Lecture 3: Mathematics of Formal Languages
Ma 191c: Mathematical Models of Generative Linguistics

Matilde Marcolli

Caltech, Spring 2024

Some References for this lecture:

1. Noam Chomsky, *Three models for the description of language*, IRE Transactions on Information Theory, (1956) N.2, 113–124.
2. Noam Chomsky, *On certain formal properties of grammars*, Information and Control, Vol.2 (1959) N.2, 137–167
3. Ian Chiswell, *A course in formal languages, automata and groups*, Springer, 2009
4. György Révész, *Introduction to formal languages*, McGraw-Hill, 1983

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

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

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

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

$$\mathfrak{A}^+ = \cup_{m \geq 1} \mathfrak{A}^m, \qquad \mathfrak{A}^\star = \cup_{m \geq 0} \mathfrak{A}^m$$

concatenation: $\alpha = a_1 \dots a_m \in \mathfrak{A}^m$, $\beta = b_1 \dots b_k \in \mathfrak{A}^k$

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

associative $(\alpha\beta)\gamma = \alpha(\beta\gamma)$ with $\epsilon\alpha = \alpha\epsilon = \alpha$
semigroup $\mathfrak{A}^+$; monoid $\mathfrak{A}^\star$
Length $\ell(\alpha) = m$ for $\alpha \in \mathfrak{A}^m$

subword: $\gamma \subset \alpha$ if $\alpha = \beta\gamma\delta$ for some other words $\beta, \delta \in \mathfrak{A}^\star$:
prefix β and suffix δ

Language: a subset of $\mathfrak{A}^\star$

Question: how is the subset constructed?

Rewriting system on $\mathfrak{A}$: a subset $\mathcal{R}$ of $\mathfrak{A}^\star \times \mathfrak{A}^\star$
$(\alpha, \beta) \in \mathcal{R}$ means that for any $u, v \in \mathfrak{A}^\star$ the word
$u\alpha v$ *rewrites* to $u\beta v$

Notation: write $\alpha \to_{\mathcal{R}} \beta$ for $(\alpha, \beta) \in \mathcal{R}$
$\mathcal{R}$-derivation: for $u, v \in \mathfrak{A}^\star$ write $u \overset{\bullet}{\to}_{\mathcal{R}} v$ if $\exists$ sequence
$u = u_1, \ldots, u_n = v$ of elements in $\mathfrak{A}^\star$ such that $u_i \to_{\mathcal{R}} u_{i+1}$

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 $\mathcal{G}$:

$$\mathcal{L}_{\mathcal{G}} = \{w \in V_T^\star \mid S \xrightarrow{\bullet}_P w\}$$

language with alphabet V_T

The Chomsky hierarchy

Types:

- Type 0: just a grammar $\mathcal{G}$ as defined above (*unrestricted grammars*)
- Type 1: *context-sensitive grammars*
- Type 2: *context-free grammars*
- Type 3: *regular grammars*, where all productions $A \to aB$ or $A \to a$ 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

We've seen in the previous lecture examples of regular, context free, context sensitive

- Examples of Type 0 but not Type 1 are more difficult to construct
  - assume non-terminals $V_T = \{V_n, n \geq 0\}$
  - alphabet $\{a, b\}$
  - can represent any context-sensitive grammar on this alphabet as a string

    $$x_1 \to y_1;\ x_2 \to y_2; \ldots; x_m \to y_m$$

    of symbols in $\{a, b, ;, \to, V_n\}$
  - encode all these possibilities as binary strings

    $$a \mapsto 010, \quad b \mapsto 0110, \quad ; \mapsto 01110, \quad \to \mapsto 011110, \quad V_n \mapsto 01^{n+5}0$$

  - in set $R = \{w_n = (01^*0)^*\}$ with enumeration by word length plus lexicographic (shortlex)
  - recursive (computable) but not context sensitive language:

    $$\mathcal{L} = \{w_n \in R \text{ encoding context sensitive } \mathcal{G}_n \text{ but } w_n \notin \mathcal{L}(\mathcal{G}_n)\}$$

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)

$M = (Q, F, \mathfrak{A}, \tau, q_0)$

- Q finite set: set of possible states
- F subset of Q: the final states
- $\mathfrak{A}$ finite set: alphabet
- $\tau \subset Q \times \mathfrak{A} \times Q$ set of transitions
- $q_0 \in Q$ initial state

computation in M: sequence $q_0 a_1 q_1 a_2 q_2 \dots a_n q_n$ where $q_{i-1} a_i q_i \in \tau$ for $1 \leq 1 \leq n$

- label of the computation: $a_1 \dots a_n$
- successful computation: $q_n \in F$
- M accepts a string $a_1 \dots a_n$ if there is a successful computation in M labeled by $a_1 \dots a_n$

Language recognized by M:

$$\mathcal{L}_M = \{w \in \mathfrak{A}^\star \mid w \text{ accepted by } M\}$$

Graphical description of FSA

Transition diagram: oriented finite labelled graph Γ 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 $a \in \mathfrak{A}$; label vertex q_0 with $-$ and all final states vertices with $+$

- computations in $M \Leftrightarrow$ paths in Γ 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

deterministic FSA

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

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

determines $\delta : Q \times \mathfrak{A}^\star \to Q$ by $\delta(q, \epsilon) = q$ and
$\delta(q, wa) = \delta(\delta(q, w), a)$ for all $w \in \mathfrak{A}^\star$ and $a \in \mathfrak{A}$

if $q_0 a_1 q_1 \ldots a_n q_n$ 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 $\mathcal{G} = (V_N, V_T, P, S)$ 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;

$$\tau = \{(B, a, C) \,|\, B \to_P aC\} \cup \{(B, a, X) \,|\, B \to_P a,\ a \neq \epsilon\}$$

then $\mathcal{L}_\mathcal{G} = \mathcal{L}_M$

- if M is a FSA take $\mathcal{G} = (Q, \mathfrak{A}, P, q_0)$ with P given by

$$P = \{B \to aC \,|\, (B, a, C) \in \tau\} \cup \{B \to a \,|\, (B, a, C) \in \tau, C \in F\}$$

then $\mathcal{L}_M = \mathcal{L}_\mathcal{G}$

Non-deterministic pushdown stack automaton

Example: some type 2 languages such as $\{0^n 1^n\}$ would require infinite available number of states (e.g. to memorize number of 0's read before the 1's)

Identify a class of infinite automata, where this kind of memory storage can be done

pushdown stack: a pile where new data can be stored on top; can store infinite length, but only last input can be accessed (first in last out)

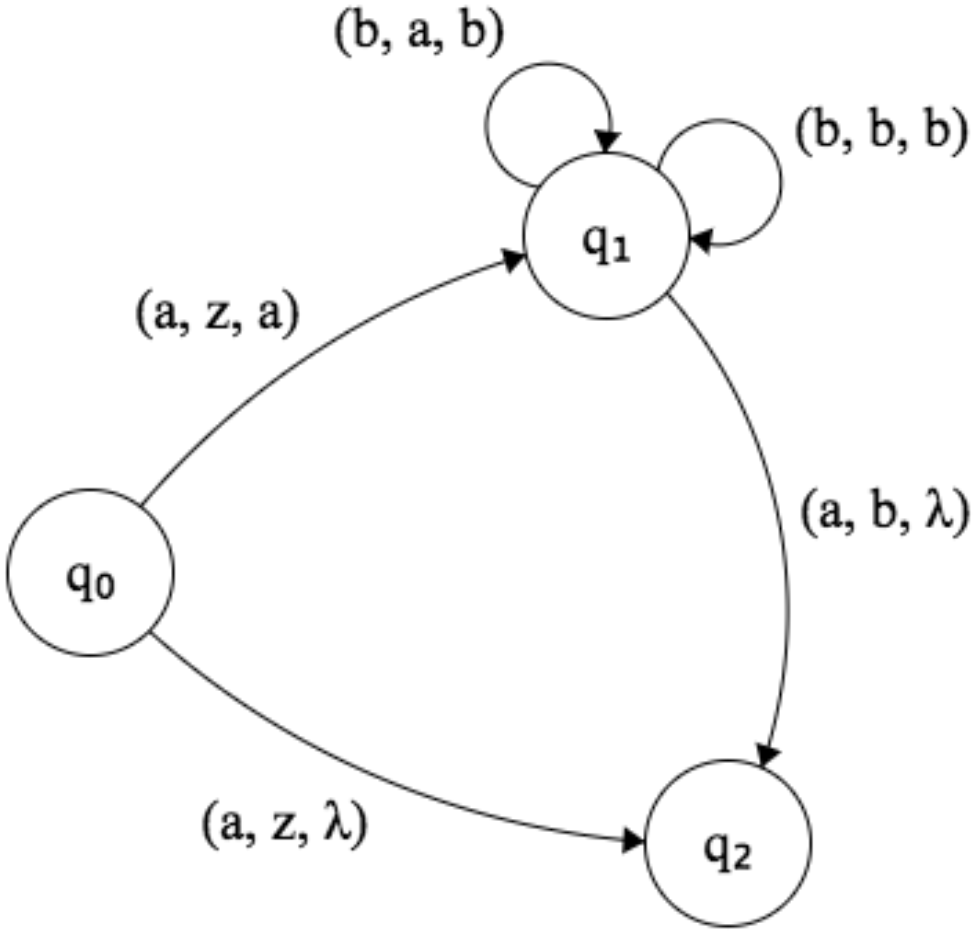
pushdown stack automaton (PDA)

$M = (Q, F, \mathfrak{A}, \Gamma, \tau, q_0, z_0)$

- Q finite set of possible states
- F subset of Q: the final states
- $\mathfrak{A}$ finite set: alphabet
- Γ finite set: *stack alphabet*
- $\tau \subset Q \times (\mathfrak{A} \cup \{\epsilon\}) \times \Gamma \times Q \times \Gamma^\star$ finite subset: set of transitions
- $q_0 \in Q$ initial state
- $z_0 \in \Gamma$ start symbol

- it is a FSA $(Q, F, \mathfrak{A}, \tau, q_0)$ together with a stack $\Gamma^\star$
- the transitions are determined by the first symbol in the stack, the current state, and a letter in $\mathfrak{A} \cup \{\epsilon\}$
- the transition adds a new (finite) sequence of symbols at the beginning of the stack $\Gamma^\star$
- a configuration of M is an element of $Q \times \mathfrak{A}^\star \times \Gamma^\star$
- given $(q, a, z, q', \alpha) \in \tau \subset Q \times (\mathfrak{A} \cup \{\epsilon\}) \times \Gamma \times Q \times \Gamma^\star$ 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 $c \to c'$ between configurations $c = c_1, \ldots, c_n = c'$ where each $c_i \to c_{i+1}$ a transition as above

Example



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 $w \in \mathfrak{A}^\star$ by final state if $\exists \gamma \in \Gamma^\star$ and $q \in F$ such that $(q_0, w, z_0) \to (q, \epsilon, \gamma)$ is a computation in M
- Language recognized by M *by final state*

$$\mathcal{L}_M = \{ w \in \mathfrak{A}^\star \mid w \text{ accepted by } M \text{ by final state } \}$$

- $w \in \mathfrak{A}^\star$ accepted by M *by empty stack*: if $(q_0, w, z_0) \to (q, \epsilon, \epsilon)$ is a computation on M with $q \in Q$
- Language recognized by M *by empty stack*

$$\mathcal{N}_M = \{ w \in \mathfrak{A}^\star \mid w \text{ accepted by } M \text{ by empty stack } \}$$

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, ϵ, 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

$$\mathcal{L} = \mathcal{L}_M \Leftrightarrow \mathcal{L} = \mathcal{N}_{M'}$$

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

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

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

$$S \overset{\bullet}{\to}_P w\alpha \quad \Leftrightarrow \quad (q, w, S) \to_M (q, \epsilon, \alpha)$$

if also $\epsilon \in \mathcal{L}$ add new state q' and new transition $(q, \epsilon, Sq', \epsilon)$, where S start symbol of a PDA that recognizes $\mathcal{L} \setminus \{\epsilon\}$

- if $\mathcal{L} = \mathcal{N}_M$ for PDA M then $\mathcal{L} = \mathcal{L}_\mathcal{G}$ with $\mathcal{G}$ context-free

for $M = (Q, F, \mathfrak{A}, \Gamma, \tau, q_0, z_0)$ define $\mathcal{G} = (V_N, \mathfrak{A}, P, S)$ where

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

with production rules P given by

1. $S \to (q_0, z_0, q)$ for all $q \in Q$
2. $(q, z, p) \to a(q_1, y_1, q_2)(q_2, y_2, q_3) \cdots (q_m, y_m, q_{m+1})$ with $q_1 = q$, $q_{m+1} = p$ and $(q, a, z, q_1, y_1 \ldots y_m)$ transition of M

$$(q, w, z) \to_M (p, \epsilon, \epsilon) \quad \Leftrightarrow \quad (q, z, p) \overset{\bullet}{\to}_P w$$

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)

Turing machine $T = (Q, F, \mathfrak{A}, I, \tau, q_0)$

- Q finite set of possible states
- F subset of Q: the final states
- $\mathfrak{A}$ finite set: alphabet (with a distinguished element B *blank symbol*)
- $I \subset \mathfrak{A} \setminus \{B\}$ input alphabet
- $\tau \subset Q \times \mathfrak{A} \times Q \times \mathfrak{A} \times \{L, R\}$ transitions
  with $\{L, R\}$ a 2-element set
- $q_0 \in Q$ initial state

$qaq'a'L \in \tau$ means T is in state q, reads a on next square in the tape, changes to state q', overwrites the square with new letter a' and moves one square to the left

• *tape description* for T: triple (a, α, β) with $a \in \mathfrak{A}$, $\alpha : \mathbb{N} \to \mathfrak{A}$, $\beta : \mathbb{N} \to \mathfrak{A}$ such that $\alpha(n) = B$ and $\beta(n) = B$ for all but finitely many $n \in \mathbb{N}$ (sequences of letters on tape right and left of a)

• *configuration* of T: (q, a, α, β) with $q \in Q$ and (a, α, β) a tape description

• configuration c' from c in a single move if either

- $c = (q, a, \alpha, \beta)$, $qaq'a'L \in \tau$ and $c' = (q', \beta(0), \alpha', \beta')$ with $\alpha'(0) = a'$ and $\alpha'(n) = \alpha(n-1)$, and $\beta'(n) = \beta(n+1)$
- $c = (q, a, \alpha, \beta)$, $qaq'a'R \in \tau$ and $c' = (q', \alpha(0), \alpha', \beta')$ with $\alpha'(n) = \alpha(n+1)$, and $\beta'(0) = a'$, $\beta'(n) = \beta(n-1)$

• *computation* $c \to c'$ in T starting at c and ending at c': finite sequence $c = c_1, \ldots, c_n = c'$ with c_{i+1} from c_i by a single move

• computation *halts* if c' *terminal configuration*, $c' = (q, a, \alpha, \beta)$ with no element in τ starting with qa

• word $w = a_1 \cdots a_n \in \mathfrak{A}^\star$ *accepted* by T if for $c_w = (q_0, a_1 \cdots a_n)$ there is a computation in T of the form $c_w \to c' = (q, a, \alpha, \beta)$ with $q \in F$

• Language recognized by T

$$\mathcal{L}_T = \{w \in \mathfrak{A}^\star \mid w \text{ is accepted by } T\}$$

• Turing machine T *deterministic* if for given $(q, a) \in Q \times \mathfrak{A}$ there is at most one element of τ starting with qa

Languages of Type 0 are recognized by Turing Machines

- $\mathcal{L} = \mathcal{L}_{\mathcal{G}}$ with $\mathcal{G}$ Type 0 $\Rightarrow \mathcal{L} = \mathcal{L}_{T}$ with $T =$ Turing machine

uses a characterization of Type 0 languages as recursively enumerable languages: code $\mathfrak{A}^{\star}$ by natural numbers $f : \mathfrak{A}^{\star} \to \mathbb{N}$ bijection such that $f(\mathcal{L})$ is a recursively enumerable set (Gödel numbering)

recursively enumerable set: A in $\mathbb{N}$ range $A = g(\mathbb{N})$ of a some *recursive function*: $\exists$ algorithm such that set of inputs on which it halts is A

enumerable set A in $\mathbb{N}$: both A and $\mathbb{N} \setminus A$ are recursively enumerable

recursive function: total functions obtained from primitive recursive (explicit generators and relations), general recursive function also minimization μ

primitive recursive functions

generators

- Successor $s : \mathbb{N} \to \mathbb{N}$, $s(x) = x + 1$;
- Constant $c^n : \mathbb{N}^n \to \mathbb{N}$, $c^n(x) = 1$ (for $n \geq 0$);
- Projection $\pi^n_i : \mathbb{N}^n \to \mathbb{N}$, $\pi^n_i(x) = x_i$ (for $n \geq 1$)

operations

- Composition (substitution) $\mathfrak{c}_{(m,m,p)}$: for $f : \mathbb{N}^m \to \mathbb{N}^n$, $g : \mathbb{N}^n \to \mathbb{N}^p$,

$$g \circ f : \mathbb{N}^m \to \mathbb{N}^p, \quad \mathcal{D}(g \circ f) = f^{-1}(\mathcal{D}(g))$$

- Bracketing (juxtaposition) $\mathfrak{b}_{(k,m,n_i)}$: for $f_i : \mathbb{N}^m \to \mathbb{N}^{n_i}$, $i = 1, \ldots, k$,

$$f = (f_1, \ldots, f_k) : \mathbb{N}^m \to \mathbb{N}^{n_1 + \cdots + n_k}, \quad \mathcal{D}(f) = \mathcal{D}(f_1) \cap \cdots \cap \mathcal{D}(f_k)$$

- Recursion $\mathfrak{r}_n$: for $f : \mathbb{N}^n \to \mathbb{N}$ and $g : \mathbb{N}^{n+2} \to \mathbb{N}$

$$h(x_1, \ldots, x_n, 1) := f(x_1, \ldots, x_n)$$

$$h(x_1, \ldots, x_n, k+1) := g(x_1, \ldots, x_n, k, h(x_1, \ldots, x_n, k)), \quad k \geq 1,$$

where recursively $(x_1, \ldots, x_n, 1) \in \mathcal{D}(h)$ iff $(x_1, \ldots, x_n) \in \mathcal{D}(f)$ and $(x_1, \ldots, x_n, k+1) \in \mathcal{D}(h)$ iff $(x_1, \ldots, x_n, k, h(x_1, \ldots, x_n, k) \in \mathcal{D}(g)$

partial recursive functions

- same three elementary operations $\mathfrak{c}, \mathfrak{b}, \mathfrak{r}$ of composition, bracketing and recursion
- additional μ operation with input $f : \mathbb{N}^{n+1} \to \mathbb{N}$ and output
-

$$h : \mathbb{N}^n \to \mathbb{N}, \quad h(x_1, \ldots, x_n) = \min\{x_{n+1} \,|\, f(x_1, \ldots, x_{n+1}) = 1\}$$

with domain

$$\mathcal{D}(h) = \{(x_1, \ldots, x_n) \,|\, \exists x_{n+1} \geq 1 \,:\, f(x_1, \ldots, x_{n+1}) = 1$$

$$\text{with } (x_1, \ldots, x_n, k) \in \mathcal{D}(f), \forall k \leq x_{n+1}\}$$

Church's thesis: partial recursive functions = semi-computable functions, $\exists$ program that, for $x \in \mathcal{D}(f)$ computes $f(x)$ but can run for an infinite time for $x \notin \mathcal{D}(f)$ (halting problem)

Part 2: Languages recognized by a Turing machine are Type 0

- $\mathcal{L} = \mathcal{L}_{\mathcal{G}}$ of Type 0 $\Leftrightarrow$ $\mathcal{L}$ recursively enumerable
- $\mathcal{L}$ recursively enumerable $\Rightarrow$ recognized by Turing machine

(0) assume $\mathfrak{A} = \{2, 3, \ldots, r-1\}$ and Gödel numbering $w = x_1 \ldots x_k \mapsto \phi(w) = x_1 + x_2 r + \cdots + x_k r^k$

(1) tape alphabet $\{0, 1, 2, \ldots, r-1\}$, input $I = \mathfrak{A}$, final state $F = \emptyset$, blank symbol 0

(2) Turing machine that, on tape description $x_1 \ldots x_k$ halts with tape description $01^{x_1} \cdots 01^{x_k} 0$

(3) Turing machine that, on tape description $01^{x_1} \cdots 01^{x_k} 0$ halts with tape description $01^{\phi(x_1 \ldots x_k)}$

(4) partial recursive function f with $\mathrm{Dom}(f) = \phi(\mathcal{L})$: Turing machine that, on input 01^x halts iff $x \in \mathrm{Dom}(f)$ with $01^{f(x)}$

(5) Composition of these three Turing machines recognizes $\mathcal{L}$

Linear bounded automaton is a Turing machine $T = (Q, F, \mathfrak{A}, I, \tau, q_0)$ where only the part of the tape where the input word is written can be used

1. input alphabet I has two symbols $\rangle, \langle$ right/left end marks
2. no transitions $q\langle q'aL$ or $q\rangle q'aR$ allowed (cannot move past end marks)
3. only transitions starting with $q\langle$ or $q\rangle$ are $q\langle q'\langle R$ and $q\rangle q'\rangle L$ (cannot overwrite $\langle$ and $\rangle$)

Languages recognized by linear bounded automata are Type 1

context-sensitive languages are recursive

- 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

Word problem for G:

- Question: when does a word in the x_j and x_j^{-1} represent the element $1 \in G$?
- When do two words represent the same element?
- Comparing different presentations
- is there an algorithmic solution?

Word problem and formal languages

- 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 $G = \langle X \mid R \rangle$

$$\mathcal{L}_G = \{w \in \hat{X}^\star \mid w = 1 \in G\}$$

set of words in the generators representing trivial element of G

- 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)

- Formal languages and solvability of the word problem:

  - Word problem solvable for G (finitely presented) iff $\mathcal{L}_G$ is a recursive language

Recursive languages (alphabet $\hat{X}$):

- $\mathcal{L}_G$ recursive subset of $\hat{X}^\star$
- equivalently the characteristic function $\chi_{\mathcal{L}_G}$ is a total recursive function
- Total recursive functions are computable by a Turing machine that always halts
- For a recursive language there is a Turing machine that always halts on an input $w \in \hat{X}^\star$: accepts it if $w \in \mathcal{L}_G$, rejects it of $w \notin \mathcal{L}_G$: so word problem is (algorithmically) solvable

Finitely presented groups with unsolvable word problem (Novikov)

• Group G with recursively enumerable presentation: $G = \langle X \mid R \rangle$ with X finite and R recursively enumerable

• Group is recursively presented iff it can be embedded in a finitely presented group (X and R finite)

• Example of recursively presented G with unsolvable word problem

$$G = \langle a, b, c, d \mid a^n b a^n = c^n d c^n, \ n \in A \rangle$$

for A recursively enumerable subset $A \subset \mathbb{N}$ that has unsolvable membership problem

• If recursively presented G has unsolvable word problem and embeds into finitely presented H then H also has unsolvable word problem.

Example: finite presentation with unsolvable word problem

- Generators: $X = \{a, b, c, d, e, p, q, r, t, k\}$
- Relations:

$$p^{10}a = ap, \quad p^{10}b = bp, \quad p^{10}c = cp, \quad p^{10}d = dp, \quad p^{10}e = ep$$

$$aq^{10} = qa, \quad bq^{10} = qb, \quad cq^{10} = qc, \quad dq^{10} = qd, \quad eq^{10} = qe$$

$$ra = ar, \quad rb = br, \quad rc = cr, \quad rd = dr, \quad re = er, \quad pt = tp, \quad qt = tq$$

$$pacqr = rpcaq, \quad p^2adq^2r = rp^2daq^2, \quad p^3bcq^3r = rp^3cbq^3$$

$$p^4bdq^4r = rp^4dbq^4, \quad p^5ceq^5r = rp^5ecaq^5, \quad p^6deq^6r = rp^6edbq^6$$

$$p^7cdcq^7r = rp^7cdceq^7, \quad p^8ca^3q^8r = rp^8a^3q^8, \quad p^9da^3q^9r = rp^9a^3q^9$$

$$a^{-3}ta^3k = ka^{-3}ta^3$$

How are such examples constructed?

A technique to construct semigroup presentations with unsolvable word problem:

• G.S. Cijtin, *An associative calculus with an insoluble problem of equivalence*, Trudy Mat. Inst. Steklov, vol. 52 (1957) 172–189

A technique for passing from a semigroup with unsolvable word problem to a group with unsolvable word problem

• V.V. Borisov, *Simple examples of groups with unsolvable word problems*, Mat. Zametki 6 (1969) 521–532

Example above: method applied to simplest known semigroup example

• D.J. Collins, *A simple presentation of a group with unsolvable word problem*, Illinois Journal of Mathematics 30 (1986) N.2, 230–234

Regular language $\Leftrightarrow$ finite group

- If G finite, use standard presentation

$G = \langle x_g,\ g \in G \,|\, x_g x_h = x_{gh} \rangle$

Construct FSA $M = (Q, F, \mathfrak{A}, \tau, q_0)$ with $Q = \{x_g \,|\, g \in G\}$,
$\mathfrak{A} = \{x_g^{\pm 1} \,|\, g \in G\}$, $q_0 = x_1$, $F = \{q_0\}$ and transitions τ given by

$$(x_g, x_h, x_{gh}), \quad g, h \in G$$

$$(x_g, x_h^{-1}, x_{gh^{-1}}), \quad g, h \in G$$

The finite state automaton M recognizes $\mathcal{L}_G$

- If G is infinite and X is a finite set of generators for G

For any $n \geq 1$ there is a $g \in G$ such that g not obtained from any word of length $\leq n$ (only finitely many such words and G is infinite)

If M deterministic FSA with alphabet $\hat{X}$ and $n = \#Q$ number of states, take $g \in G$ not represented by any word of length $\leq n$

then there are prefixes w_1 and $w_1 w_2$ of w such that, after reading w_1 and $w_1 w_2$ obtain same state

so M accepts (or rejects) both $w_1 w_1^{-1}$ and $w_1 w_2 w_1^{-1}$ but first is 1 and second is not ($w_2 \neq 1$)

so M cannot recognize $\mathcal{L}_G$

Cayley graph

- Vertices $V(\mathcal{G}_G) = G$ elements of the group
- Edges $E(\mathcal{G}_G) = G \times X$ with edge $e_{g,x}$ oriented with $s(e_{g,x}) = g$ and $t(e_{g,x}) = gx$
- for $x^{-1} \in \hat{X}$ edge with opposite orientation $e_{g,x^{-1}} = \bar{e}_{g,x}$ with $s(e_{g,x^{-1}}) = gx$ and $t(e_{g,x^{-1}}) = gx\,x^{-1} = g$
- word w in the generators $\Rightarrow$ oriented path in $\mathcal{G}_G$ from g to gw
- word $w = 1 \in G$ iff corresponding path in $\mathcal{G}_G$ is closed
- G acts on $\mathcal{G}_G$: acting on $V(\mathcal{G}_G) = G$ and on $E(\mathcal{G}_G) = G \times X$ by left multiplication (translation)
- invariant metric: $d(g, h) =$ minimal length of path from vertex g to vertex h, with $d(ag, ah) = d(g, h)$ for all $a \in G$

Main idea for the context-free case

- X set of generators of G
- if for $y_i \in \hat{X}$, a word $w = y_1 \cdots y_n = 1$ get closed path in the Cayley graph $\mathcal{G}_G$
- consider a polygon $\mathcal{P}$ with boundary this closed path
- obtain a characterization of the context-free property of $\mathcal{L}_G$ in terms of properties of triangulations of this polygon

Plane polygons and triangulations

• a plane polygon $\mathcal{P}$: interior of a simple closed curve given by a finite collections of (smooth) arcs in the plane joined at the endpoints

• triangulation of $\mathcal{P}$: decomposition into triangles (with sides that are arcs): two triangles can meet in a vertex or an edge (or not meet)

• allow 1-gons and 2-gons (as "triangulated")

• triangle in a triangulation is *critical* if has two edges on the boundary of the polygon

• triangulation is *diagonal* if no more vertices than original ones of the polygon

• Combinatorial fact: a diagonal triangulation has at least two critical triangles (for $\mathcal{P}$ with at least two triangles)

K-triangulations

- diagonal triangulation of a polygon $\mathcal{P}$ with boundary a closed path in the Cayley graph $\mathcal{G}_G$
- each edge of the triangulation is labelled by a word in $\hat{X}^\star$
- going around the boundary of each triangle gives a word in $\mathcal{L}_G$ (a word w in $\hat{X}^\star$ with $w = 1 \in G$)
- all words labeling edges of the triangulation have length $\leq K$

Context-free and K-triangulations

Language $\mathcal{L}_G$ is context-free $\Leftrightarrow \exists K$ such that all closed paths in Cayley graph $\mathcal{G}_G$ can be triangulated with a K-triangulation

Idea of argument:

If context-free grammar:

• use production rules for word $w = 1$ (boundary of polygon) to produce a triangulation:

$$S \to AB \overset{\bullet}{\to} w_1 w_2 = w \quad \text{with } A \overset{\bullet}{\to} w_1 \text{ and } B \overset{\bullet}{\to} w_2$$

$\Rightarrow$ a subdivision of polygon in to two arcs: draw an arc in the middle, etc.

If have K-triangulation for all loops in $\mathcal{G}_G$: get a context-free grammar with terminals $\hat{X}$

• for each word $u \in \hat{X}^\star$ of length $\leq K$ variable A_u and for $u = vw$ in G production $A_u \to A_v A_w$ in P

• any word $w = y_1 \cdots y_n$ from boundary of triangles in the triangulation also corresponds to $A_1 \overset{\bullet}{\to} A_{y_1} \cdots A_{y_n}$ in the grammar (inductive argument eliminating the critical triangles and reducing size of polygon)

• and productions $A_y \to y$ (terminals); get that the grammar recognizes $\mathcal{L}_G$

accessibility

To link contex-free to the existence of a free subgroup, need a decomposition of the group that preserves both the context-free property and the existence of a free subgroup, so that can do an inductive argument

- HNN-extensions: two subgroups B, C in a group A and an isomorphism $\gamma : B \to C$ (not coming from A)

$$A \star_C B = \langle t, A \,|\, tBt^{-1} = C \rangle$$

means generators as A, additional generator t; relations of A and additional relations $tbt^{-1} = \gamma(b)$ for $b \in B$

- *accessibility series*: (accessibility length n)

$$G = G_0 \supset G_1 \supset \cdots \supset G_n$$

G_i subgroups with $G_i = G_{i+1} \star_K H$ with K finite

- finitely generated G is *accessible* if upper bound on length of any accessibility series (least upper bound = accessibility length)
- assume G context-free and accessible
- inductive argument (induction on accessibility length) on existence of a free finite-index subgroup:
if $n = 0$ have G finite group; if $n > 0$ $G = G_1 \star_K H$, context-free property inherited; inductively: free finite-index subgroup for G_1; show implies free finite-index subgroup for G
- then need to eliminate auxiliary accessibility condition

Context-free $\Leftrightarrow$ free subgroup of finite index

- show that a finitely generated G with $\mathcal{L}_G$ context-free is finitely presented
- then show finitely presented groups are accessible
- Conclusion: equivalent properties for finitely generated G
  1. $\mathcal{L}_G$ is a context-free language
  2. G has a free subgroup of finite index
  3. G has deterministic word problem (using the fact that free groups do)

Word problem and geometry

• Groups given by explicit presentations arise in geometry/topology as fundamental groups $\pi_1(X)$ of manifolds

Positive results

• Groups with solvable word problem include: negatively curved groups (Gromov hyperbolic), Coxeter groups (reflection groups), braid groups, geometrically finite groups
[all in a larger class of "automatic groups"]

Negative results

• Any finitely presenting group occurs as the fundamental group of a smooth 4-dimensional manifold

• The homeomorphism problem is unsolvable

- A. Markov, *The insolubility of the problem of homeomorphy*, Dokl. Akad. Nauk SSSR 121 (1958) 218–220

Additional References:

1. S.P. Novikov, *On the algorithmic unsolvability of the word problem in group theory*, Proceedings of the Steklov Institute of Mathematics 44 (1955) 1–143
2. V.V. Borisov, *Simple examples of groups with unsolvable word problems*, Mat. Zametki 6 (1969) 521–532
3. A.V. Anisimov, *The group languages*, Kibernetika (Kiev) 1971, no. 4, 18–24
4. D.E. Muller, P.E. Schupp, *Groups, the theory of ends, and context-free languages*, J. Comput. System Sci. 26 (1983), no. 3, 295–310

Tree Adjoining Grammar (Aravind Joshi, 1969)

- developed as formal languages (as a generalization of context-free grammars)
- rooted trees with a marked foot leaf node (a word); *basic trees* and *auxiliary trees* (these have same symbol labeling root and foot)
- two operations: *substitution* (leaf/root grafting) and *adjunction* (insertion of an auxiliary tree at an internal node labelled by auxiliary root/foot label)
- main idea: these two operations should suffice to describe all syntactic dependencies
- LTAG: lexicalized tree-adjoining grammar: each elementary tree associated with an item in a lexical database (XTAG project, LTAG parser)

Tree Adjoining Grammars (Joshi, Levy, Takahashi)

Mathematical model for structural composition of parse trees: instead of production rules that rewrite strings as in the formal languages grammars, use a system of trees with tree rewriting rules

- a (finite) set of *Elementary Trees*
- *Substitution rule*: graft a terminal leaf of a tree T to the root of another tree
- *Adjoining rule*: at an internal vertex of the tree labelled by X attach a tree with root labelled by X and with one of the leaves also labelled by X with anything outgoing from original tree at X then attached to the X-labelled leaf of the inserted tree.

Note: mathematically the first is an *operad* structure the second is a *Lie algebra* structure, we'll discuss these later

Note: no additional transformations used other than substitution and adjoining

Fundamental assumptions of TAG:

• all syntactic dependencies are encoded (locally) in the elementary trees

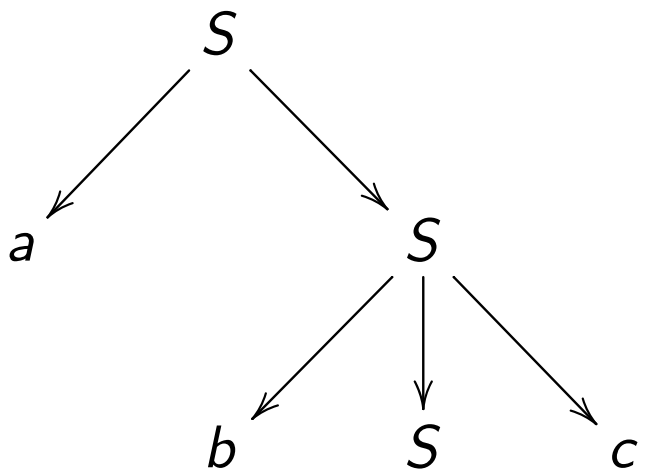
• non-local dependencies must be reducible to local ones (after contracting a certain number of adjoined trees)

TAG derivation: a combination of elementary trees via a sequence of substitutions and adjoining

Derivation structure: a tree whose vertices are labelled by elementary trees and daughter vertices of a given node T are the elementary trees that are substituted or adjoined into the tree T (requires "independence" of the operations performed)

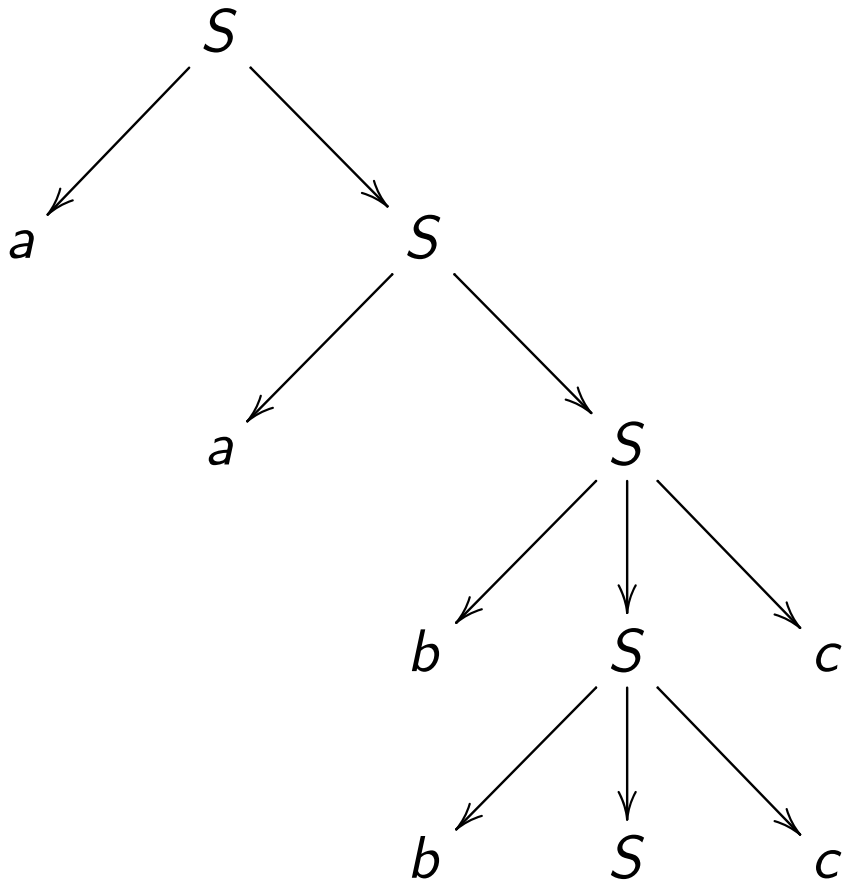
Generative power of TAG:

- All context-free languages can be generated by a TAG
- $\mathcal{L} = \{a^n b^n c^n \mid n \in \mathbb{N}\}$ not generated by a context-free grammar, but can be generated by a TAG



repeatedly adjoin copies of this elementary tree into itself at the S vertex with the first b daughter

$a^2b^2c^2$ from first adjoining, etc.



But... simple examples of context-sensitive languages that cannot be generated by TAG's: (Vijay–Shanker)

$$\mathcal{L} = \{a^n b^n c^n d^n e^n \mid n \in \mathbb{N}\}$$

Additional References

- J.E. Hopcroft, J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison–Wesley, 1979
- A.K. Joshi, L. Levy, M. Takahashi, *The tree adjunct grammars*, Journal of the Computer and System Sciences, 10 (1975) 136–163

Coming up next

- from formal languages to transformational grammars
- more details on transformational grammar
- more details on earlier versions of Minimalism