Pre-lecture teaser

Given the language:

$$L = \{ww^R | w \in \{0, 1\}^*\} \tag{1}$$

Prove that this language is non-regular

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

Instructor: Nickvash Kani

September 12, 2023

University of Illinois at Urbana-Champaign

Given the language:

$L_x = \{0^n1 \mid n > 0\} =$
$r = 0^*1$

$$L = \{ww^R | w \in \{0, 1\}^*\} \tag{2}$$

$F = \{0, 01\}$

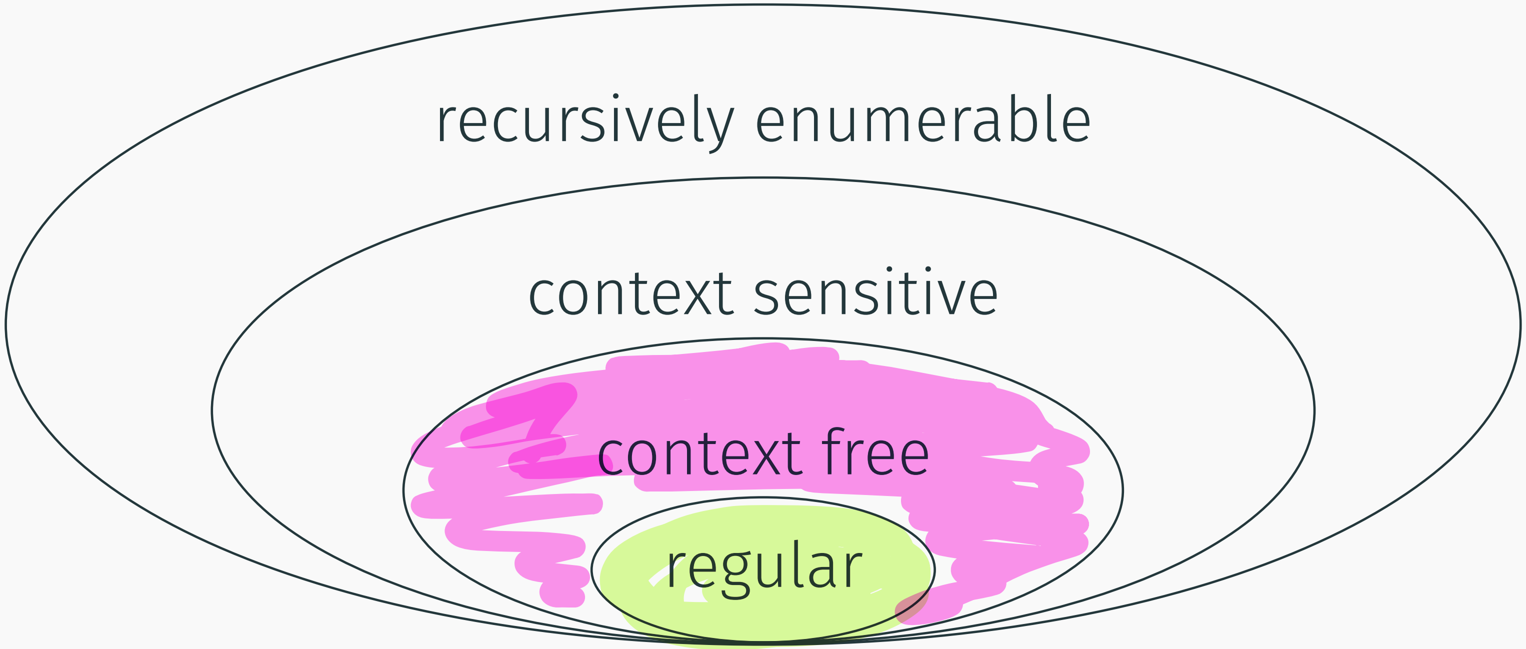
$0 \notin L_x$ $\quad w = \varepsilon$
$01 \in L_x$

Prove that this language is non-regular

- Assume L is regular. Then it must be representable by a DFA M.
- Consider the fooling set $F = \{(01)^i \mid i > 0\}$
  F is a fooling b/c for any 2 strings in F $(01)^i, (01)^j$, there is a suffix $y = (10)^i$ where $(01)^i(10)^i \in L$, $(01)^j(10)^i \notin L$
- $|F| = \infty$ therefore $|Q| = \infty$

Contradiction — assumption is wrong

Chomsky hierarchy revisited



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

- $V = \{S\}$ variables / non-terminal
- $T = \{0, 1\}$ terminals / characters in the alphabet
- $P = \{S \to \epsilon \mid 0S0 \mid 1S1\}$ → production rules
  (abbrev. for $S \to \epsilon, S \to 0S0, S \to 1S1$)

$S \to \varepsilon$

$S \to 0S0 \to 00$

$1S1 \to 10S01 \to 10\varepsilon 01 \to 1001$

Example

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

$$S \leadsto 0S0 \leadsto 01S10 \leadsto 011S110 \leadsto 011\varepsilon110 \leadsto 011110$$

Example

- $V = \{S\}$ $L = \{ w w^R \mid w \in \Sigma^* \}$
- $T = \{0, 1\}$
- $P = \{S \to \epsilon \mid 0S0 \mid 1S1\}$
  (abbrev. for $S \to \epsilon, S \to 0S0, S \to 1S1$)

$$S \leadsto 0S0 \leadsto 01S10 \leadsto 011S110 \leadsto 011\varepsilon110 \leadsto 011110$$

What strings can S generate like this?

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

$$G = \Big(\text{ Variables, } \quad \text{Terminals, } \quad \text{Productions, } \quad \text{Start var } \Big)$$

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)

$$G = \Big(\quad \text{Variables,} \quad \text{Terminals,} \quad \text{Productions,} \quad \text{Start var} \quad \Big)$$

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

  $A \to \alpha$

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

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

$aAb \longrightarrow cdABeC$

$G = \Big($ Variables, Terminals, Productions, Start var $\Big)$

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
  $A \to \alpha$
  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

$$G = \Big(\quad \text{Variables}, \quad \text{Terminals}, \quad \text{Productions}, \quad \text{Start var} \quad \Big)$$

Example formally...

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

$$G = \left(\{S\}, \quad \{0,1\}, \quad \left\{ \begin{matrix} S \to \epsilon, \\ S \to 0S0 \\ S \to 1S1 \end{matrix} \right\} \quad S \right)$$

Notation and Convention

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

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

- $\alpha_1 = \beta A \delta$
- $\alpha_2 = \beta \gamma \delta$
- $\boxed{A \to \gamma}$ is in P.

$S \to \epsilon$

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

“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 $\beta_1 \rightsquigarrow^{k-1} \alpha_2$.

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

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

Context Free Languages

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

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 $L = L(G)$.

Example

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

$V = \{S\}$

$T = \{0, 1\}$

$P = S \to \varepsilon \mid 0S1$

$S \Rightarrow S$

Example

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

$V = \{S\}$

$T = \{0, 1\}$

$P = S \to \varepsilon \mid 0S1 \mid S1$

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

$S \Rightarrow S$

Converting regular languages into CFL

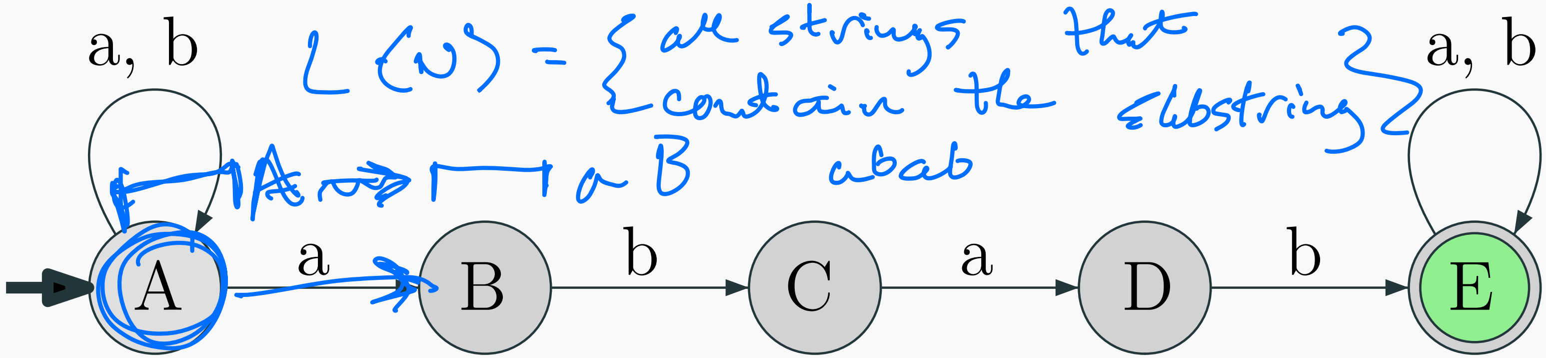
Regular Grammar

What was the grammar for a regular language?

Let's figure it out visually!

$L(N) = \{$all strings that contain the substring abab$\}$

$A \to aB$



$A \to aA$

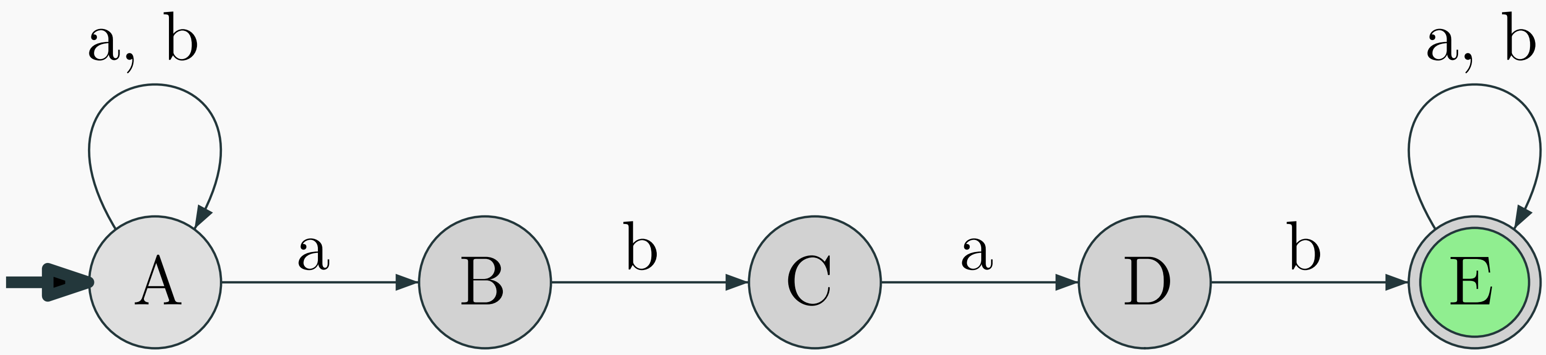
$$G = \left(\{A, B, C, D, E\}, \{a, b\}, \left\{ \begin{array}{c} A \to aA, A \to bA, A \to aB, \\ B \to bC, \\ C \to aD, \\ D \to bE, \\ E \to aE, E \to bE, E \to \varepsilon \end{array} \right\}, A \right)$$

$\boxed{V \to tVt}$

Converting regular languages into CFL II

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

$$G = \Big(\overbrace{Q}^{\text{Variables}}, \overbrace{\Sigma}^{\text{Terminals}}, \overbrace{\begin{array}{c}\{q \to a\delta(q,a) \mid q \in Q, a \in \Sigma\} \\ \cup \{q \to \varepsilon \mid q \in A\}\end{array}}^{\text{Productions}}, \overbrace{s}^{\text{Start var}} \Big)$$



Converting regular languages into CFL I

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

In regular languages:

- Terminals can only appear on one side of the production string
- Only one varibale 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

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

$0^* 1^*$

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.

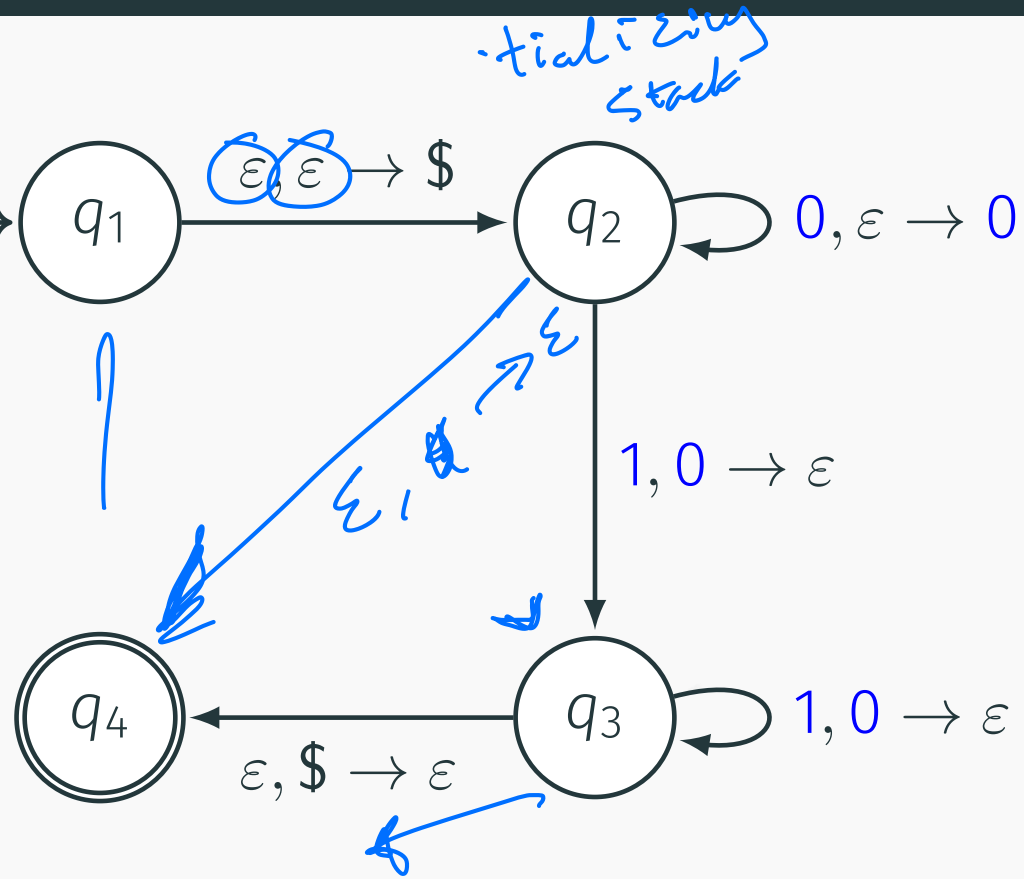
We have NFAs from regular languages. What can we add to enable them to recognize CFLs?

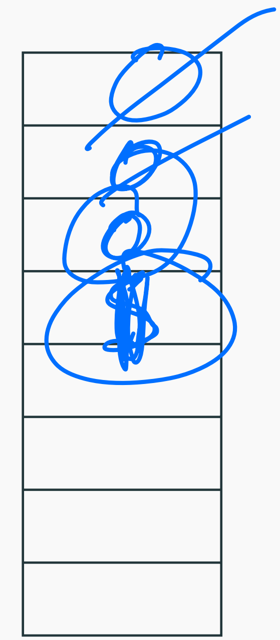
We need a stack!

Push-down automata example

00011

101



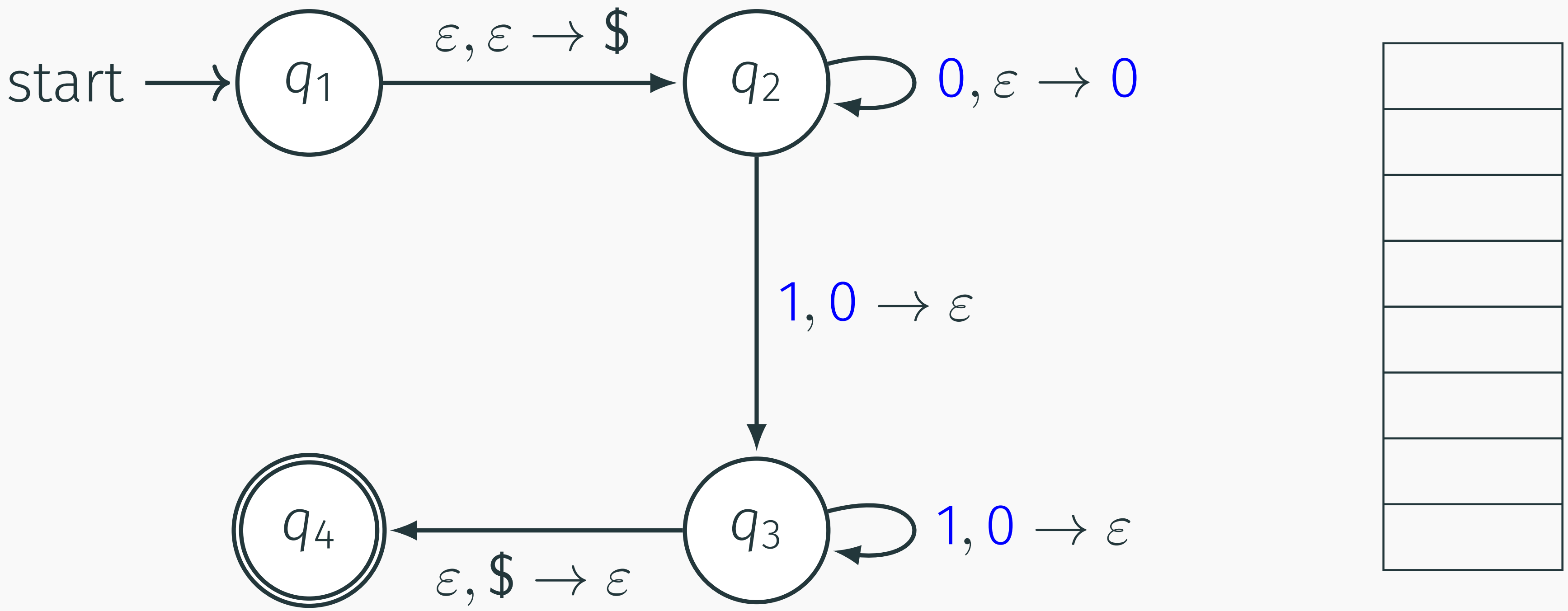


Each transition is formatted as:

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

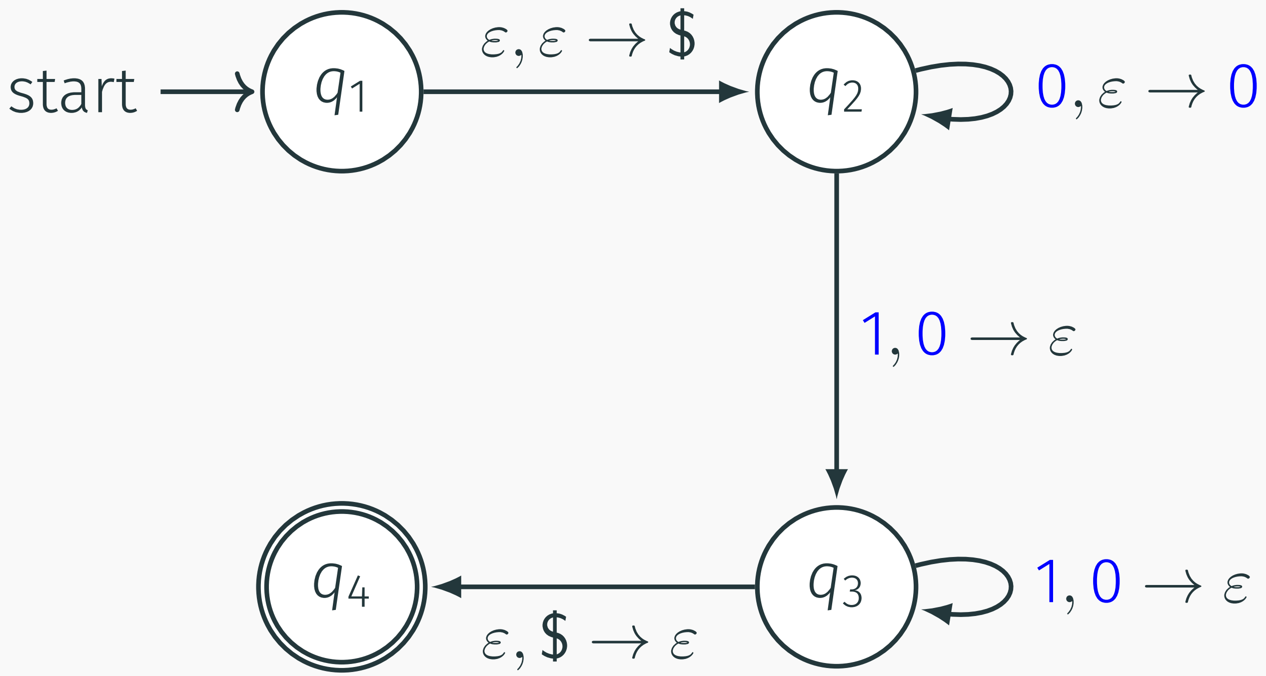
$$\langle\text{input read}\rangle, \langle\text{stack pop}\rangle \to \langle\text{stack push}\rangle \tag{3}$$

Push-down automata example



Does this machine recognize 0011?

Push-down automata example



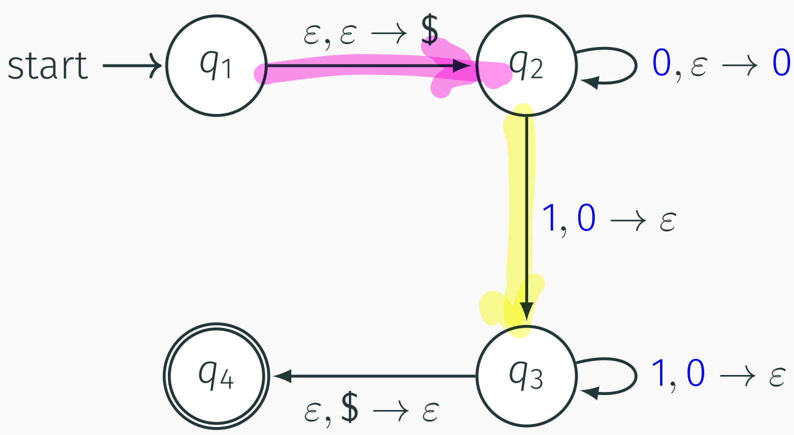
Does this machine recognize 0101?

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,
- $\delta : Q \times \Sigma \cup \{\varepsilon\} \times \Gamma \cup \{\varepsilon\} \to \mathcal{P}(Q \times (\Gamma \cup \{\varepsilon\}))$ 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-determinisitc PDAs.



- $Q = \{q_1, q_2, q_3, q_4\}$
- $\Sigma = \{0, 1\}$
- $\Gamma = \{0, \$\}$
- $s = q_1$
- $A = \{q_4\}$

$\delta =$

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

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:

$$S \to 0S|1$$

Convert a CFG to a PDA I

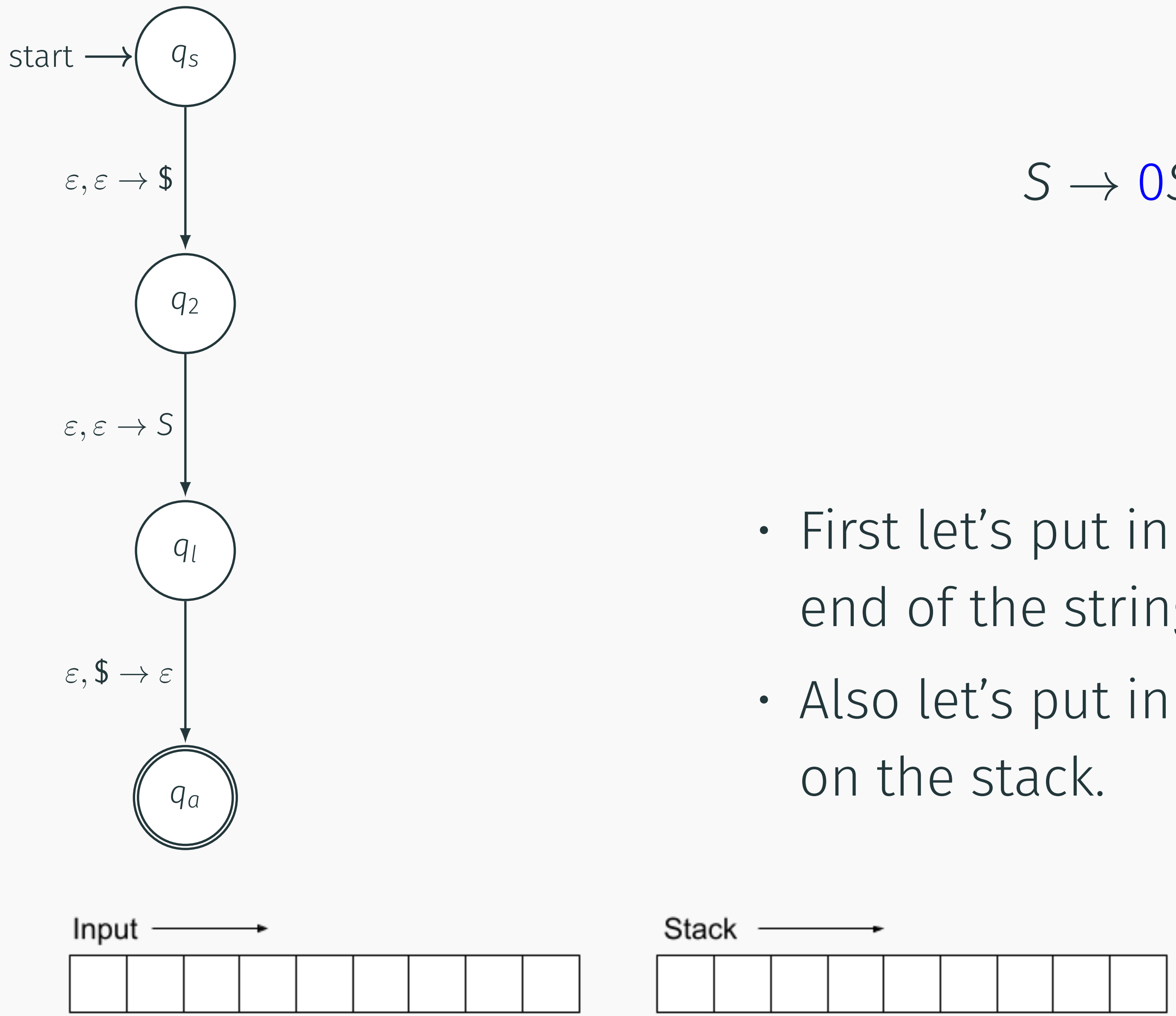
Converting a CFG to a PDA is simple (but a little tedious). Let's demonstrate via simple example:

$$S \to 0S|1$$

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.

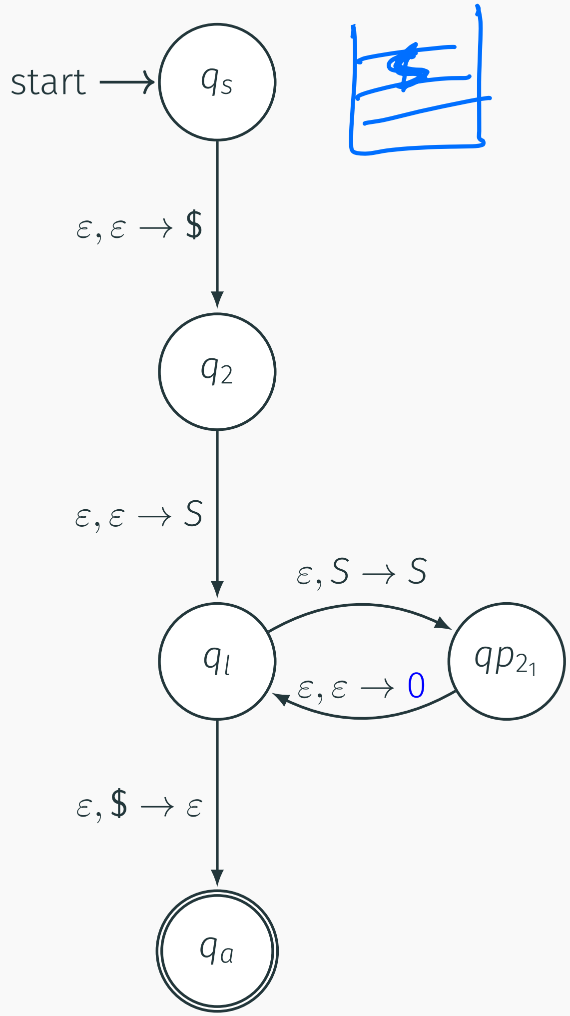
Convert a CFG to a PDA I

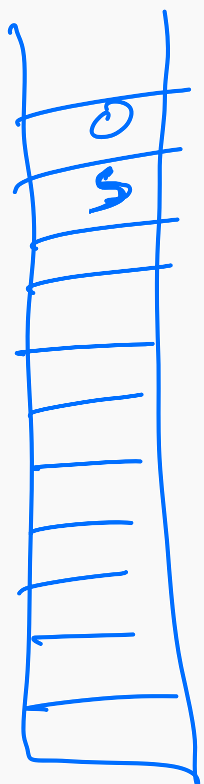


$$S \to 0S|1|\epsilon$$

- 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



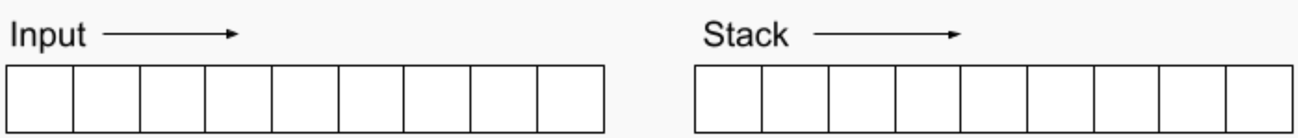


$S \to 0S \to 00S \to 000S$

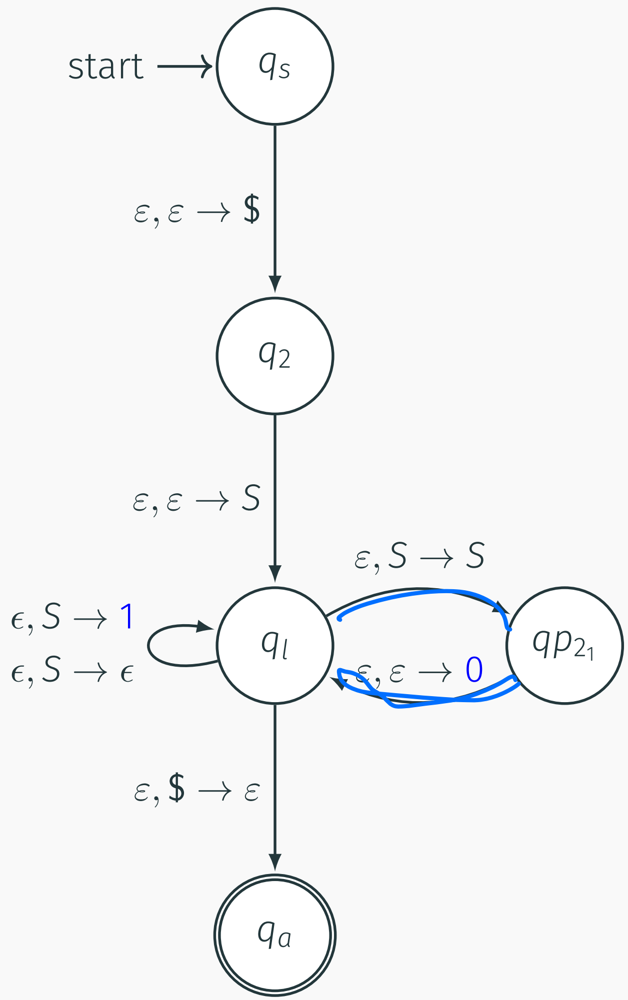
$S \to 0S|1|\epsilon$

Next we want to add a loop for every non-terminla symbol that replaces that non-terminal with the result. Consider the rule: $S \to 0S$

- 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.

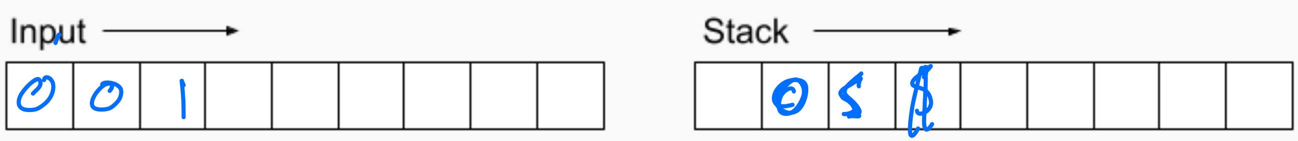


Convert a CFG to a PDA I

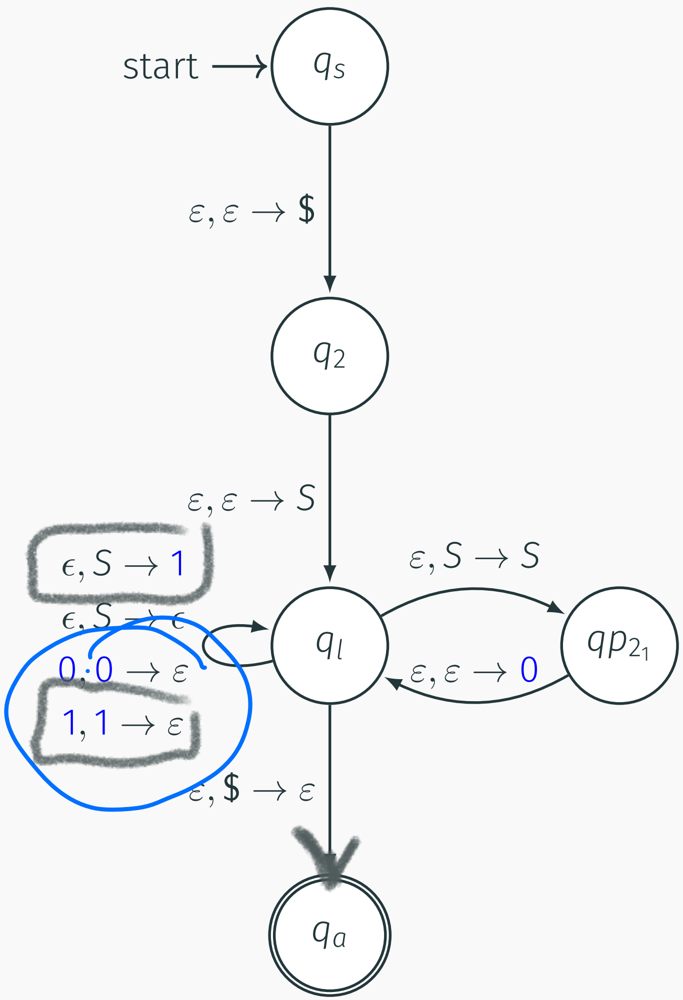


$$S \to 0S|1|\epsilon$$

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

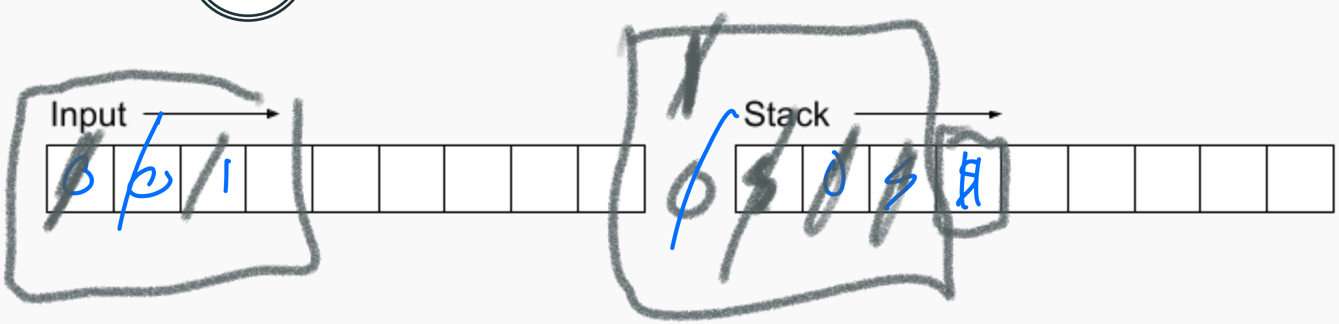


Convert a CFG to a PDA I

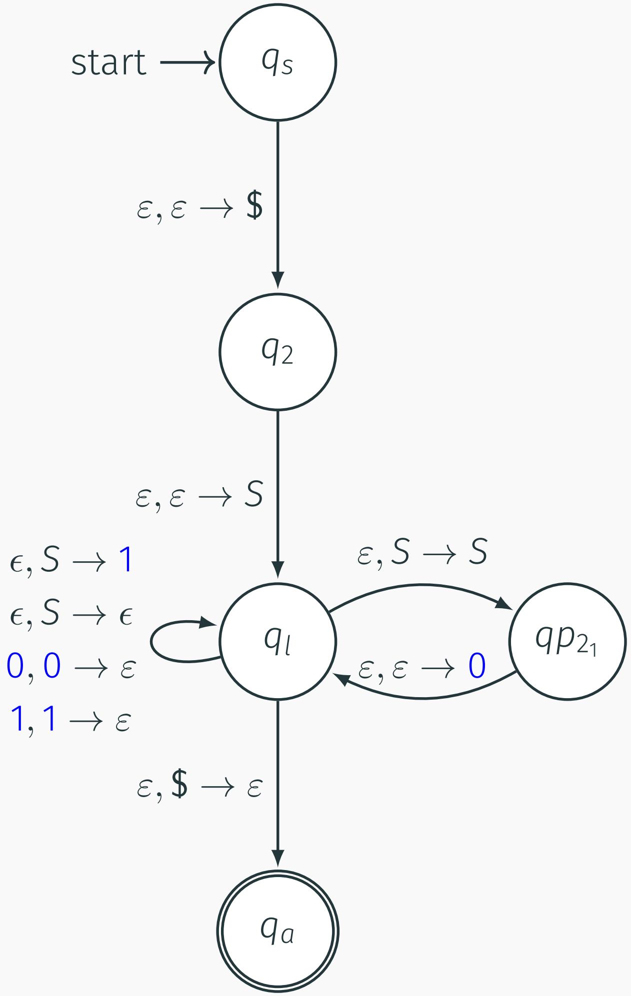


$$S \to 0S|1|\epsilon$$

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.



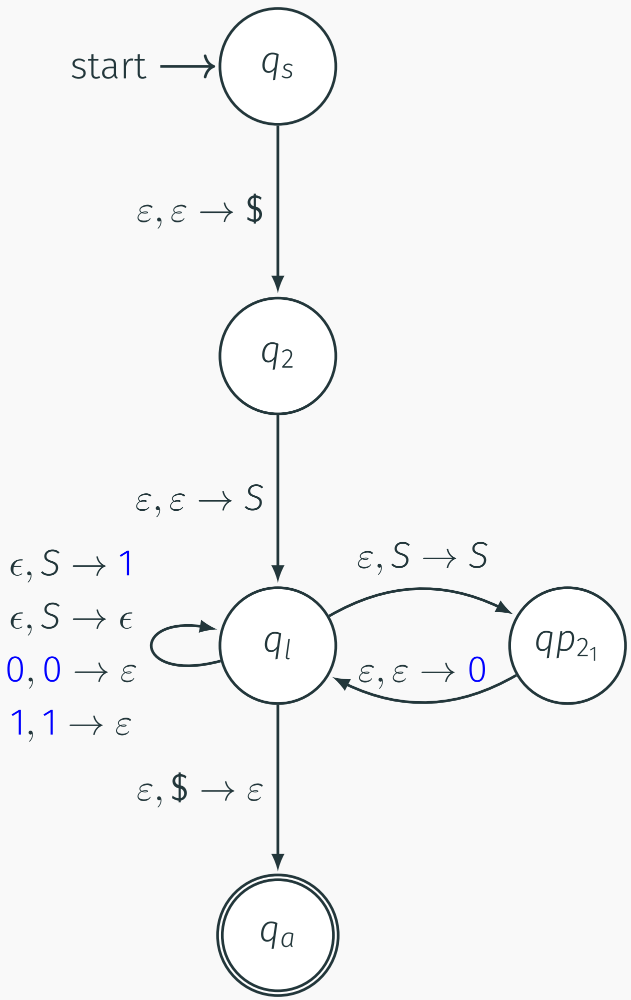
Convert a CFG to a PDA I



$$S \to 0S|1|\epsilon$$

Let's go over the operation again:

Convert a CFG to a PDA I

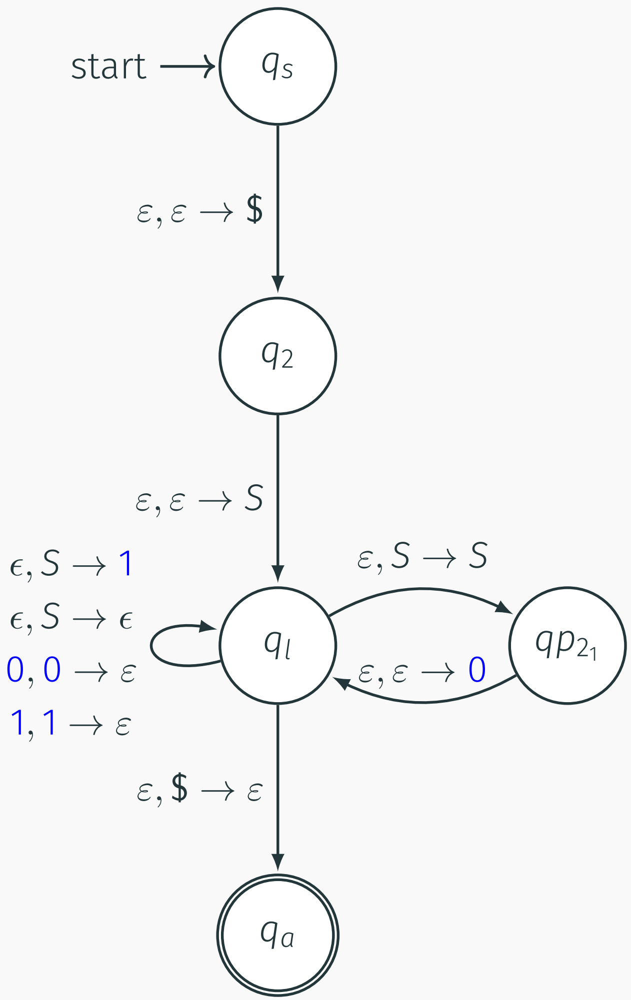


$$S \to 0S|1|\epsilon$$

Let's go over the operation again:

- Does this automata accept 001?

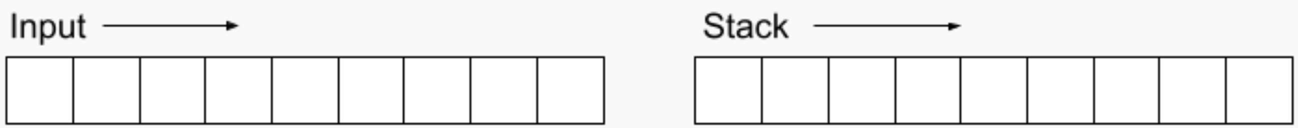
Convert a CFG to a PDA I



$$S \to 0S|1|\epsilon$$

Let's go over the operation again:

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

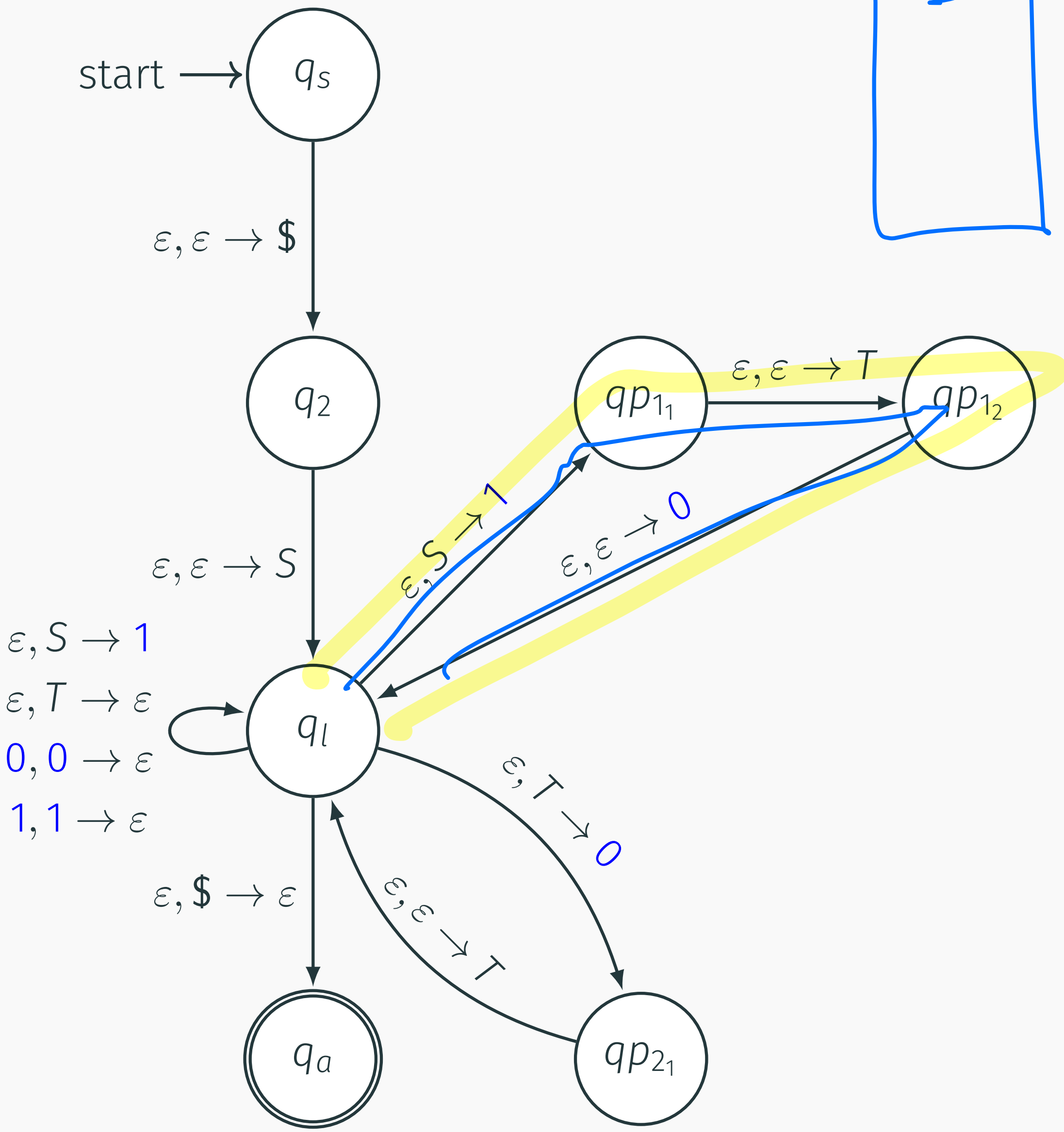


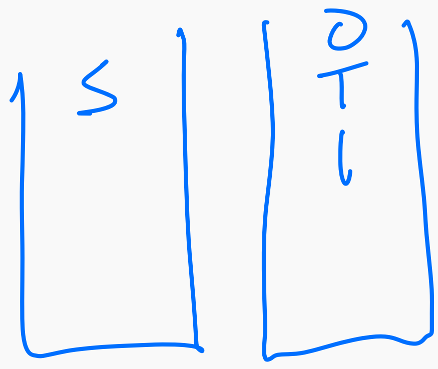
Convert a CFG to a PDA II

Let's do a harder example:

$$S \rightarrow 0T1|1$$
$$T \rightarrow T0|\varepsilon$$

Convert a CFG to a PDA II



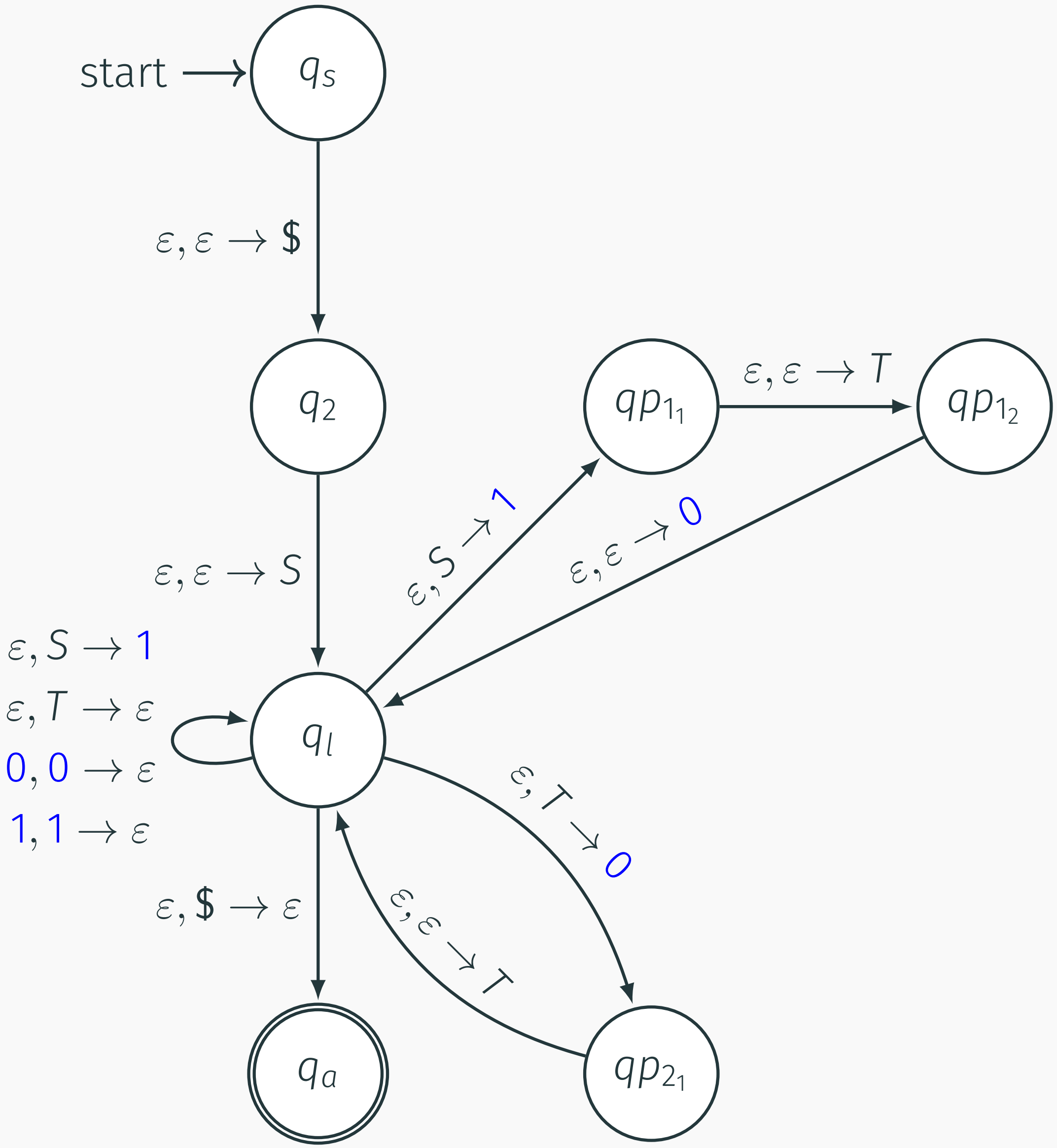


$S \to 0T1 | 1$

$T \to T0 | \varepsilon$

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

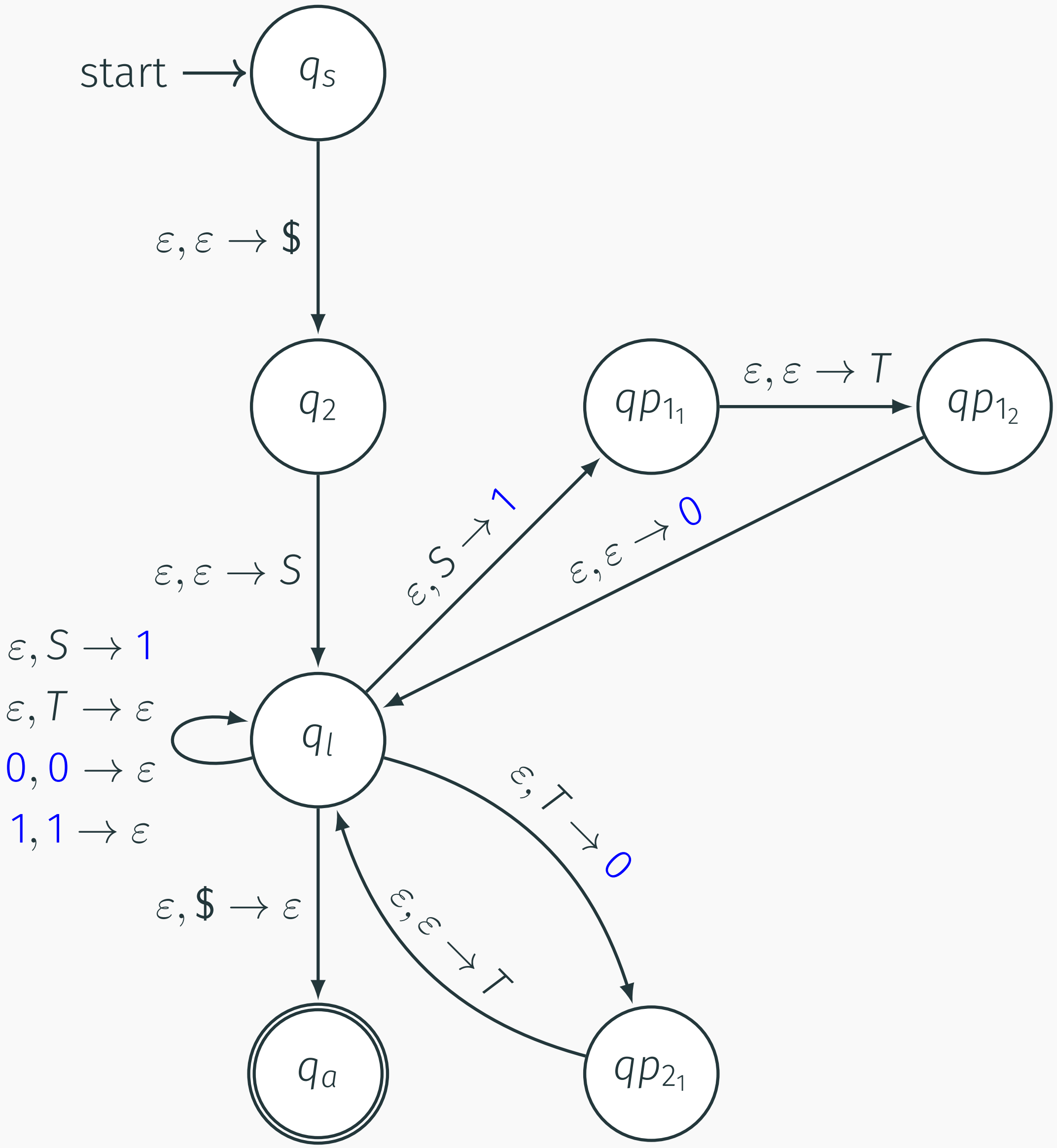


$S \to 0T1|1$

$T \to T0|\varepsilon$

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

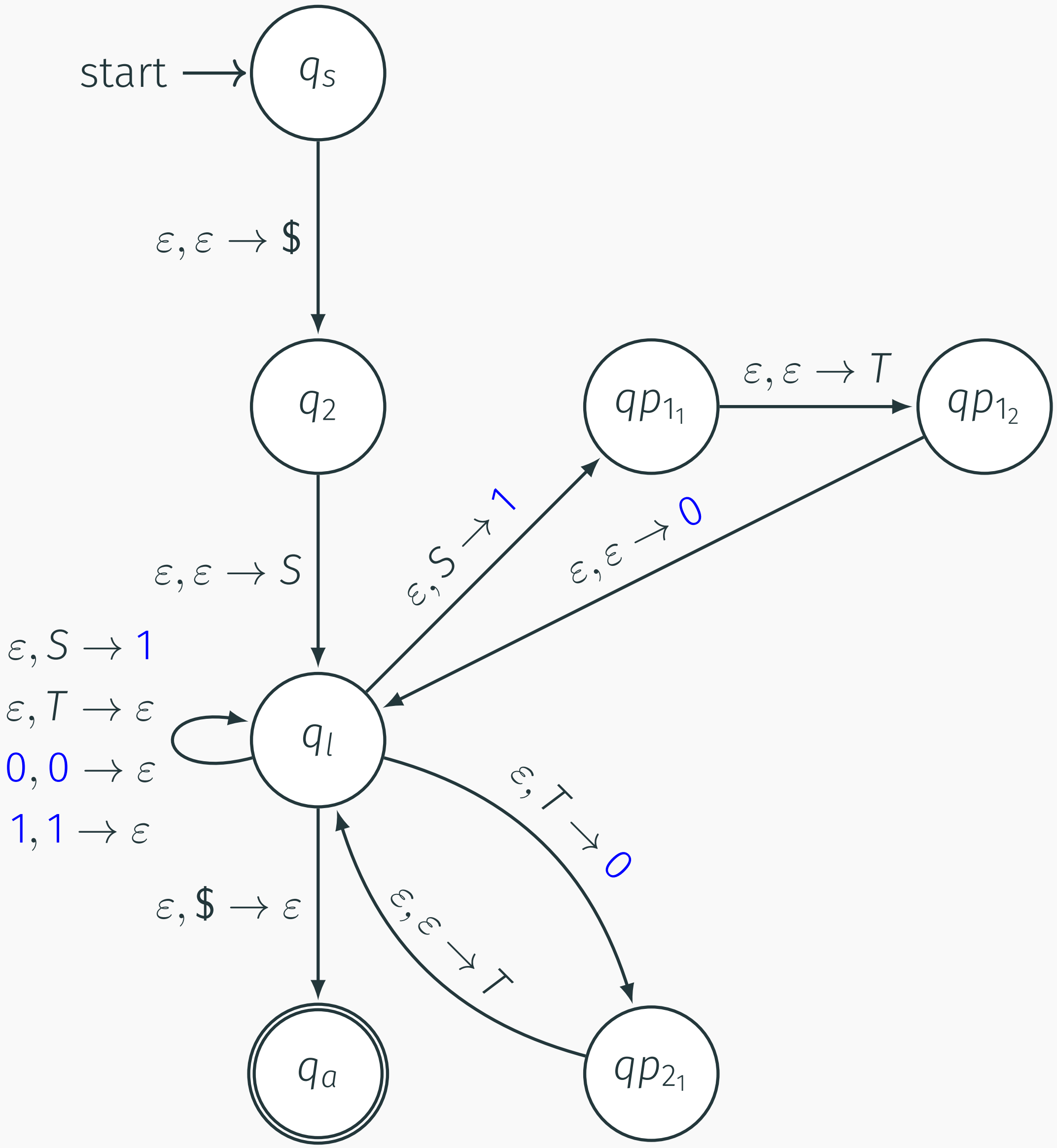
Convert a CFG to a PDA II



$$S \to 0T1|1$$
$$T \to T0|\varepsilon$$

- 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



$S \to 0T1|1$

$T \to T0|\varepsilon$

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 PDAs. 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!

$L = \{0^n1^n | n \geq 0\}$ 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.*

PDA_1 & PDA_2

$PDA_U =$ $Q = Q_1 \cup Q_2 \cup s$

δ: $\delta(s, \varepsilon, \varepsilon) \rightarrow (s_1, \varepsilon)$

$\delta \quad \rightarrow (s_2, \varepsilon)$

$G_U =$ $V_U = \{V_1, V_2, S_U\}$

$S = S_u$

$T = T$

$P_U = P_1, P_2$

$S_u \rightarrow S_1 | S_2$

Closure Properties of CFLs- Concatenation

Theorem

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

$G_U =$ $\quad V_U = \{V_1, V_2, S_U\}$

$S = S_U$

$T = T$

$P_U = P_1, P_2$

$S_U \to S_1 S_2$

Closure Properties of CFLs- Kleene star

Theorem

CFLs are closed under Kleene star.

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

$S_u \rightarrow S_u S_u \mid S_1 \mid S_2$

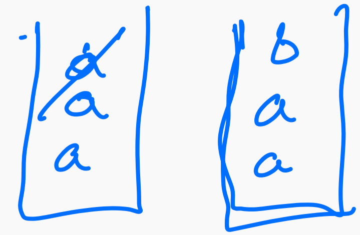
Bad news: Canonical non-CFL

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.

Theorem

CFLs are not closed under intersection.



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

$$L_2 = \{a^m b^n c^n \mid \ldots\}$$

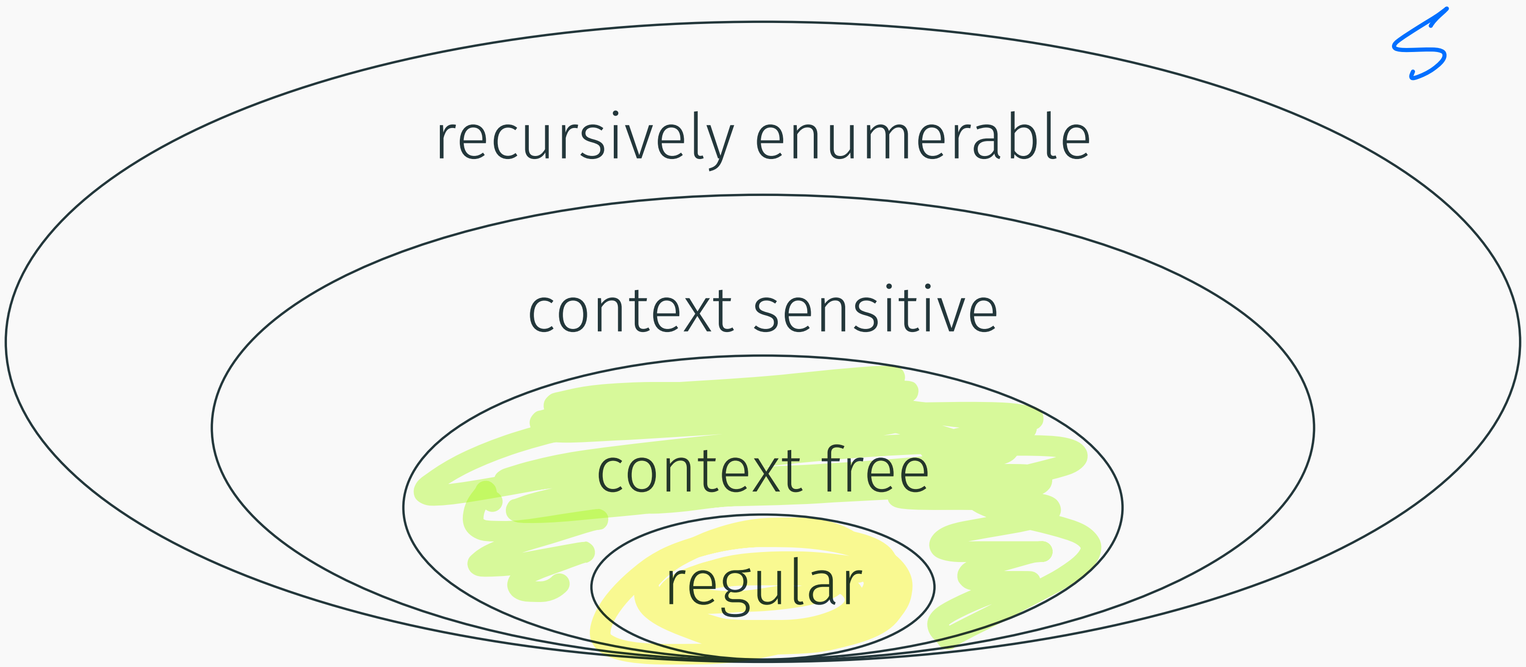
$$L_1 \cap L_2 = \{a^n b^n c^n\}$$

Even more bad news: CFL not closed under complement

Theorem

CFLs are not closed under complement.

The more you know!



We're making our way up the Chompsky 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

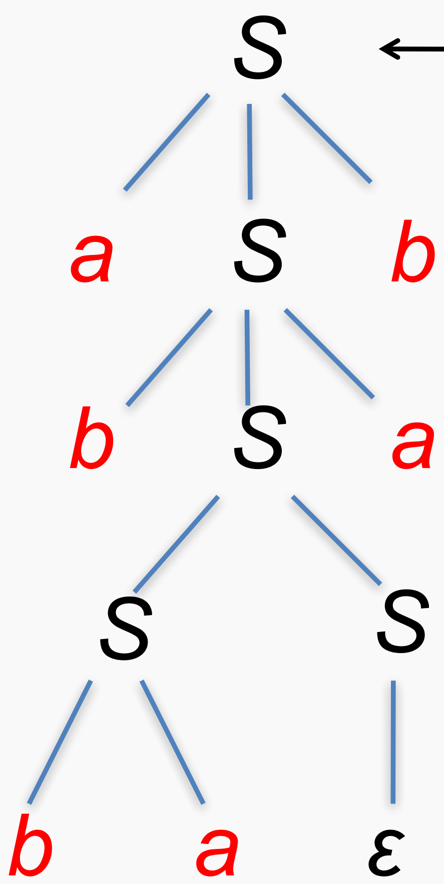
Parse Trees or Derivation Trees

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



A derivation tree for *abbaab*

(also called “parse tree”)

$S \to aSb \mid bSa \mid SS \mid ab \mid ba \mid \varepsilon$

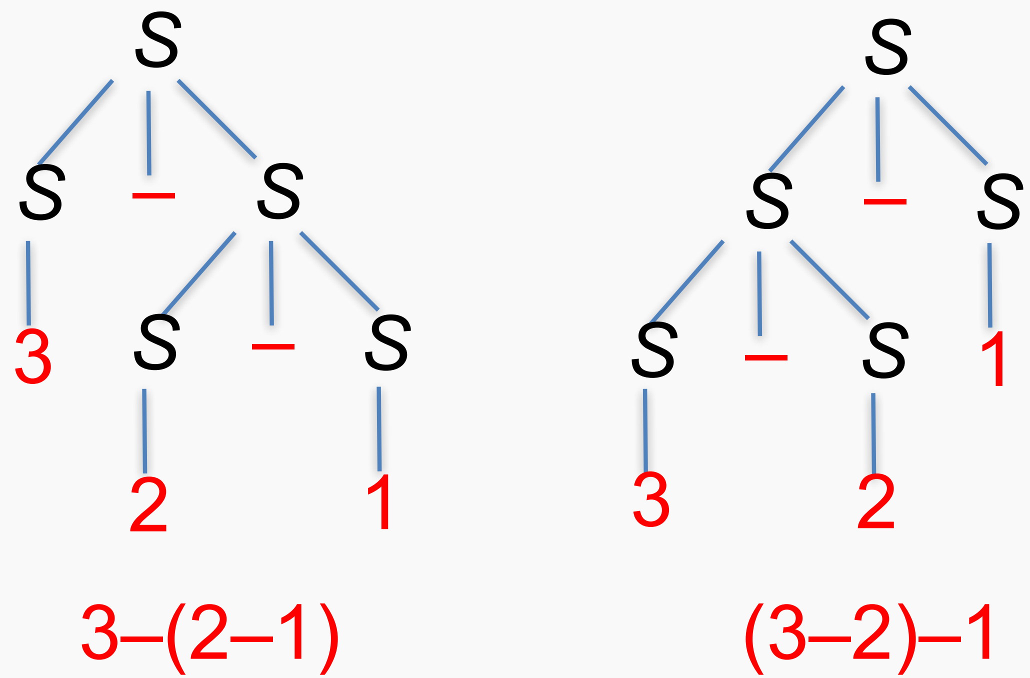
A corresponding derivation of *abbaab*

$S \Rightarrow aSb \Rightarrow abSab \Rightarrow abSSab \Rightarrow abbaSab \Rightarrow abbaab$

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: $S \to S - S \mid 1 \mid 2 \mid 3$

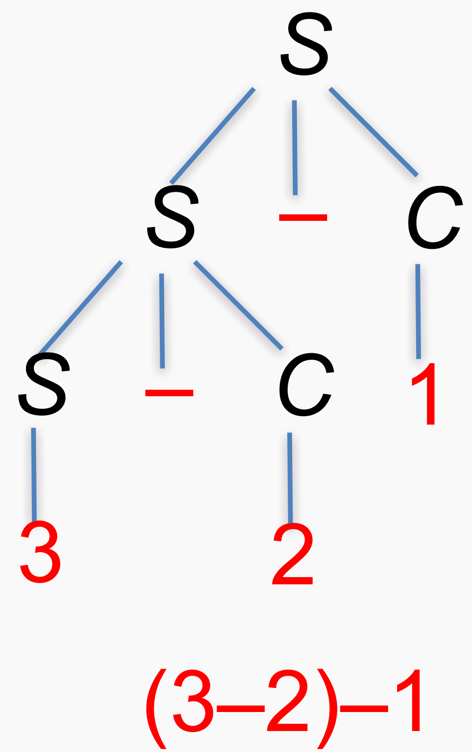


Ambiguity in CFLs

- Original grammar: $S \to S - S \mid 1 \mid 2 \mid 3$
- Unambiguous grammar:

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

  $C \to 1 \mid 2 \mid 3$



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...

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

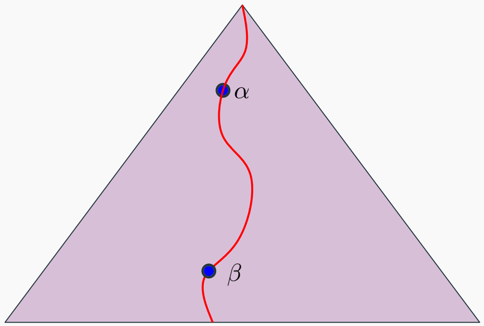
- For the sake of contradiction assume that there exists a grammar:
  G a CFG for L.
- T_i: minimal parse tree in G for $a^i b^i c^i$.

You are bound to repeat yourself...

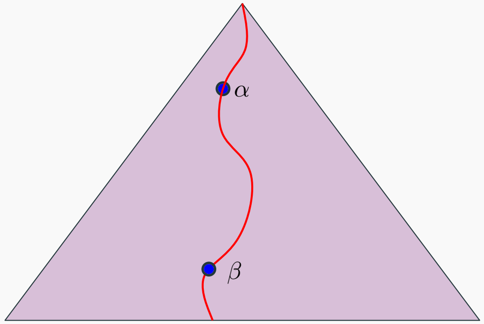
$L = \{a^n b^n c^n \mid n \geq 0\}$.

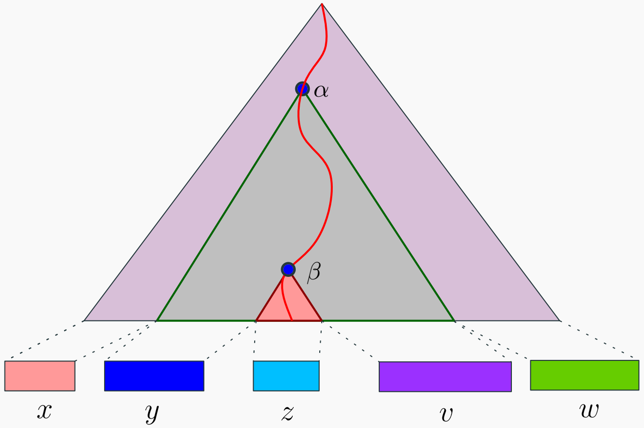
- For the sake of contradiction assume that there exists a grammar:
  G a CFG for L.
- T_i: minimal parse tree in G for $a^i b^i c^i$.
- $h_i = \mathrm{height}(T_i)$: Length of longest path from root to leaf in T_i.
- For any integer t, there must exist an index $j(t)$, such that $h_{j(t)} > t$.
- There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

Repetition in the parse tree...



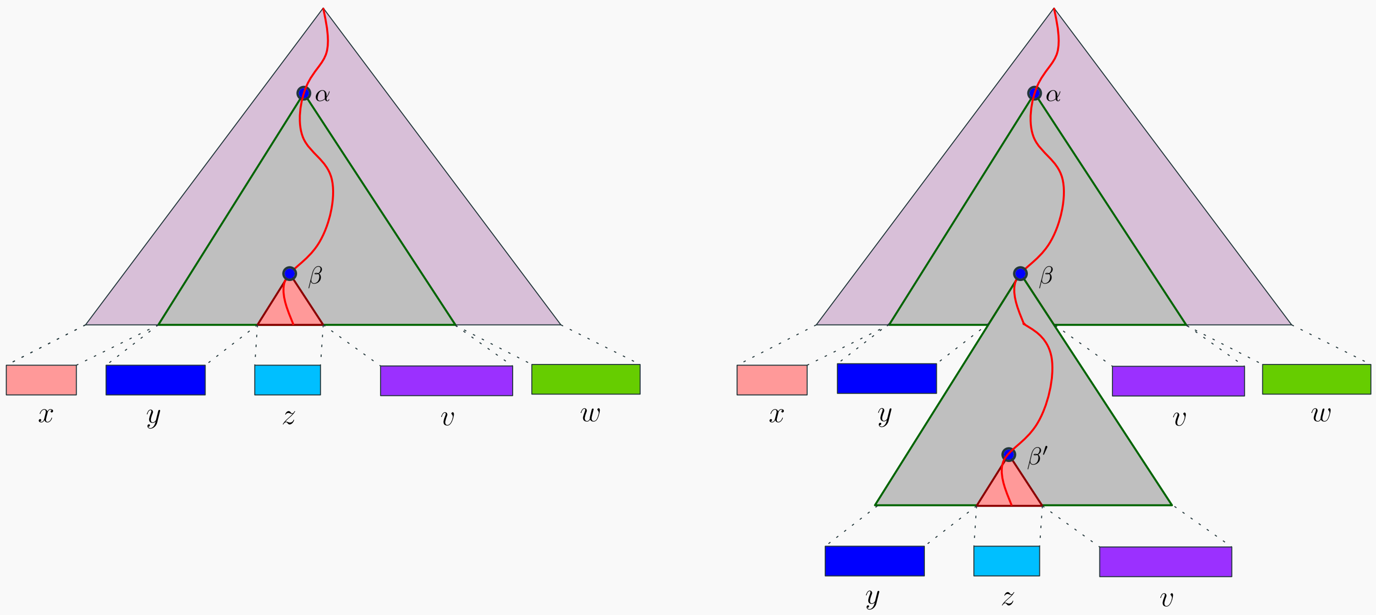
Repetition in the parse tree...





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

Repetition in the parse tree...



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

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, $\tau = ...a...b...a...b....$

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, $\tau = ...a...b...a...b....$
  Impossible, since $\tau \in L = \left\{ a^n b^n c^n \mid n \geq 0 \right\}$.

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, $\tau = ...a...b...a...b....$
  Impossible, since $\tau \in L = \left\{ a^n b^n c^n \;\middle|\; n \geq 0 \right\}$.
- 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, $\tau = ...a...b...a...b....$
  Impossible, since $\tau \in L = \left\{ a^n b^n c^n \mid n \geq 0 \right\}$.
- 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, $\tau = ...a...b...a...b....$
  Impossible, since $\tau \in L = \left\{ a^n b^n c^n \;\middle|\; n \geq 0 \right\}$.
- 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 as, and v contains only bs, then...
  $\#_{(a)}(\tau) \neq \#_{(c)}(\tau)$.
  Not possible.

Now for some case analysis...

- Similarly, not possible that y contains only as, and v contains only cs.
  Similarly, not possible that y contains only bs, and v contains only cs.

Now for some case analysis...

- Similarly, not possible that y contains only as, and v contains only cs.
  Similarly, not possible that y contains only bs, and v contains only cs.
- 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).*