

Pre-lecture teaser

Given the language:

$$L = \{ww^R \mid w \in \{0,1\}^*\} \quad (1)$$

Prove that this language is non-regular

Languages

ECE-374-B: Lecture 6 - Context-Free ~~Grammars~~

Instructor: Nickvash Kani

September 16, 2025

University of Illinois Urbana-Champaign

Pre-lecture teaser

Given the language:

$$L = \{ww^R \mid w \in \{0,1\}^*\} \rightarrow \text{All even palindromes} \quad (2)$$

Reverse

Prove that this language is non-regular

Proof by contradiction:

- Assumption: L is regular

- L is representable by a DFA

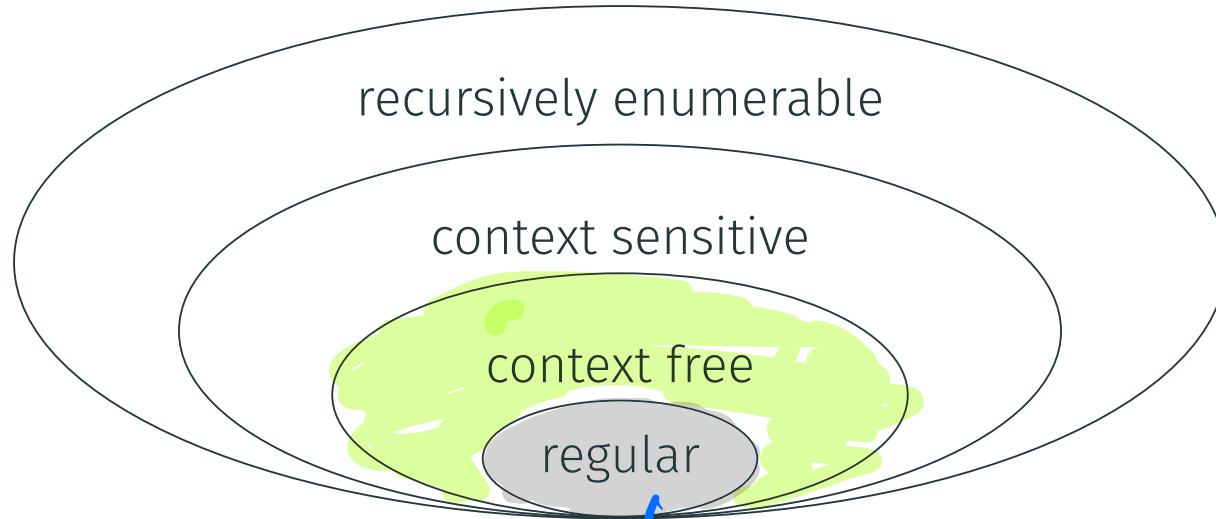
- $F = \{(10)^n \mid n \geq 0\}$ is a fooling set for L
 $x = (10)^i \quad y = (10)^j \quad i \neq j \quad z = (01)^i$
 $xz \in L \quad yz \notin L \quad \therefore F \text{ is a fooling set for } L$

- Every string in F has a distinguishable state

- But $|F| = \infty$

- Contradiction \rightarrow means L is not regular

Chomsky hierarchy revisited



→ languages are representable
by a DFA / NFA / RegEx

Example of Context-Free Languages

New addition to our toolbox

Regular languages could be constructed using a finite number of:

- Unions
- Concatenations
- Repetitions

With context-free languages we have a much more powerful tool:

Substitution (aka recursion)!

Example

- $V = \{S\}$
- $T = \{0, 1\}$
- $P = \{S \rightarrow \epsilon \mid 0S0 \mid 1S1\}$
(abbrev. for $S \rightarrow \epsilon, S \rightarrow 0S0, S \rightarrow 1S1$)

→ variables

*Terminals
Alphabet*

Grammar

Example

- $V = \{S\}$
- $T = \{0, 1\}$
- $P = \{S \rightarrow \epsilon \mid 0S0 \mid 1S1\}$
(abbrev. for $S \rightarrow \epsilon, S \rightarrow 0S0, S \rightarrow 1S1$)

$L = \left\{ \begin{array}{l} 011110 \\ \epsilon \\ 1001 \\ 1110110111 \\ \vdots \end{array} \right\}$

$S \rightsquigarrow 0S0 \rightsquigarrow 01S10 \rightsquigarrow 011S110 \rightsquigarrow 011\epsilon 110 \rightsquigarrow 011110$

$S \rightarrow \epsilon$

$S \rightarrow 1S1 \rightarrow 10S01 \rightarrow 10\epsilon 01$

$S \rightarrow 1S1 \rightarrow 11S11 \rightarrow 111S111 \rightarrow 1110S0111$
 $\rightarrow 11101S10111 \rightarrow 11101\epsilon 10111$

Example

- $V = \{S\}$
- $T = \{0, 1\}$
- $P = \{S \rightarrow \epsilon \mid 0S0 \mid 1S1\}$
(abbrev. for $S \rightarrow \epsilon, S \rightarrow 0S0, S \rightarrow 1S1$)

$$S \rightsquigarrow 0S0 \rightsquigarrow 01S10 \rightsquigarrow 011S110 \rightsquigarrow 011\epsilon 110 \rightsquigarrow 011110$$

What strings can S generate like this?

w^R

Formal definition of context-free languages (CFGs)

Context Free Grammar (CFG) Definition

Definition

A CFG is a quadruple $G = (V, T, P, S)$

- V is a finite set of non-terminal (variable) symbols

$$G = \left(\text{Variables, Terminals, Productions, Start var} \right)$$

Context Free Grammar (CFG) Definition

Definition

A CFG is a quadruple $G = (V, T, P, S)$

- V is a finite set of non-terminal (variable) symbols
- T is a finite set of terminal symbols (alphabet)

$$G = \left(\text{Variables, Terminals, Productions, Start var} \right)$$

Context Free Grammar (CFG) Definition

Definition

A CFG is a quadruple $G = (V, T, P, S)$

- V is a finite set of non-terminal (variable) symbols
- T is a finite set of terminal symbols (alphabet)
- P is a finite set of productions, each of the form

$A \rightarrow \alpha$

→ this is for CF specifically

where $A \in V$ and α is a string in $(V \cup T)^*$.

Formally, $P \subset V \times (V \cup T)^*$.

$$G = \left(\text{Variables, Terminals, Productions, Start var} \right)$$

Context Free Grammar (CFG) Definition

Definition

A CFG is a quadruple $G = (V, T, P, S)$

- V is a finite set of non-terminal (variable) symbols
- T is a finite set of terminal symbols (alphabet)
- P is a finite set of productions, each of the form

$$A \rightarrow \alpha$$

where $A \in V$ and α is a string in $(V \cup T)^*$.

Formally, $P \subset V \times (V \cup T)^*$.

- $S \in V$ is a start symbol
(variable)

$$G = \left(\text{Variables, Terminals, Productions, Start var} \right)$$

Example formally...

- $V = \{S\}$
- $T = \{0, 1\}$
- $P = \{S \rightarrow \epsilon \mid 0S0 \mid 1S1\}$
(abbrev. for $S \rightarrow \epsilon, S \rightarrow 0S0, S \rightarrow 1S1$)

$$G = \left(\{S\}, \{0, 1\}, \left\{ \begin{array}{l} S \rightarrow \epsilon, \\ S \rightarrow 0S0 \\ S \rightarrow 1S1 \end{array} \right\}, S \right)$$

$$L(G) = \{ ww^R \mid w \in \Sigma^* \}$$

Notation and Convention

Let $G = (V, T, P, S)$ then

- a, b, c, d, \dots , in T (terminals)
- A, B, C, D, \dots , in V (non-terminals)
- u, v, w, x, y, \dots in T^* for strings of terminals
- $\alpha, \beta, \gamma, \dots$ in $(V \cup T)^*$
- X, Y, X in $V \cup T$

“Derives” relation

Formalism for how strings are derived/generated

Definition

Let $G = (V, T, P, S)$ be a CFG. For strings $\alpha_1, \alpha_2 \in (V \cup T)^*$ we say α_1 derives α_2 denoted by $\alpha_1 \rightsquigarrow_G \alpha_2$ if there exist strings β, γ, δ in $(V \cup T)^*$ such that

- $\alpha_1 = \beta A \delta$
- $\alpha_2 = \beta \gamma \delta$
- $A \rightarrow \gamma$ is in P .

δ δ^*

produces → *derives*

Examples: $S \rightsquigarrow \epsilon$, $S \rightsquigarrow 0S1$, $0S1 \rightsquigarrow 00S11$, $0S1 \rightsquigarrow 01$.

$0S1 \rightarrow \underline{0\epsilon 1} \rightarrow 01$

“Derives” relation continued

Definition

For integer $k \geq 0$, $\alpha_1 \rightsquigarrow^k \alpha_2$ inductive defined:

- $\alpha_1 \rightsquigarrow^0 \alpha_2$ if $\alpha_1 = \alpha_2$
- $\alpha_1 \rightsquigarrow^k \alpha_2$ if $\alpha_1 \rightsquigarrow \beta_1$ and $\beta_1 \rightsquigarrow^{k-1} \alpha_2$.

$$\begin{aligned} \delta^*(q, w) &= q \quad w = \epsilon \\ &= \delta^*(\delta(q, a), x) \\ &\quad | \quad w = ax \end{aligned}$$

“Derives” relation continued

Definition

For integer $k \geq 0$, $\alpha_1 \rightsquigarrow^k \alpha_2$ inductive defined:

- $\alpha_1 \rightsquigarrow^0 \alpha_2$ if $\alpha_1 = \alpha_2$
- $\alpha_1 \rightsquigarrow^k \alpha_2$ if $\alpha_1 \rightsquigarrow \beta_1$ and $\beta_1 \rightsquigarrow^{k-1} \alpha_2$.
- **Alternative definition:** $\alpha_1 \rightsquigarrow^k \alpha_2$ if $\alpha_1 \rightsquigarrow^{k-1} \beta_1$ and $\beta_1 \rightsquigarrow \alpha_2$

“Derives” relation continued

Definition

For integer $k \geq 0$, $\alpha_1 \rightsquigarrow^k \alpha_2$ inductive defined:

- $\alpha_1 \rightsquigarrow^0 \alpha_2$ if $\alpha_1 = \alpha_2$
- $\alpha_1 \rightsquigarrow^k \alpha_2$ if $\alpha_1 \rightsquigarrow \beta_1$ and $\beta_1 \rightsquigarrow^{k-1} \alpha_2$.
- **Alternative definition:** $\alpha_1 \rightsquigarrow^k \alpha_2$ if $\alpha_1 \rightsquigarrow^{k-1} \beta_1$ and $\beta_1 \rightsquigarrow \alpha_2$

\rightsquigarrow^* is the reflexive and transitive closure of \rightsquigarrow .

$\alpha_1 \rightsquigarrow^* \alpha_2$ if $\alpha_1 \rightsquigarrow^k \alpha_2$ for some k .

Examples: $S \rightsquigarrow^* \epsilon$, $0S1 \rightsquigarrow^* 0000011111$.

Context Free Languages

Definition

The language generated by CFG $G = (V, T, P, S)$ is denoted by $L(G)$ where

$$L(G) = \{w \in T^* \mid S \xrightarrow{*} w\}.$$

w must have only terminal symbols
at the end

Context Free Languages

Definition

The language generated by CFG $G = (V, T, P, S)$ is denoted by $L(G)$ where $L(G) = \{w \in T^* \mid S \xrightarrow{*} w\}$.

Definition

A language L is context free (CFL) if it is generated by a context free grammar. That is, there is a CFG G such that $L = L(G)$.

Grammar where every production is of the form $V \rightarrow \{TVUV\}^*$

Example

$L = \{0^n 1^n \mid n \geq 0\}$ = non-regular
content-free?

$V = \Sigma$

$T = \{0, 1\}$

$P = \Sigma \rightarrow \epsilon \mid 0 \mid 1$

$S = \Sigma$

$\left\{ \begin{array}{l} \epsilon \\ 01 \\ 0011 \\ 000111 \\ \vdots \end{array} \right\}$

Example

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

$$V = S, A$$

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

$$L = \{0^n 1^m \mid m > n\}$$

$$P = S \rightarrow A1$$

$$A \rightarrow 0A1 \mid A1 \mid \epsilon$$

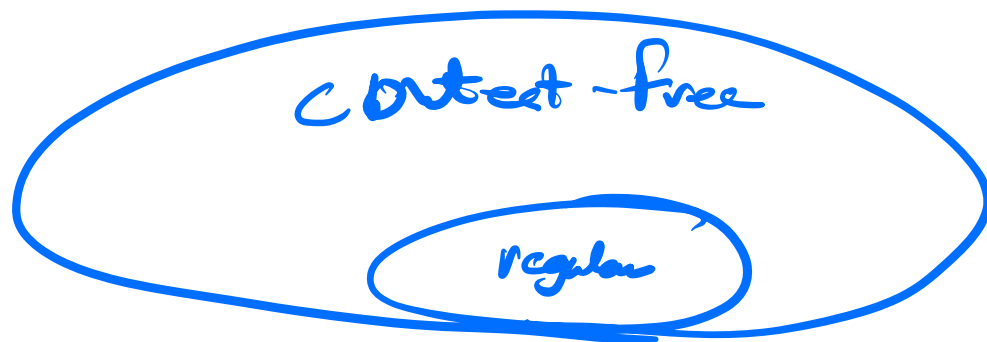
$$S = S$$

Converting regular languages into CFL

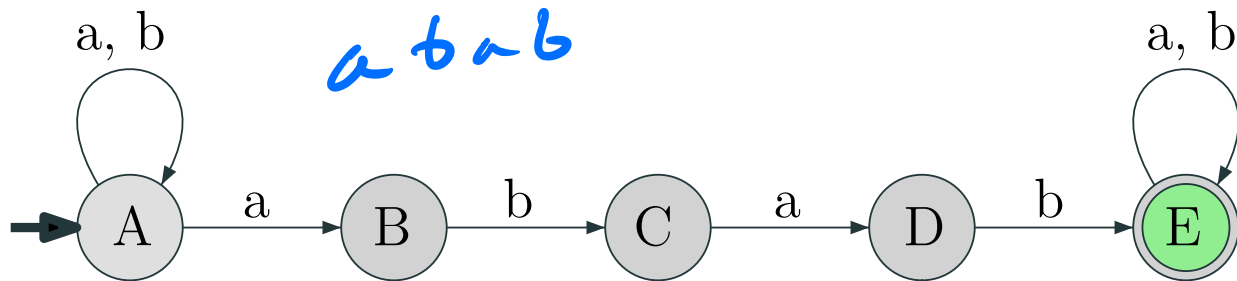
Regular Grammar

What was the grammar for a regular language?

Let's figure it out visually!



Converting regular languages into CFL I



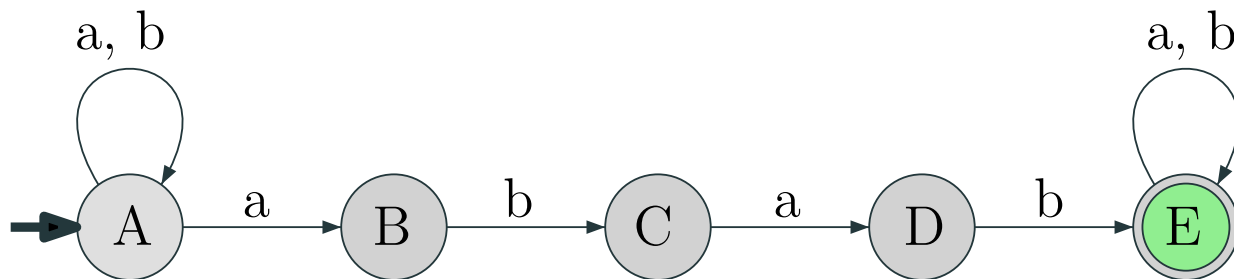
$$G = \left(\{A, B, C, D, E\}, \{a, b\}, \left\{ \begin{array}{l} A \rightarrow aA, A \rightarrow bA, A \rightarrow aB, \\ B \rightarrow bC, \\ C \rightarrow aD, \\ D \rightarrow bE, \\ E \rightarrow aE, E \rightarrow bE, E \rightarrow \epsilon \end{array} \right\}, A \right)$$

$A \rightarrow aB \rightarrow abC \rightarrow abaD \rightarrow ababE$
 $\rightarrow abab\epsilon \rightarrow abab$

Converting regular languages into CFL II

$M = (Q, \Sigma, \delta, s, A)$: ^(NFA) DFA for regular language L .

$$G = \left(\underbrace{Q}_{\text{Variables}}, \underbrace{\Sigma}_{\text{Terminals}}, \underbrace{\left\{ q \rightarrow a\delta(q, a) \mid q \in Q, a \in \Sigma \right\} \cup \left\{ q \rightarrow \varepsilon \mid q \in A \right\}}_{\text{Productions}}, \underbrace{s}_{\text{Start var}} \right)$$



Converting regular languages into CFL I

$$G = \left(\{A, B, C, D, E\}, \{a, b\}, \left\{ \begin{array}{l} A \rightarrow aA, A \rightarrow bA, A \rightarrow aB, \\ B \rightarrow bC, \\ C \rightarrow aD, \\ D \rightarrow bE, \\ E \rightarrow aE, E \rightarrow bE, E \rightarrow \varepsilon \end{array} \right\}, A \right)$$

Handwritten blue annotations:
A blue circle highlights the production rules for E : $E \rightarrow aE, E \rightarrow bE, E \rightarrow \varepsilon$.
Below the circle, the handwritten text $V \rightarrow TV$ is written.
To the right of the circle, the handwritten text "regular grammar" is written.

In regular languages:

- Terminals can only appear on one side of the production string
- Only one variable allowed in production result

The result...

Lemma

For an regular language L , there is a context-free grammar (CFG) that generates it.

Push-down automata

The machine that generates CFGs

$\{0^n 1^n \mid n \geq 0\}$ is a CFL.

We have NFAs from regular languages. What can we add to enable them to recognize CFLs?

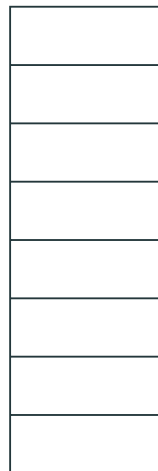
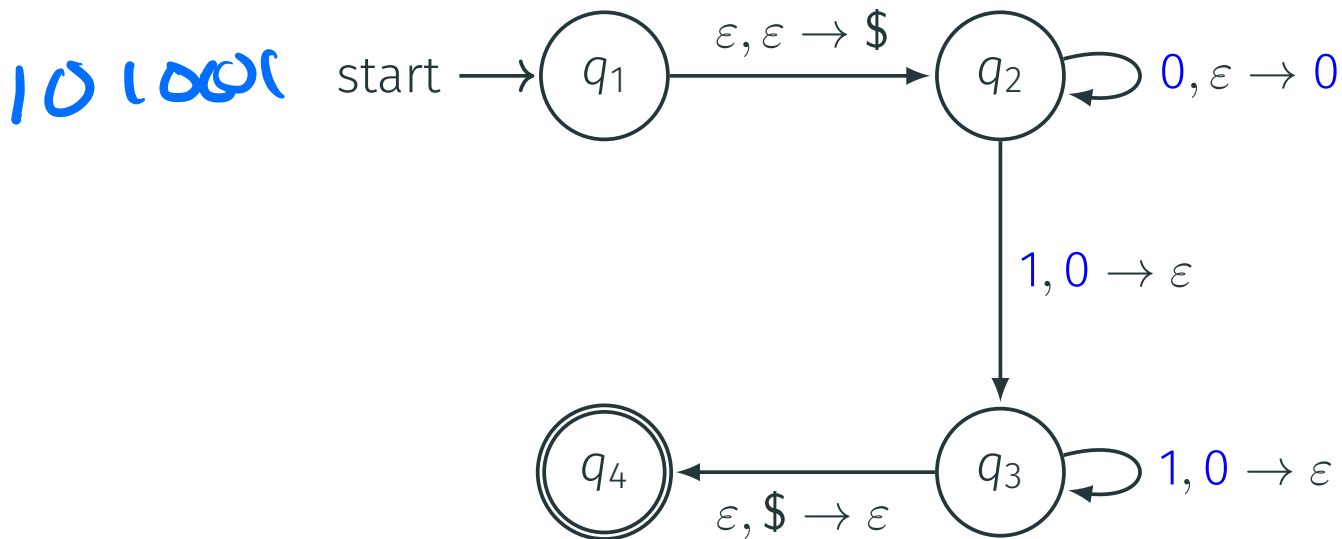
The machine that generates CFGs

$\{0^n 1^n | n \geq 0\}$ is a CFL.

We have NFAs from regular languages. What can we add to enable them to recognize CFLs?

We need a stack!

Push-down automata example

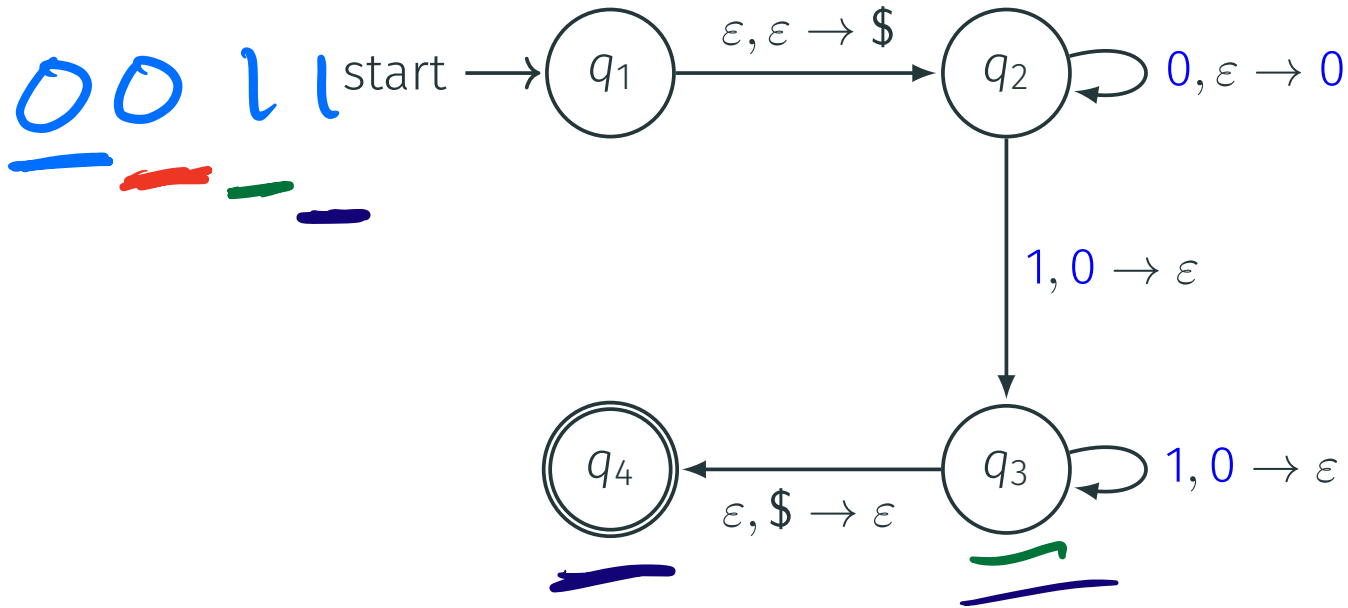


Each transition is formatted as:

$\langle \text{input read} \rangle, \langle \text{stack pop} \rangle \rightarrow \langle \text{stack push} \rangle$

(3)

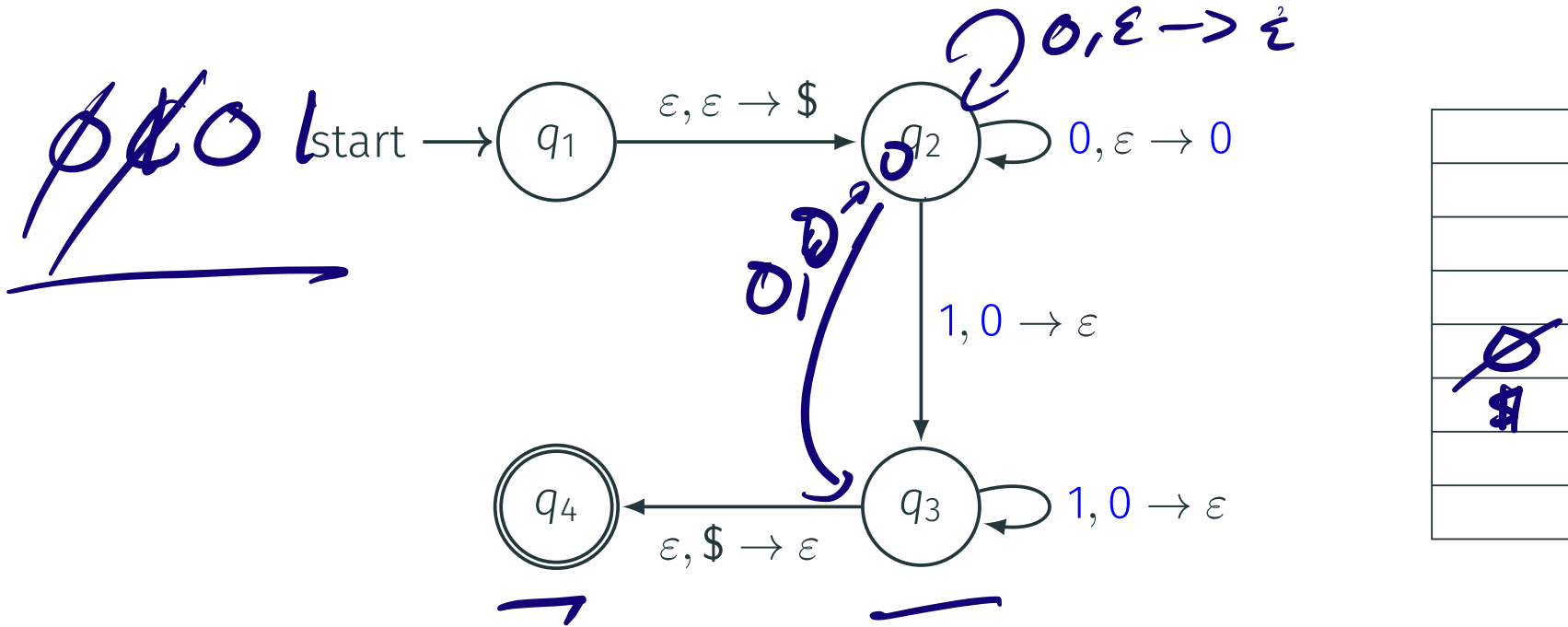
Push-down automata example



Does this machine recognize 0011?

Yes

Push-down automata example



Does this machine recognize 0101?

No

Formal Tuple Notation

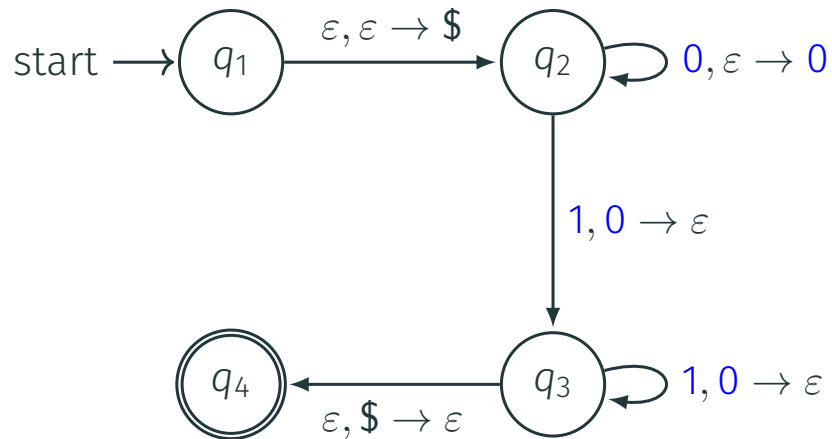
Definition

A non-deterministic push-down automata $P = (Q, \Sigma, \Gamma, \delta, s, A)$ is a **six** tuple where

- Q is a finite set whose elements are called **states**,
- Σ is a finite set called the **input alphabet**,
- Γ is a finite set called the **stack alphabet**,
- $\delta : Q \times \Sigma \cup \{\varepsilon\} \times \Gamma \cup \{\varepsilon\} \rightarrow \mathcal{P}(Q \times (\Gamma \cup \{\varepsilon\}))$ is the **transition function**
- s is the start state
- A is the set of accepting states

Non-deterministic PDAs are more powerful than deterministic PDAs. Hence we'll only be talking about non-deterministic PDAs.

Formal Tuple Notation of $0^n 1^n$



- $Q =$
- $\Sigma =$
- $\Gamma =$
- $S =$
- $A =$

	Input Stack	0			1			ϵ		
		0	\$	ϵ	0	\$	ϵ	0	\$	ϵ
$\delta =$	q_1									$\{(q_2, \$)\}$
	q_2			$\{(q_2, 0)\}$	$\{(q_3, \epsilon)\}$					
	q_3				$\{(q_3, \epsilon)\}$					
	q_4									$\{(q_4, \epsilon)\}$

CFGs and PDAs

Convert a CFG to a PDA I

Converting a CFG to a PDA is simple (but a little tedious). Let's demonstrate via simple example:

$$S \rightarrow 0S|1$$

Convert a CFG to a PDA I

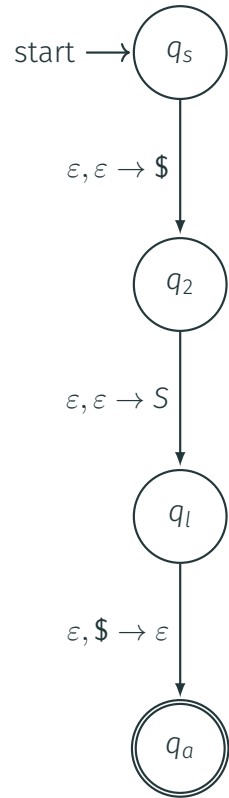
Converting a CFG to a PDA is simple (but a little tedious). Let's demonstrate via simple example:

$$S \rightarrow 0S|1$$

Idea:

- We try to recreate the string on the stack:
 - Everytime we see a non-terminal, we replace it by one of the replacement rules.
 - Everytime we see a terminal symbol, we take that symbol from the input.
- if we reach a point where there stack is empty and the input is empty, then we accept the string.

Convert a CFG to a PDA I

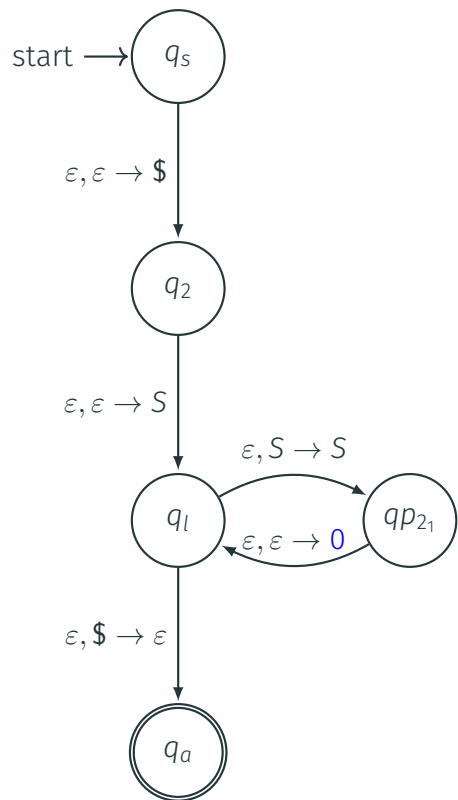


$$S \rightarrow 0S|1|\epsilon$$

- First let's put in a $\$$ to mark the end of the string
- Also let's put in the start symbol on the stack.



Convert a CFG to a PDA I



$$S \rightarrow 0S|1|\epsilon$$

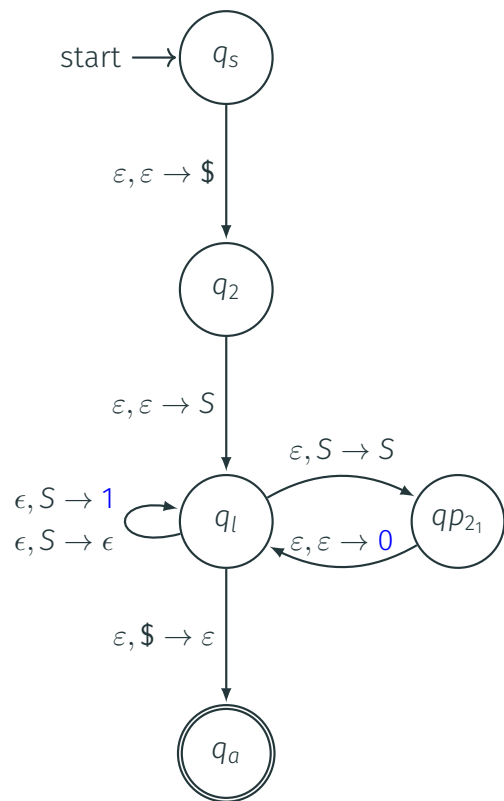
Next we want to add a loop for every non-terminal symbol that replaces that non-terminal with the result.

Consider the rule: $S \rightarrow 0S$

- So we got to pop the S non-terminal,
- Add a S non-terminal to the stack.
- And add a 0 terminal to the stack.



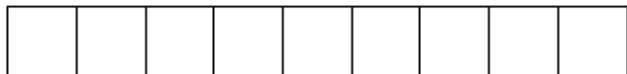
Convert a CFG to a PDA I



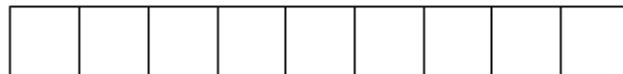
$$S \rightarrow 0S|1|\epsilon$$

Do the same thing for $S \rightarrow 1$ and $S \rightarrow \epsilon$

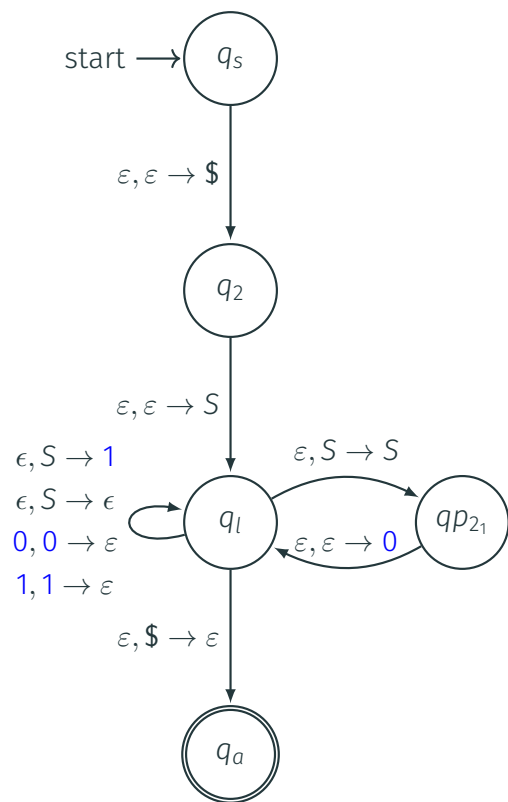
Input \longrightarrow



Stack \longrightarrow

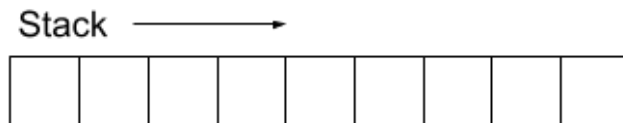


Convert a CFG to a PDA I

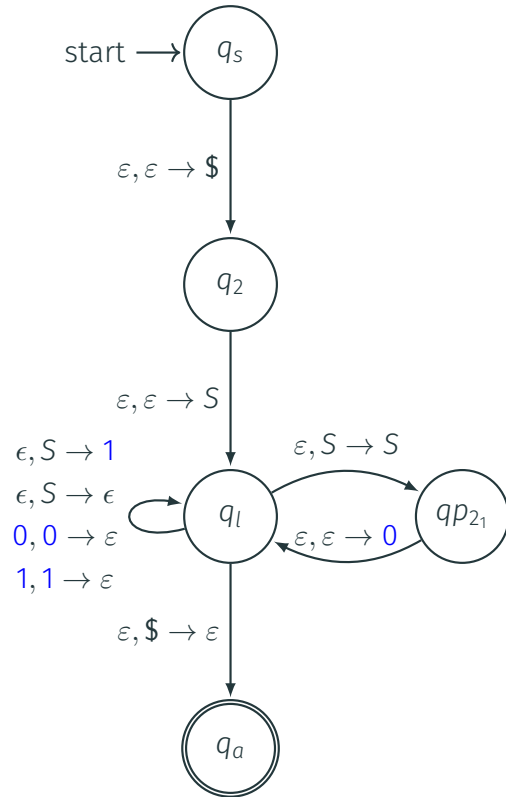


$$S \rightarrow 0S|1|\epsilon$$

If we see a non-terminal symbol on the stack, then we can cross that symbol from the input. Got to add transitions to do that.



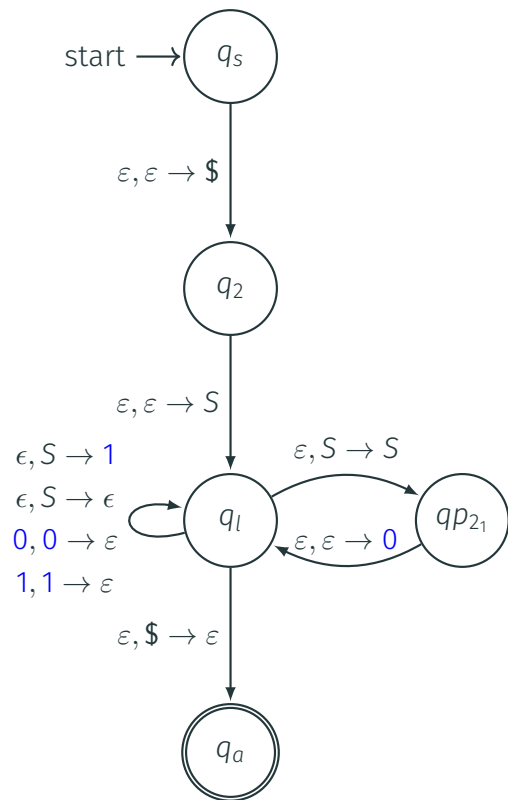
Convert a CFG to a PDA I



$$S \rightarrow 0S|1|\epsilon$$

Let's go over the operation again:

Convert a CFG to a PDA I

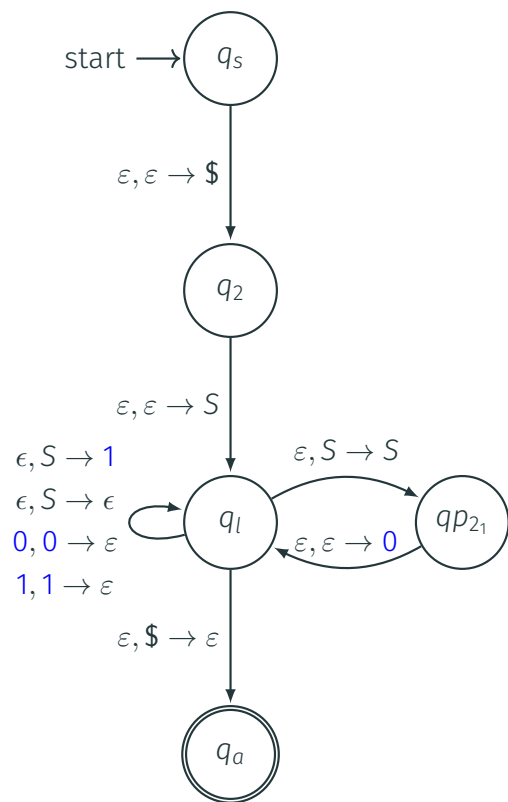


$$S \rightarrow 0S|1|\epsilon$$

Let's go over the operation again:

- Does this automata accept 001?

Convert a CFG to a PDA I



$$S \rightarrow 0S|1|\epsilon$$

Let's go over the operation again:

- Does this automata accept 001?
- Does this automata accept 010?



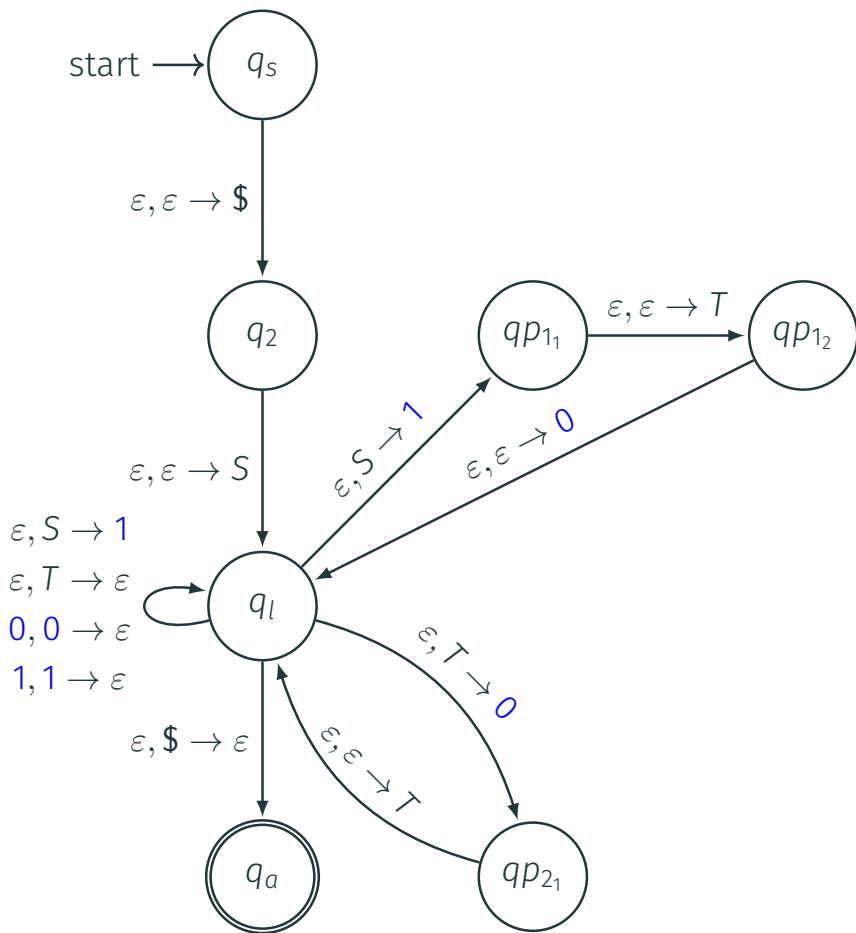
Convert a CFG to a PDA II

Let's do a harder example:

$$S \rightarrow 0T1|1$$

$$T \rightarrow T0|\varepsilon$$

Convert a CFG to a PDA II

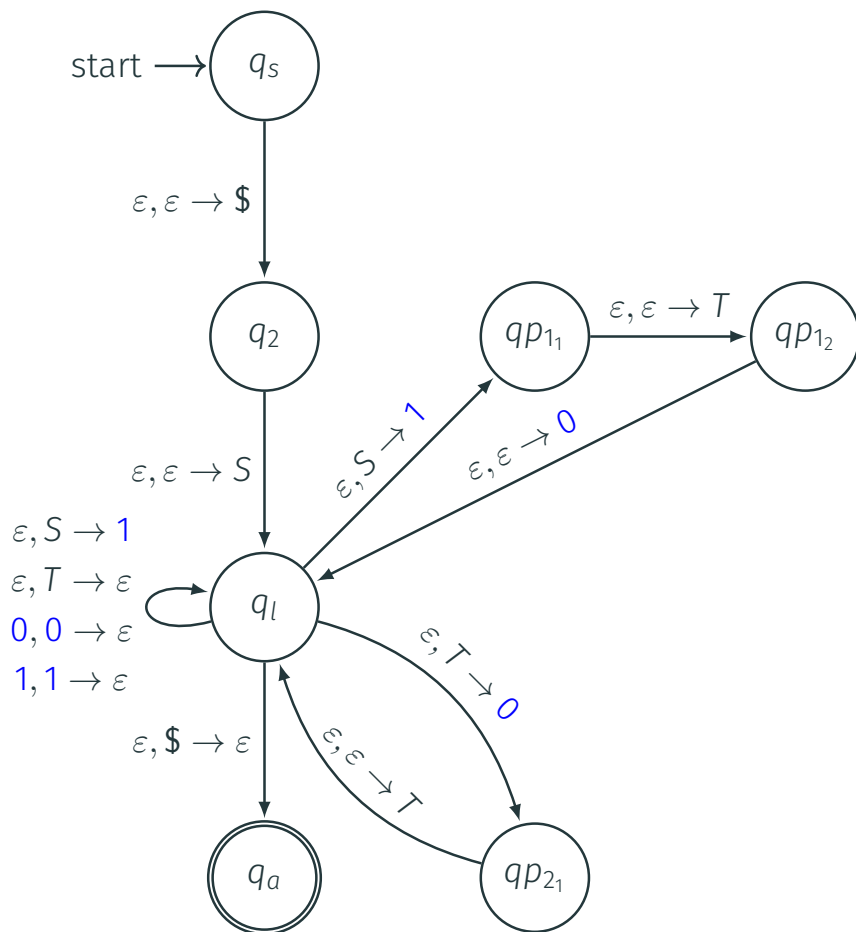


$$S \rightarrow 0T1|1$$

$$T \rightarrow T0|\epsilon$$

The goal of our PDA is to construct the string within the stack and pop off the leftmost terminals when we read those terminals on the input string.

Convert a CFG to a PDA II

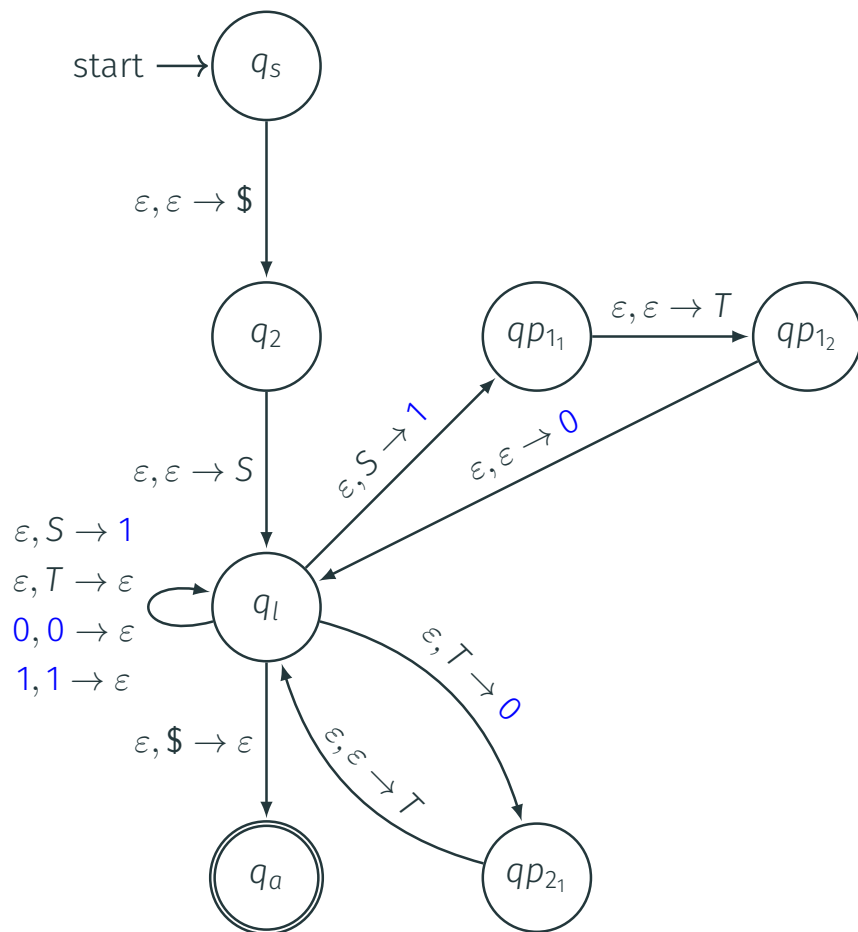


$$S \rightarrow 0T1|1$$

$$T \rightarrow T0|\epsilon$$

- First we need to mark the start of the stack.
- Then we put the start variable on the stack.

Convert a CFG to a PDA II

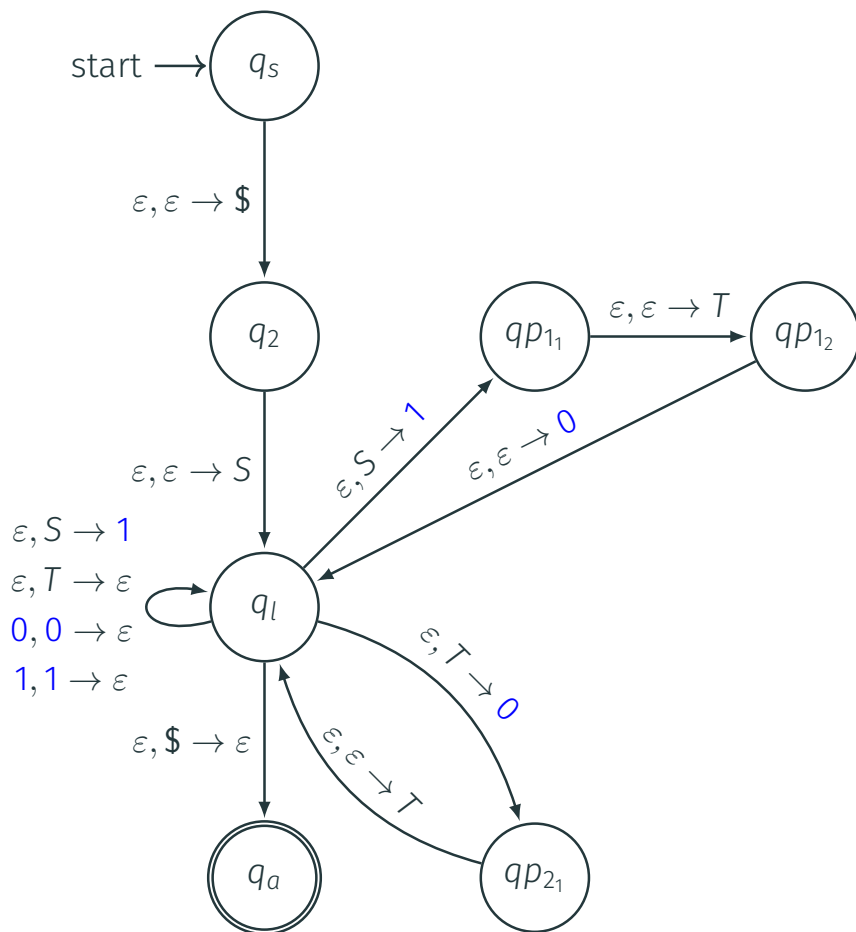


$$S \rightarrow 0T1|1$$

$$T \rightarrow T0|\epsilon$$

- We create a loop for each production rule.
- If we read a terminal that matches the input we pop it.

Convert a CFG to a PDA II



$$S \rightarrow 0T1|1$$

$$T \rightarrow T0|\epsilon$$

Computation ends when all the variables/terminals have been popped off the stack and the input is empty.

Determinism in Context-Free Languages

As you remember, deterministic finite automata (**DFAs**) and nondeterministic finite automata (**NFAs**) are equivalent in language recognition power.

Not so for **PDA**s. The previous PDA could not be completed using a deterministic **PDA** because we need to know where the middle of the input string is for determinism!

$L = \{0^n 1^n \mid n \geq 0\}$ can be modeled with a deterministic-**PDA**.

Learn more in CS 475 (**Beyond the scope of this class.**)

Closure properties of CFLs

Closure Properties of CFLs

$G_1 = (V_1, T, P_1, S_1)$ and $G_2 = (V_2, T, P_2, S_2)$

Assumption: $V_1 \cap V_2 = \emptyset$, that is, non-terminals are not shared

Closure Properties of CFLs

$G_1 = (V_1, T, P_1, S_1)$ and $G_2 = (V_2, T, P_2, S_2)$

Assumption: $V_1 \cap V_2 = \emptyset$, that is, non-terminals are not shared

Closure Properties of CFLs- Union

$G_1 = (V_1, T, P_1, S_1)$ and $G_2 = (V_2, T, P_2, S_2)$

Assumption: $V_1 \cap V_2 = \emptyset$, that is, non-terminals are not shared.

Theorem

CFLs are closed under union. L_1, L_2 CFLs implies $L_1 \cup L_2$ is a CFL.

$L_1 \cup L_2 = L_3$ is CF

L_1 representable by a CFG $\{V_1, T, P_1, S_1\}$

L_2 $\{V_2, T, P_2, S_2\}$

$L_3 = \{V_1 \cup V_2, T, P_1 \cup P_2, S_1 \mid S_2, S_1\}$

Closure Properties of CFLs- Concatenation

Theorem

CFLs are closed under concatenation. L_1, L_2 CFLs implies $L_1 \cdot L_2$ is a CFL.

$$s \rightarrow \begin{matrix} s_1 \cdot s_2 \\ (s_1 s_2) \end{matrix}$$

Closure Properties of CFLs- Kleene star

Theorem

CFLs are closed under Kleene star.

If L is a CFL $\implies L^$ is a CFL.*

$S \rightarrow \epsilon \mid \underline{SS} \mid \dots$

$S \rightarrow SS \rightarrow SSS \rightarrow SSSS \rightarrow \dots$

Bad news: Canonical non-CFL

Theorem

$L = \{a^n b^n c^n \mid n \geq 0\}$ is not context-free.

Proof based on **pumping lemma** for **CFLs**. See supplemental for the proof.

More bad news: CFL not closed under intersection

Theorem

CFLs are *not* closed under intersection.

$$L_1 = \{a^n b^n c^m \mid n, m \geq 0\}$$

$$L_2 = \{a^m b^n c^n \mid n, m \geq 0\}$$

$$L_1 \cap L_2 = \{a^n b^n c^n \mid n \geq 0\}$$

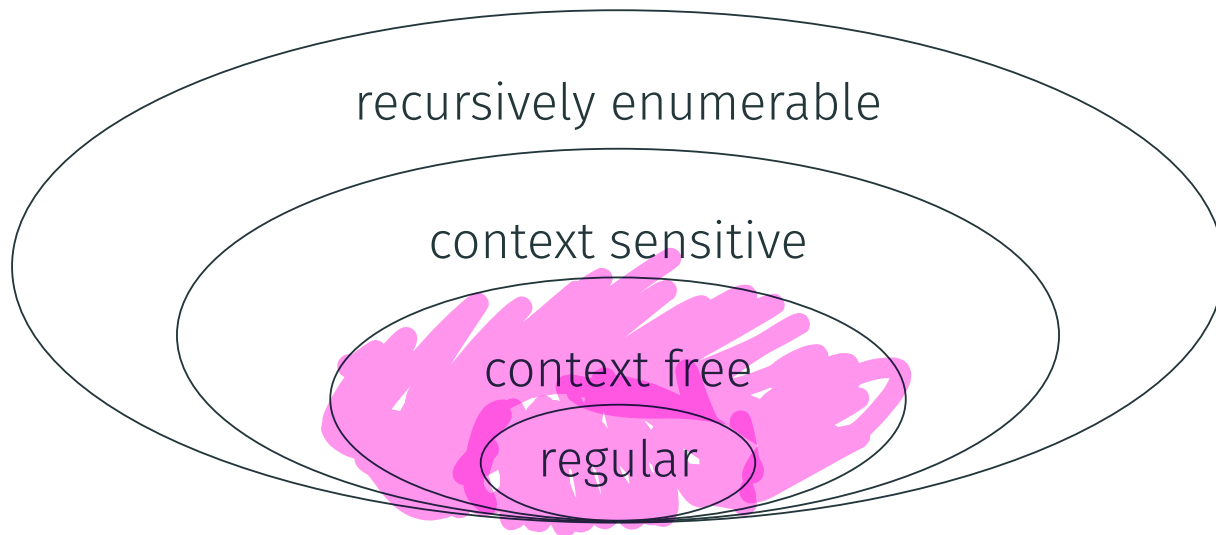
Even more bad news: CFL not closed under complement

Theorem

CFLs are *not* closed under complement.

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$$

The more you know!



We're making our way up the Chomsky hierarchy!

Next stop: context-sensitive, and decidable languages.

Parse trees and ambiguity

Parse Trees or Derivation Trees

A tree to represent the derivation $S \rightsquigarrow^* w$.

- Rooted tree with root labeled S
- Non-terminals at each internal node of tree
- Terminals at leaves
- Children of internal node indicate how non-terminal was expanded using a production rule

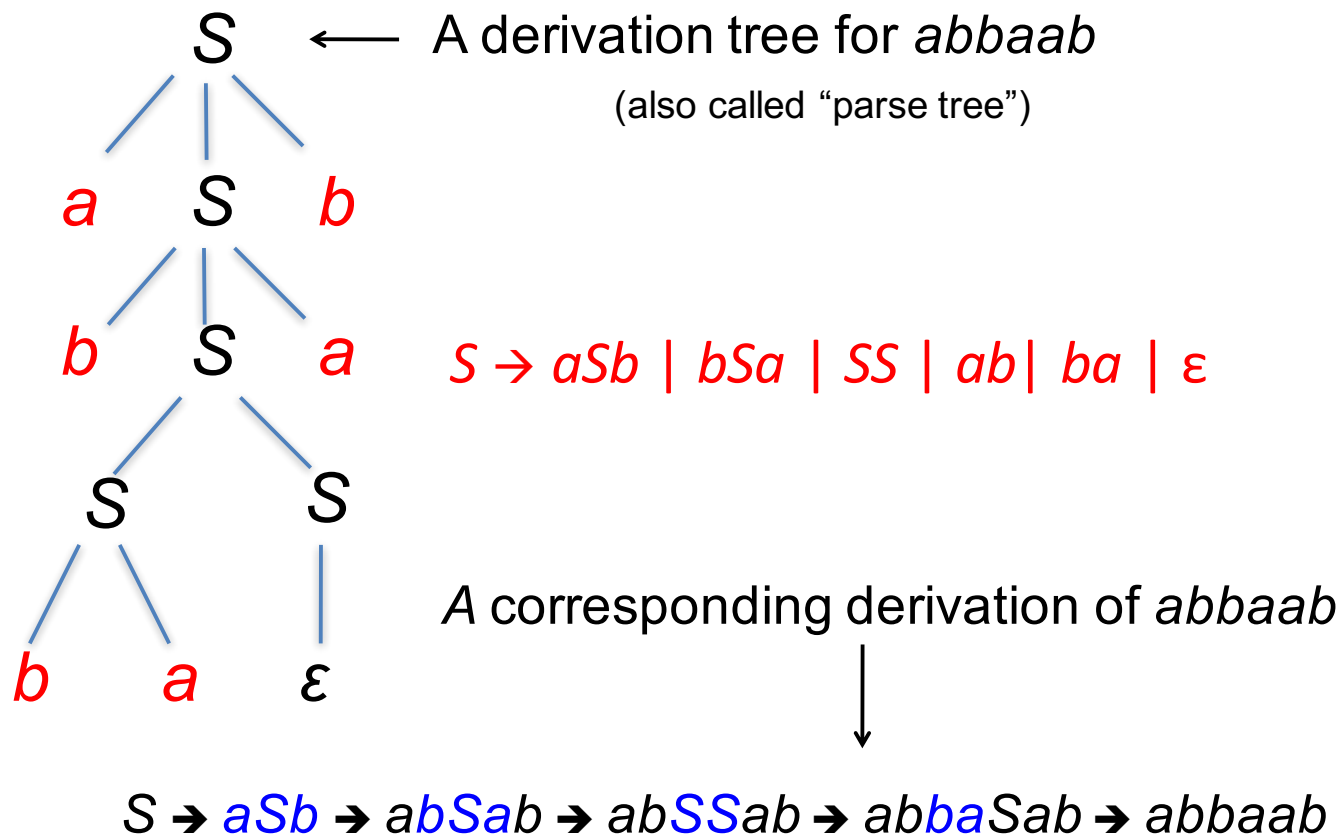
Parse Trees or Derivation Trees

A tree to represent the derivation $S \rightsquigarrow^* w$.

- Rooted tree with root labeled S
- Non-terminals at each internal node of tree
- Terminals at leaves
- Children of internal node indicate how non-terminal was expanded using a production rule

A picture is worth a thousand words

Example

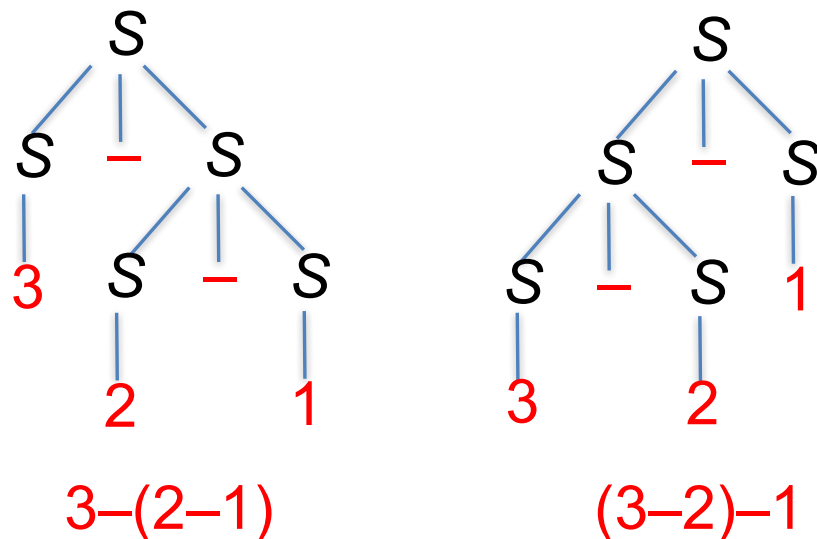


Ambiguity in CFLs

Definition

A CFG G is **ambiguous** if there is a string $w \in L(G)$ with two different parse trees. If there is no such string then G is **unambiguous**.

Example: $S \rightarrow S - S \mid 1 \mid 2 \mid 3$



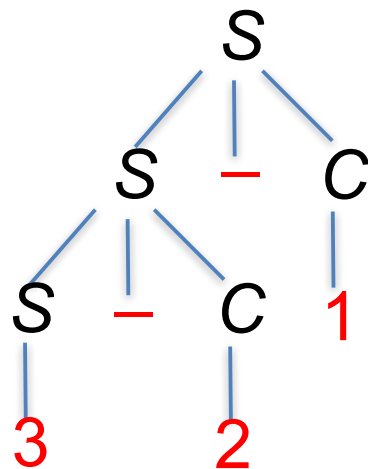
Ambiguity in CFLs

- Original grammar: $S \rightarrow S - S \mid 1 \mid 2 \mid 3$

- Unambiguous grammar:

$S \rightarrow S - C \mid 1 \mid 2 \mid 3$

$C \rightarrow 1 \mid 2 \mid 3$



$(3-2)-1$

The grammar forces a parse corresponding to left-to-right evaluation.

Inherently ambiguous languages

Definition

A CFL L is **inherently ambiguous** if there is no unambiguous CFG G such that $L = L(G)$.

Inherently ambiguous languages

Definition

A CFL L is **inherently ambiguous** if there is no unambiguous CFG G such that $L = L(G)$.

- There exist inherently ambiguous CFLs.

Example: $L = \{a^n b^m c^k \mid n = m \text{ or } m = k\}$

Inherently ambiguous languages

Definition

A CFL L is **inherently ambiguous** if there is no unambiguous CFG G such that $L = L(G)$.

- There exist inherently ambiguous CFLs.

Example: $L = \{a^n b^m c^k \mid n = m \text{ or } m = k\}$

- Given a grammar G it is **undecidable** to check whether $L(G)$ is inherently ambiguous. No algorithm!

Supplemental: Why $a^n b^n c^n$ is not CFL

You are bound to repeat yourself...

$$L = \{a^n b^n c^n \mid n \geq 0\}.$$

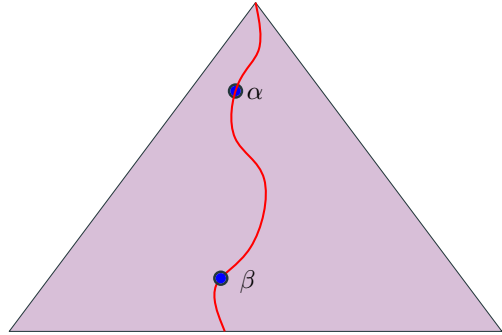
- For the sake of contradiction assume that there exists a grammar:
G a CFG for L.
- T_i : minimal parse tree in G for $a^i b^i c^i$.

You are bound to repeat yourself...

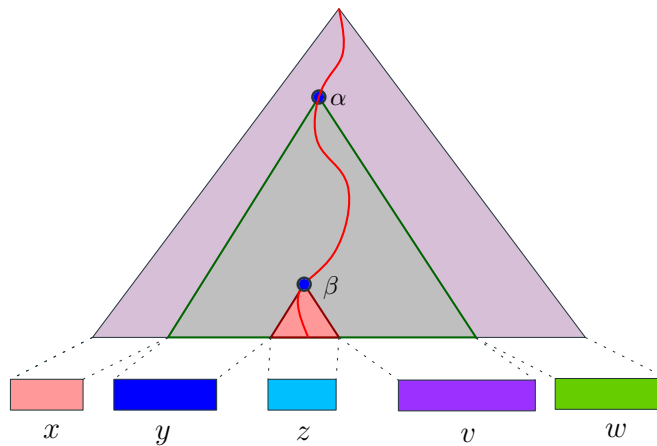
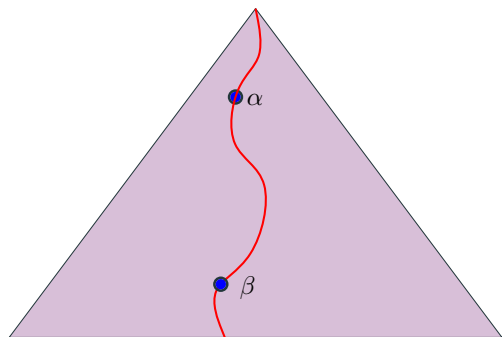
$$L = \{a^n b^n c^n \mid n \geq 0\}.$$

- For the sake of contradiction assume that there exists a grammar: G a CFG for L .
- T_i : minimal parse tree in G for $a^i b^i c^i$.
- $h_i = \text{height}(T_i)$: Length of longest path from root to leaf in T_i .
- For any integer t , there must exist an index $j(t)$, such that $h_{j(t)} > t$.
- There an index j , such that $h_j > (2 * \# \text{ variables in } G)$.

Repetition in the parse tree...

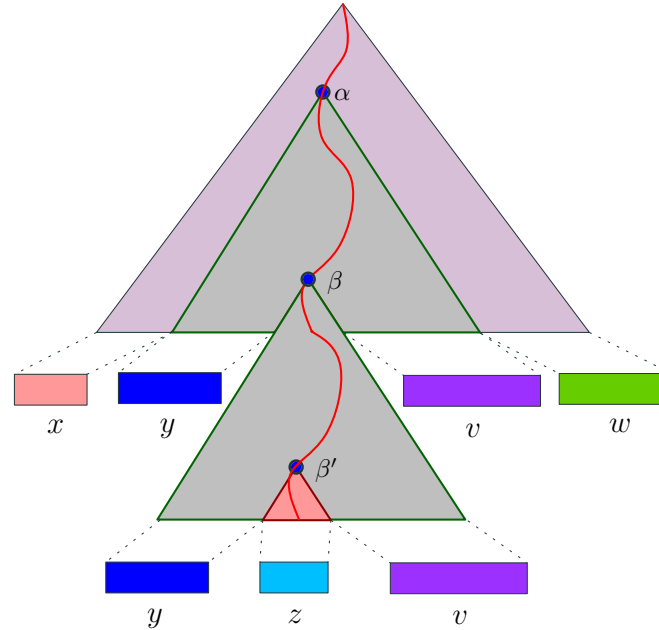
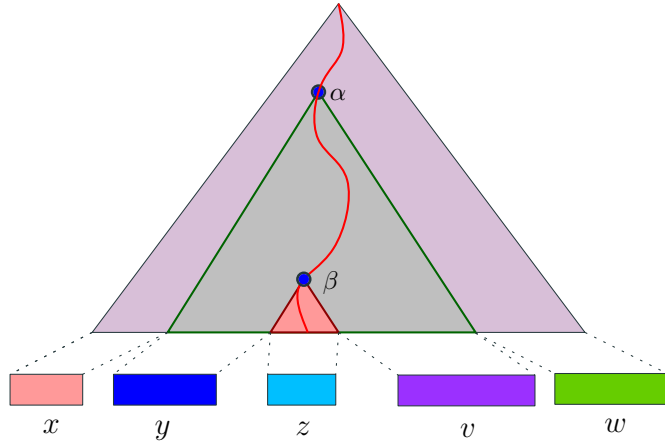


Repetition in the parse tree...



$$xyzvw = a^j b^j c^j$$

Repetition in the parse tree...



$$xyzvw = a^j b^j c^j \implies xy^2zv^2w \in L$$

Now for some case analysis...

- We know:

$$xyzvw = a^j b^j c^j$$

$$|y| + |v| > 0.$$

- We proved that $\tau = xy^2zv^2w \in L$.

Now for some case analysis...

- We know:

$$xyzvw = a^j b^j c^j$$

$$|y| + |v| > 0.$$

- We proved that $\tau = xy^2zv^2w \in L$.
- If y contains both a and b , then, $\tau = \dots a \dots b \dots a \dots b \dots$

Now for some case analysis...

- We know:

$$xyzvw = a^j b^j c^j$$

$$|y| + |v| > 0.$$

- We proved that $\tau = xy^2zv^2w \in L$.
- If y contains both a and b , then, $\tau = \dots a \dots b \dots a \dots b \dots$
Impossible, since $\tau \in L = \{a^n b^n c^n \mid n \geq 0\}$.

Now for some case analysis...

- We know:

$$xyzvw = a^j b^j c^j$$

$$|y| + |v| > 0.$$

- We proved that $\tau = xy^2zv^2w \in L$.
- If y contains both a and b , then, $\tau = \dots a \dots b \dots a \dots b \dots$
Impossible, since $\tau \in L = \{a^n b^n c^n \mid n \geq 0\}$.
- Similarly, not possible that y contains both b and c .

Now for some case analysis...

- We know:

$$xyzvw = a^j b^j c^j$$

$$|y| + |v| > 0.$$

- We proved that $\tau = xy^2zv^2w \in L$.
- If y contains both a and b , then, $\tau = \dots a \dots b \dots a \dots b \dots$
Impossible, since $\tau \in L = \{a^n b^n c^n \mid n \geq 0\}$.
- Similarly, not possible that y contains both b and c .
- Similarly, not possible that v contains both a and b .
- Similarly, not possible that v contains both b and c .

Now for some case analysis...

- We know:

$$xyzvw = a^j b^j c^j$$

$$|y| + |v| > 0.$$

- We proved that $\tau = xy^2zv^2w \in L$.
- If y contains both a and b , then, $\tau = \dots a \dots b \dots a \dots b \dots$
Impossible, since $\tau \in L = \{a^n b^n c^n \mid n \geq 0\}$.
- Similarly, not possible that y contains both b and c .
- Similarly, not possible that v contains both a and b .
- Similarly, not possible that v contains both b and c .
- If y contains only a s, and v contains only b s, then... $\#_{(a)}(\tau) \neq \#_{(c)}(\tau)$.
Not possible.

Now for some case analysis...

- Similarly, not possible that y contains only a s, and v contains only c s.
Similarly, not possible that y contains only b s, and v contains only c s.

Now for some case analysis...

- Similarly, not possible that y contains only a s, and v contains only c s.
Similarly, not possible that y contains only b s, and v contains only c s.
- Must be that $\tau \notin L$. A contradiction.

We conclude...

Lemma

The language $L = \{a^n b^n c^n \mid n \geq 0\}$ is not CFL (i.e., there is no CFG for it).