Overview

Formal Languages

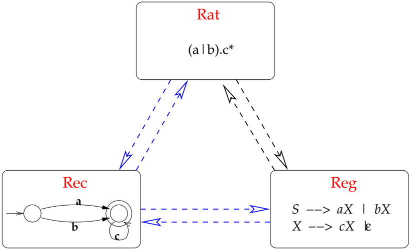
Regular Languages
- Automata
- Properties
- Regular expressions
- Definition

Formal Grammars

Formal complexity of Natural Languages

Definition

1. a regular language can be defined by rational/regular expressions
2. a regular language can be recognized by a finite automaton
3. a regular language can be generated by a regular grammar



Overview

Formal Languages

Regular Languages

Formal Grammars

Examples

Definition

Language classes

Formal complexity of Natural Languages

Example I

$$
\begin{array}{lcl}
S & \rightarrow & A\,B \\
A & \rightarrow & aA \\
 & | & b \\
B & \rightarrow & bBc \\
 & | & \varepsilon
\end{array}
$$

- Rewriting system
- Auxiliary vocabulary (N for non-terminal)
- Start symbol (engendered language)
- Multiple derivations
- Syntactic tree

Example II

$$
\begin{array}{rcl}
E & \rightarrow & E + E \\
 & | & E \times E \\
 & | & (\, E \,) \\
 & | & 0 \mid 1 \mid 2 \ldots 8 \mid 9
\end{array}
$$

- Syntactic ambiguity
- Semantic interpretation

Sorbonne Nouvelle

Example III

$$
\begin{array}{lcl}
NP & \rightarrow & Det\ N' \\
N' & \rightarrow & AdjP\ N' \\
N' & \rightarrow & N \\
N' & \rightarrow & N\ Cpt \\
AdjP & \rightarrow & Adj\ AdjP \\
AdjP & \rightarrow & Adj \\
Cpt & \rightarrow & P\ NP \\
Det & \rightarrow & \text{the} \mid \text{my} \\
N & \rightarrow & \text{cat} \mid \text{friend} \\
Adj & \rightarrow & \text{large} \mid \text{fierce} \\
Prep & \rightarrow & \text{of} \mid \text{to}
\end{array}
$$

- X-bar theory
- Recursive rules
- Center-embedding

Overview

Formal Languages

Regular Languages

Formal Grammars
- Examples
- Definition
- Language classes

Formal complexity of Natural Languages

Formal grammar

Def. 12 ((Formal) Grammar)

A **formal grammar** is defined by $\langle \Sigma, N, S, P \rangle$ where

- Σ is an alphabet
- N is a disjoint alphabet (non-terminal vocabulary)
- $S \in V$ is a distinguished element of N, called the *axiom*
- P is a set of « *production rules* », namely a subset of the cartesian product $(\Sigma \cup N)^* N (\Sigma \cup N)^* \times (\Sigma \cup N)^*$.

Sorbonne Nouvelle

Immediate Derivation

Def. 13 (Immediate derivation)

Let $\mathcal{G} = \langle \Sigma, N, S, P \rangle$ a grammar,
$r \in P$ a production rule, such that $r : A \longrightarrow u$ with $u \in (\Sigma \cup N)^*$;
$f, g \in (\Sigma \cup N)^*$ two "(proto-)words",

- f derives into g (immediate derivation) with the rule r
  (noted $f \xrightarrow{r} g$) iff
  $\exists v, w$ s.t. $f = vAw$ and $g = vuw$
- f derives into g (immediate derivation) in the grammar $\mathcal{G}$
  (noted $f \xrightarrow{\mathcal{G}} g$) iff
  $\exists r \in P$ s.t. $f \xrightarrow{r} g$.

Derivation

Def. 14 (Derivation)

$$
f \xrightarrow{\mathcal{G}*} g \text{ if } \begin{array}{ll} f = g & \text{or} \\ \exists f_0, f_1, f_2, ..., f_n \text{ s.t.} & \\ \quad f_0 = f & \\ \quad f_n = g & \\ \quad \forall i \in [1, n] : f_{i-1} \xrightarrow{\mathcal{G}} f_i & \end{array}
$$

Engendered language

Def. 15 (Language engendered by a word)

Let $f \in (\Sigma \cup N)^*$.

$L_{\mathcal{G}}(f) = \{g \in X^* / f \xrightarrow{\mathcal{G}*} g\}$

Def. 16 (Language engendered by a grammar)

The *language engendered by a grammar* $\mathcal{G}$ is the set of words of Σ^* derived from the axiom.

$L_{\mathcal{G}} = L_{\mathcal{G}}(S)$

Overview

Formal Languages

Regular Languages

Formal Grammars
- Examples
- Definition
- Language classes

Formal complexity of Natural Languages

Principle

Define language families on the basis of properties of the grammars that generate them :

1. Four classes are defined, they are included one in another
2. A language is of type k if it **can** be recognized by a type k grammar (and thus, by definition, by a type $k-1$ grammar) ; and **cannot** be recognized by a grammar of type $k+1$.

Chomsky's hierarchy

type 0 No restriction on $P \subset (X \cup V)^* V (X \cup V)^* \times (X \cup V)^*$.

type 1 (*context-sensitive* grammars) All rules of P are of the shape $(u_1 S u_2, u_1 m u_2)$, where u_1 and $u_2 \in (X \cup V)^*$, $S \in V$ and $m \in (X \cup V)^+$.

type 2 (*context-free* grammar) All rules of P are of the shape (S, m), where $S \in V$ and $m \in (X \cup V)^*$.

type 3 (*regular* grammars) All rules of P are of the shape (S, m), where $S \in V$ and $m \in X.V \cup X \cup \{\varepsilon\}$.

Examples

type 3:

$$\begin{array}{lcl} S & \rightarrow & aS \mid aB \mid bB \mid cA \\ B & \rightarrow & bB \mid b \\ A & \rightarrow & cS \mid bB \end{array}$$

Examples

type 3:

$$\begin{array}{lcl} S & \rightarrow & aS \mid aB \mid bB \mid cA \\ B & \rightarrow & bB \mid b \\ A & \rightarrow & cS \mid bB \end{array}$$

type 2:

$E \rightarrow E + T \mid T, T \rightarrow T \times F \mid F, F \rightarrow (E) \mid a$

Example 1 type 0

Type 0:

$$\begin{array}{llll} S & \rightarrow SABC & AC \rightarrow CA & A \rightarrow a \\ S & \rightarrow \varepsilon & CA \rightarrow AC & B \rightarrow b \\ AB & \rightarrow BA & BC \rightarrow CB & C \rightarrow c \\ BA & \rightarrow AB & CB \rightarrow BC & \end{array}$$

generated language :

Example 1 type 0

Type 0:

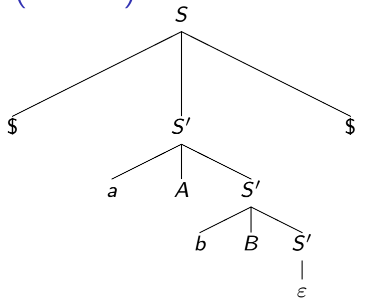
$$\begin{array}{lll} S \rightarrow SABC & AC \rightarrow CA & A \rightarrow a \\ S \rightarrow \varepsilon & CA \rightarrow AC & B \rightarrow b \\ AB \rightarrow BA & BC \rightarrow CB & C \rightarrow c \\ BA \rightarrow AB & CB \rightarrow BC & \end{array}$$

generated language : words with an equal number of a, b, and c.

Example 2: type 0

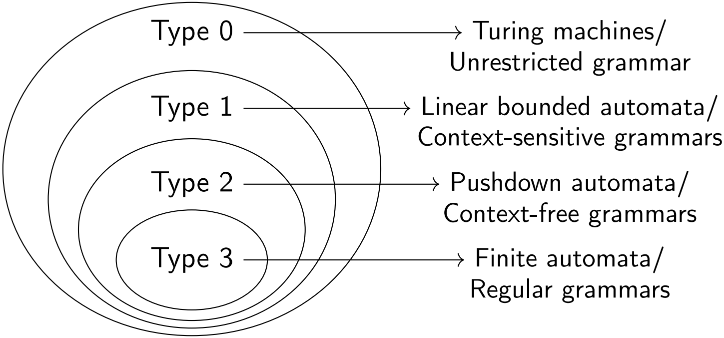
$$
\text{Type 0:}\quad
\begin{array}{lll}
S \rightarrow \$S'\$ & Aa \rightarrow aA & \$a \rightarrow a\$ \\
S' \rightarrow aAS' & Ab \rightarrow bA & \$b \rightarrow b\$ \\
S' \rightarrow bBS' & Ba \rightarrow aB & A\$ \rightarrow \$a \\
S' \rightarrow \varepsilon & Bb \rightarrow bB & B\$ \rightarrow \$b \\
 & & \$\$ \rightarrow \#
\end{array}
$$

Example 2: type 0 (cont'd)



$\$$	a	A	b	B	$\$$
a	$\$$	A	b	B	$\$$
a	$\$$	A	b	$\$$	b
a	$\$$	b	A	$\$$	b
a	b	$\$$	A	$\$$	b
a	b	$\$$	$\$$	a	b
a	b	$\#$		a	b

The Chomsky-Schützenberger hierarchy



Remarks

- Type 0 (Turing-recognizable) = recursively enumerable languages
  Type 1 (Turing-decidable) = recursive languages
- There are others ways to classify languages,
  - either on other properties of the grammars;
  - or on other properties of the languages
- Nested structures are preferred, but it's not necessary

The parsing problem: finding derivations

- Given a grammar G on some alphabet Σ...
- 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 $\mathcal{L}(G)$.)

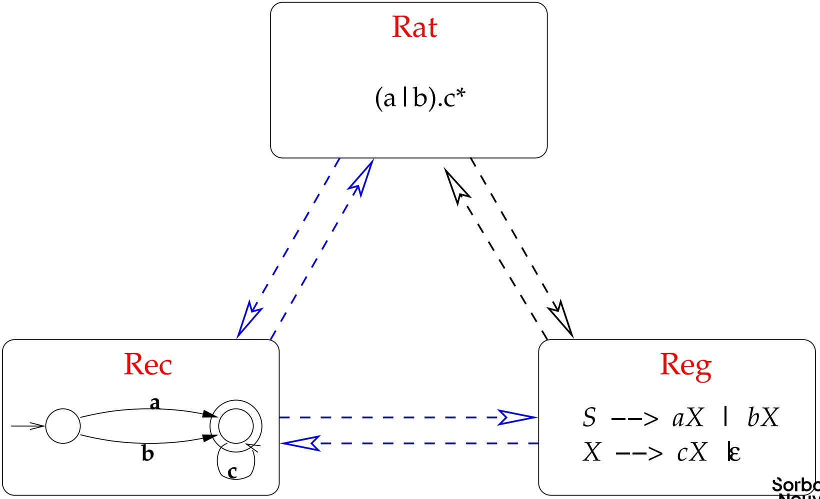
Syntactic complexity vs semantic expressivity

- 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?)

Back to regular languages



Let's play with grammars

For each of the following grammars, give the generated language, and the type they have in Chomsky's hierarchy.

$$\begin{array}{lcl} S & \rightarrow & S_1 S_2 \\ S_1 & \rightarrow & aS_1 b \mid ab \\ S_2 & \rightarrow & cS_2 \mid c \end{array}$$

$$\begin{array}{lcl} S & \rightarrow & aSBC \\ S & \rightarrow & aBC \\ CB & \rightarrow & BC \\ aB & \rightarrow & ab \\ bB & \rightarrow & bb \\ bC & \rightarrow & bc \\ cC & \rightarrow & cc \end{array}$$

Let's play with grammars (cont'd)

Give a contex-free grammar that generates each of the following languages (alphabet $\Sigma = \{a, b, c\}$).

- $L_0 = \{w \in X^* \ / \ w = a^n \ ; \ n \geq 0\}$
- $L'_0 = \{w \in X^* \ / \ w = a^n b^n ca \ ; \ n \geq 0\}$
- $L_1 = \{w \in X^* \ / \ w = a^n b^n c^p ; n > 0 \text{ et } p > 0\}$
- $L_2 = \{w \in X^* \ / \ w = a^n b^n a^m b^m ; n, m \geq 1\}$
- $L'_3 = \{w \in X^* \ / \ |w|_a = |w]_b\}$
- $L_3 = \{w \in X^* \ / \ |w|_a = 2|w]_b\}$
- $L_4 = \{w \in X^* \ / \ \exists x \in X^* \text{ tq } w = x\overline{x}\}$
- $L_5 = \{w \in X^* \ / \ w = \overline{w}\}$