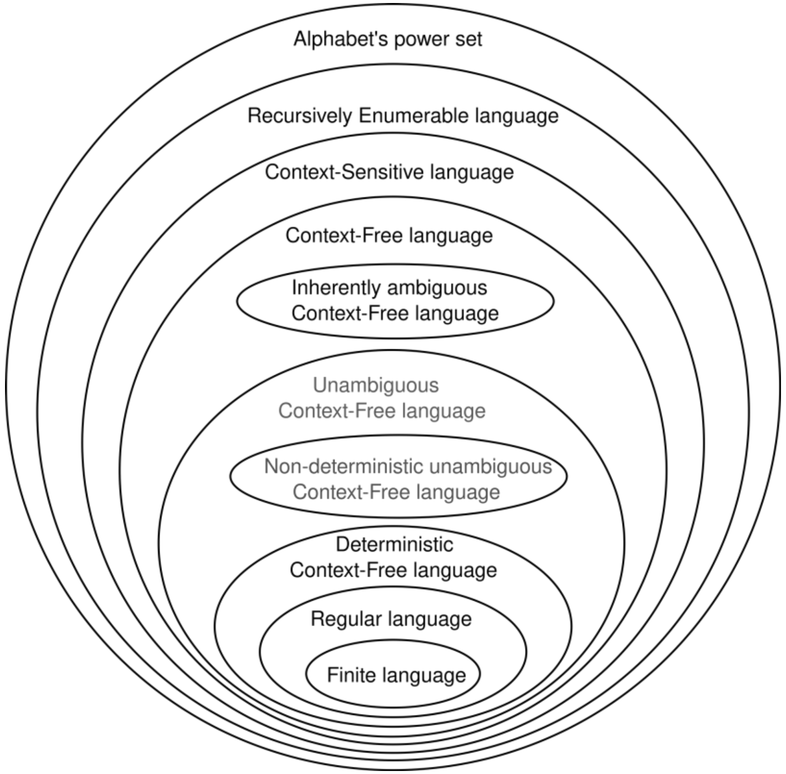
University of Ljubljana, Faculty of Computer and Information Science

Formal languages

Tiny introduction



Prof Dr Marko Robnik-Šikonja

Natural language processing, Edition 2025

Lecture outline

- regular expressions (here just for your quick revision)
- context dependent grammars
- Chomsky hierarchy

Regular expressions - a quick resume 1/3

- standard notation for characterizing text sequences
- used in all kinds of text processing and information extraction tasks
- many different syntaxes (Perl, grep, sed, awk, Python, etc)
- let‘s use regular expressions (RE) from python
- if A and B are REs then AB is RE
- a,b,…,z, A, B,… Z,0,1,…,9 are REs
- e.g. abeceda is RE
- . matches any character, e.g.: va.a matches vaba or vaza or vaya
- ^ matches the start of a string; ^.oga matches noga or joga, but not nadloga
- $ matches the end of a string
- * matches 0 or more repetitions of the previous RE: ab* matches a, ab, abb, …
- + matches 1 or more repetitions of the previous RE: ab+ matches ab, abb, … but not a

Regular expressions 2/3

- ? matches 0 or 1 repetitions of the previous RE: ab? matches a or ab
- *, + and ? are greedy: they match the longest possible string, e.g., <.*> on the string <a> b <c> matches the whole string
- *?, +?, ?? cause minimal matching of *, +, and ?, e.g.,.: <.*?> on the string <a> b <c> will match <a>
- {m} matches m repetitions of a previous RE: b{5} matches only bbbbb
- {m,n} matches from m to n repetitions of a previous RE
- {,n} is the same as {0,n}
- {m,} is the same as {m,∞}
- {m,n}? is a non-greedy variant of {m,n}
- \ is an escape character, it makes the next character special, e.g.,
  \\ matches \
  * matches *

Regular expressions 3/3

- [] represents a set of characters, e.g., [abc] matches a, b, or c;
  with [] we can represent a sequence of characters, e.g., [a-z] matches all lowercase letters from a to z
  special characters inside the set are not special, e.g., ?,+,*
- [^] (^ as the first character) represents a complement of a set, e.g., [^abc] matches all characters except a, b, and c
- | in A|B, where A and B are REs, means that RE matches A or B, several REs separated with | is tested from left to right,
  operator | is not greedy
- (…) matches RE in the parenthesis and marks a group, which can be used later or retrieved with \group_number
- (?aiLmsux), where after ? there are one or more letters means:
  a – only ASCII matches, i – ignore lower/uppercase, L – depend on the local settings,
  m – multi-line, s – the dot matches everything, etc. – check the manual
- many other useful details

Example

- Find me all instances of the word “the” in a text.

  `the`

  Misses capitalized examples

  `[tT]he`

    Incorrectly returns `other` or `theology`

  `[^a-zA-Z][tT]he[^a-zA-Z]`

Errors

- The process we just went through was based on fixing two kinds of errors:

1. Matching strings that we should not have matched (there, then, other)

   False positives (Type I errors)

2. Not matching things that we should have matched (The)

   False negatives (Type II errors)

Errors cont.

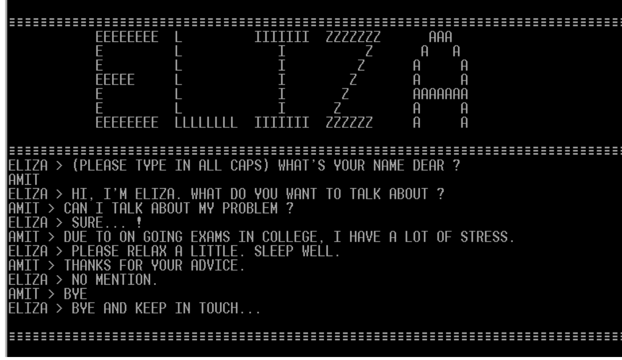
- In NLP we are always dealing with these kinds of errors.
- Reducing the error rate for an application often involves two antagonistic efforts:
  - Increasing accuracy or precision (minimizing false positives)
  - Increasing coverage or recall (minimizing false negatives).

Substitutions

- Substitution in Python and UNIX commands:

- `s/regexp1/pattern/`
- e.g.:
- `s/colour/color/`

Simple Application: ELIZA



- Early NLP system that imitated a Rogerian psychotherapist (Weizenbaum, 1966).

- Uses pattern matching to match, e.g.,:
  - `“I need X”`

and translates them into, e.g.

  - `“What would it mean to you if you got X?`

Simple Application: ELIZA

Men are all alike.
IN WHAT WAY

They're always bugging us about something or other. CAN YOU THINK OF A SPECIFIC EXAMPLE

Well, my boyfriend made me come here.
YOUR BOYFRIEND MADE YOU COME HERE

He says I'm depressed much of the time.
I AM SORRY TO HEAR YOU ARE DEPRESSED

How ELIZA works

- s/.* I’M (depressed|sad) .*/I AM SORRY TO HEAR YOU ARE \1/
- s/.* I AM (depressed|sad) .*/WHY DO YOU THINK YOU ARE \1/
- s/.* all .*/IN WHAT WAY?/
- s/.* always .*/CAN YOU THINK OF A SPECIFIC EXAMPLE?/

Summary

- Regular expressions play a surprisingly large role
  - Sophisticated sequences of regular expressions are often the first model for any text processing text
- For hard tasks, we use machine learning classifiers
  - But regular expressions are still used for pre-processing, or as features in the classifiers
  - Can be very useful in capturing generalizations

RE exercises

Write regular expressions for the following languages

- the set of all alphabetic strings;
- the set of all lower case alphabetic strings ending in a b
- the set of all strings with two consecutive repeated words (e.g., “Humbert
- Humbert” and “the the” but not “the bug” or “the big bug”);
- the set of all strings from the alphabet a,b such that each a is immediately
- preceded by and immediately followed by a b;
- all strings that start at the beginning of the line with an integer and that end at the end of the line with a word;
- all strings that have both the word grotto and the word raven in them (but not, e.g., words like grottos that merely contain the word grotto);

Formal Languages and Models

- Language: a (possibly infinite) set of strings made up of symbols from a finite alphabet
- Model of a language: can *recognize* and *generate* **all** and **only** the strings from the language
  - Serves as a definition of the formal language
- Alphabet Σ is a finite set of symbols, e.g., Σ ={0,1} or Σ={a,b,c,d}.
- String is a sequence of symbols from alphabet
- ε is an empty set
- $\Sigma \cup \Sigma\Sigma$ is a set of all strings of length 1 or 2
- Σ^* is a set of all strings from alphabet
- imprecise notation, e.g., 0 is a symbol and 0 is a string, depending on the context

Merrill, W., 2021. Formal Language Theory Meets Modern NLP. *arXiv preprint arXiv:2102.10094.*
About formal languages and their relation with neural networks.

Language

- Language is a subset of Σ* for an alphabet Σ.
- Example: language of 0 and 1, where there are no two consecutive 1s
- L = {ε, 0, 1, 00, 01, 10, 000, 001, 010, 100, 101, 0000, 0001, 0010, 0100, 0101, 1000, 1001, 1010, . . . }

Chomsky Hierarchy

- Regular language
  - Model: regular expressions, finite state automata
- Context free language
- Context sensitive language
- Unrestricted language
  - Model: Turning Machine

Regular Expressions and Languages

- A regular expression pattern can be mapped to a set of strings
- A regular expression pattern defines a language (in the formal sense) – the class of this type of languages is called a regular language

An example of non-regular language

$L_1 = \{0^n 1^n \mid n \geq 1\}$

$L_1 = \{01, 0011, 000111, \ldots\}$

An example

$L_2 = \{w \mid w \in \{(,)\}^* \text{ with balanced brackets}\}$.

E.g.: (), ()(), (()), (()()),…

Context Free Grammars (CFG)

- A *context-free grammar* is a notation for describing languages.
- It is more powerful than finite automata or RE's, but still cannot define all possible languages.
- Useful for nested structures, e.g., parentheses in programming languages.
- Basic idea is to use "variables" to stand for sets of strings (i.e., languages).
- These variables are defined recursively, in terms of one another.
- Recursive rules ("productions") involve only concatenation.
- Alternative rules for a variable allow union.

Example: CFG for $\{ 0^n 1^n \mid n \geq 1\}$

- Productions:

  S -> 01

  S -> 0S1

- 01 is part of a language
- if w is in the language, so is 0w1

Syntax

- Syntax = rules describing how words can connect to each other

- *that and after year last*
- *I saw you yesterday*
- *colorless green ideas sleep furiously*

- the kind of implicit knowledge of your native language that you had mastered by the time you were 3 or 4 years old without explicit instruction
- not necessarily the type of rules you were later taught in school.

Syntax

- Why should you care?
  - Grammar checkers
  - Question answering
  - Information extraction
  - Machine translation

CFG Formalism

- *Terminals* = symbols of the alphabet of the language being defined.
- *Variables* = *nonterminals* = a finite set of other symbols, each of which represents a language.
- *Start symbol* = the variable whose language is the one being defined.
- A *production* has the form variable -> string of variables and terminals.
- Convention:
  - A, B, C,… are variables.
  - a, b, c,… are terminals.
  - …, X, Y, Z are either terminals or variables.
  - …, w, x, y, z are strings of terminals only.
  - α, β, γ,… are strings of terminals and/or variables.

Example: Formal CFG

- Here is a formal CFG for $\{ 0^n1^n \mid n \geq 1\}$.
- Terminals = {0, 1}.
- Variables = {S}.
- Start symbol = S.
- Productions =

  S -> 01

  S -> 0S1

Derivations – Intuition

- We *derive* strings in the language of a CFG by starting with the start symbol, and repeatedly replacing some variable A by the right side of one of its productions.
  - That is, the "productions for A" are those that have A on the left side of the ->.

Derivations – Formalism

- We say $\alpha A\beta \Rightarrow \alpha\gamma\beta$ if $A \rightarrow \gamma$ is a production.
- Example: S -> 01; S -> 0S1.
- S => 0S1 => 00S11 => 000111.

Iterated Derivation

- =>* means "zero or more derivation steps."
- Basis: α =>* α for any string α.
- Induction: if α =>* β and β => γ, then α =>* γ.

Example: Iterated Derivation

- Productions: S -> 01; S -> 0S1.
- Derivation: S => 0S1 => 00S11 => 000111.
- So: S =>* S; S =>* 0S1; S =>* 00S11; S =>* 000111.

Language of a Grammar

- If G is a CFG, then L(G), the *language of G*, is {w | S =>* w}.
  - Note: w must be a terminal string, S is the start symbol.
- Example: G has productions S -> ϵ and S -> 0S1.
- L(G) = $\{0^n1^n \mid n \geq 0\}$.
- Note: ϵ is a legitimate right side.

Context-Free Languages

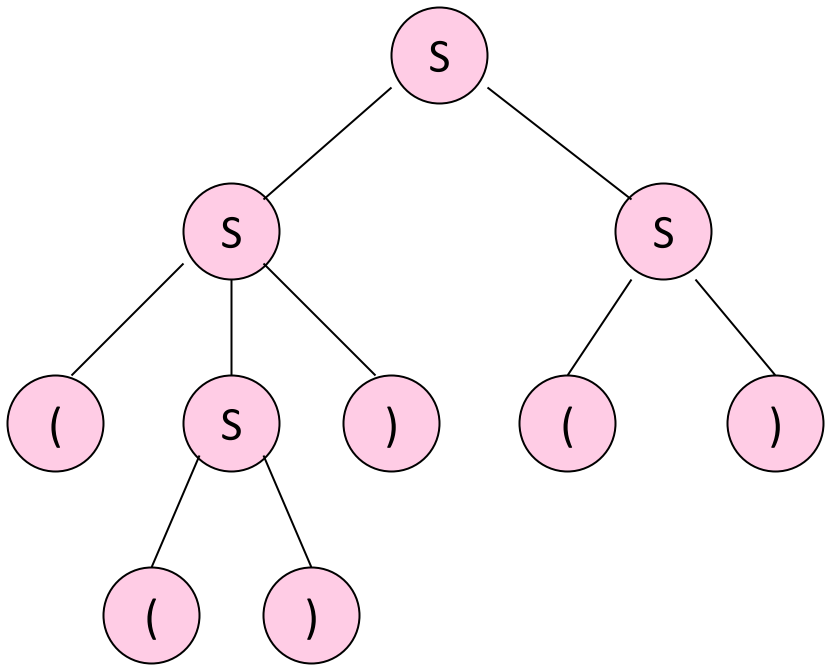
- A language that is defined by some CFG is called a *context-free language*.
- There are CFL's that are not regular languages, such as the example just given.
- But not all languages are CFL's.
- Intuitively: CFL's can count two things, not three.

Parse Trees

- *Parse trees* are trees labeled by symbols of a particular CFG.
- Leaves: labeled by a terminal or ϵ.
- Interior nodes: labeled by a variable.
  - Children are labeled by the right side of a production for the parent.
- Root: must be labeled by the start symbol.

Example: Parse Tree

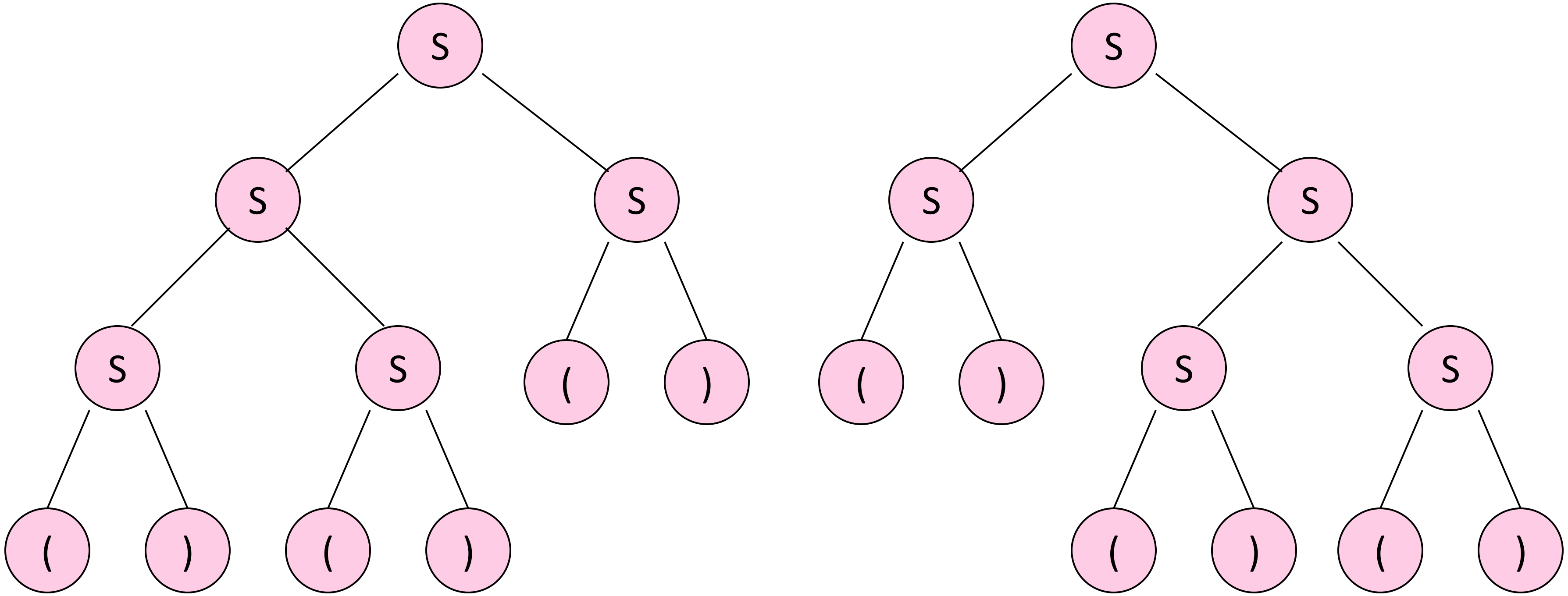
S -> SS | (S) | ()



Ambiguous Grammars

- A CFG is *ambiguous* if there is a string in the language that is the yield of two or more parse trees.
- Example: S -> SS | (S) | ()
- Two parse trees for ()()() on next slide.

Example



Ambiguity is a Property of Grammars, not Languages

- For the balanced-parentheses language, here is another CFG, which is unambiguous.

  B -> (RB | ϵ

  R ->) | (RR

B, the start symbol, derives balanced strings.

R generates strings that have one more right bracket than left.

Inherent Ambiguity

- It would be nice if for every ambiguous grammar, there were some way to “fix” the ambiguity, as we did for the balanced-parentheses grammar.
- Unfortunately, certain CFL’s are *inherently ambiguous*, meaning that every grammar for the language is ambiguous.

Example: Inherent Ambiguity

- The language $\{0^i1^j2^k \mid i = j \text{ or } j = k\}$ is inherently ambiguous.
- Intuitively, at least some of the strings of the form $0^n1^n2^n$ must be generated by two different parse trees, one based on checking the 0's and 1's, the other based on checking the 1's and 2's.

One Possible Ambiguous Grammar

S -> AB | CD

A -> 0A1 | 01

B -> 2B | 2

C -> 0C | 0

D -> 1D2 | 12

A generates equal 0's and 1's

B generates any number of 2's

C generates any number of 0's

D generates equal 1's and 2's

And there are two derivations of every string with equal numbers of 0's, 1's, and 2's. E.g.:
S => AB => 01B =>012
S => CD => 0D => 012

Exercises

- Write CFG for a language
- $L(G) = \{\text{all words of a form } a^n b^m c^k, \text{where } n + m = k\}$
- $L(G) = \{\text{all words of a form } a^n b^m c^k, \text{where } n + k = m\}$

Chomsky Normal Form

- A CFG is said to be in *Chomsky Normal Form* if every production is of one of these two forms:
  1. A -> BC (right side is two variables).
  2. A -> a (right side is a single terminal).
- Theorem: If L is a CFL, then L – {ϵ} has a CFG in CNF.

Decision properties of CFG

1. $w \in L$
2. $L = \{\}$
3. L is infinite
4. $L_1 = L_2$
5. $L_1 \cap L_2 = \{\}$

Algorithm CYK – testing membership

- CYK: Cocke – Younger – Kasami
- CFG={V,T,S,P}
- answers the question $x \in L$ (or equivalently $S \Rightarrow^* x$)
- examples
  - is a given program correct according to the given grammar
  - is the given sentence grammatically correct
- requires CFG in Chomsky normal form
- $O(n^3)$, where $n = |w|$.

CYK Algorithm

- Let $w = a_1 \ldots a_n$.
- We construct an n-by-n triangular array of sets of variables.
- X_{ij} = {variables A | A =>* $a_i \ldots a_j$}.
- Induction on $j–i+1$.
  - The length of the derived string.
- Finally, ask if S is in X_{1n}.

CYK Algorithm – (2)

- Basis: X_{ii} = {A | A -> a_i is a production}.
- Induction: X_{ij} = {A | there is a production A -> BC and an integer k, with $i \leq k < j$, such that B is in X_{ik} and C is in $X_{k+1,j}$.

CYK example

- $S \rightarrow A\ B$
  $A \rightarrow BC \mid a$
  $B \rightarrow CC \mid b$
  $C \rightarrow a$
- ? $S \rightarrow aaab$

	a	a	a	b
1	A,C	A,C	A,C	B
2	B	B	S	
3	S,A	/		
4	S			

CYK exercises

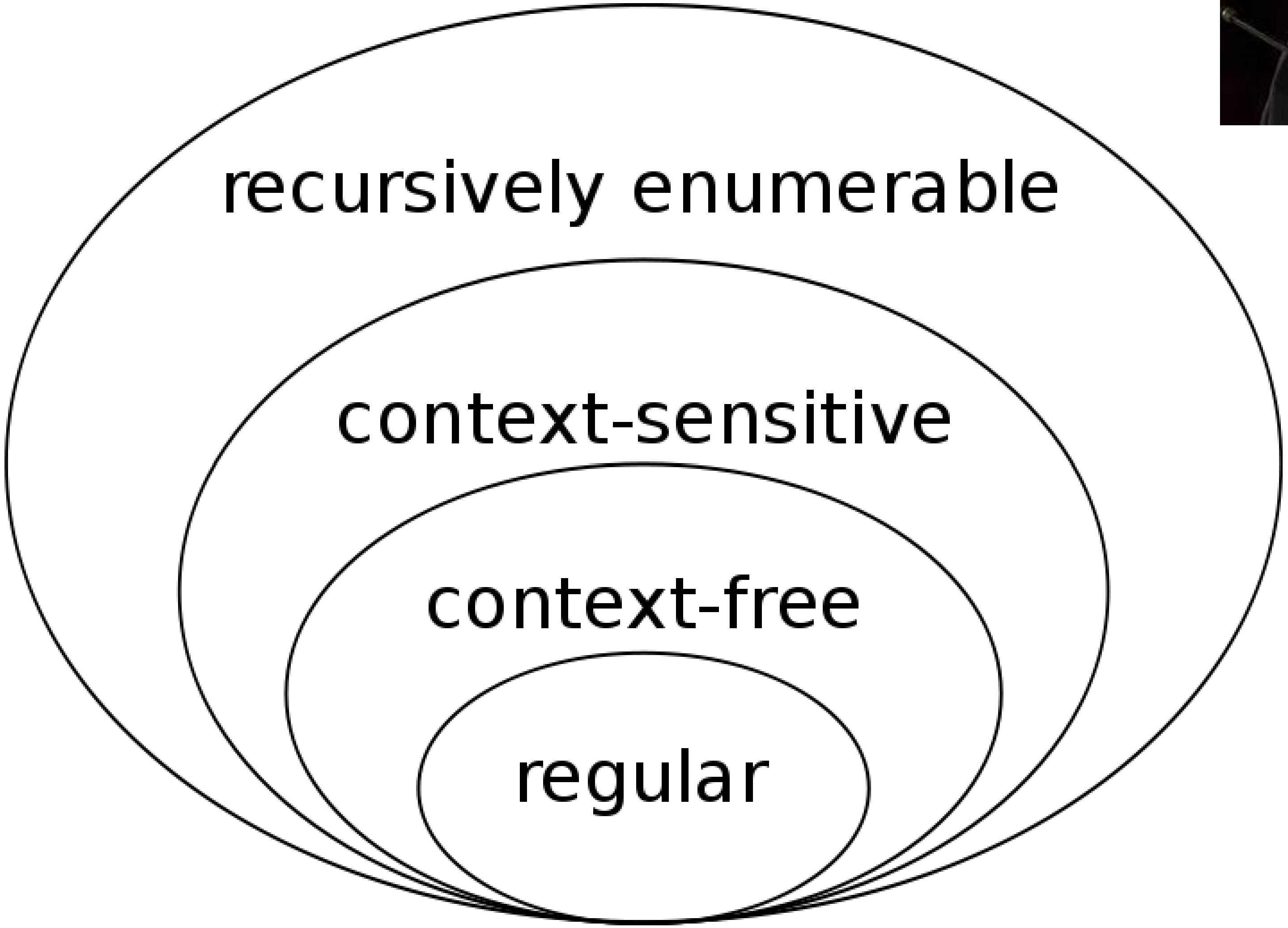
- $S \to P\ N \mid other$
  $P \to I\ E$
  $I \to if$
  $E \to expression$
  $N \to T\ S$
  $T \to then$
- is the sentence correct
  $S \to$ if expression then if expression then other

- $S \to A\ C \mid B\ D \mid A\ E$
  $C \to B\ B$
  $D \to A\ A$
  $E \to B\ A \mid A\ B$
  $A \to a \mid A\ E \mid E\ A \mid B\ D$
  $B \to b \mid B\ E \mid E\ B \mid A\ C$
- ? $S \to baabba$

Tools for grammars

- gnu programs bison and yacc
- based on CFG, they generate a recognizer code in C, C++, or java

Chomsky hierarchy



Order 3

- Order 3 grammars are regular languages
- Grammars of the form

$S \rightarrow aA$
$S \rightarrow a$

Order 2

- CFGs
- Form $A \rightarrow \alpha$
- α is a string of terminals and nonterminals
- programming languages

Order 1

- Context dependent grammars CDG
- Form $\alpha A\beta \rightarrow \alpha\gamma\beta$
- A is a variable, α, β, and γ are strings of terminals and nonterminals
- α and β can be empty, γ has to be non-empty
- natural languages

Order 0

- Unbounded (Turing) grammars and Turing languages, i.e., languages recognizable by Turing machines
- Form $\alpha \to \beta$

- There are languages unrecognizable with Turing machines – diagonal proof