1. Rewriting systems

The grammar types in the Chomsky hierarchy can all expressed as constraints on a formalisation of the idea of rewriting systems.

A rewriting system consists of

- **Terminals**
  - or *terminal symbols*: words (for now)

- **Non-terminals**
  - or *non-terminal symbols*: Names for constituents in a language

- **Rules**
  - or *productions*, each of which is a pair of
    - a **left-hand side**: a non-empty sequence of any number of terminals and non-terminals
    - a **right-hand side**: a (possibly empty) sequence of any number of terminals and non-terminals

- **Distinguished symbol**
  - One of the non-terminals
    - The starting point for all analyses
    - Usually $S$

2. Rewriting, cont'd

- **Regular** and **Context-free** grammars (CFGs) restrict the left-hand side of rules to a single non-terminal symbol.

- **Regular** grammars further restrict the right-hand side to be a pair of a non-terminal and a terminal, or a single terminal:
  - All the pairs have to be in the same direction:
    - **right-linear**
      - Every pair is $t,NT$
    - **left-linear**
      - Every pair is $NT,t$
For example, the following (right-linear) grammar defines the language $a^*b^+$:

\[
\begin{align*}
S &\to aS \\
S &\to B \\
B &\to bB \\
B &\to b
\end{align*}
\]

And the following context-free grammar defines $a^n b^n$:

\[
\begin{align*}
S &\to aSb \\
S &\to \\
S &\to
\end{align*}
\]

Note the common convention that non-terminals start with upper-case letters, terminal with lower-case.

### 3. Defining a language with a grammar

What is the language defined by a mechanism?

For FSAs, a string is in the language defined by a finite-state automaton $F$ iff there is a non-empty sequence $S$ of states of $F$ such that:

- The first state in $S$ is a start state
- The last state in $S$ is a terminal state
- Either
  - $S$ is of length one and the string is empty
    - That is, $S$ consists entirely of a state which is both a start and an end state
  - $S$ is of length two or more and there is at least one transition between every adjacent pair of states in $S$ with a label such that the concatenation of all those labels is the string.

### 4. Define a language with a grammar

There are two (main) ways to interpret a grammar as defining a language:

- The details vary slightly depending on the type
- What follows is for CFGs

A string is in the language defined by a context-free grammar $G$ iff

**Rewriting**

You can get to the string by:

- writing down $G$'s distinguished symbol
- writing down a new line by choosing a non-terminal from the line above, choosing a rule from $G$ with that symbol as its left-hand side, then re-writing the line above with the chosen symbol replaced by the right-hand side of the chosen rule
- repeating this until there are no non-terminals left
- The resulting tableau is called a **derivation**
5. CFG interpretation, cont'd

Node admissability
A string is in a language defined by a grammar G iff there is at least one labelled tree such that
- The leaf nodes of the tree, in order, correspond to the string
- The root of the tree is labelled with G's distinguished symbol
- For every non-leaf node, there is a rule in G
  - whose left-hand side is the label of the node
  - whose right-hand side corresponds to the labels of the node's children, in order

6. Trees as proofs

Under either interpretation, we can use a parse tree (strictly speaking an ordered tree) to illustrate the way in which a string belongs to the language defined by a CFG.

For example, using the grammar and lexicon for the ATIS domain given in section 12.2 of Jurafsky & Martin (2nd edition)

- For the sentence
  - I prefer a morning flight

The parse tree that 'proves' that this is in the language is

```
       S
      / | \
     NP VP
    / |   | \
   Pro Verb NP
  / |  |     | \
 I prefer Det Nom
   |       |     |
  prefer Nom Noun
       |       |     |
      a Nom Noun
             |     |
            Noun flight
                  |
                    morning
```
7. Generativity

As with FSAs and FSTs (see Lectures 2,3), you can view these rules as either analysis or synthesis machines

- Generate strings in the language
- Reject strings not in the language (recognition)
- Show how strings are in the language (parsing)

8. Reasons to love trees 2: Scaffolding for semantics

Compilers and interpreters for programming languages use parse trees to implement a compositional semantics

Consider a simple grammar for Python arithmetic expressions:

```
Expr → Expr Op1 Term | Term
Term → Term Op2 Simp | Simp
Simp → Var | Number | '(' Expr ')' | Op1 = '+' | '-'
Op2 = '*' | '/'
```

Assuming some more work to tokenise the input, this will give us an analysis for the string "x/y+1"

![Diagram of parse tree]

We can compute a value for this expression using the tree

- If we associate the appropriate computation with each production
  - (skipping a lot of details here, some of which will be covered in a few weeks)
  - And assuming that x is 4 and y is 1
We call that kind of semantic computation a **compositional** one.

"The meaning of the whole [constituent] is a function of the meaning of the parts [children]"

Logic-based approaches to natural language semantics often adopt this approach as well.

**9. Reasons to love trees 3: tackling the sparse data problem**

In a number of different areas

- including question-answering and text-to-speech synthesis

attempts are being made to generalise

- *from* languages for which we *do* have lots of data
- *to* languages where we *don't*

By abstracting linguistic structures (trees and/or dependency graphs) to a sufficient extent

We won't get to this in this course.

**10. Parsing**

If trees are useful, how do we get them?

Parsing is the process of taking a string and a grammar and returning one or more parse trees for that string

Analogous to running a finite-state transducer over a tape

- But since CFGs are more powerful
  - That is, there are languages we can capture with CFGs that we can't capture with finite-state methods
  - The parsing process is likewise more complicated
    - As we'll see in a few days
11. Exploring syntax

By grammar, or syntax, we have in mind the kind of implicit knowledge of your native language that you had mastered by the time you were 3 years old without explicit instruction. Not the kind of stuff you were later taught in “grammar” school:

- At least not in English-speaking countries :-)  
- Indeed some EFL teaching involves something much closer to what we have in mind here

12. Syntax (or Grammar)

Refers to the way words can be arranged in a given language. Grammars (and parsing) are key components in many applications:

- Grammar checkers  
- Dialogue management  
- Question answering  
- Information extraction  
- Machine translation

13. Syntax, cont'd

There's a useful (traditional) contrast between two perspectives on this topic:

**paradigmatic**
- What's interchangeable with what?
  - words, phrases, . . .

**syntagmatic**
- What co-occurs with what?
  - ordering (before/after)
  - marking (prefixes/suffixes)

Key notions that we'll cover:

- Categories (paradigmatic)
- Constituency (syntagmatic)
- Heads (syntagmatic)

Key formalism:

- Context-free grammars

14. Constituency

Groups of words can be shown to act as single units, called *constituents*. In a given language, these units form coherent classes that behave in similar ways, w.r.t.

**External behavior**
- How they relate to other units in the language
  - We can say that in English, noun phrases can come before verbs
**Internal structure**

We can describe an internal structure for the class

- This might involve disjunctions of somewhat unlike sub-classes to do this
- For example, noun phrases can consist of a pronoun, a proper noun, or a complex phrase including a common noun

**15. Constituency, cont'd: Noun Phrases**

We can observe some commonality over the behaviour of the following English phrases:

- they
- Cambodia
- my aunt's pen
- the reason I can't stay taking another look at *Moby Dick*
- three french hens

One piece of evidence is that they can all precede verbs

- That is, occur in a frame such as

  \[
  \underline{\text{-------------}} \text{ surprised him.}
  \]

- This is external, paradigmatic evidence

Internal, syntagmatic, evidence would be, for example, to observe that combining

- **determiners** such as "my" or "three"
- **qualifiers** such as "aunt's" or "french"
- **common nouns** such as "pen" or "hens"

usually results in a coherent phrase, which can fit in the above frame

**16. Grammars and Constituency**

There's nothing easy or obvious about how we come up with

- the 'right' set of constituents
- the rules that govern how they combine

That's one of reasons there are so many different theories of grammar and competing analyses of the same data

The approach we'll explore isn't exactly "cutting-edge"

- But it's a good compromise between simplicity and adequacy
- And the technology required to support it is a good introduction to what's needed for most other approaches