Regular Languages

A language is regular if and only if it is recognized by a DFA, NFA, or Regexp.

DFA

$M = (Q, \Sigma, \delta, R, q_0 \in Q, F \subseteq Q)$

NFA

DFA but with epsilon transitions. DFA equivalent: each dfa state represents a subset of NFA states.

Regular Expressions

A regexp over Σ is one of $a \in \Sigma, \epsilon, \emptyset, (R_1 \cup R_2), (R_1 \circ R_2), R_1^*$ where each R_i is a regexp.

Closure Properties

$\cup$, $\cap$, $\circ$, $\star$, $\bar{A}$

Pumping Lemma for Regular Languages

Let A be a regular language. There must exist some $p \geq 1$ such that for all $w \in A$: $|w| \geq p$ there is some split $w = xyz$:

1. $xy^iz \in A$ for all $i \geq 0$
2. $|y| \geq 0$
3. $|xy| \leq |p|$

This is a necessary but not sufficient condition to be a regular language!

Non-regular Languages

- $0^n1^n \mid n \geq 0$

Context Free Languages

A Language is Context Free if and only if it is recognized by a PDA or generated by a CFG. All regular languages are context free but not all context free languages are regular! This can be proven by converting a DFA for language L to a PDA trivially (just ignore the stack).

PDA

Think NFA but with a stack. $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$

CFG

$G = (V, \Sigma, R, S)$. Grammars *derive* strings. Ambiguity: can have multiple parse trees for a single grammar!

- V are VARIABLES and disjoint from the set of terminals
- Σ are the terminals
- $R : V \times (V \cup \Sigma)^*$ are production rules

Closure Properties

$\cup$, A^*, $\circ$ and also the following lemma: if C is a CFL and R is a regular language then $C \cap R$ is a CFL. Corollary: if R is a regular language and $A \cap R$ is a non-CFL then A is a non-CFL.

Pumping Lemma for CFLs

Let A be a context free language. There must exist some $p \geq 1$ such that for all $w \in A$: $|w| \geq p$ there is some split $w = uvxyz$:

1. $uv^ixy^iz \in A$ for all $i \geq 0$
2. $|vy| \geq 1$ v and y cannot both be ϵ
3. $|vxy| \leq p$

This is due to the pigeonhole principle: for a sufficiently long parse tree for a CFG (height of tree greater than number of variables) *some* variable R must be repeated! This is a necessary but not sufficient condition to be a CFL!

Non-Context Free Languages

Can prove a language is not context free either by the pumping lemma or the closure properties.

- $\{a^nb^nc^n \mid n \geq 0\}$ cannot pump when $n = p$
- $\{ww \mid w \in \{0,1\}^*\}$ cannot pump string $w = 0^P1^P0^P1^P$

Turing Machines

$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ Has a finite state control with infinite tape. Has read head, can move L, R.

A configuration uqv states that the read head $q \in Q$ of the machine is at the leftmost symbol of v where $u, v \in \Gamma^*$ represent the contents of the tape.

A TM *accepts* on input if the sequence of configs from the start config, given by δ reaches the accept state in some *finite number of steps*. A TM *rejects* on input if the sequence of configs from the start config, given by δ reaches the reject state in some *finite number of steps*.

Church-Turing Thesis

Turing-machines can compute *anything* that can be computed - we define "what can be computed" as "what can be computed by a Turing-machine"

Recognizability

A TM *recognizes* a language if it

1. accepts $\forall w \in L$
2. rejects **or** loops $\forall w \notin L$.

Recognizable languages are closed under: $\cup, \cap, A^*, \circ$. These are also known as "recursively-enumerable" languages.

Decidability

A TM *decides* a language if it

1. accepts $\forall w \in L$
2. rejects $\forall w \notin L$ and *does not loop*

Decidable languages are closed under: $\cup, \circ, A^*, \cap$, and complement

Equivalent Variants

- K-Tape Turing machines - can be emulated with a single tape turing machine by putting all k tapes on the single tape with dividers, etc
- Nondeterministic Turing Machines - can be emulated with a 3-tape turing machine: 1st tape input, 2nd tape simulation, 3rd tape address (think BFS of the computation tree). An NTM accepts if some thread reaches the accept state. An NTM rejects if all threads terminate and reject. An NTM loops in any other case.

Proof of NTM and TM equivalence: 3-tape TM as above. If any simulated branch accepts then accept. If all locations of some length/depth reject then reject.

Why BFS instead of DFS for a NTM? Because DFS may go forever down a branch, and BFS ensures that the NTM will visit every node in the tree until it encounters and accepting configuration!

Enumerators

An enumerator is a special type of TM with two tapes: one work tape (same as a normal tape) and one write-only printer (or output tape). Writes strings separated by a # to output tape. $L(E) = \{w : E\ outputs\ w\ eventually\}$. A language is **enumerable** if it is the language of some enumerator. Enumerators do not necessarily halt.

Enumerabibilty and Recognizability

A language is *enumerable* $\Leftrightarrow$ it is *recognizable*.

Proof: enumerable $\Rightarrow$ recognizable. Given E for language, build a TM M that recognizes as follows: run E and if the input w ever appears on its ouput tape then accept. If E halts reject. If $w \in L$ then E eventually outputs w. If $w \notin L$ then E never outputs w, no M either loops or rejects.

Proof: recognizable $\Rightarrow$ enumerable. Given TM M that recognizes L build E. Consider strings in Σ^* in some order: $w_1 = \epsilon, w_2 = 0, \ldots$. For $i = 1, 2, \ldots$ for each string $w_j, 1 \leq j \leq i$ run M for up to i steps. If M ever accepts then print w_j.

For any string $w \in \Sigma^*$ E eventually runs M for any desired finite number of steps.