IMPLEMENTATION OF PARSER USING TREE ADJOINING GRAMMAR FORMALISM FOR AAI & ROBOTICS APPLICATIONS

LINA RANI NATH*

ABSTRACT

Parser is a very crucial part of Natural Language Processing (NLP) through computational languages for syntactical analysis of sentences which includes identification of part of speech, phrases in terms of grammatical constituents, syntactic relations between phrases etc. A parser in machine translation analyzes sentences based on grammar formalism and builds a data structure corresponding to that grammar which indicates the syntactic correctness of a sentence. There are different formalisms available that can be used for analyzing sentences in natural language processing among which Tree Adjoining Grammar (TAG) formalism seems to be the most promising formalism for the analysis. A hybrid approach of parser known as Earley-type TAG parser gives better performance which can be further increased by using constraints in the operations of TAG formalism, reduction during the lexical analysis and applying heuristic rules during implementation.

The basic concept of TAG formalism i.e. the trees used for analyzing the sentences, the operations via which the trees are parsed, the recognizer algorithm, the use of TAG formalism for building a Earley-type parser and the constraints and heuristics to improve the performance of Earley-type TAG parser are elaborated in present paper.

KEYWORDS: Tree Adjoining Grammar (TAG), Parser, Earley-Type TAG Parser, Natural Language Processing (NLP).

INTRODUCTION

Machine translation is a rapidly growing field of Natural Language Processing (NLP) which needs to analyze sentences for translating it to one language from another because direct substitution of words is not a solution at all. Parser is the important part of this machine translation engine as it is responsible to find syntactical correctness of sentences based on which the translation of sentence in other language is generated.

Parsing of sentences has so many traditional approaches such as clause analysis, computational methods psycholinguistics which are mostly language dependent approaches.

*Principal Technical Officer-Applied Artificial Intelligence Group, CDAC Research & Development, Pune.
Correspondence E-mail Id: editor@eurekajournals.com
Apart from these approaches there are some other approaches of parsing known as LR Parser, Bottom-up parser, Top-down parser etc. LR parser is not suitable for parsing human languages as it doesn’t allow backtracking, it only provides a single correct parse where Bottom-up parsing is data driven and the algorithm doesn’t get affected with its first pass. These issues are solved in the Earley-type TAG parser as it allows lexicalization and is also a combined approach of LR Parser, Bottom-up parser, Top-down parser. It gives top-down information by prediction and it is a bottom-up parser which takes the advantage of lexicalization which simplifies its algorithm.

This Earley-type TAG parser is based on TAG formalism, which is a tree based formalism that comprises of two stages, firstly it collects trees for the lexicons present in the sentence for preparing set of trees & secondly based on the trees available on the set it implements the algorithm which gives output as derivation tree. Trees of TAG formalism are of two types: initial and auxiliary trees and operations of TAG formalism are of two types: substitution and adjunction operation.

This Earley-type TAG parser use a special type of tree traversal called dot traversal to parse the trees. The algorithm that is used for parser known as recognizer is based on the dot traversal of the tree. With each movement of dot position of a node of tree a state with 11 elements is produced and stored in the form of a chart. This state chart helps providing data to create the final derivation tree for parser also it helps to track the sentence parsing at any point of time. Sentence parsing always starts with an initial tree followed by substitution of initial trees & adjunction of auxiliary trees in that initial tree which gives a successful parse when the tree is fully traversed and gives a output called derivation output. A successful parse of the sentence gives a derivation output which indicates the correctness of the sentence based on the grammar. The trees used for parsing are minimized to reduce the parsing complexities, along with this the constraints applied on the complex operation like adjunction operation for the betterment of the parser performance.

The most striking feature of TAG approach is, it develops a system which is fully language independent which gives better output, takes less time as well as space by using constraints on the TAG operations and some heuristics while implementing the algorithm which are described in present paper.

The TAG parser and its mechanism which is briefed above are only responsible for the syntactic level analysis of the sentences. An enhanced analysis can be done by adding features to the tree nodes for semantic analysis along with the syntactic analysis to build a semantic TAG parser.

INTRODUCTION TO TREE ADJOINING GRAMMAR (TAG)

This paper describes TAG formalism and its use in syntactical analysis of sentences for natural language processing (NLP). Tree Adjoining Grammar is represented by trees terminal and non-terminal nodes where non-terminal nodes are divided into two subtypes that are leaf node and foot node. A tree is called auxiliary tree if it has a foot node else the tree is called initial tree, the root label and foot label of an auxiliary tree are always same.

TAG is a grammar formalism, set of 5-tuple which are described below:

\[ G = (T,N,S,I,A) \]

Where,

\[ T = \text{Set of terminal nodes} \]
\[ N = \text{Set of non-terminal nodes} \]
\[ S = \text{The start symbol} \]
I = Set of all initial trees
A = Set of all auxiliary trees

The basic trees and operations of TAG are used to identify the syntactical correctness of sentences. Parsing of sentence always starts from initial tree via combination of two operations i.e. substitution and adjunction. Substitution operation substitutes an initial tree with a frontier node and adjunction operation inserts an auxiliary tree into another tree. The node label where the operations are done must match with the tree’s root label that is getting substituted or adjunct.

An Initial represents basic valence relation between tokens which have terminal and non-terminal nodes. The non-terminal nodes can be anchored or can have a substitution symbol. A derivation always starts with an initial tree via substitution and adjunction operations.

An Auxiliary Tree is a tree having exactly one frontier node marked as foot node and the label of foot node & root node must be the same. Except these two features of auxiliary tree, it is same as initial tree. Adjunction operation is carried out with an auxiliary tree which allows recursion at the foot node.

After the introduction of the TAG grammar, the basic operations which are essential features of TAG for parsing natural language will be described.

Substitution operation is carried out at the frontier node of a tree which is having a substitution symbol. A node marked with a substitution operation does not allow an adjunction operation to perform at that node. Only initial trees are substituted at that node and an initial tree which will be substituted at the frontier node must have a root node labeled same as the frontier node. It is a simple attachment operation where the frontier node gets replaced with the initial tree as shown below:
Adjunction operation can be called as insertion operation. In adjunction operation an auxiliary tree gets inserted inside another tree. The node where adjunction operation is carried out must be labeled same as the foot node of the auxiliary tree. The part of tree attached below the frontier node of the tree (where adjunction operation is done) gets attached below the foot node of the auxiliary tree and then the whole auxiliary tree gets replaced with the frontier node as shown below in the figure.

**IMPLEMENTATION OF PARSER USING TREE ADJOINING GRAMMAR (TAG)**

In this section introduction & implementation of Parser using TAG formalism will be elaborated. The whole TAG parser algorithm is based on tree traversal. This tree traversal is called dot traversal. As shown in the figure no. 9, in dot traversal each terminal and non-terminal node of a tree is marked with 4 dots i.e. Left Above (la), Left Below (lb), Right Above (ra) and Right Below (rb). While parsing a sentence the tree is traversed from top to bottom, let to right and then bottom to up. In each dots different operations are done to traverse the tree and a state of 11 elements created which are stacked on memory and further used to build derivation tree.
As shown in the above figure the dot traversal of a tree is described as below [5]:

- If the dot is at left above position of any non-terminal node then dot is moved to the left below position.
- If the dot is at left below position of any non-terminal node then dot is moved to the left above position of its leftmost child.
- If the dot is at left above position of any internal node, the dot is moved to the left above position, if the dot is at left above position of a leaf, then the dot is moved to the right to right above position of the leaf.
- If the dot is at right below position of a node, then the dot is moved to the right above position of the same node.
- If the dot is at right above position of a node, there can be two cases:
  - If that node is having a right sibling, then the dot should be moved to the left above position of the right sibling.
  - If that node does not have a right sibling, then the dot should be moved to the right below position of the parent node.

The dot traversal is described with the help of an algorithm named as parser recognizer. The parsing algorithm is elaborated with a flow chart in this current paper which uses two data structures: state and state set. State set is the set of states. State is a set of 11 elements which is filled with each traversal of the dot position creating a state and further the state is pushed into a stack called state set for further reference. These elements are described below [6]:

A state $S$ is defined as a 11-tuple, 

$$[T, N, S, P, \text{Left}, \text{ft}, \text{fr}, \text{star}, t^\sim, b^\sim, \text{subs}]$$

where:

- $T$: is the name of the dotted tree.
- $N$: is the name of the node in the tree $T$.
- $S$: is the side (left/right) of the symbol the dot is on;
- $P$: is the position (above/below) of the dot;
- Star: is an address in a. The corresponding node in a is called the starred node.
- ft (foot left), fr (foot right), $t^\sim$ (top left of starred node), $b^\sim$ (bottom left of starred node) are indices of positions in the input string which star used while a substitution or adjunction operation is done.
- Subs: a Boolean that indicates whether the tree was substituted
The above flowchart describes the Earley-type recognizer implemented using TAG formalism. Different functions that are invoked while dot traversal of a true to parse a sentence are described with pictures in current paper:

**SCANNER:** Scanner scans the terminal nodes and matches with the input string. If the dot is at left above position of a terminal node without having substitution or foot node symbol then it checks for a match of input string. If it matches then the
dot position is shifted to the right above position of the node and a state is pushed to the stack. Number of identified token is increased. If not matched then the current operation is terminated and a new path of parsing started with new trees.

Figure 7. Dot movement during Scanner Operation

**MOVE DOT DOWN:** If the dot is at the left below position of a non-terminal node and it is not a foot node then the dot is shifted to the left above position of its leftmost child.

Figure 8. Dot movement during Move dot down Operation

**MOVE DOT UP:** If the dot is at the right above position of a non-terminal node and no adjunction operation is predicted at that node then the dot is shifted to the left above position of its sibling or moves to the right above position of its parent node.

Figure 9. Dot movement during Move dot up Operation

**LEFT PREDICTOR:** If the dot is at the right above position of any node which is not having a substitution symbol, which is not an anchor node [7] or empty node[8], which is not a foot node and which is not having any NA constraint [9] then an adjunction operation prediction is done.

If an auxiliary tree is present having the same root label then the dot position is shifted to the left above dot of the auxiliary tree’s root node else the dot position is shifted to the left below position of that node.
LEFT COMPLETOR: Only if an adjunction operation is being carried out and the dot is at the left below position of the foot node then the dot is shifted to the left below position of the node where the adjunction operation prediction is done.

RIGHT PREDICTOR: If an adjunction operation is being carried out and the dot is at the right below position of the node where the prediction for the adjunction operation is done then the dot is shifted to the right below position of the foot node of that auxiliary tree else the dot is moved up to the right above position of that node.

RIGHT COMPLETOR: If the dot is at right above position of a node except of root node of the tree then the dot is shifted to the sibling’s right above position else it is shifted to the right below position of its parent node. Only in case of root node if an adjunction or substitution operation is done then it shifted to the right above position of the node where the operation is predicted.
Figure 13. Dot movement during Right Completor Operation

Pictures will be here also brief description with state chart.

With each successful traversal a state is generated and pushed into the stack which helps to create the derivation output.

Adjunction operation is a complex operation which increases the time and space complexity of the recognizer. Some heuristics rules and constraints can be helpful to optimize these issues. Speed of parser can be increased by applying some heuristics in the implementation logic. The things that are responsible for these complexities and can be optimized are:

- **NUMBER OF TOKENS IN THE INPUT SENTENCE:** They can be minimized using pre-processing of the input text. Reduction of token will minimize the number of complex adjunction operations. Hence it will improve the parsing quality.

- **NUMBER OF INITIAL AND AUXILIARY TREES IN THE GRAMMAR:** Number of trees in the grammar may not be minimized, but they can be reduced by using linguistic rules on the structure of the sentence which may lead to better parsing.

- **NODES IN ELEMENTARY TREES:** Number of nodes in the elementary trees can be reduced.

- **ADJUNCTION OPERATION CONSTRAINTS:** Adjunction operation can be restricted by applying constraints on the nodes such as null adjunction. If a node having null adjunction constraint, an adjunction operation will not be predicted at that node. Only this operation can be done on the selective nodes. The less is the adjunction operation the more is the speed.

- **RECURSION ON FOOT NODE:** Recursion of the same tree at foot node should be restricted and number of adjunction at one node should also be restricted to increase the speed and quality of the TAG Parser.

- **CREATING TREES:** Creating initial and auxiliary trees based on the POS [9] categories of the lexicon’s instead of creating them corresponding to the lexicon will also optimize the complexity of TAG Parser.

A parsing starts at the left above position of the root node of an initial tree and after traversal it ends at the right above position of the root node of that tree. Then it is called a successful parse. Each dot traversal is stored as state and a state chart are built. This state chart helps creating derivation tree as output of parsing. A successful parse of sentence indicates the syntactical correctness of a sentence based on the elementary trees present in the grammar.
CONCLUSION

Earley-type Parsing using TAG formalism is elaborated in current paper. Using this type of parsing technique a system can be developed which can syntactically analyze the structure of a sentence any language. The beauty of TAG based parser is that using TAG formalism we can build a language independent parser for natural language processing. The parser that is described here is a two stage parser which first allows collecting trees for the set of lexicon and then according to the available set of trees it parses the sentence. Thus it improves the quality of performance and using the constraints and heuristic rules described in this paper it helps to develop an optimized system.

REFERENCES