Efficient Sharing in Ambiguous Parsing

Manuel Vilares Ferro

Facultad de Informática
Campus de Elviña, s/n
15071 La Coruña, Spain

Abstract

At the outbreak, interest in non-deterministic parsing was due to the fact that it is the context-free backbone of some natural language analyzers. In effect, several of them start with a purely context-free parsing phase which generates all possible parses as some kind of sharing structure usually called shared forest. Later, in a second phase, a more elaborate analyzer that takes into account the finer grammatical structure of the language is applied in order to filter out undesirable parses. That implies the existence of a disambiguation process, from which a single tree is to be chosen in the resulting shared forest. In this paper[1], we study the structure of these forests with respect to optimality of syntactic sharing, in relation to the parsing schema used to produce them and to the number of computations to be applied during the parse process. Finally, we give the guideline to design efficient context-free parsers in relation to both the performance and the implementation techniques.

Key Words and Phrases: Context-Free Parsing, Dynamic Programming, Dynamic Frames, Earley Parsing, Parse Forest, Push-Down Automata.

Topic: Parsing (Análisis sintáctico)

Efficient Sharing in Ambiguous Parsing

Manuel Vilares Ferro

Facultad de Informática
Campus de Elviña, s/n
15071 La Coruña, Spain

Abstract

At the outbreak, interest in non-deterministic parsing was due to the fact that it is the context-fre backbone of some natural language analyzers. In effect, several of them start with a purely context-fre parsing phase which generates all possible parses as some kind of sharing structure usually called shared forest. Later, in a second phase, a more elaborate analyzer that takes into account the finer grammatical structure of the language is applied in order to filter out undesirable parses. That implies the existence of a disambiguation process, from which a single tree is to be chosen in the resulting shared forest. In this paper[1], we study the structure of these forests with respect to optimality of syntactic sharing, in relation to the parsing schema used to produce them and to the number of computations to be applied during the parse process.

Key Words and Phrases: Context-Free Parsing, Dynamic Programming, Dynamic Frames, Earley Parsing, Parse Forest, Push-Down Automata.

1 Introduction

Using context-free grammars for the description of natural language has regained much atte in theoretical linguistics in last years [3]. Grammars of this type are well suited to computat use and it seems feasible that by using them, systems of the future could have natural lan components which are both computational efficient and linguistically elegant, in contrast t heuristic approach of many older systems. We are not suggesting that there is a contex grammar for a given natural language. It is probably more appropriate to view the gramma convenient control structure for directing the analysis of the input string. The overall analy motivated by a linguistic model which is not context-free, but which can frequently make structures determined by the context-free grammar.

Another area where general context-free parsing techniques has been considered is s recognition [8]. Here, the input language can only be approximately defined, and individual can very widely from the norm. Thus, the goal is to find a parse which most closely match input. Ambiguity arises, since each unit of the input can be considered to be a distorted v of any of several possible sounds with various probabilities.

At this point, given a sentence in a language defined by a context-free grammar, the p process consists in building a tree structure, the parse forest, that shows how this senten be constructed according to the grammatical rules of the language. In the case of gra being the context-free backbone of some natural language, the number of possible pars may become very large when the size of sentence increases, and may even be infinite for grammars[2]. Since, it is often desirable to consider all possible parse trees for semantic proc it is convenient to merge as much as possible parse trees into a single structure that allow to share common parts. This sharing save on the space needed to represent the trees, and the later processing of these trees since it may allows to share between two trees the pro of some common parts.

[1]this work was partially supported by the ESF project, Eureka Software Factory project, and by the autonomous Gove Galicia under project XUGA10501A93.

[2]which seem of little linguistic usefulness [10], except may be for analyzing ill-formed sentences [6, 12].

In this paper, we shall call shared forests such data structures used to represent simultaneously all parse trees for a given sentence. At this point, several questions may be asked in relation with shared forests:

- How to construct them during the parsing process ?
- How good is the sharing of tree fragments between ambiguous parses, and how can it be improved ?
- What is the appropriate data structure to improve sharing and reduce time and space complexity ?
- Is there a relation between the coding of parse trees in the shared forest and the parsing schema used ?

These questions are of importance in practical systems because the answers impact both the performance and the implementation techniques. For example good sharing may allow a better factorization of the computation that filters parse trees with the secondary features of the language. The representation needed for good sharing or low space complexity may be incompatible with the needs of other components of the system. These components may also make assumptions about this representation that are incompatible with some parsing schemata.

1.1 Previous work

In relation to context-free parsing, we chose to work in the context of the application of deterministic techniques to generate very efficient non-deterministic parsers, as proposed by Lang in [4]. Essentially, Lang considers a simple variation of Earley's dynamic programming construction [2]. In order to solve the problems derived from grammatical constraints, the author extends Earley's classic algorithm to push-down transducers (PDTs), separating the execution strategy from the implementation of the push-down automaton (PDA) interpreter. So, Lang obtains a family of general context-free parallel parsers for each family of deterministic context-free push-down ones, simulating all possible computations of any PDT, at worst in cubic time[3]. In addition, this work allows us to consider a general framework that encompasses all forms of chart parsing and shared forest building in a unique formalism. That is, Lang provides a uniform framework to comparing parsers. In this context, the same author proposes to represent a parse as the string of the context-free grammar rules used in a leftmost reduction of the parsed sentence, rather than as a parse tree.

At this point, the current paper is a natural continuation of the work developed by Billot and Lang in [1]. So, experimenting with several compilation schemata has shown that sophistication may have a negative effect on the efficiency of all-path parsing. In effect, sophisticated PDT construction techniques tend to multiply the number of special cases, thereby increasing the code size of the chart parser, but also preventing sharing of locally identical subcomputations because of differences in context analysis. This in turn results in lesser sharing in the parse forest.

However, the same work has shown the necessity to formalize the concept of dynamic frame, as presented by Villemonte de la Clergerie in [17]. In essence, the idea consists in disregard the general context-free parsing method applied to obtain the syntactic forest, and to treat the sharing problem from the point of view of the practical representation for the data structures used in the implementation of the parser. The final goal is to explode the existing relationship between sharing of structures and sharing of computations, simply condensing the representation of the first ones. Following this way, the same author of this paper described in [13] a method to efficiently built

[3]the method is linear in a large class of grammars.

non deterministic LALR(1) automata, and in [13, 15] the corresponding interpretor, which
starting point for this work.

1.2 A simple road map

In section 2 of this paper, we give a very formal description of general context-free parsing
on the compilation paradigm. This section is essential to understand the consideration of
paper. Section 3 is the kernel of this work and it describes, the criteria to design and effi
general context-free parser, justifying from a practical point of view tactical decisions in ord
get more sharing quality. To illustrate this discussion we shall use $\mathcal{G}$, the pico-grammar of Eng
taken from [11], which is given by the productions:

(0)	$\phi \rightarrow S \dashv$	(1)	$S \rightarrow NP\ VP$	(2)	$S \rightarrow S\ PP$	(3)	$NP \rightarrow n$
(4)	$NP \rightarrow det\ n$	(5)	$NP \rightarrow NP\ PP$	(6)	$PP \rightarrow prep\ NP$	(7)	$VP \rightarrow v\ NP$

whose LR(0) finite state machine is shown in figure 1. In section 4, we give an extensive ran
tests in relation to quality in syntactic sharing. Finally, section 5 is a conclusion about the
presented.

2 Context-Free Parsing

Though much research has been devoted to this subject in the past, most of the practically
work has commented on deterministic push-down parsing which is clearly inadequate for natu
language applications and does not generalize to more complex formalisms[5]. On the other h
there has been little formal investigation of general context-free parsing, though many prac
systems have been implemented based on some variant of Earley's algorithm.

In this sense, our contribution [13, 14, 15] has been to develop a formal model which
describe these variants in a uniform way and encompasses the construction of parse forests
use a non-deterministic PDA as a virtual parsing machine which we can simulate with an Ea
like construction; variations on Earley's algorithm[6]. This uniform framework has been use
compare experimentally parsing schemata w.r.t. parser size, parsing speed and size of t
forest. This is a very important question, since the choice of a formalism is essential with re
to computational tractability if we intend to use it for mechanical processing of natural langu
as it is especially the case for interactive systems such as natural language interfaces.

That follows is a formal overview of parsing by dynamic programming interpretati
PDAs. Our aim is to parse sentences in the language $\mathcal{L}(\mathcal{G})$ generated by a context-free gram
$\mathcal{G} = (N, \Sigma, P, S)$, where N is the set of non-terminals, Σ the set of terminal symbols, P the
and S the start symbol. The empty string will be represented by ε.

2.1 The operational model

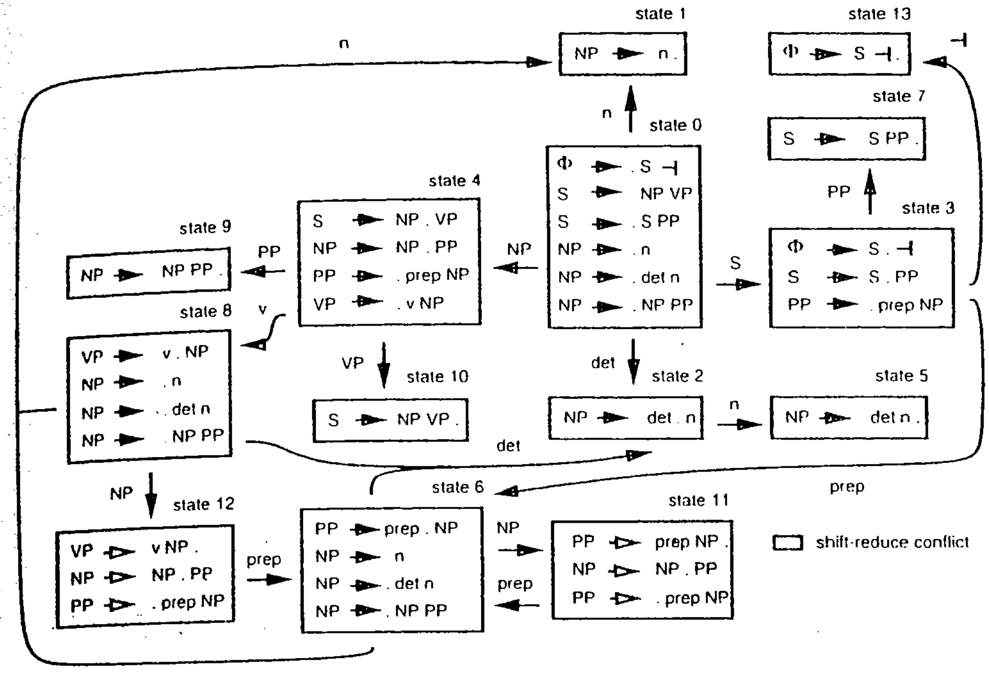
The operational model of our parser is a classic PDT, in general non-deterministic. To repres
we assume a formal definition that can fit most usual PDT construction techniques. It is d
as a 8 tuple $\mathcal{T}_{\mathcal{G}} = (\mathcal{Q}, \Sigma, \Delta, \Pi, \delta, q_0, Z_0, \mathcal{Q}_f)$, where: $\mathcal{Q}$ is the set of states, Σ the set of input
symbols, Δ the set of stack symbols, Π the set of output symbols, q_0 the initial state, Z_0 the

[4] which can be considered as a very simple context-free backbone of English.

[5] techniques to treat general context-free parsing may be applied to other linguistic formalisms as *tree adjoining grammars*
by encoding them into *definite clause programs* (DCPs). Here, the parse forest in context-free parsing case is the proof fore
Horn case. Such proof forests may be obtained by the same techniques that we use for context-free parsing [14].

[6] which suffers from an inefficiency hardly tolerable in natural language parsing.

Figure 1: The LR(0) machine for the $\mathcal{G}$ grammar



stack symbol, and Q_f the set of final states. In relation to δ, it is a finite set of transitions of the form $\tau = \delta(p, X, a) \ni (q, Y, u)$ with $p, q \in Q$; $a \in \Sigma \cup \{\varepsilon\}$; $X, Y \in \Delta \cup \{\varepsilon\}$, and $u \in \Pi^*$.

To represent the state of a PDT in a moment of the parse process, we define a *configuration* as a 4-tuple $(p, X\alpha, ax, u)$, where p is the current state, $X\alpha$ the stack contents with X on the top, ax the remaining input where the symbol a is the next to be shifted, $x \in \Sigma^*$, and u is the already produced output. The application of a transition $\tau = \delta(p, X, a) \ni (q, Y, v)$ results in a new configuration $(q, Y\alpha, x, uv)$ where the terminal symbol a has been scanned, X has been popped, Y has been pushed, and v has been concatenated to the existing output u. So, for example, if the terminal symbol a is ε in the transition, no input symbol is scanned. If X is ε then no stack symbol is popped from the stack. In a similar manner, if Y is ε then no stack symbol is pushed on the stack.

The described framework is called S^T by the dependence of the formalism of transitions on the total parse stack, which reduces the possibility of sharing computations, a fundamental problem in ambiguous parsing. We shall now reduce this dependence considering the notion of *dynamic frame*.

2.2 The concept of dynamic frame

In our context, a dynamic programming interpretation of a PDT is the systematic exploration of a space of elements called *items*. This search space is a condensed representation of all possible computations of the PDT. It is important to guarantee that all useful parts of that space are actually explored (cf. fairness, completeness), and that useless or redundant parts are ignored as much as possible (cf. admissibility). It is also necessary to assure that the representation of configurations by items is compatible with the formalism of transitions. To formalize this idea, we introduce the concept of *dynamic frame* establishing the conditions over which correctness and completeness of computations with items are verified in relation to S^T. So, given a PDT $\mathcal{T}_\mathcal{G} = (\mathcal{Q}, \Sigma, \Delta, \Pi, \delta, q_0, Z_0, \mathcal{Q}_f)$, we define a *dynamic frame* as a pair $(\Re, \mathrm{Op})$ where:

- $\Re$ is an equivalence relation on the stacks, whose classes are named *items*. We denote:
  - The class of ξ as $\bar{\xi}$.

– The set of items as $It_{\mathcal{T}_{\mathcal{G}},\Re}$, or simply It when the context is clear.

- Op is an operator of the form:

$$\begin{array}{rcl} \mathrm{Op}: \ \delta & \rightarrow & \{It^{+} \times \Sigma \cup \{\varepsilon\} \rightarrow It^{+} \times \Pi^{*}\} \\ \tau & \rightsquigarrow & \mathrm{Op}(\tau): It^{n} \times \Sigma \cup \{\varepsilon\} \rightarrow It^{m} \times \Pi^{*} \end{array}$$

verifying the following conditions:

  – *Compatibility*: all computation in S^T must have its corresponding counterpart in the dynamic frame.

  – *Completeness*: all final configuration in S^T has its corresponding counterpart in the dynamic frame.

  – *Correctness*: all final configuration in the dynamic frame has its correspondin counterpart in S^T.

where n and m depend on the nature of the transition. More exactly, they represent respectively the number of items over which we apply the transition, and the number of items resulting of this application.

Dynamic frames were originally introduced by Villemonte de la Clergerie in [17] to formalize the notion of item in relation to the use of *logical push-down automata*[7] for constructing efficient and complete definite clause programs compilers. The consideration to our particular case was proposed by the same author of this paper in [13, 14].

In practice, only two dynamic frames are considered: S^1 and S^2. Both of them construct item from the notion of *mode*. A mode is a 4-tuple (p, X, S_i^w, S_j^w), where p is the current state in ou PDT, X the last recognized symbol from state p, S_i^w a pointer to the position i in the input strin w containing the first token derived from the symbol X, and S_j^w is a pointer to the position of the token currently analized. The only difference between S^1 and S^2 consists in the numbe of modes considered. So, S^1 considers an item as an structure composed by an only mode whil S^2 considers items constructed by two modes in the form $[(p, X, S_i^w, S_j^w), (q, Y, S_k^w, S_i^w)]$, wher the final configuration of the PDT is given by the first mode. This implies, in essence, that S represents the current configuration of the PDT by the top of the stack while S^2 uses also th preceding element in the stack.

2.3 The syntactic formalism

An apparently major difference with most other parsers is that we represent a parse as the cha of the context-free grammar rules used in a leftmost reduction of the parsed sentence, rather th as a parse tree. When the sentence has several distinct parses, the set of all possible parse chai is represented in finite shared form by a context-free grammar that generates that possibly infini set. However, this difference is only appearance.

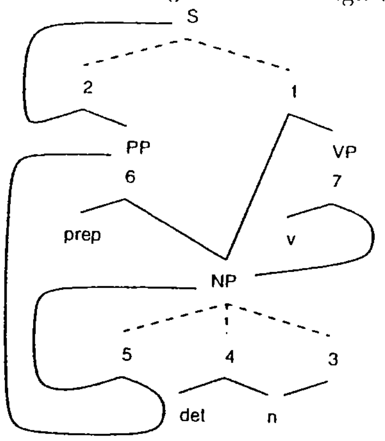
In effect, context-free grammars can be represented by AND-OR graphs that in our case i precisely the shared-forest graph. More exactly, OR-nodes are represented by the non-termi categories, and AND-nodes are represented by the rules of the grammars. There are also le nodes corresponding to the terminal categories. The OR-node corresponding to a non-terminal has exiting arcs leading to each AND-node n representing a rule that defines X. If there is o such arc, that is represented by placing n immediately under X. The sons of an AND-node are l

[7]essentially, PDAs that store atoms and substitutions on their stack, and use unification to apply transitions. They are du Lang [5], which obtains an exponential reduction in complexity over the traditional resolution methods.

grammatical categories found in the right-hand-side of the rule, in that order. The convention for orienting the arcs is that they leave a node from below and reach a node from above.

A characteristic of the AND-OR graph representing a grammar is that all nodes have different labels. Conversely, any labeled AND-OR graph such that all node labels are different may be translated into a context-free grammar such that AND-node labels are rule names, OR-node labels represent non-terminal categories, and leaf-node labels represent terminal categories. As an example, we show a modification of the pico-grammar of English in figure 2, using an AND-OR graph.

Figure 2: A representation of the pico-grammar of English using AND-OR graphs



2.4 The parser

We assume that using a standard technique we produce a recognizer for the language $\mathcal{L}(\mathcal{G})$ based on a PDA, possibly non-deterministic, from the context-free grammar $\mathcal{G}$.

The algorithm proceeds by building a collection of items. We associates a set of items S_i^w, habitually called *itemset*, for each word symbol w_i at the position i in the input string of length n, $w_{1..n}$. New items are produced by applying transitions to existing ones, until no new application is possible. However, generation of items is dependent on the type of dynamic frame. So, in S^2 the process is exactly the same applied on S^T. The reason is simple, transitions on the PDT depend on the first, and also perhaps the second element in the stack, but this information is always available in S^2. Analytically, given a transition $\tau = \delta(p, X, a) \ni (q, Y, u)$ in S^T, we translate it to S^2 in the form of a transition $Op(\tau)$ given by:

$$
\begin{array}{llllll}
1. & \tilde{\delta}([(p,X,S_j^w,S_i^w),(r,Z,S_l^w,S_k^w)],a) & \ni & ([(q,\varepsilon,S_i^w,S_i^w),(p,X,S_j^w,S_i^w)], & \varepsilon) & \textbf{if} \;\; Y = X \\
2. & \tilde{\delta}([(p,X,S_j^w,S_i^w),(r,Z,S_l^w,S_k^w)],a) & \ni & ([(p,Y,S_i^w,S_{i+1}^w),(p,X,S_j^w,S_i^w)], & I_0 \rightarrow a) & \textbf{if} \;\; Y = a \\
3. & \tilde{\delta}([(p,X,S_j^w,S_i^w),(r,Z,S_l^w,S_k^w)],a) & \ni & ([(p,Y,S_i^w,S_i^w),(r,Z,S_l^w,S_k^w)], & I_1 \rightarrow I_2) & \textbf{if} \;\; Y \in N \\
4. & \tilde{\delta}([(p,\varepsilon,S_j^w,S_i^w),(q,Y,S_k^w,S_j^w)],a) & \ni & ([(q,\varepsilon,S_k^w,S_i^w),(r,Z,S_l^w,S_k^w)], & I_3 \rightarrow I_4 I_5) & \textbf{if} \;\; Y = \varepsilon \\
 & & & \forall\, [(q,Y,S_k^w,S_j^w),(r,Z,S_l^w,S_k^w)] & &
\end{array}
$$

where:

$$\tilde{\delta} : \mathrm{It} \times \Sigma \cup \{\varepsilon\} \longrightarrow \Pi^*$$

and

$$
\begin{array}{ll}
I_0 = [(p,Y,S_i^w,S_{i+1}^w),(r,Z,S_l^w,S_k^w)] & I_1 = [(p,Y,S_i^w,S_i^w),(p,X,S_j^w,S_i^w)] \\
I_2 = [(p,X,S_j^w,S_i^w),(r,Z,S_l^w,S_k^w)] & I_3 = [(q,\varepsilon,S_k^w,S_i^w),(r,Z,S_l^w,S_k^w)] \\
I_4 = [(q,Y,S_k^w,S_j^w),(r,Z,S_l^w,S_k^w)] & I_5 = [(q,\varepsilon,S_k^w,S_i^w),(r,Z,S_l^w,S_k^w)]
\end{array}
$$

with *It* the set of all items developed in the parsing process, and Π is given by a set of context-free rules directly built from items. Succinctly, we can describe the preceding cases as follows:

1. Corresponds to a goto action from the state p to state q under transition X.
2. Corresponds to a push of terminal a from state p. The new item belongs to the itemset S^w_{i+1}.
3. Corresponds to a push of non-terminal Y from state p.
4. Corresponds to a pop action from state p, where q is an ancestor of state p under transition X in the automaton. At this point, we must remark that the item I_4 can already exists or to be generated. In this sense, we use the term of *items to see again* to refer them. Intuitively, we can recover the configuration resulting from the application of pop actions using the information represented by the second mode of the item in the top of the stack in our PDT.

In relation to S^2, S^1 considers a more compact representation for the stack. That is an important point, since the more compact are these representations, the more successful will be the sharing of computations, and later of syntactic structures. Given a transition $\tau = \delta(p, X, a) \ni (q, Y, u)$ in S^T, we translate it to S^1 in the form of a transition $Op(\tau)$ given by:

$$\begin{array}{lllll}
1. & \tilde{\delta}([p, X, S^w_j, S^w_i], a) & \ni & ([q, \varepsilon, S^w_i, S^w_i], \quad \varepsilon) & \textbf{if} \;\; Y = X \\
2. & \tilde{\delta}([p, X, S^w_j, S^w_i], a) & \ni & ([p, Y, S^w_i, S^w_{i+1}], \quad I_0 \rightarrow a) & \textbf{if} \;\; Y = a \\
3. & \tilde{\delta}([p, X, S^w_j, S^w_i], a) & \ni & ([p, Y, S^w_i, S^w_i], \quad I_1 \rightarrow I_2) & \textbf{if} \;\; Y \in N \\
4. & \tilde{\delta}([p, \varepsilon, S^w_j, S^w_i], a) & \ni & \tilde{\delta}_d([q, \varepsilon, S^w_l, S^w_i], a) \ni ([q, \varepsilon, S^w_l, S^w_j], \quad I_3 \rightarrow I_4 I_5) & \textbf{if} \;\; Y = \varepsilon \\
 & & & \forall q \in Q \text{ such that: } \exists\, \delta(q, X, \varepsilon) \ni (p, X, \varepsilon) &
\end{array}$$

with:

$$\tilde{\delta} : \mathrm{It} \times \Sigma \cup \{\varepsilon\} \longrightarrow \{\mathrm{It} \cup \tilde{\delta}_d\} \times \mathrm{It}^* \qquad\qquad \tilde{\delta}_d : \mathrm{It} \times \Sigma \cup \{\varepsilon\} \longrightarrow \mathrm{It}$$

and

$$\begin{array}{lll}
I_0 = [p, Y, S^w_i, S^w_{i+1}] & I_1 = [p, Y, S^w_i, S^w_i] & I_2 = [p, X, S^w_j, S^w_i] \\
I_3 = [q, \hat{\varepsilon}, S^w_l, S^w_i] & I_4 = [q, X, S^w_l, S^w_j] & I_5 = [p, \varepsilon, S^w_j, S^w_i]
\end{array}$$

where $\tilde{\delta}_d$ is called the set of *dynamic transitions*.

The description of the transitions 1.,2. and 3. is the same preceding considered in S^2. In relation to the transition numbered 4., it corresponds to a pop action from state p, where q is an ancestor of state p under transition X in the automaton. In this case, we do not generate a new item, but a *dynamic transition* $\tilde{\tau}_d$ to treat the absence of information about the rest of the stack.

It is important to comment the behavior of the algorithm face to a pop action, the last case represented. In effect, given that our compact representation of the stack is its top, we must consider a protocol to treat the absence of information about the rest of the stack. The solution relies to the concept of *dynamic transition*. Briefly, it consists in generating a new transition from that implying the pop action. This new transition must be built in such a manner that it is applicable not only to the configuration resulting of the first one, but also on those to be generated and sharing the same syntactic structure. A detailed explanation about this is given by the same author of this paper in [13, 15].

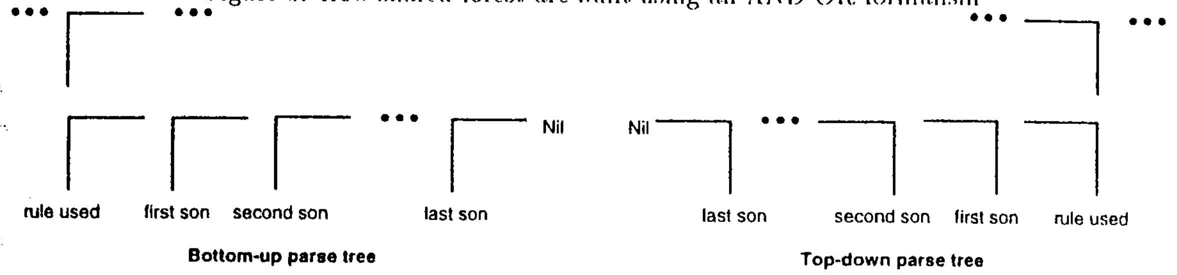
At this point, items are not only elements of the computation process, but also non-terminals of the output grammar. That allows us to identify items with nodes in the resulting parse forest, since this output grammar will be our syntactic formalism to represent the resulting shared forest.

In relation to fairness and completeness, an equitable selection order must be established to treat items. We use a technique of *merit ordering*. In essence, we process the items in an itemset in order, performing none or some transitions on each one depending on the form of the item. These operations may add more states to the current itemset and may also put states in the itemset corresponding to the following token to be analyzed from the input string. To ignore redundant items we put in place a simple subsumption relation based in the equality.

3 Sharing Forests

We shall now display the mechanisms that cause the phenomenon of tree duplication. As a consequence, we shall also find criteria to avoid duplication of computations during the parsing process since nodes in the parse forest are items in the computation process. That is, it will be very simple to establish a criterion to decide the quality of the syntactic sharing: The number of generated items. To illustrate the following discussion, we shall analyze the ambiguous sentence "*I saw a man with a telescope*", using the pico-grammar of English.

Figure 3: How shared forest are built using an AND-OR formalism



3.1 Dependence on the syntactic formalism

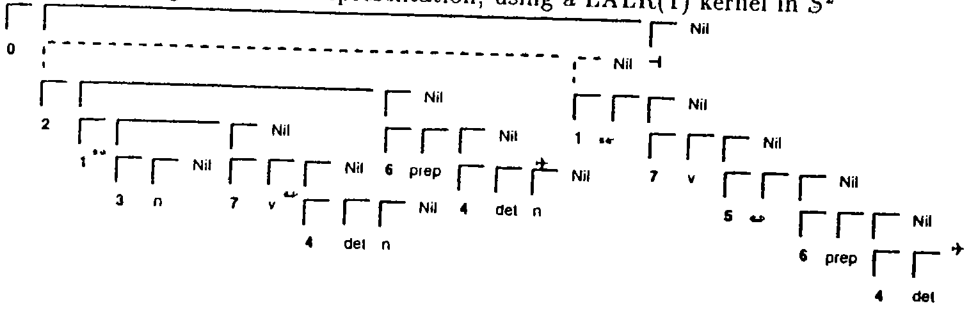
In our formalism, when the sentence has several distinct parses, the set of all possible parse strings is represented in finite shared form by a context-free grammar that generates that possibly infinite set. This technique is interesting by a lot of reasons:

- The syntactic data structures are the same used in the computation process. That allows us:
  - To establish a simple relationship between sharing of computations and syntactic sharing.
  - Given that, from a syntactic point of view, actions on the PDA depend on the first and possibly second elements in the stack; the output grammar is a 2-form one. As a consequence, we obtain two interesting features that are not usual in other context-free parsing algorithms:
    - * Time complexity for the parser is $\mathcal{O}(n^3)$, where n is the length of the sentence. This result is achieved without requiring that the language grammar be in Chomsky Normal Form.
    - * Sharing of a tail of sons in a node of the resulting forest is possible. More exactly, bottom-up parsing may share only the rightmost constituents, while top-down parsing may only share the leftmost ones. The reason is simple and relies to the type of search used to built the forest. Breadth first search results on bottom-up constructions and depth first search results on top-down ones, as it is shown in figure 3.
- We can simply represent a possible infinite set of trees.

In definitive, syntactic sharing quality seems to be better using a descriptive formalism based on AND-OR graphs to represent shared forest. To show that from a practical point of view, it is sufficient to study the shared forest represented in figure 4. In effect, the syntactic formalism used to represent these forest is not the classic, which allows us to share the tail of sons indicated by an icon with an airplane. This would not be possible using a classic representation because in this case sharing is only applicable to proper nodes, as it is shown in figure 5.

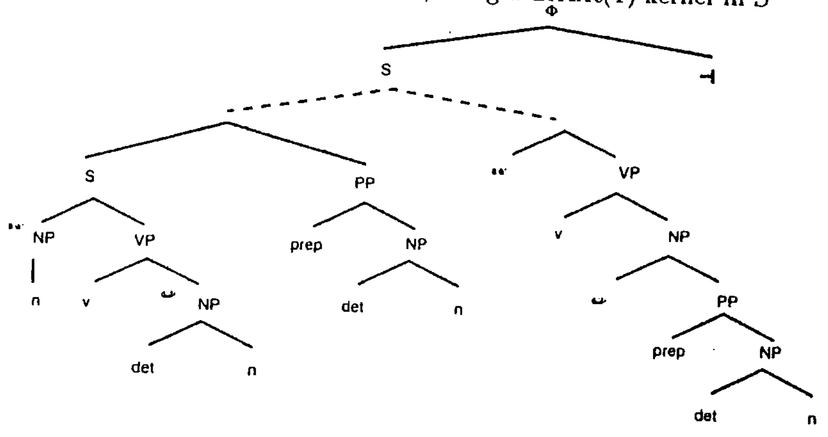
At this point, it is important to remark that in our examples, we have only considered bottom-up approaches. A reason to do it is that, in general, the consideration of S^1 will be only possible

Figure 4: ICE representation, using a LALR(1) kernel in S^2



when the parsing algorithm representing the kernel of the system assures that it can complete the absence of information represented by the second mode of the S^2. This is not the case of top-down parsers[8], where the algorithm cannot ignore the information given by the second mode because their predictive behavior, which does not allow to the parser to recover during pop transitions all the information previously predicted. For a more detailed explanation about that, the reader can consult [13].

Figure 5: Classic representation, using a LALR(1) kernel in S^2



3.2 Dependence on the parsing scheme

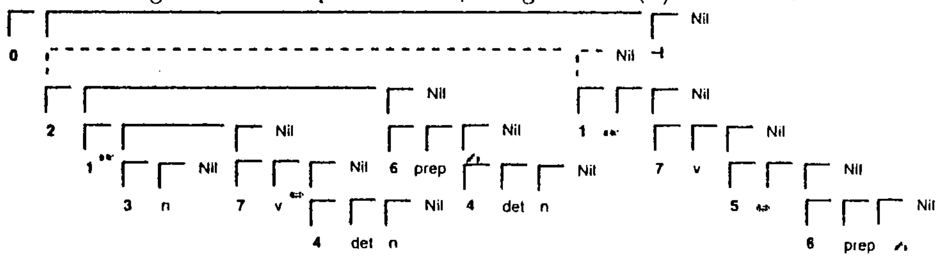
We shall now turn our attention to the properties relying to the parsing process itself. In particular, the existence of different *syntactic contexts* when analyzing a given input. Formally, given a PDT $\mathcal{T}_{\mathcal{G}} = (\mathcal{Q}, \Sigma, \Delta, \Pi, \delta, q_0, Z_0, \mathcal{Q}_f)$, $w_{1..n}$ an input string, and ξ a stack corresponding to the analysis of $w_{1..i}$, $i < n$; we say that w_{i+1} is analyzed in a *left syntactic context* ξ.

From an intuitive point of view, the left syntactic context represents the information accumulated by the system during the preceding parse. This information strongly determines which will be the continuation of the parsing process, since the current state of our virtual machine depends on it. As a consequence, if we parse a same input in different left syntactic contexts[9], the resulting forest will depend on those and sharing will not be guaranteed. To illustrate this when the kernel is an extended LALR(1) automaton, the reader can easily verify in figure 1 that the production $PP \rightarrow prep\ NP$ can be analyzed in two different syntactic contexts represented by the states numbered 3 and 12. We can translate that in a lost of efficiency in relation to syntactic sharing whichever it is the syntactic formalism considered in each case. So, for example, using an

[8]also called predictive parsers.

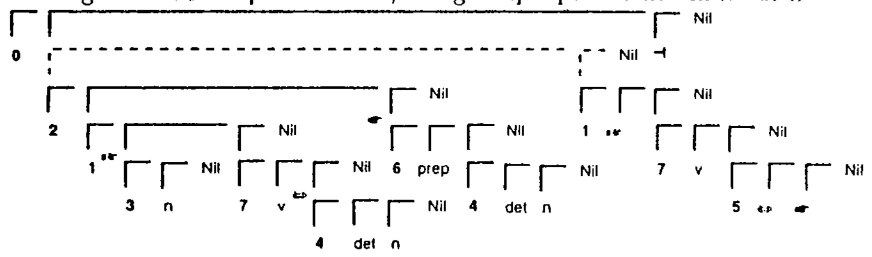
[9]as it is the case when ambiguities arise.

Figure 6: ICE representation, using a LALR(1) kernel in S^1



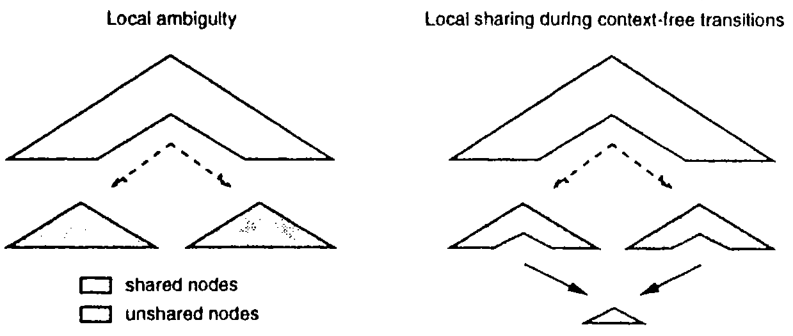
ND-OR graph representation we obtain the shared forest represented in figure 6 considering a S^1 dynamic frame. Consequently, the quality of the syntactic sharing is better when the kernel is n algorithm of simple precedence, as it is shown in figure 7, since here the concept of state cannot ifferentiate between irrelevant syntactic contexts. In this last case, we can share the previous ode, but also its father in the forest indicated by an icon representing a black hand.

Figure 7: ICE representation, using simple precedence in S^1 or S^2



At this point, it is clear that sharing of computations and sharing of structures is guaranteed hen the left syntactic context is the same for several parsing process working on a same input tring. This has a practical interest when analyzing a natural language. In effect, usually mbiguities are local in the sense that we can unify them in a same node of the parse forest. t this moment, the left context is the same and sharing is total between the trees affected by hat ambiguity as it is represented in figure 8. In our example, this unification is only possible nce we have reduced the two possible analysis for the ambiguous node S.

Figure 8: Cases to be considered in syntactic sharing



However, it is possible that sharing will be feasible even on different contexts, for certain ransitions in the automaton. A particular interesting case is represented by push and horizontal ransitions[10], that is, those depending only on the top of the stack. In this sense, given a PDA

[10] which do not change the stack configuration.

$\mathcal{A} = (\mathcal{Q}, \Sigma, \Delta, \delta, q_0, Z_0, \mathcal{Q}_f)$ and ξ_0, ξ_n two different stacks, we say that there is a *context-free derivation* in n steps from ξ_0 to ξ_n when:

- There exists a sequence $\xi_{1..n-1}$ verifying that $\xi_{i-1} \vdash \xi_i$, $\forall\, i \in \{1, ..., n\}$.
- $\forall\, i \in \{1, ..., n\}$, $\exists\, \xi'_i$ not empty, such that $\xi_i = \xi'_i \xi_0$.

and we shall denote it as $\xi_0 \Vdash^{n} \xi_n$.

Taking again into account that nodes in the resulting shared forest are items in the parsing process, we can conclude that forest building does not depend on the syntactic context during context-free derivations. From a practical point of view, this situation corresponds to the existence of some common son between the alternatives for an ambiguous node, as it is represented in figure 8. That allows us to locally sharing computations and syntactic structures while the sentence actually being analyzed is not dependent on the context. This is, for example, the case of those nodes indicated by icons representing a hand with a pencil in figure 6.

In relation with the preceding discussion, one essential guideline to achieve better sharing is to try to recognize every grammar rule in only one place of the generated chart parser code, even at the cost of increasing non-determinism. So, we can guarantee that sharing will be effective as much as possible, since the number and the extension of different left syntactic contexts will be minimized and therefore sharing quality will be maximized. This is a simple idea on which we base the formalization of the sharing problem on ambiguous parsing. More exactly, if $\mathcal{A} = (\mathcal{Q}, \Sigma, \Delta, \delta, q_0, Z_0, \mathcal{Q}_f)$ is a PDA, then we have that

$$\exists\, \xi,\ d\xi \Vdash^{n} \zeta d\xi \Leftrightarrow \forall\, \kappa,\ d\kappa \Vdash^{n} \zeta d\kappa$$

Intuitively, this implies that both push and horizontal transitions are only dependent on the top of the stack, that is, on the last performed parsing action. In essence, this result introduces the origin of the different behavior exhibit, in relation to syntactic sharing, by parsing algorithms based on a same approach to produce the parse tree[11], but considerating different determinization techniques. In effect, most of these, as the classic LR(k) ones, has as consequence the deformation of the initial grammar by virtue of the application of a predictive technique in the generation of the parser[12]. In fact, we are talking about the known state splitting phenomenon of LR(k) parsers that can introduce different syntactic contexts, since some grammar rules can be recognized in several schemes in the PDT. Such a deformation has habitually as origin some of the following two ones:

- The use of lookahead techniques, as in the case of the LR(k) algorithms, when $k \geq 1$.
- The differentiation of the recognition of a symbol in function of the *rule context* in which it has been performed. This is typically the case for LR(0) approaches and all algorithms taking as basis the CFSM[13], such as LALR(k) and SLR(k) methods. This is also one of the causes of state splitting in LR(k), when $k \geq 1$.

In this sense, from the point of view of the sharing quality, pure bottom-up techniques as simple precedence are more efficient. In effect, it takes only into account grammatical features. The reverse of the coin is represented by its more restrictive deterministic domain, which we can translate into an inefficiency in the treatment of the local determinism phenomenon. Thus, as we have before commented, ambiguities have usually a local behavior and during most of the time deterministic parsing would be possible.

[11] fundamentally bottom-up and top-down methods.

[12] in the case of the LR(k) algorithms this implies the generation of the corresponding finite state machine that differences the syntactic contexts in function of the state where the parsing process is.

[13] that is, the **C**haracteristic **F**inite **S**tate **M**achine of the LR(0) parsers.

3.3 Dependence on the dynamic frame

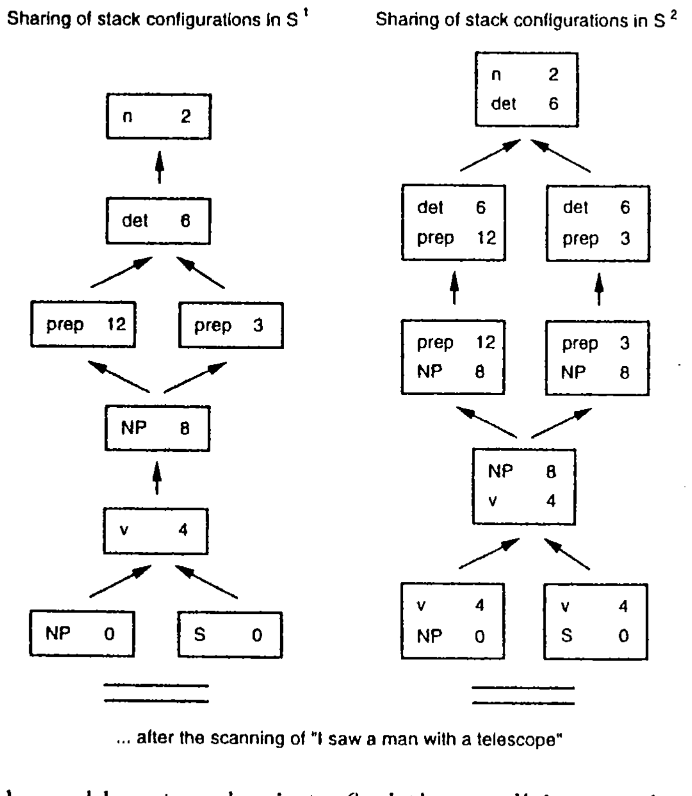
We shall now disregard the general context-free parsing method applied to obtain the forest, and we shall treat the problem from the point of view of the stack representation used during the parsing process. More exactly, we shall turn our attention to the two most commonly used dynamic frames, denoted S^1 and S^2.

Intuitively, it is not difficult to predict the consequences of the use of a given dynamic frame. In effect, items in S^1 will permit a better sharing of structures, since they contains only one mode[14] and therefore the length of the context-free derivations can be longer. In relation to this, S^2 cannot be considered as optimal because the continuous dependence on the context represented by the second mode of the items. We can illustrate this situation in our example, considering the particular case of extended LALR(1) automata. So, in figure 6 we can share the node indicated by an icon representing a hand with a pencil, while in figure 4 this is not possible. The reason is that the first one corresponds to an analyze in S^1 and the second one to an analyze in S^2. To justify this behavior it is sufficient to consider which are the configurations of the stack in the LALR(1) automaton once we have scanned the substring *"I saw a man with a telescope"*, to later apply in sequence the reductions corresponding to the rules:

$$\begin{array}{llll} (4) & \text{NP} & \rightarrow & \text{det} \quad \text{n} \\ (6) & \text{PP} & \rightarrow & \text{prep} \quad \text{NP} \end{array}$$

The reader can see such a configurations in figure 9, where it is easily verifiable that in the case of use S^1, all pop actions can be shared during the reduction of the rule numbered (4). This is not the case in S^2, because the dependence on the second mode of the items.

Figure 9: Sharing stack configurations in S^1 and S^2


Sharing of stack configurations in S 1
Sharing of stack configurations in S 2
n 2
det 6
prep 12
prep 3
NP 8
v 4
NP 0
S 0
n 2 det 6
det 6 prep 12
det 6 prep 3
prep 12 NP 8
prep 3 NP 8
NP 8 v 4
v 4 NP 0
v 4 S 0
... after the scanning of "I saw a man with a telescope"

At this point, the only problem to solve is to find the conditions under which we can consider S^1 or S^2 as dynamic frame. In effect, given that in S^T actions depend on the first and perhaps

[14]in essence, the top of the corresponding stack in S^T.

also the second element in the stack, compatibility, completeness and correctness are assured in all cases when we work in S^2. However, we have seen that the best possibilities do not correspond to S^2, but to S^1. So, we can summarize the advantages of S^1 compared to S^2 in three points, all of them related to the use of a more complex structure in S^2:

- Better space complexity.
- Higher subsumption possibilities.
- Better sharing efficiency.

As a consequence, our initial problem can be reduced to verify the conditions under which S^1 can be considered, in other cases S^2 will be the most adequate dynamic frame.

It is also important to remark that although the systematic use of S^1 guarantees the best sharing quality for a given parsing algorithm, this does not necessarily imply that the resulting forests are always prefectly shared, but only that the computations are reduced to the strictly necessary from the point of view of the parsing algorithm. In our example, this kind of phenomena is located in figure 6 where it is not possible to share those nodes representing the reduction of rule (6) because they are in different syntactic contexts in the LALR(1) automaton. If we try to share them, using such an operational formalism, the result is not a reduction of the computations, but the augmentation in the complexity of that computation process.

Summing it up, the choice of a particular dynamic frame is central to assure a good sharing computation process, essential to guarantee efficiency in a non-deterministic context. In practice, the situations justifying the consideration of S^2 instead of S^1 limit us to the case of the use of predictive parsers.

4 Experimental Results

Once introduced the essence of the problem of syntactic sharing, our goal is now to prove the veracity of this reasoning, in practice. We use the syntax of ambiguous arithmetic expressions to show efficiency of the parsing process in relation to the dynamic framework considered. This is not an example of natural language, but at this point of the discussion it is clear that to put into evidence the behavior face to sharing properties, a small context-free grammar with a high density of ambiguities is the most appropriate.

Results are shown in table 1 for the number of computations in the parsing process and in table 2 for the statistics in relation to sharing of these computations. The grammar is given by the following productions:

$$(0)\quad S \to S + S \qquad (1)\quad S \to S * S \qquad (2)\quad S \to (\ S\ ) \qquad (3)\quad S \to number$$

In relation to the nature of the tests, we must take into account some points:

1. All tests have been performed using an extended LALR(1) parser generated by ICE [13, 14, 15, 16], which has 17 shift/reduce conflicts.
2. All tests have been performed using the same input programs.
3. Test programs are of the form:

$$b\{+b\}^i$$

where i is the number of $+$'s. As the grammar contains a rule $S \rightarrow S + S$, these programs have a number of ambiguous parses which grows exponentially with i. This number is:

$$C_i = \begin{cases} 1 & \textbf{if } \; i = 0, 1 \\ \begin{pmatrix} 2i \\ i \end{pmatrix} \frac{1}{i+1} & \textbf{if } \; i > 1 \end{cases}$$

All tests have shown the better behavior of S^1 face to other dynamic frames as S^2 and S^T.

Table 1: Statistics on the number of items generated by ICE

Ambiguities	0	2	5	14	42	132	429	$1.430e+3$	$4.862e+3$	$1.6796e+4$
S^1 **items**	8	14	21	29	38	48	59	71	84	98
S^2 **items**	8	14	22	32	44	58	74	92	112	134
S^T **items**	8	16	28	44	64	88	116	148	184	224

Ambiguities	$5.8786e+4$	$2.08012e+5$	$7.429e+5$	$2.67444e+6$	$9.694845e+6$
S^1 **items**	113	129	146	164	183
S^2 **items**	158	184	212	242	274
S^T **items**	268	316	368	424	484

Ambiguities	$3.535767e+7$	$1.2964479e+8$	$4.776387e+8$	$1.767263190e+9$
S^1 **items**	203	224	246	293
S^2 **items**	308	344	382	464
S^T **items**	548	616	688	844

Table 2: Statistics on the number of stacks configurations shared by ICE

Ambiguities	0	2	5	14	42	132	429	$1.430e+3$	$4.862e+3$	$1.6796e+4$
S^1 **items**	0	2	7	15	26	40	57	71	100	126
S^2 **items**	0	2	6	12	20	30	42	56	72	90

Ambiguities	$5.8786e+4$	$2.08012e+5$	$7.429e+5$	$2.67444e+6$	$9.694845e+6$
S^1 **items**	155	187	222	260	301
S^2 **items**	110	132	156	182	210

Ambiguities	$3.535767e+7$	$1.2964479e+8$	$4.776387e+8$	$1.767263190e+9$
S^1 **items**	345	392	422	551
S^2 **items**	240	272	306	380

5 Summary and Conclusions

In practice, the published general context-free parsing algorithms do not always give shared forest with a maximum sharing. This may result in forest that are larger or more complex. However this characteristic does not invalidate their presentation. In effect, we have shown that the efficiency in context-free parsing is due to a compromise between the sharing of computations and the sharing of resulting forests.

More exactly, the described work exposes close connection between parsing process, the kind of syntactic data structure and the general parsing framework, in order to get computational efficiency. So, the best results have been obtained by using a dynamic frame S^1, where we can compact the stack representations as much as possible, and bottom-up parsers with a moderate state splitting phenomenon. However, we cannot always use S^1 as dynamic frame. In particular, S^2 would not be considered when the kernel for the parser is a top-down one.

additional drawback in relation to the classic one for top-down algorithms: Working through th input from left to right, this class of methods consider many useless alternatives when predicti how the parse might continue. For a large grammar with a high level of ambiguities, as it is the ca of context-free backbones for natural languages, this slow down the parsing process considerabl

References

[1] **Billot, S. and Lang, B.**
The Structure of Shared Forest in Ambiguous Parsing.
1989. Research Report n^o1038, INRIA Rocquencourt, France.

[2] **Earley, J.**
An Efficient Context-Free Parsing Algorithm.
1970. Communications of the ACM, vol. 13, n^o2 (pp. 94-102).

[3] **Garztar, G.; Klein, E.; Pullum, G. D. and Sag, I.**
Generalized Phrase Structure Grammar
1985. Basil Blackwell, Oxford, United Kindown.

[4] **Lang, B.**
Deterministic Techniques for Efficient Non-deterministic Parsers.
1974. Research Report n^o72, INRIA Rocquencourt, France.

[5] **Lang, B.**
Complete Evaluation of Horn Clauses, an Automata Theoretic Approach.
1988. Research Report n^o913, INRIA Rocquencourt, France.

[6] **Lang, B.**
Parsing Incomplete Sentences.
1988. COLING'88, vol. 1, Vargha, D. (ed.), Budapest, Hungary (pp. 365-371).

[7] **Lang, B.**
Towards a Uniform Formal Framework for Parsing.
1991. Current Issues in Parsing Technology, M. Tomita ed., Kluwer Academic Publishers (pp. 153-171).

[8] **Paeseler, A.**
Modification of Earley's Algorithm for Speech Recognition.
1988. NATO ASI Series, vol. F46 (pp. 466-472).

[9] **Pereira, F. C. N. and Warren, D. H. D.**
Definite Clause Grammars for Language Analysis. A Survey of the Formalism and a Comparison with Augmented Transition Networks.
1980. Artificial Intelligence, vol.13 (pp. 231-278).

[10] **Pereira, F. C. N. and Warren, D. H. D.**
Parsing as Deduction.
1983. Proc. of the 21st Annual Metting of Association for Computational Linguistics, Cambri Massachusetts, U.S.A (pp. 137-144).

[11] **Tomita, M.**
An Efficient Augmented Context-Free Parsing Algorit
1987. Computational Linguistics, vol. 13, n^o1,2 (pp. 3 36).

[12] **Tomita, M. and Saito, H.**
Parsing Noisy Sentences.
1988. COLING'88, Budapest, Hungary (pp. 561-566)

[13] **Vilares Ferro, M.**
Efficient Incremental Parsing for Context-Languages.
1992. Doctoral thesis, University of Nice, France.

[14] **Vilares Ferro, M. and Graña Gil, J.**
Parsing as Resolution.
1993. Proc. of the First Compulog Network Meeting Parallelism and Implementation Technologies, Madri Spain.

[15] **Vilares Ferro, M.**
An
Efficient Context-Free Backbone for Natural Langua Analyzers.
1994. Boletín de la SEPLN, n^o14 (pp. 201-214).

[16] **Vilares Ferro, M. and Dion B.**
Efficient Incremental Parsing for Context-Languages.
1994. Proc. of the 5th IEEE International Conference Computer Languages, Toulouse, France (pp. 241-252)

[17] **Villemonte de la Clergerie, E.**
Automates à Piles et Programmation Dynamique. 199
Doctoral Thesis, University of Paris VII, France.