Translating Spanish into Logic through Logic

Veronica Dahl

Department of Mathematics
University of Buenos Aires
Buenos Aires, ARGENTINA

We discuss the use of logic for natural language (NL) processing, both as an internal query language and as a programming tool. Some extensions of standard predicate calculus are motivated by the first of these roles. A logical system including these extensions is informally described. It incorporates semantic as well as syntactic NL features, and its semantics in a given interpretation (or data base) determines the answer-extraction process. We also present a logic-programmed analyser that translates Spanish into this system. It equates semantic agreement with syntactic well-formedness, and can detect certain presuppositions, resolve certain ambiguities and reflect relations among sets.

1. Introduction

The idea of using logic as a conceptual framework in question-answering systems is not new. The fact that it can formally deal with the notion of logical consequence makes it particularly attractive for representing meaning. Standard predicate calculus, however, does not seem adequate for representing all the semantic features of natural language (NL), e.g. presuppositions and the subtleties of meaning involved in NL quantifiers. Nevertheless, some recent developments indicate that logic can play an important role in NL processing.

In the first place, recent linguistic research [15,18] has arrived at interesting results concerning the extension of standard predicate calculus in order to provide a better formal model of language.

Secondly, programming in logic [19,29] has become possible since the development of the PROLOG programming language at Marseille [3,5,27]. Logic can now be used both as the underlying formalism and as the programming tool. As has been shown in [30], no loss in efficiency need be involved with respect to languages such as LISP, even though higher level features are supported in PROLOG (e.g. non-determinism).

Thirdly, most PROLOG implementations include a version of metamorphosis grammars (MGs), a logic-based formalism useful in particular for describing NL processors in terms of very powerful rewriting rules [5].

Finally, the evolution in data base technology has been tending more and more towards the use of logic, both for data description and for queries [14].

Drawing on these developments, we have implemented (starting in 1976) successive experimental data base query systems, each written entirely in PROLOG.

The first system [6] represented the hardware and software catalogues for the Solar 16 series of computers. The user could ask it to build up a computer configuration satisfying his particular needs. We then developed a simpler but more general system which accepts different data bases to be consulted in Spanish or in French [7]. This system was later adapted to Portuguese consultation by H. Coelho and L. Pereira (personal communication, 1978), and to English consultation by D. Warren and F. Pereira (personal communication, 1980). In both cases, notably few modifications were needed.

The data base querying features that evolved from the development of these successive systems were coupled with NL representation features into a single logical formalism. Its linguistic motivation and general outline has been described by A. Colmerauer in [4], and its motivation from the data base querying viewpoint has been studied in [10]. The formal definition corresponding precisely to our multilingual system can be found in [11].

In this paper we present a thorough description of the main principles underlying these NL processors. There is some overlap with previous work of the author [9], to make this paper self-contained.

0362-613x/81/030149-16$01.00

We first discuss what features are convenient in a computationally useful logical representation of NL sentences. Then we present an informal definition of the logical system possessing these features which serves as our internal query language. Finally, we show a step-by-step development of a PROLOG analyser for Spanish, after an informal description of our programming tools.

A complete listing of our PROLOG Spanish grammar is given in the Appendix in the Microfiche Supplement to this issue of the *Journal.*

The discussion of our analyzer is not intended to be normative: alternative solutions for the problems we encountered are certainly conceivable. Moreover, many of our choices were constrained by the hardware and software tools available to us. We merely show one way of using logic throughout a NL query system, which has proved feasible within modest computational environments. (Our system was first implemented on a 32K, T-1600 minicomputer, using a 1975 version of PROLOG.)

The research reported here has motivated further work on logic-programmed NL processors for data bases (e.g. [2,22,32], and on the need of extending the original MG formalism [24]. On the other hand, the data base component of this system, together with that of the SOLAR 16 system [6], has also influenced other researchers (cf. [2,31]).

A comprehensive description of the data base component of our system can be found in [13]. The SOLAR 16 system has only been reported in [6].

2. Mapping Natural Language into Logic

This section discusses several NL processing problems and suggests ways of solving them, through careful choices for the internal language's features. We arrive at a typed, three-valued, set-oriented logical system, which we shall call L3. Its role is a double one. On the one hand, its syntax serves as a more rigorous and informative alternative to NL, making semantic features of NL sentences explicit. On the other hand, its semantics provides a clear definition of the answer-extraction process: the evaluation of an L3 formula yields either a truth value (corresponding to a yes-no question) or the representation of a set (corresponding to a wh-question). Spanish is used as the concrete point of reference. Related work for French is [4].

2.1 Meaningfulness, Ambiguity and Semantic Types

A NL processing system must have a means for checking semantic as well as syntactic well-formedness, in order to reject semantically anomalous sentences.

A widespread solution to this problem consists in first generating a "deep structure" of the sentence, taking only syntax into account, and then performing all the necessary semantic operations and checks on it.

As has already been observed [26], this often implies a tradeoff between syntactic and semantic complexity. But it should be emphasized that it is overall simplicity and efficiency that are important. It seems likely, at least for limited computational resources and a given coverage of NL, that a several-pass analyser will take more space and time than a one-pass analyser.

Moreover, linguists themselves are not unanimous as to whether the semantic component should be separate or intermingled with the syntactic component [21].

While taking no sides in this discussion, we shall describe a framework in which syntactic and semantic analysis take place during a one-pass, non-deterministic process, and which, as we have said, has proved sufficient even on small machines.

Where logic is concerned, there is a simple and elegant way of dealing with meaningfulness: by using types. Types, by the way, are also a useful means for associating the universe of predicate calculus to the relations in a particular data base.

They are also useful for improving efficiency: a) by narrowing the search space, as only those values in a variable's associated domain (or type) need be considered, and b) by avoiding futile access to the data base, as absurd queries can be rejected by the analyser on the grounds of domain incompatibility.

Another interest in using types is that they provide an efficient means for discarding readings that are syntactically acceptable but semantically incorrect. Take for instance the query:

Cuál es el salario del empleado que vive en Lomas?
What is the salary of the employee who lives in Lomas?

From syntax alone, there is no way to decide whether the antecedent of the relative clause is "the salary of the employee" or "the employee". But in a type-checking system in which the first argument of the relation "live" is associated with the *human* domain, and in which employees—and not salaries—are known to belong to this same domain, the first reading is not even possible.

Ambiguities concerning different meanings of a word can often be resolved through domain checking.

Types can also be used to place modifiers other than relative clauses. Our system, however, does not exploit them in this way, although it does check that the modifiers it encounters are of the expected type.

Finally, let us mention that types in a finite world can contribute to solving an important problem arising in PROLOG programs in which negation is defined by proof failure (i.e., the failure to prove a given fact is taken as proof of its negative counterpart). This is the case for most practical data bases. As a discussion of this problem falls outside the scope of this paper, the interested reader is referred to [12].

2.1.1 Contextual Typing

The use of variable typing to constrain the parse and aid in disambiguation is not new. Many language processors—such as LUNAR [34], CO-OP [17], LADDER [16], PHLIQA1 [1]—resort to some kind of typing to provide these capabilities.

In our particular approach, the selection of a semantic interpretation is dynamically made on a syntactic basis. This takes place automatically, in the standard PROLOG matching of terms. A simplified explanation follows, and more precise details are given in a later section.

Briefly, referential words (nouns, verbs and adjectives) parse into predicates whose arguments are typed. For instance, the verb "to paint" might induce predicates of the form:

paint(person-x,object-y)

The expected types of a predicate's arguments are given in lexical entries. Ambiguous referential words have a lexical entry for each possible combination of meaning and syntactic role they can accept. For instance, the word "blue" can designate an object's colour or a person's mood, giving rise to the following entries:

Adjective(sad(person-x))=blue
Adjective(blue(object-x))=blue

where the function terms act as internal representations of the word. During the parsing process, which is non-deterministic, the correct parse is automatically chosen by matching appropriate terms. For instance, "Which blue door is John painting?" would generate a formula containing predicates of the form "door(t-z)", "paint(person-John,t-z)" and "p(t-z)", where p is either "sad" or "blue". Only those lexical rules allowing t to take a value (namely, t=object) that is compatible with its three occurrences will result in a successful parse. Thus, the "sad" interpretation is ruled out by type requirements.

As we shall see later, types are actually represented by expressions that reflect subcategorizations and allow for domain intersections to be found automatically simply by leaving the PROLOG interpreter to match these expressions.

2.2 Presuppositions, Quantifiers and a Three-Valued Logic

Typed calculus in itself is not enough to make all sentences meaningful. A third logical truth value would be useful, because in NL there are two ways in which a statement may fail to be true: either because its negation holds, or because something presupposed by the statement fails to be satisfied. In the latter case, the statement is felt to be pointless rather than false.

There is another reason why it must not be considered false. Take for instance the statement:

El sombrerero loco odia a Alicia.
The mad hatter hates Alice.

In a context in which no hatter is mad, it is obviously not true. However, we cannot consider it false either, since then the statement

El sombrerero loco no odia a Alicia.
The mad hatter does not hate Alice.

would have to be considered true.

The nonexistence of a referent for the definite noun phrase makes the whole sentence pointless. The existence of more than one referent would also make it pointless. This is because the Spanish singular definite article induces a presupposition of existence and uniqueness on the noun phrase's referent.

Our treatment of quantification has been devised to account for those presuppositions induced by NL quantifiers. We prefer to call them "determiners", as they include all articles, cardinal numbers and words such as "some", "many", etc.

If a sentence contains a determiner, a quantification of the form "those(x,p)" is introduced, where x is a typed variable and p is a logical formula in our system. Its evaluation yields the set of all x's in x's associated domain which satisfy p. According to the determiner's meaning, presuppositions about the cardinality of such a set are represented within the output formula. For instance, "Three blind mice run" is represented as

equal(card(those(x,and(mice(x),
and(blind(x),run(x))))),3)

which says that the cardinality of the set of those blind mice that run is 3.

Definite articles introduce the formula "if(f1,f2)", the value of which is "pointless" whenever f1 fails to be satisfied, and has the same value as f2 if f1 is true.

Figure 1 shows an example, using the easier-to-picture tree representation. The formula represented in Figure 1 will evaluate to "pointless" if the set of mad hatters does not contain exactly one element.

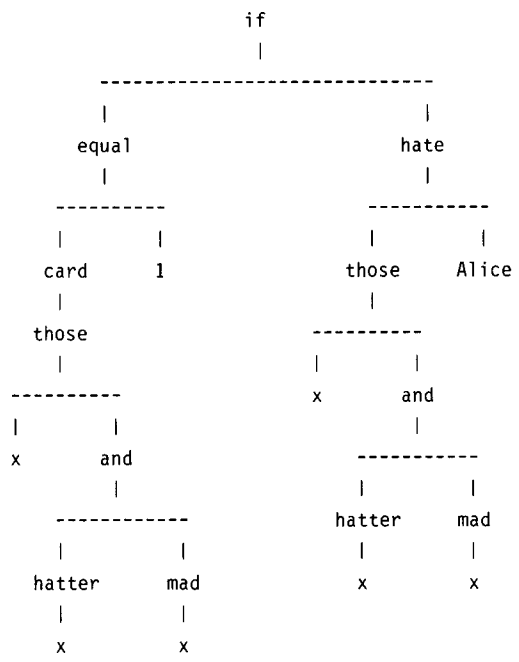


Figure 1. Representation of "The mad hatter hates Alice".

Notice that the "if" formula could be used for representing other types of presuppositions as well (e.g., those of factive predicates), although our system only uses it for the presuppositions of definite articles.

An alternative approach to false presupposition detection is the pragmatic one, in which false presuppositions are caught by noting their empty extensions in the data base, and a two-valued logic is preserved. The question of whether a pragmatic or a semantic analysis of presuppositions is best is far from settled, and we shall not attempt to solve it here. Let us merely note the following:

a) The pragmatic and semantic approaches are not incompatible: one can *both* report pragmatically detected failed assumptions, and assign non-traditional truth values to the sentences containing them.

b) Subtler truth-value assignments facilitate low-cost overall responses of the system. For instance, a given presupposition's failure can both be pointed out during its local evaluation, and carried on (via truth-values) to upper levels, for the system itself to see and to possibly take further action of its own (such as trying alternative ways of complying with the request).

Although not exploited in this sense in our system, this feature would allow a more flexible treatment of presuppositions, as it would enable the system to decide, for instance, in which cases they can be safely and usefully ignored.

Examples of pragmatically based systems are PHLIQA1 [1] and CO-OP [17]. The semantic approach is taken for instance in [23], where it serves in particular to check presuppositions induced by pronouns.

2.2.1 Spanish Determiners and Their Translations

We can now examine the general process by which a determiner introduces a "those" formula. Let us consider a sentence consisting of a noun phrase followed by a verb phrase, in which the noun phrase contains a noun introduced by a determiner. We can first represent the sentence through a three-branched quantification of the form:

$$q(x,f1,f2)$$

where q is a quantifier into which the determiner translates, f1 is the noun phrase's translation, and f2 is the verb phrase's translation. Intuitively, f1 specifies the domain of quantification, and q states what portion of the domain f2 holds for. Our previous example, for instance, can first be represented as in Figure 2.

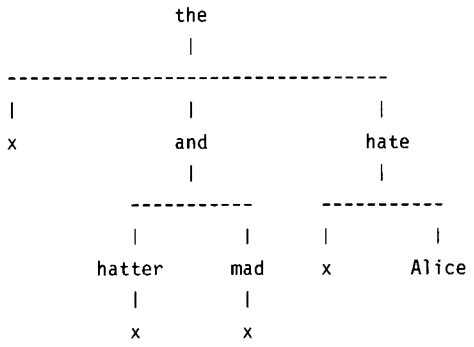


Figure 2. A first representation of "The mad hatter hates Alice".

In our implementation, a variable's associated domain depends both on f1 and f2 in the following manner: each predicate known to the data base system has a domain associated with each of its arguments. For instance, "mad" and "hatter" could require their arguments to belong to the *human* domain, whereas "hate" could require its first argument to belong to the *animal* domain.

When a three-branched quantification is generated, the variable it creates is typed by the intersection of all those domains it has been associated with by the predicates appearing in either f1 or f2. In our example, x's type would be *human* (the intersection of the *human* and the *animal* domains).

Instead of generating a different quantifier for each determiner, it is useful to represent all quantifications through a single one of the form:

for(x,p,c)

the intuitive meaning of which is: "c holds for the set E of all x's in x's domain which satisfy p". In the formula c, the set E will be represented simply by the variable x, so that x plays a double role.

Each quantification is thus assigned an equivalent "for" expression, in which the determiner's meaning is represented. Here are the representations of some of our Spanish quantifiers. The rest are considered in Section 2.2.1.2.

un(x,f1,f2) = for(x,and(f1,f2),greater-than(card(x),0))
a

todo(x,f1,f2) = for(x,and(f1,not(f2)),equal(card(x),0))
every

el(x,f1,f2) = for(x,f1,if(equal(card(x),1),f2))
singular the

los(x,f1,f2) = for(x,f1,if(greater-than(card(x),1),f2))
plural the

ningún(x,f1,f2) = for(x,and(f1,f2),equal(card(x),0))
no

i(x,f1,f2) = for(x,and(f1,f2),equal(card(x),i))
any cardinal number

unos(x,f1,f2) = algunos(x,f1,f2) =
some for(x,and(f1,f2),greater-than(card(x),1))

Notice that we have chosen to translate "un" (*a*) as "at least one" (a frequent sense for "un"). In order to avoid ambiguity, "1" should be used to mean "exactly one". This convention is particularly useful when negation is involved. For instance, "No tengo un centavo" (*I have not a cent*) would be wrongly represented in the "exactly one" interpretation: it would state that the number of cents I possess is not one,which means it can either be 0,2,3, etc.

Finally, any formula of the form "for(x,p,c)" can be replaced by just the formula c, in which all occurrences of x have been replaced by the formula:

those(x,p)

representing the subset of x's domain whose elements satisfy p. This replacement takes place in the data base component of our system.

The reader can now verify that the representations shown in Figures 1 and 2 are equivalent.

2.2.1.1 Quantifier Hierarchy

In our NL subset, quantifier hierarchy obeys the following three rules, which perhaps are too simplistic, but have proved useful. A more thorough description can be found in [4].

- Rule 1: A determiner in a verb's subject introduces a quantification which dominates all quantifications introduced by the verb's complement(s). For instance, "Toda rosa tiene (algunas) espinas" (*Every rose has (some) thorns*) is represented:

toda(x,rosa(x),algunas(y,espinas(y),tiene(x,y)))
every rose some thorns has

Notice that the representation

some(y,thorn(y),every(x,rose(x),has(x,y)))

would be incorrect, as it means instead: "there exists a particular set of thorns which every rose has".

- Rule 2: Whenever a noun has a complement, the quantification introduced by the complement's determiner dominates the one introduced by the noun's determiner. For instance, "Sábato autografía el libro de cada visitante" (*Sábato autographs the book of each visitor*) is represented:

cada(x,visitante(x),el(y,libro-de(x,y),
autografia(Sabato,y)))
each visitor the book-of autographs

- Rule 3: When a referential word (a verb, a noun or an adjective) has more than one complement, quantification takes place from right to left: the rightmost complement generates a quantification which dominates the quantification(s) introduced by the leftmost complement(s). For instance, "Raúl regala un espejo a cada niño" (*Raúl gives a mirror to each child*) is represented:

cada(x,nino(x),un(y,espejo(y),regala(Raul,y,x)))
each child a mirror gives

2.2.1.2 Determiners with a Negative Implication

As a general rule, the negation introduced by "no" in a sentence is translated by placing the operator "no" (*not*) right after the quantification introduced by the subject. For instance, "La indemnización no compensa el despido de Martín" (*The indemnity does not compensate for Martin's dismissal*) is represented:

la(x,indemnizacion(x),no(el(y,despido-de(Martin,y),
compensa(x,y))))
the indemnity not the dismissal-of compensate

But negation is not always explicit. The Spanish determiner "ningún" (*no*) can be regarded as an implicit negation, since it expresses that no portion of the domain of quantification satisfies the statement involved.

In a non-inverted subject position (e.g., "Ningún elefante vuela" -*No elephant flies*), it generates a special quantifier called "ningún", the representation of which takes this fact into account, as we have seen.

There are two other cases, however, in which the determiner "ningún" coexists with an explicit negation. These cases require a different quantifier, as otherwise

the negation would be represented twice. These cases are:

- In a subject position, with subject-verb inversion: the "ningún" determiner is assimilated to the "every" quantifier. For instance, "No vino ningún alumno" (*No student came*) is represented:

todo(x,alumno(x),no(vino(x)))
every student not came

- In a position other than the subject: the "ningún" determiner is assimilated to the indefinite article's quantifier. For instance, "Carlos no tiene ningún hijo" (*Carlos has not any child*) is represented:

no(un(x,hijo(x),tiene(Carlos,x)))
not a child has

Another special case is the negation preceding the "todo" (every) determiner, e.g. "No todo pájaro canta" (*Not every bird sings*). The analyser considers "no todo" as a single determiner generating its own associated quantifier:

```
no-todo(x,f1,f2)  =
    for(x,and(f1,not(f2)),greater-than(card(x),0))
```

2.3 Distributive, Collective, and Respective Plural

Semantically, different kinds of plurals can be distinguished. For instance, the sentence "Ana y Juan hablan español y francés" (*Ann and John speak Spanish and French*), which translates roughly into

speak({Ann,John}, {Spanish,French}),

introduces a *distributive* plural and must therefore evaluate to true (false) if the following formulas are all true (false):

speak(Ann,Spanish)
speak(John,Spanish)
speak(Ann,French)
speak(John,French)

On the other hand, the sentence "A y B son paralelas" (*A and B are parallel*), which translates into

parallel({A,B}),

introduces a *collective* plural and must evaluate to either true or false as a result of testing the whole set {A,B} for the property of being parallel.

Finally, the sentence "Ana y Juan ganan respectivamente 1000 y 800 dolares" (*Ann and John respectively earn 1000 and 800 dollars*), which translates into

earn({Ann,John}, {$1000,$800}),

introduces a *respective* plural and must evaluate to true (false) if the following formulas are both true (false):

earn(Ann,$1000)
earn(John,$800)

Notice that both distributive and respective plurals presuppose that the set of formulas to be tested all have the same truth value. Whenever such a presupposition is not satisfied, the plural predication is neither true nor false. In the logic L3, the predication is assigned the "pointless" truth value; but in an improvement of this system, we are proposing the use of a fourth truth value, called "mixed", for this situation. It seems more appropriate to differentiate "pointless" and "mixed", so that the system has easy access to locally detected semantic information, in case it needs to take further action.

Distributive and collective plurals are distinguished in the lexicon by syntactically marking the relation they translate into.

Respective plurals are not handled in our implementation: they were introduced (although with only two logical values) in the Portuguese version of our system, where the analyser recognizes them through the words "respective" and "respectively".

2.4 Sets

Relations must be allowed to apply on sets if we are to deal with collective relations. Sets are moreover natural enough in data base applications, as retrieval often concerns sets of objects satisfying certain properties. They can also be useful for defining types. We represent them either extensionally (through lists) or intensionally (through "those" formulas).

Set operations are implicit while parsing—as dynamic type checking involves intersecting various domains—and also during formula evaluation (i.e., in the data base component of our system). In both cases they are kept invisible to the user.

In particular, the user can refer to either sets or individuals when defining a new relation, and rely on the system to make appropriate inferences from his definitions.

2.5 Linguistic Coverage

Our NL subset is extendible in the sense that the user can define those referential words (nouns, verbs and adjectives) associated with his particular data base. This includes the definition of synonyms, always useful regarding different users' views of a data base.

The analyser uses a syntactic variant of PROLOG, called (normalized) metamorphosis grammars (MGs) [5]. As such, MGs share a most useful feature of logic programs: a problem's description and the program which solves it are one and the same. This is due to the existence of an *operational* as well as a *declarative* interpretation for logic programs [19].

Thus, the grammar shown in the Appendix (in the Microfiche Supplement) both provides a formal definition of our linguistic coverage (in its declarative reading) and is the analyser itself (in its procedural reading, which the PROLOG interpreter gives it).

An informal description of its coverage follows.

- *Fixed vocabulary*

Determiners:

el la los las un una unos unas
the *a* *some*

ningún todo i
no *every* *any cardinal number*

Prepositions: all Spanish prepositions

Conjunction: y *and*

Relative pronouns: que quien donde cuyo cual(es)
which who where whose

Interrogative pronouns:

qué cuánto(s) dónde quién cuál
which how much/many where who which

Negative particle: no

- *Variable vocabulary.* Each particular data base includes a definition of its associated proper names, nouns, adjectives and verbs. Only simple verbal forms in the third person are allowed.

- *Structure.* Sentences are either declarative or interrogative, in the active voice. A declarative sentence consists of a subject, an optional negation particle, a verb and its modifiers (in the restricted sense described below).

A subject consists of a noun phrase. A noun phrase is either a series of proper names or a kernel followed by noun modifiers and relative clauses (both optional).

A noun phrase's kernel consists of an optional determiner, an optional series of adjectives, a noun, and possibly a series of adjectival groups. Mass nouns are not dealt with. An adjectival group is an adjective with modifiers.

A modifier (of a noun, an adjective or a verb) is just a noun phrase as described above, possibly introduced by a preposition.

Both relative and interrogative clauses are modified versions of a declarative sentence, where some elements have changed places and some have been replaced by a relative or an interrogative pronoun.

Figure 3 gives a sample consultation session with the system. The data base being consulted concerns individuals grouped as either salesmen, administrative employees, managers, sites, departments or salaries. The first two sets are grouped to form the employee domain, which together with the manager domain forms the human domain. The data base contains the

1. Hola. *(Hello.)*
   Diga. *(I am listening.)*
2. Con quién está casada Eva?
   (Who is Eva married to?)
   Con Adán. *(To Adán.)*
3. Adán y Eva viven en Lanús?
   (Do Adán and Eva live in Lanús?)
   No.
4. Dónde viven Eva y Adán?
   (Where do Eva and Adán live?)
   En Montechingolo. *(In Montechingolo.)*
5. Los departamentos viven en Lanús?
   (Do departments live in Lanus?)
   No entiendo su frase.
   (I cannot understand your sentence.)
6. Cuántos vendedores no trabajan en Dep-1?
   (How many salesmen do not work in Dep-1?)
   2.
7. Qué vendedores no trabajan en Dep-1?
   (Which salesmen do not work in Dep-1?)
   Adán y Beatriz. *(Adan and Beatriz.)*
8. Con quién está casada la empleada cuyo salario es 3 millones?
   (Who is the employee (+fem.) whose salary is 3 million married to?)
   Con Zoilo. *(To Zoilo.)*
9. Cuáles son las empleadas con cuya jefa trabaja Quintín?
   (Which are the employees (+fem.) whose manager (+fem.) Quintin works with?)
   Beatriz y Eva. *(Beatriz and Eva.)*
10. No todos los empleados viven en Lanús.
    (Not every employee lives in Lanús.)
    Muy cierto. *(Quite true.)*
11. Ningún empleado vive en Lanús.
    (No employee lives in Lanús.)
    No es cierto! *(It is not true!)*
12. El vendedor que vive en Témperley gana 4 millones.
    (The salesman who lives in Témperley earns 4 million.)
    Hay alguna presuposición falsa en su afirmación.
    (There is a false presupposition in your statement.)
13. Cuál es el lugar donde no vive ningún empleado?
    (Which is the place where no employee lives?)
    Témperley.
14. Cuántas personas viven en Témperley?
    (How many people live in Témperley?)
    Ninguna. *(None.)*
15. Con quién está casado Daniel?
    (Who is Daniel married to?)
    Con nadie. *(To nobody.)*

Figure 3. A sample consultation session.

relations live-in, earn, married, married-to, work-at, work-with and manager-of. Unary relations named after each domain are implicit in any data base.

3. The Internal Query Language: L3

Informally, three kinds of L3 formulas can be distinguished: *typed* formulas t, *statement* formulas s, and *integer* formulas n.

- a *typed formula* t denotes a subset of a given domain. It can be either a list of constants, a variable or an expression of the form "those(x,s)".

- a *statement formula* s evaluates to either true, false, or pointless. It can take any of the forms:

r(t1, ..., tn) where r is an n-ary predicate symbol corresponding to a distributive, collective or respective relation.
and(s1,s2)
if(s1,s2)
not(s)
equal(n1,n2)
greater-than(n1,n2)

- an *integer formula* n1 denotes an integer number, and can take one of the forms:

j where j is an integer such that j>0
card(t)

Further details can be found in [11].

4. Tools for Writing the Analyser

A brief and informal introduction to logic programs and metamorphosis grammars is given here, for the sake of completeness. Fuller accounts can be found in [5,19,20,29].

4.1 On Logic Programming

Logic programs are essentially sets of clauses of the form:

B <-- A1,A2,...,An

called Horn clauses, where B and Ai are atomic formulas, and all variables in the atomic formulas are understood to be universally quantified. "<--" is read "if", and the commas stand for conjunction. An empty right-hand side denotes a non-conditional assertion of fact. For example,

1) likes(mother(*x*),*x*) <--
every x is liked by his-her-its mother

2) likes(Rover,*y*) <-- likes(*y*,Rover)
Rover likes every y who likes him

In the rest of the paper (except for the figures), variables appear in italics in order to distinguish them from constants.

With respect to a given set of clauses (i.e., a program), the user can ask for relations to be computed, by stating procedure calls, i.e., clauses of the form:

<-- A1,A2,...,An

This triggers an automatic demonstration process, during which the variables in the call take values for which "A1 and A2 and ... and An" holds. Here "<--" can be interpreted as a question mark. An unsuccessful termination implies that no such values exist.

Thus, with respect to clauses 1 and 2 above, the following call:

3) <-- likes(*z*,Rover)
Who likes Rover?

results in *z* being unified (bound) to "mother(Rover)". The same result would have been obtained from the call:

4) <-- likes(*z*,Rover), likes(Rover,*z*)
Who likes and is liked by Rover?

Alternative results for the same call may be obtained within non-deterministic programs. For instance, if we add the clause:

likes(Sweetie,Rover) <--

then call 3 can alternatively result in *z* being bound to "Sweetie".

Practical logic program interpreters, such as PROLOG, also include some extra-logical features for input/output and control functions.

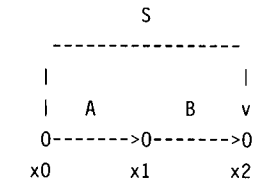
4.2 On Metamorphosis Grammars

MGs are a powerful formalism for describing and processing languages, in which:
- context-sensitive rewriting rules can be described.
- any substring of any length can be rewritten into any other string.
- grammar symbols may include arguments.
- conditions on rule application may be specified.
- sentence generation and parsing are provided by the processor.

MGs can be regarded as a convenient alternative notation for logic programs. Rather than defining them precisely, we shall exhibit some sample grammar rules and show informally one way of translating them into Horn classes, that basically follows A. Colmerauer's PROLOG axiomatization of MGs. A formal and comprehensive description can be found in [5].

4.2.1 Context-Free Rules

Grammar rules can be graphically represented by considering non-terminals as labeled arcs connecting phrase boundaries. A rule such as S --> A B is represented as

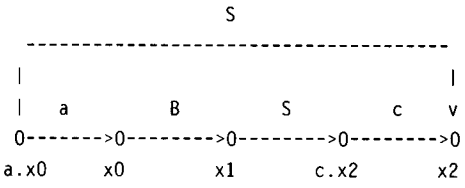


or, in Horn-clause terms:

arrow(S,*x0*,*x2*) <-- arrow(A,*x0*,*x1*), arrow(B,*x1*,*x2*)

which can be read, *for every x0, x1 and x2, there is an arrow labeled S between points x0 and x2 if there is an arrow labeled A between x0 and x1 and another one labeled B between x1 and x2.*

Terminals in a rule are included as part of an edge's name and do not give rise to extra atoms. If a terminal symbol "a" labels an arc leading to point "x", the starting point is named "a.x" (where "." is a binary operator in infix notation). Thus, the rule S --> a B S c can be represented:

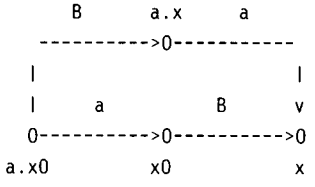


arrow(S,a.*x0*,*x2*) <-- arrow(B,*x*,*x1*), arrow(S,*x1*,c.*x2*)

Strings of consecutive terminals are treated as a single one named after their concatenation.

4.2.2 General Rewriting Rules

When the left-hand side of a rule contains more than one symbol, a single arrow is not enough to depict it: we need another path between the extreme points. For instance, the rule B a --> a B can be graphed as:



where the lower path represents the right-hand side, as usual, and the upper path represents the left-hand side of the rule. In terms of Horn clauses, this gives:

arrow(B,a.*x0*,a.*x*) <-- arrow(B,*x0*,*x*)

Notice that no restrictions need be imposed on the length of the strings on either side of the rule.

4.2.3 A Full Example

Completing the last two rules, we obtain an MG grammar for the language $\{a^n b^n c^n\}$:

1) S --> a B S c
2) S --> a b c
3) B a --> a B
4) B b --> b b

where rules 2) and 4) translate respectively into:

2') arrow(S,a.b.c.*x*,*x*) <--
4') arrow(B,b.b.*x*,b.*x*) <--

In the Horn-clause formulation of this grammar, recognition and parsing of a given string (e.g. "a.a.b.b.c.c.nil") is automatically obtained through the respective PROLOG calls:

5') <-- arrow(S,a.a.b.b.c.c.nil,*x*)
6') <-- arrow(*y*,a.a.b.b.c.c.nil,*x*)

where the value of *x* is of no interest to the user. In order that he need not specify it, a PROLOG predicate called "syn" (for "synthesize") is made available. Its general form is

syn(*x*,*y*)

where *x* stands for the internal representation of the surface sentence *y*. The above commands are therefore actually written:

5) <-- syn(S,a.a.b.b.c.c.nil)
6) <-- syn(*y*,a.a.b.b.c.c.nil)

4.2.4 Parsing and Generating

MGs can also be written for the purpose of *generating* sentences. The same syn command is used for this purpose, except that this time the second argument is the one represented by a variable, e.g.:

<-- syn(S,*x*)

In some cases, it is even possible for the same grammar to work in both ways, although this requires a very careful design. In the rest of this paper, we shall only be concerned with parsing.

4.2.5 Arguments Within Grammar Symbols

We might normally want the parser to retrieve more information than mere recognition. The grammar shown above, for instance, can be also used to retrieve the substring of a's, if it is modified as follows:

1) S(a.*x*) --> a B S(*x*) c
2) S(a) --> a b c

Call 6 would now bind *y* to S(a.a).

4.2.6 Conditions and Calculations

Right-hand sides of rules may contain PROLOG calls (which we shall note between square brackets). They must be successfully evaluated for the rule to apply. For instance, retrieval of the *number* of a's can be obtained in the above grammar by changing the first two rules into:

1) S(n) --> a B S(m) c [plus(m,1,n)]
2) S(1) --> a b c

where "plus" is a PROLOG predicate defining addition of integers. PROLOG calls can also be used to enforce conditions on rule applications.

4.2.7 Normalized MGs

PROLOG only accepts *normalized* G rules, i.e., of the form

A x --> y

where A is a non-terminal symbol, x is a string of terminals and y is any sequence of grammar symbols. This restriction is necessary, within the schema shown, for translating rules into *Horn* clauses (also called *definite* clauses), in which at most one left-hand side atomic formula is allowed. For this reason, they have also been called "definite clause grammars"[1], in a recent article [24] which compares them favourably against the Augmented Transition Network formalism, introduced by Woods [39].

As has been shown in [5], non-normalized rules can be easily replaced by an equivalent set of normalized ones. For instance, B a C b --> f g can be replaced by B a c b --> f g and C --> c, where c is a "dummy" terminal. From the parsing point of view, the results are equivalent.

We can therefore safely ignore this restriction in all that follows, for the sake of clarity.

4.2.8 Derivation Graphs

Although MGs can be understood declaratively, it is sometimes useful to follow a sentence's complete derivation, by constructing a graph which depicts the top-down, left-to-right history of rule applications. We illustrate this through the grammar:

1) Sentence(s) --> Proper-noun(k) Verb(k,s) .
2) Proper-noun(tom) --> tom
3) Proper-noun(john) --> john
4) Verb(k,laugh(k)) --> laughs

[1] In a restricted sense, DCGs only allow a single non-terminal on the left hand side, and are therefore an even more restricted form of MGs. A more accurate synonym for "normalized MG" might be "full DCG"—an expression suggested by D. Warren and F. Pereira in the first draft of [24].

The derivation graph for "John laughs" is shown in Figure 4. The numbers identify the rule applied. The substitutions needed for applying the rule appear as right-hand side labels. Through them we can reconstruct the deep structure "laugh(john)".

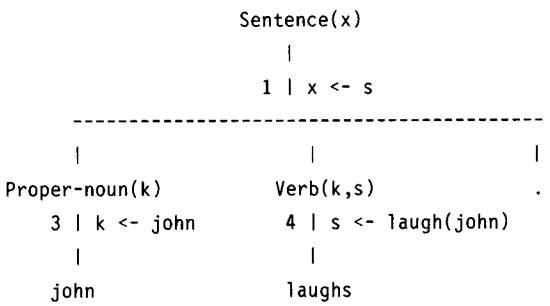


Figure 4. Derivation graph for "John laughs".

Notice that, once a variable takes a value, this value is propagated to each of its occurrences in the graph. Thus, when applying rule 4, we use the known value of k=john. Also, variable renaming must take place whenever necessary in order to ensure that the rule applied and the string it applies to share no variables.

5. Step-by-Step Development of a Spanish Analyser

We now develop a small Spanish parsing grammar, step by step. Although oversimplified, it illustrates practically all of the techniques used to develop the grammar shown in the Appendix (in the Microfiche Supplement). Deep structures and non-terminal symbols are in English, for the convenience of most readers.

5.1 Elementary Statements

The following grammar describes some simple statements constructed around proper nouns, verbs and adjectives.

S) Statement(s) --> Proper-noun(k) Verb(k,l,s) Complements(l,s)

V1) Verb(k,l,s) --> Verb1(be) Adjective(k,l,s)

V2) Verb($k1$,list(arg(in,$k2$),nil),live-in($k1$,$k2$)) --> Verb1(live)

A1) Adjective(k,nil,intelligent(k)) --> Adj1(intelligent)

A2) Adjective($k1$,list(arg(with,$k2$),nil), angry-with($k1$,$k2$)) --> Adj1(angry)

C1) Complements(nil,*s*) -->
C2) Complements(list(arg(*p*,*k*),*l*),*s*) -->
Complements(*l*,*s*) Preposition(*p*) Proper-noun(*k*)

Lexicon:
L1) Adj1(intelligent) --> inteligente
L2) Adj1(angry) --> enojado
L3) Verb1(be) --> es
L4) Verb1(be) --> está
L5) Verb1(live) --> vive
L6) Preposition(in) --> en
L7) Preposition(with) --> con
L8) Proper-noun(joan) --> juana
L9) Proper-noun(tom) --> tomás
L10) Proper-noun(london) --> londres
etc.

Figure 5 shows the derivation graph for "Tomás está enojado con Juana" (Tom is angry with Joan). Most of the substitutions shown concern the deep structure, *x*. The empty string is denoted by a lambda. Some non-terminal symbols are abbreviated. From the substitutions shown, we can see that *x* takes the value "angry-with(tom,joan)".

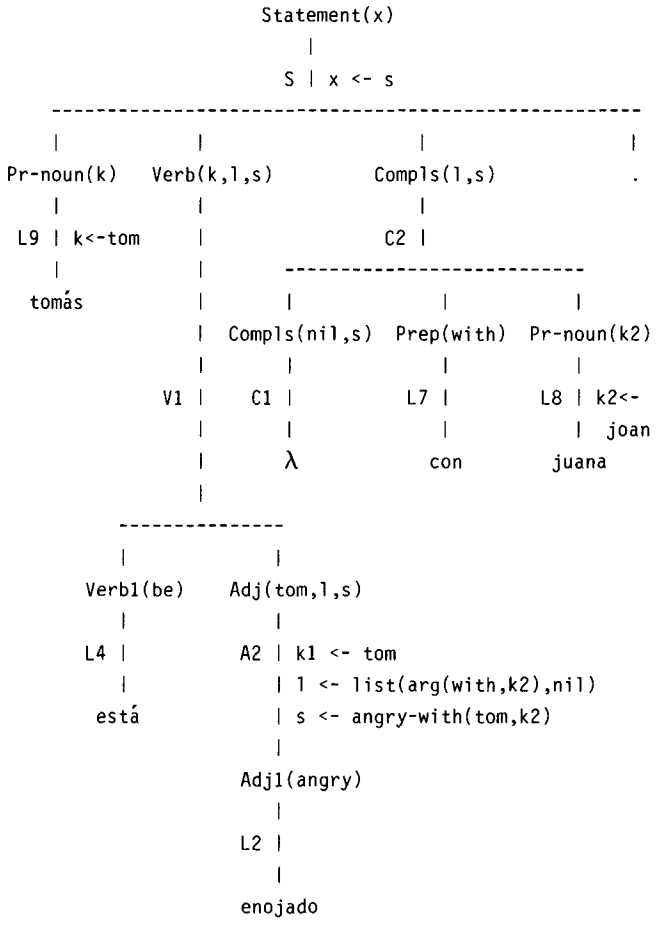


Figure 5. Derivation graph for "Tomás está enojado con Juana".

5.2 Syntactic and Semantic Agreement

Syntactic agreement can be enforced by manipulating features such as gender and number within every rule concerned with syntactic checks. For instance, the modified rules:

L2) Adj1(fem-angry) --> enojada
L2') Adj1(mas-angry) --> enojado
L8) Proper-noun(fem-joan) --> juana
L9) Proper-noun(mas-tom) --> tomás
A2) Adj(*g1-k1*,list(arg(with,*g2-k2*),nil),
angry-with(*k1*,*k2*) --> Adj1(*g1*-angry)

make it impossible to accept a sentence such as:

Tomás está enojada con Juana.
Tom is angry (+fem.) with Joan.

Semantic constraints can be enforced similarly. For a referential word to induce a distributive relation, for instance, we add a prefix such as 'dr' in the corresponding rule, e.g.:

A1) Adj(*g-k*,nil,dr(intelligent(*k*))) --> Adj(intelligent)

and establish the convention that unmarked predicates are assumed to be collective. The data base component of our system can thus distinguish each kind of relation and ensure an appropriate interpretation in each case.

Types can be represented in the same way as syntactic information. But we want them to be built up during the parse, as a function of the various types involved in a given sentence. A quick way to achieve this is through unification. We can represent a type t in a manner that reflects set inclusion relations to other types, e.g.:

nil&t&t1&...&tn

where the ti are types such that $E(t) \subset E(t1) \subset ... \subset E(tn)$, and & is a binary operator in infix notation.

Such representations may be *partially* specified, as in

v&employee&human

which can be matched with all those type representations for types contained in or equal to the "employee" type. For instance, *v* can take the values:

nil
nil&salesman
nil&manager

etc., according to the context.

In general, noun definitions will have the most weight in determining types: since it is nouns that introduce data base domains, their associated types are usually completely specified. Although this convention

might result in rejecting as semantically anomalous sentences that might deserve closer inspection (e.g. "Do all the animals speak Latin?"), it would seem a reasonable compromise between speed and coverage.

5.3 Noun Phrases

We now modify our grammar so as to handle quantified noun phrases. Agreement, both syntactic and semantic, is now left out for the sake of clarity.

For explanatory purposes, let us imagine a NL quantifier as a device which creates a variable *k* and constructs a 'for' formula *s* out of *k* and of two given formulas *s1* and *s2* (respectively standing for the noun phrase's and the verb phrase's translations). In terms of MG rules, this can be expressed as:

Determiner(*k*,*s1*,*s2*,*s*) --> det

where "det" is a given NL determiner. Two sample rules follow:

D1) Determiner(*k*,*s1*,*s2*,for(*k*,*s1*,
if(equal(card(*k*,1),*s2*)) --> el (*singular the*)

D2) Determiner(*k*,*s1*,*s2*,for(*k*,and(*s1*,not(*s2*)),
equal(card(*k*),0)) --> todo (*every*)

A noun, in turn, can be imagined as a device that takes the variable created by the quantification and constructs a relation, as in the following example:

NO1) Noun(*k*,friend(*k*)) --> amigo

We can now relate a noun phrase to a verb phrase, through the rules:

N1) Noun-phrase(*k*,*s2*,*s*) -->
Determiner(*k*,*s1*,*s2*,*s*) Noun(*k*,*s1*)

N2) Noun-phrase(*k*,*s*,*s*) --> Proper-noun(*k*)

S) Statement(*s*) -->
Noun-phrase(*k*,*s2*,*s*) Verb(*k*,*l*,*s2*) Compls(*l*,*s2*)

Thus, a noun phrase can be regarded as a device taking a formula *s2* (the verb phrase's representation), and producing a variable *k* and a formula *s* that represents the whole statement. In the case of a proper noun, *s* merely takes the same value as *s2*.

Notice that the order in which these devices are imagined to work is unimportant. They can be regarded as term (i.e., tree) constructors which fill in different gaps in those trees they share. For instance, the variable *s2* in rule S, which stands for a term of the form

r(t1,...,tn),

is given such a form by the Verb device, while the Compls device fills in the values of its arguments. The Noun-phrase device, on the other hand, can be considered a consumer of *s2*: it uses *s2* in order to build up *s*. It does not need *s2* to be completely specified, however. It merely fits *s2* into its place in *s*, expecting that sooner or later it will become completely specified.

We can now modify our rules for complements so that they will allow quantified noun phrases as well as proper nouns:

C1) Compls(nil,*s*,*s*) -->

C2) Compls(list(arg(*p*,*k*),*l*),*s1*,*s*) -->
Compls(*l*,*s1*,*s2*) Prep(*p*) Noun-phrase(*k*,*s2*,*s*)

Notice that these two simple rules are enough to handle verb, adjective, and noun complements. All we have to do is modify rules S and N2 as follows:

S) Statement(*s*) --> Noun-phrase(*k*,*s2*,*s*)
Verb(*k*,*l*,*s1*) Compls(*l*,*s1*,*s2*)

N2) Noun-phrase(*k*,*s2*,*s*) -->
Determiner(*k*,*s1*,*s2*,*s*)
Noun(*k*,*l*,*s3*) Compls(*l*,*s3*,*s*)

and add extra rules for nouns, adjectives, or verbs that accept complements, e.g.:

NO2) Noun(*k1*,list(arg(of,*k2*),nil),friend-of(*k1*,*k2*))
--> amigo

For uniformity, we rewrite NO1 into:

NO1) Noun(*k*,nil,friend(*k*)) --> amigo

The reader can now make a derivation graph for, for instance, "El amigo de Juana está enojado con Tomás" (*Joan's friend is angry with Tom*). The deep structure shown in Figure 6 should be obtained.

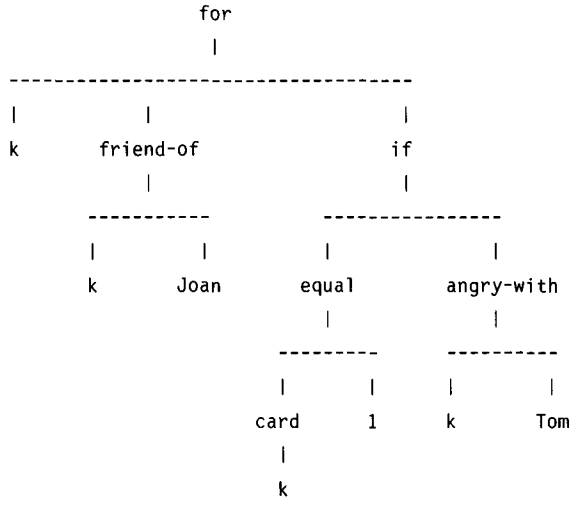


Figure 6. Internal representation for "El amigo de Juana está enojado con Tomás".

5.4 Negative Sentences

To handle negation, we can replace rule S by:

S) Statement(s) --> Kernel(l,$s1$,$s2$,s) Compls(l,$s1$,$s2$)
K) Kernel(l,$s1$,$s2$,s) --> Noun-phrase(k,$s3$,s) Neg($s2$,$s3$) Verb(k,l,$s1$)
G1) Neg(s,s) -->
G2) Neg(s,not(s)) --> no

where the Neg "device" takes a formula s and produces either s itself or not(s), according to whether the negation particle "no" is absent or present.

In sentences like "No vino ningún alumno" (*No student came*), there is subject-verb inversion, and negation is represented twice. The deep structure should read: "For every student, it is stated that he did not come". To handle this situation, we take advantage of a non-terminal Case(c) which can explicitly record the role of a given noun phrase as subject. Our rules are augmented as follows:

K) Kernel(l,$s1$,$s2$,s) --> Modifier(subject-k,$s3$,s) Neg($s2$,$s3$) Verb(k,l,$s1$)
M) Modifier(c-k,$s1$,$s2$) --> Case(c) Noun-phrase(k,$s1$,$s2$)

The subject-verb inversion rule is as follows. Its application leaves a symbol "Inv" as a marker.

I) Modifier(k,$s3$,s) Neg($s2$,$s3$) Verb(k,l,$s1$) --> Neg($s2$,$s3$) Verb(k,l,$s1$) Inv Modifier(k,$s3$,s)

This marker is used to transform a surface 'ningún' quantifier into 'todo' (every), provided it occurs in an inverted subject. Otherwise, the markers are erased:

T) Inv Case(subject) todo --> ningún
M1) Case(subject) -->
M2) Inv -->

These rules implement our general treatment of negation described earlier.

The non-terminal Case(c) is important also in handling complement noun phrases. Such noun phrases introduced by a preposition p will have associated case prep(p). Direct object noun phrases will have case "dir", and so on. We generalize rule C2 to

C2) Compls(list(arg(c,k),l),$s1$,s) --> Compls(l,$s1$,$s2$) Modifier(c-k,$s2$,s)

and add

M2) Case(prep(p)) --> Prep(p)
etc.

5.5 Interrogative and Relative Clauses

As subject-verb inversion has already been defined, we can handle Spanish yes-no questions simply by adding:

SE1) Sentence(fact(s)) --> Statement(s).
SE2) Sentence(yes-no(s)) --> Statement(s) ?

(Notice, by the way, that the analyser actually produces more information than the L3 formula s. The data base component uses this extra information to determine the form of the answer, to identify the set to be retrieved (as in rule SE3 below), etc.).

Wh-questions, on the other hand, often require modifiers to be moved around and replaced by pronouns. For instance, "Dónde vive Tomás?" (*Where does Tom live?*) can be considered as a variant for "Tomás vive en k" (*Tom lives in k*), in which "en k" has been moved to the beginning of the sentence and replaced by "Dónde".

Relative clauses usually undergo similar transformations. For instance, "El empleado cuya jefa es Juana" (*The employee whose manager is Joan*) can be considered as a variant of "El empleado [la jefa del empleado] es Juana" (*the employee [the manager of the employee] is Joan*), where "del empleado" has shifted to just before "jefa" to be subsumed, together with "la", by the relative pronoun "cuya". To handle these clauses, we use markers in the form of grammar symbols; we move the concerned modifiers and then we use context-sensitive rules to replace the appropriate constituents by a pronoun. We illustrate this for interrogative sentences such as the above example. First we add an interrogative marker:

SE3) Sentence(wh(k,s)) --> Wh-1(k) Statement(s) ?

A modifier to be moved can be handled by the extra rule:

C3) Compls(list(arg(c,k),l),$s1$,s) --> Moved-mod(k,$s2$,s) Compls(l,$s1$,$s2$)

which places it as the first complement. It must now skip the kernel so as to become the head of the sentence:

SK) Wh-1(k) Kernel(l,$s1$,$s2$,s) Moved-mod(k,$s3$,$s4$) --> Wh-2(k) Modifier(k,$s3$,$s4$) Kernel(l,$s1$,$s2$,s)

Finally, it can be replaced by a pronoun:

PR) Wh-2(k) Modifier(k,$s1$,$s2$) --> dónde

Figure 7 shows a simplified derivation graph for "Dónde vive Tomás?", from which the internal representation wh(k,live-in(Tom,k)) is obtained. Arguments and substitutions are left out in order to emphasize the structure of the derivation.

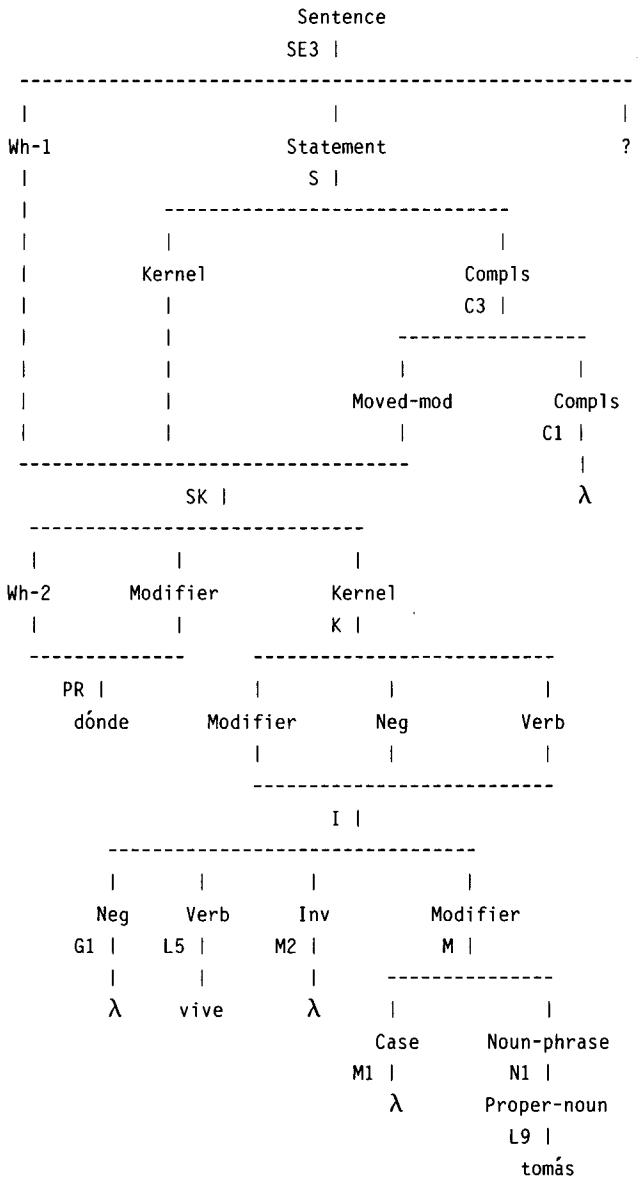


Figure 7. MG skeleton derivation graph for "Dónde vive Tomás?".

An alternative way of moving modifiers within MGs is by adding extra arguments to each non-terminal possibly dominating a modifier to be subsumed by a pronoun, as has been observed in [25]. It is however useful to be able to picture transformations through argument-stripped derivation graphs, as in the technique just exemplified. This leads naturally to ways of extending the MG formalism: Figure 7 suggests that movements might be achieved more easily if *unidentified* substrings can be referred to, so that whatever appears in between the expected pronoun and the mobile modifier can be skipped by the latter through a single rule. Such syntactic liberty is allowed in extraposition grammars (XGs) [25], where, for instance, rule PR cn be replaced by:

PR') Wh-1(k) ... Modifier(k,$s1$,$s2$) --> Dónde

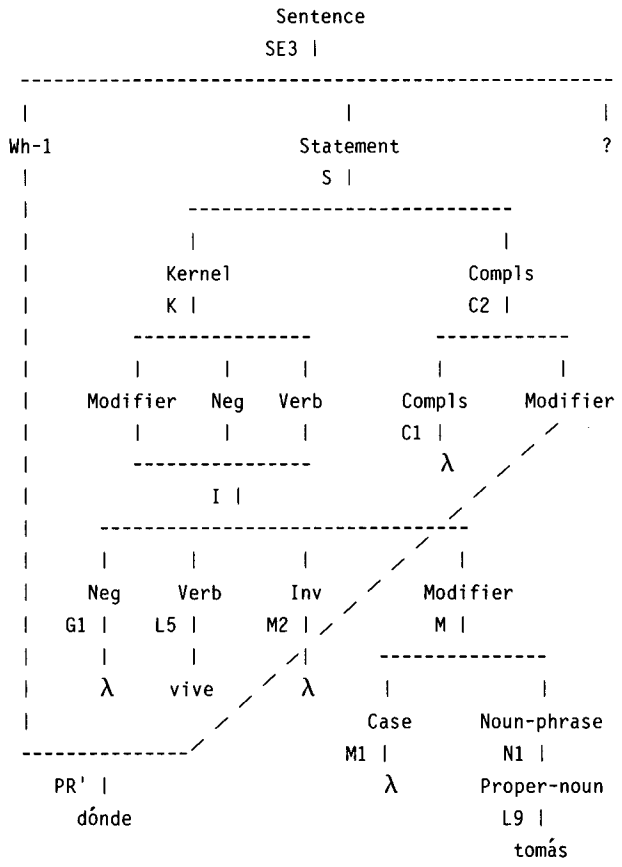


Figure 8. XG skeleton derivation graph for "Dónde vive Tomás?"

Here "..." stands for any intermediate string of symbols, which the rule's application leaves untouched to the right of "Dónde". Thus, the skeleton derivation graph would now be as shown in Figure 8. Two rules (C3 and SK) have been eliminated, and the resulting graph is clearer.

6. Extensions

Some of the limitations of our system are quite obvious from our discussion; e.g., pronoun references are not dealt with, other than for relative and interrogative pronouns. Possible extensions and related work include the following:

More flexible modifier scoping rules. The ones described here were adopted as a compromise between linguistic power and computational speed. Our choice was constrained by the inconvenience of resorting to frequent tests within grammar rules. Although allowed in PROLOG, this facility was too costly in the minicomputer version available to us. Meaning distortions resulting from too rigid quantifier scoping were, however, partly compensated for by using contextual information available through unification to choose between alternative meanings of a given determiner. An interesting treatment of modifiers in logic grammars has been recently proposed in [22]. It involves a

three-pass analyser, developed within a particularly modular framework, in which all syntactic structures are represented through a single format.

A subtler treatment of presupposition detection. Although the treatment discussed here allows for a quick detection of failed presuppositions, it fails to indicate their nature. More helpful answers should make explicit the user's wrong assumptions, and possibly correct them. The latest version of our system includes the former capability. These problems have been extensively addressed in [17], although not in the context of logic programs.

Dialogue extensions, e.g. for data base creation and updating in natural language, for clarification of the user's intended meaning, etc. Some conversational facilities have been developed recently within a logic-programmed system for consulting library information in Portuguese [2].

A wider linguistic output coverage. In its current version, our system only handles a few answer formats, constructed around the sets it retrieves and the truth values it assigns to the questions. A synthesizing grammar would be useful, particularly regarding more informative answers. Ideally, a single grammar should work both for sentence generation and parsing.

Application to other language processing problems. Notice that our choice of using English words for the deep structures in Section 5 gives the parser a translating flavour. The fact that our system has been adapted to consultation in various natural languages without substantial modifications to either the parser or the internal language's features suggests that it might be possible to use a similar framework for language translation. Another possibility is to develop a single, multilingual grammar capable of switching to the user's mother tongue as soon as the dialogue opens.

Finally, let us point out that the evolution in PROLOG's features is likely to make it possible to improve on the implementation, at least, of the ideas presented here. For instance, our set evaluation primitives rely too much upon exhaustive domain enumeration. While solving PROLOG's negation problem satisfactorily for small domains, this is inefficient for very large data bases.

Our previous solution—query reordering [6]—was on the whole more adequate, but incurred the overhead of filtering each query through a coroutining interface. A recent DEC-10 PROLOG implementation of this solution [31,32], however, has proved very efficient in the Chat-80 system, which also shares other features with SOLAR 16 (namely, the minimization of the search space through query reordering and the addition of a set constructor predicate). Some of these features have actually been incorporated as standard into recent PROLOG versions (cf. in [28]), making it possible to develop more powerful systems at a low cost.

Similarly, further MG extensions could make possible a wider linguistic coverage with no loss in efficiency. Provisions for right as well as left extraposition, for instance, would facilitate a full treatment of coordination.

7. Concluding Remarks

Similar ideas to the ones discussed in this paper (in particular, those on quantification (cf. [35])) have influenced other NL data base systems, namely LUNAR [30] and PHLIQA1 [1]. But in spite of the points in common, our general approach is markedly different. We have tried to incorporate all relevant semantic as well as syntactic NL features into a single formalism, in order to do without intermediate sublanguages and have a single process perform the analysis of an input sentence. LUNAR, on the contrary, first generates deep structures and then maps them into a semantic representation. PHLIQA1 has several successive levels of semantic analysis, each requiring a special formal language. Some of them are meant to deal with ambiguity, which in our approach, as we have seen, is dealt with through the contextual typing of variables during the quantification process.

A common disadvantage of this integrated approach—namely, that the syntactic/semantic grammar obtained is too domain-specific and therefore less transportable—is avoided by relegating all domain-specific knowledge to the domain-dependent part of the lexicon (i.e., noun, verb, and adjective definitions). Furthermore, the fact that semantic agreement is equated with syntactic well-formedness evens the relative costs of doing semantic versus syntactic tests.

The use of logic as the single formalism underlying all aspects of our system's development is a distinctive feature of the approach. Logic serves both as the theoretical framework and as the implementation means. In particular, this gives our system a definitely non-procedural flavour: our programs, as we have seen, can be understood in purely declarative terms.

The main strengths of our approach are, we feel:

Uniformity. Within our data base system, programs, parser, data, semantic interpretation and query evaluation are uniformly represented.

Formalization. Due to the generalized use of logic, important theoretical aspects—such as a rigorous characterization of our natural language subset and of the syntax and semantics of our internal query language—need not be dissociated from those practical aspects concerning the implementation.

Conciseness. We have shown how a fairly compact, one-pass analyser can suffice to process a useful and extendible natural language subset.

Clarity. The parser is modular, in the sense that each rule can be understood declaratively by itself. Problem-independent concerns (e.g. backtracking, pattern-matching, etc.) are all left to PROLOG.

Performance. These assets do not imply sacrificing efficiency. Parsing times have been shown to compare favourably against those of the LUNAR system [30], by using an adaptation of our Spanish analyser to English [24, p.276].

As we have also seen, many improvements remain to be made. With the present study we hope to motivate further research into the uses of logic for natural language processing.

Acknowledgements

The author wishes to thank Alain Colmerauer, under whose supervision the research partially reported here was developed; Michael McCord, for his encouragement to get this paper finished; and the reviewers, for their useful suggestions.

References

1. Brönnenberg W.J.H.J. et al. "The question-answering system PHLIQA1." In: *Natural Language Communication with Computers*, vol. II. L. Bolc (ed.). Carl Hanser Verlag, Munchen Wien, Macmillan, London, 1979, pp. 217-305.
2. Coelho H.M.F. "A program conversing in Portuguese providing a library service." *Ph.D. Thesis*, Univ. of Edinburgh, 1979.
3. Colmerauer A. et al. "Un système de communication homme-machine en francais." Univ. Aix-Marseille, 1973.
4. Colmerauer A. "Un sous-ensemble intéressant du francais." *R.A.I.R.O.* vol. 13, N 4, 1979.
5. Colmerauer A. "Metamorphosis grammars." In: *Natural Language Communication with Computers*, vol. I. Springer Verlag, 1978, pp. 133-189.
6. Dahl V. and Sambuc R. "Un système de banque de données en logique du premier ordre, en vue de sa consultation en langue naturelle." *D.E.A. Report*, Univ. Aix-Marseille, 1976.
7. Dahl V. "Un système déductif d'interrogation de banques de données en espagnol." *Thèse de Doctorat de Spécialité*, Univ. Aix-Marseille, 1977.
8. Dahl V. "Some experiences on natural language question-answering systems." *Proc. Workshop on Logic and Data Bases*, Toulouse, 1977.
9. Dahl V. "Quantification in a three-valued logic for natural language question-answering systems." *Proc. 6th IJCAI*, Tokyo, 1979.
10. Dahl V. "Logical design of deductive, natural language consultable data bases." *Proc. V International Conference on Very Large Data Bases*, Rio de Janeiro, 1979.
11. Dahl V. "A three-valued logic for natural language computer applications." *Proc. Tenth International Symposium on Multiple Valued Logic*, Illinois, 1980.
12. Dahl V. "Two solutions for the negation problem." *Proc. Logic Programming Workshop*, Hungary, 1980.
13. Dahl V. On database systems development through logic. To appear in: *ACM Transactions on Database Systems.*
14. Gallaire H. and Minker J. (eds.) *Logic and Data Bases.* Plenum Publ. Co., 1978.
15. Hausser R. "Quantification in an extended Montague grammar." *Dissertation*, Univ. of Texas at Austin, 1974.
16. Hendrix G.G. et al. "Developing a natural language interface to complex data." *ACM Transactions on Database Systems*, vol. 3, No. 2, June 1978.
17. Kaplan J. "Cooperative responses from a portable natural language data base query system." *MS-CIS-79-26.* Univ. of Pennsylvania, 1979.
18. Keenan E. L. "On semantically based grammars." *Linguistic Inquiry*, 1972.
19. Kowalski R. "Predicate logic as a programming language." *Proc. IFIP 74*, North-Holland Publishing Co., Amsterdam, pp. 569-574.
20. Kowalski R. *Logic for problem solving.* North-Holland, 1979.
21. Lakoff G. and Ross J.R. " Es necesaria la estructura profunda?" In: *Semantica y sintaxis en la linguistica transformatoria.* Alianza Editorial de Madrid, 1974.
22. McCord M. "Using slots and modifiers in logic grammars for natural language." *Technical Report N 69-80*, Univ. of Kentucky, 1980. To appear in *Artificial Intelligence.*
23. Pasero R. "Un essai de communication sensée en langue naturelle." Univ. Aix-Marseille, 1976.
24. Pereira F. and Warren D. "Definite clause grammars for language analysis — A survey of the formalism and a comparison with augmented transition networks." *Artificial Intelligence* 13, 1980.
25. Pereira F. "Extraposition grammars." *Proc. Logic Programming Workshop*, Hungary, 1980, pp. 231-242.
26. Petrick S.R. "On natural language based computer systems." *IBM Journal of Research and Development*, July 1976.
27. Roussel Ph. "PROLOG: manuel de référence et d'utilisation." Univ. Aix-Marseille, 1975.
28. Tärnlund S-A. (Ed.) *Logic Programming Workshop Proceedings.* Debrecen, Hungary, July 1980.
29. van Emden M.H. "Programming with resolution logic." In: *Machine Intelligence 8*, Elcock E. & Michie D. (eds.). Chichester: Ellis Horwood, 1977, pp. 266-299.
30. Warren D. et al. "PROLOG: the language and its implementation compared with LISP." *Proc. ACM Symposium on AI and Programming Languages, SIGPLAN, SIGART Newsletter*, Rochester, NY, 1977, pp. 109-115.
31. Warren D.H.D. "Efficient processing of interactive relational database queries expressed in logic." Dept. of Artificial Intelligence, Univ. of Edinburgh, 1981.
32. Warren D.H.D. and Pereira F.C.N. "An efficient easily adaptable system for interpreting natural language queries." Dept. of Artificial Intelligence, Univ. of Edinburgh, 1981.
33. Woods W.A. "Transition network grammars for natural language analysis." *Comm. ACM*, vol. 1, N 10, October, 1970.
34. Woods W.A. et al. "The lunar sciences natural language information system: Final report." *BBN Rep. 2378*, Bolt Beranek & Newman, Cambridge, Mass., 1972.
35. Woods W.A. "Semantics and quantification in natural language question answering." *Advances in Computers*, vol. 17, 1978, pp. 1-87.

Veronica Dahl is a Researcher in Computer Science for the Argentine National Council for Scientific and Technical Investigations and an Adjunct Professor in the Mathematics Department at the University of Buenos Aires. She received the Doctorat de Spécialité in Artificial Intelligence at the University of Aix-Marseille in 1977.