

## Week 5 at a glance

**Textbook reading: Section 2.2, 2.1.**

Before Monday, read Theorem 2.20.

Before Wednesday, read Example 2.18 (page 114).

Before Friday, read Figure 3.1.

For Week 6 Monday: Page 165-166 Introduction to Section 3.1.

**We will be learning and practicing to:**

- Clearly and unambiguously communicate computational ideas using appropriate formalism. Translate across levels of abstraction.
  - Describe and use models of computation that don't involve state machines.
    - \* **Use context-free grammars and relate them to languages and pushdown automata.**
  - Use precise notation to formally define the state diagram of a Turing machine
  - Use clear English to describe computations of Turing machines informally.
    - \* **Design a PDA that recognizes a given language.**
  - Give examples of sets that are context-free (and prove that they are).
    - \* **State the definition of the class of context-free languages**
    - \* **Explain the limits of the class of context-free languages**
    - \* **Identify some context-free sets and some non-context-free sets**
- Know, select and apply appropriate computing knowledge and problem-solving techniques. Reason about computation and systems.
  - Describe and closure properties of classes of languages under certain operations.
    - \* **Apply a general construction to create a new PDA or CFG from an example one.**
    - \* **Formalize a general construction from an informal description of it.**
    - \* **Use general constructions to prove closure properties of the class of context-free languages.**
    - \* **Use counterexamples to prove non-closure properties of the class of context-free languages.**

## 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>)

Review Quiz 5 on PrairieLearn (<http://us.prairielearn.com>), complete by Sunday 11/4/2024

$$G = (V, \Sigma, R, S) \quad L(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}$$

start variable

## Monday: Context-free languages

Warmup: Design a CFG to generate the language  $\{a^i b^j \mid j \geq i \geq 0\}$  over  $\{a, b\}$

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

where  $R$  is given by

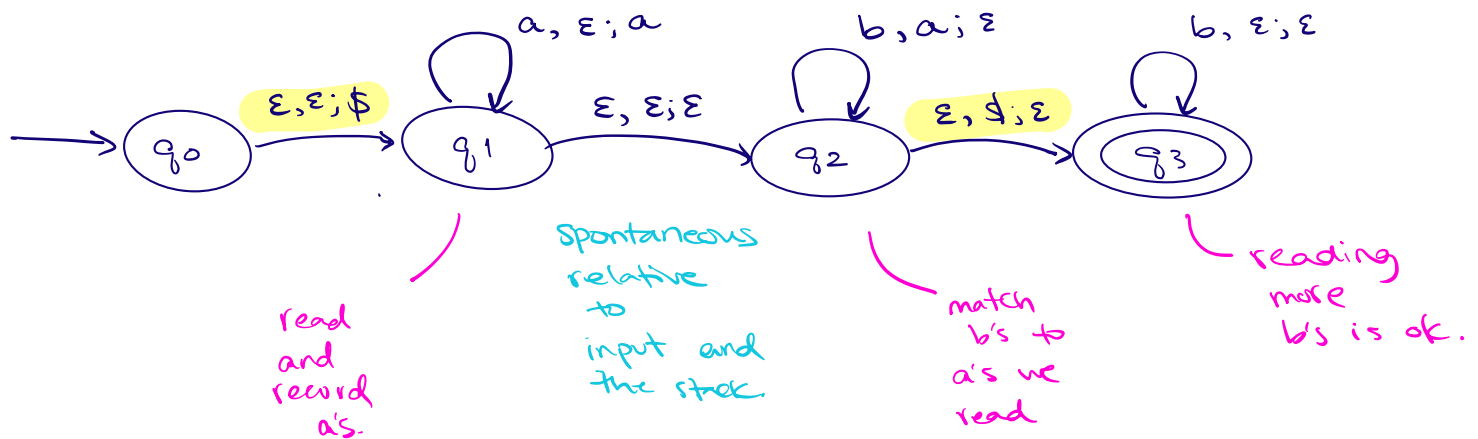
$$S \rightarrow \overset{(1)}{\varepsilon} \mid \overset{(2)}{a S b} \mid \overset{(3)}{S b}$$

Remember: to generate  $\{a^n b^n \mid n \geq 0\}$ , can use  
 $(\{S\}, \{a, b\}, \{S \rightarrow a S b \mid \varepsilon\}, S)$

Sample derivation:

$$\begin{aligned} S &\overset{(1)}{\Rightarrow} \varepsilon && \text{good bc } \varepsilon = a^0 b^0 \quad i=j=0 \\ S &\overset{(3)}{\Rightarrow} S b \overset{(3)}{\Rightarrow} S b b \overset{(2)}{\Rightarrow} a S b b \overset{(1)}{\Rightarrow} a b b b \\ &&& \text{good by } abbb = a^1 b^3 \quad i=1 \quad j=3 \end{aligned}$$

Design a PDA to recognize the language  $\{a^i b^j \mid j \geq i \geq 0\}$



Trace

ab	$\frac{}{q_0}$	$\frac{\$}{q_1}$	$\frac{a}{q_1}$	$\frac{a}{q_2}$	$\frac{\$}{q_2}$	$\frac{}{q_3}$
$\varepsilon$	$\frac{}{q_0}$	$\frac{\$}{q_1}$	$\frac{\$}{q_2}$	$\frac{}{q_3}$		
abb						
aba						

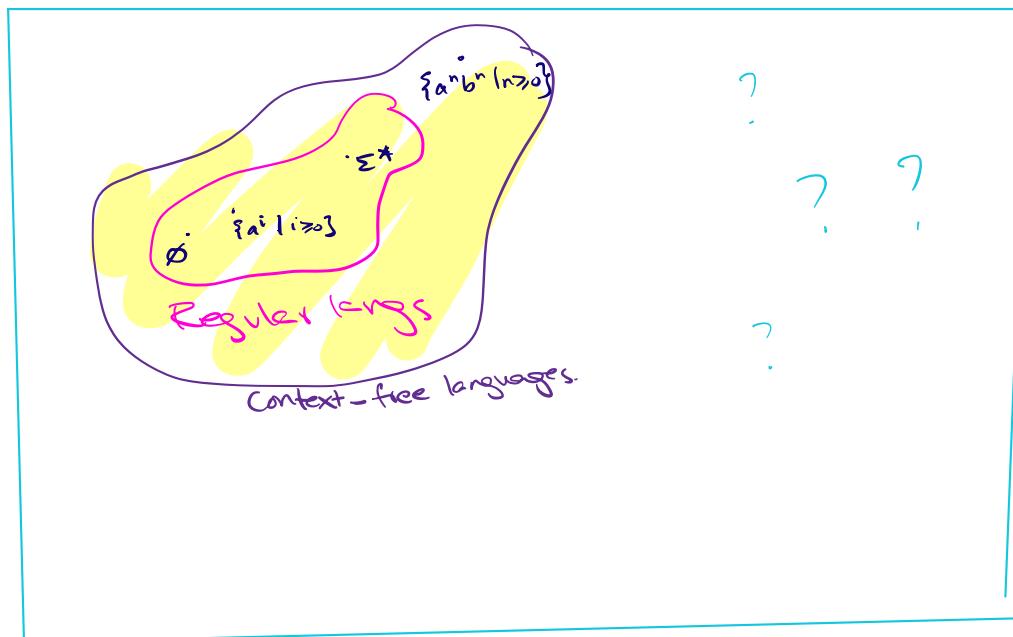
**Theorem 2.20:** A language is generated by some context-free grammar if and only if it is recognized by some push-down automaton.

Definition: a language is called **context-free** if it is the language generated by a context-free grammar. The class of all context-free language over a given alphabet  $\Sigma$  is called **CFL**.

Consequences:

- Quick proof that every regular language is context free
- To prove closure of the class of context-free languages under a given operation, we can choose either of two modes of proof (via CFGs or PDAs) depending on which is easier
- To fully specify a PDA we could give its 6-tuple formal definition or we could give its input alphabet, stack alphabet, and state diagram. An informal description of a PDA is a step-by-step description of how its computations would process input strings; the reader should be able to reconstruct the state diagram or formal definition precisely from such a description. The informal description of a PDA can refer to some common modules or subroutines that are computable by PDAs:
  - PDAs can “test for emptiness of stack” without providing details. *How?* We can always push a special end-of-stack symbol,  $\$,$  at the start, before processing any input, and then use this symbol as a flag.
  - PDAs can “test for end of input” without providing details. *How?* We can transform a PDA to one where accepting states are only those reachable when there are no more input symbols.

start w/ a NFA recognizing a language. Define PDA simulates it, recognizes the same language.  
 start w/ DFA recognizing a language. Define CFG generates language



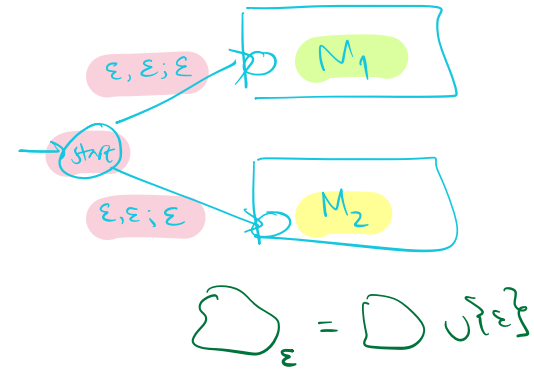
$\mathcal{P}(\Sigma^*)$   
 the set of  
 all languages  
 over  $\Sigma$

Suppose  $L_1$  and  $L_2$  are context-free languages over  $\Sigma$ . **Goal:**  $L_1 \cup L_2$  is also context-free.

### Approach 1: with PDAs

Let  $M_1 = (Q_1, \Sigma, \Gamma_1, \delta_1, q_1, F_1)$  and  $M_2 = (Q_2, \Sigma, \Gamma_2, \delta_2, q_2, F_2)$  be PDAs with  $L(M_1) = L_1$  and  $L(M_2) = L_2$ .

Define  $M = (Q, \Sigma, \Gamma, \delta, q_{\text{new}}, F_1 \cup F_2)$



Assume  $Q_1 \cap Q_2 = \emptyset$

and  $q_{\text{new}} \notin Q_1 \cup Q_2$

Define  $Q = Q_1 \cup Q_2 \cup \{q_{\text{new}}\}$

$\Gamma = \Gamma_1 \cup \Gamma_2$

$\delta: Q \times \Sigma_\epsilon \times \Gamma_\epsilon \rightarrow \mathcal{P}(Q \times \Gamma_\epsilon)$

power set

$$\delta((q, a, b)) = \begin{cases} \{(q_1, \epsilon), (q_2, \epsilon)\} \\ \delta_1((q, a, b)) \\ \delta_2((q, a, b)) \\ \emptyset \end{cases}$$

$q = q_{\text{new}}, a = \epsilon, b = \epsilon$

$q \in Q_1, a \in \Sigma_\epsilon, b \in \Gamma_{1,\epsilon}$

$q \in Q_2, a \in \Sigma_\epsilon, b \in \Gamma_{2,\epsilon}$

otherwise

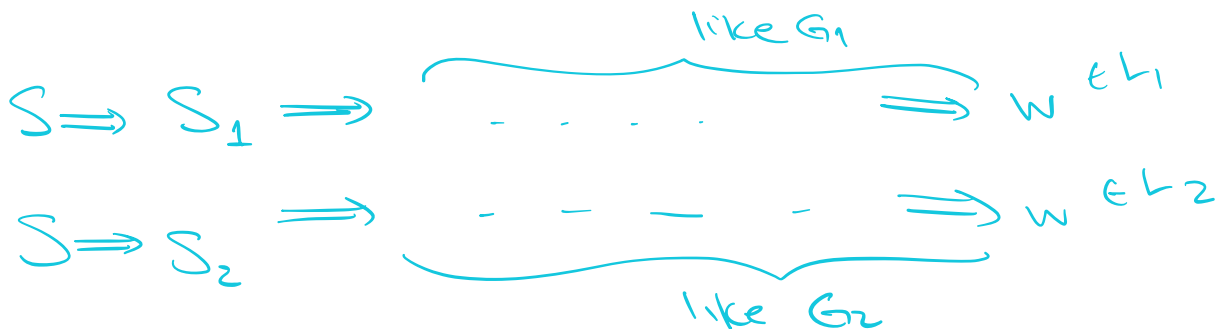
### Approach 2: with CFGs

Let  $G_1 = (V_1, \Sigma, R_1, \underline{S_1})$  and  $G_2 = (V_2, \Sigma, R_2, \underline{S_2})$  be CFGs with  $L(G_1) = L_1$  and  $L(G_2) = L_2$ .

Define  $G = (V_1 \cup V_2 \cup \{S\}, \Sigma, R_1 \cup R_2 \cup \{S \rightarrow S_1, S \rightarrow S_2\}, S)$

Assume  $V_1 \cap V_2 = \emptyset$

and  $S \notin V_1 \cup V_2$



Suppose  $L_1$  and  $L_2$  are context-free languages over  $\Sigma$ . **Goal:**  $L_1 \circ L_2$  is also context-free.

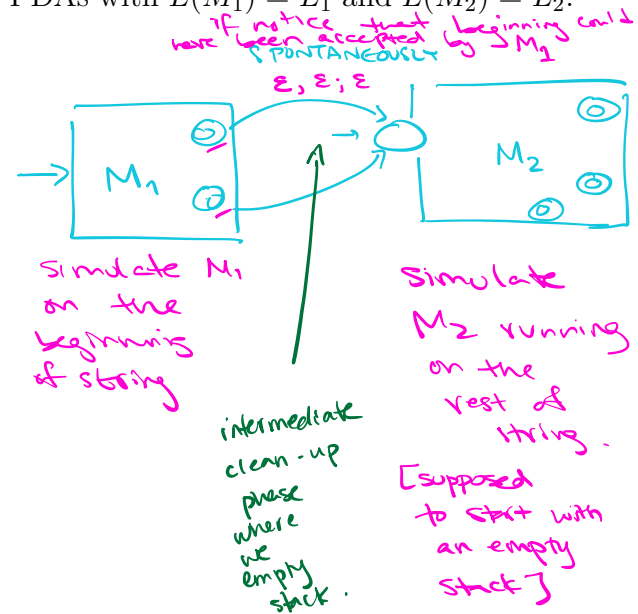
Approach 1: with PDAs

Try modifying union construction...

Let  $M_1 = (Q_1, \Sigma, \Gamma_1, \delta_1, q_1, F_1)$  and  $M_2 = (Q_2, \Sigma, \Gamma_2, \delta_2, q_2, F_2)$  be PDAs with  $L(M_1) = L_1$  and  $L(M_2) = L_2$ .

Define  $M =$

Details in homework 4.



Approach 2: with CFGs

Try modifying union construction...

Let  $G_1 = (V_1, \Sigma, R_1, S_1)$  and  $G_2 = (V_2, \Sigma, R_2, S_2)$  be CFGs with  $L(G_1) = L_1$  and  $L(G_2) = L_2$ .

Define  $G = (\underline{V_1 \cup V_2 \cup \{S\}}, \underline{\Sigma}, R, S)$

Assume  $V_1 \cap V_2 = \emptyset$

$S \notin V_1 \cup V_2$

$S \Rightarrow S_1 S_2$

first part can be derived in  $G_1$  second part can be derived in  $G_2$

where  $R = R_1 \cup R_2 \cup \{S \rightarrow S_1 S_2\}$

## Wednesday: Context-free and non-context-free languages

### Summary

Over a fixed alphabet  $\Sigma$ , a language  $L$  is **regular**

iff it is described by some regular expression

iff it is recognized by some DFA

iff it is recognized by some NFA

details of algorithms.

Over a fixed alphabet  $\Sigma$ , a language  $L$  is **context-free**

iff it is generated by some CFG

iff it is recognized by some PDA

**Fact:** Every regular language is a context-free language.

For each  $L$ , if there is  $M$  is a NFA and  $L(M)=L$ , then we can build PDA  $\tilde{M}$  with same state diagram (ignoring stack) recognizing same language

**Fact:** There are context-free languages that are not nonregular.

ie. regular

Yes, in fact all regular languages are context-free and there's at least one regular language.

**Fact:** There are context-free languages that are nonregular.  
Yes, witness e.g.  $\{0^n 1^n \mid n \geq 0\}$ .

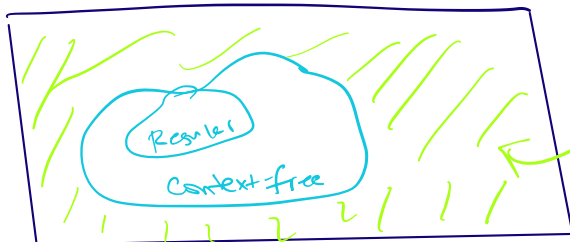
**Fact:** There are countably many regular languages.

Because regular expressions are strings

**Fact:** There are countably infinitely many context-free languages.

Because CFGs can be represented as strings

**Consequence:** Most languages are **not** context-free!

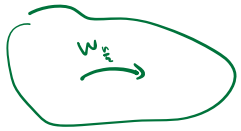


Uncountably many languages over  $\Sigma$  that are not context free

Sneak Peek: if we replace stack with a queue then get model of computation that is as powerful as Turing machine!

## Examples of non-context-free languages

$$W = W_1 W_2 \dots W_n$$



$$\{a^n b^n c^n \mid 0 \leq n, n \in \mathbb{Z}\}$$

$$\{a^i b^j c^k \mid 0 \leq i \leq j \leq k, i \in \mathbb{Z}, j \in \mathbb{Z}, k \in \mathbb{Z}\}$$

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

Context-free

compare to  $\{wOw \mid w \in \{0,1\}^*\}$   
compare to  $\{ww^R \mid w \in \{0,1\}^*\}$   
*R is reverse*

(Sipser Ex 2.36, Ex 2.37, 2.38)

There is a Pumping Lemma for CFL that can be used to prove a specific language is non-context-free: If  $A$  is a context-free language, there is a number  $p$  where, if  $s$  is any string in  $A$  of length at least  $p$ , then  $s$  may be divided into five pieces  $s = uvxyz$  where (1) for each  $i \geq 0$ ,  $uv^i xy^i z \in A$ , (2)  $|uv| > 0$ , (3)  $|vxy| \leq p$ . We will not go into the details of the proof or application of Pumping Lemma for CFLs this quarter.

Recall: A set  $X$  is said to be **closed** under an operation  $OP$  if, for any elements in  $X$ , applying  $OP$  to them gives an element in  $X$ .

## Extra Practice

True/False	Closure claim
True	The set of integers is closed under multiplication. $\forall x \forall y ( (x \in \mathbb{Z} \wedge y \in \mathbb{Z}) \rightarrow xy \in \mathbb{Z} )$
True	For each set $A$ , the power set of $A$ is closed under intersection. $\forall A_1 \forall A_2 ( (A_1 \in \mathcal{P}(A) \wedge A_2 \in \mathcal{P}(A)) \rightarrow A_1 \cap A_2 \in \mathcal{P}(A) )$
True	The class of regular languages over $\Sigma$ is closed under complementation.
True	The class of regular languages over $\Sigma$ is closed under union.
True	The class of regular languages over $\Sigma$ is closed under intersection.
True	The class of regular languages over $\Sigma$ is closed under concatenation.
True	The class of regular languages over $\Sigma$ is closed under Kleene star.
FALSE	The class of context-free languages over $\Sigma$ is closed under complementation.
True	The class of context-free languages over $\Sigma$ is closed under union.
FALSE	The class of context-free languages over $\Sigma$ is closed under intersection.
True	The class of context-free languages over $\Sigma$ is closed under concatenation.
True	The class of context-free languages over $\Sigma$ is closed under Kleene star.

## Friday: Turing machines

We are ready to introduce a formal model that will capture a notion of general purpose computation.

- *Similar to DFA, NFA, PDA*: input will be an arbitrary string over a fixed alphabet.
- *Different from NFA, PDA*: machine is deterministic.
- *Different from DFA, NFA, PDA*: read-write head can move both to the left and to the right, and can extend to the right past the original input.
- *Similar to DFA, NFA, PDA*: transition function drives computation one step at a time by moving within a finite set of states, always starting at designated start state.
- *Different from DFA, NFA, PDA*: the special states for rejecting and accepting take effect immediately.

(See more details: Sipser p. 166)

Formally: a Turing machine is  $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$  where  $\delta$  is the **transition function**

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$$

The **computation** of  $M$  on a string  $w$  over  $\Sigma$  is:

- Read/write head starts at leftmost position on tape.
- Input string is written on  $|w|$ -many leftmost cells of tape, rest of the tape cells have the blank symbol. **Tape alphabet** is  $\Gamma$  with  $\sqcup \in \Gamma$  and  $\Sigma \subseteq \Gamma$ . The blank symbol  $\sqcup \notin \Sigma$ .
- Given current state of machine and current symbol being read at the tape head, the machine transitions to next state, writes a symbol to the current position of the tape head (overwriting existing symbol), and moves the tape head L or R (if possible).
- Computation ends **if and when** machine enters either the accept or the reject state. This is called **halting**. Note:  $q_{accept} \neq q_{reject}$ .

The **language recognized by the Turing machine**  $M$ , is  $L(M) = \{w \in \Sigma^* \mid w \text{ is accepted by } M\}$ , which is defined as

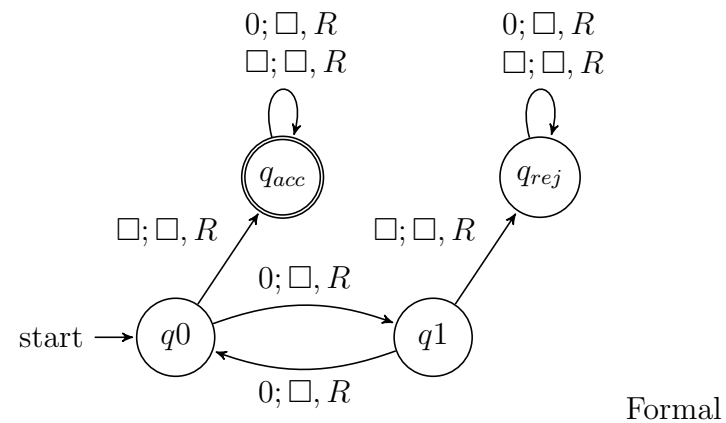
$$\{w \in \Sigma^* \mid \text{computation of } M \text{ on } w \text{ halts after entering the accept state}\}$$



definition:

Sample computation:

$q0 \downarrow$						
0	0	0	$\square$	$\square$	$\square$	$\square$

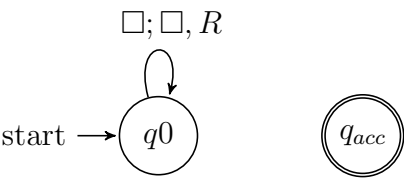


The language recognized by this machine is ...

**Describing Turing machines** (Sipser p. 185) To define a Turing machine, we could give a

- **Formal definition:** the 7-tuple of parameters including set of states, input alphabet, tape alphabet, transition function, start state, accept state, and reject state; or,
- **Implementation-level definition:** English prose that describes the Turing machine head movements relative to contents of tape, and conditions for accepting / rejecting based on those contents.
- **High-level description:** description of algorithm (precise sequence of instructions), without implementation details of machine. As part of this description, can “call” and run another TM as a subroutine.

Fix  $\Sigma = \{0, 1\}$ ,  $\Gamma = \{0, 1, \sqcup\}$  for the Turing machines with the following state diagrams:



Example of string accepted:  
Example of string rejected:

Implementation-level description

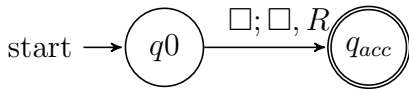
High-level description



Example of string accepted:  
Example of string rejected:

Implementation-level description

High-level description

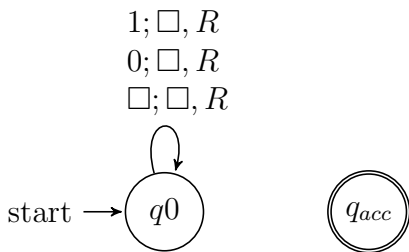


Example of string accepted:

Example of string rejected:

Implementation-level description

High-level description



Example of string accepted:

Example of string rejected:

Implementation-level description

High-level description