Natural language syntax: parsing and complexity

Timothée Bernard and Pascal Amsili

Université Paris Cité, Université Sorbonne Nouvelle
timothee.bernard@u-paris.fr, pascal.amsili@ens.fr

Ljubljana, Slovenia – August 7-11, 2023
ESSLLI foundational course in Language and Computation

Overview of the course

- Day 1: Formal languages and syntactic complexity.
- Day 2: The complexity of natural language.
- Day 3: Historic algorithms for parsing.
- Day 4: Modern approaches to parsing.
- Day 5: Neural networks and error propagation.

Day 1

Today's contents

- Formal languages.
- Automata.
- Formal grammars.
- The recognition and the parsing problems.
- The Chomsky-Schützenberger hierarchy.

Languages are sets of words (finite sequences of symbols)

- **Alphabet** (Σ): finite set of symbols called **letters**.

  (1) Examples:
  a. $\{0, 1\}$
  b. $\{a, b, c, \cdots, z\}$

- (Finite) **Word** (w): finite sequence of letters.

  (2) Examples:
  a. 000110101
  b. *bonjour*
  c. ϵ (the empty word)

- (Formal) **Language**: set of words.

(examples soon)

A letter is anything considered atomic

- "letter" $\equiv$ "atomic"
- $w = \textit{Hello world}!$ can be seen as a word on $\Sigma = \{\textit{Hello}, \textit{world}, !\}$.
- **Length**: $|w| = 3$
- **Indices**: $w_1 = \textit{Hello}$, $w_2 = \textit{world}$, $w_3 = !$

Languages can be simple or weird

- The set L_1 of Arabic numerals, on $\Sigma = \{0, 1, \dots, 9\}$.
  $0, 291, 9999 \in L_1$; $00003 \notin L_1$
  $(L_1 = \{w \in \Sigma^+ \mid w_1 \neq 0 \vee |w| = 1\})$
- The set L_2 of Roman numerals, on $\Sigma = \{I, V, X, L, C, D, M\}$.
  $I, MMXXIII, VIII \in L_2$; $IIX \notin L_2$
- The set L_3 of first-order logic formulas, on
  $\Sigma = \{\wedge, \neg, (,), p, q, r, s, \dots\}$.
  $p, (\neg p), (q \wedge r) \in L_3$; $p\neg \notin L_3$
- The set of valid zip files, on $\Sigma = \{0, 1\}$.
- The set of Python programs, on the set of characters allowed to write them.
- The set of theorems of ZFC (set theory), on the set of characters allowed to write them.

Languages can be very simple or very weird

- Given some Σ...
- The **empty language** $\emptyset$ (*no word* is in $\emptyset$).
- The **full language** $\Sigma^\star$ (*any word on* Σ is in $\Sigma^\star$).
- Some "random" language L obtained by going through all $w \in \Sigma^\star$, tossing a fair coin and including w in L in case of a head.

Natural languages can be seen as formal languages

- Let Σ be the set of English words (+ punctuation and digits).
- (English words are here considered to be atomic.)
- Let L be the grammatical sentences of English seen as sequences of symbols in Σ.
- (This definition requires binary grammaticality judgments for all sequences; $\rightarrow$ Day 2.)
- $L \subseteq \Sigma^\star$, is a formal language.

The recognition problem: computing grammaticality

- Given Σ and $L \subseteq \Sigma^\star$...
- The **recognition problem** for L:

  Given some $w \in \Sigma^\star$, is w in L?
- Very easy if L is finite.
- Easy for the set of Arabic numerals, slightly more complex for Roman numerals.
- A bit harder for the set of programs in Python.
- Quite hard for the set of theorems of ZFC.
- Impossible (except if you're *very* lucky) for a random language.
- What about a natural language such as English? $\rightarrow$ Day 2

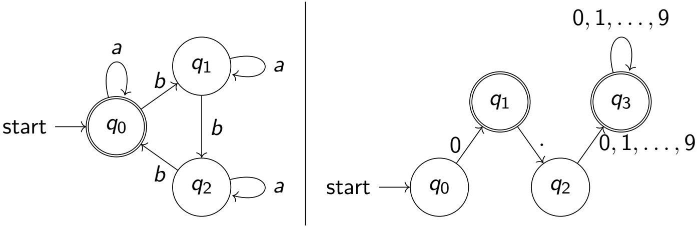
There is not just one notion of complexity

- Worst-case **time complexity** of an algorithm: Given an input of size n, *how many basic steps* are required to run the algorithm?
- Worst-case **space complexity** of an algorithm: Given an input of size n, *how much memory* is required to run the algorithm?
- ...
- These notions usually assume the *Turing machine* as model of computation.
- The recognition problem is traditionally studied using another notion of complexity, based on multiple models of computation; what *type of memory* is used?

A DFA has a finite fixed amount of memory

- **Deterministic Finite-state Automaton** (DFA): $(\Sigma, Q, q_0, F, \delta)$ where
  - Σ is an alphabet;
  - Q is a finite set (of **states**);
  - $q_0 \in Q$ (the **initial state**);
  - $F \subseteq Q$ (**final states**);
  - δ is a function $Q \times \Sigma \to Q$ (the **transition function**).
- Memory: Nothing beyond the states themselves.

A DFA encodes a formal language



- A word w is **accepted** if reading w leads from the initial state to a final state.
- For a DFA A, $\mathcal{L}(A)$ is the set of words that A accepts.
- Here?
- Not all languages are encoded (**recognised**) by a DFA; ex: $\{a^n b^n \mid n \in \mathbb{N}\}$ (proof in Day 2)

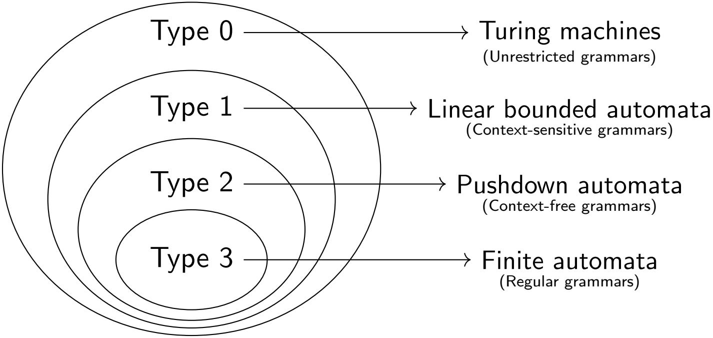
Memory is (computational) power

- Other automata models have, in addition, a memory structure that is used in transitions.
- There is also a notion of *(non-)determinism*, but let's ignore this.
- The models in the next slide have increasing computational power.
- **Computational power**: the ability to solve problems.

Stacks and tapes of memory increase computational power

- **Pushdown automaton**: an unbounded$_{\text{i}}$ *stack*$_{\text{a}}$ of memory;
  - a) only the top cell can be read/overwritten/cleared, a new can can be added on top, the stack is initially empty and the input word is still written on a dedicated buffer,
  - i) no limit to the number of cells;
- **Linear bounded automaton**: a *linearly bounded*$_{\text{ii}}$ *tape*$_{\text{b}}$ of memory;
  - b) a movable "head" points to a cell, only this cell can be read/written, the input word is initially written on the tape rather than on a dedicated buffer,
  - ii) the maximum number of cells is given by a linear function of the length of the input word;
- **Turing machine**: an unbounded$_{\text{i}}$ tape$_{\text{b}}$ of memory.

The Chomsky-Schützenberger hierarchy



- 4+1 **complexity classes** of languages are represented here.
- "+1" because some languages are beyond type 0.
- Non-deterministic versions of the models. (→ matters for types 1 and 2)

Grammars are finite sets of rewriting rules

- **Unrestricted grammar**: (N, Σ, P, S) where
  - N is a finite set (of **non-terminal symbols**);
  - Σ is an alphabet;
  - $P \subseteq (N \cup \Sigma)^+ \times (N \cup \Sigma)^\star$ is a finite set (of **production rules**);
  - $S \in N$ (**the axiom**);

  and $N \cap \Sigma = \emptyset$.
- Production rules are rewriting rules; (α, β) is noted "$\alpha \to \beta$".
- Using $bX \to Xab$, $abXc$ can be rewritten as $aXabc$; this fact is noted "$abXc \underset{bX \to Xab}{\Rightarrow} aXabc$".
- In $\alpha \to \beta$, α is the **left-hand side** and β the **right-hand side**.

Grammars generate languages

- $w \in \Sigma^\star$ is **generated** by a grammar if there is a **derivation** $S \Rightarrow \ldots \Rightarrow w$.
- Like automata, grammars encode (generate) languages.
- Example with $G = (\{S\}, \{a, b\}, \{S \to \epsilon, S \to aSb\}, S)$:
  - derivations:
    - $S \Rightarrow \epsilon$
    - $S \Rightarrow aSb \Rightarrow ab$
    - $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$
    - $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbb$
    - $\ldots$
  - $\mathcal{L}(G) = \{a^n b^n \mid n \in \mathbb{N}\}$
- Rmk: Two distinct grammars can generate the same language.

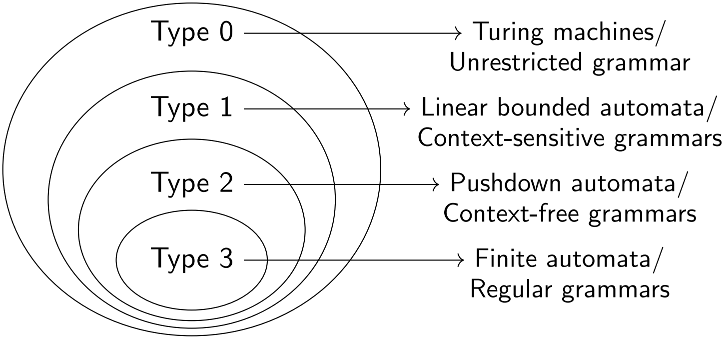
Rewriting is (expressive) power

- Other grammatical formalisms restrict the form of production rules.
- The formalisms in the next slide have decreasing expressive power.
- These formalisms match the previous models of automata.

Rewriting is (expressive) power

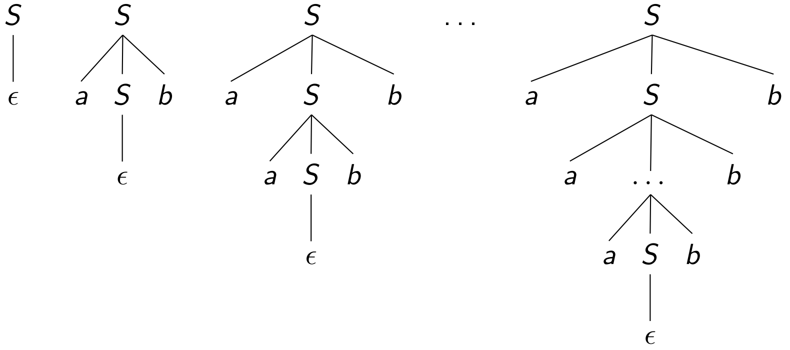
- **Context-sensitive grammar** (CSG): [intuition by examples]
  ex: $abXc \to abYzZc$
- **Context-free grammar** (CFG): the left-hand side of a rule is a single non-terminal symbol.
  ex: $X \to YzZ$
- **Regular grammar** (RG): in addition, the right-hand side of a rule is either empty (ϵ), a single non-terminal symbol, or a non-terminal followed by a terminal symbol.
  ex: $X \to \epsilon$, $X \to a$, $X \to aY$

The Chomsky-Schützenberger hierarchy



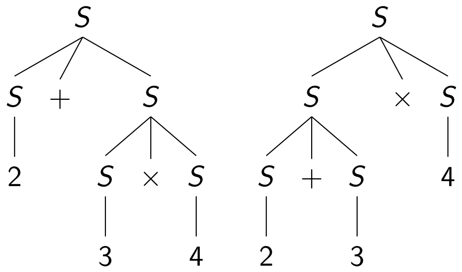
R, CF and CS derivations are constituent trees

- $G = (\{S\}, \{a, b\}, \{S \rightarrow \epsilon, S \rightarrow a\,S\,b\}, S)$ is a CFG.
- $\mathcal{L}(G) = \{a^n b^n \mid n \in \mathbb{N}\}$



Ambiguity is when a word has two structures

- A grammar G is ambiguous iff $\exists w \in \mathcal{L}(G)$ s.t. w has two distinct syntactic structures (according to G).
- $G = (\{S\}, \{0, 1, \cdots, 9, +, -\}, \{S \to 0|1| \cdots |9|S{+}S|S{\times}S\}, S)$
- $w = 2 + 3 \times 4$:



The parsing problem: finding derivations

- Given a grammar G on some alphabet $\Sigma\ldots$
- The **parsing problem** for G:

  Given some $w \in \Sigma^\star$,
  what are the derivations (if any) of w in G?

- (Solving the parsing problem for G entails solving the recognition problem for $\mathcal{L}(G)$.)
- Practical solutions to the parsing problem: Days 3-4.

Syntactic complexity vs semantic expressivity

- Context-free grammars are commonly used to describe the syntax of many logical languages (e.g. PL, FOL), some programming languages, and parts of NL ($\rightarrow$ Day 2).
- Untyped λ-calculus: CF syntax, Turing-complete semantics. "How is this possible?"
- $\rightarrow$ The syntactic complexity and the semantic expressivity of interpreted languages are two distinct notions.
- Jot (https://en.wikipedia.org/wiki/Iota_and_Jot) is $\{0, 1\}^{\star}$, a regular language, compositionally interpreted as a Turing-complete language.

The recognition/parsing problems are very general

- Consider any binary ("yes/no") problem P and see it as the set of inputs for which the answer is positive.
- Let str be a linearisation function for the possible inputs of P, and $L = \{str(in) \mid in \in P\}$.
- Solving P is equivalent to the recognition problem for L.
- More generally, any computable function f can be encoded as a grammar s.t. after parsing the input w, the output $f(w)$ can be read off the derivation.
- $\rightarrow$ One can compute "syntactically": a grammar is a program. (The parser is the machine that runs it.)
- The formalism of unrestricted grammars is a Turing-complete programming language. (syntactically regular?)

Exercise: Checking addition as CF parsing/recognition

- Unary notation of natural integers:
  - "" for 0;
  - "i" for 1;
  - "ii" for 2;
  - "iii" for 3;
  - ...
- Exercise: Write a CFG G on $\Sigma = \{i, +, =\}$ that generates exactly the strings "$a + b = c$" for all natural numbers a, b and c written in unary notation and s.t. $a + b = c$.
- With G, a CF parser can solve this arithmetic problem.
- In other words, some non-deterministic pushdown automaton can solve this problem. (in fact, a deterministic one can)

Exercise: Boolean satisfiability as CF parsing/recognition

- Consider the set of propositional logic formulas built from (at most) n propositional letters p_1, p_2, ..., p_n.
  Ex: $(p_1 \wedge (\neg p_2))$, $(\neg(\neg(p_2 \wedge p_5)))$, $(p_4 \wedge (\neg p_4))$
- Problem: Which of these formulas are satisfiable?
  Ex: $(p_1 \wedge (\neg p_2))$ and $(\neg(\neg(p_2 \wedge p_5)))$ but not $(p_4 \wedge (\neg p_4))$
- Exercise: Write a CFG G on $\Sigma = \{\wedge, \neg, (,), p_1, p_2, \cdots, p_n\}$ that generates exactly L, the set of satisfiable formular.
- With G, a CF parser can solve this satisfiability problem.
- In other words, some non-deterministic pushdown automaton can solve this problem.
- Hint: First consider an arbitrary interpretation function $\{p_1, p_2, \cdots, p_n\} \rightarrow \{0, 1\}$, then generalise.

Day 1: Summary

- Languages are sets of words (finite sequences of symbols).
- Automata are finite state machines with or without additional memory.
- Grammars are finite sets of rewriting rules.
- The parsing problem for a grammar consists in finding derivations.
- All solvable problems can be expressed as parsing problems.
- The Chomsky-Schützenberger hierarchy is a hierarchy of classes of languages, of models of automata, and of grammatical formalisms.
- For interpreted languages, syntactic complexity is not semantic expressivity.