Towards Analyzers Based on Efficient Logical Frames

Manuel Vilares Ferro Miguel A. Alonso Pardo

Abstract

A general strategy for implementing natural language analyzers is described. Over a base structure equipped with a logic push-down automaton, which provides completeness and correctness, we introduce a flexible level of control order to avoid useless computations. In relation to previous approaches, control is introduced statically, which reduces the amount of work during the evaluation process.

Our proposal ensures computation and syntactic sharing, termination for function-free programs and solves most of the problems posed by classic depth-first left-to-right traversals.

1 Introduction

The use of definite clause grammars (DCG's) for the description of natural language analyzers has regained much attention in theoretical linguistics in the last few years. However, even though considerable work has been done to optimize them, strategies for executing DCG's are still often expressed directly as symbolic manipulations of terms and rules, which is not the basis for efficient implementations. It is also well known that classic depth-first left-to-right schema are not fully adequate for describing natural languages, nor even many programming languages, because of problems with completeness in the implemented resolution algorithm.

Sharing quality is another factor to get efficiency in a framework which is not deterministic. This sharing saves on the space needed to represent the computations, and also on the later processing. Finally, it is desirable to restrict the computation effort to the useful part of the search space, which is not the case for analyzers based on classic backtracking techniques.

We search for a balance between sharing quality and parsing efficiency by working in the context of logic push-down automata (LPDA's), essentially a push-down automaton that stores logical atoms and substitutions on its stack, and uses unification to apply transitions. In this paper, we consider an evolution of the original concept [1]. For us, an LPDA is a 7-tupla $\mathcal{A} = (\mathcal{X}, \mathcal{F}, \Sigma, \Delta, \$, \$_f, \Theta)$, 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 an 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 $T_\Sigma[\mathcal{F} \cup \mathcal{X}]$, representing the control condition. The use of *it* and *bp* is equivalent to indexing the parse, which allows to implement a garbage collector facility, by deleting information relating to earlier substrings, as parsing progresses.

2 An Informal Overview

Practical experience has shown that the most efficient evaluation strategies [2] seem to be those bottom-up approaches including a predictive phase in order to restrict the search space. However, the choice of an efficient evaluation scheme does not guarantee by itself good performances. We must also take into account the general data structures considered. In this sense, we exploit the possibilities of dynamic programming taking S^1 as dynamic frame [2, 3] by collapsing stacks to obtain structures that we call *items*. More exactly, we represent a stack by its top. In this way, we improve sharing of computations in opposition to S^T, the standard dynamic frame, where stacks are represented by all their elements. To replace during pop transitions this lack of information, we redefine the behavior of transitions on

This work was partially supported by the **E**ureka **S**oftware **F**actory project, and by the Autonomous Government of Galicia under projects XUGA10501A93 and XUGA20403B95.

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.

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.

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 resulting from the pop transition, but also to those to be generated and which share the same syntactic context.
- *Push case:* $(B \longmapsto CB)(A) = C\sigma$, where $\sigma = \text{mgu}(A, B)$.

We can now formalize our evaluation strategy. Let's assume a DCG of clauses $\gamma_k : A_{k,0} : -A_{k,1}, \ldots, A_{k,n_k}$. We introduce:

- The vector $\vec{T}_k$ of the variables occurring in γ_k.
- The predicate symbol $\bigtriangledown_{k,i}$. An instance of $\bigtriangledown_{k,i}(\vec{T}_k)$ 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 grammar $\mathcal{G}^f = (\Sigma^f, N^f, P^f, S^f)$, denoting by $A^f_{k,i}$ the term in $\mathcal{G}^f$ obtained from $A_{k,i}$, and by γ^f_k the rule corresponding to the clause γ_k. Then, we consider the following set of transitions:

1. $[A_{k,n_k}, it, bp, st] \longmapsto [\bigtriangledown_{k,n_k}(\vec{T}_k), it, it, st]$ $[A_{k,n_k}, it, bp, st]$ $\{\text{action}(st, \text{token}_{it}, \text{reduce}(\gamma^f_k))\}$
2. $[\bigtriangledown_{k,i}(\vec{T}_k), it, r, st_1]$ $[A_{k,i}, r, bp, st_2] \longmapsto [\bigtriangledown_{k,i-1}(\vec{T}_k), it, bp, st_2]$ $\{\text{action}(st_2, \text{token}_{it}, \text{shift}(st_1)),$ $\text{lookahead}(A^f_{k,i}, \text{token}_{it})\}$
3. $[\bigtriangledown_{k,0}(\vec{T}_k), it, bp, st] \longmapsto [A_{k,0}, it, bp, st]$
4. $[A_{k,i}, it, bp, st_1] \longmapsto [A_{k,i+1}, it+1, it, st_2]$ $[A_{k,i}, it, bp, st_1]$ $\{\text{action}(st_1, \text{token}_{it}, \text{shift}(st_2)),$ $\text{follow}(A^f_{k,i}, \text{token}_{it})\}$
5. $[\$, 0, 0, 0] \longmapsto [A_{k,1}, 0, 0, st]$ $[\$, 0, 0, 0]$ $\{\text{action}(0, \text{token}_0, \text{shift}(st)),$ $\text{first}(\Phi, A^f_{k,1})\}$

where $action(state, token, do)$ denotes the action do of the LALR(1) automaton for $\mathcal{G}^f$, for a given $state$ and $token$. The axiom of $\mathcal{G}^f$ is Φ. 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 $\bigtriangledown^{it,it}_{k,n_k}(\vec{T}_k)$ on the stack to indicate that none of the body literals have yet been proved.
2. *Reduction of one body literal:* The position literal $\bigtriangledown^{it,r}_{k,i}(\vec{T}_k)$ indicates that all body literals of γ_k following the i^{th} literal have been proved. Now, for all stacks having $\text{A}^{r,bp}_{k,i}$ 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 $\bigtriangledown^{it,bp}_{k,0}(\vec{T}_k)$ 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 $A^{it+1,it}_{k,i+1}$ 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 [2, 3], based on these results for LALR(1) context-free parsing and bottom-up evaluation, both using S^1 as dynamic frame.

3 Preliminary Results

Although our logical engine is still a prototype, preliminary results seem to improve preceding results. At the moment, we have compared our work with SLR(1)-like methods [4] in S^T with backtracking, and state-less bottom-up methods [3] in S^1. We have not observed additional cost in time due to control, and extra space cost is not relevant. Theoretic time and space bounds are cubic on the length of the input string in the worst case, although it is possible to characterize grammars with linear complexity. This has a practical sense because this class of grammars includes those DCG's whose context-free skeleton is LALR(1) and, in consequence, linear parsing can be performed while local determinism is present.

References

[1] B. Lang, "Towards a uniform formal framework for parsing", in *Current Issues in Parsing Technology*, M. Tomita ed., Ed., pp. 153–171. Kluwer Academic Publishers, 1991.

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

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

[4] 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.