Dynamic Programming of Partial Parses

David Cabrero Souto[2], Jesús Vilares Ferro[1], and Manuel Vilares Ferro[1]

[1] Universidad de A Coruña
Departamento de Computación
Campus de Elviña s/n, 15071 A Coruña, Spain
vilares@dc.fi.udc.es, jvilares@mail2.udc.es
[2] Universidad de Vigo
Area de Ciencias de la Computación e Inteligencia Artificial
Edificio Politécnico
32004 Ourense, Spain
cabrero@uvigo.es

Abstract. The last years have seen a renewal of interest in applying dynamic programming to natural language processing. The main advantage is the compactness of the representations, which is turning this paradigm into a common way of dealing with highly redundant computations related to phenomena such as non-determinism.
Natural language parsing adds another challenge, since grammatical information is often insufficient. We describe an extension of parsing techniques for partial parsing in dynamic programming. Our aim is to obtain as much information as possible, that is incomplete parses, while preserving compactness of the representations.
Keywords: Partial parsing, dynamic programming, deductive parsing scheme.

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. A major area of application is natural language processing (NLP), where dynamic programming has been known for a long time [3]. In particular, natural language parsing comes across the problem of partial information. This lack of information is due to the errors in former stages of analysis and the fact that practical grammars and lexicons are incomplete and even incorrect.

We refer to standard parsing as *complete parsing*, reserving the term *partial parsing* for all the possible subcomputations of a complete parsing. Our aim is to obtain every correct partial parse even when there is no complete parse.

Previous studies have illustrated the practical suitability of dynamic programming for dealing with context-free grammars (CFGs) [11], middle-sensitive grammars [1] and definite clause grammars [10]. Our goal is to show the validity of these results for partial parsing. We approach the problem by extending models from complete to partial parsing while preserving the related benefits of dynamic programming.

Often, these techniques improve performance by pruning the search space by means of the inclusion of static control. Unfortunately, while dealing with partial parses, the

static control will prune some analysis branches that are useless in a complete parse, but which are necessary in some partial parses. To overcome this problem we deal with the starting and finishing points of a partial parse like any other source of non determinism. The static control is modified to take them into account.

In contrast to previous works in the domain of partial parsing, our proposal introduces a parsing framework based on the notion of deduction scheme [7]. This clearly differentiates our work from approaches oriented to particular parsing architectures [5, 9, 6], and provides a uniform description and operational formalism to validate the performance in each case.

In Section 2 of this paper we introduce a uniform parsing framework, describing different schema for both complete and partial parsing. These schema include classic top-down and bottom-up approaches, but also mixed strategies with dynamic and static control. Section 3 gives a survey of the dynamic interpretation process. In Section 4 we compare in practice the schema introduced, with preliminary experimental results. Finally, Section 5 is a conclusion about the work presented.

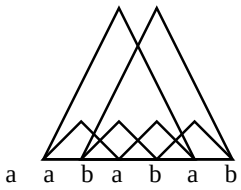

Fig. 1. Partial parses of `aababab`

2 On partial parsing

In order to obtain a practical definition of partial parsing, we must relax the conditions we apply to the standard concept. The notion of grammar includes an initial symbol or axiom, S, leading to the parse of all sentences of the language generated by the grammar. Instead of this, we use a set of initial symbols, $\mathcal{S}$, following the *entry point* concept [4], a classic auxiliary structure in the abstract syntax that enables the parsing of program fragments. We introduce a CFG as a 4-tuple $\mathcal{G} = (N, \Sigma, P, \mathcal{S})$, where N is a finite set of non-terminal categories, Σ is a finite alphabet and P is a finite set of context-free rules. As mentioned, $\mathcal{S}$ is the set of initial symbols leading to complete or partial parses. We assume $\mathcal{L}(\mathcal{G})$ to be the language generated by $\mathcal{G}$, and we try to determine the partial parses of an input string $w_{1..n}$, of length n.

In particular, we discuss the extension to the partial case of classical context-free parsing methods including pure top-down and pure bottom-up architectures, Earley's algorithm [3] as representative of a mixed-strategy with dynamic prediction and a LR(1) proposal as representative of a mixed-strategy with static prediction.

For the sake of a better exposition we have chosen a common descriptive framework, the deductive parsing scheme [7], close to the parsing schemata proposal [8].

The deduction system consists of a set of *items* representing parsing states and a set of deduction steps performing over those items.

As our running example we shall consider the language, $\mathcal{P}$, of palindromes over the alphabet $\Sigma = \{a, b\}$, generated by the grammar that follows:

$$\begin{array}{ll} \mathit{Palin} \rightarrow a & \mathit{Palin} \rightarrow b \\ \mathit{Palin} \rightarrow a\ \mathit{Palin}\ a & \mathit{Palin} \rightarrow b\ \mathit{Palin}\ b \end{array}$$

Observe, for example, that although the input string is $aababab \notin \mathcal{P}$, it contains substrings that do belong to the language, as is the case of the trees in Fig. 1.

2.1 A top-down scheme

The scheme for top-down complete parsing is shown in Fig. 2. We have a single *axiom* that predicts the analysis of the initial symbol, and a single *goal* that represents the parse of the complete input sentence. Now we will make a short description of the scheme. Each item has the form: $[\bullet\beta, j, (T)]$, stating that we have constructed the derivation $S \stackrel{*}{\Rightarrow} w_1 \cdots w_j\beta$, and T is the parse tree. The dot is a reference to position j in the input string. The parse has reached this position and has to continue from there. Soundness and correctness are proved in [7].

Item form	$[\bullet\beta, j, T]$
Invariant	$S \stackrel{*}{\Rightarrow} w_1 \cdots w_j\beta$
Axioms	$[\bullet S, 0, ()]$
Goals	$[\bullet, n, T]$
Scanning	$\dfrac{[\bullet w_{j+1}\beta, j, T_1\text{tree}(B, \alpha \bullet w_{j+1}\beta)T_2]}{[\bullet\beta, j+1, T_1\text{tree}(B, \alpha w_{j+1} \bullet \beta)T_2)]}$
Prediction	$\dfrac{[\bullet B\beta, j, T_1\text{tree}(A, \alpha \bullet B\beta T_2)]}{[\bullet\gamma\beta, j, T_1\,\text{tree}(A, \alpha\text{tree}(B, \bullet\gamma)\beta)T_2]}\ \langle r_k : B \rightarrow \gamma \in R\rangle$

Fig. 2. Top-down schema

The parse starts at position 0 and symbol S, yielding axiom item $[\bullet S, 0, ()]$. We then apply the following deduction steps:

Scanning: It moves the point one position forward. This rule is obtained after observing that items $[\bullet w_{j+1}\beta, j, (T)]$ and $[\bullet\beta, j+1, (Tw_{j+1})]$ represent the same state in the derivation $S \stackrel{*}{\Rightarrow} w_1 \cdots w_{j+1}\beta$.

Prediction: It takes the next non terminal symbol to parse (B) and rewrites it as the right hand side of a matching rule ($r_k : B \rightarrow \gamma$). It predicts the use of the rule r_k.

Item form	$[\alpha\bullet, j, T]$
Invariant	$\alpha w_{j+1} \cdots w_n \stackrel{*}{\Rightarrow} w_1 \cdots w_n$
Axioms	$[\bullet, 0, ()]$
Goals	$[S\bullet, n, T]$
Scanning	$\dfrac{[\alpha\bullet, j, T]}{[\alpha w_{j+1}\bullet, j+1, T w_{j+1}]}$
Completion	$\dfrac{[\alpha\gamma\bullet, j, T_1 T_2]}{[\alpha B\bullet, j, T_1 \text{tree}(B, T_2)]} \left\langle \begin{array}{l} r_k : B \to \gamma \in R, \\ \|\gamma\| = \|T_2\| \end{array} \right\rangle$

Fig. 3. Bottom-up schema

Finally, $w \in \mathcal{L}(\mathcal{G})$ iff the goal item $[\bullet, n, (T)]$ is generated. This means that $S \stackrel{*}{\Rightarrow} w_1 \cdots w_n$, and T is the parse tree.

In order to adapt this scheme for partial parsing, we consider a modified item form, adding a reference to the starting position of the potentially partial parse. A partial parse covers any piece of the input string. So, instead of an axiom item starting at position 0, we have now the following set of axioms:

$$\text{Axioms} = \{[\bullet A, i, i, ()], A \in \mathcal{S},\ 0 \leq i < n\}$$

The deductive steps remain as before, but keeping the starting point. As a partial parse may cover any piece of the input string, it may start at any position, and it may finish at any position after the starting point. As consequence, we have that:

$$\text{Goals} = \{[\bullet, i, j, T], A \in \mathcal{S},\ 0 \leq i \leq j \leq n\}$$

We shall generate $m \times n$ axioms, where $m = \|\mathcal{S}\|$ and n is the input length. For each of those axioms we will generate a new analysis branch, and, consequently, new items. In our running grammar, for an input string $aaba$ and a complete top-down parse, we need to create 30 items, instead of 82 in the partial case.

2.2 A bottom-up scheme

We include in Fig. 3 the scheme for bottom-up complete parsing. Items are now of the form $[\alpha\bullet, j, (T)]$, stating that $\alpha w_{j+1} \cdots w_n \stackrel{*}{\Rightarrow} w_1 \cdots w_n$, T being the parse tree. The dot indicates that α reduces the input substring till position j.

Bottom-up parsing starts at position 0 before reducing any piece of the input string. Therefore the axiom is $[\bullet, 0, ()]$. The deduction steps are:

Scanning: It shifts the next terminal and moves the dot one position forward.

Completion: It reduces the k_n symbols immediately after the dot. Those symbols must match the right hand side of rule r_k.

As usual, $w \in \mathcal{L}(\mathcal{G})$ iff the goal item $[S\bullet, n, (T)]$ is generated. This means that $S \stackrel{*}{\Rightarrow} w_1 \cdots w_n$, where T is the parse tree.

In order to deal with partial parsing, items are extended with a reference to the starting position. Regarding axioms, a partial parse may start at any position in the input string. So, we have that:

$$\text{Axioms} = \{[\bullet, i, i, ()],\ 0 \leq i < n\}$$

In deduction steps we keep the starting position. Again, the goal item is replaced by a set of items. To construct this set we must have taken into account that a partial parse may fi nish at any symbol of $\mathcal{S}$, at any input position. So, we have that:

$$\text{Goals} = \{[A\bullet, i, j, T], A \in \mathcal{S},\ 0 \leq i \leq j \leq n\}$$

As far as effi ciency issues are concerned, the main difference with complete parsing is the set of axioms. The size of this set is n, n being the length of the input. Retaking the former example, this yields 29 items for complete parsing and 54 for partial parsing.

Item form	$[A \to \alpha \bullet \beta, i, j, T],\ A \to \alpha\beta \in R$
Invariant	$S \stackrel{*}{\Rightarrow} w_1 \cdots w_j \beta$ $\alpha w_{j+1} \cdots w_n \stackrel{*}{\Rightarrow} w_1 \cdots w_n$
Axioms	$[S' \to \bullet S, 0, 0, ()]$
Goals	$[S' \to S\bullet, 0, n, T]$
Scanning	$\dfrac{[A \to \alpha \bullet w_{j+1}\beta, i, j, T]}{[A \to \alpha w_{j+1} \bullet \beta, i, j+1, T w_{j+1}]}$
Prediction	$\dfrac{[A \to \alpha \bullet B\beta, i, j, T]}{[B \to \bullet\gamma, j, j, ()]}\ \langle r_k : B \to \gamma \in R \rangle$
Completion	$\dfrac{[A \to \alpha \bullet B\beta, i, k, T_1][B \to \gamma\bullet, k, j, T_2]}{[A \to \alpha B \bullet \beta, i, j, T_1 \text{tree}(B, T_2)]}$

Fig. 4. *Earley*'s scheme

2.3 Earley's scheme

Earley's algorithm [3] will illustrate the extension from complete to partial parsing using a mixed-strategy with dynamic prediction. The complete parsing scheme is shown in Fig. 4.

We have now items of the form $[A \to \alpha \bullet \beta, i, j, T]$. The dot is still a reference to position j, but now the item represents the state in recognizing the rule $A \to \alpha\beta$, where α reduces some of the substring just before the dot and β remains to be parsed. In relation to i, it points to the begining of the substring parsed by α. Consequently

items represent a local state of parsing process instead of a global one. Having local information enclosed makes it easier to share computations among items.

The grammar is augmented with an artificial rule $S' \rightarrow S$, where S' is a distinct symbol. This facilitates the definition for axioms and goals. The parse starts with the axiom $[S' \rightarrow \bullet S, 0, 0, ()]$, at position 0 we want to reduce the initial symbol S. The deduction steps are as follows:

Scanning: After recognition of terminal w_{j+1}, it moves the pointer from position j to $j+1$.

Prediction: It predicts all the rules $B \rightarrow \gamma$, because they may reduce B from position j.

Completion: After we finish the parse of a rule $B \rightarrow \gamma$, from position k to j, it searches for items whose next symbol to analyze is B at position k. For those items it generates a new one by moving the dot to just after B, position j.

Once we generate the goal item $[S' \rightarrow S\bullet, 0, n, T]$, we know $w_{0\ldots n}$ reduces to the initial symbol S, and $w \in \mathcal{L}(\mathcal{G})$.

We need to modify axioms and goal items in order to deal with partial parses. Axioms start with any initial symbol in $\mathcal{S}$ at any position:

$$\text{Axioms} = \{[S' \rightarrow \bullet A, i, i, ()], A \in \mathcal{S},\ 0 \leq i < n\}$$

and goals finish at any position after the starting point:

$$\text{Goals} = \{[S' \rightarrow A\bullet, i, j, T], A \in \mathcal{S},\ 0 \leq i \leq j \leq n\}$$

The comparison between complete and partial parsing is similar to that we made for the top-down scheme, the number of axioms growing from one to $n \times m$, where n is the input string length and $m = \|\mathcal{S}\|$. So, returning to our running grammar, this increment means that we need 35 items for a complete parse and 52 for a partial one.

2.4 A mixed-strategy with static control

We introduce now the complete parsing scheme for an LR(0) based parse. A preliminary description of the deductive interpretation, where we have omitted the finite state control in order to make the differences with Earley's scheme clear, is shown in Fig. 5. This difference is rooted in the meaning of the items. In Earley's case, the sequence of symbols α, immediately to the left of the dot, reduce the substring $w_{i\ldots j}$. In LR(0), only the symbol X ($\alpha = \alpha' X$) immediately on the left of the dot reduces the substring. As a consequence, we need different deduction steps to manage different items:

Shift: It is similar to *scanning* in Earley's case. The starting position reflects the last symbol analyzed, w_j.

Reduce: It replaces the *completion* step of Earley. In Earley's scheme, we need an item $[B \rightarrow X_1 \cdots X_m \bullet, j_0, j_m, T_m]$ that reflects the recognition of $B \rightarrow X_1 \cdots X_m$ between positions j_0 and j_m.
In LR(0) we need m items of the form $[B \rightarrow X_1 \ldots X_i \ldots Xm, j_{i-1}, j_i, T_i]$. Each item reflects the recognition of the m symbols of the right hand side of the rule at adjacent positions between j_0 and j_m.

Item form	$[A \to \alpha \bullet \beta, i, j, T] \left\langle \begin{array}{c} A \to \alpha\beta \in R, \\ 0 \le i \le j \le n \end{array} \right\rangle$
Invariant	$S \overset{*}{\Rightarrow} w_1 \cdots w_i X \beta$ $X \overset{*}{\Rightarrow} w_{i+1} \cdots w_j$ $\alpha = \alpha' X, \exists k, k \le i, \alpha' \overset{*}{\Rightarrow} w_{k+1} \cdots w_i$
Axiom	$[S \to \bullet\alpha, 0, 0, ()]$
Goal	$[S \to \alpha\bullet, 0, n, T]$
Shift	$\dfrac{[A \to \alpha \bullet w_j \beta, i, j, T]}{[A \to \alpha w_j \bullet \beta, j, j+1, w_j]}$
Prediction	$\dfrac{[A \to \alpha \bullet B\beta, i, j, T]}{[B \to \bullet\gamma, j, j, ()]}$
Reduce	$\dfrac{\begin{array}{l}[B \to X_1 X_2 \cdots X_m \bullet, j_{m-1}, j_m, T_m], \\ \ldots, \\ [B \to \bullet X_1 X_2 \cdots X_m, j_0, j_1, ()], \\ [A \to \alpha \bullet B\beta, i, j_0, T_0]\end{array}}{[A \to \alpha B \bullet \beta, j_0, j_m, \text{tree}(B, T_1 \ldots T_m)]}$

Fig. 5. LR(0) scheme, omitting finite control

Prediction: The same as Earley's scheme.

Next, we will get a more efficient scheme adding a finite state control. This implies replacing dotted rules $A \to \alpha \bullet \beta$ by an state, st, representative of its equivalence class. In order to build the finite state control, we init $st_0 = [S \to \bullet\alpha]$. Then we build the other states with the closure of their items $[A \to \alpha \bullet B\beta]$. More precisely, the closure operation adds items $[B \to \bullet\gamma]$ for each rule $B \to \gamma$. The closure is in fact equivalent to the *prediction* step. So, it will be removed from the LR parsing scheme with finite state control, as shown in Fig. 6. Here, *action(state, token)* denotes a shift or reduce action in the automaton for a given *state* and *token*. In the same way, *goto(state, variable)* looks for a *goto* action. Finally, *reveals(state)* refers to all those states with a shift or goto action over *state*.

To extend the scheme to partial parsing, it must be allowed to start the parsing at any point of the input string, with any symbol of $\mathcal{S}$:

$$\text{Axioms} = \{[st_0, i, i, ()], 0 \le i \le n]\}$$

and, to finish at any position after the starting point:

$$\text{Goals} = \{[st_f, i, j, T], 0 \le i \le j \le n]\}$$

We need to change the initialization step when building the finite state control. So, instead of $st_0 = \{[S \to \bullet\alpha]\}$, the first state must be $st_0 = \{[A \to \bullet\alpha] \mid A \in \mathcal{S}\}$.

We can improve the LR(0) parsers with a better finite state control. Now, when building the states, we must add information about which lookahead symbols are compatible with their actions. This control could result in an LR(1) or LALR(1) parsing

Item form	$[st, i, j, T]$
Axiom	$[st_0, 0, 0, ()]$
Goal	$[st_f, 0, n, T]$
Shift	$\dfrac{[st, i, j, T]}{[st', j, j+1, w_j]}\ \langle \text{shift}_{st'} \in \text{action}(st, w_j)\rangle$
Reduce	$\dfrac{\begin{array}{l}[st_m, j_{m-1}, j_m, T_m],\\ \ldots,\\ [st_1, j_0, j_1, ()],\\ [st_0, i, j_0, T_0]\end{array}}{[st, j_0, j_m, tree(r, T_1 \ldots T_m)]}\ \left\langle \begin{array}{l}\text{reduce}_r \in \text{action}(st_m, w_j),\\ st_i \in \text{reveals}(st_{i+1}),\\ st \in \text{goto}(st_0, \text{lhs}(r)),\\ m = \text{length}(\text{rhs}(r))\end{array} \right\rangle$

Fig. 6. LR(0) schema

Item form	$[st, b, i, j, T]$
Axiom	$[st_0, \$, 0, 0, ()]$
Goal	$[st_f, \$, 0, n, T]$
Shift	$\dfrac{[st, b, i, j, T]}{[st', b, j, j+1, w_j]}\ \langle \text{shift}_{st'} \in \text{action}(st, w_j)\rangle$
Reduce	$\dfrac{\begin{array}{l}[st_m, b', j_{m-1}, j_m, T_m],\\ \ldots,\\ [st_1, b', j_0, j_1, ()],\\ [st_0, b, i, j_0, T_0]\end{array}}{[st, b, j_0, j_m, tree(r, T_1 \ldots T_m)]}\ \left\langle \begin{array}{l}\text{reduce}_r \in \text{action}(st_m, w_{j_m}),\\ st_i \in \text{reveals}(st_{i+1}),\\ st \in \text{goto}(st_0, \text{lhs}(r)),\\ m = \text{length}(\text{rhs}(r))\end{array} \right\rangle$

Fig. 7. LALR(1) schema

algoritm, depending on the computation of the lookahead symbols. Next, in order to adapt the LR(0) scheme to use the LALR(1) control, we will add preconditions to deductive steps. The preconditions will check that the lookahead is compatible with the operation. The resulting scheme is shown in Fig. 7.

Actually *action* and *goto* are the core the table of the LALR(1) automaton. This table changes for an LR(1) automaton, but its interpretation remains the same. As a consecuence, we can use the same parsing schema for LALR(1) and LR(1), provinding we change the finite state control.

Once again, the extension to partial parsing implies adding axioms and goals, but with one new consideration. Because a partial parse may finish at any input position, the finishing operation is compatible with any terminal, and not only with the end of the input terminal. So, we have that the set of axioms is

$$\text{Axioms} = \{[st_0, \_, i, i, ()], 0 \leq i \leq n, A \in \mathcal{S}\}$$

and the set of goals is

$$\text{Goals} = \{[st_j, \_, i, j, T], 0 \leq i \leq j \leq n, A \in \mathcal{S}\}$$

To build now the finite state control, we need the concept of a *variable* terminal that matches any terminal of the grammar. This may produce an exponential growth of the number of states. To mimic the intended behavior without modifying the set of states we make the input string ambiguous. At each position, we have both the original terminal, w_i, and the end of the input. The former is compatible with any operation that follows the parsing process and the latter is compatible with finishing a parse which is probably partial.

In our running example, the complete LR(0) scheme needs 35 items, while the partial one needs 52. The LALR(1) schema need, respectively, 32 and 42 items.

Item form	$[A, st, i, j, T] \cup [\nabla_{r,s}, st, i, j, T]$
Axiom	$[-, st_0, 0, 0, ()]$
Goal	$[S', st_f, 0, n, T]$
InitShift	$\dfrac{[A, st, i, j, T]}{[A_{r,1}, st', j, j+1, w_j]} \left\langle \begin{array}{l} A_{r,1} = w_j \wedge \\ \text{shift}_{st'} \in \text{action}(st, w_j) \end{array} \right\rangle$
Shift	$\dfrac{[A_{r,s}, st, i, j, T]}{[A_{r,s+1}, st', j, j+1, w_j]} \left\langle \begin{array}{l} A_{r,s+1} = w_j \wedge \\ \text{shift}_{st'} \in \text{action}(st, w_j) \end{array} \right\rangle$
Sel	$\dfrac{[A_{r,n}, st, i, j, T]}{[\nabla_{r,n}, st, j, j, ()]} \langle \text{reduce}_r \in \text{action}(st, w_j) \rangle$
Red	$\dfrac{[\nabla_{r,s}, st, k, j, T_1]\ [A_{r,s}, st, i, k, T_2]}{[\nabla_{r,s-1}, st', i, j, T_2 T_1]} \langle st' \in \text{reveals}(st) \rangle$
Head	$\dfrac{[\nabla_{r,0}, st, i, j, T]}{[A_{r,0}, st', i, j, \text{tree}(A_{r,0}, T)]} \langle st' \in \text{goto}(st, A_{r,0}) \rangle$

Fig. 8. LALR(1) scheme with implicit binarization

3 The dynamic interpretation

Given that actions on the automaton depend on the first, and possibly the second, elements in the stack, we implicitly consider a grammar which 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 the need for the language grammar to be in Chomsky Normal Form.

– Sharing of computations on the parsing of a tail of sons in a node 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 in bottom-up constructions and depth-first search results in top-down ones, as is shown in figure 9.

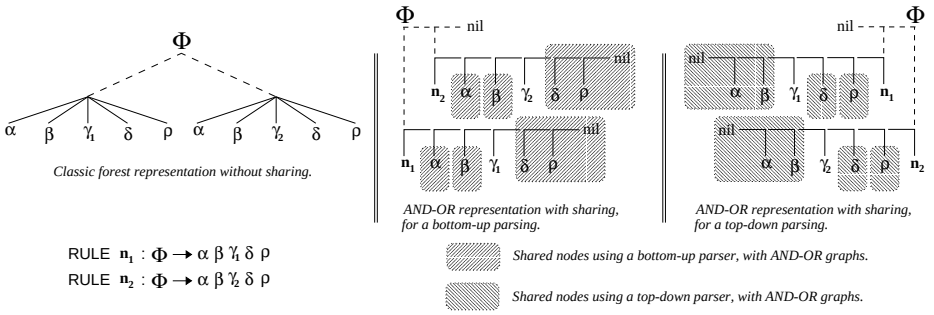

Fig. 9. Sharing of a tail of sons in a node

In order to obtain $\mathcal{O}(n^3)$ complexity in the general case, we can use a implicit binarization of rules. We do this by splitting each reduction involving m elements in the reduction of $m+1$ rules with at most 2 elements on their right-hand side. Thus, the reduction of a rule $A_{r,0} \rightarrow A_{r,1} \cdots A_{r,n_r}$ is equivalently performed as the reduction of the following $n_r + 1$ rules:

$$\begin{array}{ll} A_{r,0} \rightarrow \nabla_{r,0} & \nabla_{r,0} \rightarrow A_{r,1}\nabla_{r,1} \\ \vdots & \vdots \\ \nabla_{r,n_r-1} \rightarrow A_{r,n_r}\nabla_{r,n_r} & \nabla_{r,n_r} \rightarrow \epsilon \end{array}$$

This treatment of reductions involves a change in the form of the items. We add a new element, representing a symbol in a rule or a $\nabla_{r,i}$ meaning that elements $A_{r,i+1} \ldots A_{r,n_r}$ have been reduced[1].

With respect to deduction steps, we must now differentiate between whether we make the shift of the first symbol in the right hand side of a rule (*InitShift*) or the shift of other symbols (*Shift*).

The *Reduce* step has also been refined into three steps. The *selection* of the rule to be reduced (*Sel*), the *reduction* of the implicit binary rules (*Red*), and the *recognizing* of the left-hand symbol of the rule to be reduced (*Head*). The resulting scheme is shown in Fig. 8. This scheme corresponds to a dynamic interpretation of LALR(1) parsing algorithms using an inference system based on S^1 items [2].

[1] $\nabla_{r,i}$ is equivalent to the dotted rule $A_{r,0} \rightarrow \alpha\beta$ where $\alpha = A_{r,1} \ldots A_{r,i}$ and $\beta = A_{r,i+1} \ldots A_{r,n_r}$.

The extension for partial parsing is analogous to previous schema, following an identical approach for table construction, and adding new axioms and goals. So, the set of axioms is given by:

$$\text{Axioms} = \{[st_0, \_, i, i, ()], 0 \leq i \leq n, A \in \mathcal{S}\}$$

and the set of goals by:

$$\text{Goals} = \{[st_j, \_, i, j, T], 0 \leq i \leq j \leq n, A \in \mathcal{S}\}$$

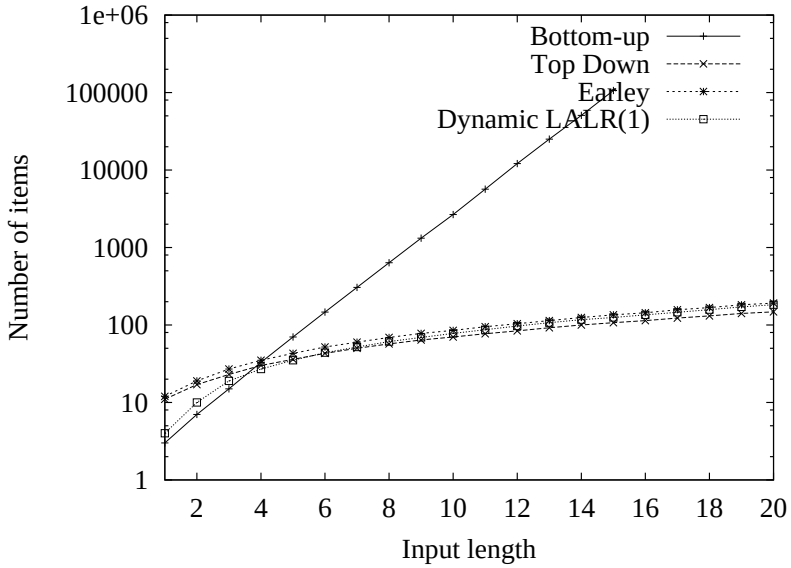

Fig. 10. Num. of items in complete parsing

4 Experimental results

To illustrate the practical aspects of our proposal, we provide now some preliminary experimental results. We have parsed several input strings with lengths varying from 1 to 20, considering our running example, the language $\mathcal{P}$. The number of items is different even for input strings of the same length. So, for each length we have parsed several input strings, computing the average number of items generated, both for complete and partial parsing. The grammar we are using seems to be well-suited for top-down parsing, as shown in Fig. 10. Bottom-up parsing only performs well for short length inputs. On the other hand, dynamic programming approaches, both Earley parsing and dynamic interpretation of LALR(1), perform as well as top-down parsing.

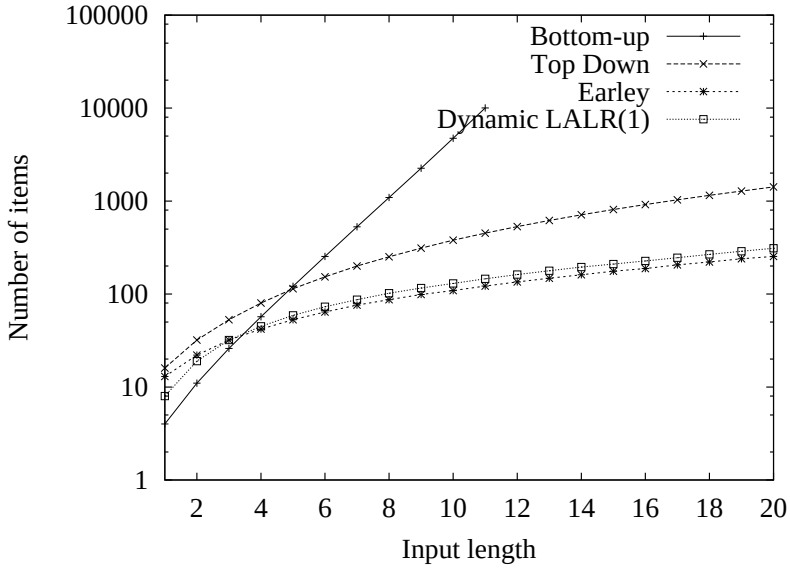

Fig. 11. Num. of items in partial parsing

Regarding partial parsing, Fig. 11, top-down parsing suffers a drop in performance, while bottom-up still performs well for short length inputs. Dynamic programming approaches continue to show a good behavior.

In Fig. 12 we illustrate the relation between complete and partial parsing, synthesizing the last two figures. We have replaced the number of generated items by the increment from the number of items in complete parsing to the number of items in partial parsing. As is shown, top-down partial parsing is not as advantageous as it was for complete parsing. It suffers from exponential grown. On the other hand, the parsing schema with some kind of bottom-up strategy scale well.

5 Conclusions

We have described a practical approach to partial parsing in the domain of CFGs. In comparison with previous works, our proposal is based on a deductive parsing scheme, which provides a uniform framework to compare performances between different parsing strategies for both complete and partial cases.

From a theoretical point of view, we have graduated the introduction of each parsing scheme in order to make clear the existing relationships with previous strategies. This leads to a better understanding of the mechanisms regulating the definition of the deduction rules and even of the structures manipulated by these. The evolution from complete parses to partial ones is also justified in each case.

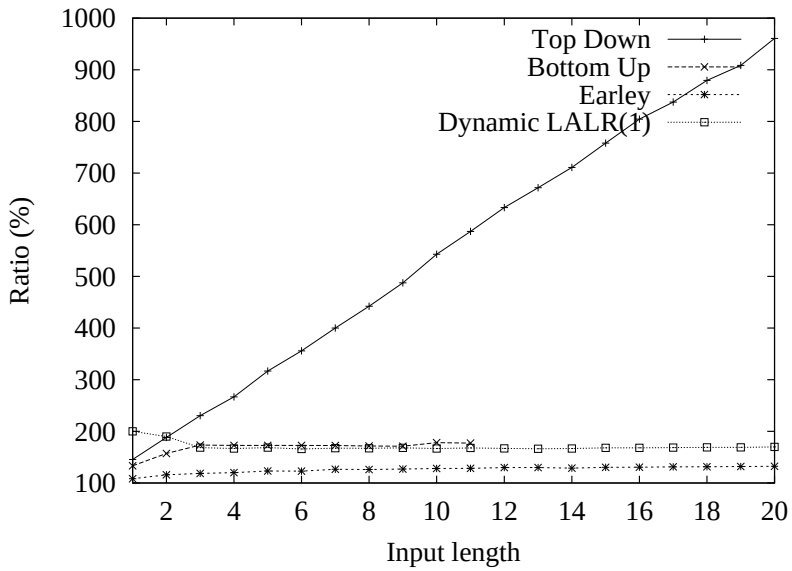

Fig. 12. Ratio partial/complete parsing

6 Acknowledgments

This work has been partially supported by the European Union, Government of Spain and Autonomous Government of Galicia under projects 1FD97-0047-C04-02, TIC2000-0370-C02-01 and PGIDT99XI10502B, respectively.

The source code of the deduction scheme interpreter comes from the original one of [7], and has been adapted by V.J. Díaz Madrigal.

References

1. M.A. Alonso, D. Cabrero, E. de la Clergerie, and M. Vilares. Tabular algorithms for TAG parsing. In *Proc. of EACL'99, Ninth Conference of the European Chapter of the Association for Computational Linguistics*, pages 150–157, Bergen, Norway, June 1999. ACL.
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. I. Jacobs. *The* CENTAUR *1.2 Manual*. INRIA, Sophia-Antipolis, France, 1992.
5. Christian Jacquemin. Receycling terms into a partial parser. In *Proc. of the 4^{th} Conf. on Applied Natural Language Processing. Stuttgart, DE, 13–15 Oct 1994*, pages 113–118, 1994.
6. V. Rocio and J. G. Lopes. Partial parsing, deduction and tabling. In *Proceedings of Tabulation in Parsing and Deduction (TAPD'98)*, pages 52–61, Paris (FRANCE), April 1998.
7. S.M. Shieber, Y. Schabes, and F.C.N. Pereira. Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24(1-2):3–36, 1995.

8. K. Sikkel. *Parsing Schemata*. PhD thesis, University of Twente, The Netherlands, 1993.
9. Michael Sperber and Peter Thiemann. The essence of LR parsing. In *Proceedings of the ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation*, pages 146–155, La Jolla, California, 21-23 June 1995.
10. M. Vilares and M.A. Alonso. An LALR extension for DCGs in dynamic programming. In Carlos Martín Vide, editor, *Mathematical and Computational Analysis of Natural Language*, volume 45 of *Studies in Functional and Structural Linguistics*, pages 267–278. John Benjamins Publishing Company, Amsterdam & Philadelphia, 1998.
11. M. Vilares and B.A. Dion. Efficient incremental parsing for context-free languages. In *Proc. of the* 5^{th} *IEEE International Conference on Computer Languages*, pages 241–252, Toulouse, France, 1994.