Parsing TAGs with Prolog

Víctor Jesús Díaz Madrigal
Miguel Toro Bonilla
Dpto. Lenguajes y Sistemas Informáticos
Facultad de Informática y Estadística
Universidad de Sevilla
Avd. Reina Mercedes s/n - 41012-SEVILLA
E-mail: vjdiaz@lsi.us.es

Abstract

Among formalisms for the computation of syntactic description of natural language sentences, *Tree Adjoining Grammars* (TAG) play a major role. Several classical *Context Free Grammars* (CFGs) parsers have been redefined for TAGs, but it is not frequent to find in the literature references to parsing TAGs from a logic programming point of view. In this paper, we will concentrate on this direction, presenting a pure top-down left-to-rigth recognizer algorithm for TAG, using Prolog, that reduces the problems found in the translation of Lang's axiomatization. Actually, the algorithm is a parser that, given a grammatical input, produces the parse forest using a compact representation of every parse tree. Also, a rule representation of TAG's elementary trees is introduced in order to write grammars that can be compiled directly into Prolog predicates in a similar way that traditionally *Definite Clause Grammars* (DCGs) does respect to CFGs.

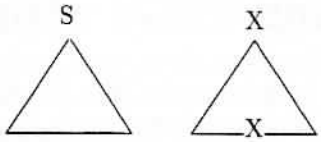
Keywords: Natural Language Parsing, TAG, DCG.

1 Introduction

In the literature, the grammatical formalism *Tree Adjoining Grammars* (TAGs) are propagated to be adequate for natural language description. The class of TAGs was first introduced in [JLT75]; since then, formal and computational properties of this class have been extensively investigated, and the linguistic relevance of TAGs has been discussed in the literature as well.

A TAG grammar is a tree generating formalism rather than a string generating system like traditional Chomky's grammars. It is defined by a finite set of trees composed by means of the operation of tree adjunction. With respect to Chomsky's hierarchy, TAGs are more powerful than *Context Free Grammars* (CFGs) but are a strict subset of *Context Sensitive Grammars* (CSGs). Several general classical parsers for CFGs (CKY [Har90], Early [ScJ88]) have been adapted to TAGs, but the excess of expresiveness of TAGs vs. CFGs results in worst time complexity.

Formaly, a TAG grammar G is a 5-tuple (V_T, V_{NT}, S, I, A) where V_T, V_{NT} are finite sets of terminal and non-terminal symbols, $S \in V_{NT}$ is the axiom symbol, and I and A are finite sets of elementary $(V_T \cup V_{NT})$-valued trees. Trees in I and A are called initial and auxiliary trees respectively and meet the following specifications (see figure 1): Internal (nonleaf) nodes in elementary trees are labeled with non terminal symbols. An initial tree has a root labeled by S and leaf nodes labeled by symbols in V_T. An auxiliary tree



æ

Figure 1: Elementary Trees.

has leaf nodes labeled by symbols in V_T with the addition of one node, called the foot node, having the same nonterminal label as the root node.

The path from the root node to the foot node of an auxiliary tree is called the *spine* of the auxilary tree. We say that $\beta \in A$ is a X-type auxiliary tree, when the label of his root node is the category X.

The main operation is the *adjunction* (see figure 2) that composes an auxiliary tree β with a tree α to produce another tree γ. Let α be a tree with a node labelled X and let β be an auxiliary tree with the root labeled with the same symbol X. (Note, that the foot node of β is X too, by definition). The tree γ is constructed as follows. The tree t dominated by X in α is excised, β is inserted at the position of node X in α, and then the tree t is attached to the foot node of β. We say that γ is a derived tree of α.

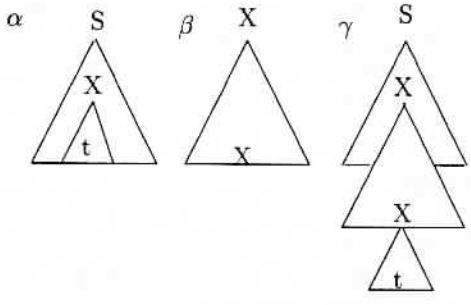


Figure 2: Adjunction operation.

In a TAG, a derivation is the process of recursive composition of elementary trees using the adjunction operation. Since adjunctions at different nodes can be performed in any order, we can adjoin derived trees into derived trees without affecting our arguments. In TAGs, unlike CFGs, the derived and parse tree of a grammatical input are different in general.

The yield $Y(\gamma)$ of a elementary (or derived) tree γ is the string of symbols composed with the leaves of the tree. The set of all initial trees modified by an arbitrary number of adjoinings, $T(G)$, is called the tree set of a TAG G. The language of a TAG, $L(G)$, is defined as the set containing all leaf strings of trees in $T(G)$.

One important characteristic of TAGs is *lexicalization* [JoS92]. Briefly, this means that every grammar structure (i. e. elementary tree), is anchored with a lexical item. In

general, this property is not fill for CFGs grammars. Lexicalization is a good computational property because it can be proved that reduces the complexity of parsers. On the other hand, lexicalization is a well linguistic motivated property.

2 Lang's axiomatization

The standard axiomatization of CFGs-rules, denominated *Definite Clauses Grammars* DCGs, consists of associating each CFG-rule with a definite clause where every categorial symbol is aumengted with two arguments that represent points of interest over the input string. It is well known that Prolog programs obtained from the DCG translation, implement top-down left-to-right recognizers or parsers. In a similar way as CFGs, [Sch90, Lan90], TAGs can be axiomatized with definite clauses. Now, each elementary tree is asociated with a definite clause and the set of definite clauses can be also interpreted in a top-down fashion.

Let $w = w_1, \ldots, w_n$ be the input string to parse and $G = (V_T, V_{NT}, S, I, A)$ a TAG grammar. In Lang's axiomatization four indices are required for every categorial symbol in the TAG's elementary trees (instead of two for CFGs). The four positions correspond to the boundaries of the strings to the left and right of a foot node of an auxiliary tree. The predicate $category(X, I, J, K, L)$ axiomatizes the fact that an auxiliary tree spanning the substrings $w_I \ldots w_J$ and $w_K \ldots w_L$, can be adjoined at a node labeled by X. Initial trees span a contiguous $w_I \ldots w_J$ and auxiliary trees span two substrings $w_I \ldots w_J$ and $w_K \ldots w_L$. The predicate $terminal(A, I, J)$ axiomatizes the fact that a terminal symbol A spans the substring from position I to J.

Suppose the elemetary trees of TAG grammar in figure 3,

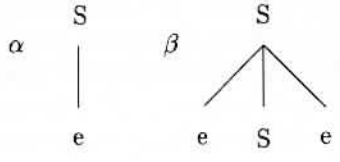


Figure 3: TAG Grammar.

the Prolog program resulting of Lang's translation, where $a0$ and $b0$ are names that identify elementary trees α and β, will be:

```
initial(a0,s,I,L) :-
   category(A,s,I,J,K,L),
   terminal(e,J,K).

auxiliary(b0,s,I,J,K,L) :-
   category(A,s,I,N,P,L),
   terminal(e,N,P),
   category(A,s,P,J,K,Q),
   terminal(e,Q,L).
```

```
terminal(T,I,J) :- ...    /* Recognize terminal T */

category(_,I,I,L,L).      /* No adjunction */
category(C,I,J,K,L) :-    /* Adjunction over auxiliary tree A */
  auxiliary(A,C,I,J,K,L).
```

This representation has several operational problems with respect to top-down parsing:

- As in DCGs, this algorithm loops on left recursive rules. This problem is acuter because of the duplication of predicates of the foot and root nodes in auxiliary trees. A two-step parsing strategy enables us to define a top-down interpretation of a lexicalized TAGs which will halt in all cases.
- If an auxiliary tree derives the string $w_1 w w_2$ where w is the derivation of its foot node, the actual derivation order is w_1, w_2, w. Therefore, the direction of parsing is not left-to-right and it implies that each internal category must predict in advance a segment w_2 of the input. In general, this way of working could be inappropiate because the prediction of w_2 does not have into account its precedence sequence w.
- The interpretation of the four boundaries arguments into actual lexemes by means of difference lists, as in DCGs, is not direct. Therefore, it is necesary to preprocess the input string to compute the relative positions of input lexemes.

3 An alternative representation

We will present a new representation of TAGs that tries to reduce the problems presented above. First of all, we will describe an alternative notation for trees that uses a word-based representation instead of the traditional graphical representation. The notation is as follows:

1. a stands for $a \in V_T$
2. $X(t_1 \ldots t_n)$ stands for the elementary tree having root $X \in V_{NT}$ and direct subtrees $t_1, \ldots, t_n$. When X has not children we will use the notation X instead of $X()$.

For technical purposes, we will express the word representation of $X(t_1 \ldots t_n)$ in a trivially equivalent form $X_L\, t_1 \ldots t_n\, X_R$. In other words, a category symbol X is split into two new non terminal symbols, X_L and X_R, that will divide the left and right side context of the symbol. The flat representation of α, in figure 3, will be $S_I\, e\, S_D$ and respectivelly $S_I\, e\, S_I\, S_D\, e\, S_D$ for β. This new notation has the minimal effect of duplicate the number of categorial symbols in a TAG grammar G. Size's grammar, $|G|$, is now in the order of $2|G|$.

The representation of an auxiliary tree will be of the form: $X_L\, Z_1\, X_L^*\, X_R^*\, Z_2\, X_R$ where Z_1 and Z_2 are sequences of symbols, being X_and X^* the root and foot symbols. If we observe carefully, we can establish that $Z_1\, X_L^*$ is just the left contextual tree, β_L, dominated by the root in β with respect to his foot node. Similarly the right context, β_R, will be $X_R^*\, Z_2$.

The adjunction operation can also be divided in two sides with respect to the spine of an auxilary tree. Supose that β is an X-type auxiliary tree with $Y(\beta) = w_1 X w_2$ being

$w_1, w2 \in V_T^*$. Let α be an initial tree that contains a category X with $Y(\alpha) = r_1 w r_2$, where $r_1, w, r_2 \in V_T^*$ and w is the string that spans the category X.

When we adjunct β in α at X, producing γ, we have $Y(\gamma) = r_1 w_1 w w_2 r_2$. We can see that w_1 (resp. w_2) is the string that spans the left (resp. right) contextual tree dominated by X in β. Both strings w_1, w_2 can be characterized with only two points of interest, i.e., we can reduce the positions associated to categorial symbols to two as DCGs does.

Briefly, the trees above, can be represented using flat notation as follows:

$$\alpha = Z_1\, X_L\, Z\, X_R\, Z_2$$

$$\beta = X_L\, Y_1\, X_L^*\, X_R^*\, Y_2\, X_R$$

where Z_1, Z, Z_2, Y_1, Y_2 are sequences of symbols (just the sequence of symbols in the flat representation). With this considerations, the next three CFG-based rules can be stated to translate the elementary trees:

$$R \leftarrow Z_1\, X_L\, Z\, X_R\, Z_2 \text{ rule for } \alpha$$

$$X_L \leftarrow Y_1\, X_L^* \text{ rule for } \beta_L$$

$$X_R \leftarrow X_R^*\, Y_2 \text{ rule for } \beta_R$$

where R is a new non-terminal symbol acting as the new axiom symbol. The flat representation of β is split into two rules representing left and right contextual sides. As the adjunction operation at a root and foot node of an auxiliary tree are equivalent, we can eliminate the reference associated to its root symbol in the rules.

When adjunction of β takes place in α, following the same considerations that we established above, we will have $X_L \rightarrow^* w_1$, $Z \rightarrow^* w$ and $X_R \rightarrow^* w_2$.

Consider a rule with a right hand side of the form $X_L\, Z\, X_R$ where X_L, X_R are the left and right symbol of the category X, and Z is a sequence of symbols not containing the symbol X_R. Every production applied over the symbols dominated by X_L must be applied to its respective symbol dominated by X_R in order to complete the adjunction operation. In other words, the adjunction operation looks like a mirror-sustitution operation between left and right parts of a same category. Operationally, we can use an adjunct argument over categorial symbols to control this restriction.

The adjunction argument can also be used to produce a compact representation of the parse tree. Given an elementary tree γ, we will represent its adjuction argument by means of a sequence based on its flat representation. Suppose that $S = [X1_L, X2_L, \ldots, XN_L]$ is the ordered subsequence respect to the flat representation of γ containing only the left side categorial symbols in γ. When constructing S for an auxiliary trees we will not consider the left side symbol of the auxiliary tree's root. If adjunction of an auxiliary tree β, named b_0, is performed at node XL_i of γ with $1 \leq i \leq n$, the symbol XL_i is substituted with $[b_0, S_\beta]$ where S_β will be the adjunction argument of β. When no adjunction is applied to XL_i the symbol XL_i in S is substituted with $[\,]$. Adjunct argument has two functions depending on the context where it takes place: in left context plays a productor role meanwhile in right context plays a consumer role.

This new axiomatization of TAGs permits us to translate elementary trees into definite clauses with the presence in every categorial symbol of only two (boundaries) arguments over lexemes and one additional argument to control the sequence of adjunctions. The predicate $terminal(T, I, J)$ axiomatizes that a terminal symbol T spans the substring from position I to J. We associate with every categorial symbol a predicate of the form

$category(X, C, I, J, A)$ that axiomatizes the fact that the C-context of categorial symbol X, where C ranges on the set $\{left, right\}$, spans the sequence $w_I, \ldots, w_J$ of input string. Argument A captures the adjunctions performed over the symbols dominated by X.

With this considerations, the interpretation of boundaries arguments as difference lists or positions is interchangable. A Prolog pure top-down parser for TAGs is obtained directly through this axiomatization. The direction of parsing is now strictly left to right, i.e., it is not needed to predict sequence of symbol in advance.

We present below the alternative translation of TAG grammar in figure 3. The two CFG-rules (left and right) associated to auxiliary trees are joined into one Prolog predicate using a new operator A ==> B whose operational semantics is nearly to the logic implication operator. This operator acts like a guarded selective operator that discriminates the left and right context side of auxiliary trees.

```
?- op(255,xfx,==>).

initial(a0,s,I,J,A) :-
  category(T,s,left,I,K,A),
  terminal(e,K,L),
  category(T,s,right,L,J,A).

auxiliary(b0,Ctx,s,I,J,A) :-
  (Ctx=left  ==> category(T,s,left,I,K,A),terminal(e,K,J)),
  (Ctx=right ==> terminal(e,I,K), category(T,s,right,K,J,A)).

A ==> B :- A,!,B.
A ==> B.
```

We present now the Prolog top-down left-to-right parser. The predicate *terminal* is translated as usually DCG does. The predicate *category* takes into account the possibilty or not of adjunction over auxiliary trees. When adjunction operation is performed over an auxiliary tree, its name is recorded in the front of its adjunction argument in order to complete the sequence of adjunctions.

```
terminal(A,[A|L],L).                  /* Recognize terminal A */

category(A,Ctx,XN,XN,[]).             /* No adjunction */
category(A,Ctx,X0,XN,[B|Adj]) :-      /* Adjunction over auxiliary tree A */
  auxiliar(B,A,Ctx,X0,XN,Adj).
```

4 TAGs compiler

We will describe how to compile a special flat representation of TAGs grammars into Prolog programs making use of the axiomatization presented before. This can be done using the standard predicate of Prolog to expand rules into predicates.

The source grammar will be the set of rules which define the TAG grammar. Each rule is related with an elementary tree, the head of the rules of initial trees is of the form $ini(I)$ where I is the name of the initial tree. Similarly, the head of the rules of auxiliary trees will be of the form $aux(A)$ with A being the name of the auxiliary tree. The body of

the rules consists of the word-based representation of the elementary tree, where terminal symbols are prefixed with the + operator and trees dominated by categorial symbols are separated with commas. The source grammar of figure 3 is as follows:

```
ini(a0) --> s(+e).
aux(b0) --> s(+e,s,+e).
```

First of all, to translate a rule we use the = .. predicate to transform the tree into a list separating the root node *Root* of the elementary tree from the trees $Trees$ dominated by the root. The treatment of *Root* and $Trees$ are different depending on the class of elementary tree being processing.

Then, given a list of symbols $Trees$, representing the flat notation of a tree (or a set of trees), the predicate $trans(Trees, P0, PN, X0, XN, TRTrees, Adj)$ is applied to genere another list $TRTrees$. Now the symbols in the $TRTrees$ list are aumengted with boundaries (using P0, PN, X0 and XN) and adjunction (using Adj) arguments. This predicate also splits categories in left and right sides, and marks the categorial symbols associated with the foot node in order to reduce the process of dividing the left and right context side of auxiliary trees.

Finally, a new predicate $divide(TRTree, BodyL, BodyR)$ is applied to construct actual rule's body and divide the left $BodyL$ and right $BodyR$ contextual sides of auxiliary trees. When applied over initial trees, only one argument is used because it is not necessary distinguish between the left and right side of the tree.

```
?- op(255,xf,+).

term_expansion((ini(A) --> B),(Head :- Body)) :-
  B=..[Root | Trees],
  trans(Trees,X0,XN,X1,X2,TRTrees,Adjs),
  divide(TRTrees,BodyTrees,_),
  Head=initial(A,Root,X0,XN,[Adj|Adjs]),
  Body=(category(Root,left,X0,X1,Adj),
        BodyTrees,
        category(Root,right,X2,XN,Adj)).
term_expansion((aux(A) --> B),(Head :- Body)) :-
  B=..[Root | Trees],
  trans(Trees,X0,XN,X0,XN,TRTrees,Adj),
  divide(TRTrees,BodyL,BodyR),
  Head=auxiliary(A,Root,Ctx,X0,XN,Adj),
  Body=((Ctx=left) ==> BodyL,(Ctx=right) ==> BodyR).

trans([],P0,PN,X,X,[],[]) :-!.
trans([+T|RT],P0,PN,X0,XN,[terminal(T,X0,X1)|Q],Adj) :- !,
  trans(RT,P0,PN,X1,XN,Q,Adj).
trans([T|RT],P0,PN,X0,XN,[Q1,Q2|RQ],[Adj|Adjs]) :-
  Q1=foot(T,left,X0,PN,Adj),
  Q2=foot(T,right,P0,X1,Adj),
  atomic(T),!,
  trans(RT,P0,PN,X1,XN,RQ,Adjs).
```

```
trans([T|RTree],P0,PN,X0,XN,Q,[Adj|Adjs]) :-
  T=..[Root|Trees],
  Q1=category(Root,left,X0,X1,Adj),
  Q2=category(Root,right,X2,X3,Adj),
  trans(Trees,P0,PN,X1,X2,RQ,Adj1),
  trans(RTree,P0,PN,X3,XN,R,Adj2),
  append(Adj1,Adj2,Adjs),
  append([Q1|RQ],[Q2|R],Q).

divide([foot(A,left,P0,PN,Aj)|Q],category(A,left,P0,PN,Aj),R) :-!,
  divide(Q,P,R).
divide([foot(A,right,P0,PN,Aj)],_,category(A,right,P0,PN,Aj)) :-!.
divide([category(A,Ctx,P0,PN,Aj)],category(A,Ctx,P0,PN,Aj),_) :-!.
divide([terminal(A,P0,PN)],terminal(A,P0,PN),_) :-!.
divide([foot(A,right,P0,PN,Aj)|Q],P,(category(A,right,P0,PN,Aj),R)) :-!,
  divide(Q,R,P).
divide([Q|RQ],(Q,RR),P) :-
 divide(RQ,RR,P).
```

5 Conclusions

Although several parsing algorithms have been defined for TAGs grammars, the study of TAGs parsing from a logic programming point of view is not frequent in the literature. Lang's axiomatization of TAGs establishes a relation between elementary trees and definite clauses. The key of this interpretation consists of associating four indices to categorial symbols that represent boundaries positions over the input. In this paper, we present an alternative representation which only uses two boundaries indices, just the number presented in DCGs. An aditional argument is needed in order to control the adjunctions composition of elementary trees.

In the context of a pure top-down parser, Lang's axiomatization presents several operational problems that the new approach reduces. Now, the Prolog translation permits a strict left to right parsing of the input and the interchangable interpretation of input lexemes with positions or difference lists. Given a grammar, the parser finds every parse tree for a grammatical input producing a compact representation of every parse tree. Adjunction arguments over categories also helps us in producing parse trees.

Finally, a PROLOG compiler of TAGs grammar is defined that uses this alternative approach in a similar way presented in DCG grammars respect to CFG-rules.

References

[AbD89] H. Abramson and V. Dahl. *Logic Grammars.* Springer Verlag, New York, 1989.

[JLT75] A. K. Joshi; L. .S. Levy and M. Takahashi. Tree Adjunct Grammars. *Journal of Computer System and Science*, 10(1), 136-63, 1975.

[JoS92] A. K. Joshi and Y. Schabes. Tree Adjoining Grammars and Lexicalized Grammars. In Nivat, M (ed.), *Tree Automata and Languages.* North Holland, 1992

[Har90] K. Harbusch. An Efficient Parsing Algorithm for TAGs. *28th Meeting of ACL*, Pittsburgh, 1990

[Lan90] B. Lang. Towards a Uniform Formal Framework for Parsing. In Tomita, M (ed.), *Current Issues in Parsing Technologies*. Kluwer Accademic Publishers, 1990

[Sch90] Y. Schabes. Mathematical and Computational Aspects of Lexicalized Grammars. Ph. D. Thesis. Departament of Computer and Information Science, University of Pennsylvania, 1990

[ScJ88] Y. Schabes and A. Joshi. An Early-Type Parsing Algorithm for Tree Adjoining Grammars. *26th Meeting of ACL*, Buffalo, 1988