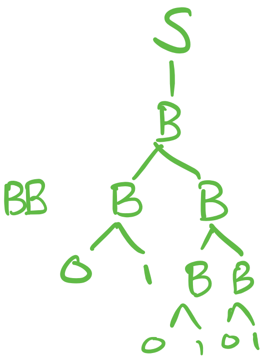
Puzzle:

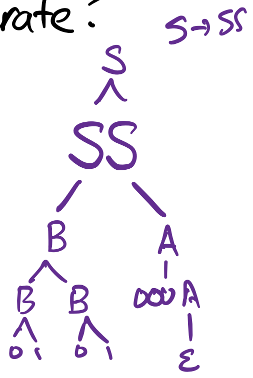
$$S \to A \mid B$$
$$A \to 000A \mid \varepsilon$$
$$B \to BB \mid 01$$

- Can this CFG derive 0101? ε? 000 0101? 01010?
- What language does this CFG derive / generate?
  - as a regular expression?

$S \Rightarrow A \Rightarrow \varepsilon$

$S \Rightarrow B \Rightarrow BB \Rightarrow 01B \Rightarrow 0101$





$$(000)^* \cup (01)^+$$

$S \to A \to \varepsilon$

$S \to A \to 000A \to 000$

$S \to A \to 000A \to 000000A \to 000000\ldots$

10, 1010, 101010...

$$S \to 0S0 \mid 1S1 \mid \varepsilon$$

$$S \Rightarrow 0S0 \Rightarrow 00S00 \Rightarrow 001S100$$
$$\Rightarrow 001100$$

Today:

1. Regular languages ⊂ CFLs
2. Pushdown Automata (PDAs).

Prop. Regular Languages $\subseteq$ CFLs.

Proof idea: Regular Expression → CFG.

Proof: By definition, every regular expression has one of six forms. We'll build a CFG for each.

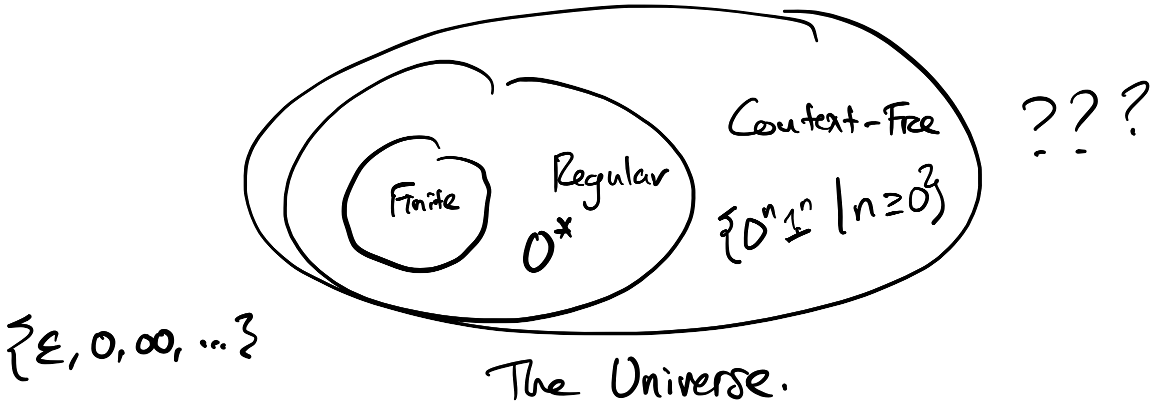
Reg. Expr.	Equivalent CFG
$a \in \Sigma$	$S \to a$
ε	$S \to \varepsilon$
$\emptyset$	'no rules' OR $S \to S$

Ind. hypothesis: Let R be a regular expression of fixed "length" k, and assume that all smaller regular expressions have an equivalent grammar.

Let R_1, R_2 be regular expressions "smaller" than R with corresponding grammars $G_1 = (V_1, \underline{\Sigma}, U_1, S_1)$, $G_2 = (V_2, \underline{\Sigma}, U_2, S_2)$.

Reg. Expr.	Equivalent CFG.
$R = R_1 \cup R_2$	$S \to S_1 \mid S_2$ (+ all rules in U_1, U_2) (assumption — no variables in common, rename if necessary)
$R = R_1 R_2$	$S \to S_1 S_2$ (+ rules in U_1, U_2)
$R = R_1^*$	$S \to S S_1 \mid \varepsilon$ (+ rules in U_1).

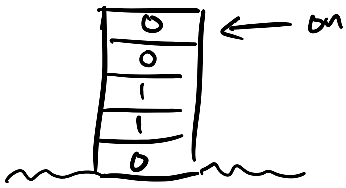
Thus, given our ind. hypothesis, R has an equivalent CFG. ∎



Automata w/ Memory: Pushdown Automata

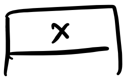
$$A = \{0^n 1^n \mid n \geq 0\}$$

Give our automaton a stack:



← only top visible/accessible.

push(x): add x to the top

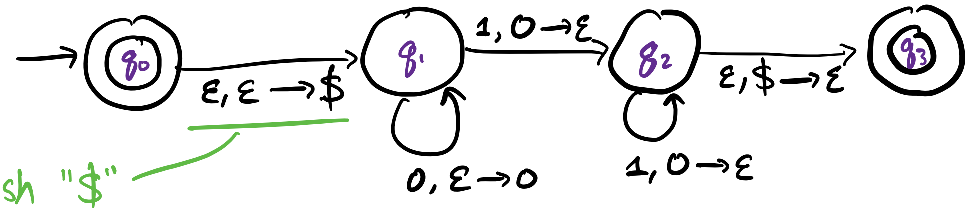


pop: take element off the top

Q: use stack to recognize A?

```
recA(string w):
    stack = ∅
    while next char = 0:
        stack.push(0)
    while next char = 1:
        stack.pop()
    if stack = ∅ and no more chars, accept
    else reject.
```

PDA state diagram:

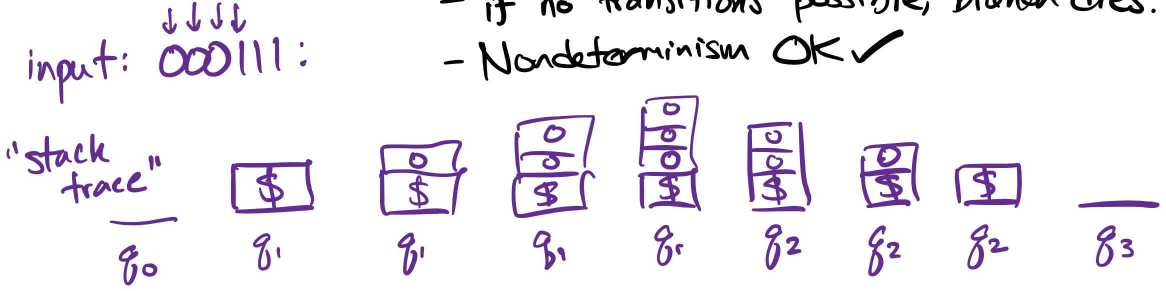


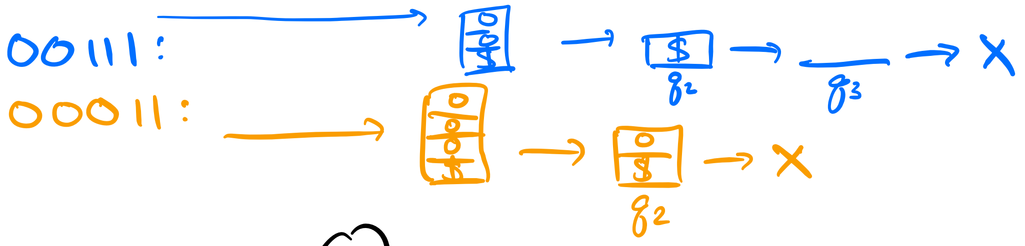
push "$"
to mark bottom
of stack

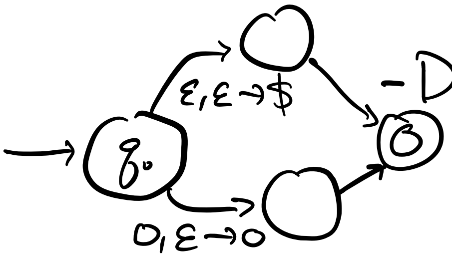
"read, pop, push"

- notes: we take a, b → c if and only if a is the next input char, b is at the top of the stack.
- if no transitions possible, branch dies.
- Nondeterminism OK ✓

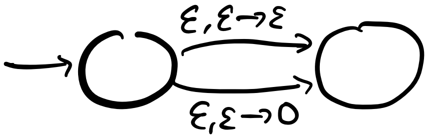
input: 000111:







- Different branches have different stacks.



Back at 2:10

ab^* : $\{a, ab, abb, \dots\}$

$(ab)^*$: $\{\varepsilon, ab, abab, \dots\}$

$a, b \to c$

"read, pop → push"

(notes – to video).

Def. PDA

A Pushdown Automaton is a 6-tuple

$$(Q, \Sigma, \Gamma, q_0, F, \delta)$$

$\gamma \Gamma$

set of accept states

finite set of states

finite input alphabet

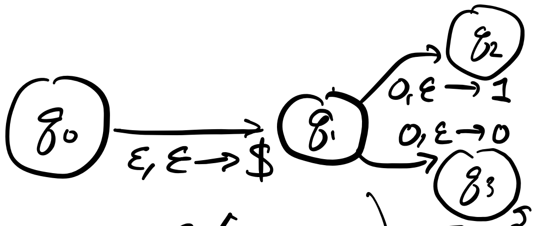
finite stack alphabet (can choose)

start state

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

$(\Sigma \cup \{\varepsilon\})$ $(\Gamma \cup \{\varepsilon\})$

a list of (state, push) pairs that specify where to go and what to push



$$\delta(q_0, \varepsilon, \varepsilon) = \{(q_1, \$)\}$$

$$\delta(q_1, 0, \varepsilon) = \{(q_2, 1), (q_3, 0)\}$$

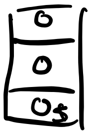
Our PDA <u>accepts</u> an input $w = w_1 w_2 \cdots w_n$, where each $w_i \in \Sigma_\varepsilon$, if there is a sequence of states $r_0, r_1, \cdots r_n \in Q$ <u>and</u> strings $s_0, s_1, \cdots s_n \in \Gamma^*$ such that $r_0 = q_0$, $s_0 = \varepsilon$, $r_n \in F$, and for $i = 0, 1, \dots, n-1$,

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

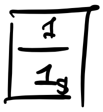
where $s_i = at$ and $s_{i+1} = bt$ for $a, b \in \Gamma_\varepsilon$ and $t \in \Gamma^*$.

$$L = \{ w \in \{0,1\}^* \mid w \text{ has the same number of 0's and 1's} \}.$$

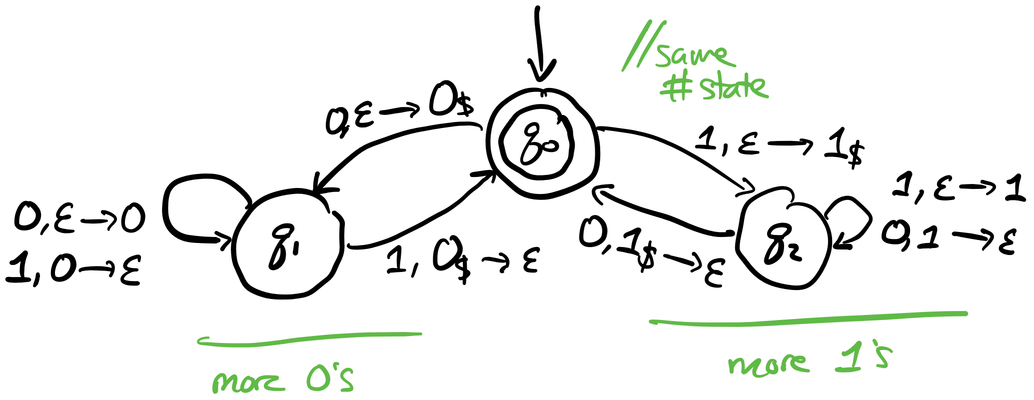
Idea: with the stack, keep track of the 0-1 "balance."



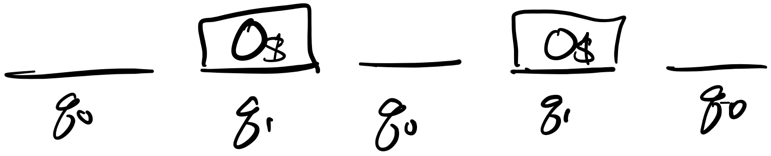
("I've seen 3 more zeroes than ones"

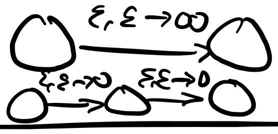


"I've seen two more ones than zeroes."



~~0101~~:





Back at 2:45

Puzzle. $(011)^R = 110$

$$C = \{ww^R \mid w \in \{0,1\}^*\}$$

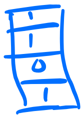
(even-length palindromes.)

Q: A PDA for this language?

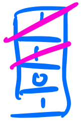
**(Just for fun: $D = \{a^i b^j c^k \mid i=j \text{ OR } i=k\}$)

1011⋮1101

① push onto stack



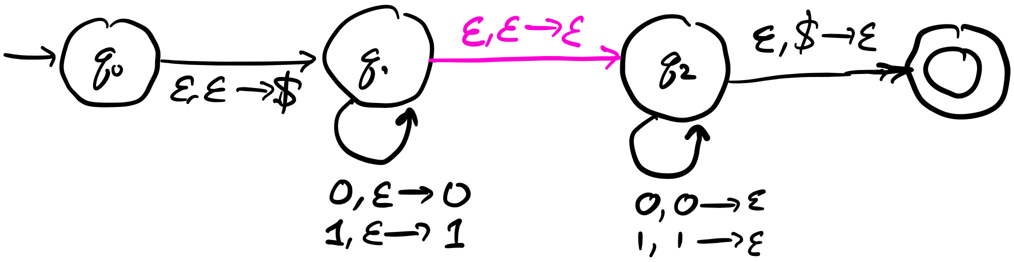
② compare element-by-element with second half



~~11~~01

Q: how do we know when to start popping?

A: nondeterministically guess!



10111101.

Accepting branch: q_0 to q_1, push  , take ε,ε→ε to q_2, pop 1, 1, 0, 1, matching against the input string, pop $ and accept

Other branches: take ε,ε→ε early or late.

Theorem: PDAs recognize exactly the CFLs.

Follows from

Lemma 1 (PDA→CFG). Any PDA has an equivalent CFG.

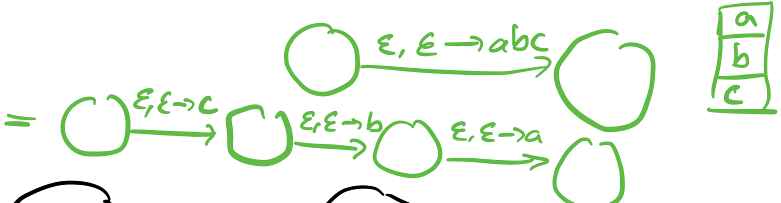
Proof omitted, see Sipser Lemma 2.27 pp 121-124.

Lemma 2 (CFG→PDA). Any CFG has an equivalent PDA. (Sipser Lemma 2.21).

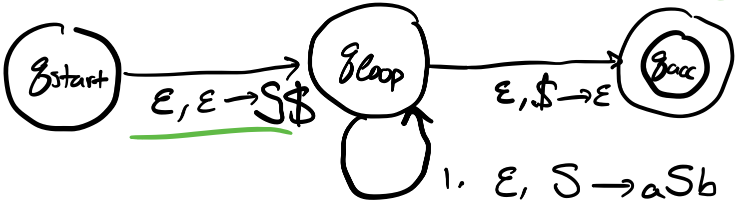
$$G: S \to aTb \mid T$$
$$T \to aT \mid \varepsilon$$

$$\underbrace{S \Rightarrow aTb \Rightarrow aaTb \Rightarrow aab.}_{\text{do on stack.}}$$

shorthand for pushing multiple characters:

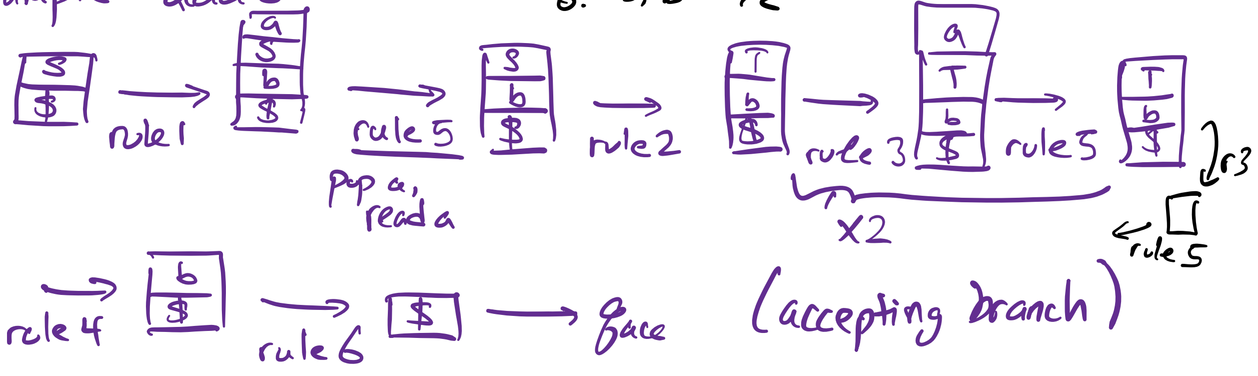


PDA for G:



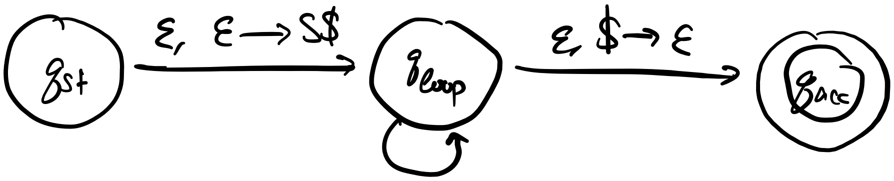
1. ε, S → aSb
2. ε, S → T
3. ε, T → aT
4. ε, T → ε
5. a, a → ε
6. b, b → ε

example: ~~a~~aab



(accepting branch)

<u>Proof sketch</u>. Given $G = (V, \Sigma, R, S)$, build P, a PDA, as follows:



From q_{loop} to itself, we have transitions $\varepsilon, A \to w$ for each rule $A \to w$ in R, where $A \in V$, $w \in (V \cup \Sigma)^*$.

Also, add the rule $a, a \to \varepsilon$ for each terminal $a \in \Sigma$.

Claim: If G generates the string w, P accepts w.

- Consider a sequence of substitution rules that derive w from S.
- From q_{loop}:
  - follow the next rule in our sequence if the top of the stack is a variable.
  - match the next input character if the top of the stack is a terminal.

Claim: If P accepts w, G derives w.

If P accept w, $\exists$ a branch of computation that:

- P pushed $\boxed{\begin{array}{c} S \\ \hline \$ \end{array}}$ to begin
- P followed derivation rules to turn all variables into terminals,
- P matched all terminals with input characters, until no stack or input character remained;
- P then popped \$ and accepted. $\square$

Takeaways:

Regular Languages $\subseteq$ CFLs

CFGs generate/derive the CFLs, PDAs recognize them.

Next time:

- a pumping lemma for CFLs.
- TURING MACHINES.

Reminders:

Office hours tonight @ 5:30 (Zoom)

HW 3 up, HW 1 posted.

Designing grammars for a given language.

$$\underbrace{(00)^*}_{A} \cup \underbrace{\varepsilon}_{B} \cup \underbrace{(\overbrace{0^+1}^{D})^*}_{C}$$

$D = 0^+1 = \{01, 001, 0001, \ldots\}$

$C = (0^+1)^* = \{\varepsilon, 01, 0101, 01001, 000101, 00010101, \ldots\}$

$S \rightarrow A \mid B \mid C$

$A \rightarrow \varepsilon \mid 00A$ // $(00)^*$

$B \rightarrow \varepsilon$

$C \rightarrow \varepsilon \mid DC$ // $(0^+1)^*$ // $\varepsilon, D, DD, DDD, \ldots = D^*$

$D \rightarrow E01$ // 0^+1

$E \rightarrow \varepsilon \mid 0E$ // 0^* // $\varepsilon, 0, 00, \ldots = 0^*$

PL: For any regular language L, there exists a number p such that any string $s \in L$ with $|s| \geq p$ can be divided $s = xyz$ satisfying

(1) $xy^iz \in L$ for $i \geq 0$,

(2) $|y| > 0$

(3) $|xy| \leq p$. // $|yz| \leq p$.

Could prove: "For my language L', for any number p, there exists a string $s \in L$ with $|s| \geq p$ that can't be divided in a way that satisfies 1, 2, and 3 above."

$H = \{0^n1^m \mid n \neq m\}$ → prove nonregular?

$A = \{0^n1^n \mid n \geq 0\}$

If H regular $\Rightarrow$ $\overline{H}$ regular $\Rightarrow$ $\underline{\overline{H} \cap 0^*1^* \text{ regular}}$

$= A$ regular

X A non-regular.

---

contradiction string $\underline{01^p}$.

- assume for contradiction $\exists$ D_H for this language

$\Longrightarrow$ D_H has a loop, in $0^{|Q|+1}$ call the loop size b.

$\Longrightarrow$ Now: $0^{|Q|+1}, 0^{|Q|+b+1}, 0^{|Q|+2b+1} \ldots$

$0^{|Q|+1}1^{|Q|+1} \in L$

$0^{|Q|+b+1}1^{|Q|+1} \in L$ accepts. X.

$|y| \in [1,p]$

$$0^p1^{\prod_{i\in[p]} i+p}$$

$$0^{p+(i-1)|y|}1^{\prod_{i\in[p]} i+p}$$

$|y|$ divides $\prod_{i\in[p]} i$

$\therefore \exists i$ s.t.

$$p + (i-1)|y| = p + \prod_{i \in [p]} i$$