A Linguistic Engineering Environment using LFG (Lexical Functionnal Grammar) and CG (Conceptual Graphs)

Xavier Briffault, Karim Chibout, Gérard Sabah, Jérôme Vapillon
{briffault,chibout,gs,vap}@limsi.fr

Groupe Langage et Cognition
LIMSI - CNRS
B.P. 133,
91403 ORSAY Cedex
FRANCE

Proceedings of the LFG97 Conference
University of California, San Diego

Miriam Butt and Tracy Holloway Kings (Editors)

1997
CSLI Publications
http://www-csli.stanford.edu/publications/

A Linguistic Engineering Environment using LFG (Lexical Functionnal Grammar) and CG (Conceptual Graphs)

Xavier Briffault, Karim Chibout, Gérard Sabah, Jérôme Vapillon

Groupe Langage et Cognition - LIMSI-CNRS

Abstract

In order to help computational linguists, we have conceived and developed a linguistic software engineering environment, whose goal is to set up reusable and evolutive toolkits for natural language processing. This environment is based on a set of natural language processing components, at the morphologic, syntactic and semantic levels. These components are generic and evolutive, and can be used separately or with specific problem solving units in global strategies built for man-machine communication (according to the general model developed in the Language and Cognition group: Caramel). All these tools are complemented with graphic interfaces, allowing users outside the field of

Computer Science to use them very easily. In this paper, we will present first the syntactic analysis, based on a chart parser that uses a LFG grammar for French, and the semantic analysis, based on conceptual graphs. Then we will show how these two analyses collaborate to produce semantic representations and sentences. Before concluding, we will show how these modules are used through a distributed architecture based on Corba (distributed Smalltalk) implementing the Caramel multi-agent architecture.

Introduction

Generalities

Natural language processing is nowadays strongly related to Cognitive Science, since linguistics, psychology and computer science have to collaborate to produce systems that are useful for man-machine communication. This collaboration has allowed formalisms that are both theoretically well-founded and implementable to emerge. In this paradigm, we have conceived and developed a linguistic software engineering environment, whose goal is to set up reusable and evolutive toolkits for natural language processing (including collecting linguistic data, analysing them and producing useful data for computer processes). Based on a large number of graphical, very intuitive, interfaces, this environment has two main goals:

Figure 1: symboles used in the figures

to provide tools usable by users outside the field of Computer Science (e.g., computational linguists) for them to be able to easily collect data and test their linguistic hypotheses
to allow computer scientists to exploit these data in computer programs

Remark: in the text, some figures describe the structure of our tools; we have used Booch's conventions (1994) about object oriented analysis and conception. They are summarized here:

Extensions to LFG formalism

Four types of equations are defined in classical LFG (Bresnan, 1981):
unifying structures (symbolised by the sign "="),
constrained unification of structures, only true if a feature is present in both structures, but may not be added (symbol "=c"),
obligatory <u>presence</u> of a feature (symbol "&"),
obligatory <u>absence</u> of a feature (symbol "~" -tilde-).

We have defined three non-standard types of equations used in our parser:
obligatory difference between two values (symbol "<>"),
disjunction of obligatory differences (a sequence of obligatory differences separated by the symbol "|")
(this can also be viewed as the negation of a conjonction of obligatory presences)
prioritary union, copy into a F-Structure the attributes of the other that are not present in the first one, nor inconsistent with it.

The LFG Environment

Foundation: a LFG parser

According to the principles of lexical functional grammars, the process of parsing a sentence is decomposed into the construction of a constituent parts structure (c-structure) upon which a functional structure is inserted. C-structure construction is based on a chart parser, that allows the system to represent syntactic ambiguities (cf. (Kay, 1967, Winograd, 1983)). In order to be used within a LFG parser, a classical chart has to be complemented with a new concept: *completed arcs* (which represent a whole syntactic structure) have to be differenciated between completed arcs linked with a correct F-Structure, and those which are linked to an F-Structure that cannot be unified or that does not respect well-formedness principles.

Visualising the Chart

Figure 2: The Chart Interface

In the Chart interface, words are separated by nodes, numbered from 1 to number_of_words + 1. Each arc is represented by a three segment polygon () (larger arcs are above the narrower, for readibility reason).

Active arcs are grey and positioned under the words. Completed arcs with uncorrect F-Structures are red and also placed under the words. Completed arcs with correct F-Structures are blue and above the words. Lastly, completed arcs with F-Structures that don't respect well formedness principles are grey and above the words. The user can select the kind of arc he is interested in. By clicking on an arc with the left button, the arc and all its daughters become green, thus showing the syntactic hierarchy. By clicking with the middle button, a menu appears within which one can choose to examine the applied rule or the F-Structures (see below for the corresponding interface).

Visualising F-Structures

As shown in Figures 3 & 4, F-Structures are represented by attribute-value pairs (a value may itself be a F-Structure). In addition to such a graphical representation, a linear representation (more suitable for storing data on files or printing them) has been developed and it is possible to switch from one to the other. This allows us to keep track of previous results and to use them for testing the evolution of the system.

Lexicon and lexicon management

Since LFG is a "lexical" grammar, it is important to have powerful and easy to use lexicon management tools. To be as flexible as possible, we have choosen to use several lexica at the same time in the same analyser. The lexicon manager contains a list of lexica ordered by access priority. For each word analysed, the list is searched, and the first analysis encountered is returned.

Figures 3 & 4: graphical and textual representations of a F-Structure

Two kinds of lexica are currently used; this kind of structuration is quite flexible:

if the user uses a big lexicon, but wants to redefine a few items for his own needs, he just has to define a new small lexicon containing the modified items, and to give it a high priority.
if the user has a big lexicon with a slow access, the access can be optimised by putting the words frequently used in a direct access lexicon stored in memory.

Our lexicon currently contains 7000 verbs, all the closed classes words (e.g., prepositions, articles, conjunctions), 12000 nouns and about 2500 adjectives. To mitigate the consequences of some lacks of this lexicon, a set of subcategorisation frames is independently associated with the lexicon (3000 frames).

The user may also define a direct access lexicon, whose equations are written in the standard LFG formalism. Dedicated interfaces have been developped for editing these lexica, with syntactic and coherence checking.

@chien={'chien canonique'
CAT = N;
↑ Pred = /chien/}
	# chiennes={'chien fem plur' "chiennes forme fléchie "
	↑ Num = plur;
	↑ Genre = fem;
	@chien=/'chien canonique'/}

Figure 5 an entry of a canonical form	Figure 6 an entry of an inflected form

All these lexica conform to the specification defined by an abstract lexicon class. It is possible, and very easy, to add new kinds of lexica, provided they conform to this specification.

Tracking failure causes

A specific feature ("*Error*") allows the system to keep a value that makes explicit the reason why the unifying process has failed. Possible situations are listed below:.

Unifying failure. The values of a given feature are different between the two F-Structures to be unified. The generated F-Structure contains the feature *Error*, whose value is an association of the two uncompatible values. Example*: Num = sing*€*plur*.

A feature present in an equation has non value in either of the two F-Structures to be unified. Example*:* with the equation*"Suj Num = Num"* and two F-Structures without the Num feature, the generated F-Structure contains *"Num = nil€nil"* .

While making a constrained unification (e.g., *Num* =c *sing*) a feature does not exist. We obtain*: Num = sing*€*nil*.

An obligatory feature is absent. *Example: Num = obligatoire*.

A forbidden feature is present. The forbidden state for a feature is represented by adding the value "~"(tilde) to the feature (e.g., *Num* = ~). Therefore, this is the same situation as the simple unification. A failure results from the case when a F-Structure contains this feature. *Example: Num=sing*€~.

A feature has a forbidden value. *Example: Num=~sing.*

When a disjunction of constraints is the reason of the failure, the block itself is set as the value of the "Error" feature in the resulting F-Structure.

These errors can be recovered through the interface (errors are highlighted in the representation), which allows the user to track them easily. Moreover, these well defined categories make it easy to find the real cause of the error and to correct the grammar and the lexicon.

Structure of the rules

Smalltalk80 specific features (mainly the notions of "image" and incremental compilation) have been heavily exploited in the definition of the internal structure of the grammar rules. Basically a rule is defined as the rewriting of a given constituent (left part of the rule), equations being linked to the right constituents. Each non terminal constituent of the grammar is then defined as a Smalltalk class, whose instance methods are the rules whose left part is this constituent (e.g., NP is a class, NP€ProperNoun and NP€ Det Adj* Noun are instance methods of this class).

The Smalltalk compiler has been redefined on these classes so that it handles LFG syntax. Therefore, all the standard tools for editing, searching, replacing (Browsers) may be used in a very natural way. A specific interface may also be used to consult the rules and to define sets rules to be used in the parser.

A great interest of such a configuration is to allow the user to define his own (sub-)set or rules by defining sub-classes of a category when he wants to define different rules for this category (since a method with a given name cannot have two different definitions)

On the use of the Envy/Manager source code manager to maintain the syntactic rules base

Figure 7: Structuring the set of rules

Envy/Manager is a source code manager for team programming in Smalltalk, proposed by OTI. It is based on a client-server architecture in which the source code is stored in a common database accessible by all the developpers.

Envy stores all the successive versions of classes and methods, and provides tools for managing the history. Applications are defined as sets of classes, methods, and extensions of classes, that can be independently edited and versioned. Very fine grained ownership and access rights can be defined on the software components. The structuration of our syntactic rules base enables us to benefit directly of these functionalities, and hence to be able to manage versions, access rights, comparisons of versions (Figure 7)... on all our linguistic data.

Content of the rules

The current grammar contains about 225 rules that covers most of the classical syntactic structures of French simple sentences. They have been tested on data coming from the TSNLP european project. In addition to these simple sentences, difficult problems are also handled: clitics, complex determiners, completives, various forms of questions, extraction and non limited dependancies, coordinations, comparatives. Some extensions are currently under development, including negation, support verbs, circonstant subordinate phrases and ellipses.

Conceptual graphs

Conceptual graphs (Sowa, 1984) form the basis of the semantic and encyclopedic representations used in our system. Conceptual graphs are bipartite graphs composed of concepts and relations. A conceptual graph database is generally composed of the following subparts:

a lattice of concepts and relation types

a set of canonical graphs, associated with concepts and relation types, used for example to express the selectionnal restrictions on the arguments of semantic relations.

a set of definitions, associated with concepts and relation types, used to define the meaning of concepts.

a set of schemas and prototypes.

a set of operations, such as join, contraction, expansion, projection...

a database containing the description of a situation in terms of conceptual graphs.

The framework we describe here aims at managing all this information in a coherent manner, and at facilitating the association with the linguistic processes described above.

Figure 8 Graphical representation of "a cheap horse is scarce" (with second order concepts).

Graphs can be visualized, modified, saved, searched through different interfaces, using graphical (see Figure 8) or textual (see Figure 9, second graph) representations.

Operations can be performed programmatically or using the interface shown in Figure 9.

Figure 9 Conceptual graph operation manager, showing the result of a join between two graphs, and the liste of available operations.

The lattice, and the different items of information associated with concepts and relations types, can be visualized, modified, searched and saved using graphical or textual representations (Figure 10).

Figure 10 Lattice visualizer, showing (bottom right) the canonical graphs of "déconcerter" (to disconcert), the "graph origin" inspector (top right), and the menu of operations (bottom left).

An "individual referents inspector" allows to inspect the cross-references between references, concepts and graphs.

Figure 11 Concepts referencing "Gros Minet" (Sylvester), and one graph referencing one of these concepts.

Analysing a sentence

The processus of analysis from sentence to semantic representation can be separated into three sub-processes. After the sentencehas been segmented, we obtain the lexical items in LFG-compliant form via the lexical manager. After parsing, we obtain some edges with their respective F-Structures. (Delmonte, 1990) has developed a parser which uses basic entries with mixed morphological, functionnal and semantic informations. The rules use different level information. We propose to map the semantic structure on the syntactic one in a manner that avoids too many interdependencies beetween all levels. We use a intermediate structure (named "syntax-semantic table") that expresses the mapping between the value of a LFG Pred and a concept, as well as connected concepts and relations. Semantic data in the lexical knowledge base are defined by using conceptual graphs, as shown in the paragraph 4.1 below about some verb examples. Selectional restrictions defined with canonical graphs are then used to filter the graphs, when more than one is obtained at this level.

Elements of the semantic verb classification in the lexical knowledge base

The lexical knowledge base is based on a hierarchical representation of French verbs. We have developped a systematic and comprehensive representation of verbs in a hierarchical structure, data coming from the French dictionary "Robert". Our method relies on classification method proposed by (Talmy, 1985) and (Miller, 1989, Miller, 1991). We chose a description with a structure composed of a basic action (the first of the most general uperclasses. e.g. *stroll* and *run* can be associated with *walk* as a basic action, and *walk, ride, pass* point at *moving,* which is a step further in generality) associated with thematic roles that specify it (i.e., object, mean, manner, goal, and method). The basic actions are in turn defined with the same structure, based on a more general basic action.

The hierarchy of verbs depends on the thematic relations associated with them. A verb V_1 is the hyperonym (respectively a hyponym) of a verb V_2 (which is noted $V_1>V_2$, respectively $V_1<V_2$) if they share a common basic action and if, in the thematic relations structure associated with it, we have:

- absence (for the hyperonym) or presence (for the hyponym) of a particular thematic relation: e.g. for the pair *divide/cut*; to cut is to divide using a sharp instrument, thus *divide > cut*
- presence of a generic value thematic relation vs. a specific value (example *cut* (object is generic: *solid object*) > *behead* (object is a *head*))

For every verb:

- the semantic description pointed out is coded in the lexical knowledge base as a definitional graph.

```
type cut (*x) is
[divide: *x]—
        (obj) -> [Object: *y] ->(car)->[solid]
        (method) -> [traverse] -> [Object: *y]
        (mean) -> [Instrument] ->(car)->[sharp] .
```

- a canonical graph makes explicit the selectional restrictions

Canonical graph for cut is
(Agent) -> [Animate]
*(Obj) -> Object: *y] ->(car)->[solid] .*

An example

Figure 12 F-Structure for "un avocat vole une pomme"

Below, we give an example for the *sentence "Un avocat vole une pomme*" (a lawyer steals an apple), where "*avocat*" is ambiguous and refers to a lawyer or to an avocado. The non-ambiguous F-Structure of this sentence is show in Figure 12. The entries in the translation table (from LFG pred [in French]to conceptual graphs types [in English]) are as follow:

'avocat'-> (Lawyer Avocado)
'pomme'->(Apple).
'voler(derober)'-> (Steal(Agent->Suj; Objet->Obj)),

Explanations: the first item between quotes is the Pred value, followed by a list of types of concepts (or types of relations) and their mapping definition structure in the F-Structure. represents the local F-Structure. represents the F-Structure that contains the local F-Structure. For example, Agent->Suj means that a concept of Type "Steal" is connected to a concept that can be found in the F-Structure of the feature "Suj" in the local F-Structure. From these data, the following graphs (Figure 13) are obtained.

Figure 13 Graphs

The "Def" feature of the F-structure gives us information about the referents of concepts. For example, the F-Structure for 'apple' contains "Def = indefini", which implies the use of a generic referent for the concept (corresponds to *an* apple, indicated by a star in Figure 13). Then, since canonical graphs express selectional restrictions, they are used to filter the results through the join operation. For example, "Steal" needs an animated agent (Figure 14), therefore graphs with the "Avocado" concept can be removed from the selection.

Figure 14 Canonical Graph for "Steal"

These principles are the bases of the system currently available, but we are working on improvements and extensions. We want to address the issue of adjunct processing, prepositional complements (with problem of second order concepts), etc.

Integration in the lexicon manager

The user can define his association tables as text files following the syntax mentioned above. He can load different tables and order them by priority. As many semantic tables as required can be associated with each of the lexica used. The search in the semantic tables follows the same principle as for the lexicon: the first analysis found is returned. With this structuration, domain specific semantic tables can be independently associated with general purpose syntactic lexicon, thus facilitating reuse.

Figure 15: Lexicon manager

Concluding remarks: Sharing the tools on a network with CORBA

The different tools described in this paper are currently being extended to be CORBA-compatible. CORBA (Common Object Request Broker Architecture) (Ben-Natan, 1995), has been defined by the OMG as an interoperability norm for heterogeneous languages (Smalltalk, C++, JAVA) and platforms (UNIX, Macintosh, PC). CORBA defines a common interface definition language (IDL), as well as a set of services (naming service, security, concurrency management...). CORBA objects can be distributed worldwide (for example using Internet) using an ORB (Object Request Broker).

Various tools implement this CORBA norm. We have used Distributed Smalltalk (Parc Place Digitalk) to realize the distributed implementation of an analyser. With this system, users can currently make an analysis, see the results of this analysis, the F-structures, see the syntactic rules base...

With this kind of architecture, systems necessiting a large amount of ressources can be distributed amongst workstations on a network and/or be used by clients having few ressources. Moreover these ressources can be physically located in any place of a network, allowing thus to distribute the responsibility of their management and maintenance to different persons. With the communication possibilities offered by Internet, it makes it possible to coordinate the cooperative efforts of several teams in the world around a single, coherent, though distributed system.

Figure 16 : Distributed System

We are continuing our work toward the implementation of a complete distributed multi-agent system, following the CARAMEL architecture (Sabah, 1993, Sabah, 1995, Sabah, 1997).

Figure 17: Partial overview of the environment

References

Ben-Natan, R. (1995). CORBA, a Guide to the Common Object Request Brocker. McGraw-Hill.

Booch G. 1994, *Analyse & conception orientées objets*, Addison-Wesley,

Bresnan Joan and Ronald Kaplan 1981, Lexical functional grammars ; a formal system for grammatical representation, *The mental representation of grammatical relations*, MIT Press, Cambridge, Mass.

Delmonte R. 1990, Semantic Parsing with LFG and Conceptual Representations, *Computers and the Humanities, Kluwer Academic Publishers, 24*, p. 461-488.

Kaplan R.M. and J.T. Maxwell 1994, Grammar Writer's Workbench, Xerox Corporation, Version 2.0.

Kay Martin 1967, Experiments with a powerful parser, Proceedings 2nd COLING, , p. #10.

Kay Martin 1979, Functional grammars, Proceedings 5th. annual meeting of the Berkeley linguistic society, Berkeley, p. 142-158.

Miller A. G., C. Fellbaum and D. Gross 1989, WORDNET a Lexical Database Organised on Psycholinguistic Principles, Proceedings IJCAI, First International Lexical Acquisition Workshop, Detroit.

Miller G. A. and C. Fellbaum 1991, Semantic networks of English, *Cognition, 41*, p. 197-229.

Pitrat Jacques 1983, Réalisation d'un analyseur-générateur lexicographique général, rapport de recherche GR22, Institut de programmation, Paris VI, 79/2.

Sabah Gérard 1995, Natural Language Understanding and Consciousness, Proceedings AISB — workshop on "Reaching for Mind", Sheffield.

Sabah Gérard 1997, The fundamental role of pragmatics in Natural Language Understanding and its implications for modular, cognitively motivated architectures, *Studies in Computational Pragmatics: Abduction, Belief, and Context*, University College Press, *à paraître,* to appear, London.

Sabah Gérard and Xavier Briffault 1993, Caramel : a Step towards Reflexion in Natural Language Understanding systems, Proceedings IEEE International Conference on Tools with Artificial Intelligence, Boston, p. 258-265.

Sowa John 1984, *Conceptual structures : information processing in mind and machine*, Addison Wesley, Reading Mass.

Talmy L. 1985, Lexicalisation patterns: Semantic structure in lexical forms, *Language typology and syntactic description, 3*, Cambridge University Press, New York, p. 57-149.

Winograd Terry 1983, *Language as a cognitive process , Volume I syntax*, Addison Wesley, Reading Mass.