Final

Finite Automata And Regular Expression

DFA: $M = (K, \Sigma, \delta, s, F)$, $\delta = \{(q, w)\}$.

NFA: $M = (K, \Sigma, \Delta, s, F)$, $\Delta = \{(q, w)\}$.

Regular expression to NFA: Naturally.

NFA to DFA:

1. List $\{E(q) \mid q \in K\}$
2. List $\delta(s')$, $\delta(Q, a) = \bigcup\{E(p) \mid p \in K \ and \ (q, a, p) \in \Delta \ for \ some \ q \in Q\}$
3. **Do not forget to continue listing on $\delta(\emptyset, \sigma) = \emptyset$, and draw the lines on diagram back to itself!**

DFA to regular expression: By eliminating states one by one.

Closure of regular expression: Union, concatenation, Kleene star, complementation, intersection ($L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$) and difference.

Prove that a language is regular:

- Accepted by FA.
- Specified by regular expression.
- Closure. (Union, concatenation, Kleene star, complementation, intersection, and difference)

Prove that a language is not regular:

- Intuitive: FA has only finite states so it can only remember a finite number of strings.
- Pumping theorem. ($\{a^n b^n \mid n \geq 0\}$, $\{a^n \mid n \text{ is a prime}\}$)
- Is not closed under intersection or complementation. ($\{w \in \{a, b\}^* \mid w \text{ has the same number of } a\text{'s and } b\text{'s}\} \cap a^* b^*$)

Pumping theorem:

Let L be a regular language, then there exists an integer $n \geq 1$ such that any string $w \in L$ with $|w| \geq n$ can be written as $w = xyz$ such that:

- $y \neq e$
- $|xy| \leq n$
- $\forall i \geq 0, xy^i z \in L$

Prove that a language is not regular by pumping theorem:

- Let L be the proposed regular language.
- There is some n, by the pumping lemma.

- Choose a string s, longer than n symbols, in the language L.
- Using the pumping lemma, construct a new string s' that is not in the language.

Caveats:

Arbitrary union of regular languages can be irregular: Any language can be written as union of all its individual elements, but not all languages are regular.

Regular because of equivalence to Σ^*: $\{xyx^R \mid x, y \in \Sigma^*\}$ is regular: Let x be e, then $L = \Sigma^*$.

Pushdown Automata And Context-Free Grammar

CFG: $G = (V, \Sigma, R, S)$, $R = \{A \to u\}$.

$\text{Regular language} \subsetneq CFG$.

PDA: $M = (K, \Sigma, \Gamma, \Delta, s, F)$, $\Delta = \{((p, a, \beta), (q, \gamma))\}$.

CFL to CFG: Construct manually.

Examples:

- $\{w \in \{a, b\}^* \mid w \text{ has the same number of } a \text{ 's and } b \text{ 's}\}$:
  $S \to e, S \to aSb, S \to bSa, S \to SS$.
- $\{a^m b^n \mid m \geq n\}$: $S \to e, S \to aS, S \to aSb$.
- $\{a^m b^n \mid m > n\}$: $S \to aS_1, S_1 \to e, S_1 \to aS_1, S_1 \to aS_1 b$.
- $\{a^m b^n c^p d^q \mid m + n = p + q\}$: $m + n = p + q = N$,
  $a^m b^n c^p d^q = a^q a^{m-q} b^{N-m} c^{N-m} c^{m-q} d^q$ when $m \geq q$, so $$; $m < q$ is similar.

CFL to PDA:

Examples:

- $\{ww^R \mid w \in \{a, b\}^*\}$: Two states, push and pop the stack.
- $\{w \in \{a, b\}^* \mid w \text{ has the same number of } a \text{ 's and } b \text{ 's}\}$: Guard with a stack bottom symbol.

CFG to PDA:

- CFG $G = (V, \Sigma, R, S)$, PDA $M = (K, \Sigma, \Gamma, \Delta, s, F)$.
- $K = \{p, q\}$.
- $\Gamma = V$.
- $s = p$.
- $F = \{q\}$.
- Δ contains the following transitions:
  - $((p, e, e), (q, S))$.
  - $((q, e, A), (q, x))$ for each rule $A \to x \in R$. (Generate)

- $((q, a, a), (q, e)), \forall a \in \Sigma$. (Match)

(Not mentioned) PDA to simple PDA, simple PDA to CFG.

Closure of CFG: Union, concatenation and Kleene star, but not complementation, intersection or difference.

Prove that a language is context-free:

- Accepted by PDA.
- Closure. (Union, concatenation, Kleene star, but not complementation, intersection or difference)
- Intersection of a CFL with a regular language is CFL.
- (So) any CFL over a single-letter alphabet is regular.

Prove that a language is not context-free:

- Pumping theorem. ($\{a^n b^n c^n \mid n \geq 0\}$)
- Is not closed under intersection with a regular language. ($\{w \in \{a, b, c\}^* \mid w \text{ has the same number of } a\text{'s} \, b\text{'s and } c\text{'s}\} \cap a^* b^* c^*$)

Pumping Theorem: Let $G = (V, \Sigma, R, S)$ be a CFG, then any string $w \in L(G)$ of length greater than $n \geq \phi(G)^{|V - \Sigma|}$ can be rewritten as $w = uvxyz$ such that:

- $vy \neq e$.
- $|vxy| \leq n$
- $\forall i \geq 0, uv^i xy^i z \in L(G)$.

Proving that a language is not CFL:

- There is some n, by the pumping theorem.
- Choose a string w longer than n symbols in language L.
- Use the pumping theorem, construct w' that is not in L.

Caveats:

Prove intersection is not free by $\{a^n b^n c^n\} = \{a^n b^n c^m\} \cap \{a^m b^n c^n\}$.

$\{a^m b^* c^n \mid m = (\text{or } \neq) n + k\}$ is context-free.

$\{ww \mid w \in \Sigma^*\}$ is not context-free.

$\{www \mid w \in \Sigma^*\}$ is not context-free: $w = a^k b a^k b a^k b$.

L is context-free and R is regular, then $L - R = L \cap \overline{R}$ is context-free, while $R - L$ is not because it can be $\overline{L}$.

Turing Machine And Grammar

TM: $M = (K, \Sigma, \delta, s, H)$, $\delta = \{(q, w\underline{a}u)\}$.

Simple TMs: a, L, R, L^n, R^n, L_a, R_a, $L_{\overline{a}}$, $R_{\overline{a}}$.

Decide language: $(s, \triangleright \sqcup w)$, accept with y or reject with n.

Decide / recursive: $(s, \triangleright \sqcup w)$, accept with y or reject with n.

Compute function / recursive: Halts with $(s, \triangleright \sqcup w) \vdash_M^* (h, \triangleright \sqcup f(w))$. For numbers use binary notation.

Semidecide / recursively enumerable: Halts or not.

Language to TM: Manually.

Language to Grammar: Manually.

Examples:

- $\{ww \mid w \in \{a, b\}^*\}$: Generate ww^R with middle marker and end transformer.
- $\{a^{2^n} \mid n \geq 0\}$: Generate any amount of crawling doublers at left.

Function to TM: Manually.

Closure of recursive language: Union, concatenation, Kleene star, intersection, complementation and difference.

n-tape, n-head, two-way and n-dimensional tape TM are equivalent to standard TM.

NTM: $M = (K, \Sigma, \Delta, s, H)$, $\Delta = \{(q, w\underline{a}u)\}$.

NTM semidecides language: Accept (Halts for once).

NTM decides language: Halts for all and halts in y at least once.

NTM computes function: Halts with one output for all.

NTM is equivalent to standard TM. (By **dovetailing**)

Dovetailing: $\cdots$

Grammar: $G = (V, \Sigma, R, S)$, $R = \{u \to w\}$.

$\{a^n b^n c^n \mid n \geq 1\}$: $S \to ABCS$, $S \to T_c$, can repair order, transformer T_c crawls from right to left and turns C to c, and optionally turns into T_b, similarly T_b and T_a.

Grammar is equivalent to TM.

Grammar computes function: $SwS \Rightarrow_G^* f(w)$.

Recursive (Grammar): Grammatically computable.

Basic functions: $zero_k$, $id_{k,j}$, $succ$.

Primitive recursive function: Basic function and those obtained by composition, recursive definition. $plus$, $mult$, exp, f_m, sgn, $pred$, $m \sim n$.

Primitive recursive predicate: Primitive recursive function with values only 0 and 1. $iszero$, $positive$, $equal$, $greater-than-or-equal$, $less-than$, $\neg$, $\vee$, $\wedge$.

Function defined by cases is also primitive recursive (by $\cdot$ and $+$). rem, div.

$digit(m,n,p) = div(rem(n,p^m),p^{m\sim 1})$, $odd(n) = digit(1,n,2)$.

$sum_f(n,m) = \Sigma_{k=0}^{m} f(n,k)$ and $mult_f(n,m) = \Pi_{k=0}^{m} f(n,k)$ are primitive recursive.

$y|x = \exists t_{(\leq x)}(y \cdot t = x)$, y divides x.

$prime(x) = (x > 1) \wedge \forall_{(<x)}(t = 1 \vee \neg(t|x))$.

The set of primitive recursive function is enumerable, then by diagonalization ($g(n) = f_n(n) + 1$) it is a proper subset of recursive function.

Minimalization:
$\mu\, m[g(n_1,\cdots,n_k,m) = 1] = \text{the least } m \text{ such that } g(n_1,\cdots,n_k,m) = 1 \text{ or } 0 \text{ otherwise}$.

Minimalizable: If brute-force method always terminates.

μ-recursive: Basic function and those obtained by composition, recursive definition, and minimalization of minimalizable functions.

Diagonalization cannot be applied to find a recursive function that is not μ-recursive because the function minimalization applied upon may not be minimalizable.

$log(m,n) = \mu\, p[greater-than-or-equal((m+2)^p, n+1)]$. ($log_m(n) = \lceil log_{m+2}(n+1) \rceil$, to avoid pitfall when $m \leq 1$ or $n = 0$.)

Function $f : \mathbb{N}^k \to \mathbb{N}$ is μ-recursive iff it is recursive.

Prove that a function is primitive recursive:

Examples:

- $factoria(n)$: $factorial(0) = 1$, $factorial(n+1) = mult(n, factorial(n))$.
- $gcd(m,n)$: $gcd(m,n) = n$ if $rem(m,n) = 0$, $gcd(n, rem(m,n))$ otherwise.

Undecidability

Church–Turing thesis:

Algorithm: A TM that always halts.

Undecidable: Not recursive.

Universal TM: $U(\text{``}M\text{''}\ \text{``}w\text{''}) = \text{``}M(w)\text{''}$

Not recursive but r.e. language:
$H = \{\text{``}M\text{''}\ \text{``}w\text{''} |\ \text{Turing Machine } M \text{ that halts on input string } w\}$.

$H_1 = \{``M" |\ \text{TM } M \text{ halts on input string } ``M"\}$ will be $halts(X, X)$, and $\overline{H_1}$ will be $diagonal(X)$ which is even not r.e..

Recursive language is a proper subset of r.e. language.

Closure of r.e. language: Union, concatenation, Kleene star and intersection, but not complementation or difference.

Reduction: There is a reduction from L_1 to L_2 ($L_1 \leq L_2$) iff $x \in L_1 \Leftrightarrow r(x) \in L_2$.

If L_1 is not recursive, and $L_1 \leq L_2$, then L_2 is not recursive. (Otherwise TM_2 will decide L_1.)

Prove that a language is not recursive:

- Find reduction from H to language, by defining recursive function *for machine* from $``M" ``w"$ to M'.
    - Notice that M' should satisfy $x \in L_1 \Leftrightarrow r(x) \in L_2$, which is the reverse of intuition. Or say M' halts when M halts on w, does not halt when M does not on w.

Prove that a language is recursive:

- Closure under union, concatenation, Kleene star, intersection, complementation and difference.
- Both L and $\overline{L}$ are r.e..

Enumerate: $L = \{w \mid (s, \triangleright \sqcup) \vdash_M (q, \triangleright \sqcup w)\}$.

Turing-enumerable: Enumerable by a TM, equivalent to recursively enumerable (by dovetailing).

Lexicographically Turing-enumerable: Derivation comes lexicographically, equivalent to recursive.

$L(M)$: Language semidecided by M.

Rice's Theorem: If C is a proper and non-empty subset of the class of recursively enumerable languages, then the following problem is undecidable: Given a TM M, is $L(M) \in C$?

So almost all questions of the form "Does TM M halt on this kind of input?" are undecidable, for example whether $L(M) = \emptyset$, $L(M)$ is finite, $L(M) = \Sigma^*$, $e \in L(M)$.

There is no algorithm to determine, given any grammar G and any string w, whether $w \in L(G)$.

To determine whether $L(G_1) \cap L(G_2) = \emptyset$ is unsolvable, where G_1 and G_2 are both context-free grammars.

Tiling is unsolvable: Tiling the whole pane with abutting edges match. ⋯

Caveats:

The set of TM is countable infinite (enumerable).

Exam

- **Closure of regular expression**: Union, concatenation, Kleene star, complementation, intersection and difference.
- **Closure of CFG**: Union, concatenation and Kleene star, but not complementation, intersection or difference.
- **Closure of recursive language**: Union, concatenation, Kleene star, intersection, complementation and difference.
- **Closure of r.e. language**: Union, concatenation, Kleene star and intersection, but not complementation or difference.

CFG has the minimal of union, concatenation and Kleene star; r.e. language has intersection in addition; others have all.

Universal TM as the last problem.