"*. Database and Expert Systems Applications. Springer-Verlag. 1992.

[10] Palomar, M.; Ferrández, A.; Moreno, L. *"Aportaciones a la Resolución de la elipsis en la coordinación"*. Procesamiento del Lenguaje Natural, nº16, 1995.

[11] Pereira, F.; Warren, D. *"Definite Clause Grammars for Language Analysis- A Survey of de Formalism and a Comparison with Augmented Transition Networks"*. Artificial Intelligence, vol. 13. 1980