A Comparison for Unification-Based Parsers

M. Vilares Ferro, D. Cabrero Souto, M.A. Alonso Pardo

Abstract

Unification-based grammars have been the object of study in computational linguistics over the last few years with the intention of creating powerful parsing environments. However, it is not common to find practical studies about what the real interest of these techniques is, and which approaches are better adapted in each case.

We attempt to justify the practical consideration of dynamic programming in the continuum of Horn-like formalisms, showing that conclusions attained for definite clause grammars are a natural extension of the same conclusions in context-free theory. Our aim is to demonstrate the practical interest of this technology, suggesting a simple decision guide for the interested reader.

Keywords: Parsing, dynamic programming, push-down automata, static analysis, structure sharing.

1 Introduction

Highly redundant computations are a characteristic drawback in non-deterministic parsing, as is the case in natural language processing (NLP). We shall differentiate between two domains: Off-line and on-line parsers, using the terminology introduced in [8]. Off-line strategies produce some kind of preliminary grammatical skeleton over which the system must filter out the undesirable parsers taking into account the finer language structure. In this way, the number of ambiguities grows in the former phase and, if no sharing structure technique is considered, exponentially increases the number of computations in the latter one.

On-line systems apply linguistic restrictions at parsing time, which often results in a syntactic context splitting phenomenon. Here, the problem of exponential complexity appears in the subsequent treatment. This is the case of unification-based grammars, where the notion of syntactic category has been extended to a possible infinite domain and therefore the number of different parse interpretations can also become infinite.

In this paper, we briefly review the techniques usually applied to solve these problems as well as possible alternative approaches in dynamic programming. We focus on the case of definite clause grammars (DCGs) for on-line parsers, and in context-free grammars (CFGs) as the usual skeleton formalism for off-line parsers. Section 2 establishes our

Work partially supported by the Government of Spain under project HF97-223, and by the Autonomous Government of Galice under projects XUGA10505B96 and XUGA20402B97.

M. Vilares and M.A. Alonso are with the Cole Lab., Computer Science Department, University of Corunna, Campus de Elviña s/n, 15071 La Coruña, Spain. E-mail: {vilares, alonso}@dc.fi.udc.es.

D. Cabrero is currently with the Ramón Piñeiro Research Center for Humanities, Estrada Santiago-Noia, Km. 3, A Barcia, 15896 Santiago de Compostela, Spain. E-mail: dcabrero@cirp.es.

formal testing framework in order to guarantee the validity of the work, introducing the architecture to be tested. Sections 3 and 4, respectively give a survey of each parsing strategy in both cases, off-line and on-line, locating them in the formal testing framework. In section 5, we include and comment on the experimental results. Finally, section 6 is a conclusion about the work presented.

2 A common framework

In order to provide a valid support to compare the different strategies, we introduce a common descriptive parsing formalism for both CFGs and DCGs, using dynamic programming or not. To do it, we extend the original concept of the *logical push-down automaton* (LPDA) introduced by Lang in [5]. We do it as a generalization of the operational model described by the authors in [13]. An LPDA is defined as a 7-upla $\mathcal{A} = (\mathcal{X}, \mathcal{F}, \Sigma, \Delta, \$, \$_f, \Theta)$, where $\mathcal{X}$ is a denumerable and ordered set of variables, $\mathcal{F}$ is a finite set of functional symbols, Σ is a finite set of extensional predicate symbols, Δ is a finite set of predicate symbols used to represent the literals stored in the stack, $\$$ is the *initial predicate*, $\$_f$ is the *final predicate*; and Θ is a finite set of *transitions*. The *stack* of the automaton is a finite sequence of *items* $[A, it, bp, st].\sigma$, where the top is on the left, A is in the algebra of terms $T_\Delta[\mathcal{F} \cup \mathcal{X}]$, σ a substitution, it is the current position in the input string, bp is the position in this input string at which we began to look for that configuration of the LPDA, and st is a state for the driver controlling the evaluation. Transitions are of three kinds:

- *Horizontal:* $B \longmapsto C\{A\}$. Applicable to stacks $E.\rho\ \xi$, iff there exists the *most general unifier* (mgu), $\sigma = \text{mgu}(E, B)$ such that $F\sigma = A\sigma$, for F a fact in the extensional database. We obtain the new stack $C\sigma.\rho\sigma\ \xi$.
- *Pop:* $BD \longmapsto C\{A\}$. Applicable to stacks of the form $E.\rho E'.\rho'\ \xi$, iff there is $\sigma = \text{mgu}((E, E'\rho), (B, D))$, such that $F\sigma = A\sigma$, for F a fact in the extensional database. The result will be the new stack $C\sigma.\rho'\rho\sigma\ \xi$.
- *Push:* $B \longmapsto CB\{A\}$. We can apply it to stacks $E.\rho\ \xi$, iff there is $\sigma = \text{mgu}(E, B)$, such that $F\sigma = A\sigma$, for F a fact F in the extensional database. We obtain the stack $C\sigma.\sigma\ B.\rho\ \xi$.

where B, C and D are items and A is in $T_\Sigma[\mathcal{F} \cup \mathcal{X}]$, a control condition to operate the transition. The use of it and bp is equivalent to indexing the parse, which allows us to reduce the search space in dynamic programming, and to implement a garbage collector facility, by deleting information relating to earlier substrings, as parsing progresses. This relies on the concept of *itemset* [3], for which we associate a set of items to each token in the input string, and which represents the state of the parsing process at that point of the scan.

In order to deal with classic context-free formalisms, it is sufficient to substitute the notion of unification by matching, taking σ, ρ and ρ' as identity in transitions. Typically, the control predicate $\{A\}$ will represent in this case a condition over a lookahead string.

Dynamic programming is introduced by collapsing stack representations on a fixed number of *items*, typically the last two or only the top one to give rise to the dynamic frames S^2 and S^1 respectively, as opposed to the standard dynamic frame S^T where a

stack is given by all its components and backtracking is the technique to simulate non-determinism. Correctness and completeness of S^2 in relation to S^T is trivial given that transitions depend in the worst case on the last two elements in the stack. For S^1 the case is different, since during pop transitions no information is available about the element under the top of the stack. In order to solve this lack of information, we redefine the behavior of transitions on items S^1, as follows:

- *Horizontal case:* $(B \longmapsto C)(A) = C\sigma$, where $\sigma = \text{mgu}(A, B)$.
- *Pop case:* $(BD \longmapsto C)(A) = \{D\sigma \longmapsto C\sigma\}$, where $\sigma = \text{mgu}(A, B)$, and $D\sigma \longmapsto C\sigma$ is the *dynamic transition* generated by the pop transition. This is applicable not only to the item from which we had computed the top, but also to those to be generated and which share the same syntactic context.
- *Push case:* $(B \longmapsto CB)(A) = C\sigma$, where $\sigma = \text{mgu}(A, B)$.

Intuitively, S^1 provides the best sharing properties and, in consequence, optimal computational tractability. However, it can be proved that correctness is only guaranteed for weakly predictive parsing strategies [12, 2], typically pure bottom-up approaches or mixed-strategies including a predictive phase, static or dynamic, complemented with a bottom-up construction of the parse.

3 Off-line parsers

We compare several context-free parsing schema, often the grammatical formalism for the skeleton in off-line approaches, in different dynamic frames. We have considered three basic parser generators: the AGFL [6] environment[1], an implementation of the classic Earley's algorithm [3], and the GALENA [13] system. These correspond, respectively, to a classic top-down approach with backtracking, a mixed-strategy with dynamic prediction, and a mixed-strategy with static prediction based on an LALR(1) extended automaton taken from the ICE [12] environment. We have chosen ICE as the most efficient representative of the family of unrestricted LR-*like* context-free parsers that includes systems such as SDF [4] and GLR [9], both based on Tomita's algorithm [11]. To locate these parsing schema in our testing frame, we introduce the categories $\bigtriangledown_{k,i},\ i \in \{1, \ldots, n_k\}$ for each rule $\gamma_k : A_{k,0} \rightarrow A_{k,1}, \ldots, A_{k,n_k}$, whose meaning we shall later detail in each case.

3.1 A top-down architecture

Here, the symbol $\bigtriangledown_{k,i}$ shows that the first i categories in the right-hand-side of rule γ_k have already been recognized, as it is shown Fig. 1. So, we obtain the following set of transitions that characterize the parsing strategy:

1. $[\$, 0, 0, \_] \longmapsto [\bigtriangledown_{0,0}, 0, 0, \_]\ \$$
2. $[\bigtriangledown_{k,i}, it, bp, \_] \longmapsto [A_{k,i+1}, it, it, \_]\ [\bigtriangledown_{k,i}, it, bp, \_]$
3. $[A_{k,0}, it, bp, \_] \longmapsto [\bigtriangledown_{k,0}, it, bp, \_]$
4. $[\bigtriangledown_{k,n_k}, it, bp, \_]\ [\bigtriangledown_{k',i}, bp, r, \_] \longmapsto [\bigtriangledown_{k',i+1}, it, r, \_]$

which we can briefly interpret as follows:

[1] AGFL is not a real unification-based parser since it is built on the notion of affix grammar. However, it can be assimilated to a DCG parser where functional symbols are not allowed. This permit us to take into account one of the most known parsing systems available in NLP.

1. Requires the recognition of the axiom $A_{0,0}$, which we represent by $\bigtriangledown_{0,0}$.
2. Selects the leftmost unrecognized category, $A_{k,i+1}$.
3. The body of γ_k becomes a sequence of new categories to be recognized.
4. After recognition of γ_k, we return to the calling rule γ'_k.

The state, here represented by "_", has no operative sense in this approach.

3.2 Earley's approach

As in the precedent case, Earley's algorithm requires the same interpretation for symbols $\bigtriangledown_{k,i}$ and states have no operative sense in items. In addition, given a category $A_{k,i}$, we shall consider the associated symbols $A'_{k,i}$ and $A''_{k,i}$ to respectively indicate that $A_{k,i}$ is yet to be recognized or has been already recognized. So, the parsing scheme can be defined by the transitions:

1. $[\$, 0, 0, \_] \longmapsto [A'_{0,0}, 0, 0, \_]\ \$$
2. $[A'_{k,0}, it, it, \_] \longmapsto [\bigtriangledown_{k,0}, it, it, \_]\ [A'_{k,0}, it, it, \_]$
3. $[\bigtriangledown_{k,i}, it, bp, \_] \longmapsto [A'_{k,i+1}, it, it, \_]\ [\bigtriangledown_{k,i}, it, bp, \_]$
4. $[\bigtriangledown_{k,n_k}, it, bp, \_]\ [A'_{k,0}, bp, bp, \_] \longmapsto [A''_{k,0}, it, bp, \_]$
5. $[A''_{k,i+1}, it, bp, \_]\ [\bigtriangledown_{k,i}, bp, r, \_] \longmapsto [\bigtriangledown_{k,i+1}, it, r, \_]$

that we informally explain as:

1. States the axiom $A_{0,0}$, which we represent by $A'_{0,0}$.
2. Requires the recognition of $A'_{k,0}$, which we represent by $\bigtriangledown_{k,0}$.
3. Selects the leftmost unrecognized category, $A'_{k,i+1}$.
4. The body of γ_k has been recognized. We push $A''_{k,0}$ to show that $A'_{k,0}$ has been recognized.
5. After recognition of $A''_{k,i+1}$, the parse advances to the next term in γ_k.

For top-down architectures	*For bottom-up architectures*
$A_{k,0} :- \boxed{A_{k,1}, A_{k,2}, \cdots, A_{k,i}} A_{k,i+1}, \cdots, A_{k,n_k}.$	$A_{k,0} :- A_{k,1}, A_{k,2}, \cdots, A_{k,i}, \boxed{A_{k,i+1}, \cdots, A_{k,n_k}}.$
... the first i categories have been recognized.	*... categories after the i^{th} position have been recognized*

Figure 1: The meaning of symbols $\bigtriangledown_{k,i}$

3.3 A bottom-up strategy with static control

We introduce the ICE parser included in GALENA. Here, a symbol $\bigtriangledown_{k,i}$ expresses that the categories in the right-hand-side of γ_k after the i position have already been recognized, as it is shown in Fig. 1. The set of transitions defines a generalized LALR(1) automaton as follows:

$$
\begin{array}{lll}
1. & [A_{k,n_k}, it, bp, st] & \longmapsto [\bigtriangledown_{k,n_k}, it, it, st]\ [A_{k,n_k}, it, bp, st] \\
 & & \{\text{action}(st, \text{token}_{it}) = \text{reduce}(\gamma_k)\} \\
2. & [\bigtriangledown_{k,i}, it, r, st_1] & \\
 & [A_{k,i}, r, bp, st_1] & \longmapsto [\bigtriangledown_{k,i-1}, it, bp, st_2] \\
 & & \{\text{action}(st_2, \text{token}_{it}) = \text{shift}(st_1)\},\ i \in [1, n_k] \\
3. & [\bigtriangledown_{k,0}, it, bp, st_1] & \longmapsto [A_{k,0}, it, bp, st_2] \\
 & & \{\text{goto}(st_1, A_{k,0}) = st_2\} \\
4. & [A_{k,i}, it, bp, st_1] & \longmapsto [A_{k,i+1}, it+1, it, st_2]\ [A_{k,i}, it, bp, st_1] \\
 & & \{\text{action}(st_1, A_{k,i+1}) = \text{shift}(st_2)\},\ i \in [0, n_k) \\
5. & [A_{k,i}, it, bp, st_1] & \longmapsto [A_{l,0}, it+1, it, st_2]\ [A_{k,i}, it, bp, st_1] \\
 & & \{\text{action}(st_1, A_{l,0}) = \text{shift}(st_2)\} \\
6. & [\$, 0, 0, 0] & \longmapsto [A_{0,0}, 0, 0, st]\ [\$, 0, 0, 0] \\
 & & \{\text{action}(0, \text{token}_0) = \text{shift}(st)\}
\end{array}
$$

where *action*, *goto*, *shift* and *reduce* are well-known concepts in LR parsing [1]. We interpret these transitions as follows:

1. Pushes $\bigtriangledown_{k,n_k}$ to indicate that the body of γ_k is to be recognized.
2. The parser advances after the refutation of $\mathrm{A}_{k,i}$.
3. All literals in the body of γ_k have been recognized, and therefore $A_{k,0}$ can also be recognized.
4. Pushes the literal $A_{k,i+1}$, assuming that it will be needed for the recognition.
5. Begins with the recognition of γ_k.
6. States the axiom $A_{0,0}$.

4 On-line parsers

We now go deep into the influence of dynamic programming when considering more complex grammatical formalisms. To do so, we take the DCG extensions of the context-free parsing strategies previously tested. To be more precise, we shall consider the AGFL [6] environment as representative of a typical top-down evaluator, an implementation of Earley's deduction scheme [8], and finally the GALENA [13] system using a simple unification oriented extension of ICE [12]. We have chosen GALENA as representative of the family of LR-*like* inference environments [7, 10].

Previous to introducing the experimental results, we locate each architecture in our unified framework. Here, we deal with clauses γ_k of the form $A_{k,0} :- A_{k,1}, \ldots, A_{k,n_k}$, where $A_{k,i}$ are now logical terms. We introduce the vector $\vec{T}_k$ of the variables occurring in γ_k, and the predicate symbol $\bigtriangledown_{k,i}$. Besides the positional meaning of this predicate symbol, which is dependent on each particular evaluation strategy, instances of $\bigtriangledown_{k,i}(\vec{T}_k)$ serve as temporary information storage structures for remaining subgoals during the evaluation.

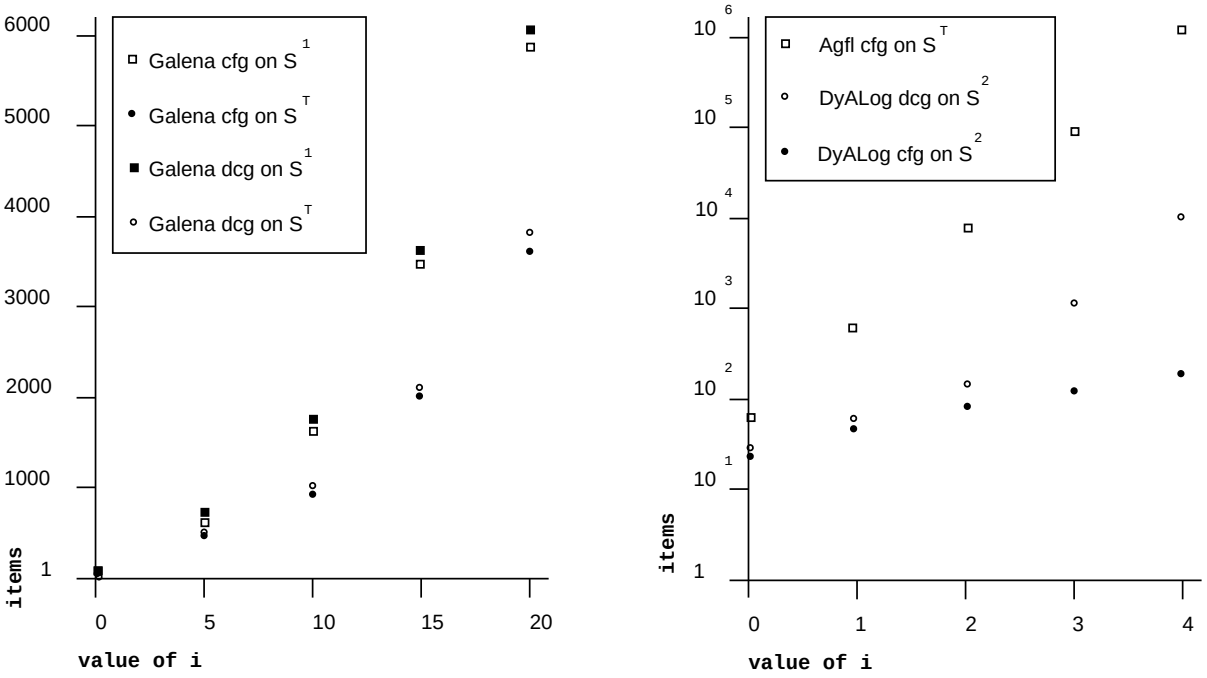

Figure 2: Experimental results with GALENA and pure top-down parsing

Intuitively, the interpretation of the following evaluation schema is analogous to those considered for off-line parsers, replacing the notions of matching, recognition, axiom, category and rule by unification, refutation, query, goal and clause; respectively.

4.1 A top-down architecture

We begin with the simulation of the top-down strategy, which is given by the transitions:

1. $[\$, 0, 0, \_] \longmapsto [\bigtriangledown_{0,0}(\vec{T}_0), 0, 0, \_]\ \$$
2. $[\bigtriangledown_{k,i}(\vec{T}_k), it, bp, \_] \longmapsto [A_{k,i+1}, it, it, \_]\ [\bigtriangledown_{k,i}(\vec{T}_k), it, bp, \_]$
3. $[A_{k,0}, it, bp, \_] \longmapsto [\bigtriangledown_{k,0}(\vec{T}_k), it, bp, \_]$
4. $[\bigtriangledown_{k,n_k}(\vec{T}_k), it, bp, \_]\ [\bigtriangledown_{k',i}(\vec{T}_{k'}), bp, r, \_] \longmapsto [\bigtriangledown_{k',i+1}(\vec{T}_{k'}), it, r, \_]$

where an instance of $\bigtriangledown_{k,i}(\vec{T}_k)$ indicates that all literals until the i^{th} literal in the body of γ_k have been proved.

4.2 Earley's approach

In the case of Earley's deduction, the set of transitions is now given by:

1. $[\$, 0, 0, \_] \longmapsto [A'_{0,0}, 0, 0, \_]\ \$$
2. $[A'_{k,0}, it, it, \_] \longmapsto [\bigtriangledown_{k,0}(\vec{T}_k), it, it, \_]\ [A'_{k,0}, it, it, \_]$
3. $[\bigtriangledown_{k,i}(\vec{T}_k), it, bp, \_] \longmapsto [A'_{k,i+1}, it, it, \_]\ [\bigtriangledown_{k,i}(\vec{T}_k), it, bp, \_]$
4. $[\bigtriangledown_{k,n_k}(\vec{T}_k), it, bp, \_]\ [A'_{k,0}, bp, bp, \_] \longmapsto [A''_{k,0}, it, bp, \_]$
5. $[A''_{k,i+1}, it, bp, \_]\ [\bigtriangledown_{k,i}(\vec{T}_k), bp, r, \_] \longmapsto [\bigtriangledown_{k,i+1}(\vec{T}_k), it, r, \_]$

with the same interpretation for instances of $\bigtriangledown_{k,i}(\vec{T}_k)$ as in the preceding case. The meaning of atoms $A'_{k,i}$ and $A''_{k,i}$ is analogous to the context-free case. They respectively indicate that the term $A_{k,i}$ in the DCG is yet to be proved or it has already been proved.

4.3 A bottom-up strategy with static control

Finally, we introduce the evaluation scheme of GALENA, given by the transitions:

1. $[A_{k,n_k}, it, bp, st] \longmapsto [\bigtriangledown_{k,n_k}(\vec{T}_k), it, it, st]\ [A_{k,n_k}, it, bp, st]$
   $\{\text{action}(st, \text{token}_{it}) = \text{reduce}(\gamma_k)\}$
2. $[\bigtriangledown_{k,i}(\vec{T}_k), it, r, st_1]$
   $[A_{k,i}, r, bp, st_1] \longmapsto [\bigtriangledown_{k,i-1}(\vec{T}_k), it, bp, st_2]$
   $\{\text{action}(st_2, \text{token}_{it}) = \text{shift}(st_1)\},\ i \in [1, n_k]$
3. $[\bigtriangledown_{k,0}(\vec{T}_k), it, bp, st_1] \longmapsto [A_{k,0}, it, bp, st_2]$
   $\{\text{goto}(st_1, A_{k,0}) = st_2\}$
4. $[A_{k,i}, it, bp, st_1] \longmapsto [A_{k,i+1}, it+1, it, st_2]\ [A_{k,i}, it, bp, st_1]$
   $\{\text{action}(st_1, A_{k,i+1}) = \text{shift}(st_2)\},\ i \in [0, n_k)$
5. $[A_{k,i}, it, bp, st_1] \longmapsto [A_{l,0}, it+1, it, st_2]\ [A_{k,i}, it, bp, st_1]$
   $\{\text{action}(st_1, A_{l,0}) = \text{shift}(st_2)\}$
6. $[\$, 0, 0, 0] \longmapsto [A_{k,0}, 0, 0, st]\ [\$, 0, 0, 0]$
   $\{\text{action}(0, \text{token}_0) = \text{shift}(st)\}$

Control conditions are built from actions in a driver given by a LALR(1) automaton built from the context-free skeleton of the DCG. This skeleton is obtained from the original program by keeping only functors in the clauses in order to obtain terminals from the extensional database, and variables from heads in the intensional one, taking into account the arity. The idea is to use the driver to cut out evaluation conflicts arising from coincidence of logical terms signatures during the proof process.

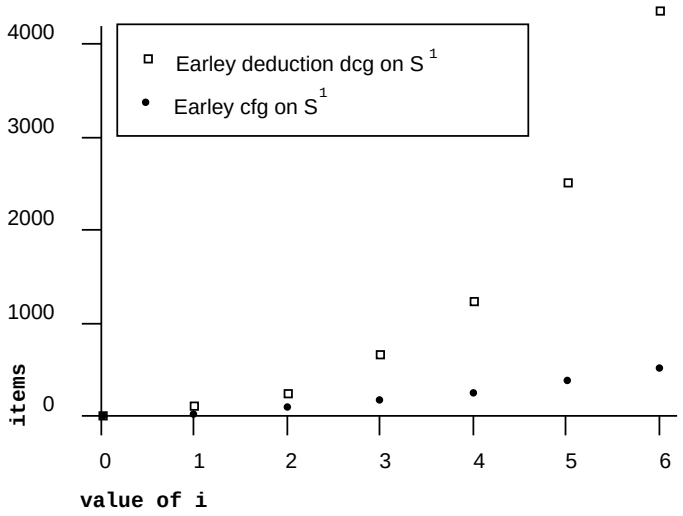

Figure 3: Experimental results using Earley-*like* strategies

5 Experimental results

To show how dynamic programming improves the efficiency of compilation schema, we shall focus on two aspects: the number of computations as contrasted with classic backtracking techniques, and a simple comparison between different parsing strategies. Although this work has been supported by a lot of different tests, we have considered an only example to illustrate our discussion, in order to facilitate understanding. We use the syntax of a simple subset of nominal sentences in English. Henceforth, we take as our guideline example the following DCG:

$$
\begin{array}{lcl}
s(esp(Tree)) & :- & sentence(Tree). \\
 & & \\
sentence(phr(Tree_1, Tree_2)) & :- & np(Tree_1, Nmbr),\ vp(Tree_2, Nmbr). \\
sentence(phr(Tree_1, Tree_2)) & :- & sentence(Tree_1),\ pp(Tree_2). \\
 & & \\
np(np(s(Wrd)), Nmbr) & :- & noun(Wrd : word, Nmbr : number). \\
np(pr(Wrd), Nmbr) & :- & pronoun(Wrd : word, Nmbr : number). \\
np(np(det(Wrd_1), s(Wrd_2)), Nmbr) & :- & determiner(Wrd_1 : word, Nmbr : number, Gndr : gender), \\
 & & noun(Wrd_2 : word, Nmbr : number, Gndr : gender). \\
np(np(Tree_1, Tree_2), Nmbr) & :- & np(Tree_1, Nmbr),\ pp(Tree_2). \\
 & & \\
pp(pp(prep(Wrd), Tree)) & :- & preposition(Wrd : word),\ np(Tree, Nmbr). \\
 & & \\
vp(vp(verb(Wrd), Tree), Nmbr) & :- & verb(Wrd : word, Nmbr : number),\ np(Tree, Nmbr).
\end{array}
$$

To simplify the explanation, we assume that lexical information is directly recovered from a specialized tagger by the name of the corresponding attribute, which allows us to focus on syntactic phenomena. So, the third clause in the definition of the predicate *np* instances the variable Wrd_1 to the value of the current input token by using the syntax $Wrd_1 : word$. Similarly, we recover the number and the gender, by using respectively the key-words *number* and *gender*. The context-free skeleton is defined by the rules:

S	→	*Sentence*	*Sentence*	→	*Np Vp*	*Sentence*	→	*Sentence Pp*
Np	→	*noun*	*Np*	→	*pronoun*	*Np*	→	*determiner noun*
Np	→	*Np Pp*	*Pp*	→	*preposition Np*	*Vp*	→	*verb Np*

We have chosen input strings of the form

$$\text{I see a father (of a son of a father)}^i$$

where $i \geq 0$ is the number of repetitions of the substring *"of a son of a father"*. Here, the number of different parses C_i increases exponentially with i. This number is:

$$C_0 = C_1 = 1 \quad \text{and} \quad C_i = \left(\begin{array}{c} 2i \\ i \end{array} \right) \frac{1}{i+1},\ \text{if } i > 1$$

allowing us to study the compilation schema when highly redundant computations appears.

Tests have been performed on dynamic frames S^T with AGFL, Earley and ICE, S^2 with AGFL and ICE, and S^1 with Earley and ICE. Tests on S^2 and S^T have no interest for Earley, given that the classic algorithm is naturally described in S^1. For AGFL, tests on S^1 are out of place due to its top-down architecture that thwarts completeness, and tests on S^2 have been obtained from an alternative adaptation of the parsing scheme based on DYALOG [2], since the original tool does not work in dynamic programming.

Finally, tests on S^T for AGFL in the DCG case have been obtained excluding all functional symbols since this facility is not available in AGFL. To avoid distortion of the results, our testing grammar has been chosen in such a way that the number of S^T items in the CFG and the corresponding DCG is the same. In effect, the absence of clauses with a common context-free skeleton, and the fact that in our example functional symbols are exclusively used in a constructive sense, permits us to attain this goal. Results on S^2 for AGFL, using our alternative implementation, include functional symbols.

Results related to GALENA and AGFL are shown in Fig. 2. In the case of Earley's algorithm results are shown in Fig. 3. In all cases, dynamic programming highlights

a better computational behavior with respect to classic approaches based on S^T. In addition, a simple view shows that the best results correspond to GALENA, a mixed-strategy with static predictions, over Earley and classic top-down parsing.

Tests using dynamic programming, both for the parsing of the sole context-free backbone and whole DCG, are shown in Fig. 4. The natural dynamic frame of each parsing model was used to obtain all the figures. One again the benefit due to GALENA's mixed-strategy is shown. By looking at both figures we can also realize that in the case of GALENA, the lowest increment was achieved in the number of generated items when going from CFG to DCG. In particular, results on DCGs proves the efficiency of GALENA to cut out unification conflicts during resolution. Finally, the apparently bad behavior of GALENA in these tests cannot be extended to other CFGs. In effect, they are a natural consequence of the reduced number of prediction computations due to the structure of our grammar, as it is claimed in [12].

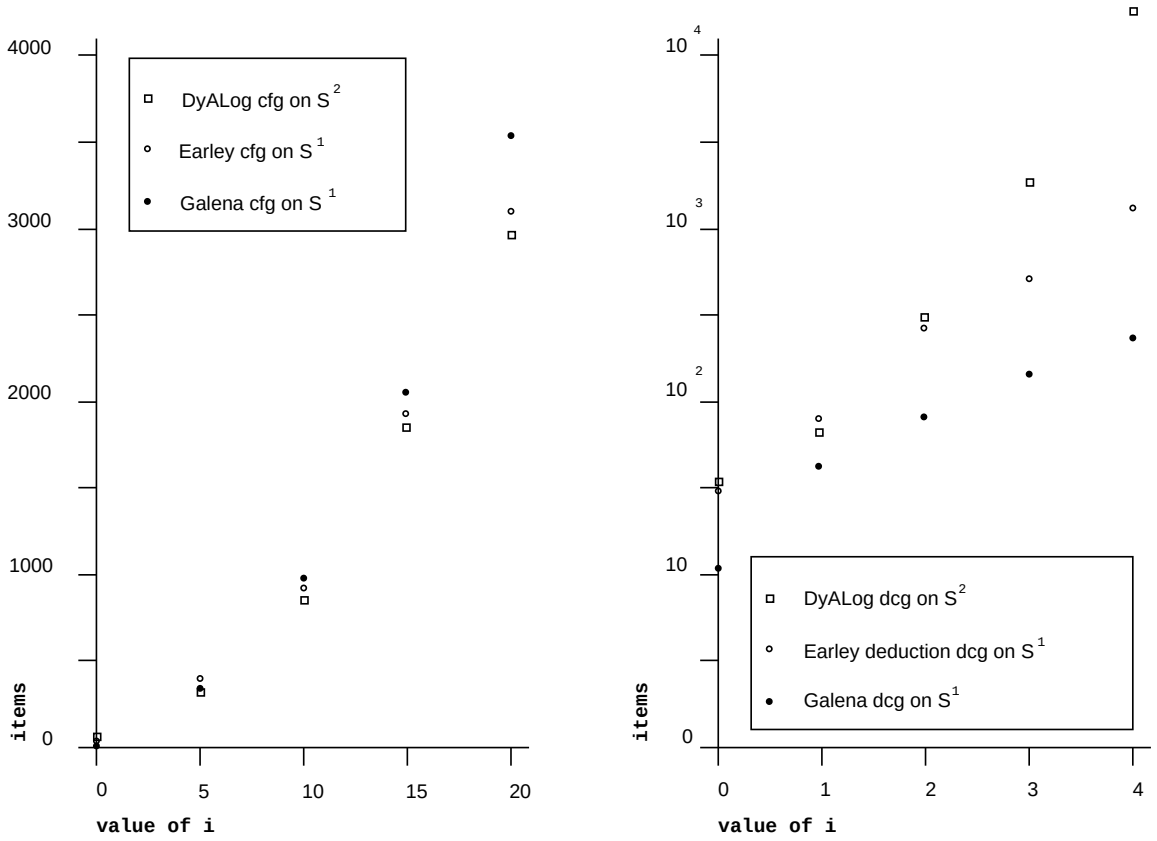

Figure 4: Comparing strategies for CFGs and DCGs

6 Conclusion

This paper reviews the fundamental computational properties of some of the best-known compilation schema in the continuum of Horn-like formalisms. We have exemplified our discussion in a well-known application domain, NLP, using both off-line and on-line approaches and considering dynamic programming as an alternative to obtain greater efficiency in practical systems.

We have considered a uniform formal framework, introducing a unique operational model which allows us to encompass all the parsing architectures considered. In

particular, we have identified the notion of item as a computation unit, which is manipulated by a set of transitions that can be divided into three classes. In this manner, the interpretation of the practical tests on this basis ensures the validity of the work presented.

As a secondary outcome, this paper seems to corroborate the superiority, often argued, of bottom-up parsers including some kind of static prediction facility, over classic top-down algorithms or bottom-up strategies with dynamic predictions. Here, it is important to remark upon the importance of an efficient indexation tool, capable of significantly reducing the search space in dynamic programming. In effect, the performance shown by top-down architecture compared to bottom-up ones in practical NLP environments is often due to the efficiency of backtracking in managing this search space. In our case, the use of itemsets permits us to limit this difference in the bottom-up case, thereby increasing the efficiency.

References

[1] A.V. Aho and J.D. Ullman. *The Theory of Parsing, Translation and Compiling*, volume 1-2. Prentice-Hall, Englewood Cliff, New Jersey, U.S.A., 1973.

[2] E. de la Clergerie. *Automates à Piles et Programmation Dynamique*. PhD thesis, University of Paris VII, France, 1993.

[3] J. Earley. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102, 1970.

[4] J. Heering, P.R.H. Hendriks, P. Klint, and J. Rekers. The syntax definition formalism SDF - reference manual. *SIGPLAN Notices*, 24(11):43–75, 1989.

[5] B. Lang. Towards a uniform formal framework for parsing. In M. Tomita, editor, *Current Issues in Parsing Technology*, pages 153–171. Kluwer Academic Publishers, 1991.

[6] H. Meijer. The project on extended affix grammars at Nijmegen. *Attribute Grammars and their Applications, SLNC*, 461:130–142, 1990.

[7] U. Nilsson. AID: An alternative implementation of DCGs. *New Generation Computing*, 4:383–399, 1986.

[8] F.C.N. Pereira and D.H.D. Warren. Parsing as deduction. In *Proc. of the* 21^{st} *Annual Metting of the Association for Computational Linguistics*, pages 137–144, Cambridge, Massachusetts, U.S.A., 1983.

[9] J. Rekers. *Parser Generation for Interactive Environments*. PhD thesis, University of Amsterdam, Amsterdam, The Netherlands, 1992.

[10] D.A. Rosenblueth and J.C. Peralta. LR inference: Inference systems for fixed-mode logic programs, based on LR parsing. In *International Logic Programming Symposium*, pages 439–453, The MIT Press, Cambridge Massachussets 02142 USA, 1994.

[11] M. Tomita. *Efficient Parsing for Natural Language. A Fast Algorithm for Practical Systems*. Kluwer Academic Publishers, Norwell, Massachusetts, U.S.A., 1986.

[12] M. Vilares. *Efficient Incremental Parsing for Context-Free Languages*. PhD thesis, University of Nice. ISBN 2-7261-0768-0, France, 1992.

[13] M. Vilares and M.A. Alonso. An LALR extension of DCGs in dynamic programming. In C. Martín Vide, editor, *Mathematical Linguistics*, volume 2, Amsterdam, The Netherlands, 1998. John Benjamins Publishing Company.