

Utilization of a Novel Ordered Context Free Grammar for Object Based Parsing and Unification Technique

Jafar Rizvi, Dr. Mutawarra Hussain, Naeem Qaiser

JafarRizvi@Gmail.com, mutawarra@pieas.edu.pk, and naim_qaiser@yahoo.com

DCIS, Pakistan Institute of Engineering and Applied Sciences (PIEAS), Islamabad, Pakistan.

Abstract: A novel ordered-context-free-grammar (OCFG) is presented, in which each rule has a particular order among other rules. The grammar is utilized in an object-based parsing technique, in which a rule object is applied sequentially from right or left, depending upon rule type. Input text is tokenized into lexical categories. Rule object compose mother object using its constituent objects taken from the array and inserting mother node back in the array. Parsing is complete if the array contains only the goal symbol. Formation of functional-structure of mother objects by unification of features and functional of its constituents is also done simultaneously as the constituent-structure is created. The technique is found to have lesser space requirements and better time efficiency.

Keywords: Natural Language Processing, Parsing, Ordered Context Free Grammar.

1. INTRODUCTION

Parsing is to find constituent structure of a sentence in a language using grammar representing the language and it is one of the important requisite for various natural language processing applications. Many efficient rule-based and statistical parsing techniques for the context-free-grammars exist. This paper explores a novel method that has advantages of object-oriented design and is especially well-suited and adapted for parsing and unification requirements of lexical functional grammar.

1.1 Lexical Functional Grammar

Lexical functional grammar (LFG) is an approach for the modeling of grammar of any natural language using universal principles. Its linguistic formalism is strong and flexible to handle diverse linguistic phenomenon and at the same time theory has mathematical vigor. It contains parallel syntactic structures namely argument structure (a-structure), constituent structure (c-structure) and functional structure (f-structure) [1, 2, 3] etc. The c-structure rules are just like context-free grammar (CFG) rules. These rules have an additional functional schemata attached to symbols on the right hand side of the rule. These functional schemata play an important role, while unification is done. Given an input sentence, in LFG, for this sentence to be grammatical it must have a parse tree, called c-structure and at the same time it must have a consistent, complete and coherent features structure after the unification.

The main feature of the method presented in this paper is that parsing is achieved by composing new object using its

constituent objects and features of new object are composed by unification of the features of its constituents.

Although the method described in this paper is primarily for LFG based grammars, but it is generic in nature and can be utilized for parsing other grammars and can be used with other unification based theories.

1.2 Review of Basic Parsing Methods

One of the traditional and basic algorithms of parsing a context-free grammar is recursive-descent (or predictive) parsing that needs to calculate the 'first' and 'follow' sets and it may be implemented for 'k' look-ahead symbols. For this method, the grammar must be in the form that the first terminal symbol of each sub expression must give information about which rule to use and therefore the grammar must have neither left recursion nor left factoring problem. This is termed as LL(k) or top-down parsing. Another basic method is shift-reduce, LR(k) or bottom-up parsing. The CFG is used to find deterministic finite automata (DFA) states and parsing table. The 'shift,' 'reduce' and 'go-to' are the basic operations for this parsing scheme which depend on the look-ahead symbol(s) and the state of DFA. [4, 5]. A more detailed categorization of parsing methods can be found in [6], which in addition to explaining various practical issues regarding parsing techniques also contains an annotated bibliography on parsing techniques.

Another division of parsing methods is rule-based parsing and probabilistic/stochastic parsing. Probabilistic parsing has been gaining popularity during recent years especially for natural language applications, yet rule-based approach is accurate. Combined rule based and statistical parsing approach is also gaining success. In one such attempt [7], at first a sentence receives rule-based analysis, and if rule-based analysis fails then statistical techniques are applied to a partial parse from the rule based parser. In this paper, rule-based bottom-up parsing along with unification is achieved through object composition.

1.3 Review of Object Oriented Methods

In this section we present a brief review on some of the attempts that has already been made on 'object oriented parsing' methods. Davis presented [8] an object-oriented approach for building a recursive descent parser using a grammar that is left recursive and therefore it may result in ambiguous sentences. The 'oops' parser generator of Kühl and Schreiner [9] requires LL(1) class of grammar. It has rule and node objects which require 'look-ahead' and 'follow-sets', hence they implemented the traditional

LL(1) algorithm in an object-oriented way. Main feature is that each node object knows its own look-ahead and follow-sets. The CUP parser presented in [5] implements LALR parsing method using object-oriented methodology. In another object oriented parsing scheme of Yonezawa and Ohsawa [10], a number of computing agents are assumed for each of terminals and non-terminal symbols in the grammar that work in parallel and send messages to each other that are partial parse trees. The main feature is that these computing agents can work in parallel but a computing agent has to wait for message from other computing agents until it gets partial-parse tree which it can process and send that tree to other computing agents. Thus in their algorithm the object-orientation is for computing agents that can work in parallel. They showed that algorithm can work for many different applications.

Our brief review of study reveals that in most of the work already done, the traditional algorithms are represented through object-oriented methodology. The method we are presenting is achieving parsing, unification using object-oriented constructs. The rule object, when given the array of input symbols, is constructing mother objects by applying itself. Each object created is representing the composition hierarchy without the use of traditional tree structure.

2. OBJECT BASED PARSING

Our object based parsing scheme utilizes a novel ordered-context-free-grammar (OCFG). Therefore, before describing the method we define the grammar. It is the extension of context free grammar and each rule OCFG has an order and type associated with it, just like probabilistic context free grammar is an extension [11] of and has a probability value associated with each rule.

2.1 Ordered Context Free Grammar

We present here the formal definition of novel ordered context free grammar (OCFG). This grammar will be utilized in our object oriented parsing technique. The ordered context free grammar is a four tuple $\{W, N, S, R\}$, where $W = \{w_1, w_2, \dots, w_u\}$ is a set of terminal symbols like words in a sentence, $N = \{n_1, n_2, \dots, n_v\}$ is a set of non-terminal symbols like noun-phrase in a sentence, $S = \{n\}$ is a set of one symbol, which is the goal symbol, and $R = \{r_1, r_2, \dots, r_w\}$ is a set of grammar rule with order. Each rule 'r' is a context free grammar rule and has an order number associated with it. In addition to that rule type is also specified. A left-rule is applied from left to right, and a right-rule is applied from right to left. A recursive rule is applied recursively. A rule can have neither empty left-hand side nor empty right-hand side. This means empty productions are not allowed.

2.2 Proposed Method

Method we adopted is utilizing the basic theme of bottom-up parsing technique but it does so without building the deterministic finite automaton, parsing table or use of

stack for parsing. It also do not use traditional tree-structure to store nodes as these are parsed but rather daughter nodes (symbols) are stored as constituent part of the mother node. Therefore it is a novel technique for parsing the sentences successfully.

Figure 1 shows the objects interactions during the parsing. The sentence to parse is first divided into strings of terminal tokens or lexical symbols by a 'tokenizer' object. The tokenizer does so by handling punctuations and identifying word boundaries.

After the sentence is divided into token strings by tokenizer, the token strings are fed to syntactic category identifier (SCI). The job of SCI is to find the syntactic category of each word and form a token object. The token object has a particular syntactic type and has an attribute-value matrix (AVM) representing features of that word. This AVM is the representation of f-structure of LFG. The SCI sends token string to lexicon object, which returns type and attributes associated with the corresponding word object. If the word is not available in the lexicon it takes help of the morphological analyzer to check whether other word forms may serve the purpose. The output of SCI is the token objects having syntactic types like nouns, verbs. These token objects are output from SCI in the form of an array.

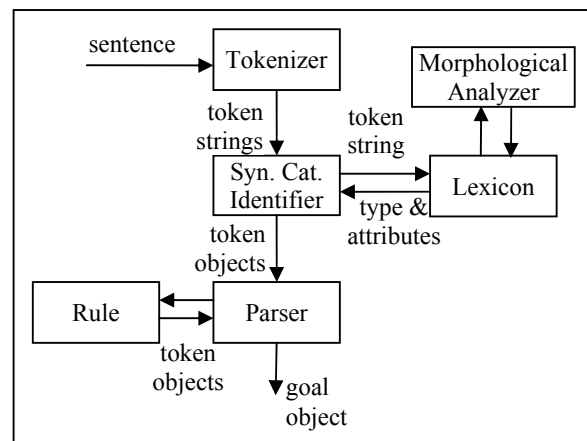


Figure 1: Objects Interactions during Parsing

The parser sends these token objects array to each of the rule one by one in order. Each rule in a grammar knows that if the objects referenced by symbols on right-hand side of the rule are available, then those can be used to construct the left-hand side object. The rule object searches for its RHS objects from left-to-right or right-to-left depending upon its own type. If the RHS symbols are found in the tokens array, then those symbols are taken out from the tokens array, the LHS symbol object is created and put back in the array in the same position from where constituent objects were taken out. The parser stops if either no rule can be applied to input token objects and reports failure to parse or if the goal symbol is found then it reports successful parse.

3. METHOD VALIDATION

In order to validate the proposed method a computer program is written and tested on various arithmetic and natural language parsing applications. In this section we present parsing of English sentence using a small ordered context free grammar for English. To show that method not only parses but also takes care of association involved in arithmetic expressions, a couple of examples of arithmetic expression parsing are also given.

3.1 Example 1: English Grammar

We present some examples to clarify the method. Consider the example tiny grammar of English in (1) and the example sentences in (2) and (3). Because the rule will construct object on LHS using objects on RHS, therefore the arrow is from right to left (\leftarrow), instead of traditional arrow in a rule from left to right (\rightarrow). The order of application of rules is important and therefore the goal symbol, which is the sentence, S, in this example, is at the bottom. This means that the goal symbol is to be constructed last. In the grammar presented only non-recursive left type rules are presented. The left-type rule applies itself from left to right on the symbols array. It should be noted that the grammar of English presented here is for demonstration of method only. A much larger grammar of English is developed and various constructions in English have been tested using the method. The symbols used in grammar (1) are shown in Table 1. We will analyze sentence (2) and (3) using this grammar.

- (1)
- | | | |
|---------------------------|---|------|
| NP \leftarrow Det N | 0 | Left |
| NP \leftarrow N | 1 | Left |
| PP \leftarrow P NP | 2 | Left |
| VP \leftarrow Aux V NP | 3 | Left |
| VP \leftarrow Aux NP PP | 4 | Left |
| S \leftarrow NP VP | 5 | Left |

(2) A girl is carrying a bucket

(3) There was a woman in the kitchen

Table 1: Symbols used for grammar (1)

V	Verb	Det	Determiner
N	Noun	NP	Noun Phrase
P	Preposition	PP	Prepositional Phrase
S	Sentence	VP	Verb Phrase
Aux	Auxiliary		

The sentence string is converted to token with syntactic categories. As shown in Figure 2, we apply rules one by one, in the given order. Rule # 0 of grammar shown in (1) is applied first. To apply a rule, the objects on right hand side of the rule are searched in the array, e.g., here objects Det and N are searched in array from left to right. Then a NP object is created that is composed of a Det object and

an N object, as shown in Figure 2. To construct an object, its constituent object are taken out from the array by the rule object, the mother object is created that is composed of its daughter objects, which is again put into the array of symbols.

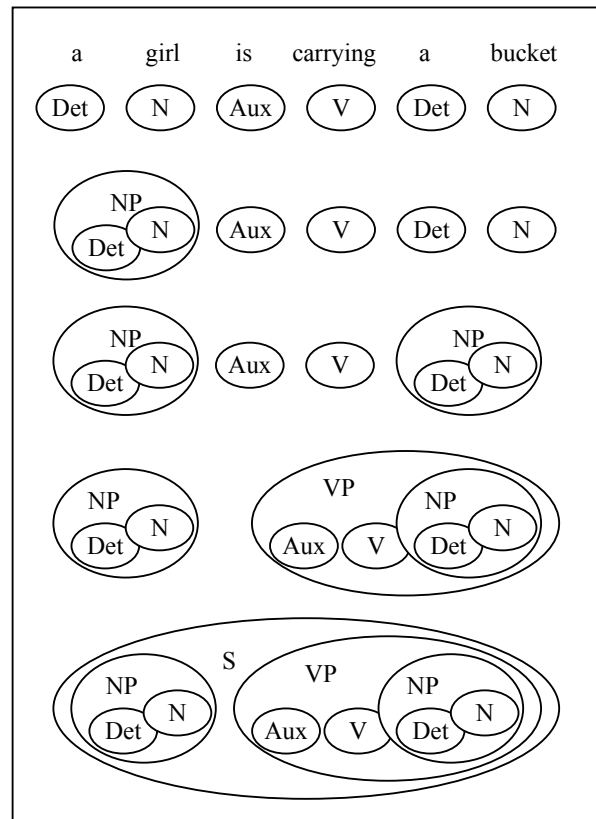


Figure 2: Object-based Parsing of Sentence (2)

After a rule has been applied, we again start from rule # 0 and see if it can be applied. It is applied again and another NP object is created. The rule # 0 can not be applied any more. The rule # 2 and 3 are not applied for current symbols array. The rule # 4 can be applied because we found Aux V and NP objects. These objects are taken out from the array and new VP object is created. Then finally S object is created, which represent the sentence. No special tree structure is utilized. Each object formed is referring to its constituent objects and thus making the tree automatically. If the goal symbol is found, the parse is successful and the goal object can be used to traverse the parse structure. The goal object contains its constituents, like here S contains VP and NP objects as shown in Figure 2. The VP contains Aux, V and NP and hence the required parse is available.

If an object can be constructed by different constituents, we have overloaded constructors for that object. Which constructor is going to be called depends on the number and type of arguments passed to constructor function of that object. For example, NP in grammar (2) can be constructed either by using Det and N objects or only by N

object. Therefore NP has two overloaded constructors, one takes Det and N type object arguments and other takes N type single object argument.

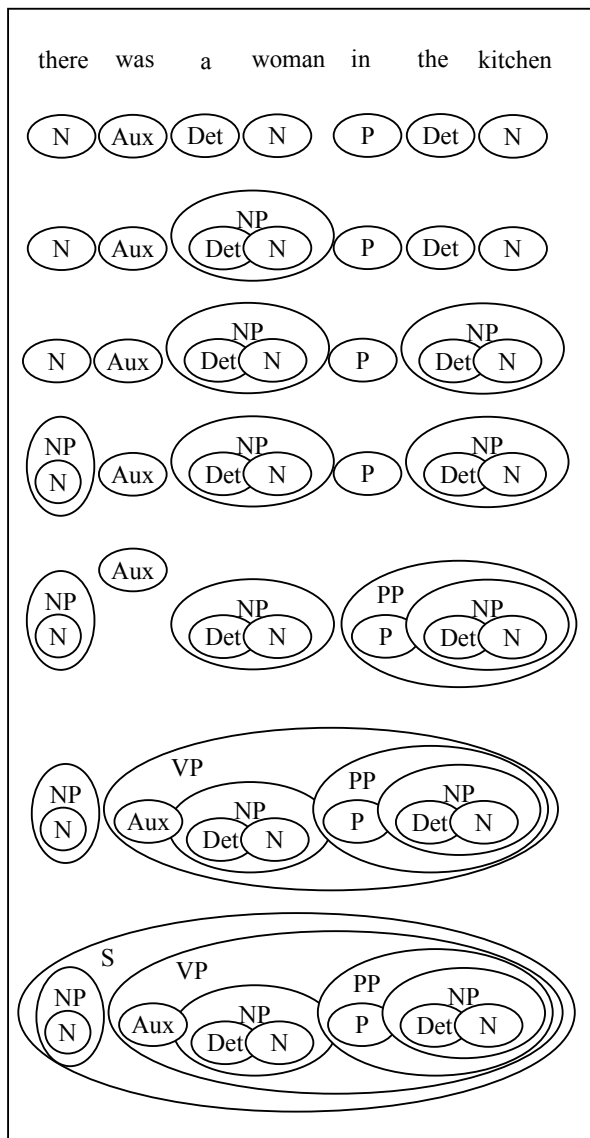


Figure 3: Object-based Parsing of Sentence (3)

The parse structure formed by applying grammar (1) on sentence (3) is shown in Figure 3. Note that rule # 0 is applied twice from left to right before converting first N using rule # 1 to a NP. This shows that order of applying rule is important. If rule # 1 is applied ahead of rule # 0, then three N will be converted to NP and parse will not be successful. So while writing rules main concern about the order in which phrases are constructed is the key idea behind this parsing scheme.

3.2 Example 2: Arithmetic Expressions

We take another example of arithmetic expressions to show that precedence and associativity problem involved in mathematical expressions may also be dealt with this algorithm. This grammar is shown in (4) and is considered highly ambiguous [4] if rule order is not important, but our scheme considers each rule in the given order and hence it parses arithmetic expressions quite successfully. Precedence is handled by applying rule in a particular order. The expression having operator * or / will be evaluated ahead of expression involving + and -, because the rule for * or / has lower order number. Associativity is handled by rule type: left or right. The sub-expression in braces is handled by recursive rule type. Therefore the grammar given below specifies the rule, its order and type.

$E \leftarrow Id = E$	0	Right/Recur	(4)
$E \leftarrow Id$	1	Right	
$E \leftarrow N$	2	Right	
$E \leftarrow (E)$	3	Left/Recur	
$E \leftarrow E O2 E$	4	Left	
$E \leftarrow E O1 E$	5	Left	

Right type rule means that it applies from right to left. Left type rule applies from left to right and recursive rule takes sub-expression out from tokens array and applies recursively. The symbols used in grammar (4) are shown in Table 2.

We parse two expressions to show steps involved and the way by which method can handle various requirements. In the same way the input strings is converted to basic token objects. The parser sends tokens array to rules one by one, which applies itself.

Table 2: Symbols used for grammar (4)

Id	Any identifier that starts with a letter
N	Numeric Value
O1	+ (plus), - (minus)
O2	* (multiply), / (divide)

The rule # 0 has recursion. Therefore, it will find the first two object Id and O3 and if found, it will send all tokens following O3 back to parser and ask the parser to find its parse. The parser will take this shorter array and send back to same set of rules one by one and try to reduce it to E. This smaller array is shown using square brackets in Figure 4. In recursive call, the rules are now applied on this array until, it reduces to an expression. So by using rule # 2 from right-to-left the N object is used to construct E. Then using rule # 1 twice, two more E objects are created using Id objects. Note that application of rule # 4 ahead of rule # 5 takes care of precedence requirement. Finally, the expression is constructed successfully for the recursive call the expression return in the previous array as an E object. Which by using rule # 0, constructs the final expression for assignment.

A = B + C * 5.3	Input expression
Id = Id O1 Id O2 N	Token objects
Id = [Id O1 Id O2 N]	Rule 0 : Recursion
Id = [Id O1 Id O2 E]	Rule 2
Id = [Id O1 E O2 E]	Rule 1
Id = [E O1 E O2 E]	Rule 1
Id = [E O1 E]	Rule 4
Id = [E]	Rule 5
Id = E	Recursion returns
E	Rule 0

Figure 4: Object-based Parsing of an expression

The grammar (4) is applied to another expression having sub-expression using braces and its parse is shown in Figure 5. The sub-expression within braces is again a candidate for recursive call. Which guarantees that first the inner sub-expressions will be evaluated ahead of out-side brace expression.

Although, this parsing scheme is basically developed for the LFG based parsing and unification for natural language processing, but the example of arithmetic expression is presented here to show that this parsing method can be powerful enough to have efficient parsing for other applications, like programming language parsing, as well.

3 + 4 * (6 - 7) / 5 + 3	Input expression
N O1 N O2 (N O1 N) O2 N O1 N	Token objects
E O1 E O2 (E O1 E) O2 E O1 E	6 times, Rule 2
E O1 E O2 ([E O1 E]) O2 E O1 E	Rule 3 (Recur.)
E O1 E O2 ([E]) O2 E O1 E	Rule 5
E O1 E O2 (E) O2 E O1 E	Recursion returns
E O1 E O2 E O2 E O1 E	Rule 3
E O1 E O2 E O1 E	Rule 4 : Left
E O1 E O1 E	Rule 4
E O1 E	Rule 5 : Left
E	Rule 5

Figure 5: Object-based Parsing of an expression

4. UNIFICATION

Each terminal symbol object has its f-structure associated with it, which it gets initially its features from the lexicon. These f-structures are represented as attribute-value matrix as part of each symbol object. Whenever a rule is applied a new symbol object is created. The new symbol has its own attribute-value matrix (representing its f-structure) that has unified effect of its constituent symbol objects. In our object oriented scheme, each symbol when gets created through constructor call, the unify function is automatically called by the constructor function and this function unifies all the f-structures of the constituents nodes according to functional schema of the daughter nodes. This means as the nodes are being created, their f-structures are also being formed by implicit call to unify function. To show how functional schema attached with each node influences unification, an example is given below for unification of a simple sentence shown in (5.)

The parse tree of the sentence called c-structure in LFG is shown in Figure 6.

(5) He walks

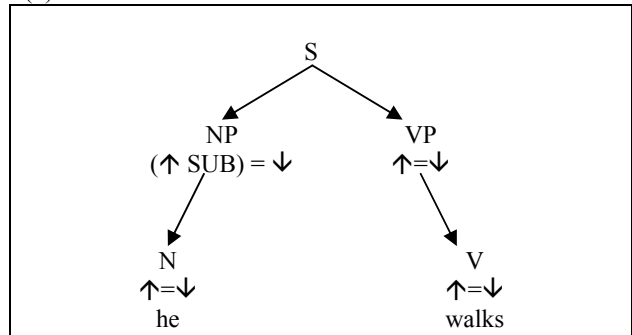


Figure 6: C-Structure of sentence 'He walks'

Note that each node in LFG based c-structure has attached with it a functional schema. The up arrow \uparrow refers to the mother node, while the down arrow \downarrow refers to the current node. $\uparrow=\downarrow$ means direct unification, i.e., features or functions of mother node are the same as that of current node. The f-structures of noun 'he', and verb 'walks' are shown in Figure 7 and 8.

PRED	'he'
NUM	SG
PERS	3

Figure 7: F-Structure of noun 'he'

PRED	'walk<...>'
TENSE	PRES
SUBJ	NUM SG
	PERS 3

Figure 8: F-Structure of verb 'walks'

NP object is the mother of N and has the same f-structure due to schema $\uparrow=\downarrow$ attached with N and similarly VP is the mother of V and has the same f-structure. But S is the mother object formed with two constituent objects NP and VP and its f-structure is formed by unification of features and functions from the f-structures of both NP and VP. All the attributes of VP will be added directly to the f-structure of S, but for NP the attributes of NP will be added to the value f-structure of the attribute SUBJ in the f-structure of S as indicated by the schema $(\uparrow \text{SUB})=\downarrow$ attached with the NP. Figure 9 shows the unified f-structure of simple sentence shown in (5.) We observe that NUM and PERS attributes in the f-structure of sentence come from both the noun 'he' and the verb 'walks'. The rule of unification suggests that if the same attribute has the same value from both the daughter f-structures, then only one is kept and other is discarded and if the same attribute has different value, then the resultant f-structure is not consistent and indicates that the sentence is not

grammatical. This means that consistency condition on f-structure is checked as they are being formed.

PRED	'walk<...>'
TENSE	PRES
	PRED 'he'
	NUM SG
SUBJ	PERS 3
	NUM SG
	PERS 3

Figure 9: Unification of f-structures of NP and VP

The completeness and coherence conditions requirement of f-structure are checked after the goal symbol, i.e., sentence has been formed. The completeness condition checks that all the functions mentioned in the argument-structure of the predicate of f-structure are present in the same f-structure. The coherence condition checks that every argument function present in the f-structure is designated by its own predicate.

5. CONCLUSIONS

The method is parsing successfully by the interaction of rule objects with the symbols array until the goal symbol is found. The rule objects are applied in a sequence indicated by the ordered context free grammar. A particular order of rules is used to disambiguate the situations when two or more rules may be applied simultaneously. Therefore, the algorithm does not need deterministic finite automaton or any kind of parsing table to decide what action to take next. Recursive rules are for sentence within sentences. The left/right type of rule is for associativity. The parsing is achieved through object composition from its constituents and construction of features and function structure is done at the time of creation of object by unification of features and functions of its constituent objects.

The parsing method presented in this paper does not require calculations and storage requirements for finding a parsing table. It needs to store only the 'n' symbols objects in an array and 'm' rule objects. Therefore space efficiency of this method is good. For time complexity, we see that for each rule 'm', having 't' symbols on its RHS we need to search linearly the array of 'n' symbol objects. The rule finds consecutive 't' symbols and constructs a new symbol. In each pass number of symbols reduce by one or more. It means that there are maximum n such passes. Therefore, the algorithm has a worst case time complexity of the order of $O(m*t*n^2)$. This time efficiency is better than other methods [12], because value of t is

small compared with n. In practice, the actual time efficiency of the method can be enhanced if the rules having more probability of application are given least order, but it can be done for those rules whose order has no effect on grammar.

REFERENCES

- [1] J. Bresnan (ed), In *The Mental Representation of Grammatical Relations*, MIT Press, 1982.
- [2] J. Bresnan, *Lexical Functional Syntax*, Blackwell, 2001.
- [3] S. M. J. Rizvi and M. Hussain, "A Novel Approach to Account Morphological Behavior of Urdu Verbs to Model Urdu Tenses Using LFG", In *Proceeding of International Multi Topic Conference INMIC 2002*, IEEE, 2003.
- [4] A. V. Aho, R. Sethi, and J. D. Ullman, *Compilers – Principals, Techniques and Tools*, Addison-Wesley, 1986.
- [5] A. W. Appel, *Modern Compiler Implementation in Java*, Foundation Books, New Delhi, 2001.
- [6] D. Grune and C. Jacobs, *Parsing Techniques – A Practical Guide*, Ellis Horwood Limited, 1994.
- [7] S. Riezler et. al., "Parsing the Wall Street Journal using a Lexical Functional Grammar and Discriminative Estimation Techniques", In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, Philadelphia, 2002.
- [8] M. S. Davis, "An Object-Oriented Approach to Constructing Recursive Descent Parsers", *SIGPLAN Notices*, ACM, pp. 29-35, February 2000.
- [9] B. Kühl and A Schreiner, "An Object-Oriented LL(1) Parser Generator", *SIGPLAN Notices*, ACM, pp. 33-40, December 2000.
- [10] A. Yonezawa and I. Ohsawa, "Object-Oriented Parallel Parsing for Context-Free Grammars," In *(Adriaens & Hahn)*, pp. 188-210, 1994.
- [11] Y. Yao and K. T. Lua, "A Probabilistic Context-Free Grammar Parser for Chinese," *Computer Processing of Oriental Languages, Vol 11, No 4*, pp. 393-407, 1998.
- [12] L. Lee, "Fast Context-Free Grammar Parsing Requires Fast Boolean Matrix Multiplication", *Journal of the ACM 49(1)*, pp. 1-15, 2002.