Formal Languages

Matilde Marcolli
CS101: Mathematical and Computational Linguistics

Winter 2015

A very general abstract setting to describe 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 $\alpha \rightarrow_{\mathcal{R}} \beta$ for $(\alpha, \beta) \in \mathcal{R}$
$\mathcal{R}$-derivation: for $u, v \in \mathfrak{A}^\star$ write $u \overset{\bullet}{\rightarrow}_{\mathcal{R}} v$ if $\exists$ sequence
$u = u_1, \ldots, u_n = v$ of elements in $\mathfrak{A}^\star$ 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 $\mathcal{G}$:

$$\mathcal{L}_{\mathcal{G}} = \{ w \in V_T^\star \,|\, S \overset{\bullet}{\to}_P w \}$$

language with alphabet V_T

Context free and context sensitive production rules

- context free: $A \to \alpha$ with $A \in V_N$ and $\alpha \in (V_N \cup V_T)^\star$
- context sensitive: $\beta A \gamma \to \beta \alpha \gamma$ with $A \in V_N$
$\alpha, \beta, \gamma \in (V_N \cup V_T)^\star$ 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 $\mathcal{G}$ 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

Examples

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

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

then $\mathcal{L}_{\mathcal{G}} = \{0^m 1^n \mid m \geq 0, n \geq 1\}$

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

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

then $\mathcal{L}_{\mathcal{G}} = \{0^n 1^n \mid n \geq 1\}$

- Type 1: $\mathcal{G} = (\{S, B, C\}\{a, b, c\}, P, S)$ 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 $\mathcal{L}_{\mathcal{G}} = \{a^n b^n c^n \mid n \geq 1\}$

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: type 2 language 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 $(q, w, z\beta)$ to a configuration $(q', wa, \alpha\beta)$

• computation in M: a chain of transitions $c \to c'$ between configurations $c = c_1, \dots, c_n = c'$ where each $c_i \to c_{i+1}$ a transition as above

- 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*

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

- $w \in \mathfrak{A}^\star$ 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*

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

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

- 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 $\mathcal{L} \smallsetminus \{\epsilon\}$

- 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

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

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*)
- $I \subset \mathfrak{A} \smallsetminus \{B\}$ 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

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

- 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 recognized by Turing Machines are Type 0

- if $\mathcal{L} = \mathcal{L}_T$ take grammar $\mathcal{G} = (V_N, V_T, P, S)$ with $V_T = I$,

$$V_N = ((I \cup \{\epsilon\}) \times \mathfrak{A}) \cup Q \cup \{S, E_1, E_2, E_3\}$$

extra letters E_1, E_2, E_3 and productions P

$$S \to E_1 E_2, \quad E_2 \to (a, a) E_2, \;\; a \in \mathfrak{A}, \quad E_2 \to E_3$$

$$E_3 \to (\epsilon, B) E_3, \quad E_1 \to (\epsilon, B) E_1, \quad E_3 \to \epsilon, \quad E_1 \to q_0$$

$$q(a, C) \to (a, D)p, \;\; \text{with } qCpDR \in \tau, \;\; a \in I \cup \{\epsilon\}$$

$$(a, C)q \to p(a, D), \;\; \text{with } qCpDL \in \tau, \;\; a \in I \cup \{\epsilon\}$$

$$(a, C)q \to qaq, \quad q(a, C) \to qaq, \quad q \to \epsilon,$$

for $a \in I \cup \{\epsilon\}$, $C \in \mathfrak{A}$, $q \in F$.
Then $\mathcal{L} = \mathcal{L}_{\mathcal{G}}$

- converse statement: $\mathcal{L} = \mathcal{L}_{\mathcal{G}}$ 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*

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

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

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

Some references:

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