EFFICIENT PARSING FOR FRENCH*

Claire Gardent
University Blaise Pascal – Clermont II and University of Edinburgh, Centre for Cognitive Science,
2 Buccleuch Place, Edinburgh EH89LW, SCOTLAND, UK

Gabriel G. Bès, Pierre-François Jurie and Karine Baschung,
Université Blaise Pascal – Clermont II, Formation Doctorale Linguistique et Informatique,
34, Ave. Carnot, 63037 Clermont-Ferrand Cedex, FRANCE

ABSTRACT

Parsing with categorial grammars often leads to problems such as proliferating lexical ambiguity, spurious parses and overgeneration. This paper presents a parser for French developed on an unification based categorial grammar (FG) which avoids these problems. This parser is a bottom-up chart parser augmented with a heuristic eliminating spurious parses. The unicity and completeness of parsing are proved.

INTRODUCTION

Our aim is twofold. First to provide a linguistically well motivated categorial grammar for French (henceforth, FG) which accounts for word order variations without overgenerating and without unnecessary lexical ambiguities. Second, to enhance parsing efficiency by eliminating spurious parses, i.e. parses with different derivation trees but equivalent semantics. The two goals are related in that the parsing strategy relies on properties of the grammar which are independently motivated by the linguistic data. Nevertheless, the knowledge embodied in the grammar is kept independent from the processing phase.

1. LINGUISTIC THEORIES AND WORD ORDER

Word order remains a pervasive issue for most linguistic analyses. Among the theories most closely related to FG, Unification Categorial Grammar (UCG : Zeevat et al. 1987), Combinatory Categorial Grammar (CCG : Steedman 1985, Steedman 1988), Categorial Unification Grammar (CUG : Karttunen 1986) and Head-driven Phrase Structure Grammar (HPSG : Pollard & Sag 1988) all present inconveniences in their way of dealing with word order as regards parsing efficiency and/or linguistic data.

In UCG and in CCG, the verb typically encodes the notion of a canonical ordering of the verb arguments. Word order variations are then handled by resorting to lexical ambiguity and jump rules[1] (UCG) or to new combinators (CCG). As a result, the number of lexical and/or phrasal edges increases rapidly thus affecting parsing efficiency. Moreover, empirical evidence does not support the notion of a canonical order for French (cf. Bès & Gardent 1989).

In contrast, CUG, GPSG (Gazdar et al. 1985) and HPSG do not assume any canonical order and subcategorisation information is dissociated from surface word order. Constraints on word order are enforced by features and graph unification (CUG) or by Linear Precedence (LP) statements (HPSG, GPSG). The problems with CUG are that on the computational side, graph-unification is costly and less efficient in a Prolog environment than term unification while from the linguistic point of view (a) NP's must be assumed unambiguous with respect to case which is not true for - at least - French and (b) clitic doubling cannot be accounted for as a result of using graph unification between the argument feature structure and the functor syntax value-set. In HPSG and GPSG (cf. also Uszkoreit 1987), the problem is that somehow, LP statements must be made to interact with the corresponding rule schemas. That is, either rule schemas and LP statements are precompiled before parsing and the number of rules increases rapidly or LP statements are checked on the fly during parsing thus slowing down processing.

2. THE GRAMMAR

The formal characteristics of FG underlying the parsing heuristic are presented in §4. The characteristics of FG necessary to understand the grammar are resumed here (see (Bès & Gardent 89) for a more detailed presentation).

* The work reported here was carried out in the ESPRIT Project 393 ACORD, «The Construction and Interrogation of Knowledge Bases using Natural Language Text and Graphics».

[1] A jump rule of the form $X/Y, Y/Z \rightarrow X/Z$ where X/Y is a type raised NP and Y/Z is a verb.

FG accounts for French linearity phenomena, embedded sentences and unbounded dependencies. It is derived from UCG and conserves most of the basic characteristics of the model : monostratality, lexicalism, unification-based formalism and binary combinatory rules restricted to adjacent signs. Furthermore, FG, as UCG, analyses NP's as type-raised categories.

FG departs from UCG in that (i) linguistic entities such as verbs and nouns, sub-categorize for a set – rather than a list – of valencies ; (ii) a feature system is introduced which embodies the interaction of the different elements conditioning word order ; (iii) FG semantics, though derived directly from InL[1], leave the scope of scoping operators undefined.

The FG sign presents four types of information relevant to the discussion of this paper : (a) Category, (b) Valency set ; (c) Features ; (d) Semantics. Only two combinatory rules – forward and backward concatenation – are used, together with a deletion rule.

A *Category* can be basic or complex. A basic category is of the form *Head*, where *Head* is an atomic symbol (*n*(oun), *np* or *s*(entence)). Complex categories are of the form *C/Sign*, where *C* is either atomic or complex, and *Sign* is a sign called the active sign.

With regard to the *Category* information, the FG typology of signs is reduced to the following.

(1)Type	Category	Linguistic entities
f0	Head	verb, noun
f1	Head/f0	NP, PP, adjective, adverb, auxiliary, negative particles
f2	(f1)/sign$_i$	
	(a) sign$_i$ = f0	Determiner, complementizer, relative pronoun
	(b) sign$_i$ = f1	Preposition

Thus, the result of the concatenation of a NP (f1) with a verb (f0) is a verbal sign (f0). Wrt the concatenation rules, f0 signs are arguments ; f1 signs are either functors of f0 signs, or arguments of f2 signs. Signs of type f2 are leaves and functors.

Valencies in the *Valency Set* are signs which express sub-categorisation. The semantics of a f0 sign is a predicate with an argumental list. Variables shared by the semantics of each valency and by the predicate list, relate the semantics of the valency with the semantics of the predicate. Nouns and verbs sub-categorize not only for "normal" valencies such as *nom*(inative), *dat*(ive), etc, but also for a *mod*(ifier) valency, which is consumed and recursively reintroduced by modifiers (adjectives, PP's and adverbs). Thus, in FG the complete combinatorial potential of a predicate is incorporated into its valency set and a unified treatment of nominal and verbal modifiers is proposed. The active sign of a f1 functor indicates the valency – if any – which the functor consumes.

No order value (or directional slash) is associated with valencies. Instead, *Features* express adjacent and non-adjacent constraints on constituent ordering, which are enforced by the unification-based combinatory rules. Constraints can be stated not only between the active sign of a functor and its argument, but also between a *valency*$_i$ of a *sign*$_j$, the *sign*$_j$ and the active sign of the f1 functor consuming *valency*$_i$ while concatenating with *sign*$_j$. As a result, the valency of a verb or of a noun imposes constraints not only on the functor which consumes it, but also on subsequent concatenations. The feature percolation system underlies the partial associativity property of the grammar (cf. §4).

As mentioned above, the *Semantics* part of the sign contains an InL' formula. In FG different derivations of a string may yield sentence signs whose InL' formulae are formally different, in that the order of their sub-formulae are different, but the set of their sub-formulae are equal. Furthermore, sub-formulae are so built that formulae differing in the ordering of their sub-formulae can in principle be translated to a semantically equivalent representation in a first order predicate logic. This is because : (i) in InL', the scope of scoping operators is left undefined ; (ii) shared variables express the relation between determiner and restrictor, and between scoping operators and their semantic arguments ; (iii) the grammar places constants (i.e. proper names) in the specified place of the argumental list of the predicate. For instance, FG associates to (2) the InL' formulae in (3a) and (3b) :

(2) Un garçon présente Marie à une fille

(3) (a) [E] [ind(X) & garçon(X) & ind(Y) & fille(Y) & présenter (E,X,marie,Y)]

(b) [E] [ind(Y) & fille(Y) & ind(X) & garçon(X) & présenter (E,X,marie,Y)]

While a scoping operator of a sentence constituent is related to its argument by the index of a noun (as in the above (3)), the relation between the argument of a scoping operator and the verbal unit is expressed by the index of the verb. For instance, the negative version of (2) will incorporate the sub-formula *neg (E)*.

In InL' formulae, determiners (which are leaves and f2 signs, cf. above), immediately precede their restrictors. In formally different InL' formulae, only the ordering of scoping operators sub-formulae can differ, but this can be shown to be irrelevant with regard to the semantics. In French, scope ambiguity is the same for members of each of the following pairs, while the ordering of their corresponding semantic sub-formu-

[1] *InL* (Indexed Language) is the semantics incorporated to UCG ; it derives from Kamp's DRT. From hereafter we refer to FG semantics as *InL'*.

lae, thanks to concatenation of adjacent signs, is inescapably different.

(4) (a) Jacques avait donné *un livre (a)* à *tous les étudiants (b).*

(a') Jacques avait donné *à tous les étudiants(b) un livre (a).*

(b) Un livre a été commandé *par chaque étudiant (a) à une librairie (b).*

(b') Un livre a été commandé *à une librairie (b) par chaque étudiant (a).*

At the grammatical level (i.e. leaving aside pragmatic considerations),the translation of an InL' formula to a scoped logical formula can be determined by the specific scoping operator involved (indicated in the sub-formula) and by its relation to its semantic argument (indicated by shared variables). This translation must introduce the adequate quantifiers, determine their scope and interpret the '&' separator as either $\wedge$ or $\rightarrow$, as well as introduce $\perp$ in negative forms. For instance, the InL' formulae in (Y) translate[1] to :

(5) $\exists$E, $\exists$X, $\exists$Y (garçon(X) $\wedge$ fille(Y) $\wedge$ présenter (E,X,marie,Y)).

We assume here the possibility of this translation without saying any more on it. Since this translation procedure cannot be defined on the basis of the order of the sub-formulae corresponding to the scoping operators, InL' formulae which differ only wrt the order of their sub-formulae are said to be semantically equivalent.

3. THE PARSER

Because the subcategorisation information is represented as a set rather than as a list, there is no constraint on the order in which each valency is consumed. This raises a problem with respect to parsing which is that for any triplet X,Y,Z where Y is a verb and X and Z are arguments to this verb, there will often be two possible derivations i.e., (XY)Z and X(YZ).

The problem of spurious parses is a well-known one in extensions of pure categorial grammar. It derives either from using other rules or combinators for derivation than just functional application (Pareschi and Steedman 1987, Wittenburg 1987, Moortgat 1987, Morrill 1988) or from having unordered set valencies (Karttunen 1986), the latter case being that of FG.

Various solutions have been proposed in relation to this problem. Karttunen's solution is to check that for any potential edge, no equivalent analysis is already stored in the chart for the same string of words. However as explained above, two *semantically equivalent* formulae of InL' need not be *syntactically identical*. Reducing two formulae to a normal form to check their equivalence or alternatively reducing one to the other might require 2^n permutations with n the number of predicates occuring in the formulae. Given that the test must occur each time that two edges stretch over the same region and given that it requires exponential time, this solution was disguarded as computationally inefficient.

Pareschi's lazy parsing algorithm (Pareschi, 1987) has been shown (Hepple, 1987) to be incomplete. Wittenburg's predictive combinators avoid the parsing problem by advocating grammar compilation which is not our concern here. Morill's proposal of defining equivalence classes on derivations cannot be transposed to FG since the equivalence class that would be of relevance to our problem i.e., ((X,Z)Y, X(ZY)) is not an equivalence class due to our analysis of modifiers. Finally, Moortgat's solution is not possible since it relies on the fact that the grammar is structurally complete[1] which FG is not.

The solution we offer is to augment a shift-reduce parser with a heuristic whose essential content is that no same functor may consume twice the same valency. This ensures that for all semantically unambiguous sentences, only one parse is output. To ensure that a parse is always output whenever there is one, that is to ensure that the parser is complete, the heuristic only applies to a restricted set of edge pairs and the chart is organized as a queue. Coupled with the partial-associativity of FG, this strategy guarantees that the parser is complete (cf. §4).

3.1 THE HEURISTIC

The heuristic constrains the combination of edges in the following way[2].

Let *e1* be an edge stretching from *S1* to *E1* labelled with the type *f0*, a predicate identifier *p1* and a sign *Sign1*, let *e2* be an edge stretching from *E1* to *S2* labelled with type *f1* and a sign *Sign2*, then *e2* will reduce with *e1* by consuming the valency *Val* of *p1* if *e2* has not already reduced with an edge *e1'* by consuming the valency *Val* of *p1* where *e1'* stretches from *S1'* to *E1* and *S1'* $\neq$ *S1*.

In the rest of this section, examples illustrate how

[1] In (5) $\exists$E can be paraphrased as "There exists an event".

[1] A structurally complete grammar is one such that :
If a sequence of categories X1 .. Xn reduces to Y, there is a reduction to Y for any bracketing of X1 .. Xn into constituents (Moortgat, 1987).

[2] A more complete difinition is given in the description of the parsing algorithm below.

this heuristic eliminates spurious parses, while allowing for real ambiguities.

Avoiding spurious parses

Consider the derivation in (6)

```
(6)  Jean    aime    Marie
     0 - Ed1 - 1 - Ed2 - 2 - Ed3-- 3
     0------- Ed4 ------- 2                 Ed4 = Ed1(Ed2,pl,subj)
     0------- Ed5 ------- 2                *Ed5 = Ed1(Ed2,pl,obj)
               1 ------- Ed6 ------- 3      Ed6 = Ed3(Ed2,pl,obj)
               1-------- Ed7-------  3      Ed7 = Ed3(Ed2,pl,subj)
     0 -------Ed8 ------------------- 3     Ed8 = Ed1(Ed6,pl,subj)
     0 ------ Ed9 ------------------- 3    *Ed9 = Ed3(Ed4,pl,obj)
     0 ------ Ed10 ------------------ 3    *Ed10 = Ed1(Ed7,pl,obj)
```

where Ed4 = Ed1(Ed2,pl,subj) indicates that the edge Ed1 reduces with Ed2 by consuming the subject valency of the edge Ed2 with predicate pl.

Ed5 and Ed10 are ruled out by the grammar since in French no lexical (as opposed to clitics and wh-NP) object NP may appear to the left of the verb. Ed9 is ruled out by the heuristic since Ed3 has already consumed the object valency of the predicate p1 thus yielding Ed6. Note also that Ed1 may consume twice the subject valency of p1 thus yielding Ed4 and Ed8 since the heuristic does not apply to pairs of edges labelled with signs of type f1 and f0 respectively.

Producing as many parses as there are readings

The proviso that a functor edge cannot combine with two different edges by consuming twice the same valency *on the same predicate* ensures that PP attachment ambiguities are preserved. Consider (7) for instance[1].

```
(7)  Regarde      le      chien      dans la rue
     0 --Ed1 --- 1 ---Ed2 - 2 - Ed3 ---- 3 --- Ed4 ------- 4
                 1 ----- Ed5 ----------- 3
     0 ------------------ Ed6 ----------- 3
                          2 ------------- Ed7 ---------- 4
                 1 ------ Ed8 ----------------------------- 4
     0 ------------------ Ed9 ----------------------------- 4
     0 ------------------ Ed10 ---------------------------- 4
```

with Ed7 = Ed4(Ed3,p2,mod)
 Ed8 = Ed2(Ed7)
 Ed9 = Ed8(Ed1,p1,obj)
 Ed10 = Ed4(Ed6,p1,mod)

where p1 and p2 are the predicate identifiers labelling the edges Ed1 and Ed3 respectively.

The above heuristic allows a functor to concatenate twice by consuming two different valencies. This case of real ambiguity is illustrated in (8).

```
(8)  Quel homme    présente     Marie      à Rose ?
     0 ---- Ed1----- 1 --- Ed2 -- 2 -- Ed3--- 3 -- Ed4--- 4
                     1 ---------- Ed4 -------- 3
                     1 ---------- Ed5 -------- 3
     0 ---------------- Ed6 ------------------- 3
     0 ---------------- Ed7 ------------------- 3
```

where Ed4 = (Ed3,p1,nom)
and Ed5 = (Ed3,p1,obj)

Thus, only edges of the same length correspond to two different readings. This is the reason why the heuristic allows a functor to consume twice the same valency on the same predicate iff it combines with two edges E and E' that *stretch over the same region*. A case in point is illustrated in (9)

```
(9)  Quel homme    présente     Marie      à Rose ?
     0 ---- Ed1----- 1 --- Ed2 -- 2 -- Ed3--- 3 -- Ed4--- 4
                     1 ---------- Ed5 -------- 3
                     1 ---------- Ed6 -------- 3
                     1 ---------- Ed7 --------------------- 4
                     1 ---------- Ed8 --------------------- 4
     0 ---- Ed9 -------------------------------------------- 4
     0 ---- Ed10 ------------------------------------------- 4
```

where *a Rose* concatenates twice by consuming twice the same – dative – valency of the same predicate.

3.2 THE PARSING ALGORITHM

The parser is a shift-reduce parser integrating a chart and augmented with the heuristic.

An edge in the chart contains the following information :

edge [Name, Type, Heur, S,E, Sign]

where Name is the name of the edge, S and E identifies the starting and the ending vertex and Sign is the sign labelling the edge. Type and Heur contain the information used by the heuristic. Type is either f0, f1 and f2 while the content of Heur depends on the type of the edge and on whether or not the edge has already combined with some other edge(s).

Heur

f0 pX where X is an integer.
 pX identifies the predicate associated with any edge.
 type f0

f1 before combination : Var
 where Var is the anonymous variable. This indicates that there is as yet no information available that could violate the heuristic.
 after combination : Heur-List
 where Heur-List is a list of triplets of the form [Edge,pX,Val] and Edge indicates an argument edge with which the functor edge has combined by consuming valency Val of the predicate pX label-

[1] For the sake of clarity, all irelevant edges have been omitted. This practice will hold throughout the sequel.

ling Edge.

f2 nil

The basic parsing algorithm is that of a normal shift-reduce parser integrating a chart rather than a stack i.e.,

1. Starting from the beginning of the sentence, for each word W either shift or reduce ,
2. Stop when there is no more word to shift and no more reduce to perform,
3. Accept or reject.

Shifting a word W consists in adding to the chart as many lexical edges as there are lexical entries associated with W in the lexicon. Reducing an edge E consists in trying to reduce E with any adjacent edge E' already stored in the chart. The operation applies recursively in that whenever a new edge E" is created it is immediately added to the chart and tried for reduction. The order in which edges tried for reduction are retrieved from the chart corresponds to organising the chart as a queue i.e., first-in-first-out. Step 3 consists in checking the chart for an edge stretching from the beginning to the end of the chart and labelled with a sign of category *s*(entence). If there is such an edge, the string is accepted – else it is rejected.

The heuristic is integrated in the *reduce* procedure which can be defined as follows.

Two edges Edge1 and Edge2 will reduce to a new edge Edge3 iff

Either (a)

1. Edge1 = [e1,Type1,H1,E2,Sign1] and
2. Edge2 = [e2,Type2,H2,E2,Sign2] and <Type1,Type2> ≠ <f0,f1> and
3. apply(Sign1,Sign2,Sign3) and
4. Edge3 = [e3,Type3,H3,E3,Sign3] and <S3,E3> = <S1,E2>

or (b)

1. Edge1 = [e1,f0,p1,S1,E1,Sign1] and
2. Edge2 = [e2,f1,H2,S2,E2,Sign2] and E1 = S2 and
3. bapply(Sign1,Sign2,Sign3) by consuming the valency Val and
4. H2 does not contain a triplet of the form [e1',p1,Val] where Edge 1' = [e1',f0,p1,S'1,S2] and S'1≠S1
5. Edge3 = [e3,f0,p1,S1,E2,Sign3]
6. The heuristic information H2 in Edge2 is updated to [e1,p1,Val]+H2

where '+ 'indicates list concatenation and under the proviso that the triplet does not already belong to H2.

Where *apply(Sign1,Sign2,Sign3)* means that Sign1 can combine with Sign2 to yield Sign3 by one of the two combinatory rules of FG and *bapply* indicates the backward combinatory rule.

This algorithm is best illustrated by a short example. Consider for instance, the parsing of the sentence *Pierre aime Marie.* Step1 shifts *Pierre* thus adding Edge1 to the chart. Because the grammar is designed to avoid spurious lexical ambiguity, only one edge is created.

Edge1 = [e1,f1,_,0,1,Sign1]

Since there is no adjacent edge with which Edge1 could be reduced, the next word is shifted i.e., *aime* thus yielding Edge2 that is also added to the chart.

Edge2 = [e2,f0,p1,1,2,Sign2]

Edge2 can reduce with Edge1 since Sign1 can combine with Sign2 to yield Sign3 by consuming the subject valency of the predicate *p1*. The resulting edge Edge3 is added to the chart while the heuristic information of the functor edge Edge1 is updated :

Edge3 = [e3,f0,p1,0,2,Sign3]
Edge1 = [e1,f1,[[e3,p1,subj]],0,1,Sign1]

No more reduction can occur so that the last word *Marie* is shifted thus adding Edge4 to the chart.

Edge4 = [e4,f1,_,2,3,Sig4]

Edg4 first reduces with *Edeg2* by consuming the *subject* valency of p1 thus creating *Edge5*. It also reduces with *Edge2* by consuming the *object* valency of p1 to yield *Edge6*.

Edge5 = [e5,f0,p1,1,3,Sign5]
Edge6 = [e6,f0,p1,1,3,Sign6]

Edge4 is updated as follows.

Edge4 = [e4,f1,[[e2,p1,subj],[e2,p1,obj]],2,3,Sign4]

At this stage, the chart contains the following edges.

```
  Pierre    aime      Marie
0 — e1 — 1 — e2 — 2 — e4 — 3
0 ——— e3 ——— 3
          1 ————— e5 — 3
          1 ————— e6 — 3
```

Now *Edge1* can reduce with *Edge6* by consuming the *subject* valency of p1 thus yielding *Edge7*. However, the heuristic forbids *Edge4* to consume the *object* valency of p1 on *Edge3* since *Edge4* has already consumed the object valency of p1 when combining with *Edge2* . In this way, the spurious parse *Edge8* is avoided.

The final chart is as follows.

```
  Pierre    aime        Marie
0 — e1 — 1 — e2 —— 2 — e4 — 3
0 ——— e3 ———— 3
          1 —————— e5 — 3
          1 —————— e6 — 3
0 ——— e7 ———————————— 3
*0 ——— e8 ——————————— 3
```

with
Edge7 = [e7,f0,p1,0,3,Sign7]
Edge4 = [e4,f1,[[e2,p1,subj],[e2,p1,obj]],2,3,Sign4]
Edge1 = [e1,f1,[[e2,p1,subj]],0,1,Sign1]

4. UNICITY AND COMPLETNESS OF THE PARSING

DEFINITIONS

1. An indexed lexical f0 is a pair <X,i> where *X* is a lexical sign of f0 type (c.f. 2) and *i* is an integer.

2. PARSE denotes the free algebra recursively defined by the following conditions.

2.1 Every lexical sign of type f1 or f2, and every indexed lexical f0 is a member of PARSE.

2.2 If P and Q are elements of PARSE, i is an integer, and k is a name of a valency then $(P+_{ik}Q)$ is a member of PARSE.

2.3 If P and Q are elements of PARSE, $(P+_{i\varpi}Q)$ is a member of PARSE, where ϖ is a new symbol.[1]

3. For each member, P, of PARSE, the string of the leaves of P is defined recursively as usual :

3.1 If P is a lexical functor or a lexical indexed argument, L(P) is the string reduced to P.

3.2 $L(P+_{ik}Q)$ is the string obtained by concatenation of L(P) and L(Q).

4. A member P of PARSE, is called a well indexed parse (WP) if two indexed leaves which have different ranges in L(P), have different indicies.

5. The partial function, S(P), from the set of WP to the set of signs, is defined recursively by the following conditions :

5.1 If P is a leave S(P) = P

5.2 $S(F+_{ik}A) = Z$ [resp. $S(A+_{ik}F) = Z$] (k ϖ)

If S(F) is a functor of type f1, S(A) is an argument and Z is the result sign by the FC rule [resp. BC rule] when S(F) consumes the valency named k in the leave of S(A) indexed by i.

5.3 $S(F+_{i\varpi}A) = Z$ [res. $S(A+_{i\varpi}F) = Z$] if S(F) is a functor of type f1 or f2, S(A) is an argument sign and Z is the result sign by the FC rule [resp. BC rule].

6. For each pair of signs X and Y we denote $X \cong Y$ if X and Y are such that their non semantic parts are formally equal and their semantic part is semantically equivalent.

[1] In 2.3 the index i is just introduced for notational convenience and will not be used ; *k,l*... will denote a valency name or the symbol ϖ.

7. If P and Q are WP
$P \cong Q$ iff
7.1 S(P) and S(Q) are defined
7.2 $S(P) \cong S(Q)$ and
7.3 L(P) = L(Q)

8. A WP is called *accepted* if it is accepted by the parser augmented with the heuristic described in §3.

THEOREM

1. (Unicity) If P and Q are accepted WP's and if $P \cong Q$, then P and Q are formally equal.

2. (Completeness) If P is a WP which is accepted by the grammar, and S(P) is a sign corresponding to a grammatical sentence, then there exists a WP Q such that :

a) Q is accepted, and

b) $P \cong Q$.

NOTATIONAL CONVENTION

F,F'...(resp. A,A',...) will denote WP's such that S(F), S(F')...are functors of type f1 (resp. S(A), S(A')..., are arguments of type f0).

The proof of the theorem is based on the following properties 1 to 3 of the grammar. Property 1 follows directly from the grammar itself (cf. §2) ; the other two are strong conjectures which we expect to prove in a near future.

PROPERTY 1 If S(K) is defined and L(K) is not a lexical leaf, then :

a) If K is of type f0, there exist i,k,F and A such that :
$K = F+_{ik}A$ or $K = A+_{ik}F$

b) If K is of type f1, there exist Fu of type f2 and Ar of type f0 or of type f1 such that :
$K = Fu+_{i\varpi}Ar$

c) K is not of type f2.

PROPERTY 2 (Decomposition unicity) :

For every i and k
if $F+_{ik}A \cong F+_{i'k'}A'$, or $A+_{ik}F \cong A'+_{i'k'}F'$
then i = i', k = k', $A \cong A'$ and $F \cong F'$

PROPERTY 3 (Partial associativity) :

For every F,A,F' such that L(F) L(A) L(F') is a substring of a string of lexical entries which is accepted by the grammar as a grammatical sentence,

a) If $S[F+_{ik}(A+_{jl}F')]$ and $S[(F+_{ik}A)+_{jl}F']$ are defined, then $F+_{ik}(A+_{jl}F') \cong (F+_{ik}A)+_{jl}F'$

b) If $S[A+_{jl}F']$ and $S[(F+_{ik}A)+_{jl}F']$ are defined, then $S[F+_{ik}(A+_{jl}F')]$ is also defined.

LEMMA 1

If $F+_{ik}A \cong A'+_{jl}F$

then $A'+_{jl}F'$ is not accepted.

Proof : L(F) is a proper substring of L(A), so there exists A" such that :

a) $S(A''+_{jl}F)$ is defined, and

b) L(A") is a substring of L(A)

But A' begins by F and F is not contained in A", so A" is an edge shorter than A'. Thus A'+F' is not accepted.

LEMMA 2

If $S[(A+_{ik}F)+_{il}F']$ is defined and

$A+_{ik}F$ is accepted, then

$(A+_{ik}F)+_{il}F$ is also accepted.

Proof : Suppose, a contrario, that $(A+_{ik}F)+_{il}F$ is not accepted. Then there must exist an edge

$A' = A''+_{ik}F$ such that :

a) $S(A'+_{il}F')$ is defined, and

b) A' is shorter than $A+_{ik}F$

This implies that A" is shorter than A.

Therefore $A+_{ik}F$ would not be accepted.

PROOF OF THE PART 1 OF THE THEOREM

The proof is by induction on the lengh, lg(P), of L(P). So we suppose a) and b) :

a) (induction hypothesis). For every P' and Q' such that P' and Q' are accepted, if $P' \cong Q'$, and $lg(P') < n$, then P' = Q'

b) P and Q are accepted, $P \cong Q$ and $lg(P) = n$
and we have to prove that

c) P = Q.

First cas : if lg(P) = 1, then we have

P = L(P) = L(Q) = Q.

Second cas : if lg(P) > 1, then we have

lg(Q) > 1 since L(P) = L(Q). Thus there exist P'_1, P'_2, Q'_1, Q'_2, i, k, j, l, such that

$P = P'_1+_{ik}P'_2$ and $Q = Q'_1+_{jl}Q'_2$

By the Lemma 1 P'_1 and Q'_1 must be both functors or both arguments. And if P'_1 and Q'_1 are functors (res. arguments) then P'_2 and Q'_2 are arguments (resp. functors). So by Property 2, we have :

$i = i'$, $k = k'$, $P'_1 \cong Q'_1$, and $P'_2 \cong Q'_2$.

Then the induction hypothesis implies that $P'_1 = Q'_1$ and that $P'_2 = Q'_2$. Thus we have proved that P = Q.

PROOF OF THE PART 2 OF THE THEOREM

Let P be a WP such that S(P) is define and corresponds to a grammatical sentence. We will prove, by induction on the lengh of L(K), that for all the subtrees K of P, there exists K' such that :

a) K' is accepted, and

b) $K \cong K'$.

We consider the following cases (Property 1)

1. If K is a leaf then K' = K

2. If $K = F+_{ik}A$, then by the induction hypothesis there exist F' and A' such that :
   (i) F' and A' are accepted, and
   (ii) $F \cong F'$, $A \cong A'$.

Then F+A' is also accepted. So that K' can be choosed as F'+A'.

3. If $K = A+_{ik}F$, we define F', A' as in (2) and we consider the following subcases :

3.1 If $A' = F1+_{jl}A1$ where $S(A1+_{ik}F')$ is not defined, then $A'+_{ik}F'$ is accepted, and we can take it as K.

3.2 If $A' = A1+_{il}F1$, then by the Lemma 2 $A'+_{ik}F$ is accepted. Thus we can define K' as $A'+_{ik}F'$.

3.3 If $A' = F1+_{il}A1$ and $S(A1+_{ik}F')$ is defined.
Let $A2 = A1+_{ik}F$.

By the Property 3 $S(F1+_{jl}A2)$ is defined

and $K \cong A'+_{ik}F \cong F1+_{jl}A2$.

Thus this case reduces to case 2.

4. If $K = Fu+_{i\omega}Ar$, where Fu is of type f2 and Ar is of type f0 or f1, then by induction hypothesis there exists Ar' such that $Ar \cong Ar'$ and Ar' is accepted. Then K can be defined as $Fu+_{i\omega}Ar'$.

5. IMPLEMENTATION AND COVERAGE

FG is implemented in PIMPLE, a PROLOG term unification implementation of PATR II (cf. Calder 1987) developed at Edinburgh University (Centre for Cognitive Studies). Modifications to the parsing algorithm have been introduced at the "Université Blaise Pascal", Clermont-Ferrand. The system runs on a SUN M 3/50 and is being extensively tested. It covers at present : declarative, interrogative and negative sentences in all moods, with simple and complex verb forms. This includes yes/no questions, constituent questions, negative sentences, linearity phenomena introduced by interrogative inversions, semi free constituent order, clitics (including reflexives), agreement phenomena (including gender and number agreement between obj NP to the left of the verb and participles), passives, embedded sentences and unbounded dependencies.

REFERENCES

Bès, G.G. and C. Gardent (1989) French Order without Order. To appear in the *Proceedings of the Fourth European ACL Conference* (UMIST, Manchester, 10-12 April 1989), 249-255.

Calder, J. (1987) *PIMPLE ; A PROLOG Implementation of the PATR-II Linguistic Environment.* Edinburgh, Centre for Cognitive Science.

Gazdar, G., Klein, E., Pullum, G., and Sag., I. (1985) *Generalized Phrase Structure Grammar.* London : Basil Blackwell.

Kamp, H. (1981) A Theory of Truth and Semantic Representation. In Groenendijk, J. A. G., Janssen, T. M. V. and Stokhof, M. B. J. (eds.) *Formal Methods in the Study of Language,* Volume 136, 277-322. Amsterdam : Mathematical Centre Tracts.

Karttunen, L. (1986) Radical Lexicalism. Report No. CSLI-86-68, Center for the Study of Language and Information, Paper presented at the Conference on Alternative Conceptions of Phrase Structure, July 1986, New York.

Morrill, G. (1988) *Extraction and Coordination in Phrase Structure Grammar and Categorial Grammar.* PhD Thesis, Centre for Cognitive Science, University of Edinburgh.

Pareschi, R. (1987) Combinatory Grammar, Logic Programming, and Natural Language. In Haddock, N. J., Klein, E. and Morill, G. (eds.) *Edinburgh Working Papers in Cognitive Science,* Volume I ; *Categorial Grammar, Unification Grammar and Parsing.*

Pareschi, R. and Steedman, M. J. (1987) A Lazy Way to Chart-Parse with Extended Categorial Grammars. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics,* Stanford University, Stanford, Ca., 6-9 July, 1987.

Pollard, C. J. (1984) *Generalized Phrase Structure Grammars, Head Grammars, and Natural Languages.* PhD Thesis, Stanford University.

Pollard, C. J. and Sag, I. (1988) *An Information-Based Approach to Syntax and Semantics : Volume 1 Fundamentals.* Stanford, Ca. : Center for the Study of Language and Information.

Steedman, M. (1985) Dependency and Coordination in the Grammar of Dutch and English. *Language,* 61, 523-568.

Steedman, M. (1988) Combinators and Grammars. In Oehrle, R., Bach, E. and Wheeler, D. (eds.) *Categorial Grammars and Natural Language Structures,* Dordrecht, 1988.

Uszkoreit, H. (1987) *Word Order and Constituent Structure in German.* Stanford, CSLI.

Wittenburg, K. (1987) Predictive Combinators : a Method for Efficient Processing of Combinatory Categorial Grammar. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics,* Stanford University, Stanford, Ca., 6-9 July, 1987.

Zeevat, H. (1986) A Specification of InL. *Internal ACORD Report.* Edinburgh, Centre for Cognitive Science.

Zeevat, H. (1988) Combining Categorial Grammar and Unification. In Reyle, U. and Rohrer, C. (eds.) *Natural Language Parsing and Linguistic Theories,* 202-229. Dordrecht : D. Reidel.

Zeevat, H., Klein, E. and Calder, J. (1987) An Introduction to Unification Categorial Grammar. In Haddock, N. J., Klein, E. and Morrill, G. (eds.) *Edinburgh Working Papers in Cognitive Science,* Volume 1 : *Categorial Grammar, Unification Grammar and Parsing*