Introduction to English Syntax

- Syntax
  - The study of the rules whereby words or other elements of sentence structure are combined to form grammatical sentences
  - How symbols (words) combine with other symbols.
  - Independent of meaning (semantics) (??)
  - “Colourless Green Ideas Sleep Furiously”

- Simple Sentences
  - Sentences which contain a single main verb (with or without auxiliaries)
  - Not “The man who came yesterday stayed for dinner”

- Sentences whose main verb expresses an action
  - Lions eat deer versus
  - (not) Lions like eating deer

A quick note about reality

- Infinite number of possible sentences
  - Most sentences discussed in NLP modules tend to be rather simple
  - “Time flies like an arrow”
  - “Fruit flies like bananas”
  - “I made her duck.”

- Everyday sentences are often very complex & ambiguous
  - “The president spoke to the nation about the problem of drug use in the schools from one coast to the other.” (720)

- Real world knowledge is often required (rather than syntactic parsing)
  - “Farmer bill debated in parliament”

- Syntactically incorrect sentences are common
  - partial parses might still be useful
Parts of Speech

- We've already discussed tagging words with POS categories
- We can view POS as the basic building blocks of syntax
  - (above morphology)
- We need to justify that such categories are valid.
  - Morphological evidence
  - Distributional evidence

Open Class Words

- Nouns
  - Most take suffix -s for plural
  - "He has no" ______
  - "John" & other Proper Nouns don't meet either test
- Verbs
  - Most take suffix -ing
  - "They can" ______
  - "They can" ______ "them"
  - Verbs have person and number
    - e.g. "am" is first person singular because it only agrees with "I"
    - Tense is also morphologically marked
- Adjectives
  - Most adjectives can have "er" added
  - "They are very" ______
  - In many Indo-European languages adjs can have number, gender and case (but not English).
  - Modify Nouns
- Adverbs
  - Most adverbs can have "ly" added
  - They did it ______
  - Modify Verbs

Closed Class Words

- Prepositions (P)
  - No morphology
  - Any word which isn't a verb and can precede "Them"
  - I am against them
  - I am with them
- Determiners (Det)
  - No morphology
  - What are you looking at? ______ hat.
  - a v. the
  - Some Indo-European languages have number, case, gender
  - English a few change with number
    - these hats v. this hat
  - Pronouns (PRN)
    - Most Pronouns change case
    - he likes her versus she likes him.
    - Pronouns can substitute for a noun without a determiner or adjective.
- Auxiliary (Aux)
  - historically derived from English Verbs
  - Some still show verb like morphology
  - immediately proceed a verb in positive sentences "he can read the book"
  - immediately followed by not to form a negative

Phrases

- Within a sentence words form phrases
  - I saw men
  - I saw the men
  - I saw the angry men
  - I saw the angry men with banners
  - I saw the angry men with black banners
  - I saw the angry men with their black banners
- Noun Phrases
- Prepositional Phrases
- Verb Phrases
  - (I)np ((saw)n (the angry men)n (with their black banners)n)
Ambiguity

- Global ambiguity
  - I saw the angry men with the black banners
    \((I)_{np} (\text{saw})_{np} (\text{the angry men})_{np} (\text{with (their black banners)})_{np}\)

- I saw the angry men with my telescope
  \((I)_{np} (\text{saw})_{np} (\text{the angry men})_{np} (\text{with (a telescope)})_{np}\)

- Local ambiguity
  - John knew Bill was coming
  - Garden Path Sentences

Formal Grammar

- The syntax of a language can be described by a formal grammar consisting of:
  - A set of non-terminal symbols.
    - Uppercase letters are NTs which can be further expanded (NP, VP)
    - Lowercase letters cannot be further expanded but can be replaced with words (det, noun, verb)
  - A Start Symbol (one of the NTs)
  - A set of terminal symbols which represent words
  - A set of productions of the form \(A \rightarrow B C \mid D C\)

- Generation -> starting with \(S\) generate a NL sentence
- Recognition -> recognize if a sentence is valid by using the grammar in reverse

A small corpus of English

- Syntactically correct sentences
  - The black cat killed the white mouse.
  - The young man slept.
  - The young man gave the white cat a piece of fish.
  - The mouse eat some cheese
  - The young man put some cheese on the table

- Syntactically incorrect versus Semantically incorrect
  - The young table slept
  - "The man put"

Recognition

- Recognition is more difficult than generation
- Most grammars will have several choice points
- Grammar defines a search space
- Recognition is Search through possible paths in the grammar
- Backtracking is essential
Lexicon

- Consider the following:
  \[ NP \rightarrow \text{Det} \ SNP \]
  \[ \text{SNP} \rightarrow \text{noun} | \text{ADJ} \ SNP \]
  \[ \text{noun} \rightarrow \text{cat} | \text{man} | \text{table} | \text{kitchen\_sink} \ldots \]

- Clearly we don't want a grammar rule which has every noun in English.
- Instead we could write:
  \[ \text{noun} \rightarrow \{\text{any word listed in the lexicon as a noun}\} \]
  or
  \[ \text{noun} \rightarrow \{\text{noun}\} \]
  \[ \text{cat} : \text{noun} \]
  \[ \text{table} : \text{noun} \ldots \]
  (Compare this to how DCGs are represented in Prolog)

Handling Agreement

- If we add plural nouns and present tense verbs to the lexicon then the grammar does not handle agreement:
  \[ \text{cat} : \text{noun} \]
  \[ \text{cats} : \text{noun} \]
  \[ \text{sleep} : \text{verb} \]
  \[ \text{sleeps} : \text{verb} \]
  \[ \text{the} : \text{det} \]
  \[ \text{that} : \text{det} \]
  \[ S \rightarrow \rightarrow ^\star \text{the cat sleep} \]
  \[ S \rightarrow \rightarrow ^\star \text{that cats sleeps} \]

Variables to handle agreement

- \[ S \rightarrow \text{NP}(N) \ \text{VP}(N) \]
- \[ \text{NP}(N) \rightarrow \text{det}(N) \ \text{SNP}(N) | \text{SNP}(N) \]
- \[ \text{SNP} \rightarrow \text{noun}(N) | \text{adj} \ \text{SNP}(N) \]
- \[ \text{VP}(N) \rightarrow \text{verb}(N) \ \text{VerbComps} \]
- \[ \text{VerbComps} \rightarrow \text{null}| \text{NP}(\_)| \text{PP}| \text{NP}(\_) \ \text{PP} \]
- \[ \text{PP} \rightarrow \text{prep} \ \text{NP}(\_) \]
- \[ \text{det}(N) \rightarrow \{\text{det}, S\} \]
- \[ \text{this}: \text{det}, S \]
- \[ \text{these}: \text{det}, P \]
- \[ \text{that}: \text{det}, S \]
- \[ \text{cat}: \text{noun}, S \]
- \[ \text{cats}: \text{noun}, P \]
- \[ \text{kill}: \text{verb}, P \]
- \[ \text{kills}: \text{verb}, S \]
- \[ \text{killed}: \text{verb}, \_ \]

What makes a good grammar?

- Given a finite set of sentences
- an infinite number of grammars can produce this set
  - does the grammar generate only grammatical sentences?
  - does the grammar accept valid new sentences?

Knowledge Representation Criteria

- Linguistic Naturalness
- Mathematical Power
- Computational Efficiency
Chomsky’s Hierarchy of Grammar

- Chomsky’s Hierarchy
  - (Recursively) Enumerable (Type 0)
  - Context Sensitive Languages (Type 1)
  - (Indexed Languages)
  - Context-Free Languages (Type 2)
  - Finite-state (Regular) languages (Type 3)

- Whether natural languages are context free is debatable.
  - Some parts of Swiss-German are definitely not.
  - As is the language a^n b^n c^n
  - (but can be dealt with a cfg and a stack i.e. an indexed grammar)

- CFGs are computationally tractable (polynomial)
- Highly efficient algorithms for assigning structure to sentences
- (Reality: grammars tend to have thousands of rules/sentences lots of possible parses)

Recognition versus Parsing

- A recogniser simply accepts or rejects strings: the string is either English or it isn’t.
  - ✓ they discussed the dogs with the policemen
  - ✗ policemen with the discussed the dogs they
  - ✗ discussed they the dogs with the policemen

- But no constituent structure!

- How do we interpret the string?
- The same string may have many different structures and interpretations.
  - British Left waffles on Falklands

Parsing

- Parsing is the recovery of structure for a string given a grammar.
- Parsing is a search problem. (So are finite-state operations such as composition.)
- Find the right route to generating parse tree(s) amongst all possible routes.
- Different parsing algorithms have different advantages and disadvantages and, especially, different time complexity.
- We can incorporate linguistic decisions/intuitions within the parser

- Basic approaches
  - Top-Down Parsers start at the ‘top’ of the grammar and predict constituents.
  - Bottom-Up Parsers start with the input words and build constituents.

- You might think that one is more intuitively correct than the other, but in fact both have their drawbacks.

Simple top-down algorithm

- Builds from the root S node to the leaves
- Assuming we build all trees in parallel:
  - Find all trees with root S (or all rules with left-hand side S)
  - Next expand all constituents in these trees/rules
  - Continue until leaves are parts of speech (POS)
  - Candidate trees failing to match POS of input string are rejected
- This describes breadth-first search.
- In depth-first search you keep expanding rules until you reach a terminal, and then when that succeeds (or fails) you backtrack and search other rules.
Simple top-down algorithm

\[
\begin{align*}
S &\rightarrow NP \ VP \\
S &\rightarrow \text{Aux} \ NP \ VP \\
S &\rightarrow VP \\
V &\rightarrow \text{book} | \text{flight} | \text{meal} | \text{money} \\
NP &\rightarrow \text{Det} \ Nom \\
NP &\rightarrow \text{PropN} \\
NP &\rightarrow \text{Nom} \rightarrow \text{H} \ Nom \\
\text{Nom} &\rightarrow V \ NP \\
\text{VP} &\rightarrow V
\end{align*}
\]

Depth-first search:

- Agenda of search states: expand search space incrementally, exploring most recently generated state (tree) each time
- When you reach a state (tree) inconsistent with input, backtrack to most recent unexplored state (tree)
- Which node to expand? Leftmost or rightmost
- Which grammar rule to use? Order in the grammar

Bottom-up parsing:

- Parser begins with words of input and builds up trees, applying grammar rules with right-hand side match:

```
Book that flight
```

Parse continues until an S root node reached or no further node expansion possible
Bottom-up parsing

- Top-Down parsers never explore illegal parses (e.g., can't form an S) — but waste time on trees that can never match the input
- Bottom-Up parsers never explore trees inconsistent with input — but waste time exploring illegal parses (no S root)

Problems with top-down parsing

- Left-recursion is a big problem for top-down parsers. With a rule such as:
  \[ \text{NP} \rightarrow \text{NP PP} \]
  a simple depth-first search will keep expanding the NP forever.
- Ambiguity in natural language means that any sentence might have hundreds or thousands of possible parses.
- With no way to filter out any valid parse, simple bottom-up and top-down parsers simply have to compute all of the parses.
- There is no storage other than the agenda and the cumulated set of successful parses. This means that tree fragments might get rebuilt many times as the parser reexplores analyses for the same string.
- British left waffles on Falkland Islands
  - \{Falkland Islands\} is correct whether left or waffles is the verb

Cocke-Younger-Kasami Algorithm

- Bottom-up Algorithm
- Uses dynamic programming.
- J&M don’t discuss this until later in the book (Chapter 12) in the context of discussion of probabilistic CFG’s.
  - But originally it was developed as a non-probabilistic algorithm
- Similar to Earley Parsing but
  - I find it slightly easier to understand
  - Easy to extend to probabilistic parsing (next week)
What is dynamic programming?

- Answer: a class of algorithms that use tables to store solutions to subproblems of larger problems. Some examples in language processing:
  - minimum edit distance
  - CYK algorithm
  - Earley algorithm
  - Viterbi algorithm
  - forward algorithm
- We’ll return to several of these later on…

Minimum edit distance

- Compute the minimum edit distance between cat and at according to the following criteria (Levenshtein Distance):
  - Substituting one letter with another costs 1 point
  - Deleting a letter costs 1 point
  - Inserting a letter costs 1 point
- Intuitively the right alignment is as follows, and costs 1 (deletion/insertion):

```
  c  a  t
-  a  t
```
- How to compute this efficiently?

An Efficient algorithm
(see J&M p 156 for pseudocode)

- Pad each string with a dummy symbol at the beginning (e.g. '§').
- Create an n x m matrix, where n and m are the lengths of the padded strings.
- Seed the matrix at [0,0] with distance (cost) 0.
- Loop over all columns i, loop over all rows j, assigning the following distance to [i, j]:

```latex
\text{distance}[i, j] \leftarrow \text{MIN} \quad (\text{distance}[i-1, j] + \text{inscost(target)}) \\
\quad \text{distance}[i, j-1] + \text{deletcost(source)} \\
\quad \text{distance}[i-1, j-1] + \text{subcost(source, target)} \\
\quad \text{distance}[i, j-1] + \text{delcost(source)}
```

---

# | a | t
---|---|---
# | - | a | 2 <--- at
c | c | a | 1  <--- c at
a | ca | a | 2 <--- cat
a | ca | 2 <--- cat
```
Minimum edit distance

- Use back pointers to recover the cheapest path
- This algorithm was independently discovered seven times.
- String matching is very important in a lot of fields including computational linguistics and computational biology
- For text processing
  - heavily used in spelling correction
  - email spam filtering etc.

Back to the CKY algorithm

- The CYK parser is
- A bottom up parser
  - We start with the terminals in the input string and subsequently compute recognised parse trees by going from already recognised RHS of productions to non-terminals on the LHS of each grammar rule.
- We build up a chart such that
  - A is added to C if A →→ w_i...w_m
  - A → B C

Chomsky Normal Form

- CYK requires that our grammar is in Chomsky Normal Form
  - All rules must be either
    - A → B C or
    - A → a
  - (where A, B & C are non terminals, a is a terminal)
- Any CFG can be converted to CNF
  - VP → V NP PP (“fried fish on the grill”) needs to be changed
  - VP → V B
  - B → NP PP
- But this might lose some linguistic intuitions
- Increases grammar size

The algorithm in full (see J&M pp 454-455 for more details)

Let the input string consist of n letters, a1 ... an.
Let the grammar contain r terminal and nonterminal symbols R1 ... Rr.
This grammar contains the subset Rs which is the set of start symbols.
Let P[n,n,r] be an array of booleans. Initialize all elements of P to false.
For each i = 1 to n
For each unit production Rj → ai, set P[i,1,j] = true.
For each i = 2 to n -- Length of span
For each j = 1 to n-i+1 -- Start of span
For each k = 1 to i-1 -- Partition of span
  For each production RA → RB RC
    If P[j,k,B] and P[j+k,i-k,C] then set P[j,i,A] = true
If any of P[1,n,x] is true (x is iterated over the set s, where s are all the indices for Rs)
  Then string is member of language
Else string is not member of language
Parsing simulation

No additional structure added for spans of 5 words!

Notes

- I have only detailed the algorithm as a recogniser
- For parsing, rather than adding just constituents we need to record full grammar rules and then extract the parse from the chart after completion
- CKY parsing is very efficient $O(n^3)
- CKY parsing is also easily extended to probabilistic parsing
  - (which we'll cover next week)
  - Often conversion to & from CNF is added as a preprocessing & postprocessing step.

Summary

- Grammar
- Context Free Grammars for English
  - Agreement
  - Lexical Information
- Parsing
  - basic issues
  - CKY parsing algorithm
- Next week we’ll look at probabilistic parsing and psycholinguistics
- Reading
- Chapters 9 & 10 of Jurafsky and Martin