Induction of Chomsky Normal Form in Context-Free Grammar of LL(1) Parser: Some Initial Results

Dilawar Naseem[1], Muhammad Shumail Naveed[1], Hassan Ali[1] , Samra Riaz[2]

***Abstract*- Parsing is to analyze the input lexeme and compilers have difficulty in processing due to human language structure, improvement in parsing processing can improve compiler speed. The paper aims at improving parsing by the introduction of Chomsky Normal Form (CNF) to Context-Free Grammar (CFG). For this research study, conventional English grammar in CFG is used and the conventional conversion method is used thoroughly. The grammar is converted into LL(1) form with the help of the LL(1) conversion algorithm and for the confirmation of successful conversion parsing table of LL(1) is conferred. For the analysis of LL (1) grammar input stack of 50 lexeme is verified by parsing clinched with the LL (1) grammar. The conventional LL(1) English grammar is induced with Chomsky Normal Form (CNF) and the resultant CNF converted LL (1) grammar is parsed with an input stack of 50 lexeme that are used for the LL(1) grammar. The study concluded that the introduction of CNF into LL(1) does not show significant improvement after the introduction of CNF into conventional LL(1) parser Introduction of CNF into LL(1) brings out parsing difficulty in processing the input stack of LL(1) into CNF introduced LL(1)**

Keywords: **Compiler, Chomsky Normal Form, LL (1) Parsing, Parsing, Top-down parsing.**

INTRODUCTION

The parser is the segment of a compiler that takes a token as input and with the help of contemporary grammar, renovates it into the equivalent parse tree. A parser is also called a Syntax Analyzer.

The term has a tad distinct meanings in different branches of linguistics and technology. Orthodox sentence parsing is often achieved as a way of information the precise meaning of a sentence or lexeme, on occasion with the resource of gadgets comprehensive sentence diagrams. It typically emphasizes the implication of grammatical dissections which include predicate.

Patricia et al. [1] elaborated that within computational linguistics the time period is used to refer to the formal evaluation by way of a processor of a sentence or other words into its parts, consequential in a parse tree displaying their syntactic relative to every different, which might also integrate semantic and other facts (p-factors).

The parsing can be preceded or followed with the aid of different steps, or these can be mixed into a unmarried step. The parser is often preceded by using a separate lexical analyzer, which creates tokens from the sequence of enter characters; alternatively, these may be blended in scanners parsing.

Patricia et al. [2] similarly elaborated that Parsers can be programmed with the aid of hand or maybe routinely or semi-mechanically engendered by means of a parser generator. Parsing is corresponding to templating, which produces formatted output. those can be carried out to one-of-a-kind domains, but regularly appear collectively, including the scanf/printf pair, or the input and output stages of a compiler. The parse tree is castoff as the idea for paraphrase. Parsers are considered reliable with the path in which they parse. the two wide classes of parsers are top-down, in which the tree is built from the basis right down to the leaves, and bottom-up, in which the parse tree is created from the leaves upward to the foundation.

Parsing algorithms that work for any explicit grammar are complex and ineffective. The complexity of such algorithms is O(n3), which means that the total time they yield is on the order of the dice of the period of the lexeme to be parsed. This highly massive amount of time is required because these algorithms regularly must lower back up and reparse part of the sentence being analyzed. Reparsing is needed while the parser has made a mistake within the parsing technique.

Parser additionally calls for that a part of the parse tree being created ought to be pull to bits and rebuilt. O(n3) algorithms are normally no longer useful for systems, as well as syntax analysis for a compiler, due to the fact they may be far too sluggish. In conditions including this, computer scientists frequently look for quicker algorithms, even though much less popular. Generality is traded for performance.

Grammar may be regarded as a device that computes the sentences of a language. The type of grammars become brought via Noam Chomsky is divided into 4 instructions: enumerable grammars, Context-sensitive grammars, CFG's, everyday grammars. A grammar "G = (V, T, S, P)" is stated to be context-free if all productions have the shape "A → x", " where "A ∈ V", and "x ∈ (V ∪ T)*"". A language "L" is said to be

[1] University of Balochistan, Quetta Country Pakistan
[2] Sardar Bahadur Khan Women University,Quetta Country Pakistan

context-free if there is a context-free grammar "G" such that "L = L (G)".

According to Alexandar [3], the control that critiques formal grammars and dialects is a part of carried out arithmetic and is taken into consideration formal language concept. Its packages are observed in theoretical computer science, linguistics, formal semantics, numerical rationale, and exclusive zones.

The formal definition of the syntax of grammar is done by the author Chomsky [4] and [5], in his work he explains via letting a grammar G includes the following components.

- Finite set N of variables that doesn't appear in the lexeme generated through G.
- Finite set Σ of terminals that are disjoint from N.
- Finite production rules P of the from

$$"(\Sigma \cup N)^* \, N(\Sigma \cup N)^* \rightarrow (\Sigma \cup N)^*"$$

- $S \in N$ distinguishes as a begin symbol.

wherein U is union operator and * is Kleene star operator. each creation runs maps starting with one series of symbols then onto the following, in which the primary string incorporates a discretionary range of symbols given no much less than one of them is a nonterminal. For the state of affairs that the second one string comprises solely of the unfilled string i.e., that it includes no symbols by using any manner and it might be supposed with unique documentation (often, e or ε) to keep a strategic distance from disarray.

The semantics of grammar as mentioned by using Chomsky is defined on operations of grammar as follows.

•A grammar "G" is given with tuples as "G = (N, Σ, P, S)" and binary relation "$\Rightarrow G$" (moves a step beforehand) on lexeme $(\Sigma \cup N)^*$ is defined as follows.

"$a \Rightarrow G$ yiff $\exists x,c,p,q \in (\Sigma \cup N)^* : (a=xcp) \wedge (p \rightarrow q \in P) \wedge (y=xqc)$"

•Then the relation "$\Rightarrow G^*$" movements in zero or extra steps is described as the transitive end of "$\Rightarrow G$".

•"$(\Sigma \cup N)^*$" can be driven in a finite no. of steps and is a member of "$\{j \in (\Sigma \cup N)^* | S \Rightarrow G^* \; j \}$" (variables symbols that are a member of "Σ^*").

The grammar "G=(N, Σ, P, S)" is efficiently the semi-thue system "(NUΣ, P)", transforming word inside the very same manner; the primary difference is in that we apprehend unique nonterminal symbols which have to be revised in revamping leads, and are simply intrigued by way of rewritings from the assigned start image to word with out variables symbols.

As very briefly explained with the aid of Johan [6], the solicitations of grammars as Grammars are an essential tool for portraying dialects. The applications that have been applied to provide the numerous kinds of sentence formalisms from grammar for feature dialects to programming dialects, to dialects used to depict improvement in science.

This examine determines the effect of introduction LL(1) grammar with Chomsky normal form (CNF) and avails consequences. This examine also squares the parsing pace of the LL(1) parser earlier than and after the introduction of CNF with context-free grammar (CFG).

LITERATURE REVIEW

Ali et al. [7] worked on LL(1) parsing and Greibach normal form (GNF) by comparing the two grammar formats with parsing expression grammar (PEG). In his work he successfully dealt with arithmetic expression Grammar in both LL(1) and GNF grammars, resulting in the successful integration of the two formats.

Naveed [8] also worked on LL(1) parsing in addition to terminal prefixing. He also successfully merged the two formats with parsing for arithmetic expression grammar. he induced the terminal prefixing into LL(1) grammar format and achieved the alteration.

Kuhl et al. [9] worked on a parser generator that turned into implemented Java. As this grammar is written in EBNF and it assessments LL (1) well that either it's far suitable for "recursive descent parsing" and additionally generates parser that consists of the set of sequential items. It gives the scanner and also classifies positive interfaces which are gratifying and can be carried out for the parser to construct parsing bushes. After Wirth's well-known parser Oops was spotted for parsing and the nodes are substances and navigate to techniques immediately related to the graph nodes. A standard parser uses a grammar graph that is specifically designed to illustrate the techniques of language popularity and how it's far performed. Oops generates a parser robotically and higher rejects the unsuitable grammar grow to be its natural manner of arranging its graph nodes. Oops uses the divide and overcome method to verify its grammar and parsing as nicely.

Dabhoiwala [10] presented a review on LL(1) parser that details the parser and its performance. It states that parsing is the second step of any program, initially it scans every lexical unit of the program. A lexical unit can be a keyword, operator any constant or identifier of any programming language. Once all the units of programming language are operated or identified by a lexical analyzer then the parsing is performed over those language units. This shows that the parser checks for accurate syntax, once it declares the standard syntax of any language or program, then it creates the parsing tree.

Sam et al. [11] designates LL (1) that when it's pragmatic to grammar "G", It produces LL (1) parser for that grammar "G" only if such a parser exists. This approach confirm about the generator and parser that either it produces that sound and complete grammar and they terminate all input on valid or

invalid without using any fuel parameters. This study mainly shows that it was on the two possible extensions of this parsing which are:
1- Ruling out parser errors as priori
2- Generating parser source code

The parser in this paper states and discussed those branches that make the extraction process survive well but slow down the resulting code even though it never comes to this state if correct parser LL (1) is being applied to the language. A very useful analogy states the difference between interpreter and parser while stating that a well-typed program can't go wrong, Robin [12] and branches can be removed from the parser while performing function with correct LL (1) parser table instead of the just-typed table.

This prior approach is more efficient to rule out the errors according to some observers discussed in this paper. The parser also discussed in this paper uses table-based interpreters which is likely inefficient as compared to generated parser code.

Michael et al. [13] in his paper discusses the systematic and lots green LL (1) slipups convalescence method applied for an LL (1) generator. It routinely generated proper messages with appropriate diagnostic information and corrected mistakes by resources of clearance some input stack and correspondingly it go off some symbols from parsing -stack to reinstate the effective conformation of the parser. This paper defines the concept of reliability that's statement based and it observes how the enter symbols vary from each different symbols incapability as recovery points. A symbol that has high reliability is probably now not located in the enter by twist of fate so because of that it become by no means discarded and saved on parsing with this image. when a few blunders is detected only then the mistake recuperation recurring is invoked so there may be no such additional habitual or time is required for parsing correct applications and this paper experimented with this error restoration method results in 90 % accuracy.

John [14] delivered a very efficient incremental LL (1) parsing set of rules for language-based totally editors that have been carried out in Fred, a dependent display screen-based editor that specially makes use of the shape popularity technique. This paper featured absolutely best-grained analysis and an outstanding method to parsing control and mistakes recovery. A display screen-primarily based editor has a keystroke intensive mode for person interplay that follows the cursor style for parsing. It also supplied incomplete LL (1) grammars for managing the complexity of full language grammars and additionally dependent editor assist for most effective partly structured undertaking languages. This approach discussed the semantics of entire grammars and additionally offered the transformation of incomplete LL (1) into complete LL (1).

James et al. [15] and Terence et al. [16] implemented LPARSER, LL (1) parser-based generation system in Turbo Pascal, and this system mainly consists of a table generator and a skeleton parser along with a lexical analyzer. This table generator read grammar description from the text files and then it generates several files and compile them along with a skeleton parser. As there are similarities among LL (1) parser and LR (1) parser this paper also discussed those parsers and most importantly this approach has been compared with the well-known parser generator YACC which is developed by Steve Johnson at bell laboratories and it also read the grammar description and generates LALR (1) parser written in C language. As YACC has semantic actions and uses the bottom-up approach for parsing and maintaining the stack other than this LPARSER uses the top-down approach for implementing and maintaining its stack without relying on parsing stack.
Parr [17] discussed a parser generator named ANTLR, which is a parser generator that has a amalgamation of "hand-coded" parser and parser code. It is informal to practice than supplementary language tools and the significant feature of ANTLR is that it delivers bases, it allows the parser with arbitrary expressions while using semantic and syntactic context. It also eliminates the hand-tweak output of ANTLR. It also integrates lexical analysis and syntactic scrutiny receives LL(k) grammars for k>1 with extended BNF code and also engenders an mechanically intellectual syntax tree.

MATERIALS AND METHODS

Research methodology for the LL(1) and CNF introduced LL(1) is syntactically and systematically defined in this study. The proposed CNF introduced LL(1) algorithm works with all the symbols of Grammar (G) and the production rules (Z). The research methodology is here.

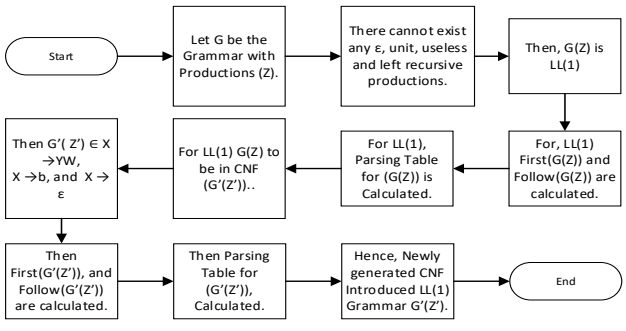

Figure 1 Research Methodology.

The main assumptions of the study are there cannot be any, unit, and useless productions for the grammar to be in LL(1) grammar structure. The LL(1) grammar structure also does not accept left recursion. Where the CNF suggests grammar to adapt to the new structure of CNF that is the production rules from the right-hand side must carry only two variables.

A) Algorithm to convert the LL(1) grammar in CNF based LL(1) parser

Algorithm: Converting LL (1) Grammar to Chomsky Normal Form

1. Consider an unambiguous grammar G, iff □ G(X) where X is any number of productions and X∈P* (P* denotes having more than one parse tree for G(X)).
2. □ G(X) there exits terminal symbols α and β where Y can be □ then each for rule X→αYβ, add a new rule X→αβ (removing span productions).
3. From G(X) remove unit productions as X→Y, then all productions are of form X → a or X → x_1 … x_k (k≥2).
4. For each terminal 'a', add a non-terminal 'Za' as production Za → a. In all rules X→x_1...x_k(k≥2), replace each a with Za.
5. For every production X→Y_1… Y_n with n≥2, add new symbols W_2, ...,W_{n-1} and replace the production with X→ Y_1 W_2, W_2 →Y_2 W_3, …, W_{n-1}→Y_{n-1} Y_n

The algorithm starts with condition that if and only if the grammar remains unambiguous and each production rule of the grammar can generate more than one parse tree. Then, for all the symbols of grammar G(X) there must be terminal symbols and non-terminal symbols can generate □ rules, following the removal of useless productions.

After removing useless productions, followed by the unit productions removal. To ensure the CNF syntax LL(1) grammar structures are modified to a CNF condition where the production set starts with non-terminal and followed only one terminal symbol.

The production rules of the CNF Introduced LL(1) Grammar from the right side can not add any variable more than 2 if any production rule carries more than 2 symbols then a new symbol must be introduced to satisfy CNF grammar syntax.

B) Algorithm to Construct Chomsky normal form (CNF) based LL(1) parsing table

The algorithm (Construction of CNF based LL (1) Parsing Table) explains the parsing procedure for parser table P'.

Algorithm: Construction of CNF based LL (1) Parsing Table.

1. Considering CNF based LL (1) parser to utilize all the non-terminal symbols with lookahead 1 then every element 'Z' belongs to Grammar 'G'.
2. For every element 'X', non-terminal 'N' and lookahead 'a' then there exists a condition Z | (N, a)
3. The algorithm works with starting from Z | (N, a) whole considering non-terminal 'N' and lookahead 'a'.
4. For LL (1) grammar paring table (T) there exists another grammar G' then there exists Z' | Z' (N, a).
5. For grammar G' there exists non-terminal t' with lookahead a'.
6. For grammar G', blank entries in the parsing table (P') are again errors.

The algorithm Starts by firstly ensuring non-terminals 'N', lookahead 'a', and productions carrying N, a, and Z. That provides knowledge of parsing table constructed by following the grammar production rules. Also, note the total number of tuples used within parsing table P'. The second last step points to non-terminal N and lookahead t and finally begins for blank or undefined spaces pointing to the errors in parsing table P'.

C) Algorithm for the construction of LL(1) first and follow.

The construction of LL (1) Grammar with respect of LL (1) Construction Algorithm two association functions are important that are first and follow. In the above algorithm construction of the first and following table is being done by considering any CNF grammar 'G' with Parsing Table 'T'.

Algorithm: Algorithm for Construction of First and Follow for LL (1) Algorithm.

1. Consider any CNF Grammar 'G' for Parsing Table 'T' while every production B of Grammar 'G'.
2. For every terminal symbol b, First of (start adding B to T [B, b].
3. If □ is in First (), then start adding B on T [B, b] for every terminal b in Follow(B).
4. If is in First (with $ in Follow (start adding B to T [B, $].
5. Blank or Undefined entries in Parsing Table 'T' are counted as errors.

Starting by considering production B for grammar 'G'. Following the step where for every terminal symbol b, First (to T [B, b]. Then, there exists a condition where If and only If happens to be the first of then for production B parsing table entry should be T [B, b]. The second last step describes another possibility for First (with $ then T [B, $] occurs. The final step defines blank entries in Parsing Table 'T' counted as errors.

This section of the article further has the following parts, section D is traditional LL(1) English Grammar and section E is the CNF introduced LL(1) English Grammar with First and Follow sets for both of the algorithms the traditional and newly generated one (CNF Introduced LL(1) English Grammar). Section D and Section E also describe the parsing table for both of the grammars (Traditional LL(1) English Grammar and CNF Introduced LL(1) English Grammar). Instead of putting all the tables in the Appendix, add all the tables in this section

D. Traditional LL(1) English Grammar

The steps defined above are applied to the LL(1) English Grammar shown below.
“S” -> “NP” “VP”.
“NP” -> “the” “Nominal” | “a” “Nominal” | “Nominal” | “ProperNoun” | “NP1” “PP”.
“NP1” -> “PP” “NP”.
“Nominal” -> “N” | “Adjs” “N”.
“N” -> “cat” | “dogs” | “bear” | “girl” | “chocolate” | “rifle”.
“ProperNoun” -> “chris” | “fluffy”.
“Adjs” -> “Adj” “Adjs”.
“Adj” -> “young” | “older” | “smart”.
“VP” -> “like” | “likes” | “thinks” | “shot” | “smells” | “VP1” “PP”
“VP1” -> “PP” “VP”
“V” -> “like” | “likes” | “thinks” | “shot” | “smells”.
“PP” -> “Prep” “VP”.
“Prep” -> “with”.

The above grammar comes with terminal symbols, non-terminal symbols. Terminal symbols are with, smells, shot, thinks, likes, like smart, older, young, rifle, chocolate, girl, bear, dogs, cat, a the, and number non-terminals S, NP, NP1, Nominal, N, ProperNoun, Adjs, Adj, VP, VP2, V, PP, Prep. The above grammar is clear because it does not have an unnecessary unit, inaccessible and □-free productions.

The First and Follow sets for LL(1) English Grammar are here.

First(“S”)={“the”,”a”,”chris”,”fluffy”,”cat”,”dogs”,”bear”,”girl”,”chocolate”,”rifle”,”young”,”older”,”smart”,”with”}
First(“NP”)={“the”,”a”,”chris”,”fluffy”,”cat”,”dogs”,”bear”,”girl”,”chocolate”,”riflsnie”,”young”,”older”,”smart”,”with”}
First(“NP1”) ={“with”}
First(“Nominal”)={“cat”,”dogs”,”bear”,”girl”,”chocolate”,”rifle”,”young”,”older”,”smart”}
First(“N”) ={“cat”,”dogs”,”bear”,”girl”,”chocolate”,”rifle”}
First(“ProperNoun”) ={“chris”,”fluffy”}
First(“Adjs”) ={“young”,”older”,”smart”}
First(“Adj”) ={“young”,”older”,”smart”}
First(“VP”) ={“like”,”likes”,”thinks”,”shot”,”smells”,”with”}
First(“VP1”) ={“with”}
First(“V”) ={“like”,”likes”,”thinks”,”shot”,”smells”}
First(“PP”) ={“with”}
First(“Prep”) ={“with”}

The Follow sets are here.
Follow(“S”) = {“$”}
Follow(“NP”) = {“like”,”likes”,”thinks”,”shot”,”smells”,”with”}
Follow(“NP1”) = {“with”}
Follow(“Nominal”) = {“like”,”likes”,”thinks”,”shot”,”smells”,”with”}
Follow(“N”) = {“like”,”likes”,”thinks”,”shot”,”smells”,”with”}
Follow(“ProperNoun”)={“like”,”likes”,”thinks”,”shot”,”smells”,”with”}
Follow(“Adjs”) = {“cat”,”dogs”,”bear”,”girl”,”chocolate”,”rifle”}
Follow(“Adj”) = {“young”,”older”,”smart”}
Follow(“VP”)={“$”,”with”,”like”,”likes”,”thinks”,”shot”,”smells”,”the”,”a”,”chris”,”fluffy”,”cat”,”dogs”,”bear”,”girl”,”chocolate”,”rifle”,”young”,”older”,”smart”}
Follow(“VP1”) = {“with”}
Follow(“V”) = {“undefined”}
Follow(“PP”)={“like”,”likes”,”thinks”,”shot”,”smells”,”with”,”the”,”a”,”chris”,”fluffy”,”cat”,”dogs”,”bear”,”girl”,”chocolate”,”rifle”,”young”,”older”,”smart”,”$”}
Follow(“Prep”) = {“like”,”likes”,”thinks”}

The parsing table for the above-mentioned LL(1) English Grammar is shown in tabular form on appendix I. Parse tree drives the construction of input lexeme from grammar (G). Let I Є G(Z), where I is the input word, along with grammar (G), and production rules (Z).

Tree nodes are terminal and non-terminal variables and in figure 2(a) parse tree of the input string “the dogs shot” for “LL(1) English Grammar” is below.

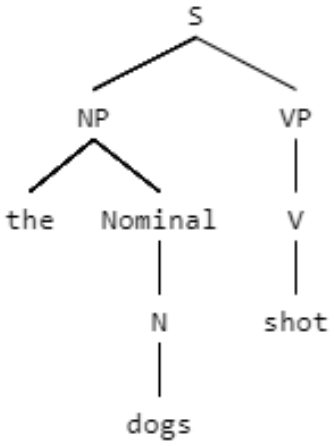

Fig 2(a) Parse tree of input string “The dogs shot” for “LL(1) English Grammar

Tree nodes for fig 2(a) are the dogs and shot along with the level of 5, depth 4d, leaf nodes (the, dogs, shot), the root node (S).

Below fig 2(b) is a derivation of input string “the dogs smells” for “LL(1) English Grammar

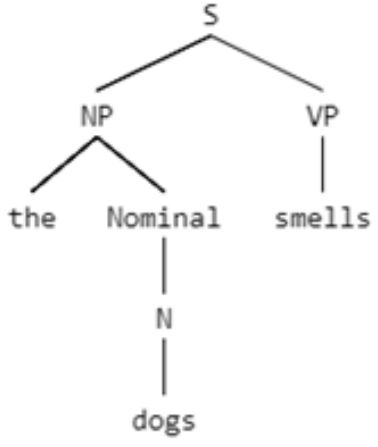

Fig 2(b) Parser tree of input string of “the dogs smells” for “LL(1) English Grammar”.

Tree nodes for fig 2(b) are the, dogs, and smells along with the level of 5, depth 4d, leaf nodes (the, dogs, smells), the root node (S).

E. CNF Introduced LL(1) English Grammar

The steps defined in section 2 are applied to the “LL(1) English grammar” and the newly generated “CNF Introduced LL(1) English Grammar” is mentioned below.

“S” -> “NP” “VP”.
“NP” -> “A” “Nominal” | “B” “Nominal” | “cat” | “dogs” | “bear” | “girl” | “chocolate” | “rifle” | “Adjs” “N” | “chris” | “fluffy” | “NP1” “PP”.
“A” -> “the”.
“B” -> “a”.
“NP1” -> “PP” “NP”.
“Nominal” -> “cat” | “dogs” | “bear” | “girl” | “chocolate” | “rifle” | “Adjs” “N”.
“N” -> “cat” | “dogs” | “bear” | “girl” | “chocolate” | “rifle”.
“Adjs” -> “Adj” “Adjs” | “young” | “older” | “smart”.
“VP” -> “like” | “likes” | “thinks” | “shot” | “smells” | “VP1” “PP”.
“VP1” -> “PP” “VP”.
“V” -> “like” | “likes” | “thinks” | “shot” | “smells”.
“PP” -> “Prep” “VP”.
“Prep” -> “with”.

The First and Follow sets are mentioned and parsing table in appendix I for “CNF Introduced LL(1) English Grammar” for legibility.

First(“S”)={“cat”,”dogs”,”bear”,”girl”,”chocolate”,”rifle”,”chris”,”fluffy”,”the”,”a”,”young”,”older”,”smart”,”with”}
First(“NP”)={“cat”,”dogs”,”bear”,”girl”,”chocolate”,”rifle”,”chris”,”fluffy”,”the”,”a”,”young”,”older”,”smart”,”with”}
First(“A”) ={“the”}
First(“B”) ={“a”}
First(“NP1”) ={“with”}
First(“Nominal”)={“cat”,”dogs”,”bear”,”girl”,”chocolate”,”rifle”,”young”,”older”,”smart”}
First(“N”)={“cat”,”dogs”,”bear”,”girl”,”chocolate”,”rifle”}
First(“Adjs”) ={“young”,”older”,”smart”}
First(“Adj”) ={“young”,”older”,”smart”}
First(“VP”)={“like”,”likes”,”thinks”,”shot”,”smells”,”with”}
First(“VP1”) ={“with”}
First(“V”) ={“like”,”likes”,”thinks”,”shot”,”smells”}
First(“PP”) ={“with”}
First(“Prep”) ={“with”}

Follow sets are mentioned here.
Follow(“S”)={“$”}
Follow(“NP”)={“like”,”likes”,”thinks”,”shot”,”smells”,”with”}
Follow(“A”)={“cat”,”dogs”,”bear”,”girl”,”chocolate”,”rifle”,”young”,”older”,”smart”}
First(“B”)={“cat”,”dogs”,”bear”,”girl”,”chocolate”,”rifle”,”young”,”older”,”smart”}
Follow (“NP1”) ={“with”}
Follow(“Nominal”)={“like”,”likes”,”thinks”,”shot”,”smells”,”with”}

Follow(“N”)={“like”,”likes”,”thinks”,”shot”,”smells”,”with”}
Follow(“Adjs”)={“cat”,”dogs”,”bear”,”girl”,”chocolate”,”rifle”}
Follow(“Adj”)={“young”,”older”,”smart”}
Follow(“VP”)={“$”,”with”,”like”,”likes”,”thinks”,”shot”,”smells”,”cat”,”dogs”,”bear”,”girl”,”chocolate”,”rifle”,”chris”,”fluffy”,”the”,”a”,”young”,”older”,”smart”}
Follow (“VP1”) ={“with”}
Follow (“V”) ={“undefined”}
Follow(“PP”)={“like”,”likes”,”thinks”,”shot”,”smells”,”with”,”cat”,”dogs”,”bear”,”girl”,”chocolate”,”rifle”,”chris”,”fluffy”,”the”,”a”,”young”,”older”,”smart”,”$”}
Follow(“Prep”)={“like”,”likes”,”thinks”,”shot”,”smells”,”with”}

Parsing table for CNF Introduced LL(1) English Grammar is shown in appendix II. Parse tree drives the construction of input word from grammar (G). Let I Є G(Z), where I is the input lexeme, along with grammar (G), and production rules (Z).

Tree nodes are terminal and non-terminal variables and in figure 3(a) parse tree of the input string “the dogs shot” for “CNF Introduced LL(1) English Grammar” is below.

Tree nodes for fig 3(a) are the, dogs, and shot along with the level of 5, depth 4d, leaf nodes (the, dogs, shot), the root node (S).

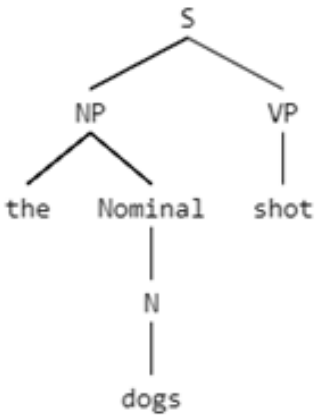

Fig 3(a) Parse tree of input string “The dogs shot” for “CNF Introduced LL(1) English Grammar”.

Below fig 3(b) is the derivation of input string “the dogs smells” for “CNF introduced LL(1) English Grammar

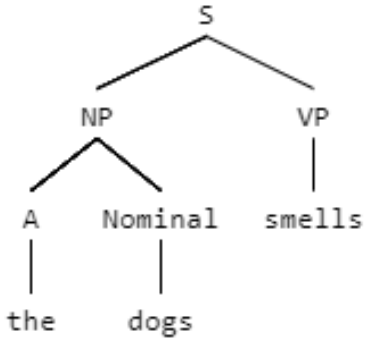

Fig 3(b) Parser tree of input string of “the dogs smells” for “CNF introduced LL(1) English Grammar”.

Tree nodes for fig 3(b) are the, dogs, and smells along with the level of 4, depth 3d, leaf nodes (the, dogs, smells), the root node (S).

RESULTS & DISCUSSIONS

During the study, 50 English Grammar input stacks () are evaluated with a traditional LL(1) parser and parsed with CNF introduced LL(1) parser The steps involved for parsing are given below inline charts.

Figure 4 Line chart for "LL(1) English Grammar".

Above "LL(1) English Grammar" line chart displays Total No. of Steps (8, and 9) for Input Stack (2 and 3). Below is the line chart for "CNF Introduced LL(1) English Grammar

Figure 5 Line Chart for "CNF Introduced LL(1) English Grammar".

The above line chart depicts Input Stack (2, and 3) for CNF Introduced LL(1) English Grammar with Total No. of Input symbols (6, and 8) performed. The introduction of CNF into LL(1) improves the processing but is not significant enough.

The descriptive statistical analysis gives knowledge for the Mann-Whitney U test. Descriptive statistics compare the (LL(1) and CNF Introduced LL(1)) parsers. The mean, median, standard deviation, skewness, and Kurtosis are also mentioned in the table.

Table 1 Normality Tests for "LL(1)" and "CNF Introduced English Grammar

	Kolmogorov-Smirnov			Shapiro-Wilk		
Gram-mar	Sta-tisti c	Df	Sig	Statis-tic	Df	Sig
LL (1)	.380	50	.000	.627	50	.00 0
CNF	.267	50	.000	.743	50	.00 0

Normality tests are conducted on each grammars. The normality take a look at of grammar (CNF) changed into assessed and the Shapiro-Wilk check indicated that the rating was W (50) =.000. The normality test of grammar LL(1) became also assessed and the Shapiro-Wilk test indicated that W (50) =0.000.

For "LL(1) English Grammar" and "CNF brought LL(1) English Grammar" underneath show the ratings of grammars (LL(1)) Mdn =57.01 became better than (CNF added LL(1)) Mdn =forty three.99 "

Table 2 Data Analysis for "LL(1) English Grammar" and "CNF Induced LL(1) English Grammar".

Parser	Mean	Median	Std. dev	Min	Max	Inter Quartile Range
LL(1)	8.42	8.00	.499	8	9	1
CNF based parser	7.62	8.00	1.338	6	9	3

The facts evaluation for "CNF Introduced LL(1) English Grammar" data evaluation is proven above. The information evaluation, normality assessments are conducted to determine the variety of everyday distribution compiled from random samples to unique information.

Table under for "LL(1) and CNF Introduced LL(1) English Grammar" display the scores of grammars (LL(1)) Mdn =57.01 became better than (CNF Introduced LL(1)) Mdn =43.99. Mann-Whitney take a look at is carried out on the steps concerned throughout parsing and ranks are shown in table.

Mann-Whitney take a look at is carried out on the steps concerned throughout parsing and ranks are shown in table

Table 3 "Mann-Whitney Test", Ranks.

Parser	N	Mean Rank	Sum of Ranks
LL (1)	50	57.01	2850.50
CNF Introduced LL (1)	50	43.99	2199.50
Total	100		

A Mann-Whitney check indicated that this distinction was no longer statistically enormous. A Mann-Whitney take a look at indicated that this distinction became now not statistically good sized, U (NLL (1) =50, NCNF added LL (1) = 50 =924.500, Z =-2.419, p <.016.

The projected CNF introduced LL(1) algorithm and conventional and LL(1) algorithm are comparable, however CNF added LL(1) parser takes does now not take notifiable distinct steps to technique input string in comparison to standard LL (1) parser. CNF added LL(1) parser manner time develops linearly with the quantity of steps worried in processing lexeme.

enter Stack and outcomes for conventional LL(1) and CNF added LL(1) English Grammar shown under.

Table 4 Input Stack for Conventional "LL(1)" and "CNF Introduced LL(1) English Grammar".

Serial Number	Input Stack	Steps for LL (1)	Total Number of symbols for LL(1) and CNF Induced LL(1)	Steps for CNF Induced LL (1).
1	cat shot	8	2	6
2	cat like	8	2	6
3	bear like	8	2	6
4	bear shot	8	2	6
5	dogs shot	8	2	6
6	girl like	8	2	6
7	girl shot	8	2	6
8	cat likes	8	2	6
9	dog likes	8	2	6
10	bear likes	8	2	6
11	cat smells	8	2	6
12	girl likes	8	2	6
13	cat thinks	8	2	6
14	dog smells	8	2	6
15	bear thinks	8	2	6
16	dogs thinks	8	2	6
17	bear smells	8	2	6
18	girl thinks	8	2	6
19	girl smells	8	2	6
20	the cat shot	9	3	9
21	the cat like	9	3	9
22	the bear like	9	3	9
23	the bear shot	9	3	9
24	the dogs shot	9	3	9
25	the girl like	9	3	9
26	the dogs like	9	3	9
27	the girl shot	9	3	9
28	the cat likes	9	3	9
29	the dog likes	9	3	9
30	the bear likes	9	3	9
31	the cat smells	9	3	9
32	the girl likes	9	3	9
33	the cat thinks	9	3	9
34	the dogs smells	9	3	9
35	the bear thinks	9	3	9
36	the bear smells	9	3	9
37	the girl thinks	9	3	9
39	fluff like	8	2	8
40	fluff likes	8	2	8
41	fluff thinks	8	2	8
42	fluff shot	8	2	8
43	fluff smells	8	2	8
44	chris like	8	2	8
45	chris likes	8	2	8
46	chris thinks	8	2	8
47	chris shot	8	2	8
48	chris smells	8	2	8
49	a girl smells	9	3	8
50	a girl thinks	9	3	8

CONCLUSION

The data analysis shows that the introduction of CNF and LL(1) parsers give birth to a hybrid CNF introduced LL(1) algorithm, which on the other hand works similar to the traditional one. Introducing this study and results provide some knowledge in the relevant field to achieve parser efficiency and often have grammatical approval parser effect.

LL(1) algorithm does not perform efficiently with the unit, useless, and ? production sets, hence for the grammar to be in LL(1) it has to be unambiguous. If ? production appears then they cancel out the whole production sets and let the algorithm go in a halt state.

The first part of the limitation comes with LL(1) and the second with CNF grammar structure. The CNF grammar structure itself restricts the grammar production from left- the hand side to only two variables. This CNF grammar structure sometimes leads the productions rules to divide into more production rules that cause the algorithm to not parse complete input lexeme that were parsed before by the conventional LL(1) algorithm. The increased number of productions of CNF introduced LL(1) algorithm also adds to overall processing speed, time, and complexity.

The planned work does not aim to consider the left-recursive and look-ahead symbols. The planned work is aimed to improve the prediction performance of LL(1) parsing, but not to increase the ambiguous grammar acceptance capability of the parser. Future work can introduce traditional LL(1) parsers to other forms of algorithms to check their significance and overall parsing speed.

REFERENCES

[1]. Kuhl, P. K., Williams, K. A., Lacerda, F., Stevens, K. N., & Lindblom, B. (1992). Linguistic experience alters phonetic perception in infants by 6 months of age. Science, 255(5044), 606-608.

[2]. Kuhl, P. K., Andruski, J. E., Chistovich, I. A., Chistovich, L. A., Kozhevnikova, E. V., Ryskina, V. L., ... & Lacerda, F. (1997). Cross-language analysis of phonetic units in language addressed to infants. Science, 277(5326), 684-686.

[3]. Meduna, A. (2014), Formal Languages and Computation: Models and Their Applications, CRC Press, p. 233.

[4]. Chomsky, N. (1956). Three Models for the Description of Language. Vol. 2. Issue. 2. Page. 113-123: IRE Transactions on Information Theory.

[5]. Chomsky, N. (1957). Syntactic Structures: The Hague, Mouton.

[6]. Jeuring, J. (2006). Applications of Grammars: Citeseer.

[7]. Ali, H., Naveed, M., Naseem, D., & Shabbir, J. (2020). LL (1) Parser versus GNF inducted LL (1) Parser on Arithmetic Expressions Grammar: A Comparative Study. Quaid-E-Awam University Research Journal of Engineering, Science & Technology, Nawabshah., 18(2), 89-101.

[8]. M. S. Naveed, (2017). "The Impact of Terminal Prefixing on LL (1) Parsing", J. Appl. Environ. Biol. Sci, vol. 7, No. 5, pp. 64-76.

[9]. Bernd Kuhl, Axel-Tobias Schreiner. (2000). An Object-

oriented LL(1) parser generator.

[10]. Faraah M. Dabhoiwala.(November 2014). A Review on LL (1) Parser (Vol. 2, Issue XI). ISSN: 2321-9653

[11]. Lasser, S., Casinghino, C., Fisher, K., & Roux, C. (2019). A verified LL (1) parser generator. In 10th International Conference on Interactive Theorem Proving (ITP 2019). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[12]. Milner, R. (1978). A theory of type polymorphism in programming. Journal of computer and system sciences, 17(3), 348-375.

[13]. Spenke, M., Muhlenbein, H., Mevenkamp, M., & Beilken+, C. (1984). A Language Independent Error Recovery Method for LL (1) Parsers (Vol. 14, Issue I).

[14]. Shilling, J. J. (1993). Incremental LL (1) parsing in language-based editors. IEEE transactions on software engineering, 19(9), 935-940.

[15]. James A. Femister. (1986). LPARSER, AN LL (1) PARSER GENERATOR.

[16]. Parr, T. J., & Quong, R. W. (1995). ANTLR: A predicated-LL (k) parser generator. Software: Practice and Experience, 25(7), 789-810.

[17]. Femister, J. A. (1986). LPARSER, an LL (1) parser generator (Master's thesis, Lehigh University).