Phrase-Driven Parser

ROUX Claude
Multilingual Theory and Technology (MLTT)
Xerox Research Centre Europe
6 chemin de Maupertuis, 38240 Meylan, France
Claude.ROUX@xrce.xerox.com
Office: (33) 04-76-61-51-38

Abstract

This paper describes an implementation of a General Phrase Structure Grammar parser based on a binary coding of categories and features. We show that using this specific coding, this implementation allows a rule extraction based on phrase description rather than on a single category, which improve significantly the performance of such a parser.

Introduction

The World Wide Web has triggered the need for robust and fast natural language tools to automatically analyze vast amount of texts. However, current parsers are based on technologies that require hefty configurations ([Van Noord 97]) to run. These constraits often lead linguists to restrict the power of the grammars. Even if the capacity of computers were to increase every year as in the past, current implementations of NLP systems will n ot achieve the goal of being useful parts of other applications and will remain artificial applications in and of themselves.

The goal of this paper is to present a new type of algorithm to implement a General Phrase Structure Grammar parser (GPSG) [Gazdar et al. 1985]. This algorithm is based on a binary coding of syntactic categories, which offers the following advantages:

- Global analysis of a sequence of categories, instead of relying on a chart as in [Shieber 84,86] implementation of the Earley's algorithm.
- Automatic extraction of properties from the grammar like the sisterhood relation, as in [Evans 87].

We also propose an improved solution for indexing rules, which is not based on a specific category as in [Blache 90] or in [Van Noord 97].

1. Background on GPSG

A grammar in GPSG is composed of two sets of rules (Gazdar et al. 1985): Immediate Dominance Rules (ID-rules) and Linear Precedence Rules (LP-rules).
The ID-rules describe a local syntactic tree but do not define any order on the sister categories under the head node.
The LP-rules control the order of these categories. Both of rules are explained below.

1.1. Immediate Dominance Rules (ID-Rules)

In order to treat free order languages, GSPS describes local trees with rules similar to context-free rules (CF-rule) but in which the right side is not ordered. Thus, the right-hand of an ID-rule does not correspond to a sequence of categories but to a *set* of categories.

Example 1

The rule: " NP → Det,N,A. " is equivalent to these six CF-rules:

1) NP → Det A N.
2) NP → Det N A.
3) NP → A Det N.
4) NP → A N Det.
5) NP → N Det A.
6) NP → N A Det.

1.2. Linear Precedence Rules (LP-rules)

The acceptable order of categories is controlled in GPSG by a second set of rules: the LP-rules. These rules impose a specific order on sister categories in a local tree. They do not define an order between specific categories, but rather between categories bearing certain features.

Example 2

This LP-rule regulates the position of the adjective in English (an adjective is a category with positive noun and positive verb features):

[Noun:+,Verb:+] <[Noun:+,Verb:--]

1.3. Other Types of Implementation of GPSG

Some of the previous implementations of GPSG are derived from the modification of the Earley algorithm by [Shieber 84,86]. Shieber proposed a Chart Parser strategy, where LP-rules were used to invalidate certain sequence of categories present in the chart. In Shieber's implementation, each ID-rule is composed of two sets of categories. The first set contains the list of categories that are expected, this set is initialized with all the categories from the right-hand of the ID-rule. The second set contains the categories that have been detected at a given moment during parsing. The parser moves a category from the first set to the second set when this category is found in the sentence. The chart contains the active rules with a set of detected categories (the second set) that is not empty. Each time a category is moved from the first set to the other, LP-rules are checked against the second set of categories to verify that no illegal sequence of categories has been encountered.

1.4. A new implementation

We propose a new implementation of GSPS that is not based on a chart but on a different handling of rules and categories. The principal feature of this method is the coding of syntactic categories, which will allow us to avoid the memory cost of a chart parser [Van Noord 97].

1.5. Category Coding

In our system, each category is replaced by a position in a vector of bits. This binary coding is a simple and efficient way to encode information in computers. Simple because all complex operations in a processor are based on a manipulation of bits, efficient because these binary operators are hard-coded in every processor. One of the most interesting properties of this coding for us, is the possibility of mapping the set operators (union, intersection) on the binary operators (OR,AND). Intersection can be mapped as a binary AND on bits, and the union as a binary OR, yielding a fast and efficient manipulation of sets as vectors of bits. Since ID-rules are composed of a set of categories on the right-hand of the rule, we describe each ID-rule as a set of bits, where each bit is a category. In a computer, a vector of bits is usually manipulated as an integer; for this reason we represent each ID-rule as an integer. See section 2.6 for restrictions that this representation implies.

Example 3

Each category is encoded as a bit. Below is a sample for a few categories:

Table of categories

Category	**S**	**NP**	**VP**
Decimal	1	2	4
Binary	0000000001	0000000010	0000000100
Category	**AP**	**PP**	**V**
Decimal	8	16	32
Binary	0000001000	0000010000	0000100000
Category	**N**	**A**	**Prep**
Decimal	64	128	256
Binary	0001000000	0010000000	0100000000
Category	**Det**		
Decimal	512		
Binary	1000000000		

Given the following rules:

1) NP → N,Det,A
2) AP → A
3) VP → V, NP,PP
4) PP → Prep, NP
5) S → NP,NV.

These rules are recorded as shown in the table below:

Table of ID-rules

ID-rules	CODING: decimal	CODING: binary
1) NP → N,A,Det	704	1011000000
2) AP → A	128	0010000000
3) VP → V,NP,PP	50	0000110010
4) PP → Prep,NP	258	0100000010
5) S → VP,NP	6	0000000110

1.6. Limitation

This system of coding presents some limitations. First, the number of categories is limited to the size of the integer representation in the system chosen for the implementation. Today, most systems offer 64 bit integers, which impose a maximum of 64 categories. If we refer to [GSPG

1985], this value is large enough to represent the 20 categories which are used in the theory.
Second, if a category occurs more than once in the right side of an ID-rule, these extra categories will not be apparent in the coding, since a bit indicates the presence of a specific category but cannot describes the number of time these categories occur. For this reason, different ID-rules can share the same coding. This can have an impact on the global efficiency of the system.
Third, this coding is only used to retrieve ID-rules.
In our system, each category is also described as a set of features. For example, LP-rules are described with these features.
These features are implemented as binary vectors in the same manner as the coding of ID-rules.
Each terminal feature value is coded as a single bit in a **list of binary vectors**, attributes are coded over these bits.
Since these values are coded over a \emph{list of binary vectors}, rather than on a single integer, the number of terminal values handled by the system is not limited to 64 terminal values.

1.7. Storing and Extracting ID-rules

In [Blache 90] and [van Noord 97], rules are indexed on specific categories. Blache proposes indexing rules on categories that occur as the first element of an ID-rule. Van Noord proposes indexing rules on the head category, rules are then retrieved according to this head.

The coding (the rule index value hereafter) of each ID-rule into a unique number based on the right-hand side of an ID-rule allows some interesting possibilities in rules management. For example, each rule can be stored in a hash table or in a balanced tree along its rule index value. It follows from this sort of storage, that an ID-rule will be identified by a digital value, which is in the same time its index and its content.

The extraction process then consists of extracting, from a list of categories, a sequence that will be encoded as an integer (the sequence index value hereafter) in the manner of an ID-rule. In a second step, this value is used to extract an ID-rule whose rule index value equals this value. Thus, the extraction of an ID-rule and the validation of a sequence of categories correspond to a unique process: a search based on a numerical index.

Example 4

Given the following sentence: *The big door*

We can translate this sentence as a list of categories: *Det(the) A(big) N(door)*

We now analyze the following sequence:
Det(the) A(big) N(door).

Each element is replaced by its binary position:
Det(512)+A(128)+N(64)= **704**

In example 3, **704** corresponds to the first ID-rule: *NP → N,A,Det.*

2. Filtering the Sequence

The utilization of this coding accelerates the extraction and the application of an ID-rule, but it does not improve the parsing of a sentence.

The simplest parser one can build using the above representation, collects sequences of categories and, for each sequence, calculates a sequence index value. Then on each of these sequences we apply LP-rules to eliminate the sequences that are composed of illegal lists of categories. Finally, we search for an ID-rule whose rule index value equals the validated sequence index value. If such a rule exists, the sequence is replaced by the mother category of this rule. The same process is then applied on the new list of categories, until this list is reduced to one element.

The first parser we developed with this method was slow and would sometimes run out of memory on large sentences. This parser presented some flaws that affected others GPSG implementations. Some of these problems were corrected through the implementation of methods developed in [Evans 87] and [Blache 90], and described below.

2.1. Sisterhood rules

[Evans 87] improved Shieber's version of Earley's algorithm with the introduction of sisterhood rules. These rules, written by the linguist, impose which categories can occur together in a valid sequence of categories. The goal of these rules is to reject the construction of illegal sequences {VERB,DET,NOUN} before the system has to find out that no rules would match these sequences. Evans showed that these rules improve the efficiency of a parser. The only drawback of this system is that it implies the maintenance of a

third set of rules. We, however, automatically extracts these rules from the grammar as follows.

Extraction of Sister Vectors

Each category is associated with a binary vector (the sister vector) containing all the categories that can occur with this category in the right side of an ID-rule.

Extraction rule

The calculation of this *sister vector* consists of scanning each ID-rule where a category C occurs. When a category *C* occurs in an ID-rule *R*, the *rule index value* of *R* is appended to the *sister vector* of *C*, using the binary operator *OR*.

Running the Parser Using Sister Vectors.

At each step, when collecting categories from the right to the left, we calculate the sequence index value that is based on the binary representation of each category. The next category will be added to this sequence if the intersection of its sister vector with the sequence index value equals the sequence index value. In other words, each of the categories extracted so far can occur with the current category in an ID-rule.

Example 5

First, we compute the *sister vector* of each category, based on the grammar presented above:

The sisters table

Category	**S**	**NP**
Decimal	0	310
Binary	0000000000	0100110110
Category	**V**	**N**
Decimal	50	704
Binary	0000110010	1011000000
Category	**VP**	**AP**
Decimal	6	0
Binary	0000000110	0000000000
Category	**A**	**Prep**
Decimal	704	258
Binary	1011000000	0100000010
Category	**PP**	**Det**
Decimal	50	704
Binary	0000110010	1011000000

Now, we analyze the previous example using the sister table method:

The big lady opens the door with a small key

We can extract from this sentence the following sequence of categories: *Verb(opens) Det(the) N(door)*

We now extract a sequence of categories starting from the right of the sentence:

1) *N(64)*, the sequence index value for this sequence is: **64**

2) *Det(512),N(64)*, the sequence index value for this sequence is 512 OR 64= **576**.
Intersection of this value with the sister value of *Det*= 704 AND 64 = 64 (*Det is valid*)

3) V(32),Det(512),N(64), the sequence index value for this sequence is 32 OR 576= **608**.
Intersection of this value with the sister value of *V*= 576 AND 50 = 0 (*V is not valid*)

V cannot be added to this sequence.
We have isolated the valid sequence Seq: Det,N = **576**.

2.2. Left Sister rules

The previous solution presents a drawback, which is illustrated with the position of the adjective in English.
In GPSG the position of the adjective is described with the following LP-rule:

[Noun:+,Verb:+] < [Noun:+,Verb:-]

This LP-rule states that an adjective in English always precedes the noun.

In case of a list of categories in which an adjective would follow a noun, the previous procedure would add this adjective to the current sequence. Eventually, the sequence would be invalidated by the LP-rule.
The improvement we implemented, to solve this problem, is to build *sisters vectors* in which only categories that can occur on the *left of the current category* are stored.

Building the left sister vectors

At each step, while building the *left sister vector* of a category C, for each category C_i that occurs with C in an ID-rule, we check if an LP-rule exists that would invalidate "C_i < C". If no LP-rules exist, then C_i is appended to the *left sister vector* of C. This verification is done at compilation time, once for all.

Extraction rule

The calculation of this sister vector consists of scanning each ID-rule where a category C occurs. When a category C occurs in an ID-rule R, the rule index value of R is appended to the sister vector of C, using the binary operator OR.

The result is a more specific *sister vector* that helps to restrict the concatenation of categories to the best candidates, reducing the number of comparisons to be made during parsing.

3. A Brief Comparison with Chart Parsers

The main difference with our implementation and other chart parser [Shieber 84,86], [Evans 87] resides in the absence of rules dot management. In a chart parser, for each category extracted from the sentence, a dot is moved after this category is consumed in each rule that contains this category. Once this dot has reached the last category, the rule is activated and the sequence is then replaced by the mother category of the rule. This system implies a heavy management of rules; one has to record each active rule with the current position of its dot.

In our system, only one cursor is handled. This cursor is moved after a sequence of categories has been identified in the sentence. If an ID-rule applies to a sequence of categories, the mother category of this ID-rule replaces this sequence and the cursor is moved back to the end of the sentence. Otherwise, the cursor is moved to the next category on the left. Since rules are extracted globally, there is no need for recording which rules are susceptible to be activated. Furthermore, as most linguistic information is coded into vectors of bits, the parser has a very compact memory footprint. For example, a grammar of 650 ID-rules, 40 LP-rules, needs 650Ko of memory. To parse a sentence of 20 words, the system requires about 5Ko. We also conducted an experiment in which we recorded previous extracted sequences to reduce the number of analyses. The result corroborated the experiments of [Van Noord 97]: if the number of sequences actually analyzed decreases, the comparison process absorbs most of the gain. An additional drawback to this extension is that the parser requires three times more memory than the basic version.

4. Some results

The grammar that has been used for our test is composed of 600 ID-rules and 22 LP-rules. This grammar has been designed to analyze a wide variety of sentences in French. This parser is fed with sentences part-of-speach tagged with Xerox tools. The parser currently runs on a 250 MHz Sun Ultra-Enterprise, Solaris 2.5. The time needed to part-of-speach tag the sentences is included in the execution time.

The system is robust in the sense that every sentence is analyzed.

corpus size	185 sentences
average size for a sentence	27 words/sentence
execution time	7 sec
speed	26 sentences/sec 728 words/sec.
Memory Footprint	656Ko (grammar and parsing space)

Example 6

A l'interprétation des sentiments présidentiels s'ajoute l'atmosphère de surenchère politique qui précède tout congrès du Parti socialiste.

To the interpretation of the presidential feelings is added an atmosphere of political escalation that precedes every meeting of the Socialist Party.

5. Conclusion

The corpus that was used to test our parser is extracted from the French newspaper: **Le Monde**. Articles in this newspaper are usually written in a rather high level French language. The numbers above show that this parser is fast and does not require a huge amount of memory.

A typical sentence from this corpus is shown is example 6. The goal of this project is to dispose of a family of compact NLP tools, which will not exhaust the resources available for a larger application that requires natural language

processing. NLP tools should be a part of an application, not the application itself.

References:

Aho, A.V. Ullmann, J.D., (1972), *The Theory of Parsing, Translation and Compiling*, v.1, Englewood Cliffs, N.J., Prentice Hall, Inc.

Barton, G., Berwick, R., Ristad E., (1987), *Computational Complexity and Natural Language*, MIT Press.

Blache, Ph., (1990), *L'analyse syntaxique dans le cadre des grammaires syntagmatiques généralisées: Interprétation et stratégies,* Thèse de doctorat, Université d'Aix-Marseille II.

Earley, J., (1970), *An Efficient Context-Free Parsing Algorithm*, Communication of the ACM, v. 13, p94-102.

Evans, R, (1987), *Theoritical and Computational Interpretations of Generalized Phrase Structure Grammar,* PhD, School of Cognitive Sciences, The University of Sussex, Brighton.

Gazdar G., Klein E., Pullum G., Sag A. I., (1985), *Generalized Phrase Structure Grammar*, Blackwell, Cambridge Mass., Harvard University Press.

Gertjan van Noord., (1997), *An efficient implementation of the head-corner parser*. Computational Linguistics, 23(3): 425-456.

Nederhof, Mark-Jan and Giorgio Satta, (1994), *An extended theory of head-driven parsing*, in Proceedings of the 34th Annual Meeting, Santa Cruz, CA. Association for Computational Linguistics.

Roux C., (1996)**,** *Une méthode de parsage efficace pour les Grammaires Syntagmatiques Généralisées*, Ph.D, Université de Montréal.

Shieber S., (1984), *Direct parsing of ID/LP grammars,* Linguistics and Philosophy, v. 7(2), p135-154.

Shieber, S., (1986), *An introduction to unification-based approaches to grammar*, Stanford, CSLI, Stanford University, no. 4.

Tomita, M., (1986), *Efficient parsing for natural language : a fast algorithm for practical systems*, Kluwer Academic Publishers.