An HPSG Grammar of Spanish Implemented in ALE: A Solution for Dealing with Subcategorisation Alternances

Salvador Climent
Xavier Farreres

Report LSI–95–33–R

AN HPSG GRAMMAR OF SPANISH IMPLEMENTED IN ALE: A SOLUTION FOR DEALING WITH SUBCATEGORISATION ALTERNANCES

Salvador Climent **Xavier Farreres**

climent@goliat.upc.es **farreres@lsi.upc.es**

Departament de Llenguatges i Sistemes Informàtics.

Facultat d'Informàtica.

Universitat Politècnica de Catalunya.

November, 1994.

Abstract.

A core of a grammar of Spanish in HPSG including a solution to deal with verbal subcategorisation alternances avoiding to list different senses in the lexicon is built in ALE. Computational efficacy of HPSG and capabilities of ALE are tested by the way.

Contents

1 Introduction[1]

The main target in this work has been to design a significant syntactic core of the grammar of Spanish according to the HPSG (Head-Driven Phrase Structure Grammar) theory developed by [Pollard & Sag 87,93], and to implement it in Robert Carpenter's ALE (Attribute Logic Engine) formalism. HPSG and ALE are presented (§2, §3); the grounds of the grammar here developed are discussed (§4); conclusions after having test HPSG and ALE are offered (§5); and finally ALE's code for a type structure intended to fully support HPSG and a grammar handling a wide range of assertive sentences of Spanish is displayed (Appendices).

In building the grammar we have developed a system to avoid proliferation of verbal lexical entries due to different subcategorisation patterns. It is well known that word meaning is modelled by the context, probably in a systematic way which is still unexplained by linguistic theory. Verbs, in particular, usually take different sets of complement arguments; each one of these alternative surface realisations provides a tinge on the remaining basic meaning of the verb.

The usual way of facing this phenomenon is posing a different lexical entry for every subcategorisation pattern of the verb (e.g., see [Briscoe & Copestake 91]). We think this is not a completely achieved solution, as long as (i) it enlarges the lexicon, which is already a complex object in grammar theories as HPSG; and (ii) fails to capture the speaker's ability of using the same word in different contexts, thus modelling its meaning. Consequently, we think lexical entries should owe a structure able to support those generalisations.

A solution to do that, restricted to the syntax of some types of verbs, has been developed: what we do is taking as a basic sense that realisation with the minimal subcategorisation pattern –which is considered as "obligatory" for that verb– and treating further complements as optative. This approach is discussed in §4.

Under the main goal of building a grammar lay two collateral aims: (a) to explore the computational efficacy of pure HPSG (not of a unification grammar similar to HPSG); and (b), to test ALE and its capacities for developing and handling a grammar of this kind. Both are discussed along the paper and our conclusions with respect to it summarised in §5.

2 HPSG

HPSG has progressively become a reference point in recent NLP (Natural Language Processing) research. It is a unification grammar, based in complex features selected from a well-defined and structured set of features and values, based in the surface of the phrasal chain, inductive, declarative, and informational.

Being a unification grammar, HPSG arranges and combines feature structures (FS) by the operation of unification and the relation of subsumption; consequently, the well-formation of phrases is understood as, and depends on, FS compatibility. HPSG's FSs admit logic operations, can be grouped in lists or sets, and can bear values which are consequence of a function[2].

The final aim of an HPSG parse is merging in one only FS all the information (phonology, morphology, syntax, semantics, pragmatics) carried by a natural language chain.

[1] We would like to thank Dr. Horacio Rodriguez for many helpful comments and and corrections on a draft version of this report. Nevertheless, of course we take sole responsibility for all remaining errors and omissions.

[2] E.g., the value of the phonological structure of a phrase, <PHON>, is the result of a function on the <PHON> values of the words which compose it.

The components of an HPSG grammar are: (lexical or phrasal) signs, rules combining signs; (universal or language-dependent) principles affecting every rule application, (language-dependent) rules on phrasal surface order, and lexical rules.

Main information in HPSG is encoded in the lexicon, whereas in other syntactic theories (as GB or GPSG) rules handle most of the work. It is basic in HPSGs philosophy that a big bunch of a well-formed sentences of the language might be accounted by a very small number of combinatory schemata -on which an as well limited set of principles must be accomplished. This is one of the points which has contributed the more to make HPSG become a very attractive formalism for NLP applications. The price to pay is, obviously, a lexical component highly complex and structured -what anyway is in tune with the lexicalism increasingly setting in the current lines of theoretic research in NLP.

2.1 Signs

The sign in HPSG, as we have pointed out, might be lexical or phrasal. It is described by means of FSs -usually represented by directed acyclic graphs. The sharing of the same structure as a value of different attributes is allowed, and in fact it is a fundamental description mechanism. Features are well-defined types of a hierarchical structure where both subsumption and monotonic multiple inheritance do operate.

Lexical and phrasal signs basic structures are identical, except for the feature <DTRS>[3] which is owned only by the last one. This feature reflects the process of phrasal sign combination which has brought about the sign actually described.

Every <DTRS> feature is in turn described by the features <HEAD_DTR> (head) and <COMP_DTR> (complements), which in turn are complex features that, if they themselves are phrasal, bear a <DTRS> feature which describe they own compositional process. It's to be noticed that (a) the values of <HEAD_DTR> and <COMP_DTR> are fulfilled signs, not just a simplified expression of their category; and (b) the representation denoted by <DTRS> is surface-order independent.

Every lexical sign is described by the features <PHON>, <SYNSEM> and <Q_STORE>. Phrasal signs add the feature <DTRS> to the previous three.

<PHON> contains information about the phonological form of the sign[4] and <Q_STORE> about quantifiers operating and their scope. <SYNSEM> is the most complex sign of the pack and contains syntactic (<CAT>); logical-semantics and concordance (<CONT>) and pragmatic (<CONTEXT>) information.

In its original formulation HPSG described <CAT> by means of the features <HEAD> (denoting the grammatical category of the sign's head) and <SUBCAT>, whose value was a list of <SYNSEM> features ordered according to the traditional criterion of obliquity. <SUBCAT> lists the simple or complex categories subcategorised by the described sign -after the style of categories cancellation in Categorial Grammar. Thus, a common_noun sign would bear as <SUBCAT> value a determiner or a quantifier; a transitive_verb sign would bear a NP in nominative and a NP in accusative: and a VP sign would bear as <SUBCAT> a NP in nominative. But theoretical problems of different kinds which one can generalise by the certainly diverse nature of phenomena supposedly feasible by <SUBCAT>[5] led to detach that one in a set of features specialised for every different kind of subcategorisation: <SUBJ> (selection of subjects); <COMPS> (of complements, e.g. compulsory complements of the verb or the NP as a complement of a preposition in a PP); <ADJ> (of adjuncts or optional elements); <MODS> (selectional restrictions of non-head elements, e.g., the nouns selected by adjectives) and <SPEC>

[3] for "daughters".

[4] In our implementation, in fact orthographic form.

[5] Intuitively, one can see the differences existent between the selection of a determiner by the noun, that of a noun by an adjective, that of an object by a verb, or that of a subject by the same verb (or by the VP?).

(specifiers, e.g. the determiner for a noun).

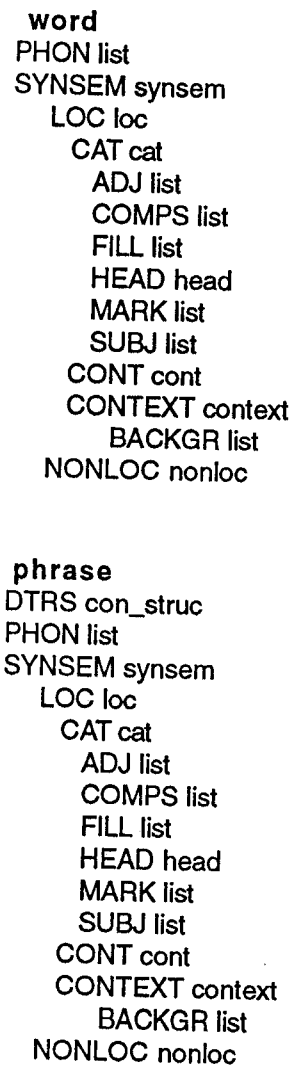


fig.1 General specifications for both a lexical and a phrasal sign.

2.2 Principles

As we said above, HPSG posits a set of principles of global effect on every rule of the grammar. The existence of both Universal and Language-Dependent Principles is supposed. Main Universal Principles are Head-Feature Principle (HFP) and Subcat Principle (SP).

HFP, following the generativist tradition, posits that phrases are projections of their heads; consequently it declares that the head (<HEAD>) of every phrasal sign shares structure with the head of its <HEAD_DTR>.

SP stands for the cancellation of categories when constructing superior phrase nodes from its components. It states that the value of the <SUBCAT> list of a mother node is equal to the <SUBCAT> list of its head-daughter minus those elements of the list corresponding to its non-head daughters. As long as, as we have said before, in later HPSG developments the <SUBCAT> feature has been detached in several other features, SP has been forced to an adaptation to the new structure arrangement; but anyway the underlying idea keeps unchanged.

Other Principles are posited for other linguistic aspects such as semantics, quantification, etc. We don't present them here for their explanation would become too long-winded and we have not made use of them in our grammar either.

2.3 Rules.

HPSG makes use of three sorts of rules: Lexical Rules (LR), Lineal Precedence Rules (LPR) and Immediate Dominance Schemata (IDS).

LRs are generalisations on the Lexicon intended to avoid redundancy in its development and maintenance. Typical examples for LRs are those of morphological derivation from a word's lemma.

LPRs establish phrasal surface order. IDS, otherwise, are surface-order-independent; they are not rewriting rules but constituent well-formation schemata, i.e., each one declares a set of conditions which have to be achieved when combining certain kinds of signs; otherwise one can not say that this combination is a well-formed phrase of the language.

It is important to get the idea that every IDS is not aimed to combine specific signs, but kinds of signs which share certain common characterisations. In IDSs descriptions of both the signs to be combined and the resultant sign are partial and underspecified. The final determination of the built-in sign is a result of the interaction of IDSs with the Principles and the Type Structure, plus the action on all them of the unification mechanism.

Consequently, for instance, the original formulation of HPSG's IDS-1 (represented in **fig. 2**) stands for the combination of a non lexical sign whose <SUBCAT> list bears one only element with precisely this subcategorised sign, giving as a result -the *mother* node- a non lexical sign bearing the empty list as a value for <SUBCAT> and its value for <HEAD> structure-shared the <HEAD> value of its head-daughter (the sign firstly described). So, this schema allows for the combination of pairs such as a subject-NP and a VP to make out a Sentence, or also a Determiner and a Nominal Group to make out a NP.

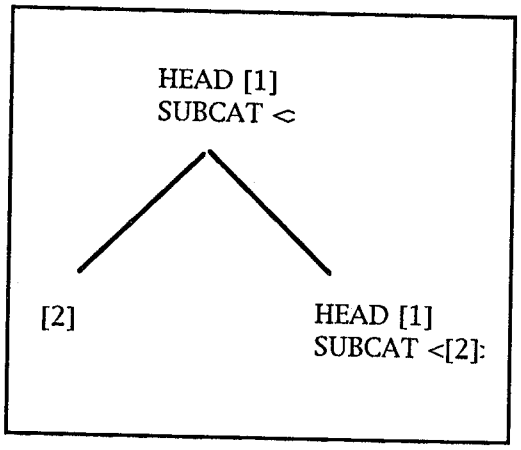


fig. 2: IDS-1

3 ALE

ALE (Attribute Logic Engine) is a computer language for natural language parsing. It is strongly typed and based on a type system with multiple monotonic inheritance. ALE's simplest elements are Feature Structures. The main difference between ALE and other feature-

based formalisms are:

(i) every feature structure belongs to a predefined type; and

(ii) every feature structure must be well typed in two senses: only appropriate features for the type can be included in the feature structure; and only feature structures belonging to types appropriate for a feature can be associated as values to that feature.

Even though it allows for multiple inheritance, it poses some restrictions to the type structure in order to make this type of inheritance controllable: on one side, there must be a unique common daughter for each group of parents who share more than one daughter; on the other side, each feature name can only be introduced once in one type in the hierarchy, and only their daughters have access to it's definition. The first restriction introduces extra types in the system, as we have to introduce an extra type where multiple inheritance exists, and all inheritance paths must cross there, but it is the only way how every two types which are unified have their unification pertaining to a type in the system. The second eliminates the need of electing a type to inherit a feature from. With these two restrictions the system can also be compiled, and then the speed highly increases. A nice thing is the concept of appropriateness, which carries the notion of which attributes are appropriate for a type. This way we know forehand if two types are compatible and unification will be successful by only looking at the attributes involved, and we can deduce immediately the final type of the unification process. This, added to the compiled type system saves parse time, and detects errors hastily and evades incisor unifications.

The type system is thus the most important component of the system, as all other components have to restrict their definitions to the types defined. A type in ALE is defined as follows

```
                    :
type    sub     [subtype1,...,subtypeN]
        intro   [feature1:type1,...,featureM:typeM].
```

The **sub** part defines the subtypes, and the **intro** part defines the definition of the structure. If a type doesn't have subtypes, there is nothing between the brackets. A type can be subtype of more than one type. A feature can be introduced in this type or in a parent type; if not, it's an erroneous feature. The **type** after the colon define the possible values for the feature. These types *must* be defined within the type system. Restrictions ensure that the unification of two types will in turn be a type in the system.

In ALE macros, lexical rules, grammar rules, lexical entries and logic predicates may be defined. Macros are expanded when called, and they are a way to save code. Lexical rules are used to take lexical entries and generate new derived lexical entries; this means only a small lexicon has to be defined, and then expanded via lexical rules. The grammar rules are of the rewriting type, this means, they take some components and rewrite them as result into another more complex one. Lexical entries are defined apart from the type system, in order to keep them stored with an efficient access method. It is also possible to include logic predicates Prolog-like, which can be introduced in any point in the rules, to do some extra tests.

The reason for chosing ALE instead of other similar systems, is the adecuation of this system to HPSG requirements. HPSH define all entries as restrited to one particular type, and ALE was the only one known system that alowed to do this. Not only this, but this restriction of features to one type, as the work done by Carpenter on well-typed feature systems points out, allows for the elimination of extra analyses. This added to the restrictivity HPSG applies to the analysis, results in a very good performance. Of course, there have been problems in encoding the principles, but ALE allows the insertion of prolog code thus making of ALE an open system, and this has given us various options at the time of deciding on the way of encoding them.

4 The grammar

The grammar built here is constrained to syntax and focused on phrasal structures caused by different sorts of verbs on assertive active sentences. Notwithstanding, the type system implemented is wide enough to support further grammar's extensions with both semantics and other syntactic phenomena -as questions, long distance dependencies etc. A quick look on the type system provided in Appendix-1 shall make the reader notice that practically every type supplied by the fundamentals chapters in [Pollard & Sag 87, 93] stand properly in our type system[6].

We also develop a sortal for synsem types intended to encode verbal polisemy. Usual approaches to lexical representation describe the meaning of a verb by enumerating every sense, corresponding every sense to a subcategorisation context. We thought this practice fails to properly represent lexical knowledge as long as, although showing different diatheses alternations, a verb (usually) keeps its basic conceptual meaning across each one of these senses. Thus we believe that solutions have to be found to maintain this regularity and prevent for lexical over-representation.

4.1 Type System and Verb Subcategorisation

The basic HPSG type system has been necessarily extended with types depending on the verbal sorts we have posited. Consequently, a bizarre branching off of the types <SYNSEM>, <LOC> and <CAT> is produced. We could have put the same information on verbs without complicating the type system by coding it in ALE's macros, but we thought that that way both we were walking off of HPSG's spirit and we were going to retrieve less informational results. In addition, as we said above, in ALE the advantages of putting information in the type system instead of in other modules are diverse -being the most relevant time saving access during parsing and pre-detection of inconsistencies.

We have structured our grammar around six types of verbs. Our basic aim was respecting the fundamental HPSG's target of accounting for many syntactic structures with a very little set of rules. This goal has been achieved as long as our grammar consists of only three rules, which allow for quite diverse sentence formations, as long as for building NPs and PPs. The strong encoding of verb sorts in the type structure permits unification to do the work of correct sentence formation without proliferation of rules.

In addition to that, as we have already pointed out, we have worked with the idea of every verb belonging to a single verbal type. It is known that verbs show different surface syntactic alternations which usually result in different entries in the lexicon as long as each one asks for different subcategorisation specifications. See the examples under (1):

(1)	*(a)*Juan comía.	*Juan ate*
	(b) Juan comía patatas.	*Juan ate potatoes*
	(c) Juan compró patatas.	*Juan bought potatoes*
	(d) Juan compró patatas para su hijo.	*Juan bought his son potatoes*

In *(1a,b)*, the verb "comer" (*to eat*) shows an intransitive*(a)*/transitive*(b)* alternation; in *(1c,d)*, "comprar" (*to buy*) alternates direct object*(c)*direct+indirect object*(d)* performances. To permit both *(a/b)* and *(c/d)* formations the usual solution is generating (for each verb) a different lexical entry per surface performance.

[6] features as <CONTEXT> or <BACKGR> (used in semantics) or <NONLOC> (used in long-distance dependencies) are unexploited in our grammar.

Thus, *(b)* entry for "comer" would subcategorise for both a subject and for a complement NP, while *(a)* entry would only subcategorise for a subject:

(2) (a) COMER <NP_{subj}>

(b) COMER <NP_{subj} $NP_{dir.obj.}$>

In order to manage these phenomena, [BRISCOE & COPESTAKE 1991] put forward generating different lexical entries for each subcategorisation model of a verb via lexical rules applied to a basic entry.

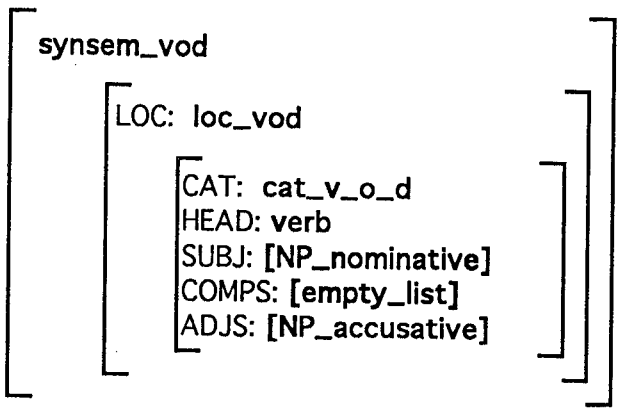
However, we think of "comer"[7] as being just one only concept independently of its syntactic environment: in both cases it expresses a particular kind of process of somebody ingesting something; the difference is that while in (2a) the ingested thing remains unexpressed, in (2b) it is explicitly stated. Therefore we have preferred to represent verbs such as "comer" as a single verb which sometimes takes a direct object and sometimes do not:

(3) COMER <NP_{subj} ($NP_{dir.obj.}$)>

We achieve this effect by considering the minimum complement-taking alternance as the basic word-sense and treating subsequent complements as optionals i compulsory complements are listed in <COMPS> and "optionals" are in <ADJ>.

E.g.: An approach to a <SYNSEM> specification for a verb such as "comer":

(4)



[Of course, in order to get an accurate syntactic parse, a real Adjunct has to be differentiated of these "optional" arguments. Thus, listing together adjuncts and optional arguments in ADJS is not the finest solution, but some problems in implementing optionality in ALE has led us to do that. The problem is Ale doesn't implement optionality within the system, and we had to build some Prolog clauses to manage this feature; this led to a loss of power because of extra backtracking introduced. In the future, either implementing optionality in COMPS list or adding a new subcategorisation feature OPT_COMPS is going to be a best resolution for the issue.]

Our approach results, as we pointed out above, in six verb sorts: **v_sa**, **v_o_o**,

[7] In straight ingestion contexts; we don't care about metaphorical sense extensions.

v_o_d, **v_d_o**, **v_d_i** and **v_di_o**, exemplified respectively by "ser", "morir", "comer", "golpear", "comprar" and "dar". This nomenclature is intended to be mnemonic: in v_X_Y, position X mean obligatory complements, while Y mean optional ones; moreover, **d** means direct object, **i** indirect object, **sa** an adjective phrase, and **o** the empty list. In intermediate types, **x** means any verb complement (not the empty list). Thus, **v_d_i** means " a verb which takes compulsorily a direct object and optionally an indirect object" and **v_o_d** " a verb that may (optionally) take a direct object". Each sort of verb admits obviously optional PP complements. This is not encoded in verb sorts as it is common to all of them: a superior type provides this information.

Examples considered are:

(5) v_sa: "ser". *José es tonto.*

v_o_o: "morir". *José murió.*

v_o_d: "comer". *Juan comía.*

Juan comía patatas.

v_d_o: "golpear". *José golpeó a Juan.*

v_d_i: "comprar". *Juan compró arsénico.*

Juan compró arsénico para José.

v_di_o: "dar". *Juan dió patatas a José.*

[Sanfilippo *et al.*, 94], also working within ALE and an HPSG-like grammar, have recently made a similarly aimed approach. They associate verb entries to underspecified lexical types; each type has subtype extensions, each one bearing a different <COMPS> pattern, which describe possible uses of the verb. Further procedural attachments (Prolog-style clauses associated to phrase formation rules) will solve type ambiguity attending to the context.

The final effect of such arrangements is different types of <CAT> features as long as it contains features <COMPS> and <ADJ>, whose contents vary with every sort of verb; consequently, <LOC> (the feature containing <CAT>) and <SYNSEM> -(that one containing <LOC>) have to be in turn organised in appropriate types.

Moreover, the need of arranging these types in lists as values of the appropriate subcategorisation features forces to typify lists depending on their syntactic contents (e.g., lists of noun-headed, prep-headed etc., elements, lists of subst-elements -*subst* lumps together noun, verb, prep and adjective-); and so on.

All this makes the background of the system sound and capable to work with a little number of phrasal rules. But the other way round it conveys a big drawback: every widening or correction made on the grammar will have to deal with a delicate reformulation of the type system, flexibility and ease of maintenance of the grammar is severely damaged.

4.2 Rules

The three phrasal rules which operate in our grammar are: subj_head, comps_head and adjs_head. As their names point out they respectively combine heads with their *subjects*, *complements* and *adjuncts*.

These terms have to be managed carefully not to be misunderstood in the present context. Now that the following remarks have to be noticed:

- as we said above, *optional* complements of verbs are listed in <ADJ>, although not being

adjuncts in strict grammatical sense. Thus the value of the <ADJ> feature of verb-headed phrases will be a list of real adjuncts (e.g., PPs) plus these *optional* complements.

- *subject* is not to be strictly understood in its usual grammatical-function meaning; not only sentence subject NP phrases are *subjects* of VPs but also determiners are treated as *subjects* of nouns in a NP. Later developments of HPSG includes information relative to articles etc. in a feature for nouns called <SPEC>. Nevertheless, we have preferred not including this feature here and maintain former HPSG configurations (which dealt with determiners as <SUBJ> of common nouns) as long as we didn't found a reason in Spanish to introduce that new feature <SPEC>.

HPSG's principles used here are Head-Feature Principle and Subcat Principle. [Semantics Principle (which passes <CONT> values from the head-daughter to the superior node) is formally implemented in our system, although it is quite contentless: the only features actually acting are <PARA:INDEX>, which state number, person and gender concordances.] Principles are declarative preconditions which extend their action on every aspect of the grammar, specially on the rules. Unfortunately there is no way of implementing such a high-level device in ALE. The same thing happens with HPSG's language-dependent Constituent Ordination Principles, supposed to fix phrasal surface order. Therefore, the solution we have chosen to implement such principles has been to make them present and acting in every rule of the grammar.

The rules are the following:

subj_head rule combines VPs with their subject NPs and Noun Groups with their Determiners. In both cases <COMPS> and <ADJ>s have to be already retrieved. The result is a saturated phrase -subcategorisation lists are all empty lists. Notice that the passing of the features <head:HEAD> and <cont:C> from the head daughter to the higher level phrase ensures the observance of both Head-Feature Principle and Semantics Principle (in fact, concordance). This effect will be also achieved in **comps_head** and **adjs_head** rules.

```
subj_head   rule
(phrase,
synsem:loc:( cat:(  head:HEAD,
                    subj:[],
                    comps:[],
                    mark:[],
                    fill:[],
                    adj:[]),
                    cont:C),
dtrs:( head_dtr:HEADDTR,
       subj_dtr:SUBJDTR))
==>
cat>
(SUBJDTR,synsem:SUBJ),

cat>
(HEADDTR,
synsem:loc:(cat:(   head:HEAD,
                    subj:[SUBJ],
                    comps:[]),
              cont:C)).
```

fig. 3: subj_head rule.

comps_head rule combines verbs with (obligatory) complements and prepositions –understood as heads of PPs- with NPs. Subcat Principle acts by erasing one element of the <COMPS> list of the head when passing that list to the superior node.

```
comps_head rule
(phrase,
synsem:loc:(cat:(   head:HEAD,
                    subj:SUBJ,
                    comps:T,
                    mark:MARK,
                    fill:FILL,
                    adj:ADJ),
              cont:C),
dtrs:( head_dtr:HEADDTR,
       comp_dtr:COMPDTR))
⟹
cat>
(HEADDTR,
synsem:loc:(cat:(   head:HEAD,
                    subj:SUBJ,
                    comps:[COMP|T],
                    mark:MARK,
                    fill:FILL,
                    adj:ADJ),
              cont:C)),
cat>
(COMPDTR,synsem:COMP).
```

fig. 4: comps_head rule.

adjs_head rule combines verbs with *optional* complements and verbs or other categories with adjuncts (e.g., nouns with adjectives).

```
adjs_head   rule
(phrase,
synsem:loc:(cat:(   head:HEAD,
                    subj:SUBJ,
                    comps:COMPS,
                    mark:MARK,
                    fill:FILL,
                    adj:T),
              cont:C),
dtrs:( head_dtr:HEADDTR,
       adj_dtr:ADJDTR))
⟹
cat>
(HEADDTR,
synsem:loc:(cat:(   head:HEAD,
                    subj:SUBJ,
                    comps:COMPS,
```

```
                    mark:MARK,
                    fill:FILL,
                    adj:[ADJ|T]),
            cont:C)),
cat>
(ADJDTR,synsem:ADJ).
```

fig.5: adjs_head rule.

As we have said, the principles have been encoded into the rules. As a matter of example, if we take the last figure, there is the encoding into ALE of the rule ADJS_HEAD. But if we look more precisely, there is a sharing of the HEAD value between the superior category and the first daughter. This is the inclusion of the HEAD principle into the ADJS_HEAD rule. This is, of course, nor the best way to encode principles, as every new principle has to be inserted into each rule, and every new rule has to be encoded with all principles into it.

These rules (applying on the type system and the lexicon) permit the formation of sentences generable by the following context-free grammar (categories prefixed by * stand for terminal vocabulary; x/y means alternative possibilities; terms in brackets are optional; categories X_1 bear accusative case, while X_2, bear dative):

S --> NP VP

NP --> *proper_noun

NP --> *det *noun (*adj) (PP)

PP --> *prep NP

VP --> *v_sa *adj

VP --> *v_o_o (PP)

VP --> *v_o_d (NP_1/PP_1) (PP)

VP --> *v_d_o NP_1/PP_1 (PP)

VP --> *v_d_i NP_1 (PP_2) (PP)

VP --> *v_di_o NP_1/PP_1 PP_2 (PP)

Agreement between nodes is provided by the unification mechanism via specifications stated in lexical entries.

4.3 The Lexicon

The lexicon in our system is by now merely representative of classes of vocabulary. Thus it is deliberately small but powerful enough since a remarkable set of words is representable by their means just by attributing word lemmas to ALE macros (see appendix).

The lexical rules device has not been exploited yet; we have only implemented a plural formation rule just in order to test this capability of ALE.

4.4 Difficulties of implementing HPSG in ALE

HPSG is a formalism based on principles. ALE is a rewriting formalism. This

discrepancy between both approaches leads to problems when implementing HPSG on ALE.

First, there appears a basic problem of any rewriting formalisms -not only of ALE. HPSG schemata (what would be called rules in ALE) are described from a structure-validating point of view. But when working with a rewriting formalism, we work from a structure-constructing point of view. Thus, sometimes there is no one-to-one relation between HPSG rules and those of the ALE grammar.

Second, some independent principles operate over HPSG schemata, and each principle apply on every schema. As ALE cannot encode principles separately, we had to embed all principles in every rule. For example, we had to encode word order rule by rule, rather than declaring a higher order principle (as HPSG theory points out). This is a potential source of problems and inconsistencies when building, enlarging or modifying the grammar.

5 Conclusions

The results of this work, i.e. the grammar for assertive sentences of Spanish headed by six different sorts of verbs each accepting different subcategorisation patterns and the type structure which supports this grammar and further enlargements in HPSG, are to be seen in the Appendix-1 below. As well, a sample parse is displayed in Appendix-2.

Next, our final conclusions after having tested ALE and HPSG for the current purposes are offered.

5.1 About ALE

- One of the aims of the work was to test the performance of ALE when working with a complex formalism, as HPSG is. Results are impressive, mainly due to the great time saving because of type system compilation.

- ALE's strong typing also carries some drawbacks; e.g. sometimes one really doesn't care about the value of some attribute (e.g.. PHON), but when working in ALE one is forced to declare everything as a type -namely, one has to include the phonetic value of each word into the type system as one cannot simply declare the type "string" and treat phonetic values as strings.

-The type system has to be maintained at different points of the code. We have to have a well defined type structure, and when we use it in a rule, lexical entry or other components, we have to fit this type structure. Consequently, if we modify the type structure for any component, we have to edit the other components to make them coherent with the modification; there is no way of automating it. When building an application, this type of modifications are very usual. There should be a way for the system to handle this better, for example deducing paths, and using partial paths into rules or components.

-Co-indexations frequently cannot be defined into the type system, only at lexical entries or rules.

-As a final comment, it is hard to correctly implement a principles and parameters formalism on a rewriting formalism. Even though we haven't tried this solution, some principles could be encoded as Prolog clauses and inserted at proper places intro the rules to ensure legal structures to be built, rather than encode principles as a part of the rules. This could be a good sollution because we could take control over where the test of the principles should be carried, and reject an analysis as long as we detect a bad analysis. The problem with this implementation is it causes extra backtracking into the system, as something that ALE would consider correct is rejected due to the principles restrictions. It should be tested wether the fast detection of inconsistencies would be worth the added backtracking.

5.2.- About HPSG.

It is a very powerful formalism for dealing with linguistic restrictions. Inasmuch as the highly structured nature of its signs permits encoding in them all their syntactic and semantic concordance requests, only extremely accurate parses will leak out unification *filter*. In our implementation, even lacking semantic content, we get one only analysis of every (non ambiguous) sentence.

The same way, sign complexity along with strong typing provides that all information obtainable from parsing is expressed and agglutinated in one only structure; one has not no maintain separate representation levels and transduce information from one to another.

But, as every sin carries its own punishment, the counterpart of this advantages lies in the same source that makes them possible: one has to work with a complex lexicon and put a lot of information on it. Although this is commonly seen as a good long-term strategy for dealing with language processing, it is pretty unsuitable for quick grammar developments and certainly utopic for wide language coverage. When one decides to work inside HPSG framework, one has to face seriously at once the problem known as the *bottleneck* of lexical acquisition.

While developing this work we have been progressively led to see HPSG more as a formalism to do theoretical linguistic inquiry than to actually build applications up. Although these are only personal opinions, we think some theoretical HPSG preemptions need to be relaxed to make the formalism handy, i.e. either maintaining a very limited set of phrasal rules while overloading the type structure (it's easier to deal with grammar modification/enlarging by working on a set of rules than on the web of types) or deeply grouping sign's features in a way that one has to go through long paths to retrieve values and to set types for every path step depending on the final value when building the type structure. At this respect we think that *flattening* signs (make them less indented) although may lead to some mismatches with the theory would make them more readable and controllable.

6 Future research

Once ALE's potentialities have been tested, we would like to explore those of LKB[8] framework to parse HPSG, as long as we would be able to handle already existent Lexical Knowledge structures to semi-automatically generate a proper HPSG lexicon, and also to take profit of its greater expressive power. About the analyser, we plan building a Principle-Based parser.

[8] A Lexical Knowledge representation system built within ACQUILEX-I (ESPRIT BRA 3030) and ACQUILEX-II (BRA 7315) projects. It is fully described in [COPESTAKE 92].

7 Appendices

7.1 The code

```
%%%%%%%%%%%%%%%%%%%%
% type hierarchy %
%%%%%%%%%%%%%%%%%%%%

bot     sub [sign, synsem, loc, cat, head, list, nonloc, pform, bool, cont, context, para, index,
psoa, con_struc, pers, num, gen, case, vform, ontologia, morfo].
sign    sub [word,phrase]
        intro [phon:list,synsem:synsem].
word    sub []
        intro [].
phrase sub []
        intro [dtrs:con_struc].
con_struc     sub [head_struc,conj_struc].
conj_struc    sub [].
head_struc    sub    [head_subj_struc,head_comp_struc,head_mark_struc,head_fille
r_struc,head_adj_struc]
        intro [head_dtr:sign].
head_subj_struc       sub []
        intro [subj_dtr:sign].
head_filler_struc     sub []
        intro [filler_dtr:sign].
head_comp_struc       sub []
        intro [comp_dtr:sign].
head_mark_struc       sub []
        intro [marker_dtr:sign].
head_adj_struc        sub []
        intro [adj_dtr:sign].
nonloc sub []
        intro [inher:wh_str,to_bind:wh_str].
cont    sub [nom_obj,verb_obj].
contextsub []
        intro [backgr:list].
verb_obj      sub [].
nom_obj       sub [expl_obj,ref_obj]
        intro [para:para].
expl_obj      sub []
        intro [para:expl].
```

```
ref_obj sub []
        intro  [para:ref,restr:list].
psoa    sub [].
para    sub  [expl,ref]
        intro  [index:index].
        expl     sub [].
        ref      sub  [npro,pron].
                 npro    sub [].
pron    sub  [ppro,ana].
        ppro     sub [].
        ana      sub  [refl,recp].
                 refl    sub [].
                 recp    sub [].
index   sub []
        intro  [pers:pers,num:num,gen:gen].
pers    sub  [prim,seg,ter].
num     sub  [sing,plur].
gen     sub  [masc,fem].
head    sub  [subst,funct].
subst   sub  [noun,verb,adj,prep]
        intro  [prd:bool].
adj     sub []
        intro  [mod:synsem].
det     sub [].
funct   sub  [det,mark].
mark    sub [].
noun    sub  [propi,comu]
        intro  [case:case].
        propi  sub [].
        comu   sub [].
case    sub  [nom,ac,dat,geni,voc,abl].
        nom      sub [].
        ac       sub [].
        dat      sub [].
        geni     sub [].
        voc      sub [].
        abl      sub [].
prep    sub []
        intro  [pform:pform].
pform   sub [].
verb    sub  [verbsa]
        intro  [vform:vform].
verbsa sub []
        intro  [prd:true].
vform   sub  [fin,nonfin].
        fin      sub [].
        nonfin sub [].
sn      sub [].
prim    sub [].
seg     sub [].
ter     sub [].
sing    sub [].
plur    sub [].
masc    sub [].
fem     sub [].
```

```
%%%%%%%%%%%%%%%%%%%%
% typ. cat   %
%%%%%%%%%%%%%%%%%%%%

cat      sub  [cat_subst,cat_funct]
        intro     [head:head,subj:list,comps:list,mark:list,fill:list,adj:list].
cat_subst       sub  [cat_noun,cat_verb,cat_adj,cat_prep].
cat_funct       sub  [cat_det,cat_mark].
cat_noun        sub  [cat_propi,cat_comu]
        intro  [head:noun].
cat_propi       sub []
        intro     [head:propi,subj:e_list,comps:e_list,mark:e_list,fill:e_lis   t,adj:e_list].
cat_comu        sub []
        intro     [head:comu,subj:list_1_det,comps:e_list,adj:list_1_adj,mark   :e_list,fill:e_list].
cat_verb        sub  [cat_v_x_x,cat_v_sa]
        intro  [head:verb,subj:list_1_noun].
cat_v_x_x       sub  [cat_v_o_x,cat_v_d_x,cat_v_di_o].
cat_v_o_x       sub  [cat_v_o_o,cat_v_o_d]
        intro  [comps:e_list].
cat_v_o_o       sub [] % morir
        intro  [adj:e_list].
cat_v_o_d       sub [] % comer
        intro  [adj:list_1_subst].
cat_v_d_x       sub  [cat_v_d_o,cat_v_d_i]
        intro  [comps:list_1_subst].
cat_v_d_o       sub [] % golpear
        intro  [adj:e_list].
cat_v_d_i       sub [] % comprar
        intro  [adj:list_1_prep].
cat_v_di_o      sub [] % dar
        intro  [comps:list_2_subst_prep].
cat_v_sa        sub []
        intro  [comps:list_1_adj,head:verbsa].
cat_adjsub []
        intro  [head:adj,subj:e_list,comps:e_list,adj:e_list,ma   rk:e_list,fill:e_list].
cat_prep        sub []
        intro     [head:prep,subj:e_list,adj:e_list,mark:e_list,fill:e_list].
cat_detsub []
        intro     [head:det,subj:e_list,comps:e_list,adj:e_list,mark:e_list,f   ill:e_list].
cat_mark        sub []
        intro  [head:mark].

%%%%%%%%%%%%%%%%%%%%
% typ. synsem   %
%%%%%%%%%%%%%%%%%%%%

synsem          sub  [synsemsubst,synsemfunct]
        intro  [loc:loc,nonloc:nonloc].
synsemsubst  sub     [synsemnoun,synsemverb,synsemadj,synsemprep]
        intro  [loc:locsubst].
synsemfunct  sub  [synsemdet,synsemmark].
synsemnoun   sub  [synsempropi,synsemcomu]
        intro  [loc:locnoun].
synsempropi  sub []
        intro  [loc:locpropi].
synsemcomu   sub []
```

```
        intro  [loc:loccomu].
synsemverb     sub  [synsemvxx,synsemvsa].
synsemvxx      sub  [synsemvox,synsemvdx,synsemvdio].
synsemvsa      sub  []
        intro  [loc:locvsa].
synsemvox      sub  [synsemvoo,synsemvod].
synsemvoo      sub  []
        intro  [loc:locvoo].
synsemvod      sub  []
        intro  [loc:locvod].
synsemvdx      sub  [synsemvdo,synsemvdi].
synsemvdo      sub  []
        intro  [loc:locvdo].
synsemvdi      sub  []
        intro  [loc:locvdi].
synsemvdio     sub  []
        intro  [loc:locvdio].
synsemadj      sub  []
        intro  [loc:locadj].
synsemprep     sub  []
        intro  [loc:locprep].
synsemdet      sub  []
        intro  [loc:locdet].
synsemmark     sub  []
        intro  [loc:locmark].

%%%%%%%%%%%%%%%%%%%%%%%%
% typ. loc      %
%%%%%%%%%%%%%%%%%%%%%%%%

loc     sub  [locsubst,locfunct]
        intro  [cat:cat,cont:cont,context:context].
locsubst        sub  [locnoun,locverb,locadj,locprep]
        intro  [cat:cat_subst].
locfunct        sub  [locdet,locmark].
locnoun         sub  [locpropi,loccomu]
        intro  [cat:cat_noun].
locpropi        sub  []
        intro  [cat:cat_propi].
loccomu         sub  []
        intro  [cat:cat_comu].
locverbsub  [locvxx,locvsa].
locvxx sub  [locvox,locvdx,locvdio].
locvox sub  [locvoo,locvod].
locvoo sub  []
        intro  [cat:cat_v_o_o].
locvod sub  []
        intro  [cat:cat_v_o_d].
locvdx sub  [locvdo,locvdi].
locvdo sub  []
        intro  [cat:cat_v_d_o].
locvdi sub  []
        intro  [cat:cat_v_d_i].
locvdiosub  []
        intro  [cat:cat_v_di_o].
locvsa sub  []
```

```
        intro  [cat:cat_v_sa].
locadj  sub []
        intro  [cat:cat_adj].
locprep         sub []
        intro  [cat:cat_prep].
locdet  sub []
        intro  [cat:cat_det].
locmark         sub []
        intro  [cat:cat_mark].

%%%%%%%%%%%%%%%%%%%%
% lists            %
%%%%%%%%%%%%%%%%%%%%

list    sub  [e_list,nelist].
e_list  sub [].
nelist  sub[list_1,list_2]
        intro  [hd:bot,tl:list].
list_1  sub     [list_1_noun,list_1_prep,list_1_det,list_1_adj,list_1_subst  ]
        intro  [tl:e_list].
list_1_noun     sub []
        intro  [hd:synsemnoun].
list_1_prep     sub []
        intro  [hd:synsemprep].
list_1_det      sub []
        intro  [hd:synsemdet].
list_1_adj      sub []
        intro  [hd:synsemadj].
list_1_subst    sub []
        intro  [hd:synsemsubst].
list_2  sub   [list_2_subst_prep]
        intro  [tl:list_1].
list_2_subst_prep     sub []
        intro   [hd:synsemsubst,tl:list_1_prep].
bool    sub  [true,false].
true    sub [].
false   sub [].

%%%%%%%%%%%%%%%%%%%%
% ontology         %
%%%%%%%%%%%%%%%%%%%%

ontologia       sub[verbo,jose,def,hombre,tonto,patata,prepos].
jose    sub [].
hombresub [].
def     sub [].
tonto   sub [].
patata  sub [].
verbo   sub  [v_x_x,v_sa].
v_sa    sub  [ser].
v_x_x   sub  [v_o_x,v_d_x].
v_o_x   sub  [v_o_o,v_o_d].
v_o_o   sub  [morir].
v_o_d   sub  [comer].
v_d_x   sub  [v_d_o,v_d_i,v_di_o].
v_d_o   sub  [golpear].
```

```
v_d_i sub [comprar].
v_di_o sub [dar].
morir  sub [].
comer  sub [].
golpearsub [].
comprar        sub [].
dar    sub [].
ser    sub [].
prepos sub [a].
a      sub [].

%%%%%%%%%%%%%%%%%%%%
% lexicon           %
%%%%%%%%%%%%%%%%%%%%

el      ---> @  det(def,masc,sing,ter).
la      ---> @  det(def,fem,sing,ter).
las     ---> @  det(def,fem,plur,ter).
los     ---> @  det(def,masc,plur,ter).
hombre---> @  comu(hombre,masc,sing).
golpea ---> @  verb_d_o(golpear,sing,ter).
come   ---> @  verb_o_d(comer,sing,ter).
muere  ---> @  verb_o_o(morir,sing,ter).
compra---> @  verb_d_i(comprar,sing,ter).
da      ---> @  verb_di_o(dar,sing,ter).
es      ---> @  verb_sa(ser,sing,ter).
jose    ---> @ propi(jose,masc).
tonto   ---> @ adj(tonto,masc,sing).
tontos  ---> @ adj(tonto,masc,plur).
patata  ---> @ comu(patata,fem,sing).
a       ---> @ preposicion(a).

%%%%%%%%%%%%%%%%%%%%
% macros            %
%%%%%%%%%%%%%%%%%%%%

nom    macro
       (word,
       synsem:synsemnoun).

comu(Ph,Gen,Num)     macro
       (word,
       phon:[Ph],
       synsem:        (synsemcomu,
              loc:(   cat:    (subj:[@  syn_det(Gen,Num,_)],
                              adj:[@  syn_adj(Gen,Num)]),
                      cont:
                      (ref_obj,
                      para:   (npro,
                              index:(gen:Gen,num:Num))))))).
syn_adj(Gen,Num)     macro
       (loc:   (locadj,
              cont:   (ref_obj,
                      para:
                              (npro,
                              index:(gen:Gen,num:Num))))).
```

```
adj(Ph,Gen,Num)        macro
        (word,
        phon:[Ph],
        synsem:(synsemadj,  @  syn_adj(Gen,Num))).

syn_det(Gen,Num,Pers)        macro
        (loc:   (locdet,
                cont:   (ref_obj,
                        para:   (npro,
                                index:(gen:Gen,num:Num,pers:Pers))))).

det(Ph,Gen,Num,Pers) macro
        (word,
        phon:[Ph],
         synsem:(synsemdet,  @  syn_det(Gen,Num,Pers))).

propi(Ph,Gen)  macro
        (word,
        phon:[Ph],
        synsem:         (synsempropi,
                 loc:cont:(ref_obj,para:(npro,index:(gen:Gen,num:sing))))).

verb_d_o(Ph,Num,Pers)        macro
        (word,
        phon:[Ph],
        synsem:         (synsemvdo,
                loc:    (cat:   (head:vform:fin,
                                 subj:[loc:(locnoun,cont:para:(AG,index:(num:Num,pers:Pers))  )],
                                comps:[loc:cont:para:PAC],
                                mark:[],
                                fill:[]),
                        cont:   (verb_obj,
                                agente:AG,
                                paciente:PAC)))).
verb_o_d(Ph,Num,Pers)        macro
        (word,
        phon:[Ph],
        synsem:         (synsemvod,
                loc:    (cat:   (head:vform:fin,
                                subj:[loc:(locnoun,cont:para:index:(num:  Num,pers:Pers))],
                                mark:[],
                                fill:[]),
                        cont:verb_obj))).
verb_o_o(Ph,Num,Pers)        macro
        (word,
        phon:[Ph],
        synsem:         (synsemvoo,
                loc:    (cat:   (head:vform:fin,
                                 subj:[loc:(locnoun,cont:para:index:(num:Num,pers:Pers))],
                                mark:[],
                                fill:[]),
                        cont:verb_obj))).
verb_d_i(Ph,Num,Pers)        macro
        (word,
        phon:[Ph],
```

```
        synsem:         (synsemvdi,
                loc:    (cat:   (head:vform:fin,
                                 subj:[loc:(locnoun,cont:para:index:(num:Num,pers:Pers))],
                                 mark:[],
                                 fill:[]),
                        cont:verb_obj))).
verb_di_o(Ph,Num,Pers)          macro
        (word,
        phon:[Ph],
        synsem:         (synsemvdio,
                loc:    (cat:   (head:vform:fin,
                                 adj:[],
                                 subj:[loc:(locnoun,cont:para:index:(num:Num,pers:Pers))],
                                 mark:[],
                                 fill:[]),
                        cont:verb_obj))).
verb_sa(Ph,Num,Pers) macro
        (word,
        phon:[Ph],
        synsem:         (synsemvsa,
                loc:    (cat:   (head:vform:fin,
                                 subj:[loc:(locnoun,cont:para:index:(num:Num,pers:Pers))],
                                 mark:[],
                                 fill:[]),
                        cont:verb_obj))).

preposicion(Phon)       macro
        (word,
        phon:[Phon],
        synsem:(synsemprep,loc:cat:comps:list_1_noun)).

%%%%%%%%%%%%%%%%%%%%%
%  lexical rules      %
%%%%%%%%%%%%%%%%%%%%%

plural_n        lex_rule
@  comu(Phon,Gen,sing)
**>
@  comu(Phon,Gen,plur)
morphs
 (X,V) becomes (X,V,s) when vocal(V),
(X,C) becomes (X,C,es).

%%%%%%%%%%%%%%%%%%%%%
% rules             %
%%%%%%%%%%%%%%%%%%%%%

subj_head       rule
(phrase,
synsem:loc:(    cat:(   head:HEAD,
                        subj:[],
                        comps:[],
                        mark:[],
                        fill:[],
                        adj:[]),
                        cont:C),
```

```
dtrs:( head_dtr:HEADDTR,
       subj_dtr:SUBJDTR))
==>
cat>
(SUBJDTR,synsem:SUBJ),

cat>
(HEADDTR,
synsem:loc:(cat:(      head:HEAD,
                       subj:[SUBJ],
                       comps:[]),
               cont:C)).

comps_head   rule
(phrase,
synsem:loc:(cat:(      head:HEAD,
                       subj:SUBJ,
                       comps:T,
                       mark:MARK,
                       fill:FILL,
                       adj:ADJ),
               cont:C),
dtrs:( head_dtr:HEADDTR,
       comp_dtr:COMPDTR))
==>
cat>
(HEADDTR,
synsem:loc:(cat:(      head:HEAD,
                       subj:SUBJ,
                       comps:[COMP|T],
                       mark:MARK,
                       fill:FILL,
                       adj:ADJ),
               cont:C)),
cat>
(COMPDTR,synsem:COMP).

adjs_head    rule
(phrase,
synsem:loc:(cat:(      head:HEAD,
                       subj:SUBJ,
                       comps:COMPS,
                       mark:MARK,
                       fill:FILL,
                       adj:T),
               cont:C),
dtrs:( head_dtr:HEADDTR,
       adj_dtr:ADJDTR))
==>
cat>
(HEADDTR,
synsem:loc:(cat:(      head:HEAD,
                       subj:SUBJ,
                       comps:COMPS,
                       mark:MARK,
                       fill:FILL,
```

```
                        adj:[ADJ|T]),
                cont:C)),
cat>
(ADJDTR,synsem:ADJ).

%%%%%%%%%%%%%%%%%%%%
% clauses          %
%%%%%%%%%%%%%%%%%%%%

vocal([a]).
vocal([e]).
vocal([i]).
vocal([o]).
vocal([u]).
```

7.2 An example of analysis

```
| ?- rec([el,hombre,tonto,da,patatas,a,jose]).

STRING:
0 el 1 hombre 2 tonto 3 da 4 patatas 5 a 6 jose 7

CATEGORY:
phrase
DTRS head_subj_struc
   HEAD_DTR phrase
        DTRS head_comp_struc
           COMP_DTR phrase
                DTRS head_comp_struc
                  COMP_DTR word
                       PHON nelist
                         HD jose
                         TL e_list
                       SYNSEM [0] synsempropi
                           LOC locpropi
                             CAT cat_propi
                               ADJ e_list
                               COMPS e_list
                               FILL e_list
                               HEAD propi
                                  CASE case
                                  PRD bool
                               MARK e_list
                               SUBJ e_list
                             CONT ref_obj
                               PARA npro
                                  INDEX index
                                     GEN masc
                                     NUM sing
                                     PERS pers
                               RESTR set
                                  ELEM bot
                           CONTEXT context
                              BACKGR set
                                 ELEM bot
                       NONLOC nonloc
```

```
            HEAD_DTR word
                PHON nelist
                  HD a
                  TL e_list
                SYNSEM synsemprep
                    LOC locprep
                      CAT cat_prep
                        ADJ [1] e_list
                        COMPS list_1_noun
                            HD [0]
                            TL [2] e_list
                        FILL [3] e_list
                        HEAD [4] prep
                            PFORM pform
                            PRD bool
                        MARK [5] e_list
                        SUBJ [6] e_list
                      CONT [7] cont
                        PARA para
                            INDEX index
                                GEN gen
                                NUM num
                                PERS pers
                      CONTEXT context
                          BACKGR set
                              ELEM bot
                    NONLOC nonloc
        PHON list
        SYNSEM [10] synsemprep
            LOC locprep
              CAT cat_prep
                ADJ [1]
                COMPS [2]
                FILL [3]
                HEAD [4]
                MARK [5]
                SUBJ [6]
              CONT [7]
              CONTEXT context
                  BACKGR set
                      ELEM bot
            NONLOC nonloc
HEAD_DTR phrase
    DTRS head_comp_struc
      COMP_DTR word
            PHON nelist
              HD patata
              TL e_list
            SYNSEM [9] synsemcomu
                LOC loccomu
                  CAT cat_comu
                    ADJ list_1_adj
                      HD synsemadj
                        LOC locadj
                          CAT cat_adj
                            ADJ e_list
                            COMPS e_list
```

FILL e_list
HEAD adj
MOD synsem
LOC loc
CAT cat
ADJ list
COMPS list
FILL list
HEAD head
MARK list
SUBJ list
CONT cont
PARA para
INDEX index
GEN gen
NUM num
PERS pers
CONTEXT context
BACKGR set
ELEM bot
NONLOC nonloc
PRD bool
MARK e_list
SUBJ e_list
CONT ref_obj
PARA npro
INDEX index
GEN [8] fem
NUM plur
PERS pers
RESTR set
ELEM bot
CONTEXT context
BACKGR set
ELEM bot
NONLOC nonloc
TL e_list
COMPS e_list
FILL e_list
HEAD comu
CASE case
PRD bool
MARK e_list
SUBJ list_1_det
HD synsemdet
LOC locdet
CAT cat_det
ADJ e_list
COMPS e_list
FILL e_list
HEAD det
MARK e_list
SUBJ e_list
CONT ref_obj
PARA npro
INDEX index
GEN [8]

NUM plur
PERS pers
RESTR set
ELEM bot
CONTEXT context
BACKGR set
ELEM bot
NONLOC nonloc
TL e_list
CONT ref_obj
PARA npro
INDEX index
GEN [8]
NUM plur
PERS pers
RESTR set
ELEM bot
CONTEXT context
BACKGR set
ELEM bot
NONLOC nonloc
HEAD_DTR word
PHON nelist
HD dar
TL e_list
SYNSEM synsemvdio
LOC locvdio
CAT cat_v_di_o
ADJ [11] e_list
COMPS list_2_subst_prep
HD [9]
TL [12] list_1_prep
HD [10]
TL [18] e_list
FILL [13] e_list
HEAD [14] verb
PRD bool
VFORM fin
MARK [15] e_list
SUBJ [16] list_1_noun
HD [28] synsemnoun
LOC locnoun
CAT cat_noun
ADJ e_list
COMPS e_list
FILL e_list
HEAD [20] comu
CASE case
PRD bool
MARK e_list
SUBJ e_list
CONT [21] ref_obj
PARA npro
INDEX index
GEN masc
NUM sing
PERS ter

RESTR set
ELEM bot
CONTEXT context
BACKGR set
ELEM bot
NONLOC nonloc
TL e_list
CONT [17] verb_obj
PARA para
INDEX index
GEN gen
NUM num
PERS pers
CONTEXT context
BACKGR set
ELEM bot
NONLOC nonloc
PHON list
SYNSEM synsem
LOC loc
CAT cat
ADJ [11]
COMPS [12]
FILL [13]
HEAD [14]
MARK [15]
SUBJ [16]
CONT [17]
CONTEXT context
BACKGR set
ELEM bot
NONLOC nonloc
PHON list
SYNSEM synsem
LOC loc
CAT cat
ADJ [11]
COMPS [18]
FILL [13]
HEAD [14]
MARK [15]
SUBJ [16]
CONT [17]
CONTEXT context
BACKGR set
ELEM bot
NONLOC nonloc
SUBJ_DTR phrase
DTRS head_subj_struc
HEAD_DTR phrase
DTRS head_adj_struc
ADJ_DTR word
PHON nelist
HD tonto
TL e_list
SYNSEM [19] synsemadj
LOC locadj

```
                CAT cat_adj
                  ADJ e_list
                  COMPS e_list
                  FILL e_list
                  HEAD adj
                    MOD synsem
                      LOC loc
                        CAT cat
                          ADJ list
                          COMPS list
                          FILL list
                          HEAD head
                          MARK list
                          SUBJ list
                        CONT cont
                          PARA para
                            INDEX index
                              GEN gen
                              NUM num
                              PERS pers
                        CONTEXT context
                            BACKGR set
                              ELEM bot
                      NONLOC nonloc
                    PRD bool
                  MARK e_list
                  SUBJ e_list
                CONT ref_obj
                  PARA npro
                    INDEX index
                      GEN masc
                      NUM sing
                      PERS pers
                  RESTR set
                    ELEM bot
                CONTEXT context
                    BACKGR set
                      ELEM bot
              NONLOC nonloc
HEAD_DTR word
    PHON nelist
      HD hombre
      TL e_list
    SYNSEM synsemcomu
      LOC loccomu
        CAT cat_comu
          ADJ list_1_adj
            HD [19]
            TL [22] e_list
          COMPS [23] e_list
          FILL [24] e_list
          HEAD [20]
          MARK [25] e_list
          SUBJ [26] list_1_det
            HD [27] synsemdet
              LOC locdet
                CAT cat_det
```

ADJ e_list
COMPS e_list
FILL e_list
HEAD det
MARK e_list
SUBJ e_list
CONT ref_obj
PARA npro
INDEX index
GEN masc
NUM sing
PERS ter
RESTR set
ELEM bot
CONTEXT context
BACKGR set
ELEM bot
NONLOC nonloc
TL e_list
CONT [21]
CONTEXT context
BACKGR set
ELEM bot
NONLOC nonloc
PHON list
SYNSEM synsem
LOC loc
CAT cat
ADJ [22]
COMPS [23]
FILL [24]
HEAD [20]
MARK [25]
SUBJ [26]
CONT [21]
CONTEXT context
BACKGR set
ELEM bot
NONLOC nonloc
SUBJ_DTR word
PHON nelist
HD def
TL e_list
SYNSEM [27]
PHON list
SYNSEM [28]
PHON list
SYNSEM synsem
LOC loc
CAT cat
ADJ e_list
COMPS e_list
FILL e_list
HEAD [14]
MARK e_list
SUBJ e_list
CONT [17]

```
        CONTEXT context
            BACKGR set
                ELEM bot
    NONLOC nonloc

ANOTHER? t.

no
|?-
```

References

BADIA T. Inicios de una gramática para el español en ALEP, un formalismo de unificación. Proceedings of the Xth SEPLN annual meeting. Santiago de Compostela. 1993.

BRISCOE EJ. and COPESTAKE AA. Sense extensions as lexical rules. Acquilex WP num. 22. 1991.

CARPENTER B. The Logic of Typed Feature Structures. Cambridge University Press. New York. 1992.

CARPENTER B. ALE: The Attribute Logic Engine User's Guide. Version ß. CMU. Pittsburg. 1992.

COPESTAKE A. The Representation of Lexical Semantic Information. PhD Thesis. Cognitive Science Research Papers 280. University of Sussex. 1992.

POLLARD C. & SAG I. .Information-Based Syntax and Semantics. CSLI, Stanford. 1987.

POLLARD C. & SAG I. Head-Driven Phrase Structure Grammar. (Draft version). CSLI. Stanford. 1993

SANFILIPPO A., BENKERIMI K. & DWEHUS D. 1994. Virtual Polysemy. COLING 94 Proceedings. Kyoto. 1994.

STEFANOVA M., ter STAL W. A comparison of ALE and PATR; practical experiences. Twente Workshop on Language Technology. 1993.

Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya

Research Reports – 1995

LSI–95–1–R "Octree simplification of polyhedral solids", Dolors Ayala and Pere Brunet.

LSI–95–2–R "A note on learning decision lists", Jorge Castro.

LSI–95–3–R "The complexity of searching implicit graphs", José L. Balcázar.

LSI–95–4–R "Design quality metrics for object-oriented software development", Alonso Peralta, Joan Serras, and Olga Slavkova.

LSI–95–5–R "Extension orderings", Albert Rubio.

LSI–95–6–R "Triangles, ruler, and compass", R. Juan-Arinyo.

LSI–95–7–R "The modifiability factor in the LESD project: definition and practical results", Nuria Castell and Olga Slavkova.

LSI–95–8–R "Learnability of Kolmogorov-easy circuit expressions via queries", José L. Balcázar, Harry Buhrman, and Montserrat Hermo.

LSI–95–9–R "A case study on prototyping with specifications and multiple implementations", Xavier Franch.

LSI–95–10–R "Evidence of a noise induced transition in fluid neural networks", Jordi Delgado and Ricard V. Solé.

LSI–95–11–R "Supporting transaction design in conceptual modelling of information systems", Joan A. Pastor-Collado and Antoni Olivé.

LSI–95–12–R "Computer (PC) assisted drawing of diagrams for forecasting soaring weather", Lluís Pérez Vidal.

LSI–95–13–R "Animats adaptation to complex environments as learning guided by evolution", Mario Martin, Màrius Garcia, and Ulises Cortés.

LSI–95–14–R "Learning to solve complex tasks by reinforcement: A new algorithm", Mario Martin and Ulises Cortés.

LSI–95–15–R "Analysis of Hoare's FIND algorithm with median-of-three partition", Peter Kirschenhofer, Conrado Martínez, and Helmut Prodinger.

LSI–95–16–R "MDCO to B-Rep conversion algorithm", Dolors Ayala, Carlos Andújar, and Pere Brunet.

LSI-95-17-R "Augmented regular expressions: A formalism to describe, recognize, and learn a class of context-sensitive languages", René Alquézar and Alberto Sanfeliu.

LSI-95-18-R "On parallel versus sequential approximation", Maria Serna and Fatos Xhafa.

LSI-95-19-R "A set of rules for a constructive geometric constraint solver", Robert Juan-Arinyo and Antoni Soto.

LSI-95-20-R "From degenerate patches to triangular and trimmed patches", Marc Vigo, Núria Pla, and Pere Brunet.

LSI-95-21-R "Learning Ordered Binary Decision Diagrams", Ricard Gavaldà and David Guijarro.

LSI-95-22-R "The complexity of learning minor closed graph classes", Carlos Domingo and John Shawe-Taylor.

LSI-95-23-R "Approximating the permanent is in RNC", J. Díaz, M. Serna, and P. Spirakis.

LSI-95-24-R "Constraint satisfaction as global optimization", Pedro Meseguer and Javier Larrosa.

LSI-95-25-R "A rule-constructive geometric constraint solver", R. Juan-Arinyo and Antoni Soto.

LSI-95-26-R "External schemas in object oriented databases" (written in Spanish), José Samos.

LSI-95-27-R "An approach to belief in strong Kleene logic", Gustavo Núñez and Matías Alvarado.

LSI-95-28-R "Dynamic belief modeling", Antonio Moreno and Ton Sales.

LSI-95-29-R "Proceedings ICLP Workshop on Deductive Databases and Abduction", to appear. H. Decker, U. Geske, A. Kakas, C. Sakama, D. Seipel, and T. Urpí (editors).

LSI-95-30-R "Difference lists and difference bags for logic programming of categorial deduction", F. Xavier Lloré and Glyn Morrill.

LSI-95-31-R "An experiment on automatic semantic tagging of dictionary senses", German Rigau.

LSI-95-32-R "Slices of meaning: a study on nouns of portions", Salvador Climent and Maria Antònia Martí.

LSI-95-33-R "An HPSG grammar of Spanish implemented in ALE: A solution for dealing with subcategorisation alternances", Salvador Climent and Xavier Farreres.

LSI-95-34-R "Optimal and nearly optimal static weighted skip lists", Conrado Martínez and Salvador Roura.

LSI-95-35-R "Evaluation of expressions in a multiparadigm framework", Xavier Burgués and Xavier Franch.

LSI-95-36-R "Equivalence between executable OOZE and algebraic specification", Vicent-Ramon Palasí Lallana.

LSI-95-37-R "Automatic deduction of the behavioral equivalence between two algebraic specifications", Vicent-Ramon Palasí Lallana.

Hardcopies of reports can be ordered from:

Nuria Sánchez
Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
Pau Gargallo, 5
08028 Barcelona, Spain
secrelsi@lsi.upc.es

See also the Department WWW pages, http://www-lsi.upc.es/www/