COMS W3261 – Lecture 6.

Some review of CFGs, Pushdown Automaton.

(V, Σ, R, S) $V = \{S, A, B\}$, $\Sigma = \{1, 0\}$

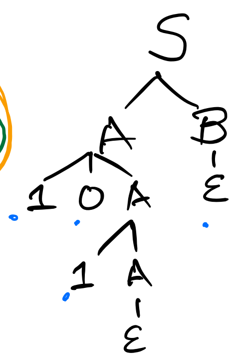
Teaser: Is this grammar: $S \to AB$, $S \to BA$

$$S \to AB \mid BA$$
$$A \to 10A \mid 1A \mid \varepsilon$$
$$B \to 01B \mid \varepsilon$$

$\varepsilon = ""$

ambiguous?

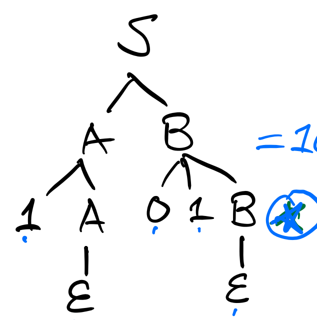
A string is ambiguously derived if it admits at least two different parse trees, or, equivalently, if it has at least two leftmost derivations.



// terminals on every leaf

// read string off left to right

$= 101$



$= 101$

YES – this grammar has at least one ambiguously derived string, so the grammar itself is ambiguous.

(Def. A leftmost derivation is a sequence of substitutions in which we always substitute the leftmost variable.)

$S \Rightarrow AB \Rightarrow 10AB \Rightarrow 101AB \Rightarrow 101B \Rightarrow 101.$

$S \Rightarrow AB \Rightarrow 1AB \Rightarrow 1B \Rightarrow 101B \Rightarrow 101.$

Announcements: HW #3 due 7/19 @ 11:59 PM EST.
HW #4 due 7/26 —— " ——.
HW #1, HW #2 — solutions up on website
grades done.

HW #2 feedback.
↳ LaTeX — see Ed post.
↳ Guidance hw answers.
↳ Moving Thursday PM hours (Tim) to 5:30–7:00 PM EST.

Reminder: write up HW sol'ns separately.

Today:
1. Review — (PL, CFG)
2. Chomsky Normal Form
3. Pushdown Automata — automata w/ memory.

---

1. Review

Example. (Pumping Lemma). Prove that the language

$$L = \{ 1^{n^2} \mid n \geq 0 \} \text{ is nonregular.}$$

Proof. Assume L is regular. Then L satisfies the pumping lemma, and there exists a pumping length p. For every string $s \in L$, $|s| \geq p$, there exists some way of dividing s into substrings x, y, and z such that $|y| > 0$, $|xy| \leq p$, for all i, $xy^iz \in L$.

Choose $s = 1^{p^2}$. $s \in L$, $|s| \geq p$.

Idea: we'll show s can't be pumped by showing that
$$p^2 < |xyyz| < (p+1)^2.$$
Suppose $s = xyz$. Then $xyyz = 1^{p^2+|y|}$.
We have $p^2 < p^2+|y| = |xyyz|$, when $|y|>0$.
$$p^2+|y| \leq p^2+|xy| \leq p^2+p < p^2+2p+1 = (p+1)^2.$$
Thus $|xyyz|$ is not a square number; $xyyz \notin L$, s cannot be pumped which is a contradiction. Thus L is nonregular. ∎

Hint for PS 3 1.1: Think about closure and think about related languages.

CFGs & CFLs

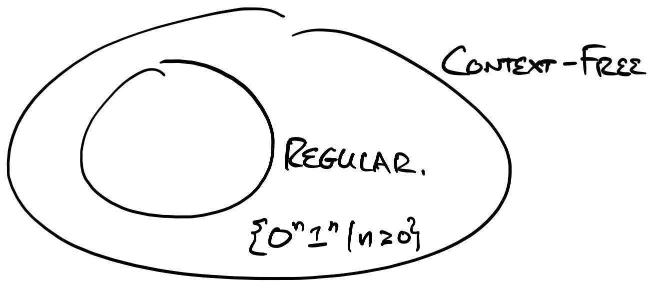
- A context-free grammar consists of substitution rules that replace variables with strings of variables and terminals.
- A string is derived ($\overset{*}{\Rightarrow}$) by substituting variables until we reach a string of only terminals.

// write rules $A \longrightarrow w$

// write derivations $uAv \Rightarrow uwv$

If I write $S \overset{*}{\Rightarrow}$ string
$$S \Rightarrow \cdots \Rightarrow \cdots \Rightarrow \cdots \Rightarrow \text{string}.$$
- The set of all strings that can be derived from the start symbol is the language of a grammar. The class of all languages described by CFGs is the context-free languages.

Last time, we saw that Regular L's $\subset$ Context-Free Languages.



- We saw that CFGs can describe nonregular languages (like $\{0^n1^n \mid n \geq 0\}$)

$$S \to 0S1 \mid \varepsilon$$

- We saw that DFAs could be converted into CFGs.

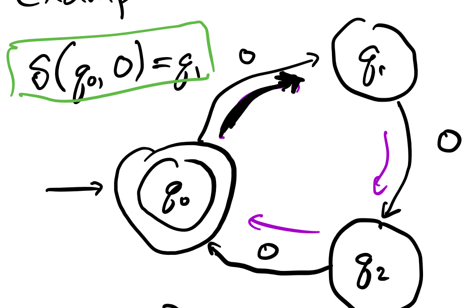
To convert a DFA into a CFG:

Idea: make sure we always have one variable at each step of our derivation that corresponds to a DFA state. Our production rules will write down terminals corresponding to an accepting string.

(for NFAs: $R_i \to \varepsilon R_j = R_i \to R_j$ on ε-transition.)

1. Create a rule $R_i \longrightarrow a R_j$ for each transition $\delta(q_i, a) = q_j$
2. Create rules $R_i \longrightarrow \varepsilon$ for each accept state R_i.

Example. $\{w \mid w$ is a string of 0's, $|w|$ is divisible by $3\}$.



$R_0 \longrightarrow 0R_1 \mid \varepsilon$
$R_1 \longrightarrow 0R_2$
$R_2 \longrightarrow 0R_0$

$R_0 \Rightarrow \varepsilon$. $\qquad \varepsilon$.
$R_0 \Rightarrow 0R_1 \Rightarrow 00R_2 \Rightarrow 000R_0 \Rightarrow 000$.
$R_0 \Rightarrow \cdots \Rightarrow 000000,\ 000000000$.

Example. Two CFGs that generate the same language.

$$S \to 0S \mid A \qquad\qquad S \to S1 \mid A$$
$$A \to A1 \mid \varepsilon \qquad\qquad A \to 0A \mid \varepsilon$$

$$S \stackrel{*}{\Rightarrow} 0000S \Rightarrow 0000A$$
$$\stackrel{*}{\Rightarrow} 0000A111 \Rightarrow 0000111$$

$\{0^n 1^m \mid n \geq 0,\ m \geq 0\}$

$$S \stackrel{*}{\Rightarrow} S111 \Rightarrow A111$$
$$\stackrel{*}{\Rightarrow} 0000A111 \Rightarrow 0000111$$

How do we tell if two CFGs recognize the same language?

Idea: Define a simple "normal form" / "standard form."

(e.g. Does $\frac{24}{18} = \frac{8}{6}$? $= \frac{4}{3}$.)

[Can be helpful in proofs to assume that a grammar is in normal form $\Rightarrow$ simple.]

Def. (Chomsky Normal Form). A context-free grammar is in Chomsky Normal Form if every rule has the form

$$A \to BC$$
$$A \to a,$$

where A, B, and C are variables and a is a terminal. (Also — the start variable appears only on the left, and we allow the special rule $S \to \varepsilon$, where S is the start variable.)

Theorem. Any CFG has an equivalent CFG in Chomsky Normal Form.

Proof idea: Given an arbitrary CFG with rules that take variables to strings of variables and terminals, show how to break our substitution rules down into simple rules that meet our requirements for normal form.

(See end of sec 2.1 in Sipser, not required.)

Solutions don't need to be in CNF (unless we ask explicitly.)

Break: back at 11:26.

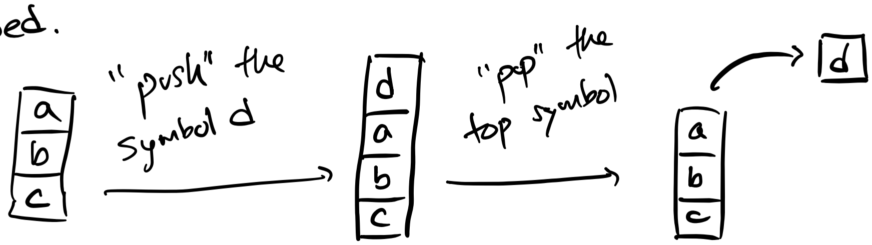
---

3. Pushdown Automata

Idea: we can use a <u>stack</u> as a simple memory.

DFAs, NFAs: memory ≈ what state you're in.

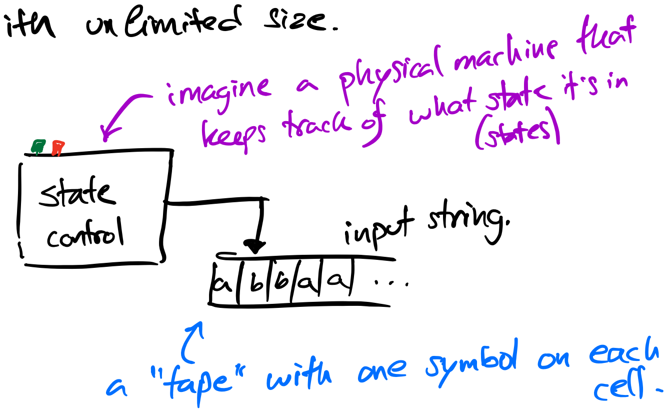
<u>Stack</u>: a finite sequence of symbols that can be pushed or popped.



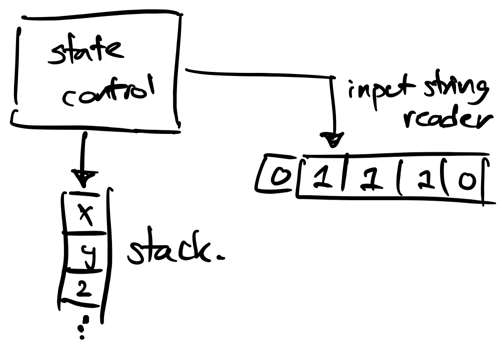
- we have access to the topmost element only. (Need to pop multiple times to access other elements.)
- we assume a stack with unlimited size.

<u>Picture</u>:

Picture of DFA/NFA:



Picture of a Pushdown Automaton (PDA):



PDA: On each computational step, we will consult the input string, (optionally) pop from the stack, change state, (optionally) push to the stack.

Example: A PDA state diagram.

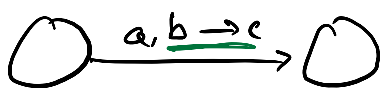
Goal: Build a Pushdown Automaton that recognizes

$$\{0^n 1^n \mid n \geq 0\}.$$

(We'll write each transition as

$$a, b \longrightarrow c.$$

Read this notation as "If we see a on the input string and we pop b from the stack, push c onto the stack and transition.

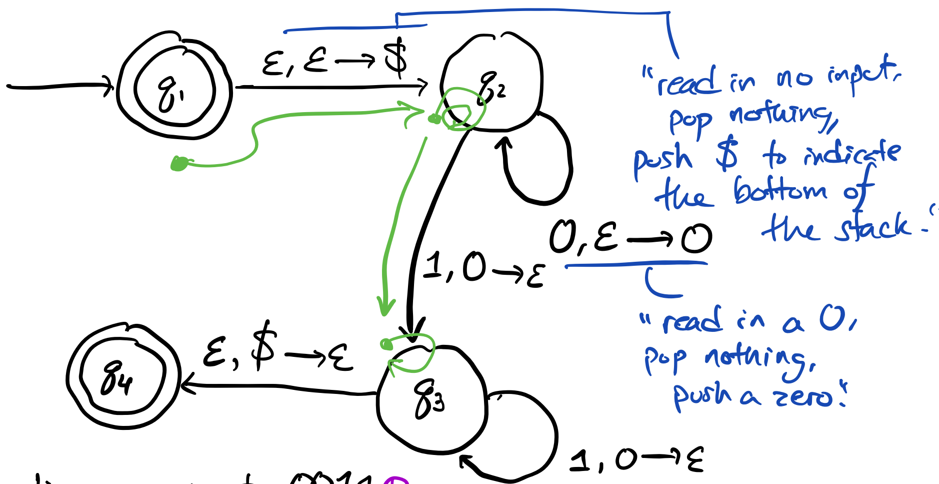


$a, \varepsilon \rightarrow \varepsilon$

We can use ε to indicate push/pop nothing.)

Idea: Push 0's onto the stack until we run out of 0's on the input string. Then, we will pop a 0 every time we read a 1. Accept if and only if the number of zeroes is equal to the number of ones.

state diagram:



Example execution: on input 00110

state	current stack
q_1	ε
q_2	\$
q_2	0\$
q_2	00\$
q_3	0\$
q_3	\$
q_4	ε ✓
$\emptyset$	ε ✗

// ignoring extra epsilon-branches. this table shows an accepting branch.

Note: every branch will have its own stack.

example: $0, 1 \longrightarrow a$
$0, 1 \longrightarrow b$

Formal def of PDA

Idea: the crux is the new transition function. We'll (re)define

$$\Sigma_\varepsilon := \Sigma \cup \{\varepsilon\}$$ // Σ will be the "input alphabet"

$$\Gamma_\varepsilon := \Gamma \cup \{\varepsilon\}$$ // Γ will be the "stack alphabet"

$\mathcal{P}(S)$ denotes the power set of S: the set of all subsets of S.

$\left(\mathcal{P}(\{a,b,c\}) = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a,b\}, \{b,c\}, \{a,c\}, \{a,b,c\}\}\right)$

Our transition function will map

(1) a state, (2) an input symbol, (3) a popped element, to some set of (state, push element) pairs.

<u>Def</u> (Pushdown Automaton). A Pushdown Automaton is 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where

Q is a finite set of states,

Σ is a finite input alphabet,

Γ is a $\underbrace{\text{finite (stack alphabet)}}$, || stack alphabet is finite, stack is arbitrarily large.

q_0 is the start state,

$F \subseteq Q$ is the set of accept states,

$$\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \longrightarrow \mathcal{P}(Q \times \Gamma_\varepsilon)$$

("Give me a state, input symbol, and symbol popped, and i'll tell you some states to branch to and what to push.")

Our PDA accepts an input string $\underline{w = w_1 w_2 \cdots w_n}$, where each $w_i \in \Sigma_\varepsilon$, if there exists a sequence of states $r_0, r_1 \cdots r_n$ and a sequence of strings $s_0, s_1, \cdots s_n$, such that: $\underline{r_0 = q_0}$, $\underline{r_n \in F}$, $\underline{s_0 = \varepsilon}$,

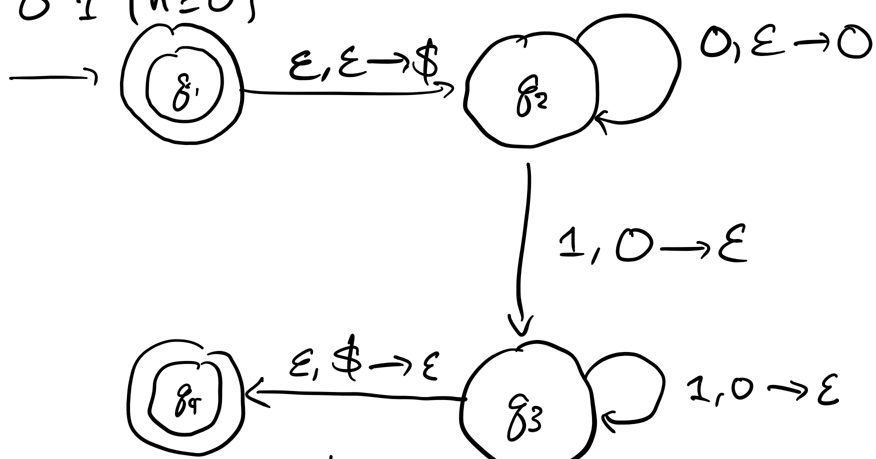
and for $i = 0, 1, \dots, n-1$, we have

$$(\underline{r_{i+1}}, \underline{b}) \in \delta(\underline{r_i}, \underline{w_{i+1}}, \underline{a}),$$

where $\underline{s_i = at}$ and $\underline{s_{i+1} = bt}$ for some $a, b \in \Gamma_\varepsilon$ and some $t \in \Gamma^*$.

Example: Formal notation for our state diagram.

$L = \{0^n 1^n \mid n \geq 0\}$



Call this PDA $P = (Q, \Sigma, \Gamma, \delta, q_1, F)$,

where $Q = \{q_1, q_2, q_3, q_4\}$

$\Sigma = \{0, 1\}$

$\Gamma = \{0, \$, (1)\}$

$F = \{q_1, q_4\}$

// note: Γ is often a superset Σ, and we can feel free to add any symbol we like.

and δ is given as follows:

$$\delta(q_1, \varepsilon, \varepsilon) = \{(q_2, \$)\}$$
$$\delta(q_2, 0, \varepsilon) = \{(q_2, 0)\}$$
$$\delta(q_2, 1, 0) = \{(q_3, \varepsilon)\}$$
$$\delta(q_3, 1, 0) = \{(q_3, \varepsilon)\}$$
$$\delta(q_3, \varepsilon, \$) = \{(q_4, \varepsilon)\}$$
$$\delta(\cdot, \cdot, \cdot) = \emptyset \text{ for all other inputs.}$$

Example. Build a PDA that recognizes $\{ww^R \mid w \in \{0,1\}^*\}$

$(abc)^R = cba$

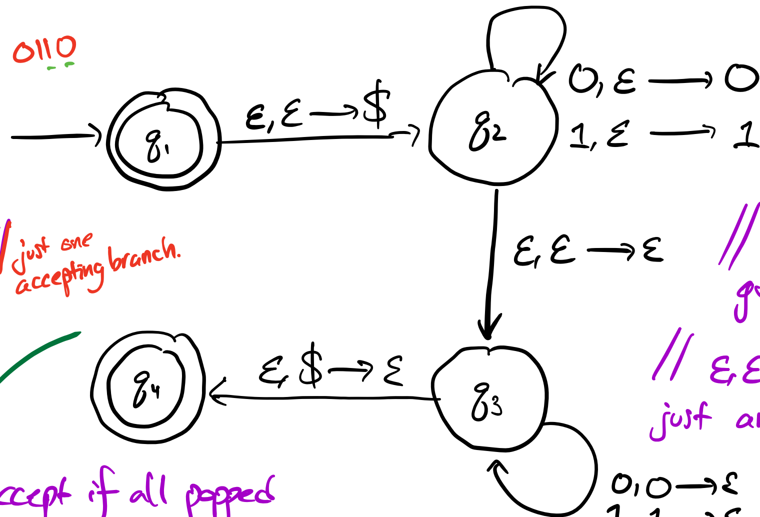
Idea: similar to $\{0^n 1^n \mid n \geq 0\}$. Push symbols onto the stack, nondeterministically guess the midpoint of the string then accept if the symbols we pop match the symbols remaining of the input.

0110

state	stack
q_1	ε
q_2	\$
q_2	0\$
q_2	10\$
q_3	10\$
q_3	0\$
q_3	\$
q_4	ε

// just one accepting branch.

✓



// nondeterministically guess middle

// $\varepsilon, \varepsilon \to \varepsilon$ is just an ε-edge.

// accept if all popped symbols match and we reach end of input / stack at same time.

// if we see that the top of the stack $\neq$ the next input symbol, the branch will die.

Next time: We'll prove that a language is recognized by a PDA $\longleftrightarrow$ it is generated by a CFG (i.e. if it is context-free.)

Reading: Sipser ch. 2.2.

1.1 HW. $L = \{ w \mid w \neq yy \text{ for any } y \in \{0,1\}^* \}$

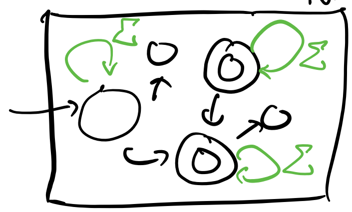
If $y = \varepsilon$, this rules out $w = \varepsilon\varepsilon = \varepsilon$.

$y = 101$, then this means $yy = 101101 \notin L$

$pre(A) = \{ xy \mid x \in A, y \in \Sigma^* \}$

One way: pre(A) = any string that starts with a string in X = any string that ever reaches an accept state in a DFA/NFA for A.

N:

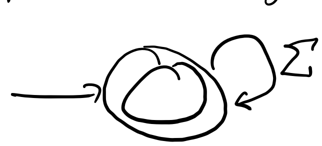


1) add edges for every symbol $\in \Sigma$ to each accept state.

(Σ as shorthand for transitions on any symbol in Σ).

$$\text{pre}(A) = A \circ \Sigma^*$$

A regular by assumption, Σ^* is regular because



exists.

$\therefore$ $A \circ \Sigma^* = \text{pre}(A)$ is regular by the closure of regular languages under concatenation.

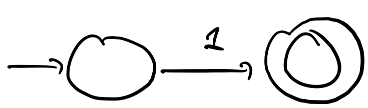
Possible alternative: NFA w/ many ε-edges?

Suppose $A = \{1\}$.

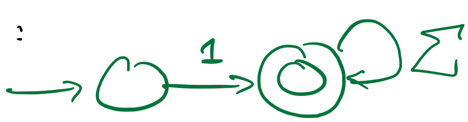
$\text{pre}(A) = \{1\} \circ \Sigma^* = \{w \mid w \text{ begins with } 1\}$.

$0110 \notin \text{pre}(A)$.

N_A:



$N_{\text{pre}(A)}$:



$O^k = OO \cdots k \text{ times} \cdots O$