Dynamic Programming as Frame for Efficient Parsing

Manuel Vilares Ferro
Universidad de La Coruña
Departamento de Computación
Campus de Elviña s/n, 15071 La Coruña, Spain
vilares@dc.fi.udc.es

David Cabrero Souto
Centro Ramón Piñeiro para a Investigación en Humanidades
Estrada Santiago-Noia km 3, A Barcia
15896 Santiago de Compostela, Spain
dcabrero@cirp.es

Miguel A. Alonso Pardo
Universidad de La Coruña
Departamento de Computación
Campus de Elviña s/n, 15071 La Coruña, Spain
alonso@dc.fi.udc.es

Abstract

The last few years have seen a renewal of interest in the consideration of dynamic programming in compiler technology. This is due to the compactness of the representations, which are turning this paradigm into a common way of dealing with highly redundant computations occurring, for instance, in natural language processing, logic programming or abstract interpretation and related to phenomena such as non-determinism or domain ordering. However, it is not usual to find practical studies about what the real interest of these techniques is, and which approaches are better adapted in each case.

We justify the consideration of dynamic programming, both in definite clause and context-free grammar parsing, highlighting the parallelism between the conclusions reached in both cases. We focus on the computational properties, suggesting a simple decision guide for the reader interested in applying this technology.

1. Introduction

Highly redundant computations are usual when we deal with complex grammar formalisms. This claim has been used to motivate parsing techniques that encode trees and computations in some kind of shared structure. Major areas of application are natural language processing (NLP), where dynamic programming has been known for a long time [3], and logic programming. Here, dynamic programming was initially been proposed [8] to address shortcomings in classic PROLOG evaluation, whose poor termination and complexity properties have often rendered it unsuitable for practical purposes. However, although in theory these strategies give an exponential reduction in complexity over traditional parsing and resolution techniques [5], the lack of a comparison frame for these proposals often leads us to describe our compilation schema on the basis of naive approaches with backtracking.

Our aim is to illustrate the practical suitability of dynamic programming to deal with grammatical formalisms in the Horn-like continuum that goes from context-free languages to Horn clause logic. To obtain a uniform understanding of the computational behavior of syntactic phenomena, we establish a formal framework for parsing. We choose to work in the context of the *logical push-down automaton* (LPDA) notion [5], an operational model generalizing the classic concept of push-down automaton to full first order Horn logic, that allows all parsing strategies to be expressed by a unique device.

We seek to cast some light on which compilation

schema, and which dynamic programming approach, are suitable in each case. The problem is posed in the context of NLP, as representative of a domain embracing all parsing formalisms in the continuum considered. In effect, parsers in NLP are usually classified in two categories [8]:

- Off-line approaches producing some kind of preliminary grammatical skeleton, typically a context-free grammar (CFG), from which the system must filter out the undesirable analysis taking into account the finer language structure.
- On-line systems applying linguistic restrictions at parsing time, as is the case of definite clause grammars (DCGs).

In both cases, the sharing of computations affects the performance. So, it may allow a better factorization to filter parse trees [11] or fully exploit the declarative power of the formalism by dealing with a possible infinite number of structures [15].

In section 2 of this paper we establish our formal testing framework. Sections 3 and 4, respectively give a survey of some representative parsing strategies in both cases, off-line and on-line, locating them in our testing framework. In section 5, we include the experimental results. Finally, section 6 is a conclusion about the work presented.

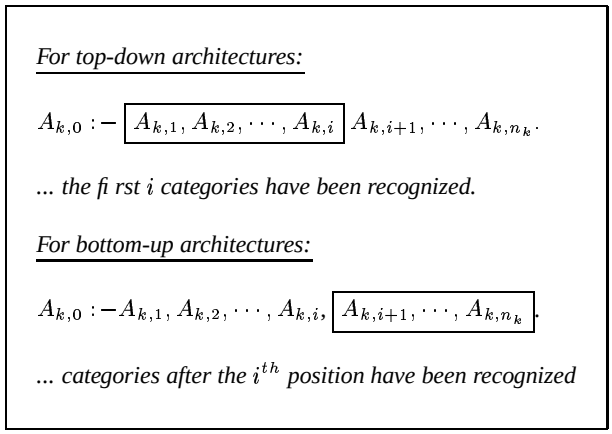

Figure 1. The meaning of symbols $\bigtriangledown_{k,i}$

2. A common framework

We introduce a common parsing device for testing, extending the original notion of LPDA [5]. We do it as a generalization of the operational model described by the authors in [14]. An LPDA is now 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 a 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. In order to deal with context-free grammars, 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* and adapting transitions in order to deal with these items [13, 2]. When the correctness, completeness and compatibility of computations in this new frame are assured in relation to classic push-down devices, we talk about the concept of *dynamic frame*. Here, the use of it allows us to index the parse in order to disregard the computational work of the lexical analyzer by separating this task from parse, focusing on syntactic phenomena and reducing the search. This relies on the concept of *itemset* [3], associating 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. We shall use bp to chain pop transitions.

Usually, only two dynamic frames are of practical interest, S^2 and S^1, where the superscript denotes the number of top stack elements used to generate items. The standard dynamic frame, where a stack is given by all its components, and backtracking is the technique to simulate nondeterminism, is denoted by S^T. Correctness and complete-

ness of S^2 in relation to S^T is trivial given that transitions in our formal model 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 a *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 $\sigma = \text{mgu}(A, B)$.

Although S^1 provides the best sharing properties, it can be proved that correctness is only guaranteed for weakly predictive parsing strategies [13, 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 [14] 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 [13] 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.

[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.

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. 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:

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

Figure 2. Guideline grammar

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.

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:

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. $[\$, 0, 0, 0] \longmapsto [A_{0,0}, 0, 0, st]$
   $[\$, 0, 0, 0]$
   $\{\text{action}(0, \text{token}_0) = \text{shift}(st)\}$

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 $A_{k,i}$.
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 [14] system using a simple unification oriented extension of ICE [13]. 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.

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.

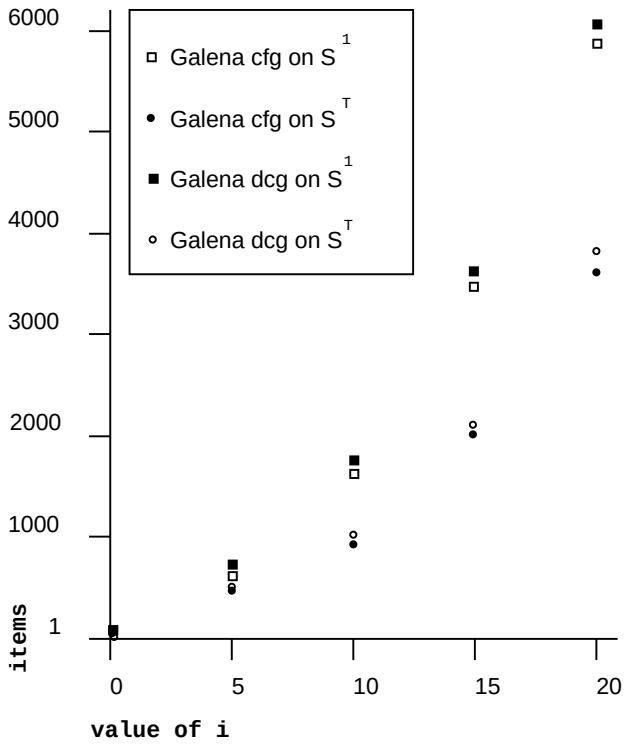

Figure 3. Tests with GALENA

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.

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 DCG shown in Fig. 2.

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	$\rightarrow$	*Sentence*
Sentence	$\rightarrow$	*Np Vp*
Sentence	$\rightarrow$	*Sentence Pp*
Np	$\rightarrow$	*noun*
Np	$\rightarrow$	*pronoun*
Np	$\rightarrow$	*determiner noun*
Np	$\rightarrow$	*Np Pp*
Pp	$\rightarrow$	*preposition Np*
Vp	$\rightarrow$	*verb Np*

We have chosen input strings of the form

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

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:

$$C_0 = C_1 = 1 \quad \text{and} \quad C_i = \binom{2i}{i} \frac{1}{i+1}, \text{ if } i > 1$$

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

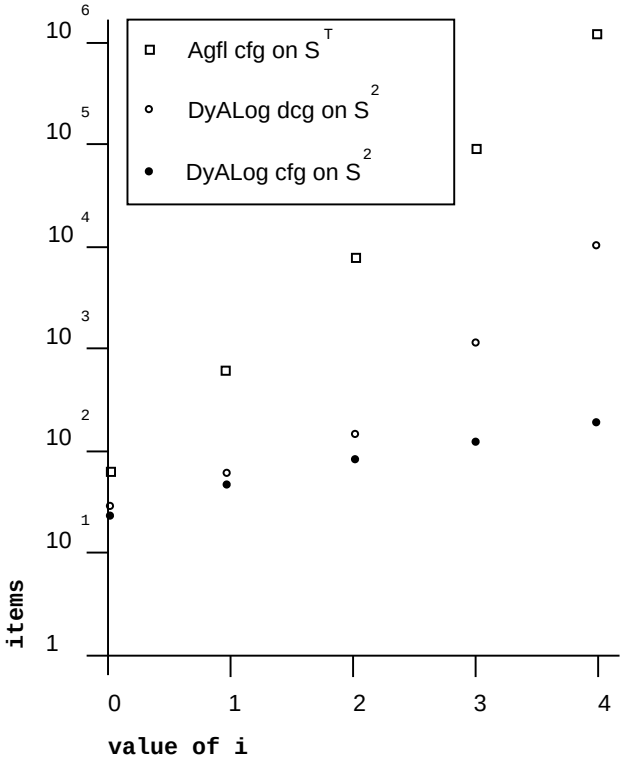

Figure 4. Tests using pure top-down parsing

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.

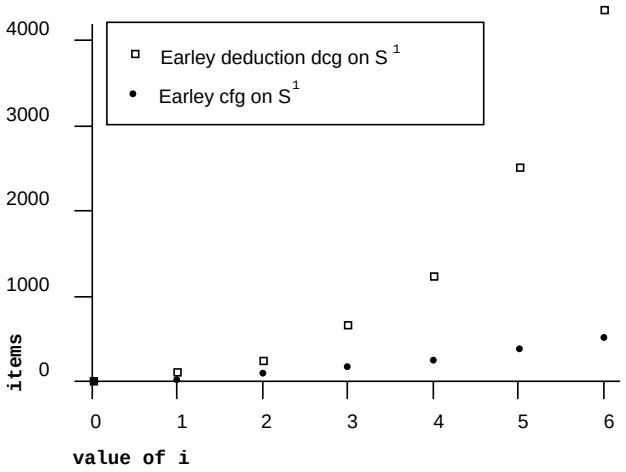

Figure 5. Tests using Earley-like strategies

Results related to GALENA and AGFL are shown in Fig. 3 and Fig. 4 respectively. In the case of Earley's algorithm results are shown in Fig. 5. 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. 6 and Fig. 7 respectively. 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 [13].

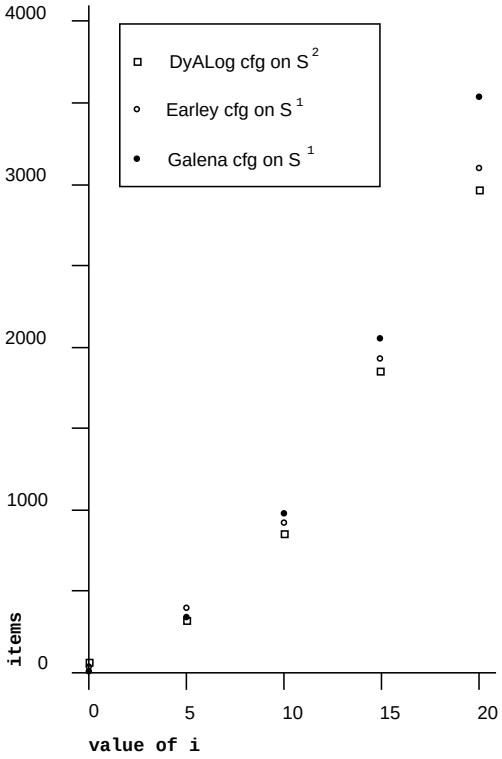

Figure 6. Comparing strategies for CFGs

6. Conclusion

This paper reviews, in a formal framework encompassing all architectures considered, the computational properties of the some of best-known parsing schema in the continuum of Horn-*like* formalisms. We have exemplified the problem of computational sharing in a frame where nondeterminism is usual, showing the practical interest of dynamic programming regardless of the parsing scheme chosen.

As a secondary outcome, we corroborate the superiority, often claimed but never proved, of bottom-up parsers over top-down and mixed strategies. Frequently the performance shown by top-down architecture in practical systems is due to the efficiency of backtracking to manage the search space. In our case, the use of indexing permits us to limit this difference in the bottom-up case, thus increasing the efficiency.

Finally, the parallelism existing between the results on the practical use of dynamic programming for both extremes in the continuum considered, leads us to argue the extension of our conclusions to other formalisms in the same continuum, such as middle-sensitive grammars [12].

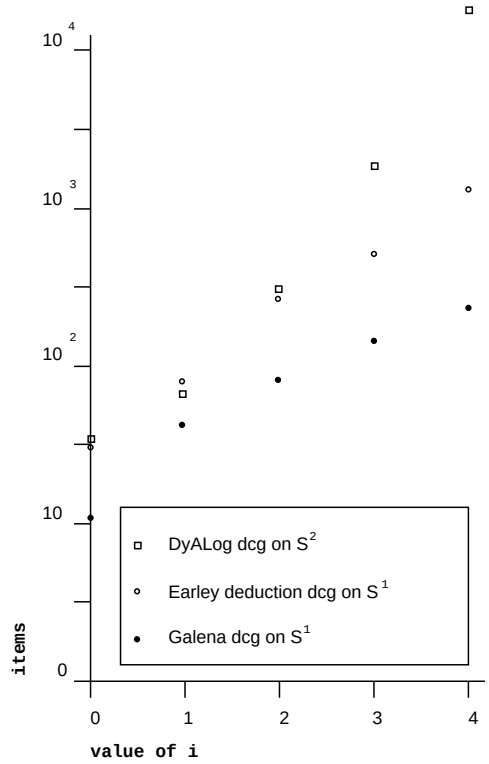

Figure 7. Comparing strategies for DCGs

7. Acknowledgments

This work has been partially supported by the FEDER of European Union (1FD97-0047-C04-02), Government of Spain (HF97-223) and Xunta de Galicia (XUGA10505B96 and XUGA20402B97).

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] K. Vijay-Shankar and D.J. Weir. The equivalence of four extensions of context-free grammars. *Mathematical Systems Theory*, 27:511–545, 1994.

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

[14] M. Vilares and M.A. Alonso. An LALR extension of DCGs in dynamic programming. In C. Martín Vide, editor, *Mathematical and Computational Analysis of Natural Language*, volume 45, pages 267–278. John Benjamins Publishing Company, Amsterdam, The Netherlands, 1998.

[15] M. Vilares, M.A. Alonso, and D. Cabrero. An operational model for parsing fixed-mode DCGs. In *Proc. of LACL'97. Int. Conf on Logical Aspects on Computational Linguistics*, pages 61–64, Nancy, France, 1997.