Formal Languages and Linguistics

Pascal Amsili

Sorbonne Nouvelle, Lattice (CNRS/ENS-PSL/USN)

PSL Master's degree in Cognitive Science, March 2026

General introduction

1. Mathematicians (incl. Chomsky) have formalized the notion of **language**

   oversimplification ?
   maybe...
2. It buys us:
   2.1 Tools to think about theoretical issues about language/s (expressiveness, complexity, comparability...)
   2.2 Tools to manipulate concretely language (e.g. with computers)
   2.3 A research programme:
      - Represent the syntax of natural language in a fully unambiguously specified way

Now let's get familiar with the mathematical notion of language

Overview

Alphabet, word

Def. 1 (Alphabet)

An *alphabet* Σ is a finite set of symbols (letters).
The *size* of the alphabet is the cardinal of the set.

Def. 2 (Word)

A *word* on the alphabet Σ is a finite sequence of letters from Σ.
Formally, let $[p] = (1, 2, 3, 4, ..., p)$ (ordered integer sequence).
Then a word is a *mapping*

$$u : [p] \longrightarrow \Sigma$$

p, the length of u, is noted $|u|$.

Examples I

Alphabet	{·, –}
Words	–––·
	·
	··–·
	...
Alphabet	{·– , –··· , –·–· , –·· , · , ... }
Words	··· ––– ···
	–·· ·· –·· · ·–· ––– –
	·– –··· –··· ·– ·– –··· –··· ·–
	...

Examples II

Alphabet	{0,1,2,3,4,5,6,7,8,9, · }
Words	235 · 29
	007 · 12
	·1 · 1 · 00 · ·
	~~3 · 1415962 . . .~~ (π)
	. . .
Alphabet	{a, woman, loves, man }
Words	a
	a woman loves a woman
	man man a loves woman loves a
	. . .

Monoid

Def. 3 (Σ^*)

Let Σ be an alphabet.
The set of all the words that can be formed with any number of letters from Σ is noted Σ^*

Σ^* includes a word with no letter, noted ε

Example:
$$\begin{aligned} \Sigma &= \{a, b, c\} \\ \Sigma^* &= \{\varepsilon, a, b, c, aa, ab, ac, ba, \ldots, bbb, \ldots\} \end{aligned}$$

N.B.: Σ^* is always infinite, except...

Monoid

Def. 3 (Σ^*)

Let Σ be an alphabet.
The set of all the words that can be formed with any number of letters from Σ is noted Σ^*

Σ^* includes a word with no letter, noted ε

Example:
$$\begin{aligned} \Sigma &= \{a, b, c\} \\ \Sigma^* &= \{\varepsilon, a, b, c, aa, ab, ac, ba, \ldots, bbb, \ldots\} \end{aligned}$$

N.B.: Σ^* is always infinite, except...
if $\Sigma = \emptyset$. Then $\Sigma^* = \{\varepsilon\}$.

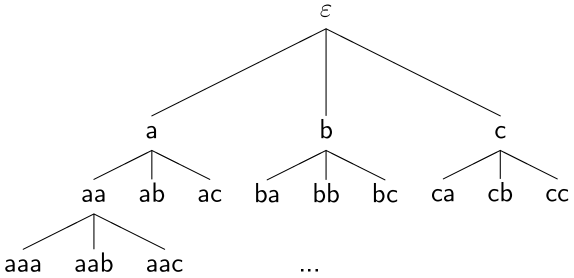
Structure of Σ^*

Let k be the size of the alphabet $k = |\Sigma|$.

Then Σ^* contains :	$k^0 = 1$	word(s) of 0 letters (ε)
	$k^1 = k$	word(s) of 1 letters
	k^2	word(s) of 2 letters
	$\dots$	
	k^n	words of n letters, $\forall n \geq 0$

Representation of Σ^*

$$\Sigma = \{a, b, c\}$$



- Words can be enumerated according to different orders
- Σ^* is a *countable* set

Concatenation

Σ^* can be equipped with a binary operation: *concatenation*

Def. 4 (Concatenation)

Let $[p] \xrightarrow{u} \Sigma$, $[q] \xrightarrow{w} \Sigma$. The concatenation of u and w, noted uw $(u.w)$ is thus defined:

$$\begin{aligned} uw : \quad & [p+q] \longrightarrow \Sigma \\ & uw_i = \begin{cases} u_i & \text{for} \quad i \in [1, p] \\ w_{i-p} & \text{for} \quad i \in [p+1, p+q] \end{cases} \end{aligned}$$

Concatenation

Σ^* can be equipped with a binary operation: *concatenation*

Def. 4 (Concatenation)

Let $[p] \xrightarrow{u} \Sigma$, $[q] \xrightarrow{w} \Sigma$. The concatenation of u and w, noted uw ($u.w$) is thus defined:

$$\begin{aligned} uw :\ & [p+q] \longrightarrow \Sigma \\ & uw_i = \begin{cases} u_i & \text{for} \quad i \in [1, p] \\ w_{i-p} & \text{for} \quad i \in [p+1, p+q] \end{cases} \end{aligned}$$

Example :	u	bacba
	v	cca

Concatenation

Σ^* can be equipped with a binary operation: *concatenation*

Def. 4 (Concatenation)

Let $[p] \xrightarrow{u} \Sigma$, $[q] \xrightarrow{w} \Sigma$. The concatenation of u and w, noted uw $(u.w)$ is thus defined:

$$\begin{aligned} uw : \quad & [p+q] \longrightarrow \Sigma \\ & uw_i = \begin{cases} u_i & \text{for} \quad i \in [1, p] \\ w_{i-p} & \text{for} \quad i \in [p+1, p+q] \end{cases} \end{aligned}$$

Example :		
	u	bacba
	v	cca
	uv	bacbacca

Factor

Def. 5 (Factor)

A *factor* w of u is a subset of adjascent letters in u.

$-w$ is a factor of u	$\Leftrightarrow$	$\exists u_1, u_2$ s.t. $u = u_1 w u_2$
$-w$ is a left factor (*prefix*) of u	$\Leftrightarrow$	$\exists u_2$ s.t. $u = w u_2$
$-w$ is a right factor (*suffix*) of u	$\Leftrightarrow$	$\exists u_1$ s.t. $u = u_1 w$

Def. 6 (Factorization)

We call *factorization* the decomposition of a word into factors.

Sorbonne Nouvelle

Role of concatenation

1. Words have been defined on Σ.
   Given any two words, it's always possible to form a new word by concatenating them.
2. Any word can be factorised in many different ways:
   $a\,b\,a\,c\,c\,a\,b$

Role of concatenation

1. Words have been defined on Σ.
   Given any two words, it's always possible to form a new word by concatenating them.
2. Any word can be factorised in many different ways:
   $a\,b\,a\,c\,c\,a\,b$
   $(a\,b\,a)(c\,c\,a\,b)$

Role of concatenation

1. Words have been defined on Σ.
   Given any two words, it's always possible to form a new word by concatenating them.
2. Any word can be factorised in many different ways:
   $a\,b\,a\,c\,c\,a\,b$
   $(a\,b)(a\,c\,c)(a\,b)$

Role of concatenation

1. Words have been defined on Σ.
   Given any two words, it's always possible to form a new word by concatenating them.
2. Any word can be factorised in many different ways:
   $a\,b\,a\,c\,c\,a\,b$
   $(a\,b\,a\,c\,c)(a\,b)$

Sorbonne Nouvelle

Role of concatenation

1. Words have been defined on Σ.
   Given any two words, it's always possible to form a new word by concatenating them.
2. Any word can be factorised in many different ways:
   $a\,b\,a\,c\,c\,a\,b$
   $(a)(b)(a)(c)(c)(a)(b)$

Role of concatenation

1. Words have been defined on Σ.
   Given any two words, it's always possible to form a new word by concatenating them.
2. Any word can be factorised in many different ways:
   $a\,b\,a\,c\,c\,a\,b$
   $(a)(b)(a)(c)(c)(a)(b)$
3. Since all letters of Σ form a word of length 1
   (this set of words is called the *base*),
4. Any word of Σ^* can be seen as a (unique) sequence of concatenations of length 1 words :
   $a\,b\,a\,c\,c\,a\,b$
   $((((((ab)a)c)c)a)b)$
   $((((((a.b).a).c).c).a).b)$

Properties of concatenation

1. Concatenation is non commutative
2. Concatenation is associative
3. Concatenation has an identity (neutral) element: ε

Notation : $a.a.a = a^3$

1. $uv.w \neq w.uv$
2. $(u.v).w = u.(v.w)$
3. $u.\varepsilon = \varepsilon.u = u$

Overview

Sorbonne Nouvelle

Language

Def. 7 (Formal Language)

Let Σ be an alphabet.
A language on Σ is a set of words on Σ.

Language

Def. 7 (Formal Language)

Let Σ be an alphabet.
A language on Σ is a set of words on Σ.
or, equivalently,
A language on Σ is a subset of Σ^*

Examples I

Let $\Sigma = \{a, b, c\}$.

Examples I

Let $\Sigma = \{a, b, c\}$.

$L_1 = \{aa, ab, bac\}$	finite language

Examples I

Let $\Sigma = \{a, b, c\}$.

$L_1 = \{aa, ab, bac\}$	finite language
$L_2 = \{a, aa, aaa, aaaa \ldots\}$	

Examples I

Let $\Sigma = \{a, b, c\}$.

$L_1 = \{aa, ab, bac\}$	finite language
$L_2 = \{a, aa, aaa, aaaa \ldots\}$ or $L_2 = \{a^i \ / \ i \geq 1\}$	infinite language

Examples I

Let $\Sigma = \{a, b, c\}$.

$L_1 = \{aa, ab, bac\}$	finite language
$L_2 = \{a, aa, aaa, aaaa \ldots\}$ or $L_2 = \{a^i \ / \ i \geq 1\}$	infinite language
$L_3 = \{\varepsilon\}$	finite language, reduced to a singleton

Sorbonne Nouvelle

Examples I

Let $\Sigma = \{a, b, c\}$.

$L_1 = \{aa, ab, bac\}$	finite language
$L_2 = \{a, aa, aaa, aaaa \ldots\}$ or $L_2 = \{a^i \ / \ i \geq 1\}$	infinite language
$L_3 = \{\varepsilon\}$	finite language, reduced to a singleton
	$\neq$

Examples I

Let $\Sigma = \{a, b, c\}$.

$L_1 = \{aa, ab, bac\}$	finite language
$L_2 = \{a, aa, aaa, aaaa \ldots\}$ or $L_2 = \{a^i \ / \ i \geq 1\}$	infinite language
$L_3 = \{\varepsilon\}$	finite language, reduced to a singleton
	$\neq$
$L_4 = \emptyset$	“empty” language

Sorbonne Nouvelle

Examples I

Let $\Sigma = \{a, b, c\}$.

$L_1 = \{aa, ab, bac\}$	finite language
$L_2 = \{a, aa, aaa, aaaa \ldots\}$ or $L_2 = \{a^i \ / \ i \geq 1\}$	infinite language
$L_3 = \{\varepsilon\}$	finite language, reduced to a singleton
	$\neq$
$L_4 = \emptyset$	"empty" language
$L_5 = \Sigma^*$	

Examples II

Let $\Sigma = \{\text{a, man, loves, woman}\}$.

Examples II

Let $\Sigma = \{$a, man, loves, woman$\}$.

$L = \{$ a man loves a woman, a woman loves a man $\}$

Examples II

Let $\Sigma = \{\text{a, man, loves, woman}\}$.

$L = \{$ a man loves a woman, a woman loves a man $\}$

Let $\Sigma' = \{\text{a, man, who, saw, fell}\}$.

Examples II

Let $\Sigma = \{\text{a, man, loves, woman}\}$.

$L = \{$ a man loves a woman, a woman loves a man $\}$

Let $\Sigma' = \{\text{a, man, who, saw, fell}\}$.

$$L' = \left\{ \begin{array}{l} \text{a man fell,} \\ \text{a man who saw a man fell,} \\ \text{a man who saw a man who saw a man fell,} \\ \ldots \end{array} \right\}$$

Set operations

Since a language is a set, usual set operations can be defined:

- union
- intersection
- set difference

Set operations

Since a language is a set, usual set operations can be defined:

- union
- intersection
- set difference

⇒ One may describe a (complex) language as the result of set operations on (~~simpler~~) languages:
$\{a^{2k} \ / \ k \geqslant 1\} = \{a, aa, aaa, aaaa, \dots\} \cap \{ww \ / \ w \in \Sigma^*\}$

Additional operations

Def. 8 (product operation on languages)

One can define the *language product* and its closure *the Kleene star* operation:

- The *product* of languages is thus defined:

$$L_1.L_2 \quad = \quad \{uv \,/\, u \in L_1 \ \& \ v \in L_2\}$$

Notation: $\overbrace{L.L.L \dots L}^{k \text{ times}} = L^k \ ; \ L^0 = \{\varepsilon\}$

- The Kleene star of a language is thus defined:

$$L^* = \bigcup_{n \geqslant 0} L^n$$

Overview

Back to “Natural” Languages

English as a formal language:

alphabet:	morphemes (often simplified to words —depending on your view on flexional morphology)
	$\Rightarrow$ Finite at a time t by hypothesis
words:	well formed English sentences
	$\Rightarrow$ English sentences are all finite by hypothesis
language:	English, as a set of an infinite number of well formed combinations of “letters” from the alphabet

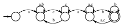
Good questions

Why would one consider natural language as a formal language?

- it allows to $\boxed{\text{describe}}$ the language in a formal/compact/elegant way
- it allows to $\boxed{\text{compare}}$ various languages (via classes of languages established by mathematicians)
- it give algorithmic tools to $\boxed{\text{recognize}}$ and to $\boxed{\text{analyse}}$ words of a language.

recognize u	: decide whether $u \in L$
analyse u	: show the internal structure of u

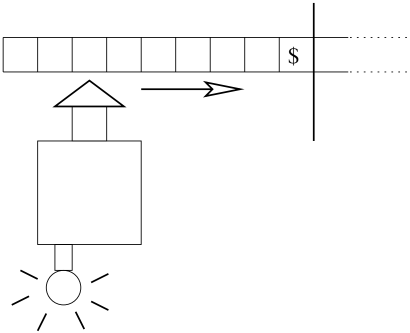
Final remarks

- We are only talking about syntax
- From now on, we'll mostly be looking for precise and efficient ways to **define** a language
  - L $= \{aa, ab, ba\}$
  - L $=$ { all the country names in English }
  - L $=$ { all the inflected forms of French *manger* }
  - L $= \{a^{2^k}$ with $k \geq 0\}$
  - L $= \{ww$ with $w \in \Sigma^*\}$
  - L $= (\{a\} \cup \{b\}.\{c\})^*$ — simplified notation $(a|bc)^*$
  - L $=$ the set of words recognized by this automaton: 
  - L $=$ the set of words engendered by this formal grammar

$$\begin{array}{llll} \mathcal{G}_3: & E & \longrightarrow & E+E \\ & & | & E \times E \\ & & | & (E) \\ & & | & F \\ & F & \longrightarrow & 0|1|2|3|4|5|6|7|8|9 \end{array}$$

Overview

Metaphoric definition



Formal definition

Def. 9 (Finite deterministic automaton (FDA))

A finite state deterministic automaton $\mathcal{A}$ is defined by :

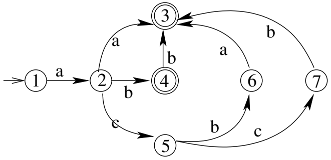
$$\mathcal{A} = \langle Q, \Sigma, q_0, F, \delta \rangle$$

Q	is a finite set of states
Σ	is an alphabet
q_0	is a distinguished state, the initial state,
F	is a subset of Q, whose members are called final/terminal states
δ	is a mapping **fonction** from $Q \times \Sigma$ to Q. Notation $\delta(q, a) = r$.

Example

Let us consider the (finite) language $\{aa, ab, abb, acba, accb\}$.
The following automaton recognizes this langage: $\langle Q, \Sigma, q_0, F, \delta\rangle$, avec $Q = \{1, 2, 3, 4, 5, 6, 7\}$, $\Sigma = \{a, b, c\}$, $q_0 = 1$, $F = \{3, 4\}$, and δ is thus defined:

$$\begin{aligned}
\delta :\quad & (1,\text{a}) \mapsto 2\\
& (2,\text{a}) \mapsto 3\\
& (2,\text{b}) \mapsto 4\\
& (2,\text{c}) \mapsto 5\\
& (4,\text{b}) \mapsto 3\\
& (5,\text{b}) \mapsto 6\\
& (5,\text{c}) \mapsto 7\\
& (6,\text{a}) \mapsto 3\\
& (7,\text{b}) \mapsto 3
\end{aligned}$$



	a	b	c
→ 1	2		
2	3	4	5
← 3			
← 4		3	
5		6	7
6	3		
7		3	

Recognition

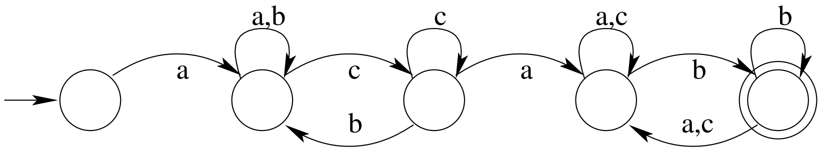
Recognition is defined as the existence of a sequence of states defined in the following way. Such a sequence is called a path in the automaton.

Def. 10 (Recognition)

A word $a_1a_2...a_n$ is **recognized/accepted** by an automaton iff there exists a sequence $k_0, k_1, ..., k_n$ of states such that:

$$k_0 = q_0$$
$$k_n \in F$$
$$\forall i \in [1, n], \quad \delta(k_{i-1}, a_i) = k_i$$

Example

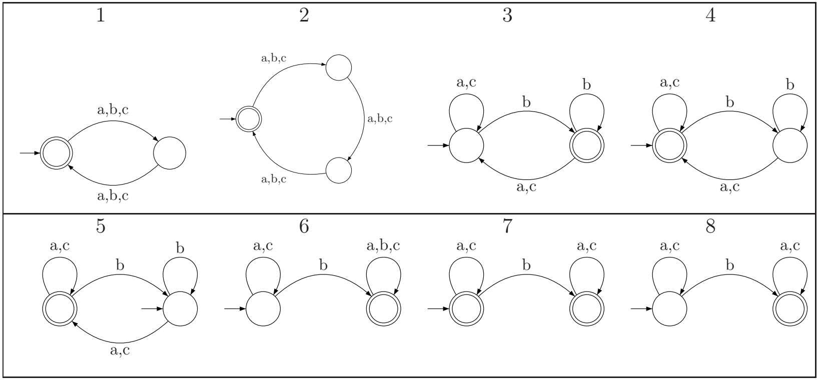


Exercices

Let $\Sigma = \{a, b, c\}$. Give deterministic finite state automata that accept the following languages:

1. The set of words with an even length.
2. The set of words where the number of occurrences of b is divisible by 3.
3. The set of words ending with a b.
4. The set of words not ending with a b.
5. The set of words non empty not ending with a b.
6. The set of words comprising at least a b.
7. The set of words comprising at most a b.
8. The set of words comprising exactly one b.

Answers


1
a,b,c
a,b,c
2
a,b,c
a,b,c
a,b,c
3
a,c
b
b
a,c
4
a,c
b
b
a,c
5
a,c
b
b
a,c
6
a,c
b
a,b,c
7
a,c
b
a,c
8
a,c
b
a,c

Overview

Ways of non-determinism

A word is recognized if there exists a path in the automaton. It is not excluded however that there be several paths for one word: in that case, the automaton is non deterministic.
What are the sources of non determinism?

- $\delta(a, S_1) = \{S_2, S_3\}$
- "spontaneous transition" $=$ ε-transition

Equivalence theorems

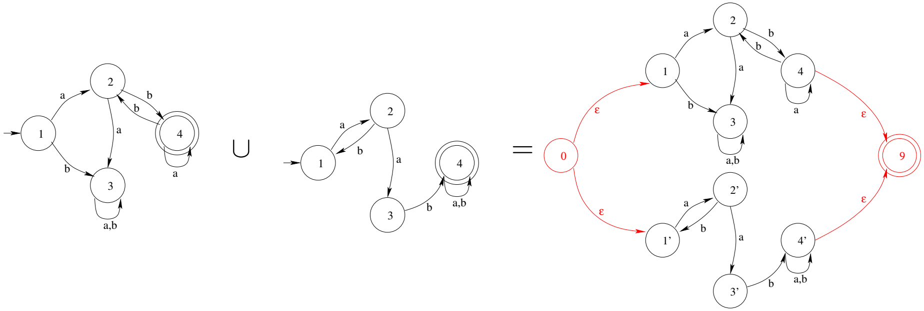
For any non-deterministic automaton, it is possible to design a complete deterministic automaton that recognizes the same language.
Proofs: algorithms (constructive proofs)
First “remove” ε-transitions, then “remove” multiple transitions.

Closure (1)

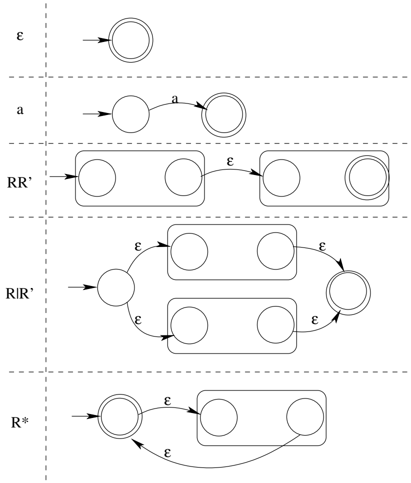
Regular languages are closed under various operations: if the languages L and L' are regular, so are:

- $L \cup L'$ (union); $L.L'$ (product); L^* (Kleene star)
  (rational operations)

Union of regular languages: an example



Rational operations



Closure (2)

Regular languages are closed under various operations: if the languages L and L' are regular, so are:

- $L \cup L'$ (union); $L.L'$ (product); L^* (Kleene star)

  (rational operations)

  $\rightarrow$ for every rational expression describing a language , there is a FSA that recognizes L

Sorbonne Nouvelle

Closure (2)

Regular languages are closed under various operations: if the languages L and L' are regular, so are:

- $L \cup L'$ (union); $L.L'$ (product); L^* (Kleene star) *(rational operations)*

  $\rightarrow$ for every rational expression describing a language , there is a FSA that recognizes L and vice-versa

Closure (2)

Regular languages are closed under various operations: if the languages L and L' are regular, so are:

- $L \cup L'$ (union); $L.L'$ (product); L^* (Kleene star)
  (rational operations)
  $\rightarrow$ for every rational expression describing a language , there is a FSA that recognizes L and vice-versa
- $L \cap L'$ (intersection); $\overline{L}$ (complement)
- ...

Intersection of regular languages

Algorithmic proof
Deterministic complete automata

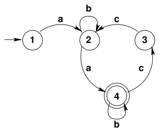
L_1	a	b
$\rightarrow$ 1	2	4
2	4	3
$\leftarrow$ 3	3	3
4	4	4

L_2	a	b
$\leftrightarrow$ 1	2	5
2	5	3
3	4	5
4	1	4
5	5	5

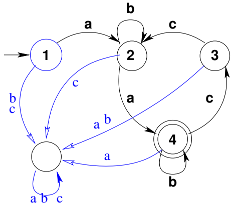
$L_1 \cap L_2$	a	b
$\rightarrow$ (1,1)	(2,2)	(4,5)
(2,2)	(4,5)	(3,3)
(4,5)	(4,5)	(4,5)
(3,3)	(3,4)	(3,5)
(3,4)	(3,1)	(3,4)
$\leftarrow$ (3,1)	(3,2)	(3,4)
(3,2)	(3,4)	(3,3)
(3,5)	(3,5)	(3,5)

Complement of a regular language

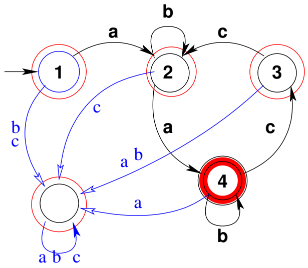
Deterministic complete automata



completed

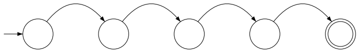


complemented



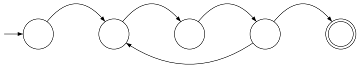
Pumping lemma (intuition)

Take an automaton A with k states.
If $\mathcal{L}(A)$ is infinite,
then $\exists w \in \mathcal{L}(A), |w| \geq k$.
Therefore, when accepting w, A goes through some state q at least twice.
That means that there is a loop $q \overset{w_{i:j}}{\rightarrow} q$.
Repeating the loop any number of times (even 0) always produces a word $(w_{1:i-1}\, {w_{i:j}}^n\, w_{j+1:|w|})$ in $\mathcal{L}(A)$.



Pumping lemma (intuition)

Take an automaton A with k states.
If $\mathcal{L}(A)$ is infinite,
then $\exists w \in \mathcal{L}(A), |w| \geq k$.
Therefore, when accepting w, A goes through some state q at least twice.
That means that there is a loop $q \overset{w_{i:j}}{\rightarrow} q$.
Repeating the loop any number of times (even 0) always produces a word ($w_{1:i-1}\, {w_{i:j}}^n\, w_{j+1:|w|}$) in $\mathcal{L}(A)$.



Pumping lemma (definition)

Pumping Lemma

Let L be a regular language.
$\exists k \in \mathbb{N}$ such that
$\forall w \in L$ such that $|w| \geq k$,
$\exists x, u, y$ such that $w = xuy$ and that

1. $|u| \geq 1$;
2. $|xu| \leq k$;
3. $\forall n \in \mathbb{N}$, $xu^n y \in L$.

$\rightarrow$ "L has the pumping property."

Is NL regular? Pumping lemma (example I)

a^*bc (i.e. $\{a^nbc \mid n \in \mathbb{N}\}$) is regular (there is a DFA).
So, it must have the pumping property.

It happens that $k = 3$ works.
For example, $w = abc \in L$ is long enough and can be decomposed:

$$\underbrace{\epsilon}_{x} \quad \underbrace{a}_{u} \quad \underbrace{b \quad c}_{y}$$

1. $|u| \geq 1$ ($u = a$);
2. $|xu| \leq k$ ($xu = a$);
3. $\forall n \in \mathbb{N}$, xu^ny (i.e. a^nbc) belongs to the language.

Pumping lemma (consequences)

regular	$\Rightarrow$	pumping property satisfied
pumping property **NOT** satisfied	$\Rightarrow$	**NOT** regular
pumping property satisfied	$\not\Rightarrow$	regular

To prove that L is

regular	provide a DFA;
not regular	show that the pumping property is not satisfied.

Pumping lemma (example II)

Let's show that $L = \{a^n b^n \mid n \in \mathbb{N}\}$ is not regular.

- Consider any $k \in \mathbb{N}$.
- Consider $w = a^k b^k \in L$ ($|w| \geq k$).
- If $w = xuy$ with $|u| \geq 1$ and $|xu| \leq k$, then u contains no b.
- But then, $xu^0 y = xy \notin L$ (strictly less as than bs).
- So no $k \in \mathbb{N}$ works; L does not have the pumping property.

A similar reasoning applies to $\{xu^n yv^n z \mid x, y, z, u, v \in \Sigma^*\}$.

Overview

Regular expressions

It is common to use the 3 *rational* operations:

- union
- product
- Kleene star

to characterize certain languages...

Regular expressions

It is common to use the 3 *rational* operations:

- union
- product
- Kleene star

to characterize certain languages...

$$(\{a\} \cup \{b\})^*.\{c\} = \{c, ac, abc, bc, \ldots, baabaac, \ldots\}$$

(simplified notation $(a|b)^*c$ — regular expressions)

Regular expressions

It is common to use the 3 *rational* operations:

- union
- product
- Kleene star

to characterize certain languages...

$$(\{a\} \cup \{b\})^*.\{c\} = \{c, ac, abc, bc, \ldots, baabaac, \ldots\}$$

(simplified notation $(a|b)^*c$ — regular expressions)

... but not all languages can be thus characterized.

Sorbonne Nouvelle

Def. 11 (Rational Language)

A rational language on Σ is a subset of Σ^* inductively defined thus:

- $\emptyset$ and $\{\varepsilon\}$ are rational languages ;
- for all $a \in X$, the singleton $\{a\}$ is a rational language ;
- for all g and h rational, the sets $g \cup h$, $g.h$ and g^* are rational languages.

Results: expressivity

- Any finite langage is regular
- $a^n b^m$ is regular
- $a^n b^n$ is not regular
- ww^R is not regular (^{R} : reverse word)

Decidable problems

- The "word problem" $w \overset{?}{\in} L(\mathcal{A})$ is decidable.

$\Rightarrow$ A computation on an automaton always stops.

Decidable problems

- The "word problem" $w \overset{?}{\in} L(\mathcal{A})$ is decidable.

$\Rightarrow$ A computation on an automaton always stops.

- The "emptiness problem" $L(\mathcal{A}) \overset{?}{=} \emptyset$ is decidable.

$\Rightarrow$ It's enough to test all possible words of length $\leq k$, where k is the number of states.

Decidable problems

- The "word problem" $w \overset{?}{\in} L(\mathcal{A})$ is decidable.

$\Rightarrow$ A computation on an automaton always stops.

- The "emptiness problem" $L(\mathcal{A}) \overset{?}{=} \emptyset$ is decidable.

$\Rightarrow$ It's enough to test all possible words of length $\leq k$, where k is the number of states.

- The "finiteness problem" $L(\mathcal{A})$ $\overset{?}{\text{is}}$ *finite* is decidable.

$\Rightarrow$ Test all possible words whose length is between k and $2k$. If there exists u s.t. $k < |u| < 2k$ and $u \in L(\mathcal{A})$, then $L(\mathcal{A})$ is infinite.

Decidable problems

- The "word problem" $w \overset{?}{\in} L(\mathcal{A})$ is decidable.

$\Rightarrow$ A computation on an automaton always stops.

- The "emptiness problem" $L(\mathcal{A}) \overset{?}{=} \emptyset$ is decidable.

$\Rightarrow$ It's enough to test all possible words of length $\leq k$, where k is the number of states.

- The "finiteness problem" $L(\mathcal{A}) \overset{?}{\text{is}}$ *finite* is decidable.

$\Rightarrow$ Test all possible words whose length is between k and $2k$. If there exists u s.t. $k < |u| < 2k$ and $u \in L(\mathcal{A})$, then $L(\mathcal{A})$ is infinite.

- The "equivalence problem" $L(\mathcal{A}) \overset{?}{=} L(\mathcal{A}')$ is decidable.

$\Rightarrow$ it boils down to answering the question:

$$\left(L(\mathcal{A}) \cap \overline{L(\mathcal{A}')}\right) \cup \left(L(\mathcal{A}') \cap \overline{L(\mathcal{A})}\right) = \emptyset$$

À quoi ça sert?

Why would you want to define (formally) a language?

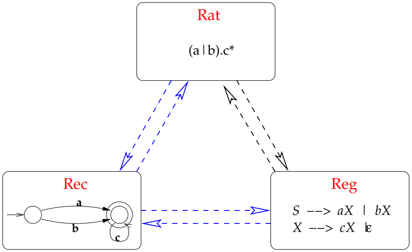
- to formulate a request to a search engine (`mang.*`)
- to associate actions to (classes of) words (e.g., transducers)
    - formal languages (math. expressions, programming languages...)
    - artificial (interface) languages
    - (subpart of) natural languages

Overview

Sorbonne Nouvelle

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

Sorbonne Nouvelle

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

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

- Syntactic ambiguity
- Semantic interpretation

Sorbonne Nouvelle

Example III

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	$\rightarrow$	the | my
N	$\rightarrow$	cat | friend
Adj	$\rightarrow$	large | fierce
$Prep$	$\rightarrow$	of | to

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

Overview

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 $(\Sigma \cup N)^* N (\Sigma \cup N)^* \times (\Sigma \cup N)^*$.

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

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:

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

generated language :

Example 1 type 0

Type 0:

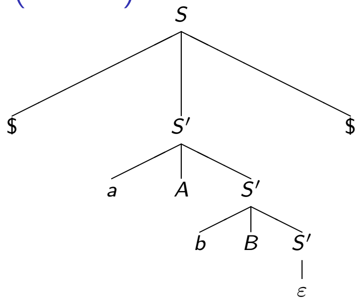
$$\begin{array}{lll} S \rightarrow SABC & AC \rightarrow CA & A \rightarrow a \\ S \rightarrow \varepsilon & CA \rightarrow AC & B \rightarrow b \\ AB \rightarrow BA & BC \rightarrow CB & C \rightarrow c \\ BA \rightarrow AB & CB \rightarrow BC & \end{array}$$

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

Example 2: type 0

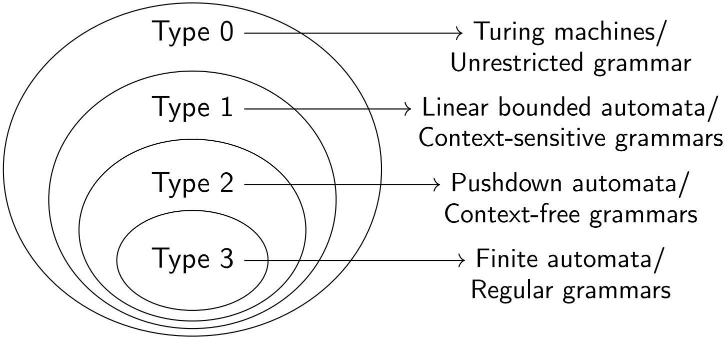
$$\text{Type 0:}\quad \begin{array}{lll} S \rightarrow \$S'\$ & Aa \rightarrow aA & \$a \rightarrow a\$ \\ S' \rightarrow aAS' & Ab \rightarrow bA & \$b \rightarrow b\$ \\ S' \rightarrow bBS' & Ba \rightarrow aB & A\$ \rightarrow \$a \\ S' \rightarrow \varepsilon & Bb \rightarrow bB & B\$ \rightarrow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

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 $\mathcal{L}(G)$.)

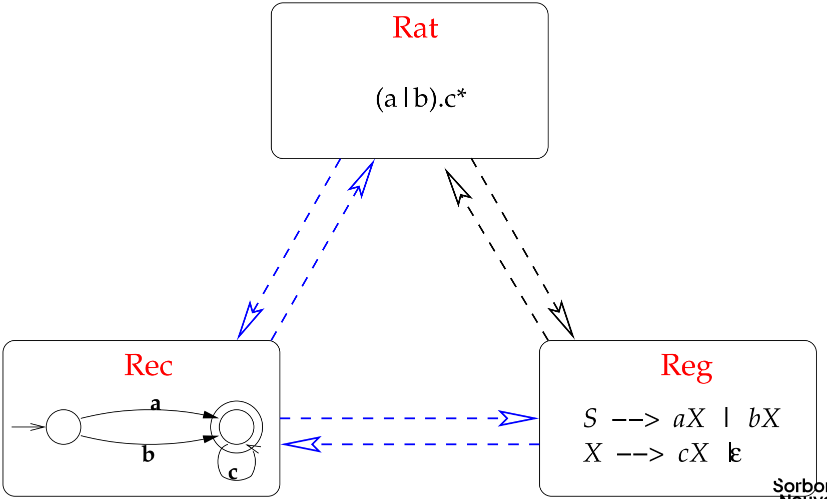
Syntactic complexity vs semantic expressivity

- 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.

$$\begin{aligned} S &\rightarrow S_1 S_2 \\ S_1 &\rightarrow aS_1b \mid ab \\ S_2 &\rightarrow cS_2 \mid c \end{aligned}$$

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

Let's play with grammars (cont'd)

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

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

Overview

Motivation

Why an inquiry into the formal complexity of Natural Language(s)?

- It gives us knowledge about the **structure** of natural languages,
- It helps us assess the **adequation** of linguistic formalisms,
- It gives bound for the **complexity** of NLP tasks,
- It provides us with **predictions** about human language processing.

Hypotheses

We assume that:

- We can talk about "natural language" in general: all languages have a similar structure, a similar power
- Natural languages are recursively enumerable, i.e. they are formal languages
- Natural languages are infinite

$\Rightarrow$ Under these hypotheses, it is possible to ask the question: what is the complexity of natural languages?

An infinite number of sentences

Arbitrary long sentences can be built by adding new material:

(1) A stranger arrived.

An infinite number of sentences

Arbitrary long sentences can be built by adding new material:

(1) A tall stranger arrived.

An infinite number of sentences

Arbitrary long sentences can be built by adding new material:

(1) A tall handsome stranger arrived.

An infinite number of sentences

Arbitrary long sentences can be built by adding new material:

(1) A dark tall handsome stranger arrived.

An infinite number of sentences

Arbitrary long sentences can be built by adding new material:

(1) A very dark tall handsome stranger arrived.

An infinite number of sentences

Arbitrary long sentences can be built by adding new material:

(1) A very very dark tall handsome stranger arrived.

An infinite number of sentences

Arbitrary long sentences can be built by adding new material:

(1) A very very dark tall handsome stranger arrived.

A very^n handsome stranger arrived $\in$ NL

An infinite number of sentences

More interestingly, arbitrary long sentences can be built through center-embedding. In this case, there is a dependancy between arbitrary far apart elements:

An infinite number of sentences

More interestingly, arbitrary long sentences can be built through center-embedding. In this case, there is a dependancy between arbitrary far apart elements:

(2) The cats hunt.

An infinite number of sentences

More interestingly, arbitrary long sentences can be built through center-embedding. In this case, there is a dependancy between arbitrary far apart elements:

(2) The cats the neighbor owns hunt.

An infinite number of sentences

More interestingly, arbitrary long sentences can be built through center-embedding. In this case, there is a dependancy between arbitrary far apart elements:

(2) The cats the neighbor who arrived owns hunt.

An infinite number of sentences

More interestingly, arbitrary long sentences can be built through center-embedding. In this case, there is a dependancy between arbitrary far apart elements:

(2) The cats the neighbor who arrived owns hunt.

center-embedding: embedding a phrase in the middle of another phrase of the same type

Discussion

(3) A man (that a man)^n (hired)^n fired another man.

(4) #A girl that the man that the doctor knows like was fired.

Good examples:

(5) A foreman that an employee who were recently hired talked with was fired.

Discussion: processing problems with nested structures

Psycholinguistic evidence that (6b) is more accepted than (6a) (Fodor, Frazier)

(6) a. The patient who the nurse who the clinic had hired admitted met Jack.
b. The patient who the nurse who the clinic had hired met Jack.

Other factors:

(7) a. The pictures which the photographer who I met yesterday took were damaged by the child.
b. ?The pictures which the photographer who John met yesterday took were damaged by the child.

(8) a. Isn't it true that example sentences [that people [that you know] produce] are more likely to be accepted? (De Roeck et al, 1982)
b. A book [that some Italian [I've never heard of] wrote] will be published soon by MIT Press (Frank, 1992)

(Gibson & Thomas, 1997)

Discussion (end)

- Obvious problems of performance
- however in writing, or with an appropriate intonation, there doesn't seem to be a hard-wired limit

Overview

Chomsky's first attempt

Consider the 3 structures:

- If S_1, then S_2.
- Either S_1 or S_2.
- The man who said S_1 is coming today.

1. The colored items are *dependent* one from the other
2. It is possible to create nested sentences of arbitrary length:

(9) If either the man who said S_a is coming today, or S_b, then S_c.

« Since such sentences are instances of mirroring and since the mirror language is not regular, then English is not regular » *(Chomsky, 1957, p. 22)*.
erroneous claim: **a regular language may contain a non regular sub-language**

Classical argument I

Let's consider the sentence(s):

(10) A man fired another man.

Classical argument I

Let's consider the sentence(s):

(10) A man that a man hired fired another man.

Classical argument I

Let's consider the sentence(s):

(10) A man that a man that a man hired hired fired another man.

Classical argument I

Let's consider the sentence(s):

(10) A man that a man that a man hired hired fired another man.
A man (that a man)^2 (hired)^2 fired another man.

Classical argument I

Let's consider the sentence(s):

(10) A man that a man that a man hired hired fired another man.
A man (that a man)^2 (hired)^2 fired another man.

The sentences (11) are all well-formed sentences (for any *n*).

(11) A man (that a man)^n (hired)^n fired another man.

Classical Argument II

Let x = that a man
y = hired
w = a man
v = fired another man

- wx^*y^*v is regular
- English $\cap\ wx^*y^*v = wx^ny^nv$ (11)
- If English is regular, then wx^ny^nv must be regular (for the intersection of two regular languages is regular)
- But wx^ny^nv is not regular (pumping lemma).
  Contradiction $\Rightarrow$ English is not regular.

(Schieber, 1985)

Sorbonne Nouvelle

Overview

Pumping lemma: intuition

1. If a word is long enough, then there is (at least) one non terminal symbol appearing several times in its derivation.

"long enough" ?

$$\begin{array}{lcl} S & \rightarrow & A\ B \\ A & \rightarrow & abaccabca \\ & | & abSba \\ B & \rightarrow & ccccc \end{array}$$

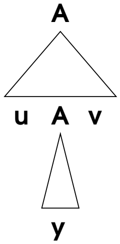
Minimal length : 14:

$S \rightarrow AB \rightarrow abaccabcaB \rightarrow abaccabcaccccc$

Sorbonne Nouvelle

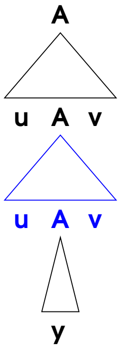
Pumping lemma: intuition

2 Let's call this non terminal symbol A.



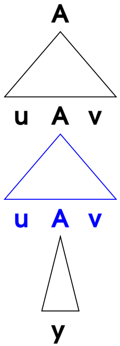
Pumping lemma: intuition

2 Let's call this non terminal symbol *A*.



Pumping lemma: intuition

2 Let's call this non terminal symbol A.



$$A \xrightarrow{*} uAv$$

$$A \xrightarrow{*} uAv \xrightarrow{*} uzv$$

$$A \xrightarrow{*} uAv \xrightarrow{*} uuAvv \xrightarrow{*} \underbrace{u \dots u}_{n} z \underbrace{v \dots v}_{n}$$

Pumping Lemma for CF languages

Def. 17 (Star lemma – CF languages)

If L is context-free, there exists $p \in \mathbb{N}$ such that:
$\forall w$ s.t. $|w| \geqslant p$,
w can be factorized $w = rstuv$,
with:

$$\begin{array}{rl} & |su| \geqslant 1 \\ & |stu| \leqslant p \\ \forall i \geqslant 0, & rs^i tu^i v \in L \end{array}$$

(Bar-Hillel *et al.* , 1961)

Pumping lemma: Consequences

The pumping lemma gives us a tool to prove that a language is **not** context-free.

$\mathcal{L}$ context-free	$\Rightarrow$	pumping lemma $(\forall i, rs^i tu^i v \in \mathcal{L})$
pumping lemma	$\not\Rightarrow$	$\mathcal{L}$ context-free
NO pumping lemma	$\Rightarrow$	$\mathcal{L}$ **NOT** context-free

to prove that $\mathcal{L}$ is

context-free provide a type 2 grammar

not context-free show that the pumping lemma does not apply

Results: expressivity

- well-parenthetized words (dyck's language) is context-free
  $S \rightarrow (S)S \mid \varepsilon$
- $a^n b^n (n \geqslant 0)$ is a context-free language
  $S \rightarrow aSb \mid \varepsilon$
- $ww^R, w \in \Sigma^*$ (mirror language) is a context-free language
  $S \rightarrow aSa \mid bSb \mid \varepsilon$
- $ww, w \in \Sigma^*$ (copy language) is not context-free
  proof: pumping lemma
- $a^n b^n c^n$ is not context-free
  proof: pumping lemma
- $a^m b^n c^m d^n$ is not context-free
  proof: pumping lemma
- $xa^m b^n yc^m d^n z$ is not context-free
  proof: pumping lemma

Sorbonne Nouvelle

Closure properties I

- CF languages are closed under rational operations

- ▶ union (gather all the rules, avoiding name conflicts, and adding a new start rule $S \to S_1|S_2$),
- ▶ product ($S \to S_1 S_2$),
- ▶ and Kleene star ($S \to S_1 S \mid \varepsilon$).

Closure properties II : intersection

- CF languages are not closed under intersection

Example

$L_1 = \{a^i b^i c^j \mid i, j \geq 0\}$ is context-free: $S \rightarrow XY$
$X \rightarrow aXb \mid \varepsilon$
$Y \rightarrow cY \mid \varepsilon$

$L_2 = \{a^i b^j c^j \mid i, j \geq 0\}$ is also context-free: $S \rightarrow XY$
$X \rightarrow aX \mid \varepsilon$
$Y \rightarrow bYc \mid \varepsilon$

But $L_1 \cap L_2 = \{a^n b^n c^n \mid n \geq 0\}$ is not contex-free.

Closure properties III: other results

- CF languages are not closed under complement (since they are not closed under intersection)
- CF languages are closed under intersection with a regular language
- a sub-class of CF languages, *deterministic CF languages* are closed for set complement, but not for union (one can easily define an intrinsequely non deterministic language as the union of two “independant” languages)

Final argument I

After many attempts by various scholars, attempts which are severely critized and ruined in (Gazdar & Pullum, 1985), Schieber (1985) came up with a widely accepted answer:

1. In swiss-german, subordinate clauses can have a structure where all NPs precede all Vs. (12)

(12)

Jan säit	das	mer	NP*	es	huus	haend	wele	V*	aastrüche
Jan said	that	we	NP*	the	house	have	wanted	V*	paint

‘Jan said that we have wanted (that) V* NP* paint the house’

2. Among those subordinate clauses, those where all the dative NPs precede all the accusative NPs are well-formed. (13)

(13)

…	das	mer	d’chind	em Hans	es huus	haend	wele	laa	hälfe	aastrüche
…	that	we	the_children.ACC	Hans.DAT	the house.ACC	have	wanted	let	help	paint

‘… that we have wanted to let the children help Hans to paint the house’

Final argument II

3. The number of verbs requiring a dative has to be equal to the number of dative NPs, the same for accusative.
4. The number of verbs in a subordinate clause is limited only by performance

Let R be the language:

R $=$ {Jan säit das mer (d'chind)^h (em Hans)^i es huus haend wele (laa)^j (hälfe)^k aastrüche, $i, j, k, h \geqslant 1$}

Then let $L =$ Swiss-German $\cap$ $R =$

{Jan säit das mer (d'chind)^m (em Hans)^n es huus haend wele (laa)^m (hälfe)^n aastrüche, $m, n \geqslant 1$}

L is not context-free, whereas R is regular.

$\Rightarrow$ Swiss-German is not context-free.

Overview

Joshi's proposal

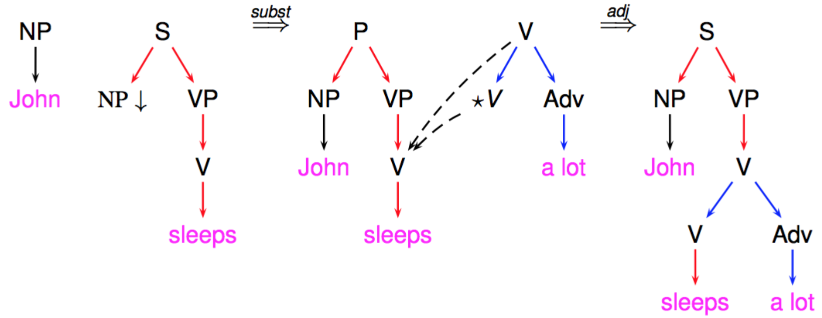
Joshi (1985): what's needed is a class of grammars/languages that are only slightly more powerfull than CFGs.
A class of mildly context-sensitive grammars should have the following properties:

- limited cross-serial dependencies (cf. Swiss-German)
- constant growth (a^{2^i} should not belong to the class)
- polynomial parsing

The class should of course also include all CFG languages.

Formal definitions still needed; note that parsing depends on the grammar rather than on the language

Tree Adjoining Grammars



TAG = MCSL

Tree Adjoining Grammars define the class of MCSL, which have the following properties (among others):

- ww is MCS
- $a^n b^n c^n$ is MCS
- $a^n b^n c^n d^n$ is MCS
- $a^i b^j c^i d^j$ is MCS
- $a^n b^n c^n d^n e^n$ is not MCS
- www is not MCS
- $ab^h ab^i ab^j ab^k ab^l, h > i > j > k > l \geqslant 1$ is not MCS
- a^{2^i} is not MCS

Sorbonne Nouvelle

TAG = MCSL

Tree Adjoining Grammars define the class of MCSL, which have the following properties (among others):

- ww is MCS
- $a^n b^n c^n$ is MCS
- $a^n b^n c^n d^n$ is MCS
- $a^i b^j c^i d^j$ is MCS
- $a^n b^n c^n d^n e^n$ is not MCS
- www is not MCS
- $ab^h ab^i ab^j ab^k ab^l, h > i > j > k > l \geqslant 1$ is not MCS
- a^{2^i} is not MCS

Conjecture : NL $\in$ MCSL

Sorbonne Nouvelle

Categorial Combinatorial Grammars

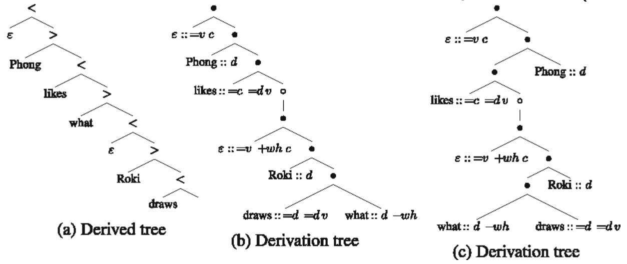
A formalism introduced by Steedman (see (Steedman *et al.* , 2012))

$$\dfrac{\dfrac{\dfrac{\textbf{the}}{NP/N} \qquad \dfrac{\textbf{dog}}{N}}{NP}> \qquad \dfrac{\dfrac{\textbf{bit}}{(S\backslash NP)/NP} \qquad \dfrac{\textbf{John}}{NP}}{S\backslash NP}>}{S}<$$

Vijay-Shanker & Weir (1994) proved the équivalence between CCG and TAG

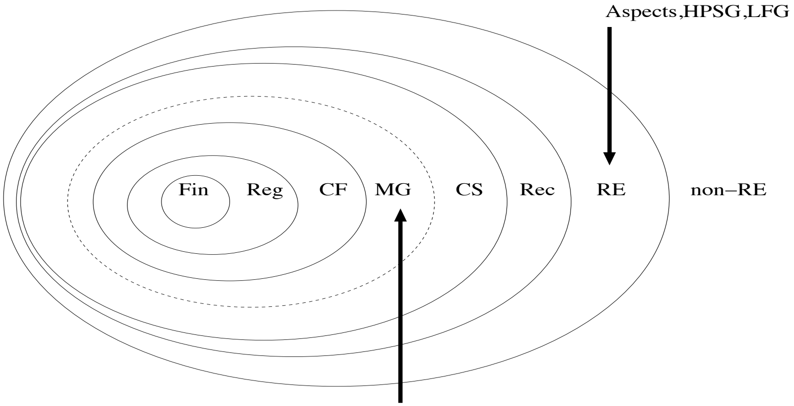
Other formalisms

From the minimalist programme Chomsky (1995), a formalism called Minimalist Grammars was introduced by Stabler (2011).



It has been demonstrated that the class of languages definable by MGs is exactly the class definable by multiple CFG (MCFGs), linear context-free rewrite systems (LCFRSs), and other formalisms.

Big picture (Stabler, 2011)



Th: $CF \subset \boxed{TAG \equiv CCG} \subset \boxed{MCFG \equiv LCFRS \equiv MG} \subset CS$

References I

Bar-Hillel, Yehoshua, Perles, Micha, & Shamir, Eliahu. 1961. On formal properties of simple phrase structure grammars. *STUF-Language Typology and Universals*, 14(1-4), 143–172.

Chomsky, Noam. 1957. *Syntactic Structures*. Den Haag: Mouton & Co.

Chomsky, Noam. 1995. *The Minimalist Program*. Vol. 28. Cambridge, Mass.: MIT Press.

Gazdar, Gerald, & Pullum, Geoffrey K. 1985 (May). *Computationally Relevant Properties of Natural Languages and Their Grammars*. Tech. rept. Center for the Study of Language and Information, Leland Stanford Junior University.

Gibson, Edward, & Thomas, James. 1997. The Complexity of Nested Structures in English: Evidence for the Syntactic Prediction Locality Theory of Linguistic Complexity. *Unpublished manuscript, Massachusetts Institute of Technology*.

Joshi, Aravind K. 1985. *Tree Adjoining Grammars: How Much Context-Sensitivity is Required to Provide Reasonable Structural Descriptions?* Tech. rept. Department of Computer and Information Science, University of Pennsylvania.

Langendoen, D Terence, & Postal, Paul Martin. 1984. *The vastness of natural languages*. Basil Blackwell Oxford.

Mannell, Robert. 1999. *Infinite number of sentences*. part of a set of class notes on the Internet. http://clas.mq.edu.au/speech/infinite_sentences/.

Schieber, Stuart M. 1985. Evidence against the Context-Freeness of Natural Language. *Linguistics and Philosophy*, 8(3), 333–343.

Stabler, Edward P. 2011. Computational perspectives on minimalism. *Oxford handbook of linguistic minimalism*, 617–643.

Steedman, Mark, *et al.* . 2012 (June). *Combinatory Categorial Grammars for Robust Natural Language Processing*. Slides for NASSLLI course http://homepages.inf.ed.ac.uk/steedman/papers/ccg/nasslli12.pdf.

Vijay-Shanker, K., & Weir, David J. 1994. The Equivalence of Four Extensions of Context–Free Grammars. *Mathematical Systems Theory*, **27**, 511–546.