PLIN0068 Constructed Languages

2024–25

Lecture 7: Formal syntax

Syntax

How to put words together to form complex phrases

Obviously not every sequence of words should be **grammatical**, e.g.

- *Colourless green ideas sleep furiously*
- **Sleep green furiously ideas colourless*

A language can be finite or infinite, but your language should be infinite

An infinite language has a finite set of grammatical rules that decide which strings of words are grammatical and which ones are not

Languages and sets

A language L can be understood to be a set of strings of words

- Let's call the entire set of words in L W_L
- We assume W_L to be finite

We denote the set of all possible strings of words in W_L by W_L^* (**Kleene Closure** of W_L)

- ϵ is an empty string
- $W_L^+ = W_L^* - \{\epsilon\}$
- $L \subseteq W_L^*$

If a string s of words is a member of L ($s \in L$) we say s is **grammatical** in L; otherwise ($s \notin L$) s is **ungrammatical** in L

Syntax as a function

The syntax of L should tell us for any $s \in W_L^*$ whether $s \in L$ or $s \notin L$

The syntax of L can be seen as a single function f_L: for any string $s \in W_L^*$:

$$f_L(s) = \begin{cases} 1 & \text{if } s \in L \\ 0 & \text{if } s \notin L \end{cases}$$

NB: This assumes that grammaticality is binary

- We stick to this assumption, as per most syntactic theories
- Alternatively one could understand grammaticality as a more gradient notion, in which case f_L will return values in $[0, 1]$, for example
- Cf. grammaticality vs. acceptability

Decision problems and computability

The problem of determining whether $s \in L$ or $s \notin L$, for an arbitrary $s \in W*$ is a **decision problem**

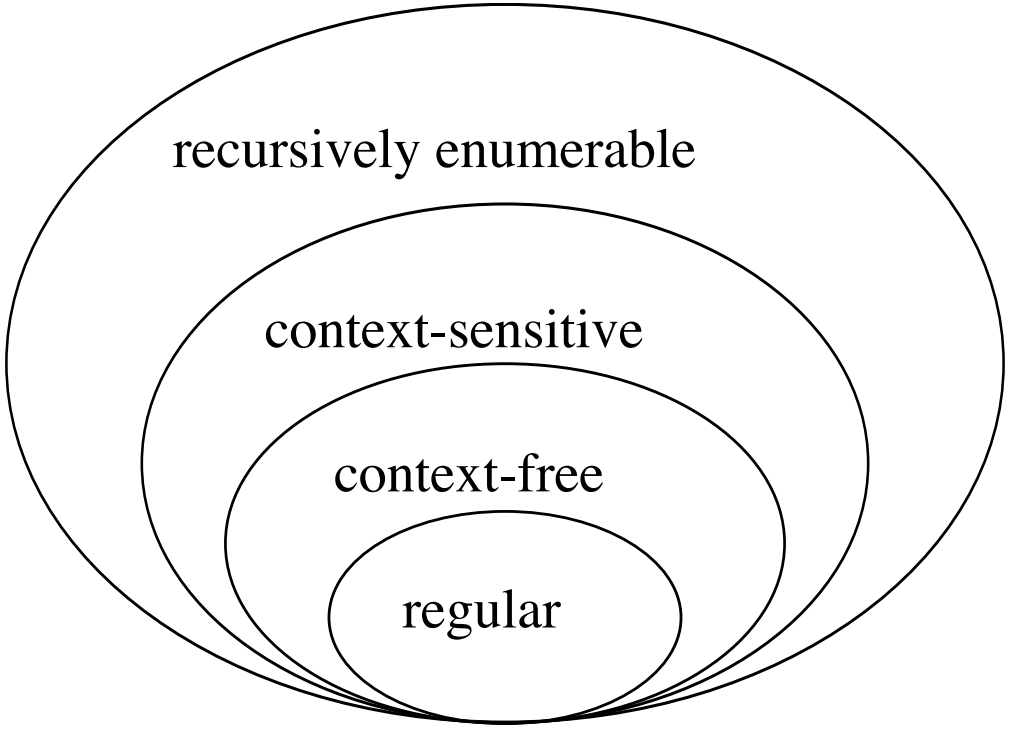
Some decision problems are **undecidable**
= there is no general way of answering them
= f_L is not **computable** (connection to Computability Theory)

The fact that native speakers can (quickly) determine (un)grammatataticality suggests that the syntax function f_L of any natural language L is computable

Chomsky hierarchy

Some computable functions are 'simpler' than others

Correspondingly there are different levels of syntaxes/languages



By J. Finkelstein - CC BY-SA 3.0, Link

Regular Languages

Example 1: A finite regular language

- $W = \{\texttt{a}, \texttt{b}, \texttt{c}\}$
- $W^* = \{\epsilon, \texttt{a}, \texttt{b}, \texttt{c}, \texttt{aa}, \texttt{ab}, \texttt{ac}, \texttt{ba}, \texttt{bb}, \ldots, \texttt{cacba}, \ldots\}$
- $L_1 = \{\texttt{a}, \texttt{abc}, \texttt{abca}\}$

The syntactic rules of L_1:

- A grammatical sentence must contain at least one **a** and at most two **a**'s.
- **b** must be preceded by **a** and followed by **c**.
- **a** can be preceded by **c** but not by **a** or **b**.

Production rules

The syntax of regular languages is described by a finite set of **production rules**

- T: set of terminal symbols = W
- N: set of non-terminal symbols

A production rule of a regular language has to look like:

- $\mathtt{A} \rightarrow \mathtt{xB}$; or
- $\mathtt{C} \rightarrow \mathtt{x}$; or
- $\mathtt{D} \rightarrow \epsilon$

where $\mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{D} \in N$ and $\mathtt{x} \in T$

One member (usually $\mathtt{S}$) of N is a **start symbol**

Production rules for L_1

- $T = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$
- $N = \{\underline{\mathtt{S}}, \mathtt{K}, \mathtt{D}, \mathtt{W}\}$

Production rules

- $\mathtt{S} \to \mathtt{a}$
- $\mathtt{S} \to \mathtt{aK}$
- $\mathtt{K} \to \mathtt{bD}$
- $\mathtt{D} \to \mathtt{c}$
- $\mathtt{D} \to \mathtt{cW}$
- $\mathtt{W} \to \mathtt{a}$

L_1 is the set of all strings $s \in T^*$ that can be generated from the start symbol $\mathtt{S}$ by applying the production rules

- $\mathtt{S} \Rightarrow \mathtt{a}$
- $\mathtt{S} \Rightarrow \mathtt{aK} \Rightarrow \mathtt{abD} \Rightarrow \mathtt{abc}$
- $\mathtt{S} \Rightarrow \mathtt{aK} \Rightarrow \mathtt{abD} \Rightarrow \mathtt{abcW} \Rightarrow \mathtt{abca}$

Example 2: An infinite regular language

- $W = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$
- $W^* = \{\epsilon, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{aa}, \mathtt{ab}, \mathtt{ac}, \mathtt{ba}, \mathtt{bb}, \ldots, \mathtt{cacba}, \ldots\}$
- $L_2 = \{\mathtt{a}, \mathtt{abc}, \mathtt{abca}, \mathtt{abcabc}, \mathtt{abcabca}, \mathtt{abcabcabc}, \ldots\}$

Production rules

- $\mathtt{S} \to \mathtt{a}$
- $\mathtt{S} \to \mathtt{aK}$
- $\mathtt{K} \to \mathtt{bD}$
- $\mathtt{D} \to \mathtt{c}$
- $\mathtt{D} \to \mathtt{cS}$

This grammar involves recursion: we start from S and can come back to it

$$\ldots \mathtt{S} \Rightarrow \ldots \mathtt{aK} \Rightarrow \cdots \Rightarrow \ldots \mathtt{S}$$

Recusion yields infinite languages

Remarks

- Every finite language is a regular language (proof omitted)
- What we defined above is **right regular** languages

Right regular

- $\mathtt{A} \to \mathtt{xB}$; or
- $\mathtt{C} \to \mathtt{x}$; or
- $\mathtt{D} \to \epsilon$

Left regular

- $\mathtt{A} \to \mathtt{Bx}$; or
- $\mathtt{C} \to \mathtt{x}$; or
- $\mathtt{D} \to \epsilon$

- For any right regular language, there is an equivalent left regular language, and vice versa (proof omitted)

Remarks (cont.)

- Every regular language is recognised by a finite state automaton, and every language recognised by a finite state automaton is a regular language (proof omitted)
- A finite state automaton is a kind of machine that does not have (long-term) memory
- Regular languages can encode certain types of dependencies
  - In both L_1 and L_2, $\mathtt{b}$ must be immediately followed by $\mathtt{c}$
  - Such dependencies can be long-distance, e.g., if there is a $\mathtt{b}$, there must be a $\mathtt{c}$ following it at some point

Non-regular languages

- However, regular languages are not capable of unlimited counting, so there is no way to formulate dependencies like the following (proof omitted)
  - There can be any number of `b`'s, but there must be the same number of `c`'s following all the `b`'s
  - Similarly for non-terminal levels, e.g., if there are n-many `NP`'s, there must be n-many `VP`'s
- Natural languages show such dependencies, e.g., centre embedding
- It's been claimed that birdsongs are regular languages

Context Free Languages

Context Free Languages

Context free languages are generated by productions rules of the form:

$$\mathtt{A} \rightarrow \alpha$$

where α is any sequence of terminal and/or non-terminal symbols

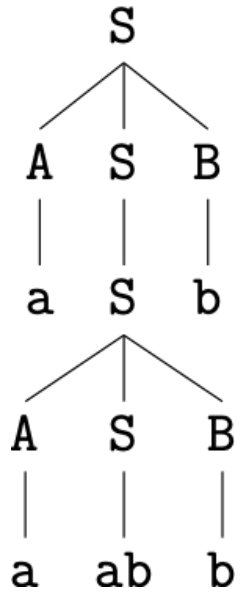
- Every regular language is a context free language
- There are context free languages that are not regular, e.g. $\{\mathtt{a}^n\mathtt{b}^n | n \geq 1\}$
  - $\mathtt{S} \rightarrow \mathtt{ASB}$
  - $\mathtt{S} \rightarrow \mathtt{ab}$
  - $\mathtt{A} \rightarrow \mathtt{a}$
  - $\mathtt{B} \rightarrow \mathtt{b}$

Example derivations in CFL

$S \Rightarrow ab$

$S \Rightarrow ASB \Rightarrow aSB \Rightarrow aASBB \Rightarrow aaSBB \Rightarrow aaabBB \Rightarrow aaabbB \Rightarrow aaabbb$

$S \Rightarrow ASB \Rightarrow AASBB \Rightarrow AAabBB \Rightarrow AAabBb \Rightarrow aAabBb \Rightarrow aAabbb \Rightarrow aaabbb$



Remarks

- Context free languages can have unlimited center embedding dependencies, so more powerful than regular languages
- Every context free language is recognised by a push-down automaton, and every language recognised by a push-down automaton is a context free language (proof omitted)
- A push-down automaton is a machine with a stack memory

Non-context free languages

- There are languages that are not context free, e.g., $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n | n \geq 1\}$ (proof omitted)
- Syntacticians argue that some natural languages have unbounded numbers of cross-serial dependencies, so are not context free
- Chomsky's Transformational Grammar:
  - Prase-Structural Rules are context free and generate D-structures
  - D-structures are mapped to S-structures by Transformations
  - Transformations need to be constrained so that natural languages are not entirely context-sensitive or recursively enumerable

More powerful languages

- The next class of languages in the Chomsky hierarchy is **context sensitive languages**, whose production rules are either of the following:
  - $\mathbf{S} \rightarrow \epsilon$
  - $\alpha \mathbf{A} \beta \rightarrow \alpha \gamma \beta$
    where α, β, γ are any strings of terminal or non-terminal symbols and $\gamma \neq \epsilon$
- Natural language syntax does not have the full power of context-sensitive languages
  - E.g., no rules like "the same number of determiners, nouns, and verbs in the same sentence but in any order"

More powerful languages

- Natural language is mostly context free, but it goes a bit outside of it with certain types of cross-serial dependencies
- Mildly context free languages
    - Tree Adjoining Grammar
    - Multiple Context Free Grammar
    - Minimalist Grammar

For your language

- No need to make syntactic rules for the entire language
- Focus on some constructions, e.g.
    - Simple intransitive and transitive sentences name + predicate (+ name)
    - Structures of DPs
- If you like, you can write some context-free rules for some (or all) of the constructions
- Make sure to have recursion so that your language will be infinite

Example: Context free rules for DPs in English

- $\texttt{DP} \rightarrow \texttt{D NP}$
- $\texttt{DP} \rightarrow \texttt{DP's NP}$
- $\texttt{D} \rightarrow \texttt{the}|\texttt{every}|\texttt{a}$
- $\texttt{NP} \rightarrow \texttt{A NP}$
- $\texttt{A} \rightarrow \texttt{smart}|\texttt{expensive}|\texttt{horrible}$
- $\texttt{NP} \rightarrow \texttt{scientist}|\texttt{equipment}|\texttt{mother}$

DP recursion

- [$_{DP}$[$_{DP}$ the scientist]'s equipment]
- [$_{DP}$[$_{DP}$[$_{DP}$the scientist]'s mother]'s equipment]
- [$_{DP}$[$_{DP}$[$_{DP}$[$_{DP}$the scientist]'s mother]'s mother]'s equipment]

Note that there will be unacceptable cases, e.g. "a horrible equipment's mother's expensive scientist"

Example: Mirror English

- $\texttt{DP} \rightarrow \texttt{NP D}$
- $\texttt{DP} \rightarrow \texttt{NP DP's}$
- $\texttt{D} \rightarrow \texttt{the}|\texttt{every}|\texttt{a}$
- $\texttt{NP} \rightarrow \texttt{NP A}$
- $\texttt{A} \rightarrow \texttt{smart}|\texttt{expensive}|\texttt{horrible}$
- $\texttt{NP} \rightarrow \texttt{scientist}|\texttt{equipment}|\texttt{mother}$

- Equipment expensive scientist every's
- Scientist smart every
- Mother mother scientist the's's