

## Practical matters

The second part of the course will be somewhat different:

- The focus will shift more towards Computational Linguistics topics / applications
- We will review more specialized data structures and algorithms (e.g., automata, parsing)
- Some overlap with parsing class (but with more emphasis on practical sides)
- Less focus on programming

A quick poll: opinions about switching to Python.

## An overview of the upcoming topics

- Background on formal languages and automata (today)
- Finite state automata and regular languages
- Finite state transducers (FST)
  - FSTs and computational morphology
- Dependency grammars and dependency parsing
- Context-free grammars and constituency parsing

## Assignments

- Assignment policy is similar to the first part of the course
- Two graded assignments:
  - Finite state methods (due early Jan)
  - Parsing (due mid Feb)
- There will be more ungraded assignments – they are part of the course work, they are not ‘optional’

## This lecture

### An overview

- Background: some definitions on phrase structure grammars and rewrite rules
- Chomsky hierarchy of (formal) language classes
- Background: computational complexity
- Automata, their relation to formal languages
- Formal languages and automata in natural language processing
- A brief note on learnability of natural languages

## Why study formal languages

- Formal languages are an important area of the theory of computation
- They originate from linguistics, and they have been used in formal/computational linguistics

## Definitions

### Alphabet

- An *alphabet* is a set of symbols
- We generally denote an alphabet using the symbol  $\Sigma$
- In our examples, we will use lowercase ASCII letters for the individual symbols, e.g.,  $\Sigma = \{a, b, c\}$
- Alphabet does not match the every-day use:
  - In some cases one may want to use a binary alphabet,  $\Sigma = \{0, 1\}$
  - If we want to define a grammar for arithmetic operations, we may want to have  $\Sigma = \{0, 1, 2, 3, \dots, 9, +, -, \times, /\}$
  - If we are interested in natural language syntax our alphabet is the set of natural language words,  $\Sigma = \{\text{the, on, cat, dog, mat, sat, } \dots\}$

## Definitions

### Strings

- A *string* over an alphabet is a finite sequence symbols from the alphabet
  - $a, ab, acbcaa$  are example strings over  $\Sigma = \{a, b, c\}$
- The *empty string* is denoted by  $\epsilon$
- The  $\Sigma^*$  denotes all strings that can be formed using alphabet  $\Sigma$ , including the empty string  $\epsilon$
- The  $\Sigma^+$  is a shorthand for  $\Sigma^* - \epsilon$
- Similarly  $a^*$  means the symbol  $a$  repeated zero or more times,  $a^+$  means  $a$  repeated one or more times
- We use  $a^n$  for exactly  $n$  repetitions of  $a$
- The length of a string  $u$  is denoted by  $|u|$ , e.g.,  $|abc| = 3$ , or if  $u = aabbcc$ ,  $|u| = 6$
- Concatenation of two string  $u$  and  $v$  is denoted by  $uv$ , e.g., for  $u = ab$  and  $v = ca$ ,  $uv = abca$

## Definitions

### Language

- A (formal) language is a set of string over an alphabet
  - The set of strings of length 2 over  $\{0, 1\}$ :  $\{00, 01, 10, 11\}$
  - The set of strings with even number of 1's over  $\{0, 1\}$ :  $\{\epsilon, 101, 0, 11, 111110, \dots\}$
  - The set of string that retain alphabetical ordering over  $\{a, b, c\}$ :  $\{a, ab, abc, ac, abcc, \dots\}$
  - The set of strings of words that form grammatically correct English sentences
- Strings that are member of a language is called *sentences* (or sometimes *words*) of the language

## Definitions

### Grammar

- A *grammar* is a finite description of a language
- A common way of specifying a grammar is based on a set of *rewrite rules* (or *phrase structure rules*)
- We represent *non-terminal symbols* with uppercase letters
- We represent *terminal symbols* with lowercase letters
- S is the *start symbol*
- If a string can be generated from S using the rewrite rules, the string is a valid sentence in the language

$$\begin{array}{lcl} S & \rightarrow & A B \\ S & \rightarrow & S A B \\ A & \rightarrow & a \\ B & \rightarrow & b \end{array}$$

Q: What does this grammar define?

## Definitions

### Phrase structure grammars: more formally

A phrase structure grammar is a tuple  $G = (\Sigma, N, S, R)$  where

$\Sigma$  is an alphabet of terminal symbols

$N$  are a set of non-terminal symbols

$S$  is a special 'start' symbol  $\in N$

$R$  is a set of rules of the form

$$\alpha \rightarrow \beta$$

where  $\alpha$  and  $\beta$  are strings from  $\Sigma \cup N$

A string  $u$  is in the language defined by  $G$ , if it can be derived from  $S$ .

## Definitions

### Grammars and derivations

#### Grammar

$$\begin{array}{lcl} S & \rightarrow & A B \\ S & \rightarrow & S A B \\ A & \rightarrow & a \\ B & \rightarrow & b \end{array}$$

#### Derivation of abab

$$\begin{array}{ll} S \Rightarrow SAB & aBAB \Rightarrow abAB \\ SAB \Rightarrow ABAB & abAB \Rightarrow abaB \\ ABAB \Rightarrow aBAB & abaB \Rightarrow abab \end{array}$$

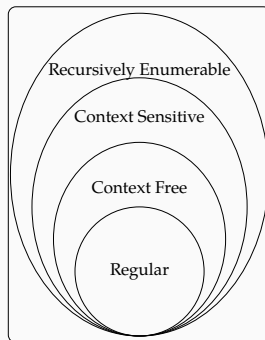
- Intermediate strings of terminals and non-terminals are called *sentential forms*
- $S \xRightarrow{*} abab$ : the string is in the language

Q: What if string was not in the language?

Q: Is there another derivation sequence?

## Chomsky hierarchy of (formal) languages

- Defined for formalizing natural language syntax
- Definitions are in terms of the restrictions on production rules of the grammar
- Also part of theory of computation
- Each language class corresponds to a class of (abstract) machines
- Other well-studied classes exist



## Regular grammars

#### Left regular

- $A \rightarrow a$
- $A \rightarrow Ba$
- $A \rightarrow \epsilon$

#### Right regular

- $A \rightarrow a$
- $A \rightarrow aB$
- $A \rightarrow \epsilon$

- Least expressive, but easy to process
- Used in many NLP applications
- Defines the set of languages expressed by *regular expressions*
- Regular grammars define only regular languages (but reverse is not true)
- We will discuss it in more detail soon

## Regular grammars

### an example

Write a right- and a left-regular grammar  $ab^*c$

#### left

$$\begin{array}{l} S \rightarrow Ac \\ A \rightarrow Ab \\ A \rightarrow a \end{array}$$

#### right

$$\begin{array}{l} S \rightarrow aA \\ A \rightarrow bA \\ A \rightarrow c \end{array}$$

Can you define a regular grammar for

- $a^n b^n$ ?
- $a^5 b^5$ ?

Derive the string  $abbbc$  using one of your grammars

#### left

$$S \Rightarrow Ac \Rightarrow Abc \Rightarrow Abbc \Rightarrow Abbbc \Rightarrow abbbc$$

#### right

$$S \Rightarrow aA \Rightarrow abA \Rightarrow abbA \Rightarrow abbbA \Rightarrow abbbc$$

These grammars are *weakly equivalent*: they generate the same language, but derivations differ

## Context-free grammars (CFG)

#### CFG rules

$$A \rightarrow \alpha$$

where  $A$  is a *single* non-terminal  $\alpha$  is a possibly empty sequence of terminals and non-terminals

- More expressive than regular languages
- Syntax of programming languages are based on CFGs
- Many applications for natural languages too (more on this later)

## Context-free grammars

an example

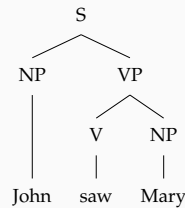
The example grammar:

Example CFG			
S	→	NP VP	VP → V NP
NP	→	John   Mary	V → saw

Exercise: derive 'John saw Mary'

Derivation

$S \Rightarrow NP VP \Rightarrow \text{John VP}$   
 $\Rightarrow \text{John V NP} \Rightarrow \text{John saw NP}$   
 $\Rightarrow \text{John saw Mary}$   
 or,  $S \stackrel{*}{\Rightarrow} \text{John saw Mary}$



## Context-free languages

more exercises / questions

- Define a (non-regular) CFG for language  $ab^*c$
- Can you define a CFG for  $a^n b^n$ ?
- Can you define a CFG for  $a^n b^n c^n$ ?
- Can you define a CFG for  $a^n b^m c^n d^m$ ?

## Context-sensitive grammars

### Context-sensitive rules

$$\alpha A \beta \rightarrow \alpha \gamma \beta$$

where  $A$  is a non-terminal symbol,  $\alpha$  and  $\beta$  are possibly empty strings of terminals and non-terminals, and  $\gamma$  is a non-empty string of terminal and non-terminal symbols.

- There is also an alternative definition through non-contracting grammars
- A rule of the form  $S \rightarrow \epsilon$  is allowed

## Context-sensitive grammars

an example

- Can you define a context-sensitive grammar for  $a^n b^n c^n$ ?
- Can you define a context-sensitive grammar for  $a^n b^m c^n d^m$ ?

## Unrestricted grammars

- The most expressive class of languages in the Chomsky hierarchy is *recursively enumerable* (RE) languages
- RE languages are those for which there is an algorithm to enumerate all sentences
- RE languages are generated by *unrestricted grammars*
- Unrestricted grammars do not limit the rewrite rules in any way (except LHS cannot be empty)
- Mostly theoretical interest, not much practical use

## A(nother) review of computational complexity

Big-O notation

*Big-O notation* is used for describing *worst-case order of complexity* of algorithms

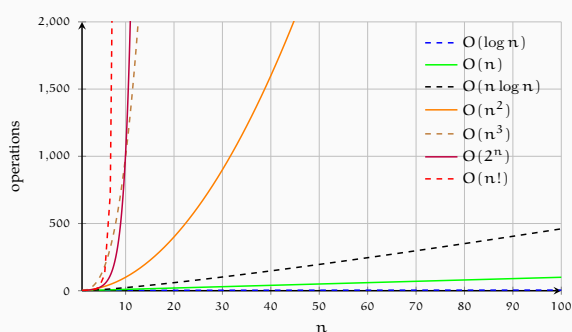
$O(1)$  constant  
 $O(\log n)$  logarithmic  
 $O(n)$  linear  
 $O(n \log n)$  log linear  
 $O(n^2)$  quadratic  
 $O(n^3)$  cubic  
 $O(2^n)$  exponential  
 $O(n!)$  factorial

Given  $T(n)$ , what is  $O(n)$ ?

- $T(n) = \log(5n)$
- $T(n) = 5n$
- $T(n) = n + \log n$
- $T(n) = n^2 + 10$
- $T(n) = n^5 + n^4$
- $T(n) = n^5 + 4^n$
- $T(n) = n! + 2^n$

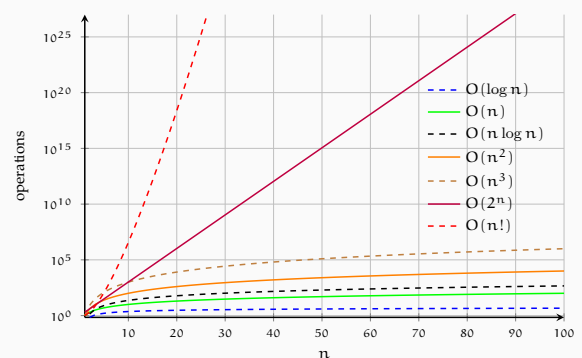
## Big-O notation and order of complexity

the picture



## Big-O notation and order of complexity

the picture (with log y-axis)



## A(nother) review of computational complexity

P, NP, NP-complete and all that

- A major division of complexity classes according to Big-O notation is between
  - P polynomial time algorithms
  - NP non-deterministic polynomial time algorithms
- A big question in computing is whether  $P = NP$
- All problems in NP can be reduced in polynomial time to a problem in a subclass of NP, (*NP-complete*)
  - Solving an NP complete problem in P would mean proving  $P = NP$

Video from <https://www.youtube.com/watch?v=YX40hbAHx3s>

## RE languages and Turing machines

- Recursively enumerable languages can be generated by *Turing machines*
- Turing machine is an simple model of computation that can compute any computable function
  - A Turing machine manipulates symbols on an infinite tape, using a finite table of rules
- A Turing machine can enumerate all string defined by an unrestricted phrase structure grammar
- The membership problem of RE languages is not decidable

## Context-free languages and pushdown automata

- Context-free languages are recognized by *pushdown automata*
- Pushdown automata consist of a finite-state control mechanism and a stack
- Computationally feasible solutions exists for many problems related to context-free grammars
- There are polynomial time algorithms for recognizing strings of context-free languages (we will return to these in lectures on parsing)

## Chomsky hierarchy and natural language syntax

Where do natural languages fit?

- The class of grammars adequate for formally describing natural languages has been an important question for (computational) linguistics
- For the most part, context-free grammars are enough, but there are some examples, e.g., from Swiss German (Shieber 1985)  
Jan säit das...

...mer **em** Hans **es** huss **hãlfed** **aastriiche**  
 ...we Hans (DAT) the house (ACC) helped paint

Note that this resembles  $a^n b^m c^n d^m$ .

## Grammars and automata

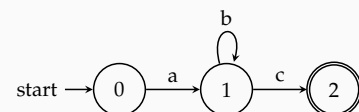
Language	Grammar	Automata
Regular	Regular	Finite-state
Context-free	Context-free	Push-down
Context-sensitive	Context-sensitive	Linear-bounded
Recursively-enumerable	Unrestricted	Turing machines

## Context-sensitive languages and LBA

- Context-sensitive languages can be generated using a restricted form of Turing machine, called *linear-bounded automata*
- Although decidable, recognition of a string with a context-sensitive grammar is computationally intractable (PSPACE-complete)

## Regular languages and FSA

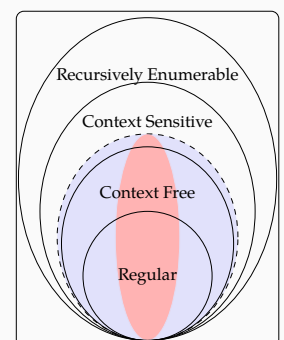
- Regular languages can be recognized using *finite-state automata* (FSA)
- A FSA consist of a finite set of states with directed edges between them
- Edges are labeled with the terminal symbols, and tell the automation to which state to move on a given input symbol



## Where do natural languages fit?

the picture

- Often a superset of CF languages, *mildly context-sensitive languages* are considered adequate
- Note, though, we do not even need full RE expressivity
- Modern/computational theories of grammars range from mildly CS (TAG, CCG) to Turing complete (HPSG, LFG?)



## Learnability natural languages

language acquisition & nature vs. nurture

- A central question in linguistics have been about ‘learnability’ of the languages
- Some linguists claim that natural languages are not learnable, hence, humans born with a innate *language acquisition device*
- A poplar theory of the *language acquisition device* is called *principles and parameters*
- This has created a long-lasting debate, which is also related to even longer-lasting debate on nature vs. nurture

## Formal languages and learnability

- Some of the arguments in the learnability debate has been based on results on formal languages
- It is shown (Gold [1967](#)) that none of the languages in the Chomsky hierarchy are learnable from positive input
- The applicability of such results to human language acquisition is questionable
- Computational modeling/experiments may help here (another job for computational linguists)

## Wrapping up

- Formal languages has a central role in the theory of computation, as well as in formal/computational linguistics
- Practically-useful classes of languages in Chomsky hierarchy is regular and context-free languages (we will return to these in more detail)
- Natural language syntax can be described mostly by CFGs

Next:

- Finite state automata

## References / additional reading material

- The classic reference for theory of computation is Hopcroft and Ullman ([1979](#)) (and its successive editions)
- Sipser ([2006](#)) is another good textbook on the topic
- A popular nativist account of language acquisition debate is Pinker ([1994](#))
- A popular non-nativist (somewhat empiricist) book on language acquisition is Clark and Lappin ([2011](#)), which also covers discussion of (Gold [1967](#)) and later work

## References / additional reading material (cont.)

- 📖 Clark, Alexander and Shalom Lappin (2011). *Linguistic Nativism and the Poverty of the Stimulus*. Oxford: Wiley-Blackwell. ISBN: 978-1-4051-8785-5.
- 📖 Gold, E. Mark (1967). “Language identification in the limit”. In: *Information and Control* 10.5, pp. 447–474.
- 📖 Hopcroft, John E. and Jeffrey D. Ullman (1979). *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Series in Computer Science and Information Processing. Addison-Wesley. ISBN: 9780201029888.
- 📖 Pinker, Steven (1994). *The language instinct: the new science of language and mind*. Penguin Books.
- 📖 Shieber, Stuart M. (1985). “Evidence against the context-freeness of natural language”. In: *Linguistics and Philosophy* 8.3, pp. 333–343. DOI: [10.1007/BF00630917](#).
- 📖 Sipser, Michael (2006). *Introduction to the Theory of Computation*. second. Thomson Course Technology. ISBN: 0-534-95097-3.