An Experience on Natural Language Parsing[1]

M. Vilares Ferro[2] M. A. Alonso Pardo[3] D. Cabrero Souto[4]

We propose an interactive development environment for the automatic generation of natural language analyzers from definite clause grammars. Our system has been baptized GALENA, and it is organized around a generator of logical push-down automata, which are interpreted in dynamic programming. An extensible user graphical interface provides a complete set of customization and trace facilities for the system. In relation to previous approaches, our proposal guarantees operational completeness and correctness at the time that efficiency increases.

Key Words: Definite Clause Grammar, Push-Down Automaton, Language Prototyping.

1. Introduction

Grammar formalisms based on the encoding of grammatical information in complex-valued feature systems, such as definite clause grammars (DCGs), have been popular in these recent years in natural language processing (NLP). The most salient feature of this approach is its declarativeness, but descriptive adequation does not guarantee operational efficiency, which is required for practical systems. In fact, natural language analyzers today are often based on depth-first left-to-right parsing schema with backtracking. This allows the consideration of effective space recovery techniques, a crucial point in NLP due to the complexity of its computational treatment. However, these implementations do not guarantee sharing of computations even operational completeness. As a consequence, actual systems in NLP often go far from the declarative model due to the user, and efficiency decreases.

To deal with this problem, we have developed a parser generation environment based on the concept of logical push-down automaton (LPDA) [1]. In essence, an LPDA is a classic push-down automaton that stores logical atoms and substitutions on its stack and uses unification to apply transitions. This allows us to guarantee completeness and correctness in the parsing of DCGs without functional symbols, at the price to surcharge the evaluation process. To solve this drawback , we have adapted the formal

[1] This work was partially supported by the Autonomous Government of Galicia under project XUGA20403B95

[2] M. Vilares is with the Computer Science Department, University of Corunna, Campus de Elviña s/n, 15071 A Coruña, Spain. E-mail: vilares@dc.fi.udc.es.

[3] M. A. Alonso is currently at INRIA, Domaine de Voluceau, Rocquencourt, B.P.105, 78153 Le Chesnay Cedex, France. E-mail: Miguel.Alonso-Pardo@inria.fr.

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

model to increase parsing efficiency by reducing the amount of computations. In particular, we have optimized the sharing of these computations as well as the restriction of the computation process to a useful part of the search space.

In section 2 of this paper, we introduce our approach to dynamic programming in LPDAs. Section 3 describes our parsing scheme. In section 4 we show the programming environment and an overview of the system at work. Section 5 compares our proposal with previous approaches. Finally, section 6 is a conclusion about the work presented.

2. The operational formalism

Formally, our parser engine is an LPDA of the form A = (χ, F, Σ, Δ, \$, $\$_f$, Θ), where: χ is a denumerable and ordered set of variables, 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*.

Our automaton manipulates *items* more than simple stacks. Items are of the form $[A,it,bp,st].\sigma$, where A is in the algebra of terms $T_\Delta[F \cup \chi]$, σ is 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 an state for a driver controlling the evaluation. Transitions are of three kinds:

- *Horizontal:* $B \longrightarrow C\{A\}$. Applicable to items E, iff there exists the *most general unifier* (mgu), σ=mgu(E,B) such that $F\sigma = A\sigma$, for a fact F in the extensional database. We obtain the new item $C\sigma$.
- *Pop:* $BD \longrightarrow C\{A\}$. Applicable to items E, iff there is σ = mgu(E,B) , such that $F\sigma = A\sigma$, for a fact F in the extensional database. The result will be the *dynamic transition* $D\sigma \longrightarrow C\sigma$.
- *Push:* $B \longrightarrow CB\{A\}$. We can apply it to items E, iff there is σ=mgu(E,B), such that $F\sigma = A\sigma$, for a fact F in the extensional database. We obtain the item $C\sigma$.

Where B, C and D are items and A is in the algebra of terms $T_\Sigma[F \cup \chi]$, representing a control condition.

Our transitional formalism improves the computational efficiency from two different points of view: First, it manipulates items in the *dynamic frame* S^1, which implies that stacks are represented by their top, to favour sharing of computations. Second, it indexes the parsing by string position to reduce the *search space*, that is, the set of items to be explored when we apply a transition. This also permits the space to be recovered, as parsing progresses, by deleting information relating to earlier string positions. Furthermore, we shall call *itemset* the set of items generated at the same position in the input string.

esp(tree)	:	PHRASE(tree)
PHRASE(phr(tree1, tree2)	:	NP(tree1, nmbr), VP(tree2, nmbr)
PHRASE(phr(tree1, tree2)	:	PHRASE(tree1), PP(tree2)
NP(np(s(wrd)), nmbr)	:	NOUN(wrd:word, nmbr:number)
NP(pr(wrd), nmbr)	:	PRONUON(wrd:word, nmbr:number)
NP(np(det(wrd1), s(wrd2)), nmbr)	:	DETERMINER(wrd1:word, nmbr:number, gndr:gender), NOUN(wrd2:word, nmbr:number, gndr:gender)
NP(np(tree1, tree2), nmbr)	:	NP(tree1, nmbr), PP(tree2)
PP(pp(prep(wrd), tree))	:	PRESPOSITITON(wrd:word), NP(tree,nmbr)
VP(vp(verb(wrd), tree), nmbr)	:	VERB(wrd:word, nmbr:number), NP(tree, nmbr)

Example DGC.

3. The parsing strategy

To illustrate our discussion, we shall consider, through the rest of this paper, the above DCG as running example. This set of clauses describes a simple subset of Spanish, establishing some known restrictions in relation to the number and the gender. So, for example, we force that the first rule to parse a phrase must guarantee the congruence of the number between the nominal and the verbal predicates. The mechanism for information passing is based on unification. We also recover information from the tagging set of the lexical categories in a simple manner. In relation to this, GALENA includes a generator of taggers described in [2]. So, for example, the first clause for NP recovers the string and the number of the noun, by using the key words *word* and *number*. The results of these queries are saved, in this case, in the attributes *wrd* and *nmbr* respectively. The number *nmbr* will be used to initialize the corresponding attribute of the nominal predicate, while the string *wrd* will be used to build the abstract tree *np(s(wrd))*.

To introduce formally our parsing strategy, let's assume a DCG of clauses γ_k of the form $A_{k,0} : A_{k,1}, \ldots, A_{k,n_k}$. We consider:

- The vector T_k of the variables occurring in γ_k.
- The predicate symbol $\nabla_{k,i}$. An instance of $\nabla_{k,i}(T_k)$ indicates that all literals from the i^{th} in the body of the clause γ_k, have been proved.

We first recover the context-free skeleton of the logic program, by keeping only functors in the clauses to obtain terminals from the extensional database, and variables from heads in the intensional one. Terms with the same name, but different number of arguments correspond to different symbols in the skeleton. The result is the augmented context-free grammar $G^f=(\Sigma^f,N^f,P^f,S^f)$, that in our running example is given by the following rules:

Φ	⟶	S	NP	⟶	noun
S	⟶	PHRASE		|	pronoum
PHRASE	⟶	NP VP		|	determiner noun
	|	PHRASE PP		|	NP PP
VP	⟶	verb NP	PP	⟶	preposition NP

Table 1

1. $[A_{k,nk}, it\ , bp, st]$	⟶	$[\nabla_{k,nk}(T_k), it, it, st]\ [A_{k,nk}, it\ , bp, st]$
		{action(st, token_{it}, reduce(γ_k))}
2. $[\nabla_{k,i}(T_k), it, r, st_1]\ [A_{k,nk}, r\ , bp, st_2]$	⟶	$[\nabla_{k,i-1}(T_k), it, bp, st_2]$
		{action(st_2, token_{it}, shift(st_1)), lookahead($A^f_{k,i}$, token_{it})}
3. $[\nabla_{k,0}(T_k), it, bp, st]$	⟶	$[A_{k,0}, it, bp, st]$
4. $[A_{k,i}, it, bp, st_1]$	⟶	$[A_{k,i+1}, it+1, st_2]\ [A_{k,i}, it, bp, st_1]$
		{action(st_1, token_{it}, shift(st_2)), follow($A^f_{k,i}$, token_{it})}
5. [\$, 0, 0, 0]	⟶	$[A_{k,1}, 0, 0, st]$ [\$, 0, 0, 0]
		{action(0, token_0, shift(st)), first(Φ,$A^f_{k,1}$)}

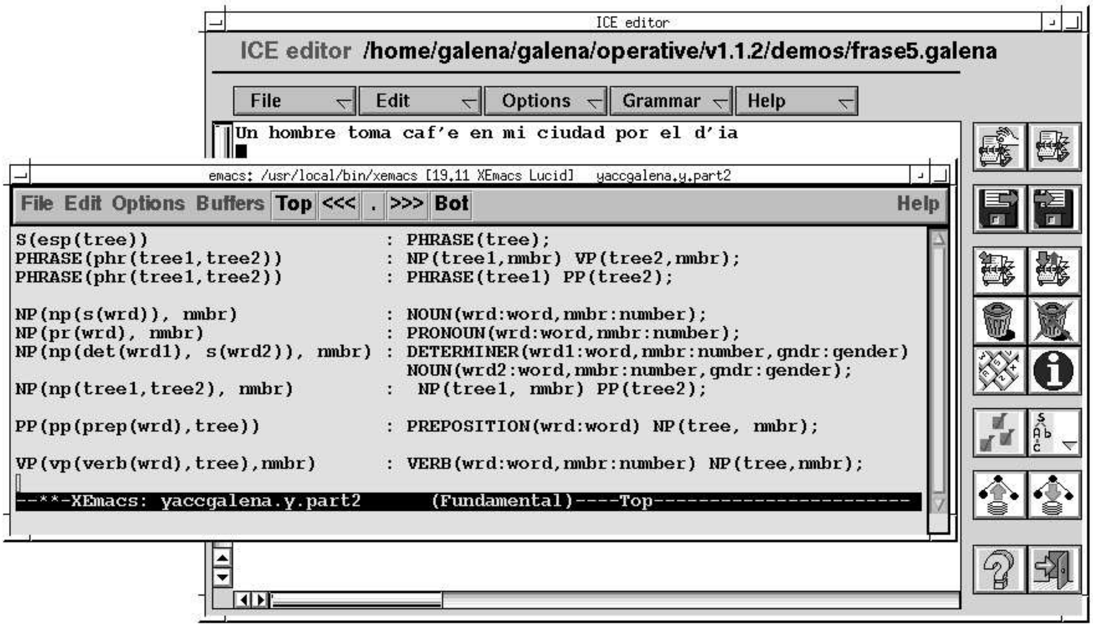
Table 2

We denote by $A^f_{k,i}$ the term in^f G obtained from $A_{k,i}$, and by γ^f_k the rule corresponding to the clause γ_k. Then, we consider the set of transitions in *table 2*: where *action*(*state*,*token*,*do*) denotes the action *do* of the LALR(1) automaton for G^f, for a given *state* and *token*. We denote by *first*, *follow* and *lookahead* the known concepts on formal languages [3] of the type LALR(1). Briefly, we can interpret these transitions as follows:

1. *Selection of a clause:* Select the clause γ_k whose head is to be proved; then push $\nabla_{k,nk}(T_k)$ on the stack to indicate that none of the body literals have yet been proved.
2. *Reduction of one body literal:* The position literal $\nabla_{k,i}(T_k)$ indicates that all body literals of γ_k following the ith literal have been proved. Now, for all stacks having $A_{k,i}$ just below the top, we can reduce them and in consequence increment the position literal.
3. *Termination of the proof of the head of clause γ_k:* The position literal $\nabla_{k,0}(T_k)$ indicates that all literals in the body of γ_k have been proved. Hence, we can replace it on the stack by the head $A_{k,0}$ of the rule, since it has now been proved.
4. *Pushing literals:* The literal $A_{k,i+1}$ is pushed onto the stack, assuming that they will be needed in reverse order for the proof.
5. *Initial push transition:* The initial predicate will be only used in push transitions, and exclusively as the first step of the LPDA computation.

The parsing algorithm proceeds by building items from the initial configuration, by applying transitions to existing ones until no new application is possible. An equitable selection order in the search space assures fairness and completeness. Redundant items are ignored by a subsumption-based relation. Correctness and completeness are easily obtained from [3,4], based on these results for LALR(1) context-free parsing and bottom-up evaluation, both using S^1 as dynamic frame. It can be easily proved that time complexity for a successful parsing, including online unification and subsumption checking is $O(\Sigma^n_{i=0}\ i^2) = O(n^3)$, in the worst case, for a input string of length *n*. Space complexity is $O(n^2)$, in the worst case.

We can also characterize the class of grammars which the algorithm does in time and space O(n). They are the *bounded item grammars*, for which the number of items in a given itemset cannot grow indefinitely. This is the case of DCGs whose context-free skeleton is LALR(1). This has a


ICE editor
ICE editor /home/galena/galena/operative/v1.1.2/demos/frase5.galena
File
Edit
Options
Grammar
Help
emacs: /usr/local/bin/xemacs [19.11 XEmacs Lucid] yaccgalena.y.part2
File Edit Options Buffers Top <<< . >>> Bot Help

```
Un hombre toma caf'e en mi ciudad por el d'ia
```

```
S(esp(tree))                        : PHRASE(tree);
PHRASE(phr(tree1,tree2))            : NP(tree1,nmbr) VP(tree2,nmbr);
PHRASE(phr(tree1,tree2))            : PHRASE(tree1) PP(tree2);

NP(np(s(wrd)), nmbr)                : NOUN(wrd:word,nmbr:number);
NP(pr(wrd), nmbr)                   : PRONOUN(wrd:word,nmbr:number);
NP(np(det(wrd1), s(wrd2)), nmbr)    : DETERMINER(wrd1:word,nmbr:number,gndr:gender)
                                      NOUN(wrd2:word,nmbr:number,gndr:gender);
NP(np(tree1,tree2), nmbr)           :  NP(tree1, nmbr) PP(tree2);

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);

--**-XEmacs: yaccgalena.y.part2      (Fundamental)----Top-----------------------
```

Figure 1: The grammar editor

practical sense, since it permits us to benefit from local determinism during the parsing process. In effect, on the often large parts of the computation which are not ambiguous the algorithm becomes linear.

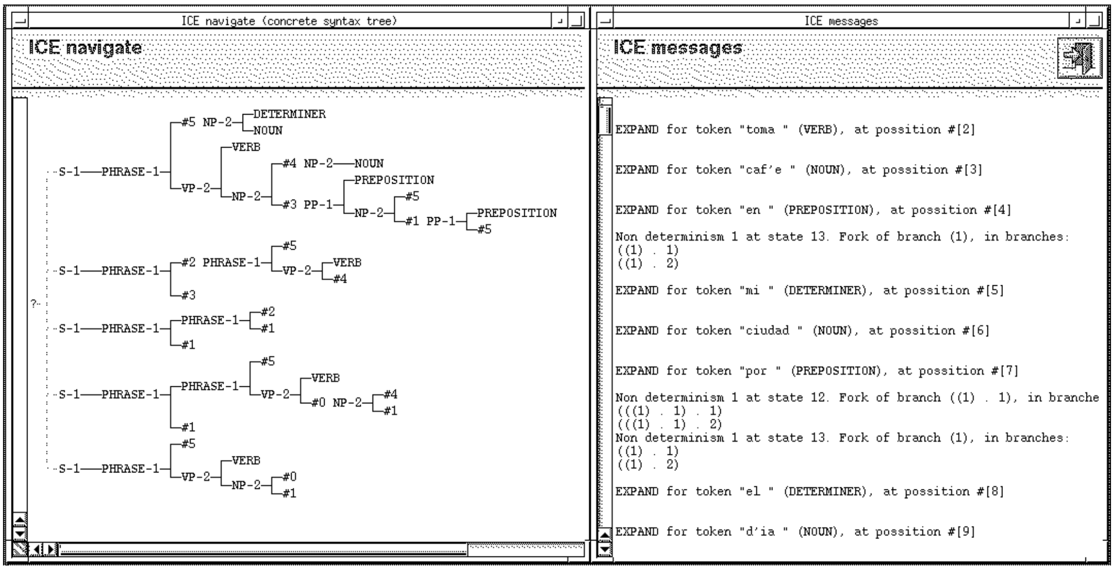
4. The system at work

The GALENA system has a multi-window, menu-driven user interface [5], running under X11. To explain the behavior of the environment at work, we shall build a parser for the subset of Spanish described by the DCG of our running example. Our goal will be to parse the input string *``Un hombre con telescopio toma café en mi ciudad por el día''* from two different points of view:

1.- The concrete syntax defined by the predicate symbols. This establish, for example, that the first alternative to parse a phrase is a tree labeled *PHRASE* with two sons labeled *NP* and *VP* respectively.

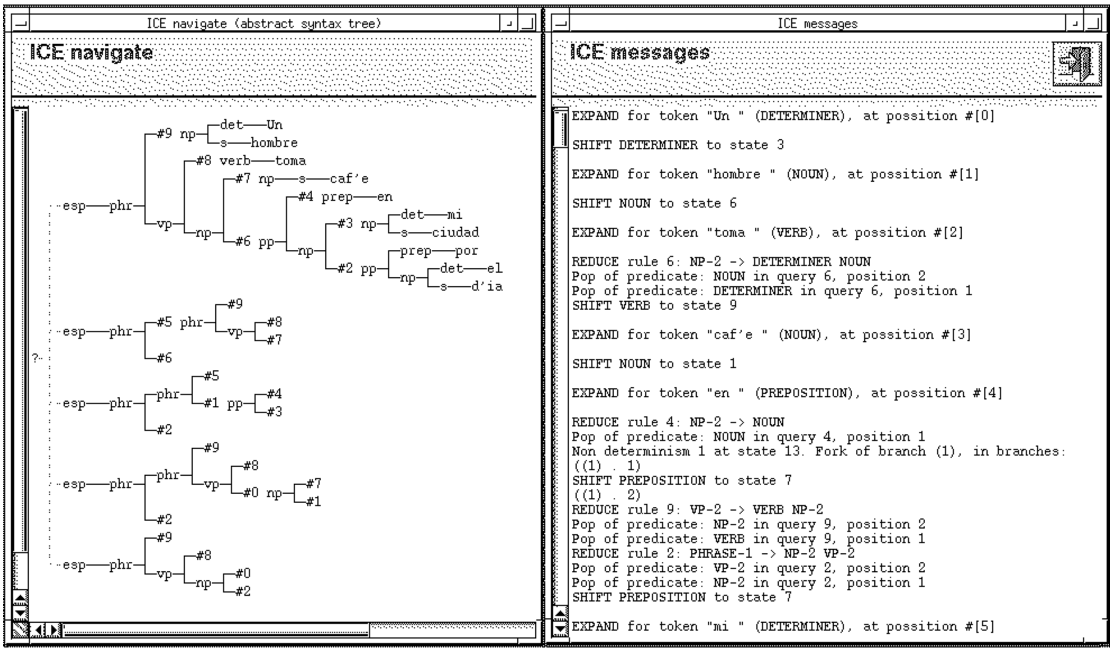
2.- The abstract syntax defined by the functional symbols. This establish, for example, that the first alternative to parse a phrase is a tree labeled *phr* with two sons, *tree1* and *tree2*, which are themselves the abstract trees resulting of the parse of a nominal predicate and a verbal predicate respectively.

To illustrate this discussion, we shall refer to Fig. 1, 2 and 3, representing the external appearance of GALENA at different moments of the parsing process. As first step, we must create the file describing the grammar. For this purpose, the system foresees the interaction with most of standard text editors in UNIX. So, for example, Fig. 1 shows the system working with the EMACS editor [6]. Once we have generated the parser, it is time to test it. The ICEeditor tool has an editor to write the source text, which is shown in the right upper corner of Fig. 1. The input text can also be directly recovered from a file previously edited. Each time we parse a text, the window ICEmessages with the resulting messages is activated, as shown in the



```
EXPAND for token "toma " (VERB), at possition #[2]


EXPAND for token "caf'e " (NOUN), at possition #[3]


EXPAND for token "en " (PREPOSITION), at possition #[4]

Non determinism 1 at state 13. Fork of branch (1), in branches:
((1) . 1)
((1) . 2)

EXPAND for token "mi " (DETERMINER), at possition #[5]


EXPAND for token "ciudad " (NOUN), at possition #[6]


EXPAND for token "por " (PREPOSITION), at possition #[7]

Non determinism 1 at state 12. Fork of branch ((1) . 1), in branche
(((1) . 1) . 1)
(((1) . 1) . 2)
Non determinism 1 at state 13. Fork of branch (1), in branches:
((1) . 1)
((1) . 2)

EXPAND for token "el " (DETERMINER), at possition #[8]


EXPAND for token "d'ia " (NOUN), at possition #[9]
```

Figure 2: A concrete shared-forest



```
EXPAND for token "Un " (DETERMINER), at possition #[0]

SHIFT DETERMINER to state 3

EXPAND for token "hombre " (NOUN), at possition #[1]

SHIFT NOUN to state 6

EXPAND for token "toma " (VERB), at possition #[2]

REDUCE rule 6: NP-2 -> DETERMINER NOUN
Pop of predicate: NOUN in query 6, position 2
Pop of predicate: DETERMINER in query 6, position 1
SHIFT VERB to state 9

EXPAND for token "caf'e " (NOUN), at possition #[3]

SHIFT NOUN to state 1

EXPAND for token "en " (PREPOSITION), at possition #[4]

REDUCE rule 4: NP-2 -> NOUN
Pop of predicate: NOUN in query 4, position 1
Non determinism 1 at state 13. Fork of branch (1), in branches:
((1) . 1)
SHIFT PREPOSITION to state 7
((1) . 2)
REDUCE rule 9: VP-2 -> VERB NP-2
Pop of predicate: NP-2 in query 9, position 2
Pop of predicate: VERB in query 9, position 1
REDUCE rule 2: PHRASE-1 -> NP-2 VP-2
Pop of predicate: VP-2 in query 2, position 2
Pop of predicate: NP-2 in query 2, position 1
SHIFT PREPOSITION to state 7

EXPAND for token "mi " (DETERMINER), at possition #[5]
```

Figure 3: An abstract shared-forest

right-hand-side of Fig. 2 and 3. The user can choose the level of information that will be displayed in this window, from nothing at all to have every elementary action available.

The user can also navigate in the shared forest, which is shown in the ICEnavigate window, in the left-hand-side of Fig. 2 and 3, containing respectively the concrete and the abstract shared-forests. In both cases, ambiguities are represented by dotted-lines, and shared nodes by expressions of the form #*n*. If necessary, the structures generated during the parsing process can be saved on disk and recovered in following sessions.

To get a more friendly environment, we can select the language in which the system interacts with us: English, French, Spanish and Galician[5] are

[5] The co-official language, together with Spanish, in the

currently available. A help facility is available every time to solve questions about the editors, parser generation and parsing facilities.

5. Comparison with previous approaches

We now summarize some of the contributions of our proposal in relation to previous approaches. So, Nilsson [7], and Rosenblueth and Peralta [8] propose SLR(1)-like evaluators, more efficient than grammar oriented algorithms, as [9], because they drastically limit backtracking. The difference between both approaches is due to the form in which they incorporate the contextual information present in DCGs. Nilsson ignores it to generate the driver, delaying its consideration until reduction occurs. To avoid this, Rosenblueth and Peralta concentrate the contextual information into the clauses of the extensional database, which forces to rewriting in a non trivial manner the original DCG, while the application domain is restricted to fixed-mode DCGs. Both of them, Nilsson and Rosenblueth et al., work in S^T and none indexing technique is considered. In consequence, the sharing quality is low.

The reduction of the search space can also be attained by a goal-oriented strategy, whose efficiency depends on the amount of significant work required to evaluate the control predicates introduced. This is the case of the Magic Set methods [10, 11], which disregard the sharing problem. Lang [1] and Villemonte de la Clergerie [4] exploit the dynamic programming construction of LPDAs, but they always consider state-less automata, which rests efficiency to evaluation. The size of the search space depends only on the evaluation scheme and efficiency depends also in fine on the DCG and the corpus of sentences to be analized.

Our proposal guarantees completeness and correctness, in the case of absence of functional symbols, for unrestricted DCGs. Control is given by an LALR(1) driver, with a moderate state splitting phenomenon and large deterministic domain. The dynamic programming construction avoids backtracking, the dynamic frame S^1 assures optimal sharing for the evaluation scheme, and the use of itemsets as synchronization structures facilitates the reduction of the search space. In an empirical comparison, these features seems convert GALENA in the most efficient formalism in relation with the rest of the proposals considered in this section.

6. Conclusion

In this paper, we try to reconcile declarativeness and completeness in efficient unification-based parsing. Our proposal is based on a predictive bottom-up evaluation scheme, where control is provided by a LALR automaton capturing the context-free backbone of the unification grammar. The operational frame is a logical push-down transducer that uses dynamic programming to share computations.

Preliminary results seem demonstrate the adequation of our proposal for computational requirements. Although these results must be extrapolated with caution because our system is still a prototype, we feel that this technique

Autonomous Community of Galicia, Spain.

can be used to make those increasingly powerful grammar formalisms computationally feasible.

7. References

[1] B. Lang, ``Complete evaluation of Horn Clauses, an automata theoretic approach", Tech. Rep. 913, INRIA, Rocquencourt, France, 1988.

[2] M. Vilares Ferro, A. Valderruten Vidal, J. Graña Gil, and M. A. Alonso Pardo, ``Une approche formalle pour la gènèration d'analyseurs de langages naturels", in *Actes de TALN'95*, Marseille, France, 1995.

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

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

[5] M. A. Alonso Pardo, ``Edición interactiva en entornos incrementales", Master's thesis, Computer Sciences Department, University of A Coru\~na, A Coru\~{n}a, Spain, 1994.

[6] R.M. Stallman, *GNU EMACS Manual. Version 18*, Free Software Foundation, Inc., 675 Mass Avenue, Cambridge, MA 02139, U.S.A., 1991.

[7] U. Nilsson, ``AID: An alternative implementation of DCGs", *New Generation Computing*, vol. 4, pp. 383-399, 1986.

[8] 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*, The MIT Press, Cambridge Massachussets 02142 USA, 1994, pp. 439-453.

[9] F.C.N. Pereira and D.H.D. Warren, ``Parsing as deduction", in *Proc. of the 21st Annual Metting of the Association for Computational Linguistics*, 37-144, Ed., Cambridge, Massachusetts, U.S.A., 1984.

[10] F. Bancilhon, D. Maier, Y. Sagiv, and J. Ullman, ``Magic-set and other strange ways to implement logic programs", in *Proc. of the 5th ACM SIGMOD-SIGACT Symp. on Principles of Database Systems*, 1986.

[11] U. Nilsson, ``Abstract interpretation: A kind of magic", in *Proc. of PLILP'91*, 1991.