Assignment 3: CKY algorithm for constituency parsing with PCFG

Aniruddha Bala
Indian Institute of Science
Bangalore, India
aniruddhab@iisc.ac.in

1 Introduction

In this assignment we implement a constituency parser using the CKY algorithm. The algorithm makes use of probabilistic context free grammar which is extracted from the training set. From the CKY algorithm we obtain the best parse for the given sentence. The evaluation is done on the test dataset and the labelled precision, labelled recall and F1 score are reported.

2 Dataset

For this assignment we use 10 percent of the Penn Treebank dataset which is available through the nltk api. The dataset contains 3914 parsed trees of english sentences. I have splitted the data into 80-20 split keeping 3131 trees for training and 783 trees for testing.

3 CKY Algorithm

The Coke-Kasami-Younger (CKY) is a dynamic programming algorithm to find the best constituency parse tree for a given sentence. The input to the algorithm is the tokenized words of a sentence and the grammar rules with their corresponding probabilities. Given these inputs the algorithm then builds up the table in a bottom up fashion. We first start by filling up the lexical entries (rules of the form $A \rightarrow word$) in cells corresponding span 1 along the table diagonal. Then we check for unary rules if any of the form $A \rightarrow B$ and fill up the cells calculating the net probability of the tree as $P(A \rightarrow B) * P(B \rightarrow word)$. In this process we also update the backpointer if we happen to find a path with better probability. Next we find the constituency over bigger spans taking binary rules of the form $(A \rightarrow B, C)$ into consideration. We vary the split from $(begin + 1, end)$ such that B belongs to span $(begin, split)$ and C belongs to span $(split + 1, end)$. We update the probabilities if we find a better parse till that point as $score[begin][split][B] * score[split][end][C] * P(A \rightarrow B, C)$ and also update the backpointer. This process is carried on till we reach span $(0, num\_words)$. We finally return the parse with maximum probability or the tree rooted at a nominated start category like S

4 Methodology

The first phase is training phase where we extract out the grammar and calculate the probabilities of the rules from the training set. To do this first I load the data using the nltk api, split it at 80-20 percent for training and testing. Then I preprocess the trees by converting each of the labels at the nodes to their base categories (for eg. the base category of NP-SBJ is NP). Then I convert all the trees to their chomsky normal forms such that each node has at max 2 children. This conversion is essential in order to guarantee worst case run time complexity cubic in sequence length and number of non terminals. The chomsky normal form is done by right factoring the trees and setting horizontal markovization as 2. The processed trees in the training set are used to extract out grammar rules and their corresponding probabilities. To extract out the grammar rules I do a simple BFS traversal on the tree and add the rules to a dictionary. The rule probabilities are calculated using the ML estimate on the training set. The formula for which is mentioned below

$$P(A \rightarrow B, C) = \frac{count(A \rightarrow B, C)}{count(A)}$$

and simillarly

$$P(A \rightarrow B) = \frac{count(A \rightarrow B)}{count(A)}$$

where A belongs to the set of non terminals and B, C belong to either non terminals or terminals.

Apart from this I also calculate the probability of occurence of each POS tag which I use in the smoothing technique. The formula for that is

$$P(A) = \frac{count(A)}{count(.)}$$

where $count(.)$ represents the total count of the tags over the training set. Now that we have these probabilities I calculate the smoothed estimate for a rule using two different techniques - interpolation and back-off using the following formulae.

Interpolation:

$$P_{interpolation}(A \rightarrow B, C) = \alpha * P(A \rightarrow B, C)$$
$$+(1 - \alpha) * P(A)$$

Back-off:

If rule in grammar :

$$P_{backoff}(A \rightarrow B, C) = P(A \rightarrow B, C)$$

otherwise:

$$= \alpha * P(A)$$

where α is a hyperparameter which controls the smoothing, Note that though the above formulae are written for rules of the type $(A \rightarrow B, C)$, they can be easily extended to rules of the type $(A \rightarrow B)$ by just replacing $P(A \rightarrow B, C)$ with $P(A \rightarrow B)$.

Smoothing is essential here because we have very less training data. So the probabilities estimated from the training data may be very high for few rules whereas very low for some other rules. Also for the words which are not present in our training set the pre-terminal rule $(A \rightarrow word)$ probabilities will be zero. Hence we need to start with some pre-terminal probabilities so that we don't end up getting an empty parsed tree for words in test sentence not in our training set. Since the tags are unknown for the new words in our test sentence, the best we can do in such case is use the statistics about the frequency of the tags in our training set. So the intuition here is if nouns are more frequent in my training data then the unknown word is more likely to be a noun hence use of $P(A)$ is justified. Finally for a given sentence we apply the CKY algorithm discussed above and return the score and the back pointer matrices. From the back pointer matrix obtained we start with the S node and build the the tree recursively and unchomsky normalize the tree. The final tree is returned, the returned tree represents the best possible parse for the given sentence with root as S.

5 Experimental Setup

At first I carry out the training and save the grammar file. Then during test time I load the grammar file and parse the test sentences using CKY algorithm. The parsed results and gold sentences are written to a file. These files are then used to carry out the evaluation. Here we find the labelled precision, labelled recall and the F1 score on the test data. To carry out the evaluation I have used the EVALB library https://nlp.cs.nyu.edu/evalb/ which is a C library for bracket scoring. The parameters used for scoring gives the same scoring behaviour as the scorer used in (Collins 97). The results are obtained on the entire test set and correspond to hyperparameter $\alpha = 0.9$ for interpolation technique and $\alpha = 0.8$ for back-off technique. These hyperparameters were selected based on the parser performance on the test set. Using the best parser I have compared the parses obtained for 2 short and 2 long sentences using existing parsers available online http://tomato.banatao.berkeley.edu:8080/parser/parser.html and http://nlp.stanford.edu:8080/parser/index.jsp.

6 Evaluation

Here I list out few of the formulae that the library makes use of to calculate different measures. # symbol is to be read as number of.

Labelled Precision =

$$\frac{\text{\#(correct constituents)}}{\text{\#(constituents in the parsed file)}}$$

Labelled Recall =

$$\frac{\text{\#(correct constituents)}}{\text{\#(constituents in the gold file)}}$$

In both of the above the constituents along with their brackets are matched to obtain the labelled precision and recall.

$$\text{F1 score} = \frac{2 * LP * LR}{LP + LR}$$

where LP and LR stand for labelled precision and labelled recall.

Complete match = percentage of sentences where

	All sentences	Sentences with length less than 40
Number of sentences	783	730
Number of error sentences	135	120
Number of valid sentences	648	610
Labelled Precision (%)	61.09	62.65
Labelled Recall (%)	55.25	56.63
F1 score (%)	58.02	59.49
Average crossing	4.06	3.55
Tagging accuracy (%)	86.50	86.48

Table 1: Result shows the performance of parser using interpolation smoothing with $\alpha = 0.9$ on the NLTK PennTreeBank Test Set

	All sentences	Sentences with length less than 40
Number of sentences	783	730
Number of error sentences	167	145
Number of valid sentences	616	585
Labelled Precision (%)	71.46	72.56
Labelled Recall (%)	68.64	69.65
F1 score (%)	70.02	71.08
Average crossing	3.02	2.65
Tagging accuracy (%)	89.54	89.50

Table 2: Result shows the performance of parser using back-off smoothing with $\alpha = 0.8$ on the NLTK PennTreeBank Test Set

precision and recall are both 100%
Average Crossing =

$$\frac{\text{\#(constituents crossing a gold file constituent)}}{\text{\#(sentences)}}$$

Tagging Accuracy = percentage of correct POS tags

7 Results

The results section contains two tables showing the performance on the test set with two different smoothing techniques used. The test set contains 783 sentences, complete evaluation on the test set takes approximately 650 mins. While evaluation the labels (TOP, -NONE-, , , :, “, ”, .) are ignored according to the standard parameter file (Collins 97). Deletion of these brackets causes length mismatch in some parses and hence the evaluator marks them as error sentences. The tables 1 and 2 therefore show the result on the valid sentences which are obtained after removal of these error sentences. The tables 3 and 4 show the comparison of my best parser (the one obtained with back-off smoothing) with two different online parsers. The figures show the same parses in tree structure. The comparison of the figure is done only with the Berkeley parser since the Stanford parser does’nt produce a figure for their parse.

8 Observations and Conclusions

From table 1 and 2 we see that the performance of the parser is better with back-off smoothing. The idea in interpolation is that we mix the two distributions and form a new distribution this will alter the existing probabilities of the rules, even those about which the parser is confident enough, whereas in back-off we maintain the original distribution $P(A \rightarrow B, C)$ but if the parser encounters unseen rules we back-off to a lower order distribution (here $P(A)$).

Analysis of parse outputs from different parsers: Tables 3 and 4 show the parses of short and long sentences respectively by my parser, Stanford’s parser and Berkeley’s parser. The Stanford and Berkeley’s parser use an additional ROOT tag to denote the root of the tree, however, in my parses this tag is absent to maintain uniformity with the NLTK Penn Treebank gold trees.

Short sentences: From table 3 sentence 1 we see that my parser incorrectly tags the words Virat, Kohli and cricket. This is because these words

Input sentence	Virat Kohli plays cricket.
My Parser	(S (NP (DT virat) (NN kohli)) (VP (VBZ plays) (NP (-NONE- cricket))) (. .))
Stanford Parser	(ROOT (S (NP (NNP Virat) (NNP Kohli)) (VP (VBZ plays) (NP (NN cricket))) (. .)))
Berkeley Parser	(ROOT (S (NP (NNP Virat) (NNP Kohli)) (VP (VBZ plays) (NP (NN cricket))) (. .)))
Input sentence	The movie will be released next week.
My Parser	(S (NP (DT the) (NN movie)) (VP (MD will) (VP (VB be) (NP (VBN released) (JJ next) (NN week)))) (. .))
Stanford Parser	(ROOT (S (NP (DT The) (NN movie)) (VP (MD will) (VP (VB be) (VP (VBN released) (NP (JJ next) (NN week))))) (. .)))
Berkeley Parser	(ROOT (S (NP (DT The) (NN movie)) (VP (MD will) (VP (VB be) (VP (VBN released) (NP (JJ next) (NN week))))) (. .)))

Table 3: Table shows the comparison of the parses from the three parsers for short sentence length < 10

Input sentence	After the end of their last match, Russell questioned the team's decision to send him lower down the order when the likes of Robin Uthappa struggled.
My Parser	(S (PP (IN After) (NP (NP (DT the) (NN end)) (PP (IN of) (NP (PRP$ their) (JJ last) (NN match))))) (, ,) (NP (-NONE- Russell)) (VP (VBD questioned) (S (NP (NP (DT the) (NN team)) (PP (IN) (NP (DT s) (NN decision)))) (VP (TO to) (VP (VB send) (NP (PRP him)) (ADVP (JJR lower) (PP (IN down) (NP (DT the) (NN order)))) (SBAR (WHADVP (WRB when)) (S (NP (NP (DT the) (NN likes)) (PP (IN of) (NP (DT Robin) (NN Uthappa)))) (VP (VBD struggled)))))))) (. .))
Stanford Parser	(ROOT (S (PP (IN After) (NP (NP (DT the) (NN end)) (PP (IN of) (NP (PRP$ their) (JJ last) (NN match))))) (, ,) (NP (NNP Russell)) (VP (VBD questioned) (NP (NP (DT the) (NN team) (POS 's)) (NN decision) (S (VP (TO to) (VP (VB send) (S (NP (PRP him)) (VP (VB lower) (PRT (RP down)) (NP (NP (DT the) (NN order)) (SBAR (WHADVP (WRB when)) (S (NP (NP (DT the) (NN likes)) (PP (IN of) (NP (NNP Robin) (NNP Uthappa)))) (VP (VBD struggled)))))))))) (. .)))
Berkeley Parser	(ROOT (S (PP (IN After) (NP (NP (DT the) (NN end)) (PP (IN of) (NP (PRP$ their) (JJ last) (NN match))))) (, ,) (NP (NNP Russell)) (VP (VBD questioned) (NP (NP (DT the) (NN team) (VBZ s)) (NN decision) (S (VP (TO to) (VP (VB send) (NP (PRP him)) (ADVP (RBR lower)) (PP (IN down) (NP (DT the) (NN order))) (SBAR (WHADVP (WRB when)) (S (NP (NP (DT the) (NN likes)) (PP (IN of) (NP (NNP Robin) (NNP Uthappa)))) (VP (VBD struggled)))))))) (. .)))

contd. on next page

Input sentence	Google began in January 1996 as a research project by Larry Page and Sergey Brin when they were both PhD students at Stanford University.
My Parser	(S (NP (-NONE- Google)) (VP (VBD began) (PP (IN in) (NP (NP (NNP January) (CD 1996)) (PP (IN as) (NP (NP (DT a) (NN research) (NN project)) (PP (IN by) (NP (DT Larry) (NN Page) (CC and) (NN Sergey) (NN Brin))))))) (SBAR (WHADVP (WRB when)) (S (NP (PRP they)) (VP (VBD were) (NP (DT both) (NN PhD) (NNS students)) (PP (IN at) (NP (DT Stanford) (NN University))))))) (. .))
Stanford Parser	(ROOT (S (NP (NNP Google)) (VP (VBD began) (PP (IN in) (NP (NNP January) (CD 1996))) (PP (IN as) (NP (DT a) (NN research) (NN project))) (PP (IN by) (NP (NP (NNP Larry) (NNP Page)) (CC and) (NP (NP (NNP Sergey) (NNP Brin)) (SBAR (WHADVP (WRB when)) (S (NP (PRP they)) (VP (VBD were) (DT both) (NP (NP (NNP PhD) (NNS students)) (PP (IN at) (NP (NNP Stanford) (NNP University)))))))))) (. .)))
Berkeley Parser	(ROOT (S (NP (NNP Google)) (VP (VBD began) (PP (IN in) (NP (NNP January) (CD 1996))) (PP (IN as) (NP (NP (DT a) (NN research) (NN project)) (PP (IN by) (NP (NP (NNP Larry) (NNP Page)) (CC and) (NP (NNP Sergey) (NNP Brin)))) (SBAR (WHADVP (WRB when)) (S (NP (PRP they)) (VP (VBD were) (NP (NP (DT both) (NNP PhD) (NNS students)) (PP (IN at) (NP (NNP Stanford) (NNP University)))))))))(. .)))

Table 4: Table shows the comparison of parses from the three parsers for long sentence length > 10

are out of training set and to estimate $P(A \rightarrow word)$ probabilities the parser uses the distribution $P(A)$. Consider the word cricket, if we look at the probabilities then we see that $P(-NONE-) = 0.0307$ and $P(NN) = 0.0622$ but $P(NP \rightarrow -NONE-) = 0.1073$ and $P(NP \rightarrow NN) = 0.0378$ and therefore $P(-NONE-) * P(NP \rightarrow -NONE-) > P(NN) * P(NP \rightarrow NN)$. To correct this we need to decrease the probability mass $P(NP \rightarrow -NONE-)$ and increase that of $P(NP \rightarrow NN)$. For the words Virat and Kohli analysis shows that $P(NP \rightarrow DT, NN)$ is greater than $P(NP \rightarrow NNP, NNP)$. This can simillarly be corrected by decreasing probability mass of $(NP \rightarrow DT, NN)$ and increasing that of $(NP \rightarrow NNP, NNP)$. In sentence 2 all words belong to the vocabulary, but here the error is that the phrase "released next week" is grouped incorrectly under NP which should actually be grouped under VP. Analysis shows that the phrase next week which is tagged as JJ and NN finds it's parent as $(NP| < JJ - NN >)$ because $P(NP \rightarrow JJ, NN) < P(NP| < JJ - NN > \rightarrow JJ, NN)$. The node $(NP| < JJ - NN >)$ is actually a dummy node created due to chomsky normalization it just indicates that JJ and NN occured more frequently with some other siblings in the unchomsky normalized trees. Hence the parser chooses the subtree rooted at $(NP \rightarrow VBN, NP| < JJ - NN >)$ than $(VP \rightarrow VBN, NP)$. To solve this error we can either remove the rule $(NP| < JJ - NN > \rightarrow JJ, NN)$ or decrease its probability mass and add to $(NP \rightarrow JJ, NN)$.

Long sentences: The longer length sentences are a bit cumbersome to analyse so here I will explain only some parts of the tree that could be corrected by modifying rules/probabilities. In the first sentence in table 4, the unknown words are likes, Robin and Uthappa. Out of them the name Robin Uthappa has been tagged incorrectly as (DT, NN) this is again due to the fact that $P(NP \rightarrow DT, NN)$ has more probability mass than $P(NP \rightarrow NNP, NNP)$. The solution to this we have seen above. Another error is PP is incorrectly grouped under ADVP. The exact reason for such errors are difficult to explain in deeper trees but one of the reasons can be that a rule originating from ADVP to PP might be having a higher probability than the rule originating from VP. Correction in such cases would involve decreasing the probability of rules to PP originating from ADJP and increasing the same for those originating from VP. In sentence 2 the unknown words

are Google, Sergey, Brin and Phd. We observe that parser makes similar types of error while tagging words Sergey, Brin and Google as was discussed before for the short sentences. Also the word phd gets mistagged as NN instead of NNP. The subtree rooted at SBAR gets wrongly grouped under VP which should actually get grouped under VP. Analysis shows that $P(VP \rightarrow VBD, SBAR)$ is greater than $P(NP \rightarrow NP, SBAR)$ which might be one of the reasons for the parser grouping SBAR under VP. The solution to this would be to decrease probability of the rule $(VP \rightarrow VBD, SBAR)$ and increase that of $(NP \rightarrow NP, SBAR)$.

From the above experiments we see that for both shorter and longer sentences the major problem is out of vocabulary terminal words. For longer sentences we observe that the parser gets many local subtree structures correct but the overall structure of the tree is not always the same as that of other online parsers. We can build better parsers by making use of the complete Penn Treebank dataset. More data helps in obtaining a better ML estimate. We also conclude that smoothing plays a vital role in obtaining good performance of the parser for lesser training data.

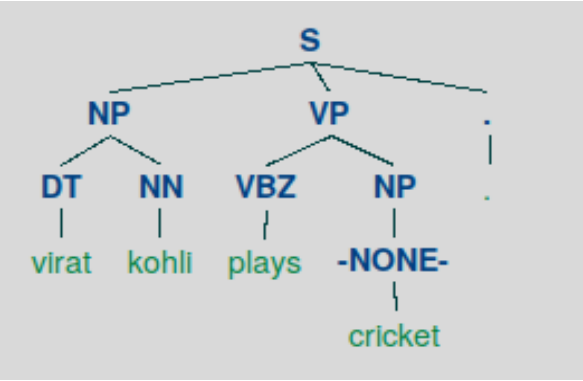

Figure 1: My parser short sentence 1 in table 3

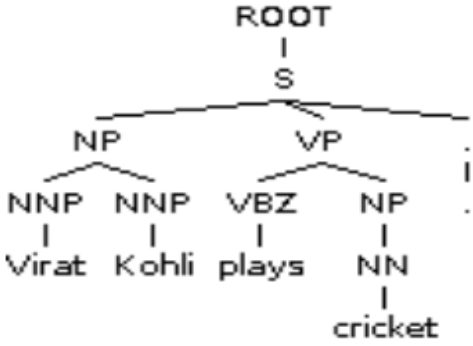

Figure 2: Berkeley parser short sentence 1 table 3

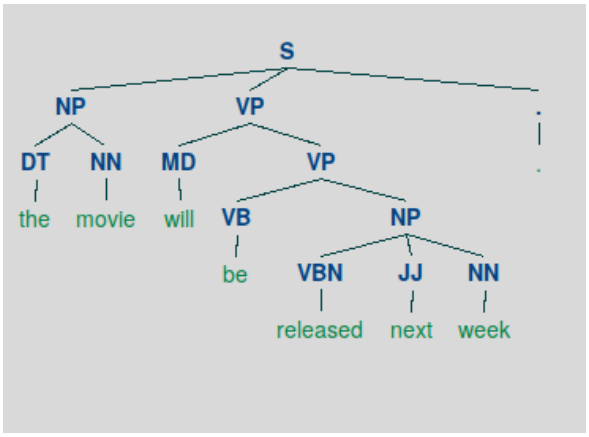

Figure 3: My parser short sentence 2 table 3

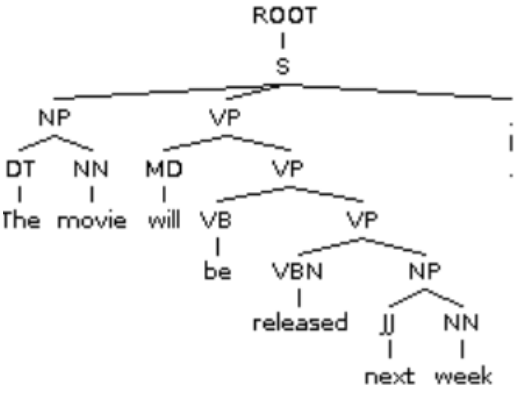

Figure 4: Berkeley parser short sentence 2 table 3

9 Appendix

Following is the path to shell script on the server that takes a user sentence and displays the parse:
/home/e1-246-5/assignment3/cky_run.sh
Run the script as follows:
$ ./cky_run.sh
On execution this will prompt for the input sentence. Provide the sentence to be parsed the output will be displayed on the console as well as written to parsed_tree.tst under logs directory.

References

Cristopher Manning. *LSA 354 lecture slides*.

Satoshi Sekine and Michael J. Collins. 1997. Evalb.

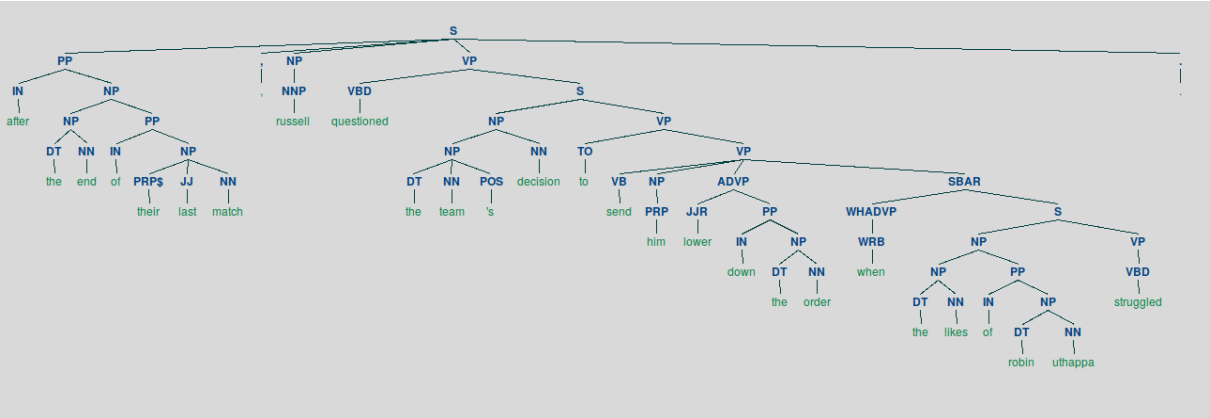

Figure 5: My parser long sentence 1 table 4

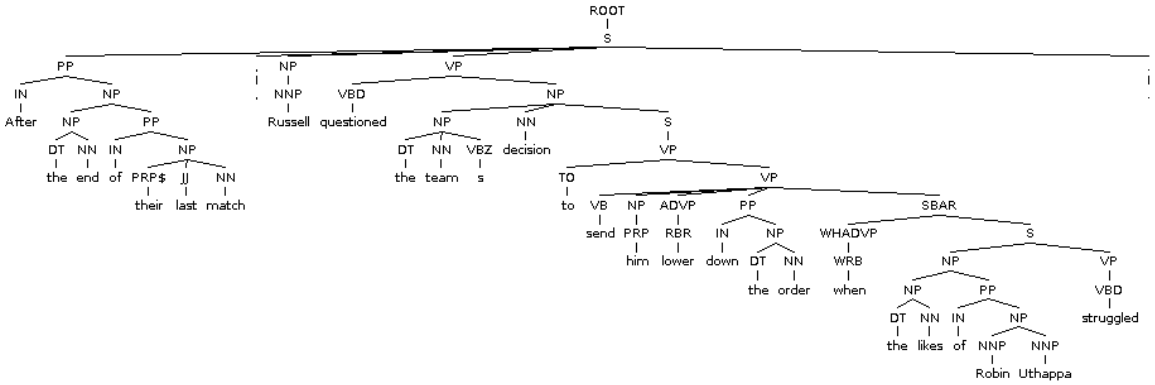

Figure 6: Berkeley parser long sentence 1 table 4

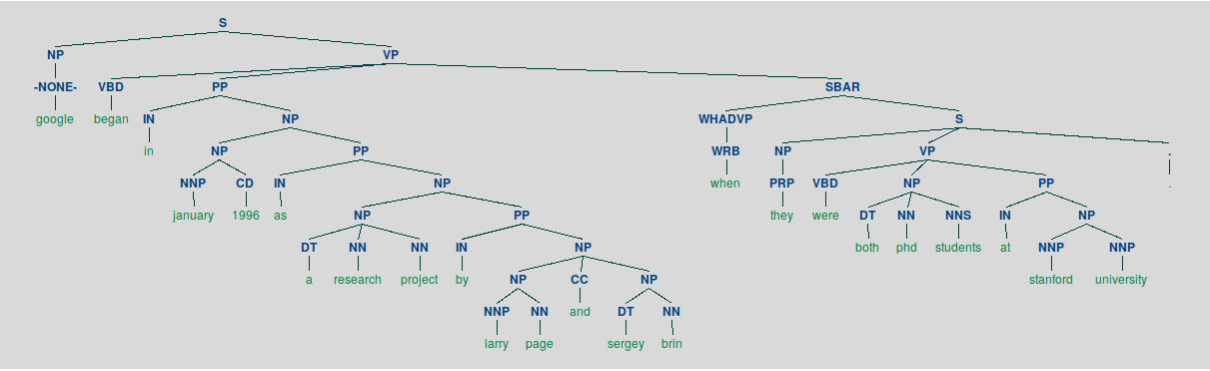

Figure 7: My parser long sentence 2 table 4

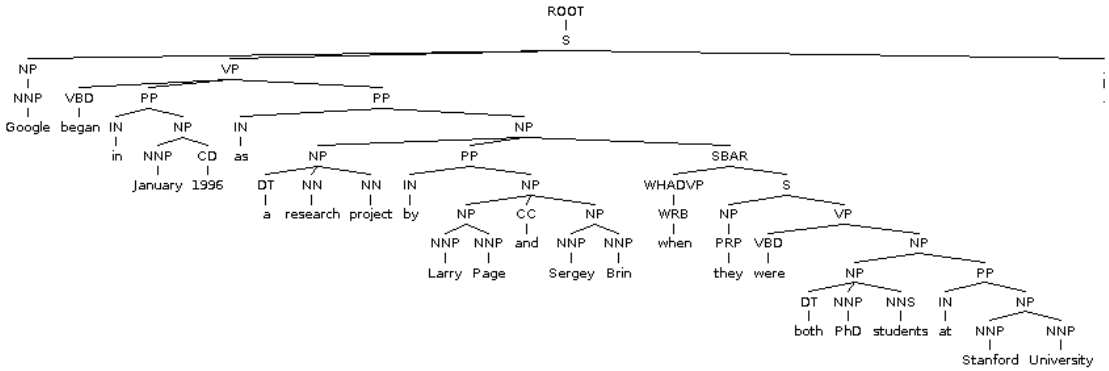

Figure 8: Berkeley parser long sentence 2 table 4