Week 4 at a glance

Textbook reading: Section 2.2, 2.1.

Before Monday, read Definition 2.13 (page 111-112) introducing Pushdown Automata.

Before Wednesday, read Example 2.18 (page 114).

Before Friday, read Introduction to Section 2.1 (pages 101-102).

For Week 5 Monday: Theorem 2.20.

We will be learning and practicing to:

- Clearly and unambiguously communicate computational ideas using appropriate formalism. Translate across levels of abstraction.
  - Give examples of sets that are regular (and prove that they are).
    - **State the definition of the class of regular languages**
    - **Explain the limits of the class of regular languages**
    - **Identify some regular sets and some nonregular sets**
  - Use precise notation to formally define the state diagram of a PDA
  - Use clear English to describe computations of finite automata informally.
    - **Define push-down automata informally and formally**
    - **State the formal definition of a PDA**
    - **Trace the computation(s) of a PDA on a given string using its state diagram**
    - **Determine if a given string is in the language recognized by a PDA**
    - **Translate between a state diagram and a formal definition of a PDA**
    - **Determine the language recognized by a given PDA**
  - Describe and use models of computation that don't involve state machines.
    - **Identify the components of a formal definition of a context-free grammar (CFG)**
    - **Derive strings in the language of a given CFG**
    - **Determine the language of a given CFG**
    - **Design a CFG generating a given language**

TODO:

Schedule your Test 1 Attempt 1, Test 2 Attempt 1, Test 1 Attempt 2, and Test 2 Attempt 2 times at PrairieTest (http://us.prairietest.com)

Homework 3 submitted via Gradescope (https://www.gradescope.com/), due Tuesday 10/22/2024

Review Quiz 4 on PrairieLearn (http://us.prairielearn.com), complete by Sunday 10/27/2024

Monday: Pushdown Automata

recognizable by a DFA

Example regular sets over $\{0,1\}$:
eg. $\{0\}^* = \{0^i \mid i \geq 0, i \text{ int}\}$ $\quad \{0,1\}^*$

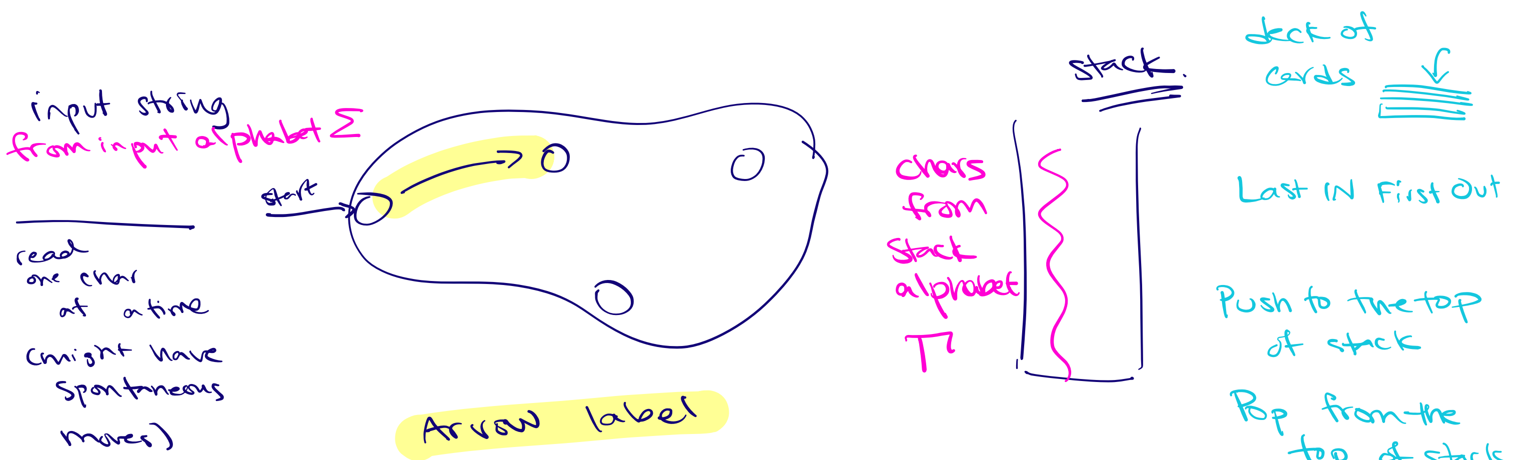
Regular sets are not the end of the story

- Many nice / simple / important sets are not regular eg. $\{0^p 1^p \mid p \geq 0, p \text{ int}\}$
- Limitation of the finite-state automaton model: Can't "count", Can only remember finitely far into the past, Can't backtrack, Must make decisions in "real-time"
- We know actual computers are more powerful than this model...

The **next** model of computation. Idea: allow some memory of unbounded size. How?

Sec 2.1
- To generalize regular expressions: **context-free grammars**

Sec 2.2.
- To generalize NFA: **Pushdown automata**, which is like an NFA with access to a stack: Number of states is fixed, number of entries in stack is unbounded. At each step (1) Transition to new state based on current state, letter read, and top letter of stack, then (2) (Possibly) push or pop a letter to (or from) top of stack. Accept a string iff there is some sequence of states and some sequence of stack contents which helps the PDA processes the entire input string and ends in an accepting state.



Arrow label

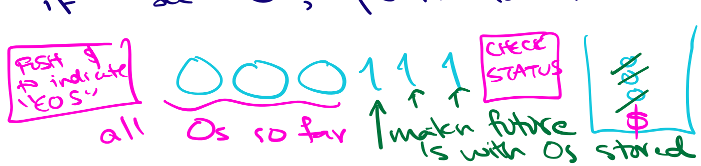
$\underline{\quad}, \underline{\quad}; \underline{\quad}$

input char to read OR ε ; top char to pop from stack OR ε ; effect of this transition, char to push to top of stack OR ε

Is there a PDA that recognizes the nonregular language $\{0^n 1^n \mid n \geq 0\}$?

COUNT # 0s BY STORING IN STACK.

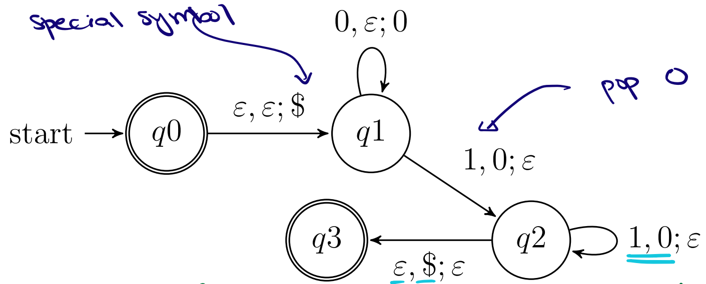
Idea: As read string
- if see empty string, accept
- if see 1 before a 0, reject.
- if see 0, push to stack to match against a future 1.



initialize stack with special symbol

store 0s for future

pop 0 for first 1 read, and switch "modes"



pop 0 for each 1 read

no outgoing transitions labelled 0. no outgoing transitions labelled 0.

The PDA with state diagram above can be informally described as:

> Read symbols from the input. As each 0 is read, push it onto the stack. As soon as 1s are seen, pop a 0 off the stack for each 1 read. If the stack becomes empty and we are at the end of the input string, accept the input. If the stack becomes empty and there are 1s left to read, or if 1s are finished while the stack still contains 0s, or if any 0s appear in the string following 1s, reject the input.

Trace ~~the~~ a computation of this PDA on the input string 01.

q0 , q0 , q1 , q2 , q3 ACC

(stacks: empty; $; 0 over $; $; empty)

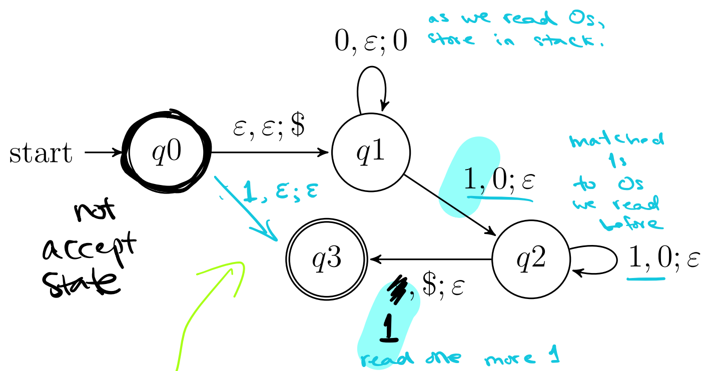
Trace the computations of this PDA on the input string 011.

1, 011, 00111, ...   e.g. accept 1, 011
reject ε, 0, 01

A PDA recognizing the set $\{ 0^n 1^{n+1} \mid n \geq 0\}$ can be informally described as:

> Read symbols from the input. As each 0 is read, push it onto the stack. As soon as 1s are seen, pop a 0 off the stack for each 1 read. If the stack becomes empty and there is exactly one 1 left to read, read that 1 and accept the input. If the stack becomes empty and there are either zero or more than one 1s left to read, or if the 1s are finished while the stack still contains 0s, or if any 0s appear in the input following 1s, reject the input.

Modify the state diagram below to get a PDA that implements this description:



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

continue on Wednesday!

<u>Bonus</u>: try a couple of other designs.

Wednesday: More Pushdown Automata

Definition A **pushdown automaton** (PDA) is specified by a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where Q is the finite set of states, Σ is the input alphabet, Γ is the stack alphabet,

nondeterministic

$$\underbrace{\delta}_{} : \underbrace{Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon}_{\text{curr state}} \to \mathcal{P}(\underbrace{Q \times \Gamma_\varepsilon}_{\text{arrow label effect}})$$

is the transition function, $q_0 \in Q$ is the start state, $F \subseteq Q$ is the set of accept states.

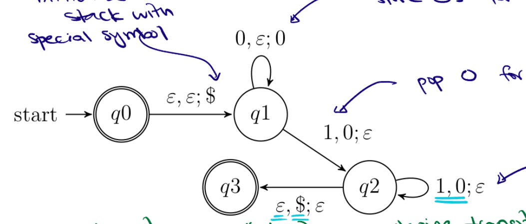
Draw the state diagram and give the formal definition of a PDA with $\Sigma = \Gamma$.

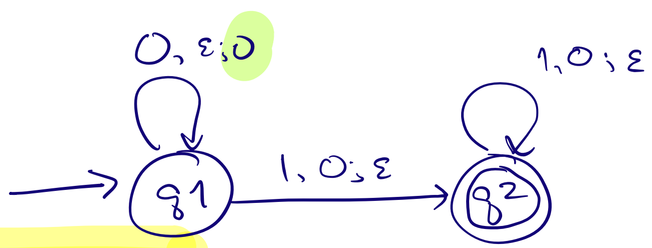
For reference, from previous example:

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

$\Gamma = \{0, \$\}$

$\Sigma \neq \Gamma$, $\Sigma \cap \Gamma \neq \emptyset$





001 accepted
01 accepted
ε not accepted

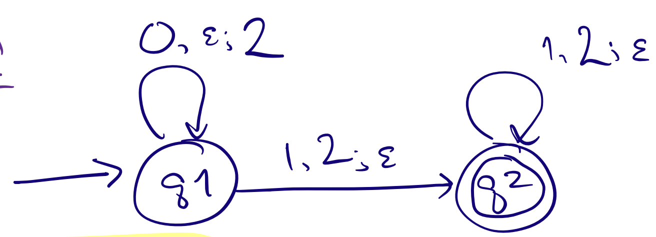
$\Sigma = \{0, 1\}$

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

$\{0^i 1^j \mid \begin{smallmatrix} 1 \leq j \\ i \geq j \end{smallmatrix}\}$

Draw the state diagram and give the formal definition of a PDA with $\Sigma \cap \Gamma = \emptyset$.

Relabel

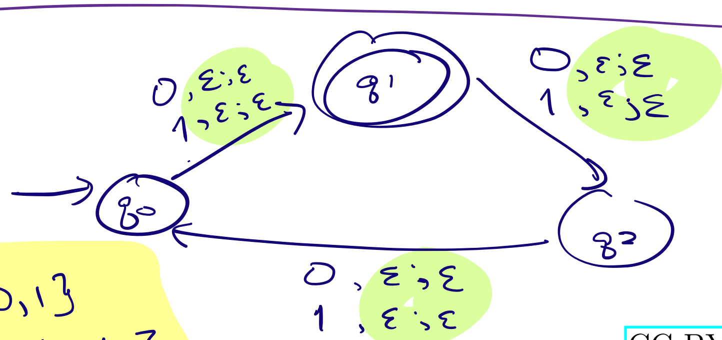


every time we read a 0 from input, push a 2 to stack

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

$\Gamma = \{2, 3\}$

Ignore stack



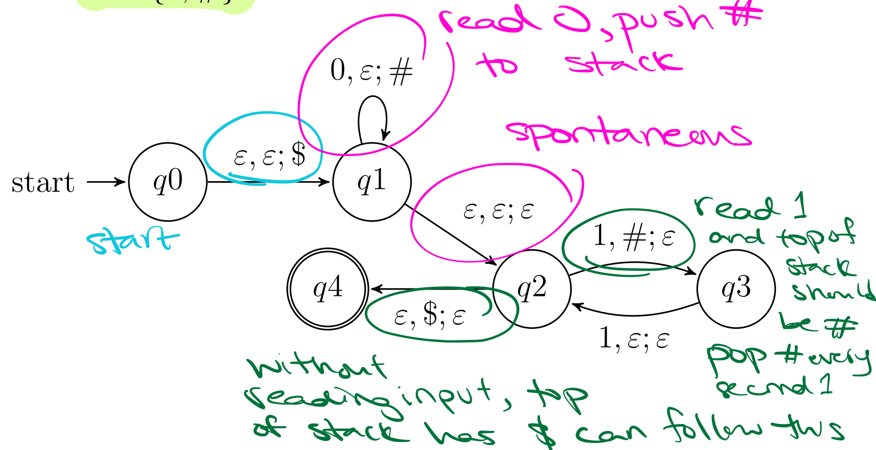
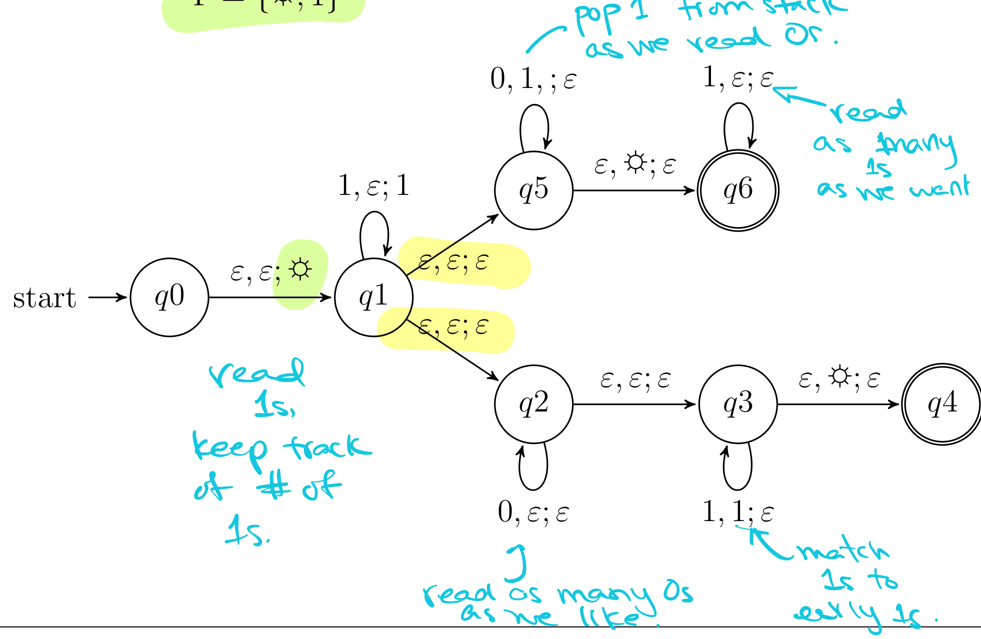
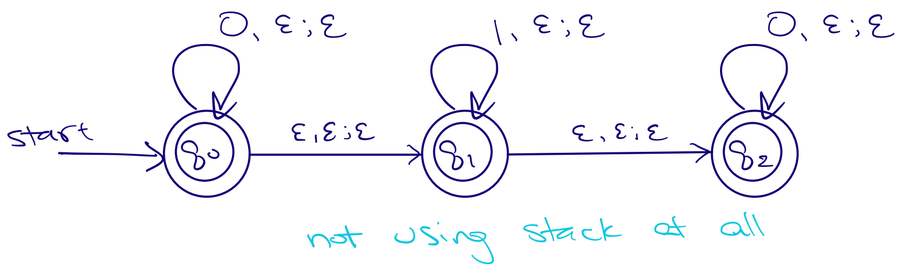
$\{w \in \{0,1\}^* \mid |w| \bmod 3 = 1\}$

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

$\Gamma = \{\$, \star\}$

Corollary: For each language, if there's an NFA N, $L(N)=L$, then there's a PDA M with $L(M)=L$.

For the PDA state diagrams below, $\Sigma = \{0,1\}$.

Mathematical description of language (subset of $\{0,1\}^*$)	State diagram of PDA recognizing language
ε in out q_0 q_1 q_2 q_4 \$ \$ $\{0^i 1^{2i} \mid i \geq 0\}$ nonregular	$\Gamma = \{\$, \#\}$ 
$\{1^n 0^n 1^m \mid n,m \geq 0\} \cup$ $\{1^n 0^m 1^n \mid n,m \geq 0\}$ nonregular	$\Gamma = \{☼, 1\}$ 
$\{0^i 1^j 0^k \mid i,j,k \geq 0\}$ regular	

Note: alternate notation is to replace ; with →

Friday: Context-free Grammars

Big picture: PDAs were motivated by wanting to add some memory of unbounded size to NFA. How do we accomplish a similar enhancement of regular expressions to get a syntactic model that is more expressive?

DFA, NFA, PDA: Machines process one input string at a time; the computation of a machine on its input string reads the input from left to right.

Regular expressions: Syntactic descriptions of all strings that match a particular pattern; the language described by a regular expression is built up recursively according to the expression's syntax

Context-free grammars: Rules to produce one string at a time, adding characters from the middle, beginning, or end of the final string as the derivation proceeds.

Definitions below are on pages 101-102.

Term	Typical symbol or **Notation**	Meaning
Context-free grammar (CFG)	G	$G = (V, \Sigma, R, S)$
The set of **variables**	V	Finite set of symbols that represent phases in production pattern
The set of **terminals** derivation ends when only terminals	Σ	Alphabet of symbols of strings generated by CFG $V \cap \Sigma = \emptyset$
The set of **rules**	R	Each rule is $A \to u$ with $A \in V$ and $u \in (V \cup \Sigma)^*$
The **start** variable	S	Usually on left-hand-side of first/ topmost rule
Derivation	$S \Rightarrow \cdots \Rightarrow w$	Sequence of substitutions in a CFG (also written $S \Rightarrow^* w$). At each step, we can apply one rule to one occurrence of a variable in the current string by substituting that occurrence of the variable with the right-hand-side of the rule. The derivation must end when the current string has only terminals (no variables) because then there are no instances of variables to apply a rule to.
Language **generated** by the context-free grammar G	$L(G)$	The set of strings for which there is a derivation in G. Symbolically: $\{w \in \Sigma^* \mid S \Rightarrow^* w\}$ i.e. $\{w \in \Sigma^* \mid$ there is derivation in G that ends in $w\}$
Context-free language		A language that is the language generated by some context-free grammar

$G = (V, \Sigma, R, S)$ V is set of variables Σ is set of terminals R is set of rules S is start variable

Examples of context-free grammars, derivations in those grammars, and the languages generated by those grammars

$G_1 = (\{S\}, \{0\}, R, S)$ with rules $\overline{R}$ $R = \{S \to 0S, S \to 0\}$

$$\begin{aligned} &(1)\quad S \to 0S \\ &(2)\quad S \to 0 \end{aligned}$$

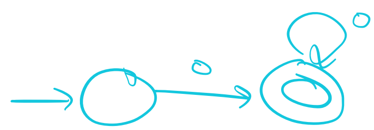
In $L(G_1)$ … need derivation $S \Rightarrow \ldots \Rightarrow \underbrace{}$ string of just terminals

$S \overset{(2)}{\Longrightarrow} 0$ witnesses that $0 \in L(G_1)$

$S \overset{(1)}{\Longrightarrow} 0S \overset{(1)}{\Longrightarrow} 00S \overset{(2)}{\Longrightarrow} 000$ witnesses that $000 \in L(G_1)$

$L(G_1) = \{0^i \mid i > 0\}$ Note: $\varepsilon \notin L(G_1)$

DFA recognizing $L(G_1)$



regular expression describing $L(G_1)$ 00^*

Not in $L(G_1)$ …

ε

$\varepsilon \notin \{0,1\}$

$R = \{ S \to 0S, S \to 1S, S \to \varepsilon \}$

$G_2 = (\{S\}, \{0,1\}, R, S)$

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

In $L(G_2)$ ...

$S \Rightarrow \varepsilon$ ← string (of length 0) of terminals.

$S \Rightarrow 0S \Rightarrow 0$

$S \Rightarrow 1S \Rightarrow 1$

$S \Rightarrow 0S \Rightarrow 01S \Rightarrow 011S \Rightarrow 0111S \Rightarrow 0111$

Not in $L(G_2)$ ...

NOPE

$L(G_2) = \{0,1\}^*$

$G_3 = (\{S,T\}, \{0,1\}, R, S)$ with rules

$$S \to T1T1T1T$$
$$T \to 0T \mid 1T \mid \varepsilon$$

T serves as a placeholder for "any string"

In $L(G_3)$ ...

$S \Rightarrow T1T1T1T \Rightarrow T11T1T \Rightarrow T11T1 \Rightarrow 0T11T1 \Rightarrow 00T11T1 \Rightarrow 0011T1 \Rightarrow 00111$

Not in $L(G_3)$ ...

$L(G_3) = \{ w \in \{0,1\}^* \mid w \text{ has at least three 1s} \}$.

$(\{S\}, \{0,1\}, R, S)$ where rules are $S \to 0S \mid S1 \mid \varepsilon$

$S \Rightarrow 0S \Rightarrow 0S1 \Rightarrow 00S1 \Rightarrow 000S1 \Rightarrow 0001$

generates the languages $\{ 0^i 1^j \mid i,j \geq 0 \}$

$G_4 = (\{A, B\}, \{0, 1\}, R, A)$ with rules

$$A \to 0A0 \mid 0A1 \mid 1A0 \mid 1A1 \mid 1$$

In $L(G_4)$ ...

$A \stackrel{2}{\Rightarrow} 0A1 \stackrel{3}{\Rightarrow} 01A01 \stackrel{5}{\Rightarrow} 01101$

Not in $L(G_4)$ ...

0

or 1

or 00

$L(G_4) = \{ w \in \{0,1\}^* \mid$ w has odd length and middle character is 1 $\}$

Design a CFG to generate the language $\{a^n b^n \mid n \geq 0\}$

$(\{S\}, \{a, b\}, R, S)$

where rules are $S \to \overset{①}{aSb} \mid \overset{②}{\varepsilon}$

Sample derivation:

$S \overset{①}{\Rightarrow} aSb \overset{①}{\Rightarrow} aaSbb \overset{①}{\Rightarrow} aaaSbbb \overset{②}{\Rightarrow} aaabbb$