

Parsing Natural Language with Context Free Grammars

CS404/504 Programming Project
Due on October 30 (Thu) by 12:00pm

1 Motivation

Enabling computers to understand human language (speech and text) is an important task in Artificial Intelligence. In the area of automated text processing, people have traditionally focused on problems such as text categorization and clustering, information extraction, and semantic parsing, to name just a few. A significant number of approaches for solving these problems require the text to be split in sentences and annotated with syntactic phrase structures. For example, the sentence “*John plays with the dog*” corresponds to the syntactic tree in Figure 1.

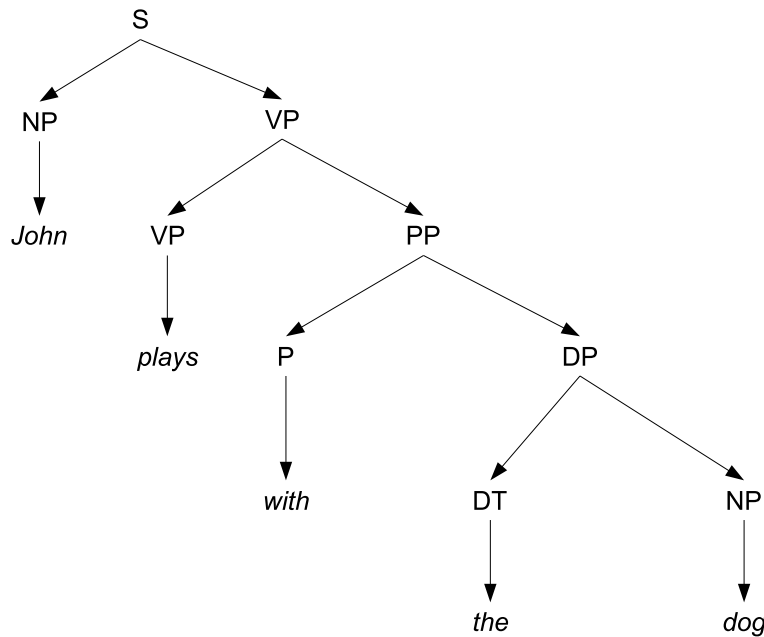


Figure 1: Syntactic structure.

The non-terminal nodes in the tree have the following meaning: 'S' = sentence'; 'NP' = noun phrase / noun; 'DP' = noun phrase with determiner; 'VP' = verb phrase / verb; 'P' = preposition; 'PP' = prepositional phrase; 'DT' = determiner.

Syntactic structures can be seen as an intermediate step in the process of text understanding. Accurate syntactic parses are very useful as they allow computers to perform a syntax-directed semantic analysis of the text. A full grammar of the English language should specify a set of rewriting rules that can be used to generate all syntactically correct English sentences. Designing a grammar for the entire English language is a daunting, difficult task; for the sake of simplicity, in this project we will work with simple grammars that can generate only a subset of English. For example, using the simple grammar from Table 1 below, one can generate the sentence from Figure 1 by repeatedly applying the rewriting rules, starting from the non-terminal symbol S.

$S \rightarrow NP VP$	0.45	$VP \rightarrow plays$	0.10	$P \rightarrow with$	0.50	$P \rightarrow like$	0.50
$S \rightarrow DP VP$	0.55	$VP \rightarrow flies$	0.09	$DT \rightarrow the$	0.50		
$PP \rightarrow P NP$	0.45	$VP \rightarrow time$	0.02	$DT \rightarrow an$	0.25		
$PP \rightarrow P DP$	0.55	$VP \rightarrow dog$	0.01	$DT \rightarrow a$	0.25		
$VP \rightarrow VP NP$	0.25	$NP \rightarrow flies$	0.07	$DP \rightarrow DT NP$	1.00		
$VP \rightarrow VP DP$	0.25	$NP \rightarrow arrow$	0.10	$NP \rightarrow NP PP$	0.50		
$VP \rightarrow VP PP$	0.20	$NP \rightarrow plays$	0.05	$NP \rightarrow John$	0.09		
$VP \rightarrow like$	0.08	$NP \rightarrow dog$	0.10	$NP \rightarrow time$	0.09		

Table 1: A simple English grammar

One possible derivation of “*John plays with the dog*” using this grammar is:

$$\begin{aligned}
 S &\rightarrow NP VP && 0.45 \\
 &\rightarrow John VP && 0.09 \\
 &\rightarrow John VP PP && 0.20 \\
 &\rightarrow John plays PP && 0.10 \\
 &\rightarrow John plays P DP && 0.55 \\
 &\rightarrow John plays with DP && 0.50 \\
 &\rightarrow John plays with DT NP && 1.00 \\
 &\rightarrow John plays with the NP && 0.50 \\
 &\rightarrow John plays with the dog && 0.10
 \end{aligned}$$

Given a context free grammar like the one in Table 1, automatically parsing a natural language sentence means deciding whether the sentence can be generated by the grammar, and, if the answer is yes, building a syntactic tree that describes how the sentence can be generated using the rewriting rules from the grammar.

2 Context Free Grammars (CFGs)

A **Context Free Grammar (CFG)** in Chomsky Normal form is a tuple $G = (N, \Sigma, P, S)$, in which:

- N is a set of nonterminal symbols (e.g. S, NP, VP , etc)
- Σ is a set of terminal symbols (e.g. *dog, the, plays*, etc)
- P is a set of production/rewriting rules of the form:
 1. $A \rightarrow B C$, where A, B , and C are all nonterminal symbols (e.g. $S \rightarrow NP VP$).
 2. $A \rightarrow \alpha$, where A is a nonterminal, and α is a terminal symbol (e.g. $VP \rightarrow plays$).
- S is a designated start symbol.

3 Syntactic Parsing using Dynamic Programming

Given a context free grammar $G = (N, \Sigma, P, S)$ in Chomsky Normal form and a sequence of words $w = w[1..n]$, it is possible to determine whether the sentence w can be generated by G via dynamic programming. In addition, the information that was saved during the execution of the dynamic programming algorithm can be used to generate the syntactic parse tree for the input sentence.

Sketch for a possible solution:

1. Associate a numerical index with each nonterminal in the grammar (e.g. $S = N^1$, $NP = N^2$, $VP = N^3$, etc).
2. For every nonterminal N^i in the grammar, define $\delta_i(p, q)$ as follows:

$$\delta_i(p, q) = \begin{cases} true, & \text{if nonterminal } N^i \text{ can generate } w[p..q] \\ false, & \text{otherwise} \end{cases}$$

Then the sentence $w = w[1..n]$ can be generated by the grammar if and only if $\delta_1(1, n)$ is *true*.

3. If one of the productions in P is $N^i \rightarrow N^j N^k$, then $\delta_i(p, q)$ will be true if there exists $p \leq r < q$ such that $\delta_j(p, r)$ and $\delta_k(r + 1, q)$ are both true.

4 Implementation

Write a program in C/C++/Java/Scheme/Python/Ruby that decides whether an input sentence can be generated by a given grammar. When the answer is yes, the program should also output a parse tree for the sentence. The program will run with 3 command line parameters in this order:

1. <model-file> is the name of the input file that contains the grammar, using the format described in Section 4.1.
2. <raw-file> is the name of the input file containing a sentence representend as a sequence of words separated by white spaces.
3. <prs-file> is the name of the output file, where the answer Yes/No will be stored, together with the parse tree.

Graduate Students:

A **Probabilistic Context Free Grammar (PCFG)** is a CFG in which each rule in P is augmented with a conditional probability. A PCFG is thus a 5-tuple $G = (N, \Sigma, P, S, D)$, where D is a function assigning probabilities to each rule in P .

1. Rules of the type $A \rightarrow BC$ wil be associated probabilities $p(A \rightarrow BC|A)$ i.e. the probability of expanding A into B C, given that we know the left hand side of the rule is A.
2. Rules of the type $A \rightarrow \alpha$ will be associated probabilities $p(A \rightarrow \alpha|A)$.

Assuming all rewriting rules are independent, the probability of a parse tree can be computed as the product of the probabilities of all rules used in the derivation of the parse tree. For example, the probability of the parse tree in Figure 1 will be:

$$P(T) = 0.45 \times 0.09 \times 0.20 \times 0.10 \times 0.55 \times 0.50 \times 1.00 \times 0.50 \times 0.10$$

A PCFG may generate more than one parse tree for any given sentence – take for example the sentence “*Time flies like an arrow*” and try to parse it using the grammar from Table 1. Implement a dynamic programming algorithm that outputs *the most likely parse* for a sentence (i.e the parse with the maximum probability), together with its probability.

4.1 Model file

The program is supposed to read the grammar from a model file. Ignore all empty lines, or comment lines (starting with the symbol '#'). Each rule in the grammar will be described in one separate line. Terminal symbols will be enclosed between quotation signs, as illustrated in Figure 2.

```
# A simple English grammar
S -> NP VP 0.45
S -> DP VP 0.55
...
...
...
NP -> 'John' 0.09
NP -> 'time' 0.09
```

Figure 2: Model file format.

4.2 Parse file

When the sentence can be generated by the grammar, the program should output a parse tree. Use the indented format to represent the parse tree, as illustrated in Figure 3.

5 Programming Style

Place all the files in one directory, and name it '*lastname*_*firstname*'. Use the standard software engineering principles to design your program. Make sure the program is well documented. Include a header comment for each file in C/C++ (class in Java) and each function in C/C++ (method in Java). Header comments should describe the input and output to the procedure, the preconditions and postconditions of the procedure and other pertinent information. For more information, see the guidelines at:

<http://ace.cs.ohiou.edu/~chelberg/classes/361/style-guide.html>

Make sure that you also include a Makefile, so that the command “*make*” correctly compiles and builds your program. Your program should compile and run correctly on the departmental Unix machines. Write also a project report in which you describe in detail your dynamic programming solution (the four steps discussed in class).

```
# Can the input sentence be generated by the grammar?
Yes 1.11375E-05
# A parse tree generated by the simple English grammar.
S
  NP
    John
  VP
    VP
      plays
    PP
      P
        with
      DP
        DT
          the
        NP
          dog
```

Figure 3: Output file format.

6 Submission

Create a gzipped, tar ball archive of your program, and send it by email to bunescu@ohio.edu by 12:00pm on Thursday, Oct 30, with the subject line “*CS404/504 Project*”. Late submissions will not be accepted.

For example, if the name is John Williams, creating the archive can be done using the following commands:

- > tar cvf williams_john.tar williams_john
- > gzip williams_john.tar

These two steps will create the file 'williams_john.tar.gz' that you can send to me by email.