The Mathematical Theory of Formal Languages

Matilde Marcolli and Doris Tsao

Ma191b Winter 2017
Geometry of Neuroscience
References for this lecture:

1. Ian Chiswell, *A course in formal languages, automata and groups*, Springer, 2009


5. A.V. Anisimov, *The group languages*, Kibernetika (Kiev) 1971, no. 4, 18–24

A very general abstract setting to describe languages (natural or artificial: human languages, codes, programming languages, ...)

**Alphabet:** a (finite) set $\mathcal{A}$; elements are *letters* or *symbols*

**Words (or strings):** $\mathcal{A}^m =$ set of all sequences $a_1 \ldots a_m$ of length $m$ of letters in $\mathcal{A}$

**Empty word:** $\mathcal{A}^0 = \{ \epsilon \}$ (an additional symbol)

$$\mathcal{A}^+ = \bigcup_{m \geq 1} \mathcal{A}^m, \quad \mathcal{A}^* = \bigcup_{m \geq 0} \mathcal{A}^m$$

**Concatenation:** $\alpha = a_1 \ldots a_m \in \mathcal{A}^m$, $\beta = b_1 \ldots b_k \in \mathcal{A}^k$

$$\alpha \beta = a_1 \ldots a_m b_1 \ldots b_k \in \mathcal{A}^{m+k}$$

**Length** $\ell(\alpha) = m$ for $\alpha \in \mathcal{A}^m$

**Language:** a subset of $\mathcal{A}^*$

**Question:** how is the subset constructed?
Rewriting system on $\mathbb{A}$: a subset $\mathcal{R}$ of $\mathbb{A}^* \times \mathbb{A}^*$

$(\alpha, \beta) \in \mathcal{R}$ means that for any $u, v \in \mathbb{A}^*$ the word $u\alpha v$ rewrites to $u\beta v$

Notation: write $\alpha \rightarrow_\mathcal{R} \beta$ for $(\alpha, \beta) \in \mathcal{R}$

$\mathcal{R}$-derivation: for $u, v \in \mathbb{A}^*$ write $u \Rightarrow_\mathcal{R} v$ if $\exists$ sequence $u = u_1, \ldots, u_n = v$ of elements in $\mathbb{A}^*$ such that $u_i \rightarrow_\mathcal{R} u_{i+1}$

Grammar: a quadruple $\mathcal{G} = (V_N, V_T, P, S)$

- $V_N$ and $V_T$ disjoint finite sets: non-terminal and terminal symbols
- $S \in V_N$ start symbol
- $P$ finite rewriting system on $V_N \cup V_T$

$P = production$ $rules$

Language produced by a grammar $\mathcal{G}$:

$$\mathcal{L}_\mathcal{G} = \{ w \in V_T^* | S \Rightarrow_P w \}$$

language with alphabet $V_T$
Production rules can be seen as *parsing trees*

Example: Grammar: $\mathcal{G} = \{\{S, A\}, \{a, b\}, P, S\}$ with productions $P$

- $S \rightarrow aAS,
- S \rightarrow a,
- A \rightarrow SbA,
- A \rightarrow SS,
- A \rightarrow ba$

- this is a possible parse tree for the string $aabbaa$ in $\mathcal{L}_G$
Context free and context sensitive production rules

• context free: \( A \rightarrow \alpha \) with \( A \in V_N \) and \( \alpha \in (V_N \cup V_T)^* \)

• context sensitive: \( \beta A \gamma \rightarrow \beta \alpha \gamma \) with \( A \in V_N \)
  \( \alpha, \beta, \gamma \in (V_N \cup V_T)^* \) and \( \alpha \neq \epsilon \)

context free is context sensitive with \( \beta = \gamma = \epsilon \)

“context free” languages: a first attempt (Chomsky, 1956) to model natural languages; not appropriate, but good for some programming languages (e.g. Fortran, Algol, HTML)
The Chomsky hierarchy

Types:

- Type 0: just a grammar $G$ as defined above (unrestricted grammars)
- Type 1: context-sensitive grammars
- Type 2: context-free grammars
- Type 3: regular grammars, where all productions $A \rightarrow aB$ or $A \rightarrow a$ with $A, B \in V_N$ and $a \in V_T$

(right/left-regular if $aB$ or $Ba$ in r.h.s. of production rules)

Language of type $n$ if produced by a grammar of type $n$
Examples

- Type 3 (regular): $G = (\{S, A\}, \{0, 1\}, P, S)$ with productions $P$ given by

  $$S \rightarrow 0S, \quad S \rightarrow A, \quad A \rightarrow 1A, \quad A \rightarrow 1$$

  then $L_G = \{0^m1^n \mid m \geq 0, n \geq 1\}$

- Type 2 (context-free): $G = (\{S\}, \{0, 1\}, P, S)$ with productions $P$ given by

  $$S \rightarrow 0S1, \quad S \rightarrow 01$$

  then $L_G = \{0^n1^n \mid n \geq 1\}$
• Type 1 (context-sensitive): \( G = (\{S, B, C\}, \{a, b, c\}, P, S) \) with productions \( P \)

\[
S \rightarrow aSBC, \quad S \rightarrow aBC, \quad CB \rightarrow BC,
\]
\[
aB \rightarrow ab, \quad bB \rightarrow bb, \quad bC \rightarrow bc, \quad cC \rightarrow cc
\]

the \( \mathcal{L}_G = \{a^n b^n c^n \mid n \geq 1\} \)

Main Idea: a generative grammar \( G \) determines what kinds of recursive structures are possible in the language \( \mathcal{L}_G \)
Why is it useful to organize formal languages in this way?

Types and Machine Recognition

Recognized by:

- Type 0: Turing machine
- Type 1: linear bounded automaton
- Type 2: non-deterministic pushdown stack automaton
- Type 3: finite state automaton

What are these things?
Finite state automaton (FSA)

\[ M = (Q, F, \mathcal{A}, \tau, q_0) \]

- \( Q \) finite set: set of possible states
- \( F \) subset of \( Q \): the final states
- \( \mathcal{A} \) finite set: alphabet
- \( \tau \subset Q \times \mathcal{A} \times Q \) set of transitions
- \( q_0 \in Q \) initial state
computation in $M$: sequence $q_0a_1q_1a_2q_2\ldots a_nq_n$ where
$q_{i-1}a_iq_i \in \tau$ for $1 \leq i \leq n$

- label of the computation: $a_1\ldots a_n$
- successful computation: $q_n \in F$

- $M$ accepts a string $a_1\ldots a_n$ if there is a successful computation
  in $M$ labeled by $a_1\ldots a_n$

Language recognized by $M$:

$$\mathcal{L}_M = \{w \in A^* | w \text{ accepted by } M\}$$
Graphical description of FSA

Transition diagram: oriented finite labelled graph $\Gamma$ with vertices $V(\Gamma) = Q$ set of states and $E(\Gamma) = \tau$, with $e_{q,a,q'}$ an edge from $v_q$ to $v_{q'}$ with label $a \in A$; label vertex $q_0$ with $-$ and all final states vertices with $+$

- computations in $M \Leftrightarrow$ paths in $\Gamma$ starting at $v_{q_0}$
- an oriented labelled finite graph with at most one edge with a given label between given vertices, and only one vertex labelled $-$ is the transition diagram of some FDA
Examples of finite state automata with marked final states
deterministic FSA

for all $q \in Q$ and $a \in \mathcal{A}$, there is a unique $q' \in Q$ with $(q, a, q') \in \tau$  

$\Rightarrow$ function $\delta : Q \times \mathcal{A} \rightarrow Q$ with $\delta(q, a) = q'$, transition function

determines $\delta : Q \times \mathcal{A}^* \rightarrow Q$ by $\delta(q, \epsilon) = q$ and

$\delta(q, wa) = \delta(\delta(q, w), a)$ for all $w \in \mathcal{A}^*$ and $a \in \mathcal{A}$

if $q_0a_1q_1 \ldots a_nq_n$ computation in $M$ then $q_n = \delta(q_0, a_1 \ldots a_n)$

non-deterministic: multivalued transition functions also allowed
Languages recognized by (non-deterministic) FSA are Type 3

- for \( G = (V_N, V_T, P, S) \) type 3 grammar construct an FSA

\[
M = (V_N \cup \{X\}, F, V_T, \tau, S)
\]

with \( X \) a new letter, \( F = \{S, X\} \) if \( S \to_P \epsilon \), \( F = \{X\} \) if not;

\[
\tau = \{(B, a, C) \mid B \to_P aC\} \cup \{(B, a, X) \mid B \to_P a, a \neq \epsilon\}
\]

then \( L_G = L_M \)

- if \( M \) is a FSA take \( G = (Q, \Delta, P, q_0) \) with \( P \) given by

\[
P = \{B \to aC \mid (B, a, C) \in \tau\} \cup \{B \to a \mid (B, a, C) \in \tau, C \in F\}
\]

then \( L_M = L_G \)
pushdown stack automaton (PDA)

\[ M = (Q, F, \mathcal{A}, \Gamma, \tau, q_0, z_0) \]

- \( Q \) finite set of possible states
- \( F \) subset of \( Q \): the final states
- \( \mathcal{A} \) finite set: alphabet
- \( \Gamma \) finite set: stack alphabet
- \( \tau \subset Q \times (\mathcal{A} \cup \{ \epsilon \}) \times \Gamma \times Q \times \Gamma^* \) finite subset: set of transitions
- \( q_0 \in Q \) initial state
- \( z_0 \in \Gamma \) start symbol
• it is a FSA \((Q, F, \mathcal{A}, \tau, q_0)\) together with a stack \(\Gamma^*\)

• the transitions are determined by the first symbol in the stack, the current state, and a letter in \(\mathcal{A} \cup \{\epsilon\}\)

• the transition adds a new (finite) sequence of symbols at the beginning of the stack \(\Gamma^*\)

• a configuration of \(M\) is an element of \(Q \times \mathcal{A}^* \times \Gamma^*\)

• given \((q, a, z, q', \alpha) \in \tau \subset Q \times (\mathcal{A} \cup \{\epsilon\}) \times \Gamma \times Q \times \Gamma^*\) the corresponding transition is from a configuration \((q, aw, z\beta)\) to a configuration \((q', w, \alpha\beta)\)

• computation in \(M\): a chain of transitions \(c \rightarrow c'\) between configurations \(c = c_1, \ldots, c_n = c'\) where each \(c_i \rightarrow c_{i+1}\) a transition as above
a transition labelled \((a, b, c)\) between vertex \(q_i\) and \(q_j\) means read letter \(a\) on string, read letter \(b\) on top of memory stack, remove \(b\) and place \(c\) at the top of the stack: move from configuration \((q_i, aw, b\alpha)\) to configuration \((q_j, w, c\alpha)\)
• computation stops when reach final state or empty stack

• PDA $M$ accepts $w \in \mathbb{A}^*$ by final state if $\exists \gamma \in \Gamma^*$ and $q \in F$ such that $(q_0, w, z_0) \rightarrow (q, \epsilon, \gamma)$ is a computation in $M$

• Language recognized by $M$ by final state

\[ \mathcal{L}_M = \{ w \in \mathbb{A}^* | w \text{ accepted by } M \text{ by final state } \} \]

• $w \in \mathbb{A}^*$ accepted by $M$ by empty stack: if $(q_0, w, z_0) \rightarrow (q, \epsilon, \epsilon)$ is a computation on $M$ with $q \in Q$

• Language recognized by $M$ by empty stack

\[ \mathcal{N}_M = \{ w \in \mathbb{A}^* | w \text{ accepted by } M \text{ by empty stack } \} \]
deterministic PDA

1. at most one transition \((q, a, z, q', \alpha) \in \tau\) with given \((q, a, z)\) source
2. if there is a transition from \((q, \epsilon, z)\) then there is no transition from \((q, a, z)\) with \(a \neq \epsilon\)

First condition as before; second condition avoids choice between a next move that does not read the tape and one that does.

Fact: recognition by final state and by empty stack equivalent for non-deterministic PDA

\[
\mathcal{L} = \mathcal{L}_M \iff \mathcal{L} = \mathcal{N}_{M'}
\]

Not equivalent for deterministic: in deterministic case languages \(\mathcal{L} = \mathcal{N}_M\) have additional property:

Prefix-free: if \(w \in \mathcal{L}\) then no prefix of \(w\) is in \(\mathcal{L}\)
Languages recognized by (non-deterministic) PDA are Type 2 (context-free)

- If $\mathcal{L}$ is context free then $\mathcal{L} = \mathcal{N}_M$ for some PDA $M$

$\mathcal{L} = \mathcal{L}_G$ with $G = (V_N, V_T, P, S)$ context-free, take $M = (\{q\}, \emptyset, V_T, V_N, \tau, q, S)$ with $\tau$ given by the $(q, a, A, q, \gamma)$ for productions $A \rightarrow a\gamma$ in $P$

then for $\alpha \in V_N^*$ and $w \in V_T^*$ have

$$S \xrightarrow{P} w\alpha \iff (q, w, S) \rightarrow_M (q, \epsilon, \alpha)$$

if also $\epsilon \in \mathcal{L}$ add new state $q'$ and new transition $(q, \epsilon, S q', \epsilon)$, where $S$ start symbol of a PDA that recognizes $\mathcal{L} \setminus \{\epsilon\}$
• if $L = \mathcal{N}_M$ for PDA $M$ then $L = \mathcal{L}_G$ with $G$ context-free for $M = (Q, F, \mathcal{A}, \Gamma, \tau, q_0, z_0)$ define $G = (V_N, \mathcal{A}, P, S)$ where

$$V_N = \{(q, z, p) | q, p \in Q, z \in \Gamma\} \cup \{S\}$$

with production rules $P$ given by

1. $S \rightarrow (q_0, z_0, q)$ for all $q \in Q$

2. $(q, z, p) \rightarrow a(q_1, y_1, q_2)(q_2, y_2, q_3) \cdots (q_m, y_m, q_{m+1})$ with $q_1 = q, q_{m+1} = p$ and $(q, a, z, q_1, y_1 \ldots y_m)$ transition of $M$

$$(q, w, z) \rightarrow_M (p, \epsilon, \epsilon) \iff (q, z, p) \rightarrow_P w$$

Similar arguments show Type 0 = recognized by Turing machine; Type 1 (context sensitive) = recognized by “linear bounded automata” (Turing machines but only part of tape can be used)
Representing natural languages?

- **Question**: How good are context-free grammars at representing natural languages?
  - Originally conjectured to be the right class of formal languages to contain natural languages
  - Not always good, but often good (better than earlier criticism indicated)
  - Some explicit examples not context-free (cross-serial subordinate clause in Swiss-German)


Are natural languages context-free?

- Try to show they are not by finding cross-serial dependencies of arbitrarily large size

- Example: the language $L = \{xx^R \mid x \in \{a, b\}^*\}$ has cross serial dependencies of arbitrary length (the $i$-th and $(n+i)$-th term have to be the same ($x^R$ = reversal of $x$)

- if cross serial dependencies of arbitrary length not context-free
The Swiss German Example

Swiss German cross-serial order in dependent clauses

\[ w^a b^m x^c y^d \]

Jan säit das mer (d’chind)\(^n\) (em Hans)\(^m\) es huus haend wele (laa)\(^n\)
(häfte)\(^m\) aastrüche

non-context-free language


- Context-free class too small
- Context-sensitive class too large
- Intermediate candidates:
  1. Tree Adjoining Grammars
  2. Merge Grammars
Other Problem: Clearly there are many more formal languages that do not correspond to natural (human) languages (even within the appropriate class that contains natural languages)

Example: Programming Languages: Fortran is context-free; C is context-sensitive; C++ is Type 0, ...

Examples: Formal Languages constructed from finitely presented discrete groups
Formal Language of a finitely presented group

- Group $G$, with presentation $G = \langle X \mid R \rangle$ (finitely presented)
  - $X$ (finite) set of generators $x_1, \ldots, x_N$
  - $R$ (finite) set of relations: $r \in R$ words in the generators and their inverses

- for $G = \langle X \mid R \rangle$ call $\hat{X} = \{x, x^{-1} \mid x \in X\}$ symmetric set of generators

- Language associated to a finitely presented group $G = \langle X \mid R \rangle$

  \[ \mathcal{L}_G = \{ w \in \hat{X}^* \mid w = 1 \in G \} \]

  set of words in the generators representing trivial element of $G$

- Question: What kind of formal language is it?
• Algebraic properties of the group $G$ correspond to properties of the formal language $\mathcal{L}_G$:

1. $\mathcal{L}_G$ is a regular language (Type 3) iff $G$ is finite (Anisimov)
2. $\mathcal{L}_G$ is context-free (Type 2) iff $G$ has a free subgroup of finite index (Muller–Schupp)

Example: Take $G = \text{SL}_2(\mathbb{Z})$, infinite so $\mathcal{L}_G$ not regular; generators

$$S = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \text{ and } T = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$

with relations $S^2$ and $(ST)^3$

$$\begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix} \text{ and } \begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix}$$

generate a free subgroup $F_2$ of index 12 in $\text{SL}_2(\mathbb{Z})$ (of index 2 in $\Gamma(2)$ that has index 6 in $\text{SL}_2(\mathbb{Z})$) so $\mathcal{L}_{\text{SL}_2(\mathbb{Z})}$ is context-free
The “Boundaries of Babel” Problem

- Given a class of formal languages good enough to contain natural languages
- How to characterize the “region” within this class of formal languages that is populated by actual human (natural) languages?
- What is the geometry of the space of natural languages inside the space of formal languages?


Want: a characterization and parameterization of the syntax of human languages
Broad Question: Is it possible to develop something like the mathematical theory of formal languages for Vision instead of Language?

- Best attempt so far: Pattern Theory