Contribution of a Category Hierarchy to the Robustness of Syntactic Parsing.

Damien GENTHIAL, Jacques COURTIN, Irène KOWARSKI
Laboratoire de génie informatique - Imag Campus - BP53X
F-38041 GRENOBLE CEDEX - France
Tél : 76 51 48 78
E-Mail : courtin@imag.imag.fr
courtin@imag.UUCP

Abstract

We describe how the use of a hierarchy of lexical categories instead of a simple set of categories leads to the definition of a flexible and precise language for the description of dependency structures. After specifying the formalism we use to decorate these structures, we present an application aiming to detect and correct errors in a written text. We outline how the use of the hierarchy improves the manipulation of unknown words.

1. Introduction

The work presented in this paper is part of a more general project which aims towards a complete system for detection and correction of errors in a written text. Our interest here is the creation of a syntactic-semantic module which builds dependency structures decorated with attribute-pairs lists integrating a mechanism for the inheritance of properties. We show the contribution of hierarchisation of lexical categories to the construction of syntactical structures.

2. Construction of dependency structures

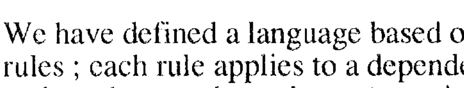
Dependency structures are trees which give a description of the structure of a sentence by establishing direct links between the words (or lexical items : the terminal symbols according to constituent grammars). The idea is that the structure of a phrase can be thought of as a particular word (the head or governor) modified by the other words (the modifiers or dependents). Dependents can themselves be modified to produce a tree strucure : the governor as root and dependents as his sons.

Complex information (e.g. syntactic functions or semantic relationships) can easily be added on the links of such trees and rules of agreement are conveniently expressed. For example, in French, the agreement in gender and number betweeen a noun and its determiner and adjectives implies the same gender and number for both the dependents (determiner and adjectives) and their governor (noun). On the other hand, it is difficult to express phrase properties on dependency structures, because the properties of a phrase governed by a word are not necessarily limited to the properties of the word alone.

In order to describe such structures, we write binary relations in "governor-dependent" form. The formalism proposed by Tesnières [20] (dependency grammars) is very precise, but all possible arrangements of the dependents of a governor must be described. In Courtin's work [8], weighted dependency relations are defined, which are well suited to computation, but limited in power of expression.

We have attempted to design a language for the description of dependency structures retaining the precision of grammars, but more appropriate for automatic treatment.

To build these structures, we must be able to determine, for any two words, caracterized by their lexical category : det, noun, verb, ..., which one governs the other. More generally, given two dependency trees, we must know how to merge them into a unique tree.
Example : [1]



We have defined a language based on rewriting rules ; each rule applies to a dependency **forest** and produces a dependency **tree.** A set of such

[1]Examples given are simple English adaptations of the French originals

rules constitutes a dependency grammar, which can be applied to a sentence by means of an interpreter. This interpreter can be viewed as a tree-transducer.

Example of a simple rule : (the "--" begins comments)

```
N_V [                                -- Name
(1:{N}, (0, $F:{P})2:{V})      -- Forest
=>
( ( 1, $F ) 2 )        -- Resulting tree
]
```

This rule applies to any forest which includes a sequence of an N and a V, whose left dependents are only preverbal particles P. It builds a new tree where the N is added as a dependent of the V.

The advantage of these rules, compared to simple binary relations, is that it is possible to express the context of each category which appears. It is thus possible to restrict a governor to one or two dependents only, or to forbid more than one occurrence of a given category,... One can also define linked pairs of binary relations, as for coordination conjunctions :

```
N_coco [
(1:{N}, 2:{coco}, 3:{N})
=>
( ( 1 ) 2 ( 3 ))
]
```

On the other hand, they present the drawback of the primitive dependency grammars : there must be a rule for almost every pair of lexical categories (LC). To avoid this problem, we have chosen to use a **hierarchy** of LCs instead of the usual **linear set** of LCs. This hierarchy is a set, partially ordered by the is-a relation (Figure 1).

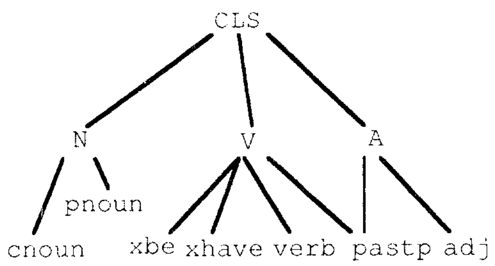


Figure 1 : Example of hierarchy

We can, in this manner, express very general rules like the two given above (N_V and N_coco) or more specific ones like :

```
aux_ppas [
(1:{xbe ; xhave}, 2:{pastp})
=>
( ( 1 ) 2 )
]
```

Thanks to is-a({cnoun, pnoun}, N) and is-a({xbe, xhave, verb, pastp}, V) relations, the N_V rule for instance may be applied to all the following pairs of categories :

```
(cnoun, xbe)        (pnoun, xbe)
(cnoun, xhave)      (pnoun, xhave)
(cnoun, verb)       (pnoun, verb)
(cnoun, pastp)      (pnoun, pastp)
```

We can thus define a set of basic categories which describe words in a very specific way, and use these categories for lexical indexing. The categories can then be grouped in "meta-categories" according to the structures we want to build. Finally, we can write the rules which effectively build these structures.

By using this method, we can avoid the usual compromise between a very fine set of LCs (which multiplies morphological ambiguities and syntactic rules) and a very general set (which multiplies syntactic ambiguities). We also obtain a fairly robust syntactic parsing : all unknown words are given the most general category (CLS), to which any rule can apply (see §4).

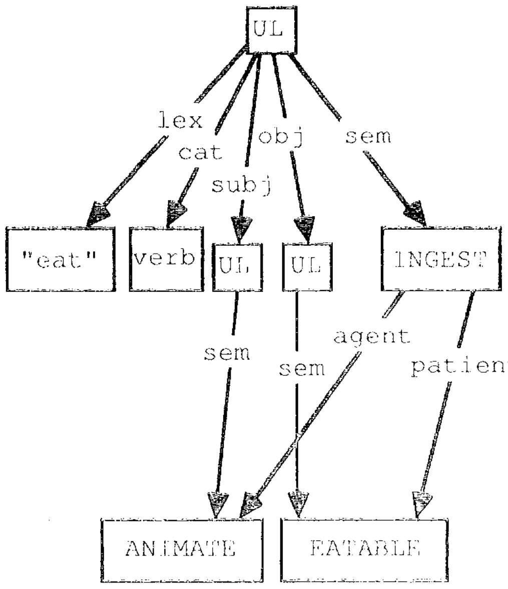
Similar type hierarchies have already been used in work on language semantics to represent the taxonomy of semantic types. We shall therefore use the same formalism for the representation of syntactic and semantic knowledge.

3. Type hierarchies and Ψ-terms

We have chosen to represent knowledge about words and trees with a unique formalism : Ψ-terms [2].

Ψ-terms are case frame structures which permit the description of types (in the sense of classical programming languages such as Pascal), i.e. sets of values. Ψ-terms are directed graphs (Figure 2) in which nodes are symbols associated to fundamental types (simple types) and arcs are labelled with attribute symbols. Each node of the graph includes a reference tag which can be used to designate it, thus allowing information to be shared.

Simple types are defined in the **signature** which is a set partially ordered by the is-a relation. This order is extended to Ψ-terms by the unique operation used to manipulate them : unification [1, 2]. The unification of two simple types is defined as the set of lower bounds of these two types (in the is-a relation).



Linear form :

```
UL(lex => "eats";
   cat => verb;
   subj => UL(sem => S:ANIMATE);
   obj => UL(sem => O:EATABLE);
   sem => INGEST(agent => S;
                      patient => O))
```

Figure 2 : Example of Ψ-term

Unification allows implicit inheritance of properties, and can be efficiently implemented [3].

Example of unification :
The two Ψ-terms :

```
UL(lex => "dog";
   cat => cnoun;
   nbr => sin;
   gnr => mas;
   sem => CANINE)
UL(cat => N;
   sem => ANIMATE)
```

unify as :

```
UL(lex => "dog";
   cat => cnoun;
   gnr => mas;
   nbr => sin;
   sem => CANINE)
```

under the condition that the associated signature unifies CANINE and ANIMATE as CANINE.

We can define a set-semantics on simple types [1, 19] ; this semantics can be extended to Ψ-terms giving the following interpretation of unification : if p_1 and p_2 are two Ψ-terms describing respectively two sets e_1 and e_2 of values, then unification of two Ψ-terms $\sqcup(p_1, p_2)$ describes the set $e_1 \cap e_2$.

To transduction rules we have added expressions which enable us to test and modify Ψ-terms attached to the trees we are manipulating. We can thus simultaneously build a syntactic structure (dependency tree) and a semantic structure (Ψ-term, which also contains morphological and syntactical information).

Example of rules and application :
We have two words :

```
UL(lex => "dog";
   cat => cnoun ;
   sem => CANINE)
UL(lex => "eats";
   cat => verb;
   subj =>
      UL(sem => S:ANIMATE);
   obj =>
      UL(sem => O:EATABLE);
   sem => INGEST(agent => S;
                      patient => O))
```

and the rule :

```
subject [ (1 : {N}, 2:{V})
/Unif(1, 2.subj)/         --- Conditions
=>
( ( 1 ) 2 ) ;
ASSIGN(2.subj, 1) ;       --- Actions
]
```

The root of the resulting tree is decorated by :

```
UL(lex => "eats";
   cat => verb;
   subj => UL(lex => "dog";
                 cat => cnoun;
                 sem => S:CANINE);
   obj => UL(sem => O:EATABLE);
   sem => INGEST(agent => S;
                      patient => O))
```

4. Applications : a robust parser of French and syntactical verification

We have implemented on a microcomputer a prototype of the dependency-tree transducer. This prototype is integrated in a system for detection and correction of errors in a written text as a syntactic filter (Figure 3) .

The prototype uses an algorithm for the application of rules adapted to syntactic-semantic parsing : the text is parsed from left to right ; each time a word is recognized by the morphological parser, it is transmitted to the syntactic module which includes it in the current state of the analysis. This state is represented by a list of dependency forests to which the transducer tries to attach the new word, according to the rules.

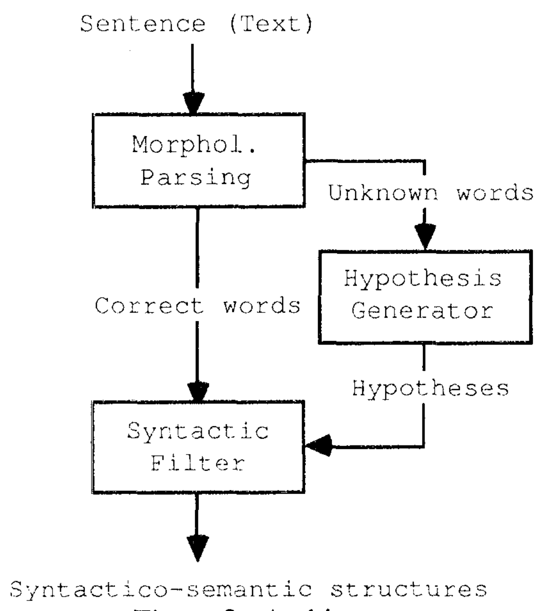


Figure 3 : Architecture

If part of the entry string is not recognized, it is passed on to the hypothesis-generator which attempts to correct it by means of three techniques (skeleton key [16], phonetics [9], and morphological generation [7]). The hypotheses are then passed on to the syntactic module which handles them exactly in the same way as morphological ambiguities. It must be noted that the three modules can function almost simultaneously (pipe-line) and that the hypothesis-generator always transmits something to the syntactic module.

If a word is so ill-formed as to render its correction impossible, the hierarchical structure of categories can be used to transmit the most general possible word, i.e. : `UL(cat => CLS)`. Any rule can apply to `CLS` (which is the most general category), so the choice of the rule to be applied is determined only by the **context** of the unknown word, and this rule will in turn determine which category the word should have had.
Example :
With a forest such as :
```
( 1:{cnoun}, 2:{coco}, 3:{CLS} )
```
we shall obtain :
```
( (1:{cnoun}) 2:{coco} (3:{N}) )
```
after applying the rule `N_coco`.

The syntactic filter works like a parser but does not take into account agreement in number and gender between words. A specialized module in charge of verification of these agreements is now being designed. A prototype of such a module has been implemented in Prolog ; it detects agreement mistakes and can propose corrections by means of a morphological generator. We are now working on rewriting it in the transducer language.

The main use of the syntactic filter is therefore to validate the lexical category of the hypotheses generated by the lexical corrector by building dependency trees which take into account the semantic information attached to the words.

Example :
With the phrase "`sun and` *`moun`*" we obtain the following hypotheses for *`moun`* :
```
UL(lex => "morn";
   cat => cnoun;
   sem => TIME)
UL(lex => "moon";
   cat => cnoun;
   sem => CELESTIAL-OBJECT)
UL(lex => "mount";
   cat => verb;
   subj => UL(sem => S:ANIMATED) ;
   obj => UL(sem => O:PLACE) ;
   sem => MOVE(agent => S ;
               where => O)
   )
```
Each of these hypotheses is considered an interpretation of the unknown word `moun`.
The rule of coordination is
```
N_coco [ (1:{N}, 2:{coco}, 3:{N})
/Unif(1.sem, 3.sem)/
=>
( ( 1 ) 2 ( 3 ) ) ;
ASSIGN(2.sem, Unif(1.sem, 3.sem));
ASSIGN(2.nbr, plu)
]
```
with for `sun` :
```
UL(lex => "sun";
   cat => cnoun;
   sem => CELESTIAL-OBJECT).
```
The rule cannot be applied to `mount` because a verb is not a `N`. It can only be applied to the noun `moon` by unification of the semantic features of `moon` and `sun`.
With a phrase such as "`sun and` *`mizrn`*", the hypothesis generator gives for *`mizrn`* :
```
UL(cat => CLS)
```
The application of the rule `N_coco` will give the tree of the figure 4.

5. Conclusion

The use of a category hierarchy simplifies the writing of the rules and introduces a way of manipulating unknown words which is not part of the mechanisms of the system but which is integrated in the objects it manipulates. We can then write rules without thinking about ill-formedness (i.e. it is not necessary to make the rules tolerant because the tolerance is implicit in the system).

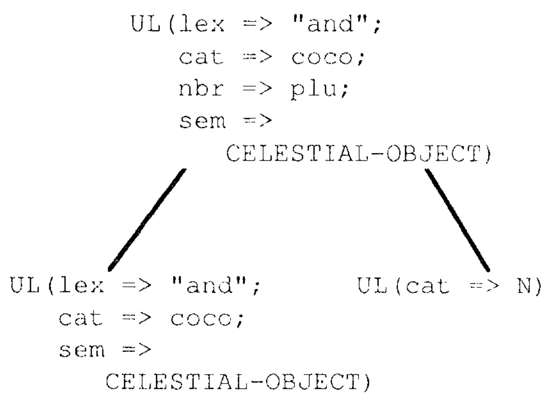


Figure 4 : Decorated tree

The three modules have each been implemented on a microcomputer, we are now working on integrating the three modules and adding the module for agreement verification. We are also improving the performance of the transducer :
- by integrating a factorization technique for the intermediate forests in the form of a graph-structured stack [21],
- by adding a finer control (graph of rule application) precomputed at compilation time.

References

[1] : H. Aït Kaci
An Algebraïc Approach to the effective resolution of type equations.
Theoretical Computer Science 45, 1986, pp 293-351

[2] : H. Aït Kaci, P. Lincoln
LIFE : A natural language for natural language.
MCC Technical Report, Number ACA-ST-074-88, February 88

[3] : H. Aït Kaci et al.
Efficient implementation of Lattice Operations.
ACM Transactions on Programming Languages and Systems 11:1, 1989, pp 116-146

[4] : C. Boitet
Representation and computation of units of translation for Machine Interpretation of spoken texts.
GETA & ATR Technical Report TR-I-0035, August 88

[5] : J.G. Carbonell & P.J. Hayes
Recovery Strategies for Parsing Extragrammatical Language.
AJCL 9:3-4, 1983

[6] : E. Charniak
On the use of framed knowledge for language compréhension.
AI 11, 1978.

[7] : B. Cohard
Logiciel de détection et de correction des erreurs lexicales.
Mémoire CNAM, Mars 1988

[8] : J. Courtin
Algorithmes pour le traitement interactif des langues naturelles.
Thèse d'état , USMG, Octobre 1977

[9] : J. Courtin, D. Dujardin, I. Kowarski, D. Genthial, V. L. Strube de Lima
Correção de erros de ortografia através da fonética em textos escritos em francês.
XIV Conferencia Latinoamericana de Informática, 17avas Jornadas Argentinas de Informática e Investigación Operativa, Buenos Aires, Sep. 1988. pp 873-891.

[10] : J. Courtin, D. Dujardin, I. Kowarski, D. Genthial, V. L. Strube de Lima
Interactive Multi-Level Systems for Correction of Ill-Formed French Texts.
Proceedings of the 2nd Scandinavian Conference on Artificial Intelligence, Tampere, Finland, June 1989

[11] : L. Emirkanian, L. Bouchard
Knowledge integration in a robust and efficient morpho-syntactic analyser for French.
12th CoLing, Budapest, August 1988, pp 166-171.

[12] : J. P. Fournier, J. Véronis
Traitement des erreurs dans la communication homme-machine en langage naturel.
Actes des premières journées nationales du GRECO-PRC Communication Homme-Machine, Paris, Novembre 88

[13] : R.H. Granger
The NOMAD System : Expectation-Based Detection and Correction of Errors during Understanding of Syntactically and Semantically Ill-Formed Text.
AJCL 9:3-4, 1983, pp 188-196,

[14] : Peter Hellwig
Dependency Unification Grammar
11th CoLing, Bonn, August 1986, 195-198.

[15] : G. Lapalme, D. Richard
Un système de correction automatique des accords des participes passés.
Techniques et Sciences Informatiques, 4, 1986

[16] : J. J. Pollock & A. Zamora
Automatic spelling correction in scientific and scholarly text.
CACM 27:4, 1984

[17] : D. Scott
Data Types as Lattices.
SIAM Journal on Computing 5:3, 1976, pp 522-587

[18] : S. M. Shieber
An Introduction to Unification-Based Approach to Grammar.
CSLI Lecture Notes 4, 1986

[19] : G. Smolka and H. Aït-Kaci
Inheritance Hierarchies : Semantics and Unification.
Journal of Symbolic Computation 7, 1989, pp 343-370

[20] : Tesnières
Eléments de syntaxe structurale.
Klincksiek, Paris, 1959

[21] : M. Tomita
Graph-structured Stack and Natural Language Parsing.
Proceedings of the 26th Annual Meeting of the ACL, Buffalo, USA, June 88

[22] : R. Zajac, M. Emele
Multiple Inheritance in RETIF.
Report of the ATR Interpreting Telephony Research Laboratories.