AN EFFICIENT CONTEXT-FREE PARSER FOR AUGMENTED PHRASE-STRUCTURE GRAMMARS

Massimo Marino*, Antonella Spiezio, Giacomo Ferrari*, Irina Prodanof+
*Linguistics Department, University of Pisa,
Via S. Maria 36, I-56100 Pisa - Italy
+ Computational Linguistics Institute - Cnr
Via Della Faggiola 32, I-56100 Pisa - Italy

ABSTRACT

In this paper we present an efficient context-free (CF) bottom-up, non deterministic parser. It is an extension of the ICA (Immediate Constituent Analysis) parser proposed by Grishman (1976), and its major improvements are described.

It has been designed to run Augmented Phrase-Structure Grammars (APSG) and performs semantic interpretation in parallel with syntactic analysis.

It has been implemented in Franz Lisp and runs on VAX 11/780 and, recently, also on a SUN workstation, as the main component of a transportable Natural Language Interface (SAIL = Sistema per l'Analisi e l'Interpretazione del Linguaggio). Subsets of grammars of italian written in different formalisms and for different applications have been experimented with SAIL. In particular, a toy application has been developed in which SAIL has been used as interface to build a knowledge base in MRS (Genesereth et al. 1980, Genesereth 1981) about ski paths in a ski environment, and to ask for advice about the best touristic path under specific weather and physical conditions.

1. INTRODUCTION

Many parsers for natural language have been developed in the past, which run different types of grammars. Among them, the most successful are the CF grammars, the augmented phrase-structure grammars (APSGs), and the semantic grammars. All of them have different characteristics and different advantages. In particular APSGs offer a natural tool for the treatment of certain natural language phenomena, such as subject-verb agreement. Semantic grammars are prone to a compositional algorithm for semantic interpretation.

The aim of our work is to implement a parser which associates the full extension of an APSG to compositionality of semantics. The parser relies on the well stabilized ICA algorithm. This association allows a wide range of applications in syntactic/semantic analyses together with the efficiency of a CF parser.

2. Functional description of the parsing algorithm

The parsing algorithm consists of the following modules:
- a preprocessor;
- a parser itself;
- a post-processor and interpreter;

and interacts with:
- a dictionary, which is used by the preprocessor;
- the grammar, used by the parser.

Figure 1 shows the structure of the system we have designed. Some of the modules, such as the spelling corrector, the robusteness component, and the NL answer generator, are still being developed.

2.1. The dictionary

The dictionary contains the 'word-forms', known to the interface, with the following associated information, called 'interpretation':
- syntactic category;
- semantic value;
- syntactic features as gender, number, etc.;

A form can be single (a single word) or multiple (more than one word). Multiple forms are frequent in natural language and are in general referred to as 'idioms'. However, in semantic grammars, the use of multiple words is wider than in syntactic ones as also some simpler phrases may be more conveniently treated in the dictionary. This is the reason why multiple forms are treated by specific algorithms which optimize storage and search.

The description of this algorithm is not the aim of this paper.

Figure 2 shows an example of such a dictionary, which contains the single forms **che (that** as conjunction), **e' (is)**, **noto (well-known)** and the multiple forms **e' noto (it's well-known)** and **e' noto che (it's well-known that)**. The mark **EOW** indicates a final state in the interpretation of the form currently being scanned.

2.2. The grammar

The grammar is a set of complex grammatical statements (CGS), represented in BNF as follows:

CGS::=<RULE> <EXPRESSION>
<RULE>::=<PRODUCTION> <TESTS> <ACTIONS>
<PRODUCTION>::=<LEFT-SYMBOL>
<RIGHT-PATTERN>
<LEFT-SYMBOL>::= a non terminal symbol
<RIGHT-PATTERN>::= a sequence of categories
<TESTS>::= a whatever predicate
<ACTIONS>::= a whatever action
<EXPRESSION>::= a semantic interpretation in any chosen formalism

As we have already stated, the <PRODUCTION>'s can be instantiated both with syntactic and with semantic grammars. The schema of the rule and the order of the operations are fixed, regardless of the chosen instance grammar.

<TESTS> are evaluated before the application of a rule and can inhibit it if they fail. <ACTIONS> are activated after the application of a rule and perform additional structuring and structure moving. Both participate into a process of syntactic recognition and are to be considered as the syntactic augmentation of the rules. When using a semantic grammar the <ACTIONS> are, in general, not used.

<EXPRESSION>'s are the semantic augmentation and specify the interpretation of the sentence, for top level rules, or (partial) constituents, for the other rules. These two augmentations improve the syntactic power of the grammar, by adding context sensitiveness, and add a semantic relevance to the structuring of constituents, due to the one-to-one correspondence between syntactic and semantic rules.

The set of rules of a grammar is partitioned into packets of rules sharing the same rightmost symbol of the <RIGHT-PATTERN> of productions. This partitioning makes their application a semi-deterministic process, as only a restricted set of them is tried, and no other choice is given.

2.3. The preprocessor

The preprocessor scans the sentence from left to right, performs the dictionary look-up for each word in the input string, and returns a structure with the syntactic and semantic information taken from the dictionary. At the end of the scanning the input string has been transformed into a sequence of such lexical interpretations. The look-up takes into account also the possibility that a word in input is part of a multiple form.

2.4. The parser

The parser is an extension of the ICA algorithm (Grishman 1976). It shares with ICA the following characteristics:

- it performs the syntactic recognition bottom-up, left-to-right, first selecting reduction sets by an integrated breadth and depth-first strategy. It does not reject sentences on a syntactic basis, but it only rejects rule by rule for a given input word. If all the rules have been rejected with no success, the next word in the preprocessed string is read and the loop continues.

  Termination occurs in a natural way, when no more rule can be applied and the input string has come to an end;

- it gives as output a graph of all possible parse trees; the complete parse tree(s) is (are) extracted from the graph in a following step. This characterizes the algorithm as an all-path-algorithm which returns all possible derivations for a sentence. Therefore, the parser is able to create structure pieces also for ill-formed sentences, thus outputting, even in this case, partial analyses. This is particularly useful for diagnosis and debugging.

The following are the major extensions to the basic ICA algotrithm:

- it is designed to run an APSG, in particular it evaluates the tests before applying a rule;

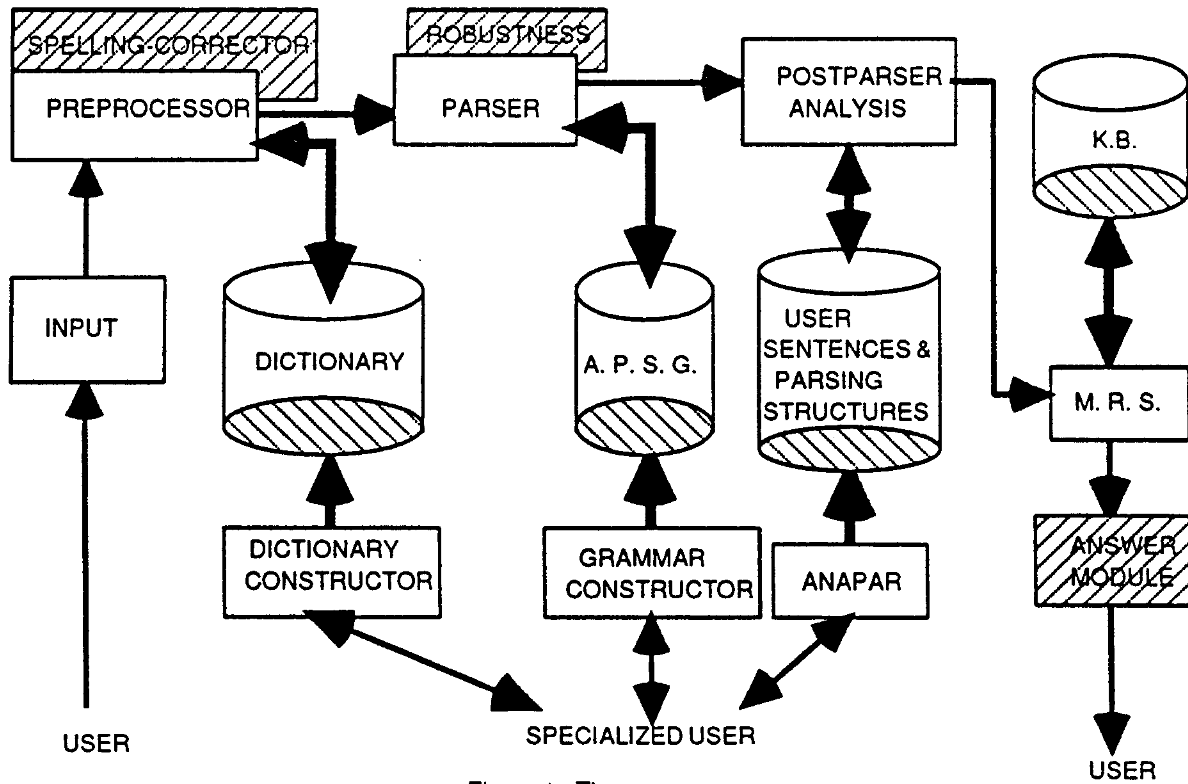


Figure 1. The system.

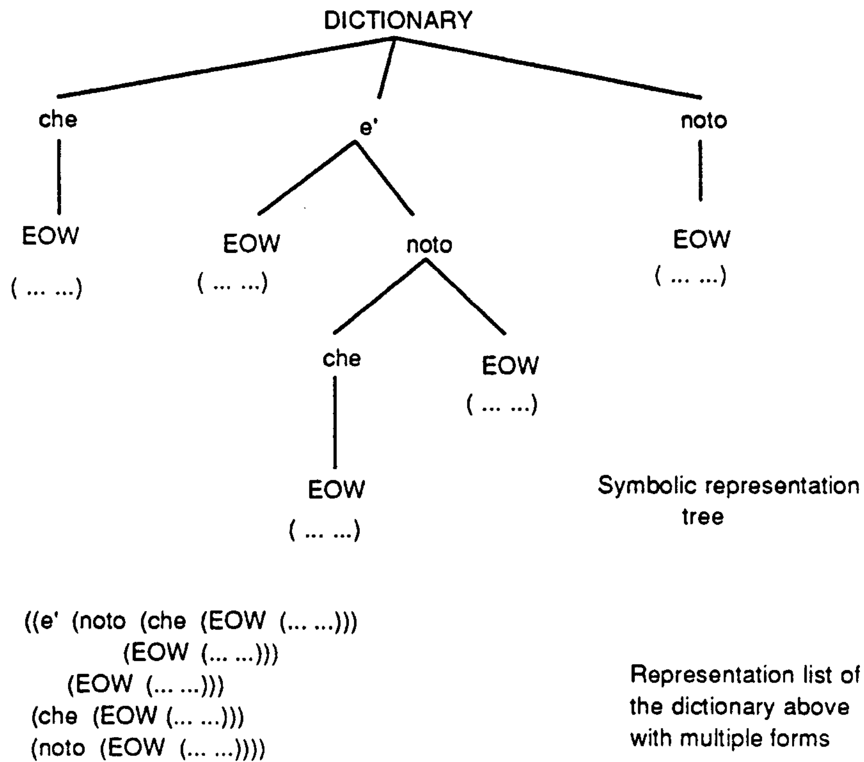


Figure 2. The dictionary representation.

- it handles lexical ambiguities during parsing by representing them in special multiple nodes (see below);
- the partition of the rules into packets makes the selection of the rules semi-deterministic;
- it carries syntactic and semantic analysis in parallel.

2.5. Post-processor and Interpreter

The graph built by the parser is the data structure out of which the parse tree is extracted by the post-processor. To this end the necessary conditions are that:

a. there exists at least one top level node among the nodes of the graph;

b. at least one of the top level nodes cover the whole sentence.

If one of these conditions is not met, i.e. if there is no top level node or no top level node covers the entire sentence, the analyser does not carry any interpretation but displays a message to the user, indicating the more complete partial parsing, where the parser stopped.

In case of ambiguity more than one top level node covers the entire sentence and more than one semantic interpretation is proposed to the user who will select the appropriate one. If, instead, only one top level node is found, the semantic interpretation is immediately produced.

3. Data structure and algorithm

3.1. Data structure

The algorithm takes in input a preprocessed string and returns a graph of all possible parse trees. The nodes in the graph can be either terminals (forms), or non terminals (constituents). Nodes are identified as follows:

- the 'name' can be either FORMi or CONSTITUENTj, according to the type. i and j are indexes, and forms and constituents have two independent orderings;
- a general sequence number.

The following two types of structural information are associated with each node:

a. the 'annotation' specifies the associated 'interpretation', i.e.:

- the syntactic category of the node (the label);
- its semantic value;
- its features.

For terminal nodes, their interpretation, i.e. their annotation coincides with the interpretation associated to the form by the preprocessor. For non terminal nodes, instead, the interpretation is made during the building of the node and the applied rule gives all necessary information;

b. the 'covering structure' of a node contains the information necessary to identify in the graph the subtree rooted in that node. Each node in the graph dominates a subtree and covers a part of the input, i.e. a sequence of terminal nodes. In this sequence, the form associated with the leftmost terminal node is a 'first form'. The form immediately to the right of the form associated to rightmost terminal node is the 'anchor'. For terminal nodes the covering structure contains:

- the first form (the node itself);
- the anchor (the next form in the input string);
- the list of parent nodes;
- the list of anchored nodes, i.e. the nodes which have as anchor the form itself;

while for non terminal nodes it consists of:

- the first form;
- the anchor;
- the list of parents;
- the list of sons.

Two trees T1 and T2 are called adjacent if the anchor of T1 is the first form of T2.

3.2. The algorithm

The parser is a loop realized as a recursion. It scans the preprocessed string and creates a terminal node for every scanned form. As a terminal node is created, the algorithm attempts to perform all the reductions which are possible at that point. A 'reduction set' is defined as the set of nodes N1,N2,...,Nn which are roots of adjacent subtrees and correspond, in the same order, to the <RIGHT-PATTERN> of the examined production. If no (more) reduction is possible, the parser scans the next form. The loop continues until the string is exhausted.

The parser operates on the graph and has in input two more data structures, i.e.:

- the stack of the active nodes, which contains all the nodes which are to be examined; this is

accessed with a LIFO policy;
- the list of rule packets, which contains the rules potentially applicable on the current node.

The loop starts from the first active node. Its annotation is extracted and the corresponding rule packet is selected, i.e. the one whose rightmost symbol corresponds to the current node category. The reduction sets are thus selected. A reduction set is searched by an integrated breadth and depth-first strategy as alternatives are retrieved and stored all together as for breadth-first search, but are then expanded one by one.
The choice of the possible applicable rules is not a blind one and the rules are not all tested, but they are pre-selected by their partition into packets. More than one set is possible at each step, i.e. the same rule can be applied more than once. During the matching step reduction sets are searched in parallel; reductions and the building of new nodes are also carried in parallel.
Once a reduction set is identified, the tests associated with the current rule are evaluated. If they succeed, the corresponding rule is applied and a new node which has as category the <LEFT-SYMBOL> of the production is created and inserted in the active node stack. This becomes the root of the (sub)tree whose sons are in the reduction set. The evaluation of tests prior to entering a rule is a further improvement in efficiency.
The annotation of the new nodes is now created by the execution of the actions, which insert new features for the node, and the evaluation of the expression which assigns to it a semantic value.
If the tests fail, the next reduction set is processed in the same way. If there is no (more) reduction set, the next rule in the packet is examined until no more rule is left.
When the higher level loop is resumed the next active node is examined. Termination occurs when the input is consumed and no more rule can be applied.

3.3. Lexical ambiguity

The algorithm can efficiently handle lexical ambiguity.
For those forms which have more than one interpretation, a special annotation is provided. It contains a certain number of interpretations and each interpretation has the following form:

```
(#i ((<cat> <sem_val>)
    ((<feat_name> <feat_val>)*)))
```

where #i is the ordering number of the interpretation. This structure is called 'multiple node'. Figure 3 shows multiple nodes participating to different structures.

4. An example

The most relevant application of SAIL is its use as a NL interface towards a knowledge base about ski environments. Natural language declarations about lifts, snow and weather conditions, and classification of slopes are translated into MRS facts, and correspondently NL questions, including advice requests, are processed and inserted.

Let's take the question:

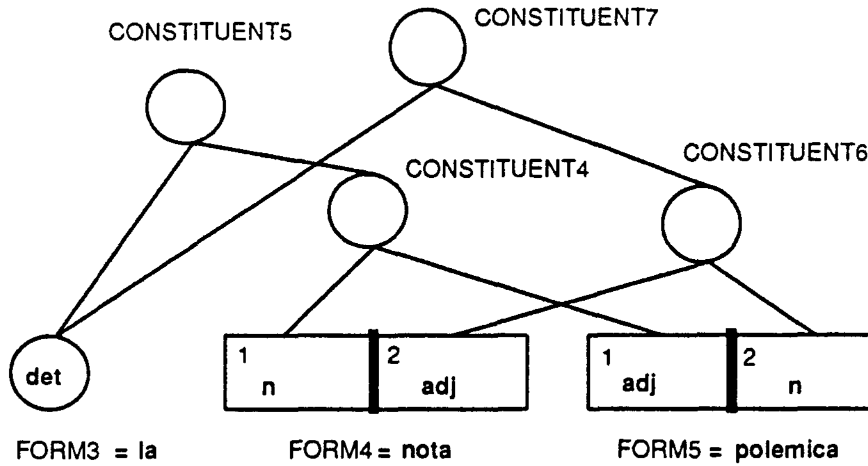
'Come si sale da Cervinia al Plateau Rosa ?'
'How can one get on the Plateau Rosa from Cervinia ?'

and the grammar:

```
Rule1:
PROD: TG -> come <connette> <partenza>
          <arrivo> ?
TESTS: t
ACTIONS: t
EXPRESSION:(trueps
           '(connette (SEMVAL '<partenza>)
              (SEMVAL '<arrivo>)
              $mezzo))

Rule2:
PROD: <partenza> -> da <luogo>
TESTS: t
ACTIONS: t
EXPRESSION: (SEMVAL '<luogo>)

Rule3:
PROD: <arrivo> -> al <luogo>
TESTS: t
ACTIONS: t
EXPRESSION: (SEMVAL '<luogo>)
```



CONSTITUENT5 recognizes **'la nota polemica'** **'the polemic note'**

CONSTITUENT7 recognizes **'la nota polemica'** **'the well-known controversy'**

Figure 3. Multiple nodes.

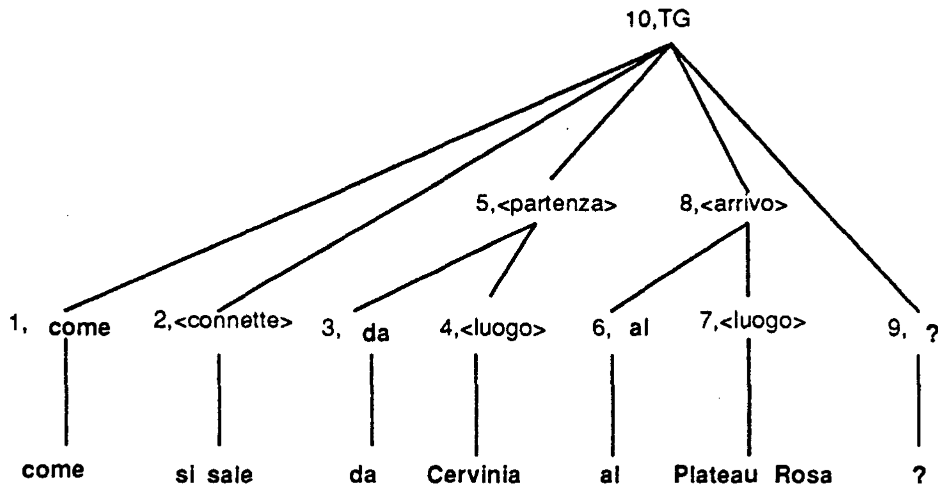


Figure 4. The parse-tree of the example.

DICTIONARY-FORM#1:<connette> -> **si sale**
DICTIONARY-FORM#2:<connette> -> **si giunge**
DICTIONARY-FORM#3:<luogo> -> **Cervinia**
DICTIONARY-FORM#4:<luogo> -> **Plateau Rosa**

SEMVAL is a function that gets the semantic value from the node having the category specified by its parameter; this category must appear in the right-hand side of the production. trueps is an MRS function that checks the knowledge base for the presence or not of a predicate.
The parser starts by creating the terminal nodes:

node1: $form_0$: **come**
node2: $form_1$: **si sale**
node3: $form_2$: **da**
node4: $form_3$: **Cervinia**

and the rule2 can be applied on nodes node3 and node4. The following node is created:

node5: $constituent_0$: **da Cervinia**

In an analogous way other nodes are added.

node6: $form_4$: **al**
node7: $form_5$: **Plateau Rosa**
node8: $constituent_3$: **al Plateau Rosa**
node9: $form_6$: **?**
node10: $constituent_4$: **come si sale da Cervinia al Plateau Rosa ?**

As the syntactic category of node10 is TG (Top Grammar) and it covers the entire input, the parsing is successful. Figure 4 shows the parse-tree for this sentence.

5.Conclusions and future developments

At present the parser described above has been efficiently employed as a component of a natural language front-end. The natural language is Italian and typical input sentences either give information about the possible trips (paths/alternative paths) and their characteristics (type of lift, condition of snow, weather), or have the following form:

'Qual'e' il percorso migliore per andare da X a Y per uno sciatore provetto ?'
'What is the best path from X to Y for an excellent skier ?'

Three different improvements are in progress:

- the implementation of a spelling corrector and of a dictionary update system.The parser rejects such sentences where some forms occur that are not in the dictionary. A form not included in the dictionary cannot be distinguished from a form incorrectly typed but present in the dictionary. The two cases correspond to different situations and need distinct solutions. In the former case the defective form may be inserted in the dictionary by means of an appropriate update procedure. In the latter case the typing error may be corrected on the basis of a classification of errors compiled according to some user's model;
- another perspective is making the parser more powerful also about more strictly linguistic phenomena as the resolution of ellipsis and anaphora;
- finally, the identification of general semantic functions to be employed in the <EXPRESSION> part of the rule has been started.

REFERENCES

Genesereth, M. R., Greiner, R. & Smith, D. E. (1980). **MRS Manual.** Technical Report HPP-80-24, Stanford University, Stanford CA.

Genesereth, M. R. (1981). **The architecture of a multiple representation system.** Technical Report HPP-81-6, Stanford University, Stanford CA.

Grishman, R. (1976). A survey of syntactic analysis procedures for natural language. **AJCL,** Microfiches 47, 2-96.

Marino, M., Spiezio, A., Ferrari, G. & Prodanof, I. (1986). SAIL: a natural language interface for the building of and interacting with knowledge bases. In **Proceedings of AIMSA 86** (on microfiches), Varna, Bulgaria.

Winograd, T. (1983). **Language as a Cognitive Process. Vol.1: Syntax.** Addison-Wesley.