***Dialectologia* 7** *(2011), 75-92.*
ISSN: 2013-2247

Received 14 September 2010.
Accepted 21 December 2010.

PARSING OF KUMAUNI LANGUAGE SENTENCES AFTER MODIFYING EARLEY'S ALGORITHM

Rakesh PANDEY[1] & Hoshiyar S. DHAMI[2]
[1]Amrapali Institute of Technology and Sciences, Haldwani, Uttrakhand (India)
rakeshpandeyaits@gmail.com
[2]Kumaun University, Almora Uttrakhand (India)
profdhami@rediffmail.com

Abstract

Kumauni language is one of the regional languages of India, which is spoken in one of the Himalayan region Kumaun. Since the language is relatively understudied, in this study an attempt has been made to develop a parsing tool for use in Kumauni language studies. The eventual aim is help develop a technique for checking grammatical structures of Kumauni sentences. For this purpose, we have taken a set of pre-existing Kumauni sentences and derived rules of grammar from them. While selecting this set of sentences, effort has been made to select those sentences which are representative of the various possible tags of parts of speeches of the language, as used currently. This has been done to ensure that the sentences constitute all possible tags. These derived rules of Kumauni grammar have been converted to a mathematical model using Earley's algorithm suitably modified by us. The mathematical model so developed has been tested on a separate set of pre-existing Kumauni sentences and thus verified. This mathematical model can be used for the purpose of parsing new Kumauni sentences, thus providing researchers a new parsing tool.

Keywords
Kumauni Language, Context-free Grammar, Earley's Algorithm, Natural Language Processing, Parsing

EL ETIQUETADO DE FRASES DEL KUMAUNI DESPUÉS DE MODIFICAR EL ALGORITMO DE EARLEY'S

Resumen

La lengua kumauni es una de las lenguas regionales de la India, hablada en el área de Kumaun en la región del Himalaya. Puesto que esta lengua ha sido muy poco estudiada, en este trabajo se ha pretendido desarrollar una herramienta de etiquetado útil para los estudios sobre el kumauni. El objetivo final es contribuir a desarrollar una técnica para la comprobación de de las estructuras gramaticales en las oraciones del kumauni. Con esta finalidad, se ha escogido un conjunto de oraciones preexistentes del kumauni y a partir de ellas se han derivado reglas gramaticales. Además de esta selección, se ha intentado elegir aquellas oraciones que se usan actualmente y que son representativas de las posibles etiquetas en que pueden marcarse partes del habla. Esta elección se ha realizado para asegurar que en las oraciones aparezcan todas las etiquetas posibles. Las reglas derivadas de la gramática del Kumauni se han convertido a un modelo gramatical mediante el uso del algoritmo de Earley's previamente modificado. El modelo matemático desarrollado se ha verificado aplicándolo a un conjunto separado de oraciones preexistentes del Kumauni. Este modelo puede usarse para etiquetar nuevas oraciones del kumauni, ofreciendo a los investigadores una nueva herramienta de etiquetaje.

Palabras clave
lengua kumauni, gramática libre de contexto, algoritmo de Earley's, Procesamiento del lenguaje natural, etiquetado

1. Introduction

Parsing can be done in three stages. The first stage is *Token Generation*, or lexical analysis, by which the input character stream is split into meaningful symbols defined by a grammar of regular expression. The next stage is *Parsing* or syntactic analysis, which involves checking that the tokens form an allowable expression. This is usually done with reference to a Context Free Grammar (CFG) that recursively defines components which can make up an expression and the order in which they must appear. The final phase is *Semantic Parsing* or analysis, which requires working out the implications of the expression just validated and taking the appropriate action. In the case of a calculator or interpreter, the action is to evaluate the expression or program; a compiler, on the other hand, generates some kind of code. Attribute grammars can also be used to define these actions. Brian Roark (2001) presents a lexicalized probabilistic top-down parser which performs very well, in terms of both the accuracy of returned parses and the efficiency with which they are found, relative to the best broad-coverage statistical parsers.

Top-down backtracking language processors have some advantages compared to other methods, i.e.

1) They are general and can be used to implement ambiguous grammars.

2) They are easy to implement in any language that supports recursion.

3) They are highly modular, i.e. the structure of the code is closely related to the structure of the grammar of the language to be processed.

4) Associating semantic rules with the recursive functions that implement the syntactic productions rules of the grammar is straightforward in functional programming.

Languages which cannot be described by CFG are called Context Sensitive Languages. Tanaka (1993) has developed an algorithm for CFG. An informal description of a new top-down parsing algorithm has been developed by Richard A. Frost et al. (2006) that accommodates ambiguity and left recursion in polynomial time. Shiel (1976) noticed the relationship between top-down and Chart parsing and developed an approach in which procedures corresponding to non-terminals are called with an extra parameter, indicating how many terminals they should read from the input. Fujisaki Tetsunosuke (1984) has tested a corpus to parse it using Stochastic Context Free Grammar and probability theory to make the parse tree. R. Frost et al. (2007) presented a method by which parsers can be built as modular and efficient executable specifications of ambiguous grammars containing unconstrained left recursion. In 2008 the same authors, Frost et al. (2008), described a parser combinator as a tool that can be used to execute specifications of ambiguous grammar with constraints left recursion, which execute polynomial time and which generate compact polynomial sized representation of the potentiality.

Devdatta Sharma (1985), a leading linguist, was the first to study Kumauni language linguistically. To carry his initiative further, we have taken Kumauni language for information processing, i.e. to check the grammars of input sentences. Parsing process makes use of two components; a parser, which is a procedural component and a grammar, which is declarative. The grammar changes depending on the language to be parsed while the parser remains unchanged. Thus by simply changing the grammar, a system would parse a different language. We have taken Earley's Parsing Algorithm for parsing Kumauni sentences according to a grammar that we have defined for Kumauni language, using a set of pre-existing Kumauni sentences.

2. Earley's Parsing Algorithm

The task of the parser is essentially to determine if and how grammar of a pre-existing sentence can be determined. This can be done essentially in two ways, Top-down Parsing and Bottom- up parsing.

Earley's algorithm is a top-down dynamic programming algorithm. We use Earley's dot notation: given a production X → *xy*, the notation X → *x* • *y* represents a condition in which *x* has already been parsed and *y* is expected.

For every input position (which represents a position between tokens), the parser generates an ordered state set. Each state is a tuple (X → *x* • *y*, *i*), consisting of

- the production currently being matched (X → *x y*);
- our current position in that production (represented by the dot);
- the position *i* in the input at which the matching of this production began: the origin position.[1]

The state set at input position *k* is called S(*k*). The parser is seeded with S(0), consisting of only the top-level rule. The parser then iteratively operates in three stages: prediction, scanning, and completion.

- Prediction: For every state in S(*k*) of the form (X → *x* • Y *y, j*) (where j is the origin position as above), add (Y → • *z*, *k*) to S(*k*) for every production in the grammar with Y on the left-hand side (Y → *z*).
- Scanning: If a is the next symbol in the input stream, for every state in S(*k*) of the form (X → *x* • a *y, j*), add (X → x a • *y, j*) to S(*k+1*).
- Completion: For every state in S(*k*) of the form (X → *z* •, *j*), find states in S(*j*) of the form (Y → *x* • X *y*, *i*) and add (Y → *x* X • *y*, *i*) to S(*k*).

For example, let we take a sentence.

[1] Earley's original algorithm included a look-ahead in the state. Later research showed this to have little practical effect on parsing efficiency and it has subsequently been dropped from most implementations.

Let the input sentence be: “You eat the food in the restaurants”. The following numeric key can be supplied to the words of this sentence:

“0 *You* 1 *eat* 2 *the* 3 *food* 4 *in* 5 the 6 *restaurant* 7”.

Here the numbers appeared between words are called position numbers.

For CFG rule S →NP VP we will have three types of dotted items:

• [S→ .NP VP, 0, 0]

• [S→ NP.VP, 0, 1]

• [S→ NP VP., 0, 4]

Here,

S → Starting Symbol

NP → Noun Phrase

VP → Verb Phrase

1. The first item indicates that the input sentence is going to be parsed applying the rule S → NP VP from position 0.

2. The second item indicates the portion of the input sentence from the position number 0 to 1 that has been parsed as NP and the remainder left to be satisfied as VP.

3. The third item indicates that the portion of input sentence from position number 0 to 4 has been parsed as NP VP and thus S is accomplished.

Using Earley’s parsing algorithm

1. For each production S→ x, create [S→ *x*, 0, 0]
2. For j = 0 to n (n is the length of the input sentence)
3. For each item in the form of [X→ *x*.Yy, *i*, *j*] apply Predictor operation while a new item is created.
4. For each item in the form of [Y→ *z.i*, *j*] apply Completer operation while a new item is created.
5. For each item in the form of [X→ *x.wy*, *i*, *j*] apply Scanner operation.
6. If we find an item of the form [S→ *x*., 0, *n*] then we accept it.

Let us take another example.

“0 *you* 1 *eat* 2 *the* 3 *food* 4”.

Consider the following grammar:

1. S → NP VP	2. S → S PP
3. NP → n	4. NP → art n
5. NP → NP PP	6. PP → p NP
7. VP → v NP	8. n → You
9. n → food	10. v → eat
11. art → the	

Now, parsing the sentence using Earley’s parsing technique:

Step no.	**Formula**	**Used operation**
1	[S→.NP VP, 0, 0]	Initialization
2	[S→.S PP, 0, 0]	Apply Predictor to step 1 and step 2
3	[NP→.n, 0, 0]	
4	[NP→.art n, 0, 0]	
5	[NP→.NP PP, 0, 0]	Apply Predictor to step 3
6	[n→.“You”, 0, 0]	Apply scanner to 6
7	[n→ “You”, 0, 1]	Apply Completer to step 7 with step 3
8	[NP→n., 0, 1]	Apply Completer to step 8 with step 1 and step 5
9	[S→NP.VP, 0, 1]	
10	[NP→NP.PP, 0, 1]	Apply Predictor to step 9
11	[VP→.v NP, 1, 1]	Apply Predictor to step 11
12	[v→.“eat”, 1, 1]	Apply Predictor to step 10
13	[PP→.p NP, 1, 1]	Apply Scanner to step 12
14	[v→ “eat”.1, 2]	Apply Completer to step 14 with step 11
15	[VP→v.NP, 1, 2]	Apply Predictor to step 15
16	[NP→.n, 2, 2]	
17	[NP→.art n, 2, 2]	
18	[NP→.NP PP 2, 2]	Apply Predictor to step 17
19	[art → .“the”, 2, 2]	Apply Scanner to step 19
20	[art → “the”., 2, 3]	Apply Completer to step 20 with step 17

21	[NP → art .n, 2, 3]	Apply Predictor to step 21
22	[n → .“food”, 3, 3]	Apply Scanner to step 22
23	[n → “food”., 3, 4]	Apply Completer to step 23 with step 21
24	[NP → art n., 2, 4]	Apply Completer to 24 with 15
25	[VP → v NP., 1, 4]	Apply Completer to 25 with 9
26	[S → NP VP., 0, 4]	Complete

Table 1. Parsing of the sentence by Earley’s algorithm

3. Derivation of Kumauni language grammar and modification of Earley’s Algorithm

It is next to impossible to collect all types of sentences of any language; hence we have taken some pre-existing Kumauni sentences randomly and tried to derive rules of grammar from them. In this section we are making an attempt to develop a grammar of a language, which is understudied and underdeveloped and some of its folk characters are at the brink of extinct. It is the case of Kumauni, the language spoken in the panoramic locations of valleys of mountains of central Himalaya, which is a tranquil land in mist. This language can be divided into different dialects according to social and geographical differences.

Structure of Kumauni grammar is same as Hindi grammar:

Subject+ object+ verb.

We can see that a sentence can be written in different forms, which have the same meaning, i.e. positions of tags are not fixed. Therefore, we can not fix the grammar rule for one sentence, which might cause the grammar rule to become very long. The grammar rules that we have derived may not apply to all the sentences in Kumauni language since we have not considered all types of sentences possible in Kumauni language. Some of the sentences that have been used to make the rules of grammar for Kumauni language are given below:

Kumauni	**In English**	**Grammar**
kAn Je re?	Where are you going?	PP –VP
sab thEk Chan	They all are fine	NP- ADJ- VP
Ook byAh pichal sAl haigou	he got married last year	PN - ADJ- NP - VP
theek cha pein ItvAr din milOn	Well, see you on Sunday.	ADVP- PP – NP - VP
mein itvAr din Onake koshish karou	I will try to come on Sunday	PN- NP- ADV- VP
main pushp vihar sAketak paas roo(n)chou	I live in Pushp Vihar near Saket	PN- NP- PP- VP
Par jAno pein	Good Night	VP
myAr bAbu fauj me naukari kareni	my father is serving in indian army	NP- NP- PP- VP
jaduk AshA, utuk haber jyAdA hainch	It is more then expected	NP- ADJ- VP
champAwat bahute bhal jAg chuu	Champawat is a very beautiful place	NP- ADJ-VP
makai wanki ligi bahut door chaln pado	I have to go far for that place	NP- PP- ADJ- VP
ter mukh to nai buwAr jas chamakano	Your face is shining like a new bride.	NP - PP- VP
ab mee jaa	Now I am going	ADVP- PN- VP

Table 2. Grammar generation for Kumauni

Let K be the set of all parts of speeches in Kumauni language,

K = (NP, PN, VP, ADV, ADJ, PP, ART, IND)

Where

NP → Noun

PN → Pronoun

VP → Verb

ADV → Adverb

ADJ → Adjective

PP → Preposition

ART → Article

IND → Indeclinable

Formation of vector space for a language

Using English language, since it has 8 parts of speeches, we can form a matrix (callled connection matrix) of the order 8 x 8, where rows and columns are represented by parts of speeches. This matrix pertains to the FOLLOW relation.

PREV (x) = {Set of all lexical categories that can precede x in a sentence}

= {y: (Row y, Column x) is 1}

FOLLOW (x) = {y: (Row x, Column y) is 1}

For example, we take a sentence

"John is looking very smart"

Parsing it in parts of speeches, it becomes

NP VP ADV NP

Its connection matrix representation is depicted as:

$$\begin{array}{c|cccccccc|}
 & \text{NP} & \text{PN} & \text{VP} & \text{ADV} & \text{ADJ} & \text{PP} & \text{ART} & \text{IND} \\
\text{NP} & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
\text{PN} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\text{VP} & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 \\
\text{ADV} & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\text{ADJ} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\text{PP} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\text{ART} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\text{IND} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}$$

Table 3. Adjacency matrix of the sentence

Using a text document, we get several sentences and each sentence can be represented by a connection matrix of the order 8 x 8. Thus, a set of all matrices of the order 8x8 forms a vector space V of dimension 64 over the field of integers under addition and usual multiplication. Therefore, in a text document each sentence is an element of this vector space. Now, in any sentence, there are several parts of speeches hence it can be a subspace of the vector space generated for language. Similarly, parts of a sentence will also be a subspace of the sentence.

To carry this argument further, we propose some linear transformations of subspaces of Kumauni sentence. In the following sequence:

- T is linear transformation of Sentence subspace in Kumauni.
- U is linear transformation of the Proposition-phase subspace in Kumauni.
- W is the linear transformation of the Noun-phase subspace in Kumauni.

Additionally, Identity transformation has also been used.

T: (S)	**U: (PP)**	**W: (NP)**
T_1: (S)= PP VP	U_1: (PP)= PN NP	W_1: (NP)= NP PP
T_2: (S) = PP	U_2: (PP)= NP PN	W_2: (NP)= PP NP
	U_3: (PP)= ADJ NP	W_3: (NP)= ADV NP
	U_4: (PP)= NP ADJ	W_4: (NP)= PP
	U_5: (PP)= NP	W_5: (NP)= ART NP
	U_6: (PP)= ADJ	W_6: (NP)= NP ART
	U_7: (PP)= IND NP	W_7: (NP)= IND PN
	U_8: (PP)= PN	W_8: (NP)= PN IND
	U_9: (PP)= ADV NP	W_9: (NP)= VP
	U_{10}: (PP)= ADV	

Table 4. Transformation rules for phrase subspaces

3.1. Modification of Earley's Algorithm for Kumauni Text Parsing

We know that Earley's algorithm uses three operations, Predictor, Scanner and Completer. We add Predictor and Completer in one phase and Scanner operation in another phase.

Let *x*, *y*, *z*, PP, VP are sequence of terminal or nonterminal symbols and S, B are non terminal symbols.

Phase 1: (Predictor + Completer)

For an item of the form [S → *x* .B*y*, *i*, *j*], create [S → *x.zy*, *i*, *j*] for each production of the [B → *z*]. Mathematically in phase 1 we apply the transformations suggested earlier.

Phase 2: (Scanner)

For an item of the form [S → *x.wy*, *i*, *j*] create [S → *xw.y*, *i*, *j*+1], if w is a terminal symbol appeared in the input sentence between *j* and *j*+1. When the transformation is successfully applied then it allows us to move in to next position or transformation.

Our Algorithm:

Input: Tagged Kumauni Sentence

Output: Parse Tree or Error message

Step 1: If Verb is present in the sentence then [T: S → .PP VP, 0, 0] then we use transformation T_1.

Else [T: S → .PP, 0, 0] then we use transformation T_2.

Step 2: Use the transformation U and W and do the following steps in a loop until there is a success or error

Step 3: For each item of the form of [S → *x*.By, *i*, *j*], and we use transformations T_i, U_i, W_i.

Step 4: For each item of the form of [S→ *.xwy*, *i*, *j*], apply phase 2

Step 5: If we find an item of the form [S →*nx.* , 0, *n*], i.e the transformations work successfully, then we accept the sentence as success else error message. Where n is the length of input sentence.

And then come out from the loop.

Step 6: Generate the parse trees for the successful sentences according to the used transformations.

A transformation is said to be success if it same as any member of table 1.

Some other modifications of Earley's algorithm:

1. Earley's algorithm blocks left recursive rules [W: NP→ .NP PP, 0, 0], when applying Predictor operation. Since Kumauni Language is a Free-Word-Order language. We are not blocking this type of rules.

2. Earley's algorithm creates new items for all possible productions, if there is a non terminal in the left hand side rule. But we reduce these productions by removing such type of productions, which create the number of total productions in the stack, greater then total tag length of the input sentence.

3. Another restriction we used in our algorithm for creating new item is that, if the algorithm currently analyzing the last word of the sentence, then it selects only the single production in the right hand side (example [U: PP→NP]). The other rules (which have more then one production rules in right hand side (example [U: PP→PN NP])) are ignored by the algorithm.

3.2. Parsing Kumauni text using proposed grammar and algorithm

Let us take a Kumauni sentence.

मी त्यार दगड़ बजार जू (Mee tyar dagad bazAr joo)

In English it means,

"I will go to market with you"

Now the position number for the words are placed according to which word will be parsed first.

0 *Mee* 1 *tyar* 2 *dagad* 3 *bazaar* 4 *joo* 5

Where in our sentence

1. PN → *"mee"* 2. PN → *"tyar"* 3. PP → *"dagad"* 4. NP → *"bazaar"*
5. VP → *"joo"*

Now we use the transformation defined earlier (Table 3)

Parsing process will proceed as follows:

Sr. No	Rule	Phase applied
1	[S → .PP VP , 0, 0] by T1	Apply Phase 1
2	[S → .NP VP, 0 , 0] by U5	Apply Phase 1
3	[S → .PP NP VP, 0, 0] by W2	Apply Phase 1
4	[S → .PN NP NP VP, 0, 0] by U1	Apply Phase 1
5	[S →. "mee" NP NP VP, 0, 0]	Apply Phase 2
6	[S →. "mee" .NP NP VP, 0, 1] by identity	Apply Phase 1

	transformation	
7	[S → “mee” .PP NP VP, 0, 1] by W4	
8	[S → “mee” PN NP NP VP, 0, 1] by U1	Apply Phase 1
9	[S → “mee”. “tyar” NP NP VP, 0, 1]	Apply Phase 2
10	[S → “mee”. “tyar” .NP NP VP, 0, 1] by identity transformation	Apply Phase 1
11	[S → “mai” “tyar” .PP NP VP, 0, 2] by W4	Apply Phase 1
12	[S → “mee” “tyar”.“dagad” NP VP, 0, 2]	Apply Phase 2
13	[S → “mee” “tyar” “dagad” .NP VP, 0, 3] by identity transformation	Apply Phase 1
14	[S → “mee” “tyar” “dagad”..“bazaar” VP, 0, 3]	Apply Phase 2
15	[S → “mee” “tyar” “dagad”..“bazaar” .VP, 0, 4] by identity transformation	Apply Phase 1
16	[S → “mee” “tyar” “dagad”..“bazaar” . “joo”, 0, 4]	Apply Phase 2
17	[S → “mee” “tyar” “dagad”..“bazaar” . “joo”, 0, 5]	Complete

Table 5. Parsing of the sentence by modified Earley’s algorithm

In the above example, we have shown only the steps which proceeds to the goal. The other steps are ignored.

4. Stages of the model

In the model there are 3 stages:

• Lexical Analysis

• Syntax Analysis

• Tree Generation

In the Lexical Analysis stage, program finds the correct tag for each word in the sentence by searching the database.

There are seven databases (NP, PN, VP, ADJ, ADV, PP, ART, IND) for tagging the words.

In Syntax Analysis stage, the program tries to analyze whether the given sentence is grammatically correct or not.

In Tree Generation stage, the program finds all the production rules which lead to success and generates parse tree for those rules. If there are more then one path to success, this stage can generate more than one parse trees. It also displays the words of the sentences with proper tags. The following figure shows a parse tree generated by the model. The original parse tree for the above sentence is.

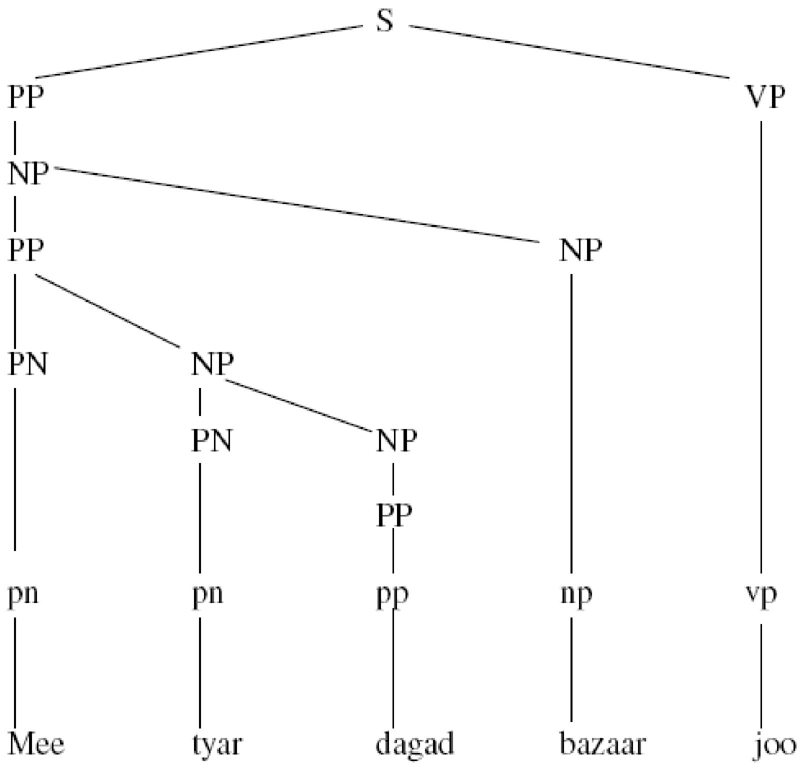

Figure 1. Parsed tree structure of sentence

5. Verification of program

After implementation of Earley's algorithm using our proposed grammar, it has been seen that the algorithm can easily generate parse tree for a sentence if the sentence structure satisfies the grammar rules. For example, we take the following Kumauni sentence:

मेर नाम कमल छ | (Mer nAma Kamal chh)

The structure of the above sentence is NP-NP-VP. This is a correct sentence according to the Kumauni literature. According to our proposed grammar, a possible top down derivation for the above sentence is:

1. S [Handle]	
2. >>PP VP	*[T_1: S→PP VP]*
3. >> NP VP	*[U: PP→NP]*
4. >>NP PP VP	*[W: NP→NP PP]*
5. >>NP NP VP	*[U: PP → NP]*
6. >>mer nAma NP VP	*[W: NP → mer nAma]*
7. >>mer nAma kamalVP	*[W: NP→ Kamal]*
8. >>mer nAma kamal chh	*[VP → chh]*

From the above derivation it is clear that the sentence analysed by the model is correct according to the proposed grammar, thus proving that our parsing model generates a parse tree successfully. The actual programme shall be as follows.

```
Input sentence- Mer nAma Kamal chh.
Sentence recognized
Tree ---->
1.          S
2.          [S ---> (PP VP)]
3.          [PP ---> (NP)] VP
4.          [NP ---> (NP PP)] VP
5.          [NP ---> (np :Mer nAma)]PP VP
6.          [PP]VP
7.          [PP ---> (NP)]VP
8.          [NP ---> (np : Kamal)]VP
9.          [VP]
10.          [VP ---> (vp :chh)]
```

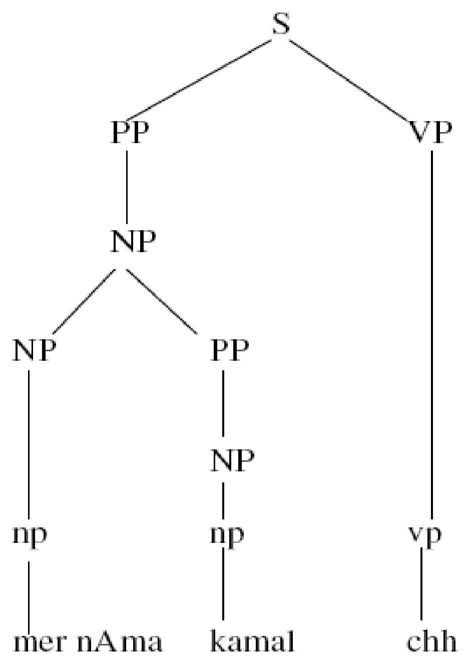

Figure 2. Verified tree structure by Earley's Algorithm

This model tests only the sentence structure according to the proposed grammar rules. So, if the sentence structure satisfies the grammar rules and follows Earley's algorithm then the model recognizes the sentence as a correct sentence and generates a parse tree. Otherwise it gives an error output.

6. Conclusion

We have developed a context free grammar (CFG) for simple Kumauni sentences, studied the issues that arise in parsing Kumauni sentences and produced an algorithm suitable for those issues. This algorithm is a modification of Earley's Algorithm, which has proved to be simple and effective. Whereas the traditional Earley's algorithm so many steps in parsing, our model reduces the length of parsing steps. It has an added feature in the sense that whereas Earley's algorithm contains three stages, our model works only in two steps.

7. Future Work

In this work, we have considered a limited number of Kumauni sentences for deriving the grammar rules. We have also considered only the seven main tags. In future work(s) related to the field of study covered in this paper, an attempt can be made to consider many more Kumauni sentences and more tags, for developing a more comprehensive set of grammar rules.

References

DEVIDATTA, Sarma (1985) *The formation of Kumauni language* (SILL: series in Indian languages and linguistics), Bahri Publications.

EARLEY, Jay (1970) *An efficient context free parsing algorithm*, Communications of the ACM, Volume 13, no 2, February-1970.

FROST, R., R. HAFIZ and P. CALLAGHAN (2007) "Modular and Efficient Top-Down Parsing for Ambiguous Left-Recursive Grammars., *10th International Workshop on Parsing Technologies (IWPT),* ACL-SIGPARSE, June 2007, Prague, 109-120.

FROST, R., R. HAFIZ and P. CALLAGHAN (2008) "Parser Combinators for Ambiguous Left-Recursive Grammars, *10th International Symposium on Practical Aspects of Declarative Languages (PADL)*, ACM-SIGPLAN, January 2008, San Francisco, Springer, vol. 4902/2008, 167-181.

FROST, Richard A. and Rahmatullah HAFIZ (2006) "A New Top-Down Parsing algorithm to Accommodate Ambiguity and Left Recursion in Polynomial Time", *ACM SIGPLAN Notices*, Vol. 41 (5), 45-54.

FUJISAKI, Tetsunosuke (1984) "A Stochastic approach to sentence parsing", *Annual Meeting of the ACL Proceedings of the 10th International Conference on Computational Linguistics and 22nd annual meeting on Association for Computational Linguistics*, Stanford, California, 16-19.

ROARK, Brian (2001) "Probabilistic top-down parsing and language modeling", *Computational Linguistics MIT Press***,** volume 27, issue 2 (June 2001), 249-276.

SHIEL, B. A. (1976) *Observations on context-free parsing. Technical Report* TR 12-76, Center for Research in Computing Technology, Aiken Computational Laboratory, Harvard University.

TANAKA, Hojumi (1993) *Current trends on parsing - a survey,* TITCS TR93-00031. Available at www.citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.56.83