# Annotation and PCFG for Disambiguation in Treebank Parsing

Timothy Liu
University of California at Berkeley
CS294-19
timothytliu@berkeley.edu

Dounan Shi
University of California at Berkeley
CS294-19
dounan@berkeley.edu

**Abstract**

This paper presents a detailed first look at parsing a natural language. The main focus is building a parser that can process text with its corresponding structural syntax. The process of building a parser is introduced including various methodologies to choose between. The CKY Parser is the chosen methodology and the algorithm is also carefully explained. Following this, statistical error analysis and parsing behaviors are presented to highlight areas of improvement. Several extensions are provided that improve the test results and focus on problematic areas within parsing. The final analysis generalizes the results and observations to the entire topic of natural language processing.

**General Terms**

Lexicon, Probabilistic Context Free Grammar, CKY Algorithm

**Keywords**

Grammar Rules, Markovization, Precision, Recall, $F_1$

## 1. Introduction

Parsing or syntactic analysis is a process in both linguistics and computer science that deals with written text. Among many real world applications, some specific examples are Human-Computer Interaction (HCI) and Machine Translation. Parsing determines grammatical structure according to a specific grammar by analyzing a sequence of tokens or words in text. Therefore, a parser requires specific components: a precise description of a language called a grammar, a process or algorithm to structure tokens of text into constituents, and a way to verify the phrases or constituents correspond to the given grammar.

Three units of measurement are often used to characterize the proficiency of parsers.[1]

*Precision* - The fraction of correct constituents in the guess parse out of the correct constituents in the gold parse.

$$P = \frac{\#\text{ of correct constituents in hypothesis parse of s}}{\#\text{ of correct constituents in reference parse of s}}$$

*Recall* - The fraction of correct constituents in the guess parse out of all constituents in the guess parse.

$$R = \frac{\#\text{ of correct constituents in hypothesis parse of s}}{\#\text{ of total constituents in hypothesis parse of s}}$$

*Accuracy, $F_1$* - Harmonic mean of per-node labeled precision and recall.

$$F_1 = \frac{1}{\frac{1}{P} + \frac{1}{R}}$$

[1] http://acl.ldc.upenn.edu/J/J03/J03-1002.pdf

**2. Parsing**

In order to parse sentences, we first construct a corpus of grammars from our training corpus. For our experiments, we will use Probabilistic Context-Free Grammar (PCFG) which is defined by four parameters: $N, \Sigma, R$, and $S$.

$N$ - a set of non-terminal symbols (variables).

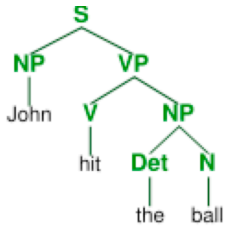
$\Sigma$ - a set of terminal symbols (disjoint) from N, where the terminal symbols are the actual words of the sentence.

$R$ - a set of rules or productions, each of the form $A \rightarrow \beta$ [p], where $A$ is a non-terminal, $\beta$ is a string of symbols from the infinite set of strings ( $\Sigma \cup N$)*, and p is a number between 0 and 1 expressing $P(\beta \mid A)$.

S – a designated start symbol.

In order to simplify parsing, we will annotate our grammars so that the resulting grammar rules $A \rightarrow \beta$ [p] are either of the form $A \rightarrow X$ [p] (unary) or $A \rightarrow XY$ [p] (binary). While similar to CNF, our grammar allows for unary rules other than from pre-terminal nodes to terminal nodes.

We can then build parse trees from these rules:




**3. Ambiguity**

There are two types of ambiguities which plague parsing. The first is part-of-speech ambiguity, as discussed in project 3. The other is structural ambiguity. Given a sentence, there are multiple parse trees that fit the grammar. Structural ambiguity can be further divided into two subclasses of ambiguity: attachment ambiguity and coordination ambiguity.

Attachment ambiguity can be seen in this example, “We saw the Eiffel Tower flying to Paris.” In this sentence, “flying to Paris” can mean that the Eiffel Tower itself is flying to Paris. Conversely, “flying to Paris” can be an adjunct that modifies the header “saw.”

An example of coordination ambiguity is “old men and women.” This phrase can be interpreted in two ways: “[old [men and women]]” or “[old men] and [women].”

These ambiguities can be handled with fair accuracies by using PCFG. Since each grammar rule is given a probability, the probability of a parse tree T can be obtained by:

$$P(T,S) = P(T) * P(S \mid T) = P(T) = \prod_{i=1}^{n} P(RHS_i \mid LHS_i)$$

By picking the parse tree with the highest probability, we can solve many ambiguity issues. However, as we will see, the most probable tree is not always the correct tree. Just because a certain grammar rule occurs often, it does not mean it should always be applied. This problem can be partially solved by annotating our grammar. This makes our grammar rules much more specific and thus allows for assignments that can better match specific cases.

**4. Annotations**

A PCFG differentiates between parses through a statistical manner, allowing us to choose the more likely parse based on how often specific constructions of phrases occur. However, often times the probabilities themselves do not accurately model natural language, which is much more complicated. Two specific flaws appear in PCFGs: poor independence assumption and lack of lexical conditioning.

The probabilities use a naive construct placing too much importance on the independence of phrases, omitting the extremely dependent structure of language. An example is that a Noun Phrase (NP) has three totally different

distributions of structure depending on the parent of the Noun Phrase. Furthermore, ambiguities aforementioned still remain because the grammar rules do not model syntactic facts about specific words. There is no way to differentiate one rule from another without differentiating words and whether it should be left branching or right branching.

A way to address these problems is to introduce annotations. The typical grammar rules in a PCFG leave room for improvement, so adding annotations to the grammar create dependencies among the rules and change the rules to focus on specific words. This process creates a lexicalized grammar whose new rules are formed by these methods: head annotation, Markovization, tag splits, hierarchical refinement, and adaptive splitting. Therefore, annotations refine the grammar to improve statistical fit of the grammar.

The parser in this paper makes use of binarization, lexicalization, Markovization, and tag splitting. Binarization in this case creates a new tree out of the sibling nodes on the same level. The left most node is kept the same, but a new right node is introduced that is the parent of the rest of the siblings. This process propagates down to binarize the entire tree. These intermediate nodes become new rules to show dependency.

To approach lexicalization, more information is added onto the parent nodes of each tree. This correctly differentiates between different attachments and whether or not to branch left or branch right. Horizontal Markovization is accomplished by keeping track of siblings as the tree is binarized. Vertical Markovization is accomplished by keeping track of the parents of the node in the tree. These create new dependencies, as now the rules are a combination of both depth and breadth. Tag splitting is simply using the pre-terminal tags for the words and making them less ambiguous, helping the parser to focus on specific words or groups of words.

For more in depth annotations, further experiments were conducted on the type of annotations used. A new combination of annotations adding the future sibling or future child and grandchild to a specific node to either of the two left and right branch nodes made for a considerable extension of the current annotations. These new annotations will be later examined for usefulness.

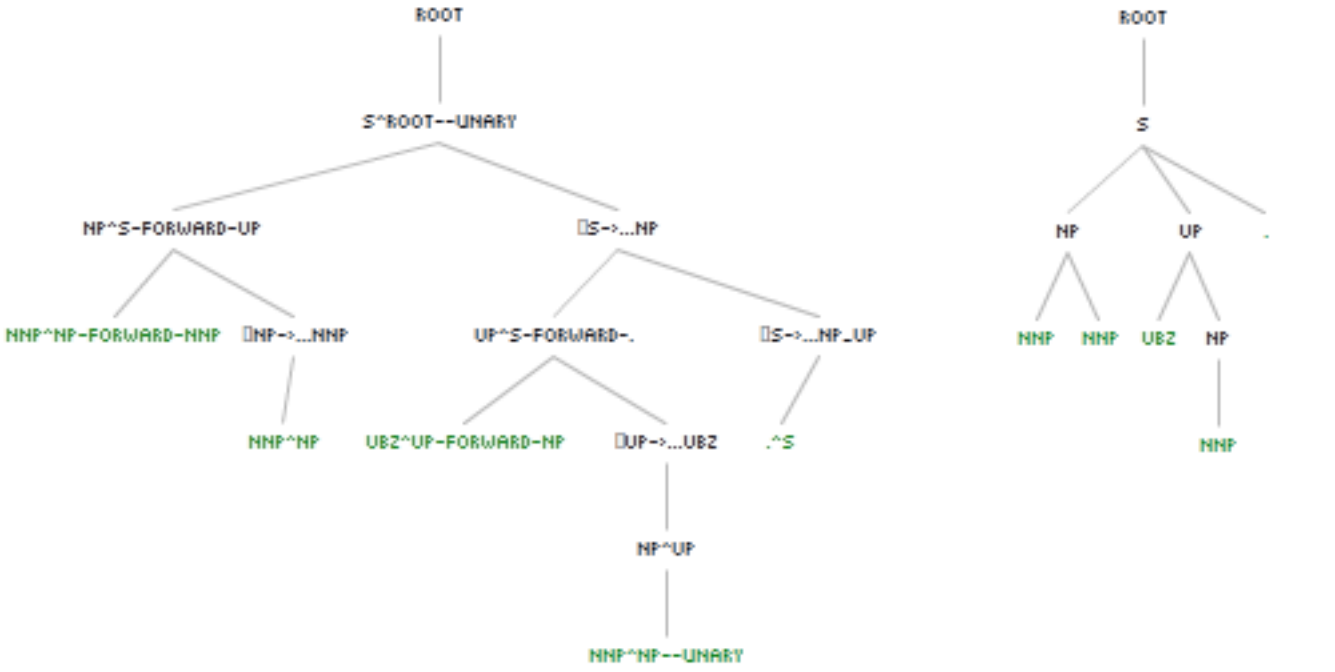



**Figure 4.1 – Annotated tree (left) and its un-annotated counter part (right)**

Noticeably, the final outcome of a parsed sentence does not includes weird symbols; therefore, it is necessary to un-annotate the parse tree after the optimal parse has been obtained using the grammar rules and added annotations. As a result, many annotations create more grammar rules; however, too many rules can cause over fitting to the specific training parse sentences and worsen the optimal parse. The problems within PCFGs must be remedied; therefore, annotations are used to create a better statistical model.

**5. CKY Algorithm**

In order to build the best parse tree, we must find the tree T with the highest P(T) as described above. This is done through a dynamic program known as the CKY algorithm.

Before defining the dynamic algorithm, we first introduce the concept of span. For each sentence of length n, numbers 0 to n are inserted at the beginning, between, and end of the sentence to separate the words. For example:

$$_0\ This\ _1\ paper\ _2\ is\ _3\ the\ _4\ best\ _5$$

Therefore, a span from 1 to 3 would include the words "paper is."

The CKY algorithm sub-problem is defined as follows:

$Score[i,j,A]$ = the best score of the sub-tree T' spanning i to j starting with non-terminal A.

For binary rules: $Score[i,j,A] = \max_{A \to XY[p]} \max_{i<k<j} \{p * Score[i,k,X] * Score[k,j,Y]\}$

For unary rules: $Score[i,j,A] = \max_{A \to X[p]} \{p * Score[i,j,X]\}$

Base cases: $Score[i,i+1,A]_{\text{for } 0<=i<n} = \max_{A \to X[p],\ X \in \sum} \{p\}$

Desired Answer: $Score[0,n,ROOT]$

When implementing the algorithm, we must make sure that when calculating a span of size s, all spans of size less than s must already exists. This means that all binary rules must be looped through first for a given span before any unary rules can be checked. Also, we must start with span of size 1 and move up to size n.

In order to build the tree, a back-trace is stored which keeps track of which rules were used to get the highest score for span. While not the most efficient, this method does the job.

**6. Experiment Data**

**Table 6.1 – Results for Baseline Parser and CKY Parser with various annotations**

| | Precision | Recall | F1 | EX |
|---|---|---|---|---|
| Baseline Parser | 16.3 | 3.17 | 5.31 | 0 |
| 1st Order Vert./Inif. Order Horz. | 74.57 | 69.43 | 71.91 | 9.04 |
| 2nd Order Vert./ 2nd Order Horz. | 81.03 | 79 | 80 | 15.15 |
| 2nd Order Vert./2nd Order Horz. + 1 Forward | 82.19 | 81.37 | 81.78 | 16.75 |
| 2nd Order Vert./2nd Order Horz. + 2 Forward | 81.72 | 81.32 | 81.52 | 16.48 |
| 2nd Order Vert./ 2nd Order Horz. + 1 Forward + Unary Label on Child | 82.22 | 82.16 | 82.19 | 16.75 |
| 2nd Order Vert./ 2nd Order Horz. + 1 Forward + Unary Label on Parent | 82.15 | 81.46 | 81.8 | 17.55 |
| 2nd Order Vert./2nd Order Horz. + 1 Forward + Tag Split | 82.09 | 81.4 | 81.74 | 17.28 |
| 2nd Order Vert./2nd Order Horz. + 1 Forward + Unary Label on Child + Tag Split | 82.15 | 81.46 | 81.8 | 17.55 |

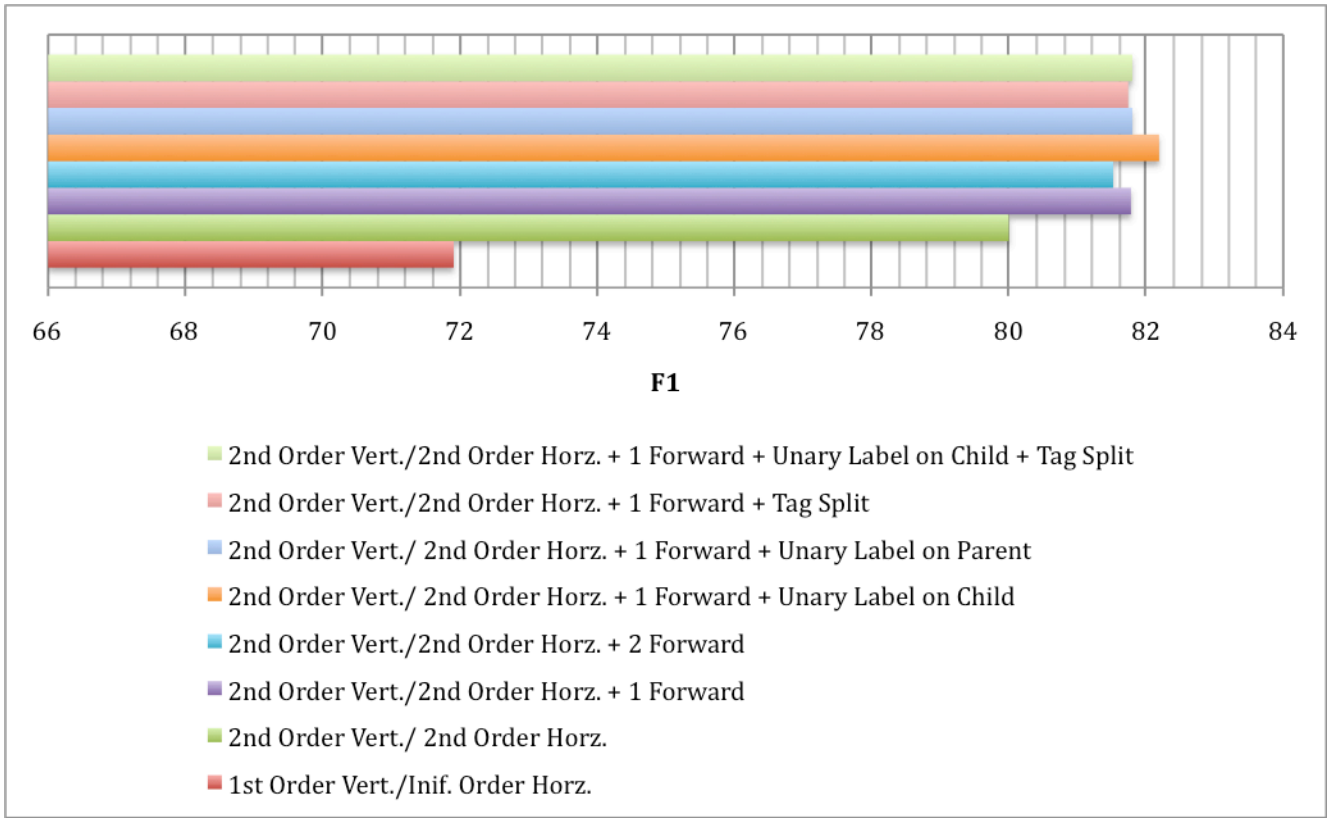



**Figure 6.1 – F1 values for CKY parser with various annotations**

**7. Error Analysis**

The baseline parser does not accurately represent the syntactic structure of natural language in the slightest sense. The way the sentences are parsed, the phrases are always right branching meaning that all language syntactically incorporates larger phrases as the sentence lengthens. As expected, the results in **Table 6.1** show that the $F_1$ accuracy of the right branching parser is a pitiful 5.31.

The completely finished CKY Parser uses probabilistic models to determine the most likely parse of phrases, which better represents syntax in language. In Table 6.1 the $F_1$ accuracy increased to 71.91. This is because the default CKY Parser contains infinite order horizontal Markovization and first order vertical Markovization. With infinite order horizontal Markovization or simply keeping a history of all siblings that have appeared before the current node, contextual information is grasped. This is analogous to an infinite-order n-gram in an HMM. A shared problem however is over fitting and sparsity, meaning the model has become too fit or specific to the training data and less effective on the test data. The first order vertical contains no contextual information and does not help at all.

The next change was to find the optimal order both vertically and horizontally for the CKY parser. The result was second order horizontal and vertical Markovization. The second order horizontal disambiguates like the infinite order model; however, it does not over fit or become too sparse. The second order vertical Markovization disambiguates between identical symbols with different parents. This more accurately represents how syntax is multiple-level tree like. It also helps show phrasing begets more phrasing because more symbols are produced helping to create new grammatical structure. This new CKY Parser produced an even better F1 score of 80.

Annotations have various forms and can target specific problems. A specific extension of the annotations includes the notion of differentiating unary tags. Unary tags occur in two cases: modifying the parent and modifying the child. The first occurs when only parent *p* knows that *p* has only one child. The second occurs when child *c* knows that it is an only child, instead of the parent p knowing that p has only one child. With the second case, a second order

Markovization will let the children of *c* know that they are coming from an only child. Both of these help to distinguish phrasal structure and dependencies beyond the second order vertical Markovization.

Another aspect of annotations is a variation of unary and Markovization that looks for context in the future. It is simply second order +1 forward horizontal Markovization. As there is no context about the future in our parse trees, this helps to differentiate syntactic phrases that are about to split. This resulted in our best $F_1$ score of 82.19.

The final extension is focusing on the tags themselves. Ambiguous tags like '*IN*' often show up in different syntactic clauses; therefore, there are clusters of words that need to be separated. However, grammatically many of these words contain the same parts of speech, but are often parsed and included in different syntactic structures and phrases. When words are tagged incorrectly, the phrasal structure will change because different rules will apply. Therefore, tag splitting can create new tags that separate clusters with different structure, which all used to be under the same tag.

Our results show no improvement from tag splitting. Our annotations have already split these word clusters enough, thus adding tag splitting only caused sparsity within our grammar. This in turn resulted in the slight decline in performance.

Looking at the tree in **Figure 7.1**, we see that not enough context exists to correctly put "one" in the correct phrase. As you can see, CD's parent is a NP, so with this knowledge, it is more apt to put it under NP with JJ and NN as its siblings. However, when the second order +1 forward horizontal Markovization is applied (**Figure 7.2)**, the necessary information within the sentence is captured to put “one” in the correct place.

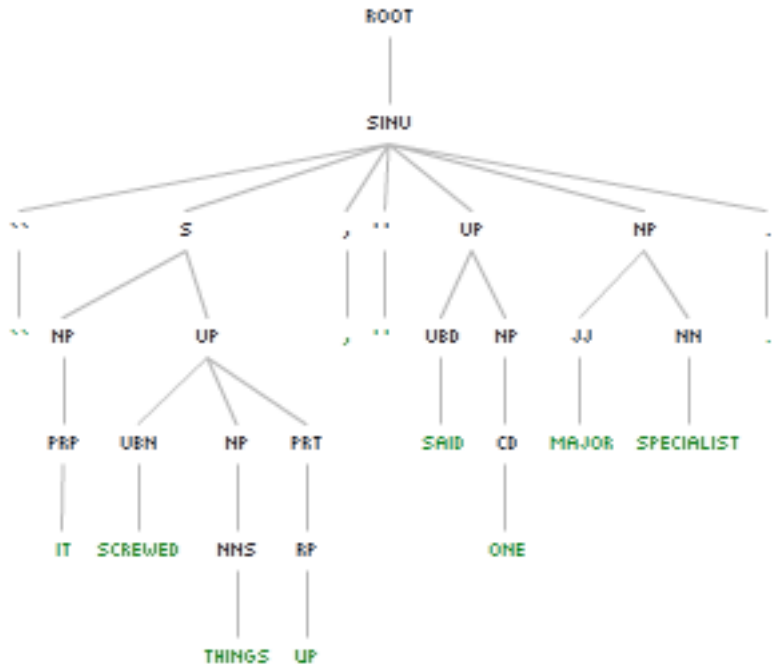



**Figure 7.1 – CKY Parse tree with 1st order vertical and infinite order horizontal Markovization.**

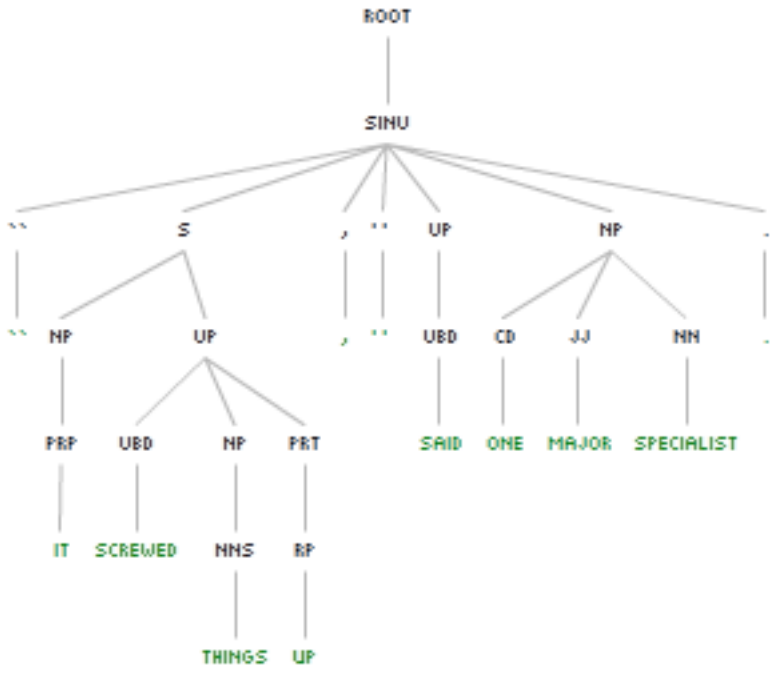



**Figure 7.2 – Correct CKY Parse tree with 2nd order +1 forward horizontal Markovization**

## 8. Conclusion

The parsing process requires various aspects to function correctly and efficiently. State of the art parsers can parse a sentence in one-tenth of a second for a sentence with over 40 words with as high as 90.2% accuracy. The parser in this paper parsed a sentence every several seconds and had a decent accuracy of 82.2%. Understanding the parts of a parser, the limitations and problems encountered, and the temporary solutions become valuable tools in the world of natural language processing. Parsing after all is the microcosm of Natural Language Processing.