

Basic Notation (Begin Week 1 Notes)

- ϵ Empty string (zero length)
- Σ Alphabet (e.g. $\{0, 1\}$ or $\{a, b, c, \dots, z\}$)
- Σ^k Set of all strings of length k consisting of letters found in Σ
- $\Sigma^0 = \{\epsilon\}$
- $\Sigma^1 = \Sigma$
- Σ^* Set of strings which can be formed from the alphabet Σ (i.e. $\bigcup_{i=0}^{\infty} \Sigma^i$)
- L Set of strings in a language ($L \subseteq \Sigma^*$)
- $|w|$ Length of string w

Deterministic Finite Automata (DFA) Notation

- $M = (Q, \Sigma, \delta, q_0, F)$
...where...
- Q set of states
- Σ alphabet
- δ set of state transitions, in the form $\delta(q_1, a) = q_2$
- q_0 initial state
- F set of final states

Non-Deterministic Finite Automata (NFA) Notation

- $M = (Q, \Sigma, \delta, q_0, F)$
- Q set of states
- Σ alphabet
- δ set of state transitions, in the form $\delta(q_1, a) = \{q_2, q_3\}$
- q_0 initial state
- F set of final states

Extended Transition Functions

- $\hat{\delta}(q, w)$ set of states that can be reached starting at q and processing w
- $\hat{\delta}(q, \epsilon) = \{q\}$
- $\hat{\delta}(q_0, w) \in F$ iff w is in the language L
- $L(N) = \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \cap F \neq \emptyset\}$

ϵ -NFAs (Begin Week 2 Notes)

- Same as NFAs, except " ϵ -moves" are allowed (e.g. $\delta(q_1, \epsilon) = \{q_2, q_3, q_4\}$)
- ϵ close set of state reachable from q using only ϵ -moves

Regular Expression Notation

| | |
|----------------|---|
| $L(\epsilon)$ | $= \{\epsilon\}$ |
| $L(\emptyset)$ | $= \{\}$ |
| $L(a)$ | $= \{a\}, \forall a \in \Sigma$ |
| $L(R + S)$ | $= L(R) \cup L(S)$ (union) |
| $L(R \cdot S)$ | $= L(R) \cdot L(S) = \{w = xy \mid x \in L_1, y \in L_2\}$ (concatenation) |
| L^k | $= \{w \mid w = x_1x_2x_3 \dots x_k \text{ such that } \forall i, x_i \in L\}$ |
| L^* | $= L^0 \cup L^1 \cup L^2 \cup \dots$ |
| L^+ | $= LL^* = L^1 \cup L^2 \cup L^3 \cup \dots$ |
| Precedence: | (highest) $*$ $>$ \cdot $>$ $+$ (lowest) |
| L_{ij} | $= \{w \mid \hat{\delta}(q_i, w) = q_j\}$ (strings taking M from q_i to q_j) $= L(M_{ij})$ where $M_{ij} = (Q, \Sigma, \delta, q_i, \{q_j\})$ |
| L_{ij}^k | strings taking M from q_i to q_j without passing <i>through</i> and state $q_{\ell > k}$ |

Pumping Lemma (Begin Week 3 Notes)

If L is regular, then $\forall n, \forall w \in L$ with $|w| \geq n, \exists$ a decomposition $w = xyz$ such that $|y| \geq 1, |xy| \leq n$, and $\forall k \geq 0, xy^kz \in L$.

Regular Language Closure Properties

If L_1 and L_2 are regular, then so are the following:

| | |
|------------------|---------------------------------------|
| $L_1 \cup L_2$ | (union) |
| $L_1 \cdot L_2$ | (concatenation) |
| $\overline{L_1}$ | $= \Sigma^* - L_1$ (complement) |
| $L_1 \cap L_2$ | (intersection) |
| L_1^R | $= \{w^R \mid w \in L_1\}$ (reversal) |

Decision Problems

Able to test for:

| | |
|-------------|------------------|
| Emptiness | $L = \emptyset?$ |
| Membership | $w \in L?$ |
| Equality | $L_1 = L_2?$ |
| Fininteness | $ L $ finite? |

Context-Free Grammars (Begin Week 4 Notes)

| | |
|-----|---|
| G | $= (V, T, P, S)$...where... |
| T | set of terminals (basically, $T = \Sigma$) |
| V | set of variables (non-terminal symbols) |
| S | start symbol ($S \in V$) |
| P | set of production rules, (e.g. $S \rightarrow \epsilon \mid 0A \mid 1B$) |

Example: $G_{Eq} = (V, T, P, S)$ where

| | |
|-----|--|
| T | $\{0, 1\}$ |
| V | $\{S, A, B\}$ |
| P | $\{S \rightarrow \epsilon \mid 0A \mid 1B, A \rightarrow 1S \mid 0AA, B \rightarrow 0S \mid 1BB\}$ |

Derivations

For CFG G , $\forall \alpha, \beta, \gamma \in (V \cup T)^*$, $\forall A \in V$
we say $\alpha A \beta \Rightarrow \alpha \gamma \beta$ (directly derives)
if the rules $A \rightarrow \gamma$ is in G .

$\alpha_1 \Rightarrow \alpha_2 \Rightarrow \alpha_3 \Rightarrow \dots \Rightarrow \alpha_r$ implies $\alpha \xRightarrow{*} \alpha_r$

Reflexive Property: $\alpha \xRightarrow{*} \alpha$

Transitive Property: $\alpha \xRightarrow{*} \beta$ and $\beta \xRightarrow{*} \gamma$ implies $\alpha \xRightarrow{*} \gamma$

Push-Down Automata (PDAs) (Begin Week 5 Notes)

$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$
...where...

Q set of states

Σ input alphabet

Γ stack alphabet

q_0 start state

Z_0 start stack symbol ($Z_0 \in \Gamma$)

F final states ($F \subseteq Q$)

Example transition $\delta(q, a, X) = \{(p_1, \alpha_1), (p_2, \alpha_2), \dots\}$

$q \in Q$

$p_i \in Q$

$a \in \Sigma$ or $a = \epsilon$

$X \in \Gamma$

$\alpha_i \in \Gamma^*$

Instantaneous Description (ID) $\langle q, x, \alpha \rangle$

$q \in Q$ (current state)

$X \in \Sigma^*$ (unread input)

$\alpha_i \in \Gamma^*$ (stack contents—top of stack to left, bottom to right)

$(p_i, \alpha_i) \in \delta(q, a, x)$ allows the following transition: $\langle q, ax, X\beta \rangle \vdash \langle p_i, x, \alpha_i\beta \rangle$

Accepts if at least one execution trace which leads to a final state when input-end is reached.

Empty Stack Language $N(M) = \left\{ w \in \Sigma^* \mid \langle q_0, w, Z_0 \rangle \vdash^* \langle p, \epsilon, \epsilon \rangle \text{ for any } p \right\}$

Deterministic Push-Down Automata (DPDAs)

- NOT as powerful as PDAs (example: no DPDA for $L = \{ww^R \mid w \in \Sigma^*\}$)
- Deterministic Context Free Grammars (DCFLs) is the class of languages accepted by DPDAs
- Regular \subset DCFL \subset CFL
- DCFLs are always unambiguous

Simplifying Grammars

Useful symbols are those such that for any $X \in V \cup T$, $S \xRightarrow{*} \alpha X \beta \xRightarrow{*} w$ (i.e. they are *generating* and *reachable*). Can simplify by removing non-generating symbols, then unreachable symbols.

ϵ -Productions are any that can produce ϵ . $X \in V$ is *nullable* if $X \xRightarrow{*} \epsilon$. ϵ -productions can be removed to simplify a grammar. This has the "glitch" of changing the grammar if S is nullable.

Pumping Lemma for CFLs (Begin Week 5(b) Notes)

$\exists m$ such that $\forall z \in L$ with $|z| > m$, \exists a decomposition $z = uvwxy$ such that $|vwx| \leq m$, $|vx| > 0$ and $\forall i \geq 0, uv^iwx^iy \in L$.

CFL Closure Properties (Begin Week 6 Notes)

If L_1 and L_2 are CFLs, then so are the following:

$L_1 \cup L_2$ (union)

$L_1.L_2$ (concatenation)

L_1^*

$L_1^R = \{w^R \mid w \in L\}$ (reversal)

If L_1 and L_2 are CFLs, then the following *may or may not be*:

$\overline{L_1} = \Sigma^* - L_1$ (complement)

$L_1 \cap L_2$ (intersection)

HELLO-WORLD Reductions

The HELLO-WORLD problem is proven to be *undecidable*. Another problem can be proven to be undecidable by showing that if it was decidable, it would be used to create a decidable HELLO-WORLD program.

Turing Machine Notation

$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$

...where...

Q set of states

Σ input alphabet

Γ tape alphabet (note $\Sigma \subseteq \Gamma$ because input starts on tape)

δ set of state transitions, of form $\delta(q, x) = (p, Y, L)$

q_0 initial state

B Blank symbol (note $B \in \Sigma - \epsilon$)

F set of final states

Instantaneous Description (ID) $\alpha_1 q \alpha_2$

$\alpha_1 \in \Gamma^*$ (string to left of position, unnecessary blanks omitted)

$q \in Q$ (current state)

$\alpha_2 \in \Gamma^*$ (symbol at position followed by string to right of position, unnecessary blanks omitted)

Given DTM M , $L(M) = \{w \mid q_0 w \vdash^* \alpha_1 p \alpha_2, \text{ where } p \in F \text{ and } \alpha_1, \alpha_2 \in \Gamma^*\}$.

World of Languages (Begin Week 7 Notes)

| Generality | Name | Example | Defining Machines |
|------------|--------------------------------|---------------|--|
| Lowest | Regular | 0^n | DFA, NFA, ϵ -NFA, RE |
| | N/A | $0^n 1^n$ | NPDA |
| | Context-Free | ww^R | CFL, (nondeterministic) PDA |
| | Recursive (P and NP fall here) | $0^n 1^n 2^n$ | Algorithms (i.e. <i>decidable</i> TMs) |
| | Recursively Enumerable | L_u | Procedures (i.e. TMs) |
| Highest | EVERYTHING | L_d | N/A |

If L is recursive, then \bar{L} is recursive. Also, if L and \bar{L} are recursively enumerable, then both are also recursive (because a decidable TM can be created from an undecidable one and its complement).

Reductions

Definition- L_1 reduces to L_2 (denoted $L_1 < L_2$) if there exists a function f (called the reduction) such that:

- Some turing machine M_f computes f by taking as input a string w and halting with string $f(w)$ on its tape
- f is such that $w \in L_1 \Leftrightarrow f(w) \in L_2$