Context Free Languages

- Reminder: A grammar is Regular if all the rules have one of the following forms:

$A \to aB$
$A \to a$
$A \to \lambda$

$A, B \in V$
$a \in \Sigma$

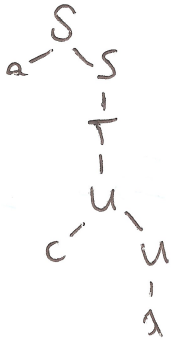
Note: the above grammar can be defined as

$A \to aB \mid a \mid \lambda$

Ex) for G: $S \to aS \mid T$
$T \to bT \mid U$
$U \to cU \mid \lambda$

we have this derivation:
$S \Rightarrow aS \Rightarrow aT \Rightarrow aU \Rightarrow acU \Rightarrow ac$

with this parse tree:



- L regular <=> L generated by regular grammar.

To construct a grammar for a regular language L:
- get a DFA for L
- make a variable for each state
- the starting variable is that of the initial state
- variables of accepted states $\to \lambda$.

Note: regular grammars have regular languages
regular languages don't necessarily have regular grammars

A grammar $G: \{V, \Sigma, R, S\}$ is context free if all rules in $R \in (V \cup \Sigma)^*$

∴ regular grammars are CFG.

- Union of two grammars:

G_1: $S_1 \to 0 S_1 1 \mid \lambda$
G_2: $S_2 \to 1 S_2 0 \mid \lambda$

add $S \to S_1 \mid S_2$ for union
(and $S \to S_1 S_2$ for concatenation)

- For any grammar G and language L, to prove $L(G) = L$:

prove $L(G) \subseteq L$
usually trivial, just

prove $L \subseteq L(G)$
two ways
by induction

Example: (with induction) $L=\{w\in\{a,b\}^* : |w|_a=|w|_b\}$

$G:\{V,\Sigma,R,S\}$ $\quad V=\{S\}$ $\quad \Sigma=\{a,b\}$

$R: S\to aSb \mid bSa \mid SS \mid \lambda$

In this case we have no exact form for $w\in L$ so instead we can resort to a mere explanation.

$L(G)\subseteq L$:

all 4 rules in R involve the same number of as as bs so every string that can be derived will also have $\#a=\#b$.

$L\subseteq L(G)$:

we will prove it by induction

- $S\Rightarrow\lambda$ $\quad 0=0$

  $S\Rightarrow aSb\Rightarrow ab$ $\quad 1=1$

  $S\Rightarrow bSa\Rightarrow ba$ $\quad 1=1$

- Let $w\in L$ of length $2n+2$

  w can come in 3 forms:

  * $w=anb$

    $S\Rightarrow aSb \overset{*}{\Rightarrow} anb=w$ (by induction)

  * $w=bna$

    $S\Rightarrow bSa \overset{*}{\Rightarrow} bna$ (by ind.)

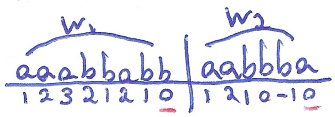
  * $w=ana$

    By a simple counting argument, we can deduce that there must be a cut in w into two (w_1 & w_2).

    $w=w_1w_2$

    $S\Rightarrow SS \overset{*}{\Rightarrow} w_1S \overset{*}{\Rightarrow} w_1w_2\Rightarrow w$ (by ind. / by ind.)

- Assume every string n of length 2n having the same # of a & b can be derived.



Examples: (without induction)

$L=\{a^nb^n : n\in\mathbb{N}^*\}$ $\quad G: S\to \overset{①}{a}S\overset{②}{b}\mid\lambda$

$L(G)\subseteq L$:

$S\Rightarrow\lambda$

$S\Rightarrow aSb\Rightarrow ab$

$aaSbb\Rightarrow aabb$

$aaaSbbb\Rightarrow\cdots$

$L\subseteq L(G)$:

let $w=a^ib^i\in L$

w can be derived by using rule ① i times & then terminating by ②.

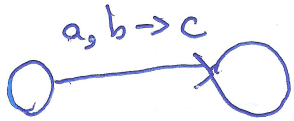
Note: when using the DFA of a language to derive the grammar, no need to prove $L=L(G)$.

Pushdown Automata (NPDA)

It is an NDFA with a stack.

Because it can process with memory, it can handle things a finite automata can't like $\{a^n b^n ; n \in \mathbb{N}^*\}$

transitions look like this:



a: read a from input (same as any transition)

b: pop b from the stack to make this transition (while also reading a)

Note: just like we can't perform a transition if the current input isn't a, we can't do so if the top of the stack isn't b.

c: push c into the stack.

λ as b or c means to not pop from or push into the stack during transitions

Note:

we use these to assure that, at the end, the stack is empty.

Note: b and c can be strings

ex) $L = \{a^i b^j ; j \leq i \leq 2j\}$