2.4 **Combinational Categorical Grammar (CCG)**

Parsing natural language has always been a daunting task. Recent work in this area like Berkeley parser (**Johnson, 1999**), Stanford parser (**Klein, 2010**) has been successful in developing customized parsers that parse natural language by a pre-defined specification of grammar rules. In this work, we have used a methodology similar to the work of **Zettlemoyer (2009)**. A *Combinational Categorical Grammar (CCG)* is a formal grammar mechanism that is lexicalized and the grammar's rules are entirely based on combinatory logic. Linguistically speaking, a grammar is *lexicalized* when it is formed from individual words, lexical morphemes. A *morpheme* is the smallest grammatical unit in a language. Say, word **cannot** is lexicalized from the two words **can** and **not**. One of the goals of CCG is to assign natural language a syntactic structure. Say, For example, the sentence: **The book is on the table in the box** can be analyzed and understood in two different connotations. Sentences from natural language are assigned a grammatical structure using CCG to avoid this ambiguity. To understand how CCG forges relations and assigns a structure to sentences from natural language, the underlying components of CCG are presented in detail as below:

- *Categories:* They specify the list of constituents that make up CCG
- *Lexicon:* It assigns lexical categories to words
- *Combinator and rules:* They specify how the individual entities combine and what combinations are valid.

A detailed description of the components that make up CCG and its strategy follows:

A *category* is a syntactic category in a grammar specification. A lexical category can be either an atomic category such as: **S, NP, PP** or may be a complex lexical category. An

atomic category is one that is simple with no usage of operators and slashes. An example of a atomic lexical category is given as: **N, NP, S** and **NNP**. An atomic category is also known as simple category as it does not involve any rules or combinations. A *complex lexical category* is one that is built from basic atomic categories applied along with some rules. An example of a complex lexical category that maps a phrase in natural language is given as: **walked - S\NP**. The backward slash indicates that given an atomic category **S**, atomic category **NP** follows. This notation of using a slash (/ or \) along with syntactic category is termed as *Lambek notation.* By employing Lambek notation, complex categories can be derived. Some examples to illustrate these are given below:

- Intransitive verb: **S\NP**
- Transitive verb: **(S\NP)/NP**
- Adverb: **(S\NP)\(S\NP)**
- Prepositional phrase: **(NP\NP)/NP**

Interaction between categories, semantic and syntactic types are presented in the next section.

A CCG *lexicon* is defined as an entity that specifies all of the categories that a given word can have. Thus, a lexicon assigns categories to words. Combinatory rules then help in identifying the combinations that are valid and not valid. An important mechanism involving combinators and rules is functional application.

Functional application refers to the process of combining function with its argument. An example to illustrate this process is given as:

Which player scored …?

NP S S\NP

Two important rules that are used for function application are forward and backward application. By employing forward and backward application of rules in conjunction with combinators, lexical derivations are obtained. Given A, B two lexical categories is given as below:

A/B	B	⇒	A	-	Forward application (>)
B	B\A	⇒	A	-	Backward application (<)

A '/' indicates that given a category B to the left, category A follows the function application. From the example presented above, forward application can be performed when either atomic or complex lexical categories present in the syntax as discussed. Similarly a '\' indicates that given a category A to the right, category B follows. In function application the direction of the slash (either forward or backward) indicates the position of the argument with respect to the function. A sentence from natural language and its corresponding mapping in atomic and complex lexical categories to illustrate forward and backward slashes is presented as below:

played	*matches*	*at*	*Lords*
(S/NP)\NP	**NP**	**PP**	**(S/NP)**

The rules presented above illustrate how lexical categories are inferred. An *inference* in Machine Learning terms is similar to a conclusion. An inference is drawn when it is nearly certain that an event happens. For a simple grammar, a formal system can be deduced based on categorical grammars and thereby, derive an expression through parsing it from natural language and produce its resulting tree structure as follows. Some syntactic categories such as S, N, NNP, NP are utilized for this task. An example from

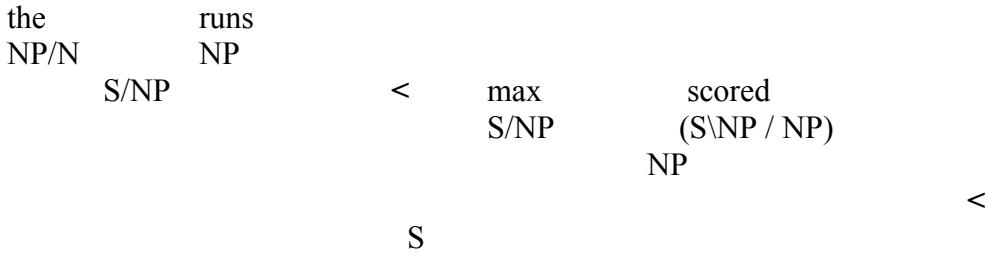
the Cricket domain that works by employing the aforementioned rules is given as:

scored the max runs

The categorical type information for this sentence is presented as:

scored	*the*	*max*	*runs*
(S\NP / NP)	**(NP/N)**	**S/NP**	**NP**

The categorical inference drawn from this sentence in natural language is given as:



Thus the categorical inference for prediction has been made as S, proving the face that the model has made a connection to a word in natural language such as What, Who, Which Whom.

A *combinator* acts as a basis for design and development of programming languages. A combinator is to Computer Science, what a variable is to Mathematics. Combinators simplify the usage of variables and help with simplification of grammar. Apart from forward and backward functional application, some common combinatory rules are forward type-raising, type-composition. Some complex lexical categories that employ forward and backward slashes and map phrases from natural language sentences may be given as: **respected – (S\NP)/NP**, **gave – ((S\NP)/NP)/NP**. The approach that is followed within this work with CCG allows for capturing certain kinds of relationships that might

not be possible to explore with any other approach. An example to present this methodology is given as below:

What is the name of the player who scored…?

Give me the name of the player …?

Who scored …?

Speaking in terms of natural language, all of the questions mentioned above, point to the same answer. For a machine to be able to effectively understand the same, requires it to understand how relations are forged in between words in natural language. For this task, CCG is employed. CCG is chosen and it is formalized to this thesis by using the works of **Elbridge (2002) and Zettlemoyer (2009).** For working constraints the lexical items derived from the parsing mechanism and the grammar would be in triplet form presented as:

word |-- syntactic category : logical form

An example to present this can be given as:

SACHIN |-- NP : player

Where SACHIN is the name of a corresponding lexeme, NP is the syntactic category in this case, player denotes the corresponding form, say, For example: odi, test. The major advantage of using a CCG grammar as devised above is that it erases the boundary between syntax and semantic structure of the logical items. The idea that drives this approach, is that CCG categories coupled with other information may be used as syntactic arguments in a parse. The order in which a syntactic category combines with its syntactic arguments is the same as the order in which the associated logical-expression combines with its semantic arguments. The '**λ**' operator is used to define and relate

semantic functions in our calculus. Every categorical grammar thus consists of basic syntactic categories some basic and some complex. CCG elucidated as part of this thesis typically consists of basic syntactic entities or atomic categories and interaction amongst these categories by making use of operators. Any number of CCG queries can then be generated basing on these entities and relations among them. An example to present this: Let the category of individual unit entities i.e. syntactic categories be {S, NP, N, PP, . . . Let the grammatical relation that needs to be generated be one for an ODI player. This is represented as: **player |-- (S\NP)/NP : x, y odiplayer (y, x)**

After building sentences and examining derivations, a mechanism is needed to parse through the derivations. The Cocke Younger Kasami (CKY) algorithm parses through sentence derivations along with meanings. Presented below is a generalized version of the CKY algorithm that parses through categorical grammars whereas the actual CKY algorithm specific to this thesis is presented in the Implementation section.

Let **w[p,d]** be the substring of w of length d starting from the p^{th} symbol of w. Let **P[i,p,d]** be the boolean array where P[i,p,d] means that x_i derives w[p,d]

Input: Sentence [$w_{0:1}$, . . . , $w_{n1:n}$], Lexicon □, CCG combinators Comb Output: A Filled Chart

1. **Initialize all P[i,p,d] to false.**
2. **For all i from r+1 to m, and for all p from 1 to n, if x_i = w[p,1], then assign P[i,p,1] to be true.**

3. **For each production of the form x_j -> x_i, where j is between 1 and p (i.e., x_j is a variable) and i is between r+1 and m (i.e., x_i is a terminal), and for each p from 1 to n, if P[i,p,1], then assign P[j,p,1] to be true.**
4. **Execute the following for all d, starting with d=2 and ending with d=n.**

    1. **Execute the following for every production of the form x_j -> x_jx_k**

        for all p from 1 to n-d+1,

        for q from p+1 to p+d-1,

        if P[j,p,q-p] and P[k,q,d+p-q] then assign P[i,p,d] to true.

    2. **If P[1,1,n] then return Yes else return No.**

Algorithm 2.1 The Cocke-Younger-Kasami Parsing Algorithm

A major advantage with CYK algorithm is that it not only determines whether a sentence belongs to the denoted grammar but also if its follows the specified syntax as provided by the grammar. With parsing, it builds up a parse tree from the individual nodes thus completing the parse tree.