CONTEXT-FREE LANGUAGES

DISCRETE MATHEMATICS AND THEORY 2

MARK FLORYAN

GOALS!

1. Very quick review of the Chomsky Hierarchy (overall picture).

2. Our second model of computation, the pushdown automata!! Let’s add memory to that finite state machine!!

3. What languages can this new model of computation now recognize? Can we find languages it cannot recognize now?

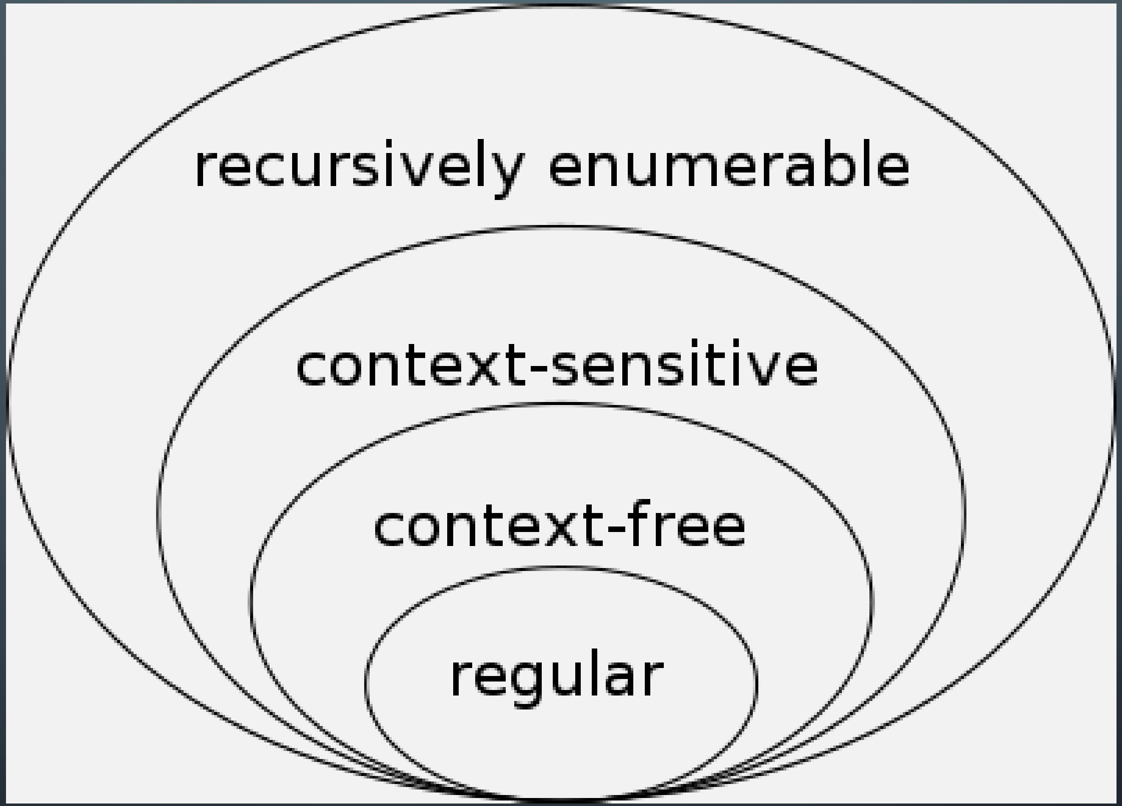
PART 1: REMINDER OF WHERE WE ARE / CHOMSKY HIERARCHY

TYPES OF PROBLEMS

Name	Decision Problem	Function	Language
XOR	Are there an odd number of 1's?	$f(b) = \begin{cases} 0 & \text{number of 1s is even} \\ 1 & \text{number of 1s is } odd \end{cases}$	$\{b \in \Sigma^* \| b \text{ has and odd number of 1s}\}$
Majority	Are there more 1s than 0s?	$f(b) = \begin{cases} 0 \text{ more 0s than 1s} \\ 1 \text{ more 1s than 0s} \end{cases}$	$\{b \in \Sigma^* \| b \text{ has more 1s than 0s}\}$
Thing you want to compute	Does it have have a property?	$f(b) = 1\ if\ it\ does\ have\ the\ property$	$\{b \in \Sigma^* \| b\ has\ the\ property\}$
Is1	Is the string exactly "1"?	$f(b) = 1\ if\ b == 1$	$\{1\}$
Is_infinite	Is the length of the string infinite?	$f(b) = 0$	$\emptyset$

CHOMSKY HIERARCHY

A description of languages and their relationship to one another



Each language has a computational model that recognizes it

In this deck, we will see the context-free languages and the machines that recognize them

OVERVIEW OF THIS DECK

These are all equivalent in their expressive power.

Pushdown Automata

NFA w/ a stack. Can recognize exactly the context-free languages

Context-Free Languages:

A Class of languages that are more expressive than regular languages

Context-Free Grammar:

A "string" description of a context-free language (by definition)

At the end of this deck, we will also see that context-free languages have a pumping lemma that can be used to prove some languages are NOT context-free.

For now, we allow non-determinism freely with this computational model. We will discuss (briefly) the ramifications for this later in the deck.

PART 2: CONTEXT-FREE GRAMMARS

INTRODUCTION: WHAT IS A CONTEXT-FREE GRAMMAR

QUICK ASIDE: FINITE AUTOMATA AND REGULAR LANGUAGES

***<u>Motivating Question</u>**: Computational models are often of interest in the application of programming languages and compilers. Given a program, is it a valid program in the given language that does not contain any syntax errors.*

Purple words do not need any "computation" to confirm. Just make sure the word matches something in a set of known valid keywords for types.

```
String x = "hellothere123";
double y = 23.456;
int z = 67.8;    //syntax error
```

Green words are perfect for a finite automata. Reg. Ex. is $\Sigma^*\backslash K$ where K is the set of reserved keywords.

Regular expressions really shine here. Each type has an automata that recognizes if string is in the valid format.

for String: "Σ^*"

for double: $[0-9]^+.[0-9]^+ \cup [0-9]^+$

for int: $[0-9]^+$

QUICK ASIDE: FINITE AUTOMATA AND REGULAR LANGUAGES

<u>Motivating Question</u>: *Computational models are often of interest in the application of programming languages and compilers. Given a program, is it a valid program in the given language that does not contain any syntax errors.*

```
String x = "hellothere123";
double y = 23.456;
int z = 67.8;    //syntax error
```

Can even handle some entire lines of code with finite automata. declarations of double:

$$double\ \Sigma^{+} = [0-9]^{+}.[0-9]^{+} \cup [0-9]^{+};$$

But in reality, code is more complicated than this. Consider:

- *the right side of assignment can be a bigger expression, or maybe result of an if statement.*
- *For loops can have nothing inside or a whole set of assignments.*

FORMAL DEFINITION OF A CFG

A context free grammar is:

A 4-tuple (V, Σ, R, S) where:

1. *V is a finite set called the* ***variables***
2. *Σ is a finite set, disjoint from V, called the* ***terminals***
3. *R is a finite set of* ***rules****, with each rule being a variable and a string of variables and terminals*
4. *$S \in V$ is the start variable*

SOME SIMPLE EXAMPLES

Alphabet for all these grammars:

$$\Sigma = \{0, 1, \ldots, 9, a, b, \ldots, z, ., +, -, =, ;\}$$

*Variables are denoted with capital letters. The first variable here is called the **start variable***

$$A = (\Sigma - \{., +, -, =, ;\})^*$$

$$S \rightarrow SS \mid 0 \mid 1 \mid \ldots \mid \epsilon$$

Terminal characters like these cannot be replaced, but ensure the expression will eventually be completed through enough substitutions

Each of these production rules can be applied to substitute for variables of the same name

Example:

Generate String "x10":

S

SS

SSS

xSS

x1S

x10

SOME SIMPLE EXAMPLES

Alphabet for all these grammars:

$$\Sigma = \{0, 1, \dots, 9, a, b, \dots, z, ., +, -, =, ;\}$$

A from previous slide

$$S \to SS \mid 0 \mid 1 \mid \dots \mid \epsilon$$

Repeated application of first rule to expand the size of the string then replace each S with individual characters.

$\mathrm{I} = d^+$

$$I \to II \mid 0 \mid \dots \mid 9 \mid$$

$\mathrm{D} = d^+ . d^+$

$$D \to I . I$$

$\mathrm{I}' =$ "int"

$$I' \to A_9 A_{14} A_{20}$$
$$A_9 \to i$$
$$A_{14} \to n$$
$$A_{20} \to t$$

$\mathrm{D}' =$ "double"

$$D' \to A_4 A_{15} A_5$$
$$A_4 \to d$$
$$A_{15} \to o$$
$$\dots$$
$$A_5 \to e$$

EXAMPLE CONTEXT-FREE GRAMMAR

$$V \rightarrow T\ N = E;$$
$$T \rightarrow I' \mid D'$$
$$N \rightarrow S$$
$$E \rightarrow C \mid E + E \mid E - E$$
$$C \rightarrow D \mid I$$

<u>Recall from previous slides that</u>:

- I' and D' lead to "int" and "double"
- S leads to any string of numbers and letters (variable name)
- I and D lead to valid int and double values respectively

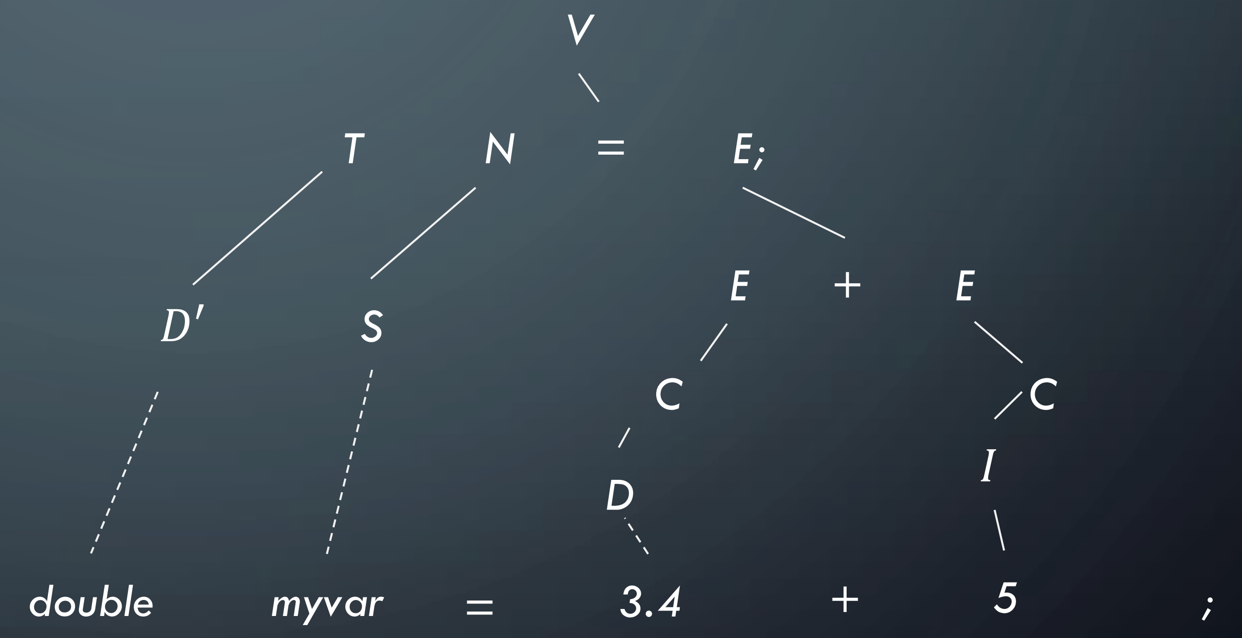
EXAMPLE CONTEXT-FREE GRAMMAR

$$V \rightarrow T\ N = E;$$
$$T \rightarrow I' \mid D'$$
$$N \rightarrow S$$
$$E \rightarrow C \mid E + E \mid E\ - E$$
$$C \rightarrow D \mid I$$

<u>*Example string in this grammar*</u>*:*

$$double\ myvar = 3.4 + 5;$$

<u>*Derivation of that string*</u>*:*



ANOTHER EXAMPLE CFG

We learned last time that the language $L = 0^n 1^n$ is NOT regular.

Can you generate a context-free grammar that recognizes it?

Hint: You do not need regular expressions as terminal characters here. Your terminal characters will be 0 and 1.

ANOTHER EXAMPLE CFG

We learned last time that the language $L = 0^n 1^n$ is NOT regular.

S → B

B → 0B1

B → ϵ

<u>Example Derivation of $0^3 1^3$</u>:

S

B

0B1

00B11

000B111

000111

CHALLENGE

Can you generate the following grammar on your own:

*The grammar of all well formed arithmetic expressions containing at most two variables (x and y), parentheses, and two operators (+ and *).*

Example string: (x+y)(x*x)*

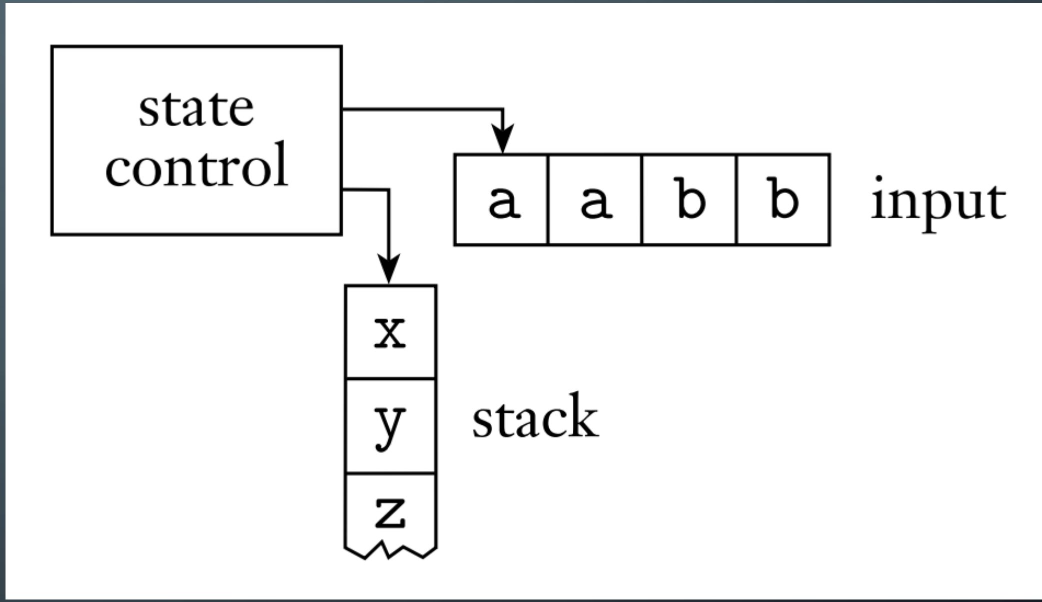
PART 2: PUSHDOWN AUTOMATA

PUSHDOWN AUTOMATA

This state control is just a DFA/NFA just like before!

Input is read in character by character (same as NFA)

<u>Pushdown Automata</u>: *Informally, is a machine that combines an NFA (state control) but adds a stack. The machine can push and pop to the stack*

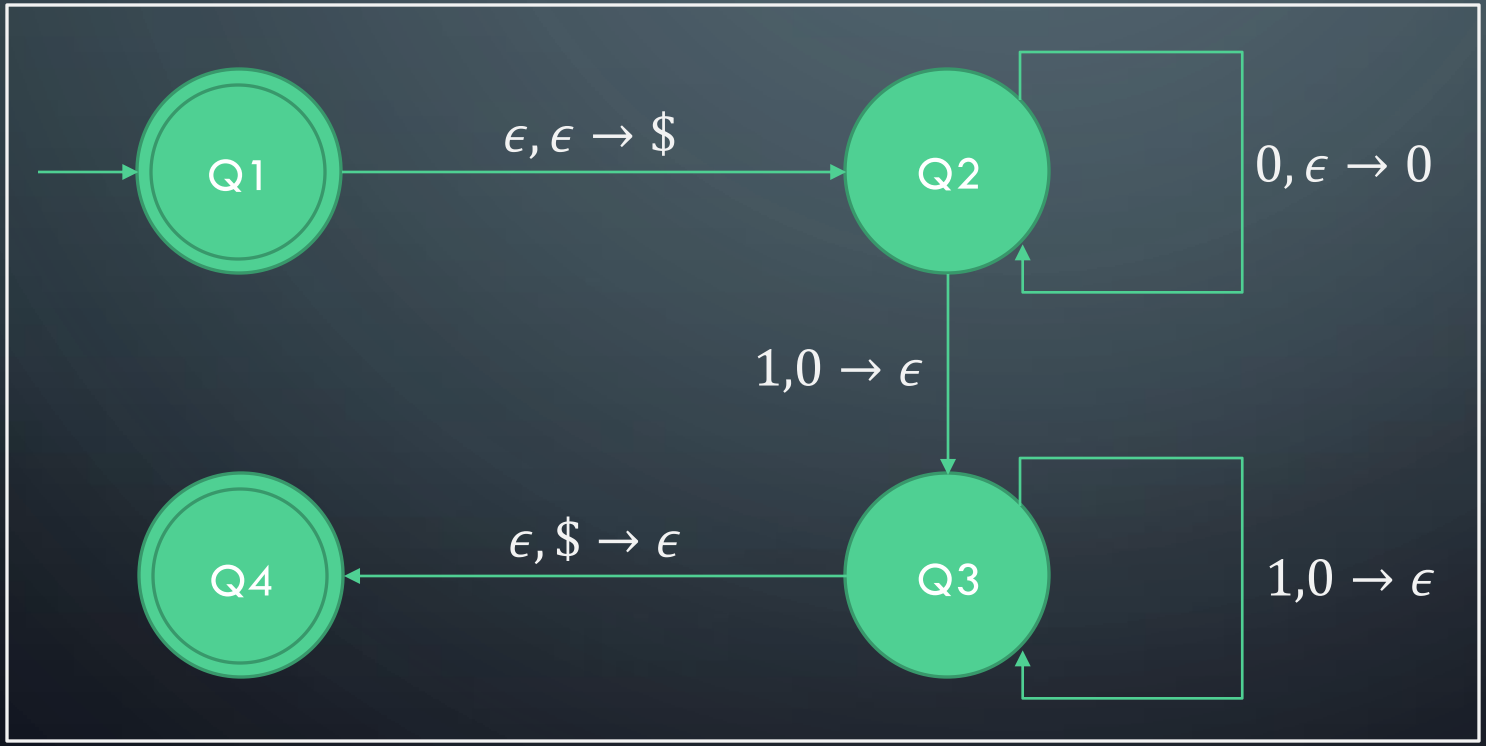


The **stack** provides the machine with memory. The machine can **push** or **pop** from the top of the stack only during any state transition (one push or pop per transition)

PUSHDOWN AUTOMATA

This pushdown automata recognizes the language:

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



Make sure you know how to read these transitions.

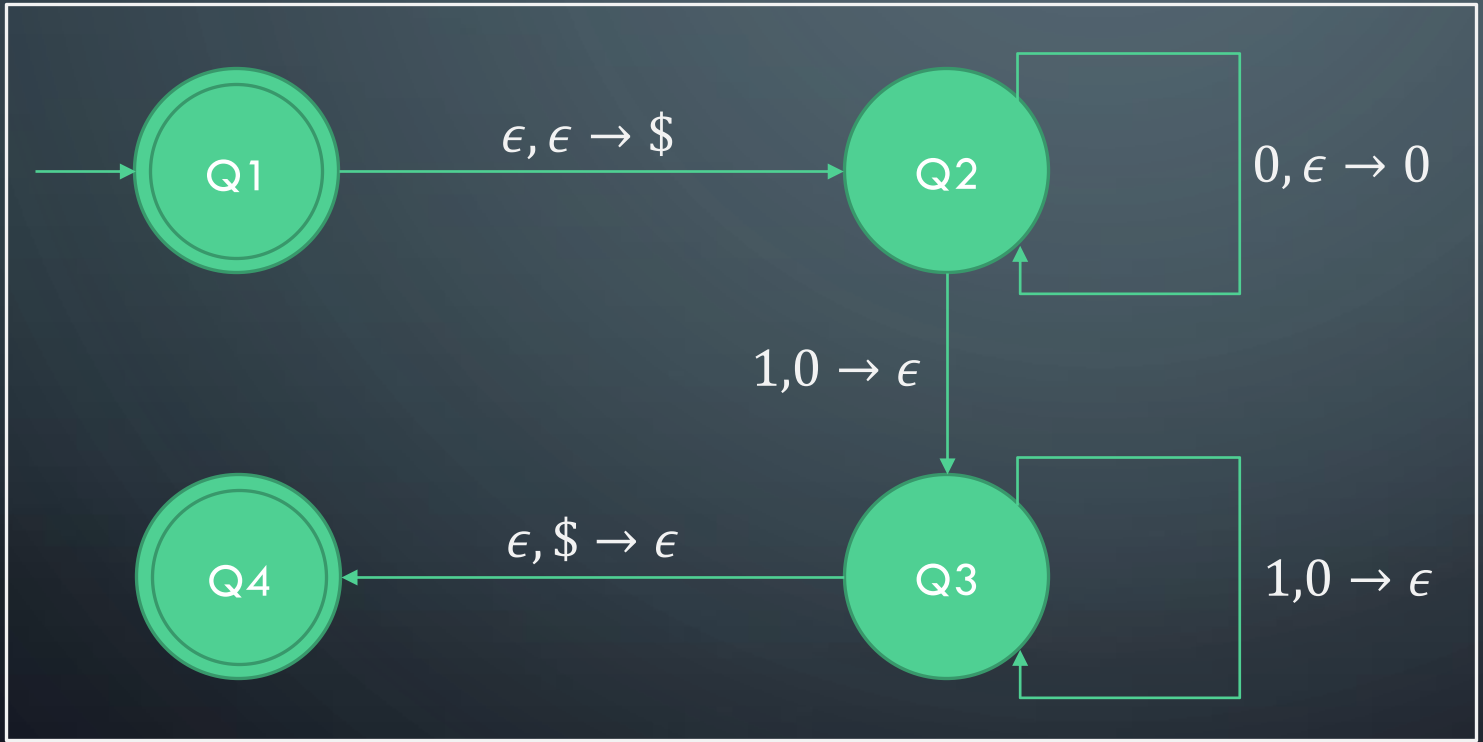
$1{,}0 \rightarrow \epsilon$

means if you read a 1 from input and a 0 is on top of stack (pop it), and don't push (epsilon) anything

PUSHDOWN AUTOMATA

This pushdown automata recognizes the language:

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



Let's step through w/ input:

000111

PUSHDOWN AUTOMATA

This pushdown automata recognizes the language:

$$L = a^i b^j c^k \mid i, j, k \geq 0 \ and \ i = j \ or \ i = k$$

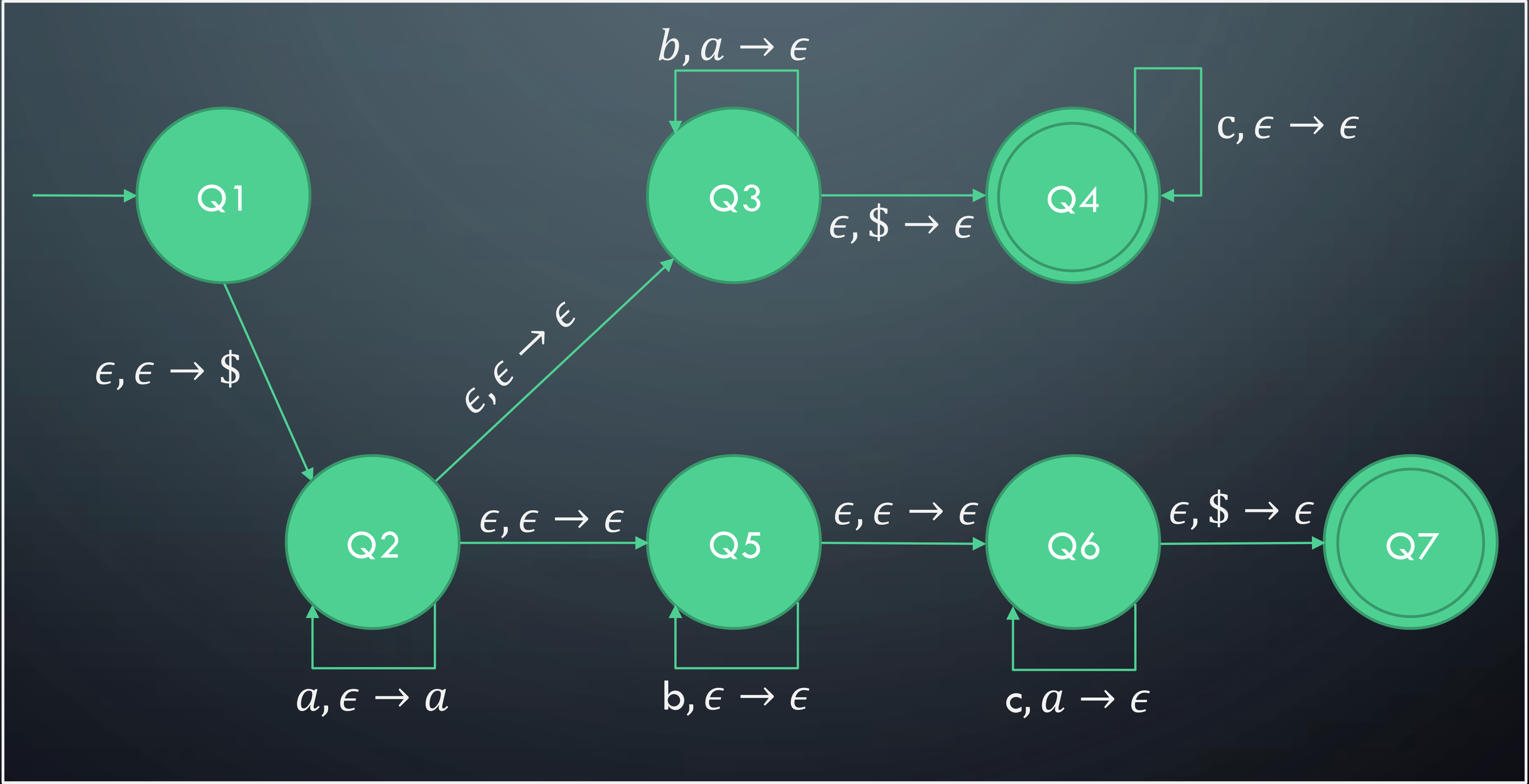
Basic Idea: Push the a's to the stack and then read them off to match them to either the b's or the c's

Do not be afraid to use non-determinism here. We don't really know whether to match the number of a's with the b's or the c's. Can we try both?

PUSHDOWN AUTOMATA

This pushdown automata recognizes the language:

$$L = a^i b^j c^k \mid i, j, k \geq 0 \; and \; i = j \; or \; i = k$$



FORMAL DEFINITION PUSHDOWN AUTOMATA

A **pushdown automata** is:

A 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where Q, Σ, Γ, and F are all finite sets and:

1. Q is the set of states
2. Σ is the input alphabet
3. Γ is the stack alphabet
4. $\delta: Q \; x \; \Sigma_\epsilon \; x \; \Gamma_\epsilon \to \mathcal{P}(Q \; x \; \Gamma_\epsilon)$ is the transition function
5. $q_0 \in Q$ is the start state
6. $F \subseteq Q$ is the set of accept states

TRY IT ON YOUR OWN!

Can you create PD that recognizes:

$$L = ww^R \mid where\ w^R\ is\ the\ string\ w\ reversed$$

EQUIVALENCE WITH CONTEXT-FREE GRAMMARS

YOU KNOW THE DRILL!

<u>Theorem</u>: A language L is context-free if and only if some pushdown automata recognizes it

How to prove:

Direction 1: Assume L is context-free, show how to construct the pushdown automata for it.

Direction 2: Assume we have a pushdown automata, show how to construct the context-free grammar that describes it.

YOU KNOW THE DRILL!

<u>Theorem</u>: A language L is context-free if and only if some pushdown automata recognizes it

Direction 1: Assume L is context-free, show how to construct the pushdown automata for it.

YOU KNOW THE DRILL!

Theorem: A language L is context-free if and only if some pushdown automata recognizes it

Direction 1: Assume L is context-free, show how to construct the pushdown automata for it.

High level idea of proof:

Let L be a context-free grammar, this means it can be described by a set of substitutions of variables / terminals (see formal definition of CFG)

To construct the PDA that recognizes it:

1. Put the start variable on the stack
2. Loop: Pop a variable off the stack, look at rules that can be substituted for, non-deterministically branch off for each one and put new symbol on the stack.
3. If terminal is on the stack, pop it and check. it against the next character of input.

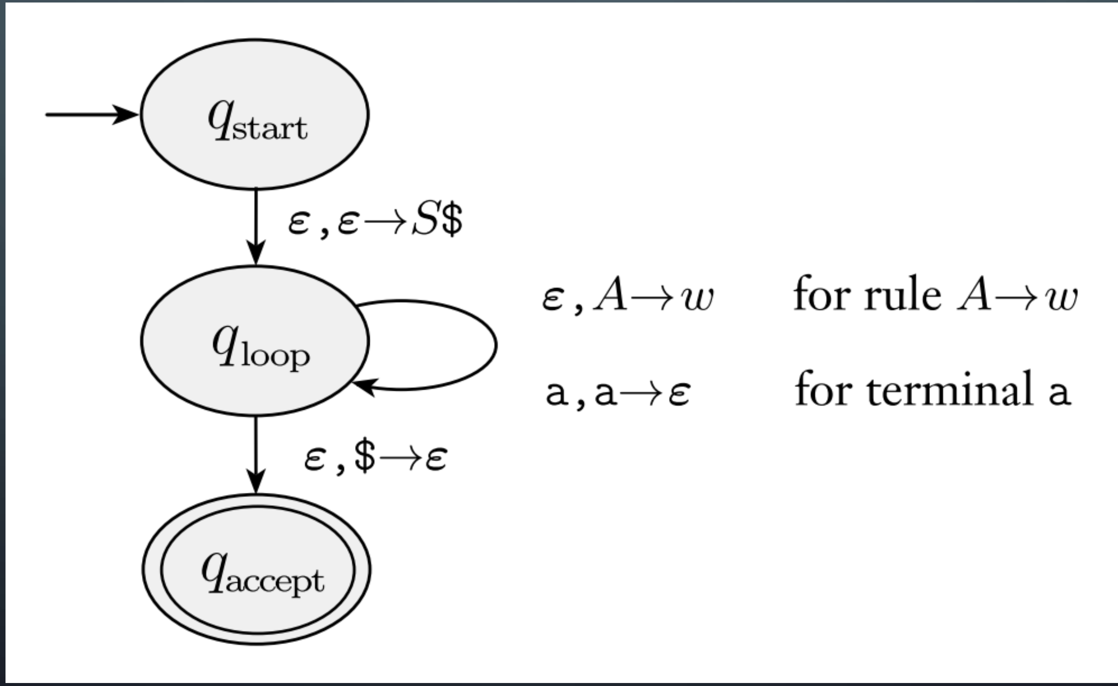
YOU KNOW THE DRILL!

Theorem: A language L is context-free if and only if some pushdown automata recognizes it

Direction 1: Assume L is context-free, show how to construct the pushdown automata for it.

Start by pushing $ onto the stack following by start variable S

This main loop has a non-deterministic condition for every possible rule substitution



if we see a variable A on top of stack (e.g.,), pop A off and push the things it can be substituted with onto stack one character at a time

If we see a terminal a on top of stack, pop it off and check it against the next character of input

YOU KNOW THE DRILL!

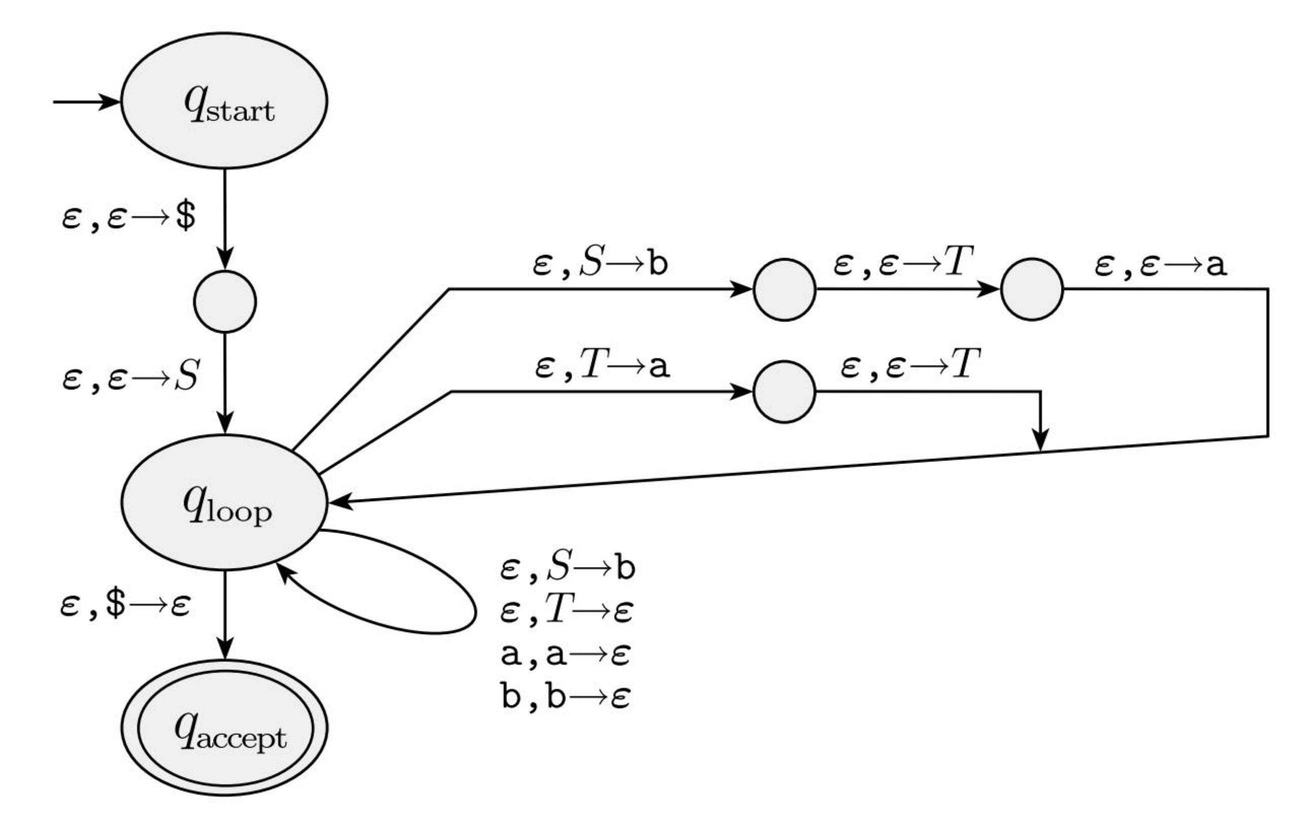
Theorem: A language L is context-free if and only if some pushdown automata recognizes it

Direction 1: Assume L is context-free, show how to construct the pushdown automata for it.

EXAMPLE:

Consider the grammar:

$$S \rightarrow aTb \mid b$$
$$T \rightarrow Ta \mid \epsilon$$



YOU KNOW THE DRILL!

<u>Theorem</u>: A language L is context-free if and only if some pushdown automata recognizes it

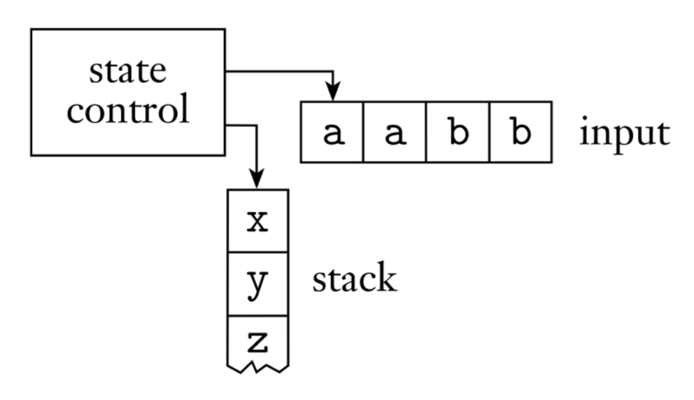
Direction 2: Assume we have a pushdown automata, show how to construct the context-free grammar that describes it.

YOU KNOW THE DRILL!

Theorem: A language L is context-free if and only if some pushdown automata recognizes it

Direction 2: Assume we have a pushdown automata, show how to construct the CFG that describes it.

Start with an arbitrary PDA, called P



Convert into

A grammar that is equivalent to P:

$S \rightarrow A_{rs}A_{sq}$

$A_{rs} \rightarrow \epsilon$

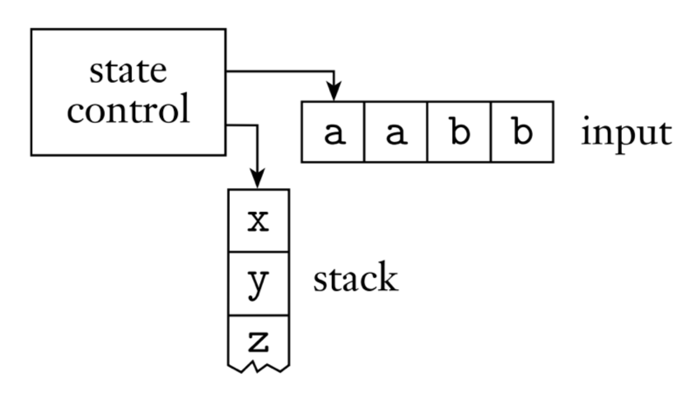
…

YOU KNOW THE DRILL!

Theorem: A language L is context-free if and only if some pushdown automata recognizes it

Direction 2: Assume we have a pushdown automata, show how to construct the CFG that describes it.

Start with an arbitrary PDA, called P



Step 1: *Let's simplify P a little bit so we know SOMETHING about it's structure.*

We make the following changes to P:

1. *If P has multiple accept states, change it to have only one*
2. *If P has elements on the stack before accepting, empty the stack first*
3. *Every transition either pushes a symbol, or pops a symbol, but not both*

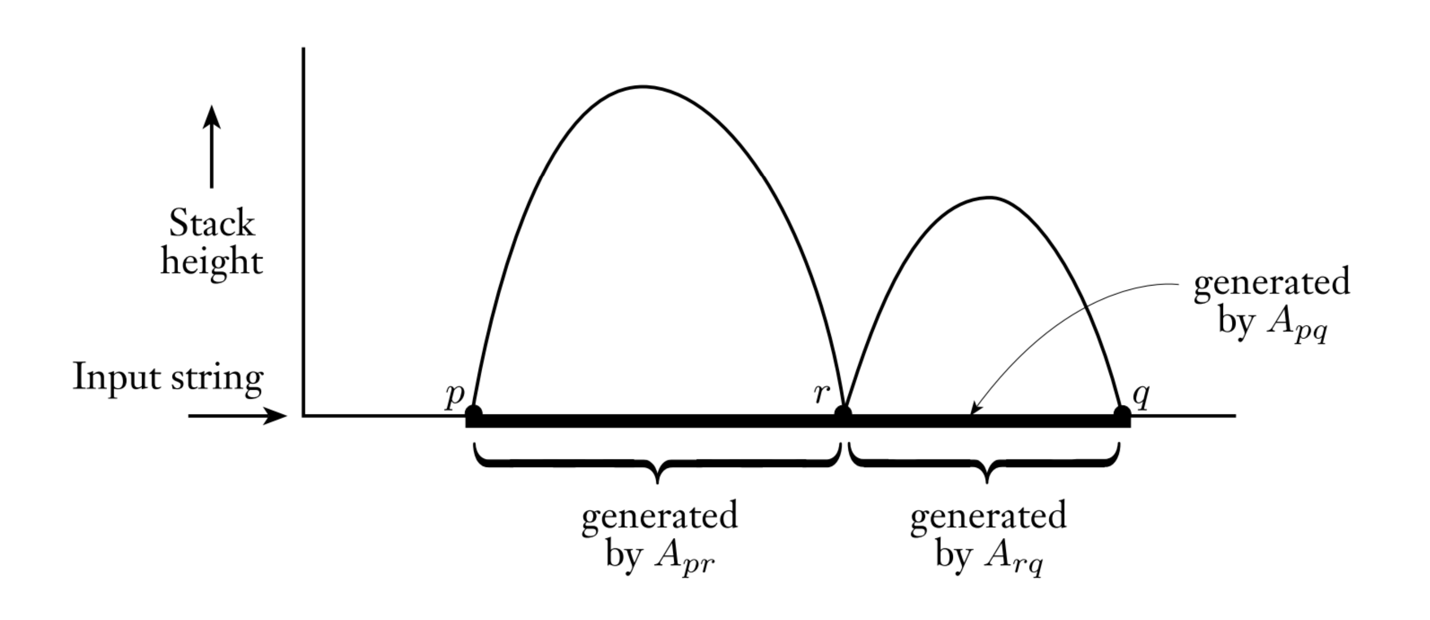
These changes are not too difficult to make. I will verbally describe how to do all 3 of these.

YOU KNOW THE DRILL!

Theorem: A language L is context-free if and only if some pushdown automata recognizes it

Direction 2: Assume we have a pushdown automata, show how to construct the CFG that describes it.

Overall idea: We need to get from start state (empty stack) to accept state (empty stack)



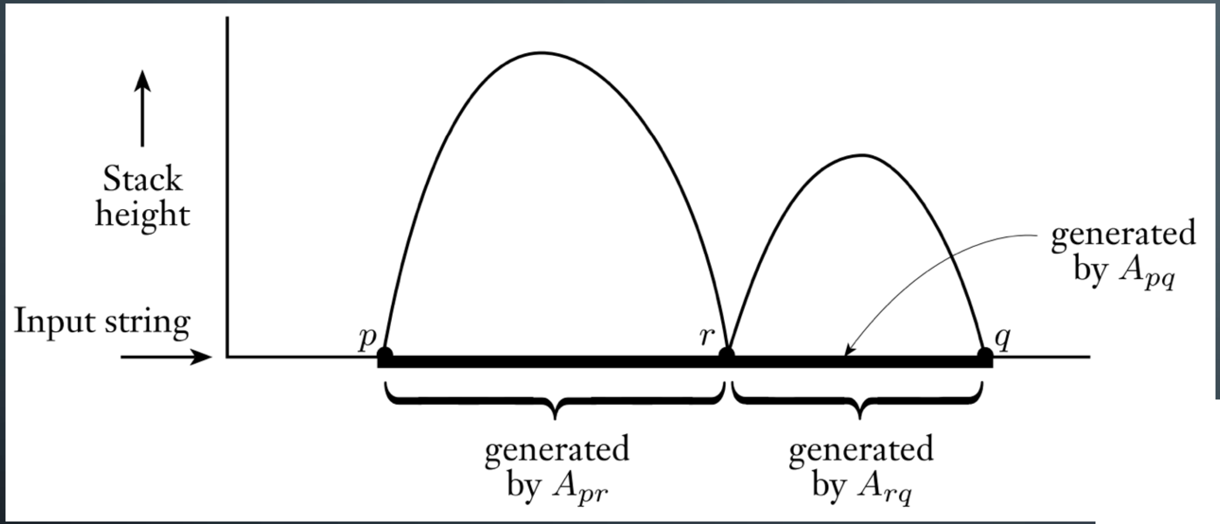
Variable A_{pq} represents moving from state p to state q without changing the state of the stack

Notice that in this case, we move from state p to r (without altering stack) and then again from r to q (without altering stack)

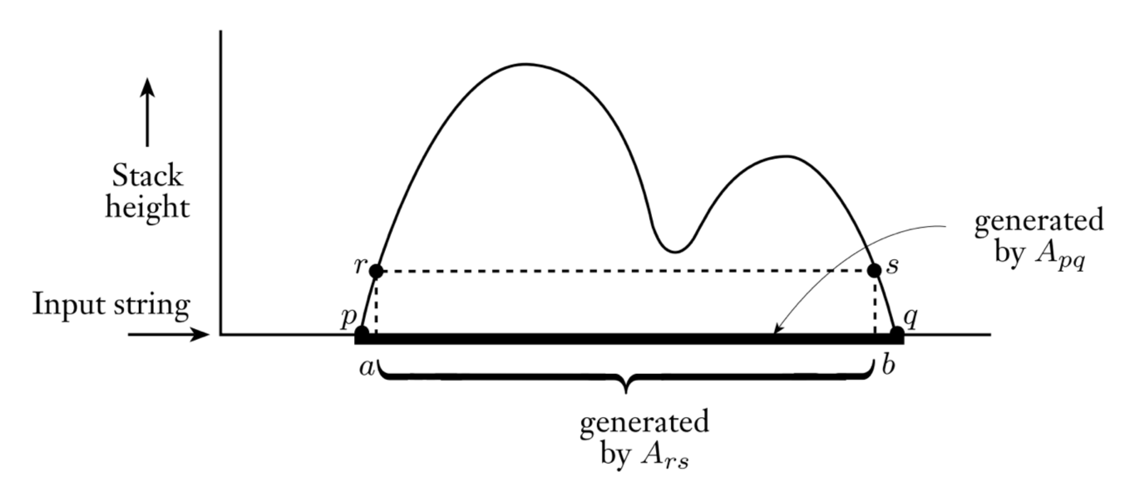
YOU KNOW THE DRILL!

Theorem: A language L is context-free if and only if some pushdown automata recognizes it

Direction 2: Assume we have a pushdown automata, show how to construct the CFG that describes it.



Situation can also look like this. Stack moves up and down but eventually comes back to being empty. The first symbol that is pushed must match the last symbol popped.



YOU KNOW THE DRILL!

Theorem: A language L is context-free if and only if some pushdown automata recognizes it

Direction 2: Assume we have a pushdown automata, show how to construct the CFG that describes it.

Let's start constructing the grammar from the PDA. The variables represent moving from each pair of states p,q by using, but not altering the stack (empty stack to empty stack).

start variable will simply be a dummy variable that represents getting from the start state to the accept state

Given a PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{accept}\})$, construct grammar G:

Variables of G are $\{A_{pq} \mid p, q \in Q\}$

Start variable is $A_{q_0\, q_{accept}}$

YOU KNOW THE DRILL!

Theorem: A language L is context-free if and only if some pushdown automata recognizes it

Direction 2: Assume we have a pushdown automata, show how to construct the CFG that describes it.

This next rule covers every pair of states the push and pop the same symbol (e.g., first step is to push a symbol t, then a bunch of stuff happens, then eventually we pop off the t).

Given a PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{accept}\})$, construct grammar G:

Variables of G are $\{A_{pq} \mid p, q \in Q\}$

Start variable is $A_{q_0\, q_{accept}}$

For each $p, q, r, s \in Q, t \in \Gamma$, and $a, b \in \Sigma_\epsilon$, if $\delta(p, a, \epsilon)$ contains (r, t) and $\delta(s, b, t)$ contains (q, ϵ), put the rule $\boldsymbol{A_{pq} \rightarrow aA_{rs}b}$ **into** $\boldsymbol{G}$

YOU KNOW THE DRILL!

Theorem: A language L is context-free if and only if some pushdown automata recognizes it

Direction 2: Assume we have a pushdown automata, show how to construct the CFG that describes it.

Given a PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{accept}\})$, construct grammar G:

Variables of G are $\{A_{pq} \mid p, q \in Q\}$

Start variable is $A_{q_0\, q_{accept}}$

For each $p, q, r, s \in Q, t \in \Gamma$, and $a, b \in \Sigma_\epsilon$, if $\delta(p, a, \epsilon)$ contains (r, t) and $\delta(s, b, t)$ contains (q, ϵ), put the rule $A_{pq} \rightarrow aA_{rs}b$ into G

For each $p, q, r \in Q$, put the rule $A_{pq} \rightarrow A_{pr}A_{rq}$ in G

This rule covers all cases where you empty the stack twice (at least). Once from state p to r and again from state r to q.

YOU KNOW THE DRILL!

Theorem: A language L is context-free if and only if some pushdown automata recognizes it

Direction 2: Assume we have a pushdown automata, show how to construct the CFG that describes it.

Last rule, base case! Nothing needs to happen when going from a state p to itself.

Given a PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{accept}\})$, construct grammar G:

Variables of G are $\{A_{pq} \mid p, q \in Q\}$

Start variable is $A_{q_0\, q_{accept}}$

For each $p, q, r, s \in Q, t \in \Gamma$, and $a, b \in \Sigma_\epsilon$, if $\delta(p, a, \epsilon)$ contains (r, t) and $\delta(s, b, t)$ contains (q, ϵ), put the rule $A_{pq} \rightarrow aA_{rs}b$ into G

For each $p, q, r \in Q$, put the rule $A_{pq} \rightarrow A_{pr}A_{rq}$ in G

Finally, for each $p \in Q$, put the rule $A_{pp} \rightarrow \epsilon$ in G

YOU KNOW THE DRILL!

Theorem: A language L is context-free if and only if some pushdown automata recognizes it

Direction 2: Assume we have a pushdown automata, show how to construct the CFG that describes it.

So, is it the case that if a string X accepts in the original automata P, then this grammar will definition accept it (and similarly for rejection)?

Yes, let's verbally describe why.

Given a PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{accept}\})$, construct grammar G:

Variables of G are $\{A_{pq} \mid p, q \in Q\}$

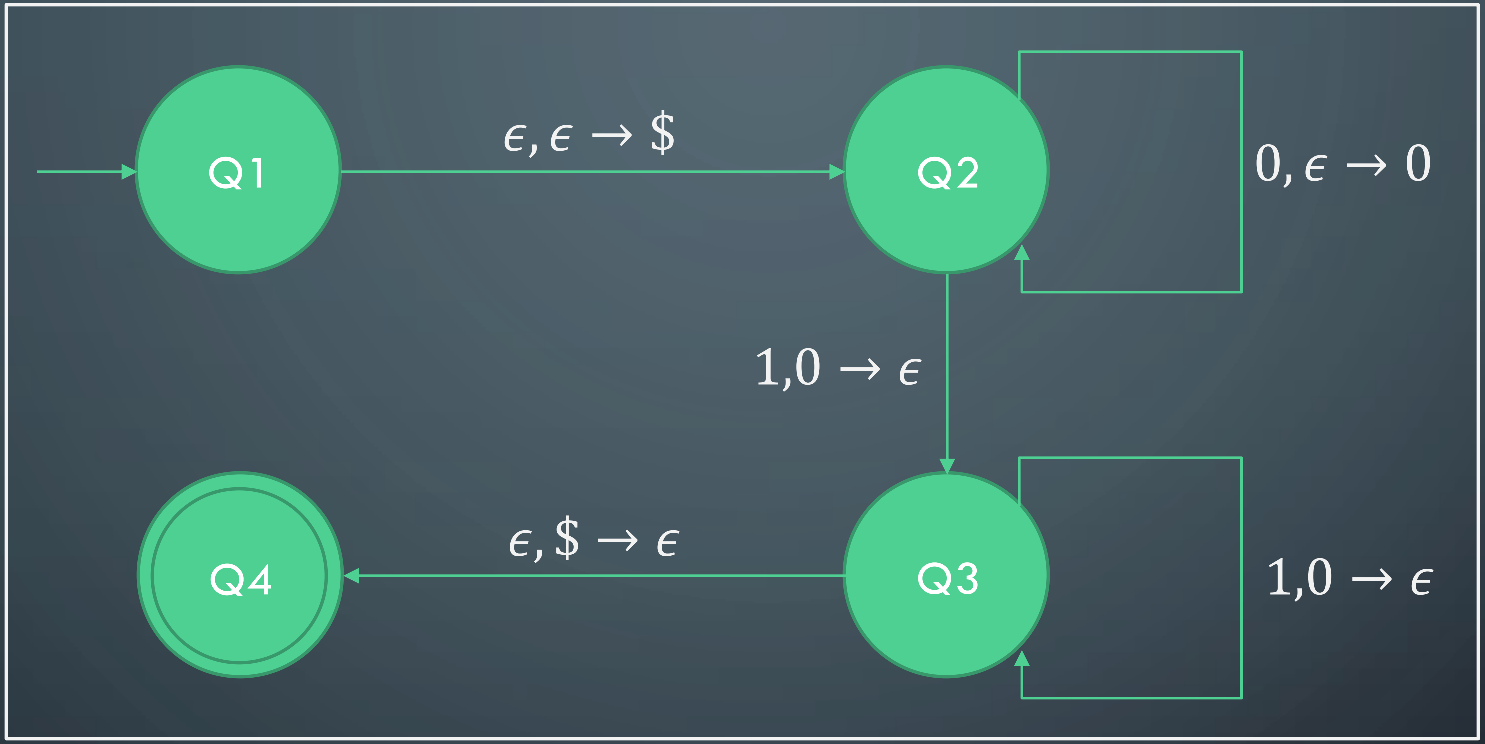
Start variable is $A_{q_0\, q_{accept}}$

For each $p, q, r, s \in Q, t \in \Gamma$, and $a, b \in \Sigma_\epsilon$, if $\delta(p, a, \epsilon)$ contains (r, t) and $\delta(s, b, t)$ contains (q, ϵ), put the rule $A_{pq} \rightarrow aA_{rs}b$ into G

For each $p, q, r \in Q$, put the rule $A_{pq} \rightarrow A_{pr}A_{rq}$ in G

Finally, for each $p \in Q$, put the rule $A_{pp} \rightarrow \epsilon$ in G

EXAMPLE: $0^n1^n \mid n \geq 1$

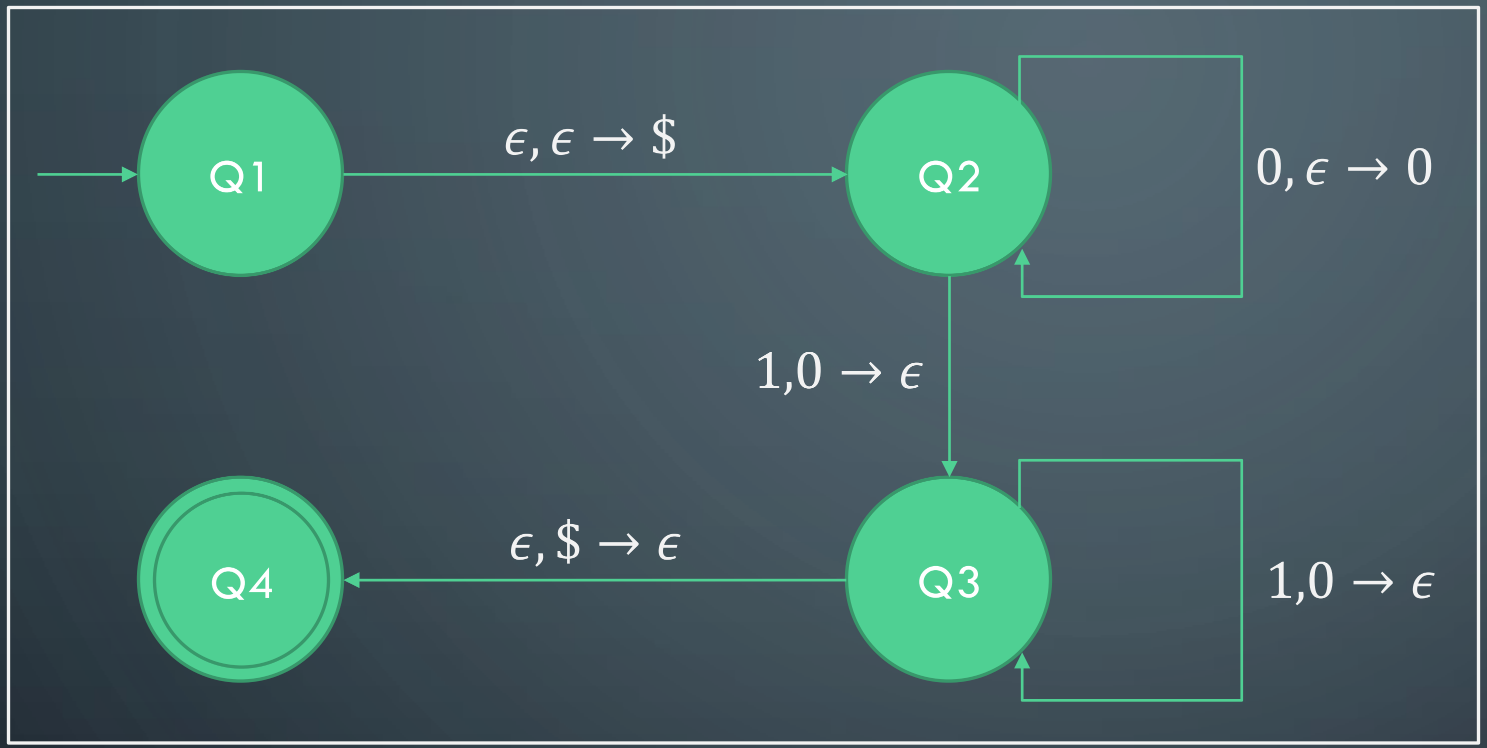


Stack is empty when we start and stop, good!

Only one start and accept state, good!

All transitions only push, pop but not both. Yep!

EXAMPLE: $0^n 1^n \mid n \geq 1$



WE DID IT!

<u>Theorem</u>: A language L is context-free if and only if some pushdown automata recognizes it

Great! So, pushdown automata and context-free grammars are equivalent in their descriptive power, and they are MORE powerful than regular languages / NFAs

PART 3: NON-CONTEXT-FREE LANGUAGES

PUMPING LEMMA FOR CFL

The pumping lemma for context-free languages:

If A is a context-free language, then there is a number p (the pumping length) where, if s is any string such that $s \in A$ and $|s| \geq p$, then s may be divided into five pieces $s = uvxyz$ satisfying the following:

1. for each $i \geq 0$, $uv^ixy^iz \in A$

2. $|vy| > 0$

3. $|vxy| \leq p$

PUMPING LEMMA FOR CFL

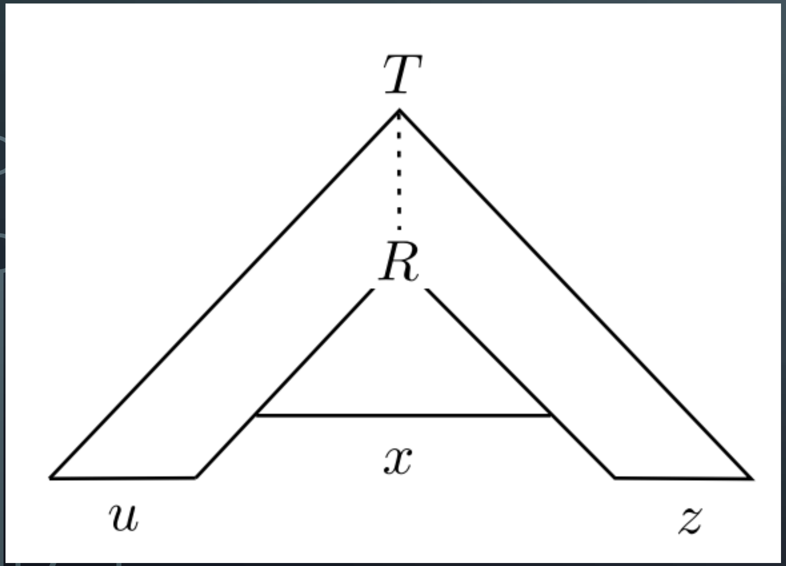
The pumping lemma for context-free languages:

If A is a context-free language, then there is a number p (the pumping length) where, if s is any string such that $s \in A$ and $|s| \geq p$, then s may be divided into five pieces $s = uvxyz$ satisfying the following:

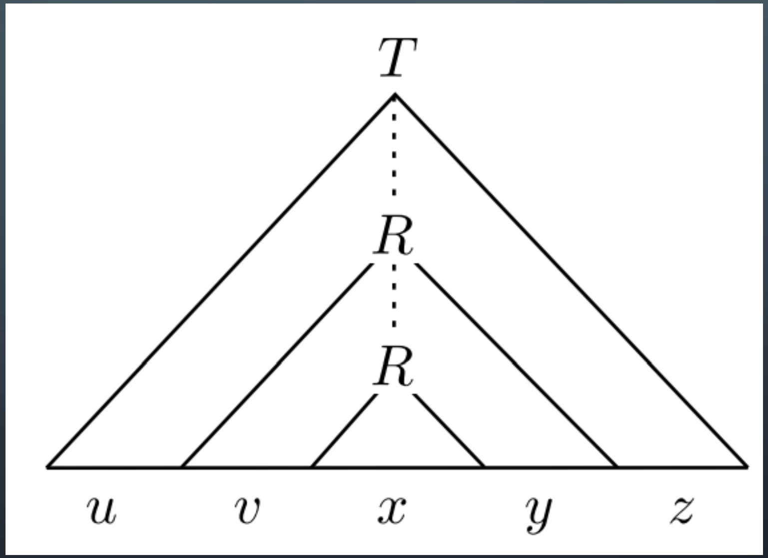
1. for each $i \geq 0$, $uv^ixy^iz \in A$

2. $|vy| > 0$

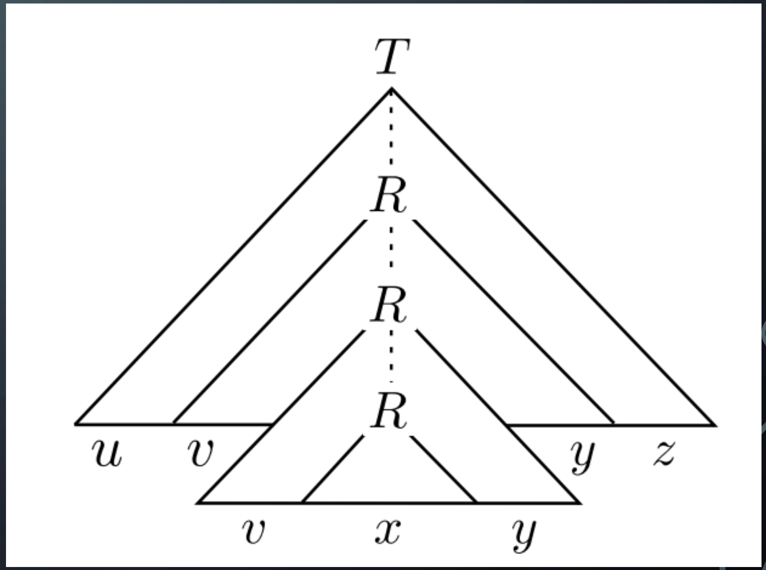
3. $|vxy| \leq p$



Pump down to remove v and y



Here, R is a variable that is "re-used"

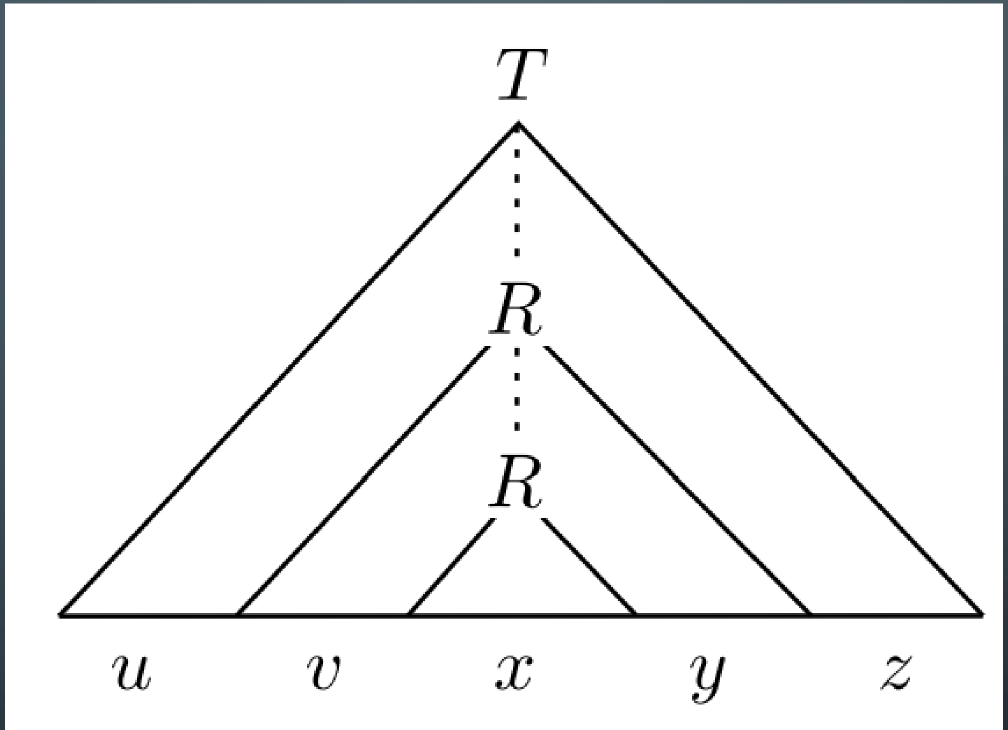


Pump up to add v and y

PUMPING LEMMA FOR CFL

Regarding the pumping length p

p needs to be set so that the length of uvxyz is long enough to guarantee that some variable R is "re-used".



How to choose p? Given the number of variables and the number of characters that can be substituted for each, we can calculate a tree height that guarantees some variable occurs twice (pigeonhole principle).

Then, find a length for uvxyz that guarantees that tree height. Your book shows the exact calculation if you are interested.

Condition 3 of the lemma says that the substring vxy is less than or equal to the pumping length p

EXAMPLE!

Proof: Use the pumping lemma to show that $B = \{a^n b^n c^n \mid n \geq 0\}$ is NOT context-free

EXAMPLE!

Proof: Use the pumping lemma to show that $B = \{a^n b^n c^n \mid n \geq 0\}$ is NOT context-free

Step 1: Pumping lemma states that there is some length string p, such that any string of that length can be pumped.

Step 2: Given p, we choose the string $a^p b^p c^p$, we then show that this string cannot be pumped.

EXAMPLE!

Proof: Use the pumping lemma to show that $B = \{a^n b^n c^n \mid n \geq 0\}$ is NOT context-free

Step 1: Pumping lemma states that there is some length string p, such that any string of that length can be pumped.

Step 2: Given p, we choose the string $a^p b^p c^p$, we then show that this string cannot be pumped.

Case 1: **divide string such that v and y both contain only one character each.**

Contradiction: **There are 3 different letters so when pumped, there won't be equal numbers of a, b, and c**

Case 2: **divide string such that at least one of v and y contain two characters.**

Contradiction: **When pumped, the letters will be out of order (e.g., aabb becomes aabbaabb)**

EXAMPLE!

Proof: Use the pumping lemma to show that D= $\{ww \mid w \in \{0,1\}^*\}$ is NOT context-free

EXAMPLE!

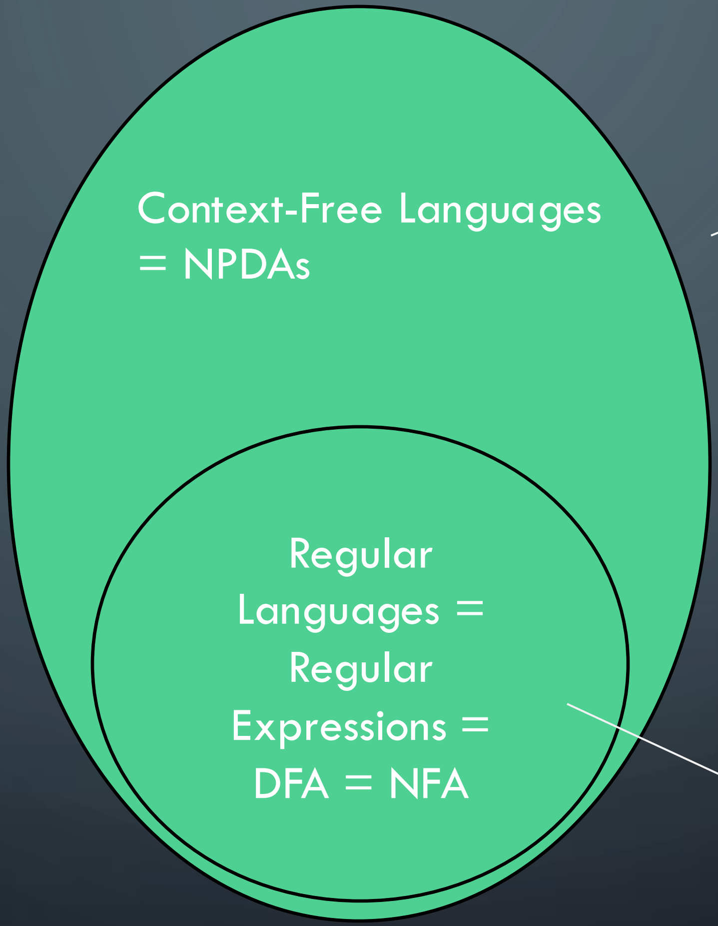
Proof: Use the pumping lemma to show that D= $\{ww \mid w \in \{0,1\}^*\}$ is NOT context-free

Step 1: Pumping lemma states that there is some length string p, such that any string of that length can be pumped.

Step 2: Given p, we choose the string $0^p 1^p 0^p 1^p$, we then show that this string cannot be pumped.

ABOUT DETERMINISM VERSUS NON-DETERMINISM WITH PUSHDOWN AUTOMATA

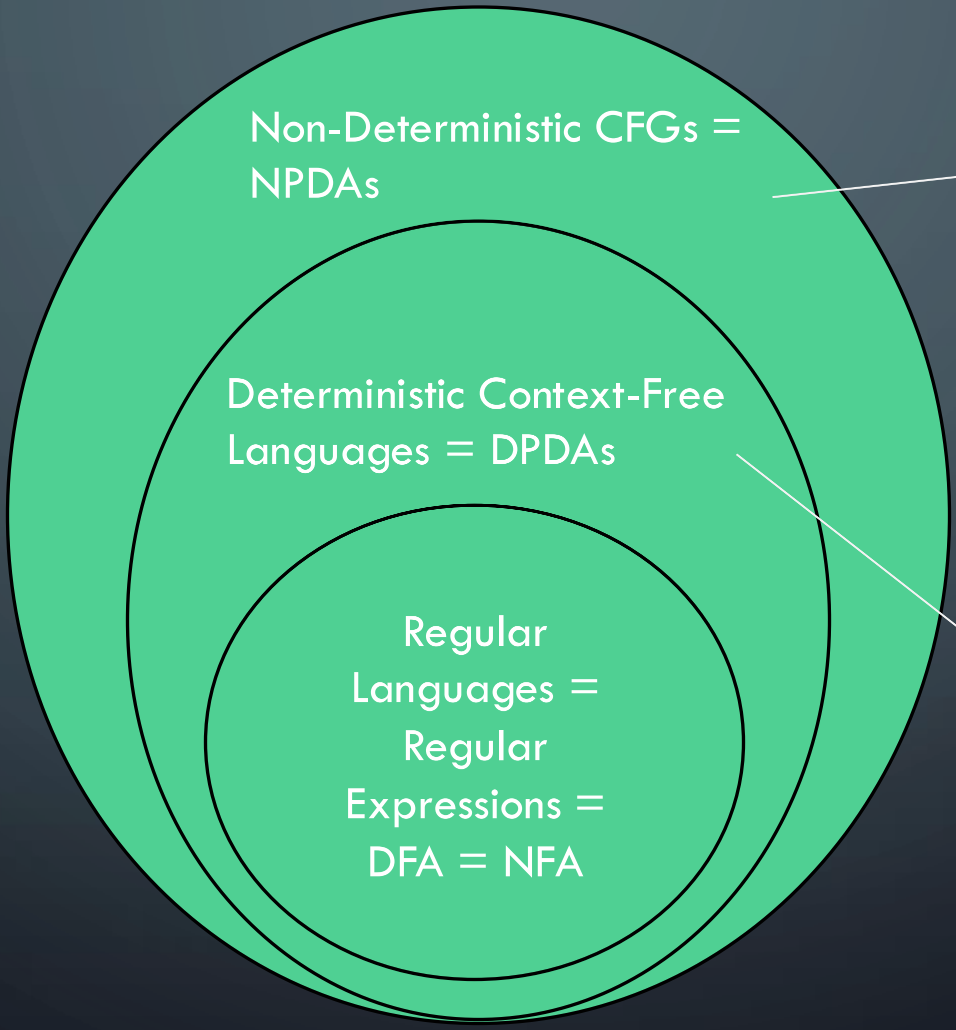
WHAT WE KNOW ABOUT COMPUTATION SO FAR



NPDA is non-deterministic pushdown automata. Remember that everything we've done so far is allowing non-determinism.

With regular languages, determinism and non-determinism models were equivalently descriptive.

WHAT WE KNOW ABOUT COMPUTATION SO FAR



Non-determinism is a MORE powerful descriptor within context-free languages. There are some languages that determinism cannot recognize (see book for details)

This means that programming language designers need to be careful because non-deterministic machines cannot currently be built. Need to stay in this middle section here

CONCLUSIONS

WHAT YOU LEARNED IN THIS DECK!

These are all equivalent in their expressive power.

Pushdown Automata	Context-Free Languages:	Context-Free Grammar:
NFA w/ a stack. Can recognize exactly the context-free languages	*A Class of languages that are more expressive than regular languages*	*A "string" description of a context-free language (by definition)*

Using another pumping lemma, we can find languages that are non-context free. Next we will see the most general computational model: The Turing Machine!